

Audio/Video Connectivity Solutions for Virtex-5 FPGAs

Reference Designs for the Broadcast Industry: Volume 2

XAPP1014 (v1.2) November 9, 2009



Xilinx is disclosing this user guide, manual, release note, and/or specification (the "Documentation") to you solely for use in the development of designs to operate with Xilinx hardware devices. You may not reproduce, distribute, republish, download, display, post, or transmit the Documentation in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Xilinx expressly disclaims any liability arising out of your use of the Documentation. Xilinx reserves the right, at its sole discretion, to change the Documentation without notice at any time. Xilinx assumes no obligation to correct any errors contained in the Documentation, or to advise you of any corrections or updates. Xilinx expressly disclaims any liability in connection with technical support or assistance that may be provided to you in connection with the Information.

THE DOCUMENTATION IS DISCLOSED TO YOU "AS-IS" WITH NO WARRANTY OF ANY KIND. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE DOCUMENTATION, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT OF THIRD-PARTY RIGHTS. IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOSS OF DATA OR LOST PROFITS, ARISING FROM YOUR USE OF THE DOCUMENTATION.

© 2009 Xilinx, Inc. XILINX, the Xilinx logo, Virtex, Spartan, ISE, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. PCI, PCIe, and PCI Express are trademarks of PCI-SIG and used under license. All other trademarks are the property of their respective owners.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/29/09	1.0	Initial Xilinx release.
08/19/09	1.1	<ul style="list-style-type: none">Updated GTP_DUAL Clock Multiplication PLL, page 46.Added Chapter 4, Implementing SMPTE Serial Digital Interfaces with RocketIO GTX Transceivers and Chapter 8, Triple-Rate SDI for Virtex-5 FXT and TXT Devices.Updated Figure 5-6, page 149, Figure 18-8, page 396, and Figure 19-11, page 429.Updated description of ext_audio_pulse signal in Table 21-3, page 483.
11/09/09	1.2	<ul style="list-style-type: none">Removed "patent pending" from description of sampling method in Data Recovery Unit, page 316.

Table of Contents

Revision History	2
------------------------	---

Preface: About This Guide

Guide Contents	17
Section I: Introduction to the SMPTE Standards	17
Section II: Multirate SD/HD/3G-SDI Using Virtex-5 FPGA RocketIO Transceivers	17
Section III: SD-SDI Using Virtex-5 FPGA SelectIO LVDS	18
Section IV: DVB-ASI Using Virtex-5 FPGA SelectIO LVDS	18
Section V: AES Digital Audio	18
Section VI: Miscellaneous Audio and Video Topics	19
Section VII: Appendices	19
References	19
Additional Resources	20
Conventions	20
Typographical	20
Online Document	21

Chapter 1: Introduction

Section I:

Introduction to the SMPTE Standards

Chapter 2: Introduction to the SMPTE Serial Digital Interface Standards

Introduction	27
SD-SDI	27
SD-SDI Bit Rates	27
SD-SDI Physical Layer	28
SD-SDI Data Stream Format	28
SD-SDI Encoding	29
Error Detection for SD-SDI	31
HD-SDI	36
HD-SDI Bit Rates	36
HD-SDI Physical Layer	36
HD-SDI Data Stream Format	36
HD-SDI Encoding	39
Video Formats Supported by HD-SDI	39
Dual Link HD-SDI	41
3G-SDI	42
Ancillary Data	43
Conclusion	44

Chapter 3: Implementing SMPTE Serial Digital Interfaces with RocketIO GTP Transceivers

Introduction	45
Basics of Using the GTP Transceiver to Implement SDI	46
GTP_DUAL Tiles	46
GTP_DUAL Clock Multiplication PLL	46
GTP Reference Clocks	48
GTP Receiver	52
Receiving 270 Mb/s SD-SDI and DVB-ASI	52
Important RX Ports	53
RX Clocking	54
Important RX Attributes	55
RX Elastic Buffer	57
Cable Equalizer	57
GTP Transmitter	58
Transmitting 270 Mb/s SD-SDI and DVB-ASI	58
Important TX Ports	59
TX Clocking	60
Important TX Attributes	63
TX Buffer	63
Cable Driver	64
Control Module	65
Reset Logic	66
GTPRESET	66
Transmitter Resets	68
Receiver Resets	69
DRP Control	72
Mode Selection	72
Reference Clock Selection and Routing	73
Rate Detection	73
Connecting the v5gtp_sdi_control Module	74
Global Clock Multiplexer	79
Using the RocketIO GTP Transceiver Wizard	80
GTP Transceiver Reference Clock Connections	86

Chapter 4: Implementing SMPTE Serial Digital Interfaces with RocketIO GTX Transceivers

Summary	89
Introduction	89
Differences from GTP Transceivers	90
Using the GTX Transceiver to Implement SDI	90
GTX_DUAL Tiles	90
GTX Transceiver PMA PLL Section	91
GTX Transceiver Reference Clocks	93
GTX Receiver	96
Receiving 270 Mb/s SD-SDI and DVB-ASI	97
Important RX Ports	98
RX Clocking	98

Important RX Attributes	100
RX Elastic Buffer	101
Cable Equalizer	101
GTX Transmitter	102
Transmitting 270 Mb/s SD-SDI and DVB-ASI	103
Important TX Ports	103
TX Clocking	104
Important TX Attributes	106
TX Elastic Buffer	107
Cable Driver	107
Control Module	108
Reset Logic	110
GTXRESET	110
Transmitter Resets	111
Receiver Resets	113
DRP Control	116
Mode Selection	116
Reference Clock Selection & Routing	116
Rate Detection	117
Connecting the v5gtx_sdi_control Module	118
Global Clock Multiplexer	122
Using the RocketIO GTX Wizard	123
GTX Transceiver Reference Clock Connections	129

Section II:

Multirate SD/HD/3G-SDI Using Virtex-5 FPGA RocketIO Transceivers

Chapter 5: Triple-Rate SDI Receiver for Virtex-5 LXT and SXT Devices

Summary	133
Introduction	133
Supported Video Formats	134
Triple-Rate SDI Receiver Features	135
Light Triple-Rate SDI Receiver Reference Design	136
Light Triple-Rate Receiver Clocking	142
SD-SDI Output Timing	143
HD-SDI Output Timing	144
3G-SDI Output Timing	145
Full-Featured Triple-Rate SDI Receiver	147
Full-Featured Triple-Rate Receiver Clocking	161
Video and Timing Output Signals	162
SD-SDI Output Timing	164
HD-SDI Output Timing	165
3G-SDI Output Timing	166
Ancillary Data Outputs	170
Dual Link HD-SDI Mode	173
FPGA Resource Usage	178
Timing	178

Reference Design	180
Conclusion.....	180

Chapter 6: Triple-Rate SDI Transmitter for Virtex-5 LXT and SXT Devices

Summary	181
Introduction	181
Supported Video Formats.....	182
Triple-Rate SDI Transmitter Features	183
Light versus Full-Featured	184
Light Triple-Rate SDI Transmitter Reference Design	185
Light Triple-Rate Transmitter SD-SDI Operation	187
Light Triple-Rate Transmitter HD-SDI Operation	188
Light Triple-Rate Transmitter 3G-SDI Operation	189
Light Triple-Rate Transmitter Dual Link HD-SDI Operation	192
Light Triple-Rate Transmitter Details	201
SMPTE 352M Packet Insertion	201
Line Number Insertion	202
CRC Generation and Insertion	203
EDH Generation and Insertion	203
Full-Featured Triple-Rate SDI Transmitter Reference Design	203
Full-Featured Triple-Rate Transmitter SD-SDI Operation.....	212
Full-Featured Triple-Rate Transmitter HD-SDI Operation	213
Full-Featured Triple-Rate Transmitter Dual Link HD-SDI Operation.....	214
Full-Featured Triple-Rate Transmitter 3G-SDI Level A Operation	219
Full-Featured Triple-Rate Transmitter 3G-SDI Level B Operation.....	223
Full-Featured Triple-Rate Transmitter Details	224
SMPTE 352M Packet Insertion	225
Line Number Insertion	226
CRC Generation and Insertion	226
EDH Generation and Insertion.....	226
FPGA Resource Usage.....	226
Timing	227
Reference Design	228
Conclusion	228

Chapter 7: Triple-Rate SDI Pass-through Design for Virtex-5 LXT and SXT Devices

Summary	231
Introduction	231
Synchronous Systems	231
Frame Sync	232
Asynchronous Systems	233
Design Description	234
SD-SDI Mode	236
HD-SDI Mode	238
3G-SDI Level A Mode	239
3G-SDI Level B Mode	241
Other Considerations	242

Reference Design	243
Conclusion.....	243

Chapter 8: Triple-Rate SDI for Virtex-5 FXT and TXT Devices

Summary	245
Introduction	245
Supported Video Formats	246
GTX Triple-Rate SDI RX Reference Design.....	248
Triple-Rate SDI RX Modes	254
Triple-Rate SDI Clocking	254
SD-SDI Output Timing	255
HD-SDI Output Timing	256
3G-SDI Output Timing	256
Other Triple-Rate SDI RX Design Considerations	257
Dual-Link HD-SDI	257
Processing Embedded Audio and Other Ancillary Data	257
SMPTE 352M VPID Packets	257
SD-SDI EDH Error Detection	258
SD-SDI Data Recovery Unit	258
GTX Transceiver Triple-Rate SDI TX Reference Design.....	259
Triple-Rate SDI TX Datapath Modules	261
Triple-Rate SDI VPID Insert Module	261
Triple-Rate SDI Output Module	264
Triple-Rate SDI TX Modes	266
Triple-Rate SDI TX SD-SDI Operation	266
Triple-Rate SDI TX HD-SDI Operation.....	267
Triple-Rate SDI TX 3G-SDI Operation	268
Triple-Rate SDI TX Dual-Link HD-SDI Operation	271
Summary of Triple-Rate TX Modes	275
Triple-Rate TX Details	275
SMPTE 352M Packet Insertion	275
Line Number Insertion	276
CRC Generation and Insertion	277
EDH Generation and Insertion	277
Ancillary Data Insertion	277
FPGA Resource Usage.....	278
Timing.....	278
Reference Design	279
Conclusion.....	279

Section III:

SD-SDI using Virtex-5 FPGA SelectIO LVDS

Chapter 9: SD-SDI Receiver

Summary	283
LVDS SD-SDI Receiver (SDI Receiver).....	283

8X Oversampler	284
Data Recovery Unit	284
Video Decoder	284
Error Detection and Handling Processor	285
Clocking	285
Module I/O	286
Reference Design	287

Chapter 10: SD-SDI Transmitter

Summary	289
SD-SDI Transmitter	289
Video Pattern Generator	290
Standard Detect and Flywheel	290
Ancillary Data/EDH Processor	291
Video Encoder	292
Serializer	292
Clocking	292
Module I/O	292
Reference Design	295

Chapter 11: SD-SDI Receiver/Transmitter Demonstration Design

Summary	297
Overview	297
SD-SDI Receiver Demonstration	298
Required Equipment	298
Receiver Demonstration Platform Setup	298
SD-SDI Transmitter Demonstration	301
Required Equipment	301
Transmitter Demonstration Platform Setup	301
Design Summary	303
Running the Demonstration	303

Section IV:

DVB-ASI using Virtex-5 FPGA SelectIO LVDS

Chapter 12: DVB-ASI Introduction and Layer 0 Implementation

Background	307
DVB-ASI	307
Layer 2	308
Layer 1	309
Layer 0	310
Implementing DVB-ASI Layer 0 Using SelectIO Technology	311
DVB-ASI Receiver Implementation Using SelectIO Interface	311
DVB-ASI Transmitter Implementation Using SelectIO Interface	312

Conclusion	313
-------------------------	-----

Chapter 13: DVB-ASI Layer 1 and 2 Receiver

Summary	315
ASI Receiver	315
Data Recovery Unit	316
ASI Parallel Framer	316
8B/10B Decoder	317
Link Controller	318
Rate-Matching FIFO (Comma Correction)	319
Clocking	322
Reference Design	322
Design Hierarchy	322
DVB-ASI Receiver Ports	323
FPGA Resource Usage	324
Implementation Instructions	325
Generating the Rate-Matching SelectRAM Memory FIFO	325
Conclusion	326

Chapter 14: DVB-ASI Layer 1 and 2 Transmitter

Summary	327
ASI Transmitter	327
8B/10B Encoder	327
Serializer	328
Clocking	328
Reference Design	328
Design Hierarchy	328
ASI Transmitter Ports	329
FPGA Resource Usage	329
Implementation Instructions	330
Generating the Rate-Matching SelectRAM Memory FIFO	330
Conclusion	331

Chapter 15: DVB-ASI Layer 1 and 2 Pass-through Demonstration Design

Summary	333
Reference Design	333
Design Hierarchy	335
FPGA Resource Usage	336
Implementation Instructions	337
Generating the Rate-Matching SelectRAM Memory FIFO	337
Running the Demonstration Design	338
Required Equipment	338
Setup Instructions	338
Design Summary	341
Conclusion	343

Section V: AES Digital Audio

Chapter 16: Introduction to Digital Audio for Video Broadcasting

Introduction to the AES3 Digital Audio Standard	347
Data Format	347
Subframes, Frames, and Blocks.	348
Preambles	348
Biphase-Mark Encoding	349
Valid Bit	349
Channel Status	349
User Data	349
Parity Bit	349
Data Rate	350
SMPTE 272M: Embedded Digital Audio for SD-SDI	350
SD Embedded Audio Packets	351
SD Audio Data Packets	351
SD Extended Data Packets	353
SD Audio Control Packets	354
SD Audio Sample Distribution	356
SMPTE 299M Specification	356
Embedded Digital Audio for HD-SDI, 3G-SDI, and Dual Link HD-SDI.	356
HD Audio Data Packets	356
HD Audio Control Packets	357
HD Audio Sample Distribution	357
Audio Sample Rate Conversion	358
Introduction	358
Clarification of Terms	359
Methods of Sample Rate Conversion	359
Synchronous	359
Asynchronous	360
ASRC Operation	363
Ratio Control	363
Resampler	364
Synchronous SRC Operation	366
Lagrange Interpolation of Filter Coefficients	366
Performance Factors	368
Prototype Filter Design	368
Typical Applications	369
Digital Audio Reference Designs	371

Chapter 17: AES3 Serial Digital Audio Interfaces

Summary	373
Reference Design	373
Receiver	373
Transmitter	380
Transmitter clocking example	381
Channel Status CRC	382

Electrical Interface	384
Jitter	384
Module FPGA Resource Usage	385
Conclusion.....	385

Chapter 18: Asynchronous Sample Rate Converter

Summary	387
Structure.....	387
Modules	388
Functional Description	390
Ratio Control Functional Block	390
Ratio Detection	391
Ratio Calculation	393
Ratio Filtering for Jitter Tolerance.....	394
Ratio Regulation	396
Lock Status Indicators.....	397
Input Sample Storage	397
Clock Domain Considerations	400
Resampler Functional Block.....	401
Prototype Filter.....	401
Coefficient Interpolation.....	404
FIR Filter	406
Shared Divider	406
Control	407
Interface Timing	409
Performance	410
THD+N.....	410
Latency	411
FPGA Resource Utilization and Performance	412
Additional Channels.....	412
Data Flow Spreadsheet	413
Filter Phase Interpolation and FIR Filter.....	413
Output Sample Normalization	413
Reference Design	414
Conclusion.....	414

Chapter 19: Multi-Channel Asynchronous Sample Rate Converter

Summary	415
Structure.....	415
Multi-Channel Configurations.....	416
Four-Channel Configuration	416
16-Channel Configuration	417
Inputs and Outputs	419
Modules	420
Functional Description	421
Ratio Control Functional Block	421
Ratio Detection	423
Ratio Calculation	424
Ratio Filtering for Jitter Tolerance.....	425
Ratio Regulation	428

Lock Status Indicators	430
Input Sample Storage	430
Clock Domain Considerations	433
Resampler Functional Block	433
Prototype Filter	434
Coefficient Interpolation	437
FIR Filter	439
Signed Fractional Divider	440
Control	441
Interface Timing	443
Performance	444
THD+N	444
Maximum Conversion Ratios	446
Latency	446
FPGA Resource Utilization and Performance	447
Data Flow Spreadsheet	448
Filter Phase Interpolation and FIR Filter	448
Output Sample Normalization	449
Design Files	449
Conclusion	449

Chapter 20: Audio Multiplexer for HD-SDI Video

Introduction	451
Features	451
Embedded Audio	451
Audio Data Packets	452
Audio Sample Words	453
Audio Clock Phase Data	453
Error Correction Codes	454
Reference Design	454
Inputs and Outputs	456
Modules	458
Data Packet Multiplexer for C _B C _R Video Stream (hd_audio_mux_data)	459
Data Packet Multiplexer for Y Video Stream (hd_audio_mux_contr)	459
Ancillary Data Parsing (anc_det)	459
Audio Packet Multiplexer (packet_mux)	462
Clock Phase Determination (clock_phase)	463
Synchronizing One-Shot (sync_one_shot)	464
Audio Data Packet Construct (aud_data_constr)	465
Audio Control Packet Construct (aud_contr_constr)	468
Audio FIFOs (aud_fifos)	469
FIFO (fifo_28x16)	470
Error Correction Encode (ecc_encode)	472
Usage Models	473
Embedding Audio from a Synchronous Audio Source	473
Synchronizing Audio via Sample Rate Conversion	473
Embedding Asynchronous Audio	474
Replacing Audio Embedded in Received Video	474
Reembedding Audio after Separate Video/Audio Processing	475
FPGA Resources	475
Design Files	476

Conclusion	476
------------------	-----

Chapter 21: Audio Demultiplexer for HD-SDI Video

Introduction	477
Features	477
Embedded Audio	477
Audio Data Packets	478
Audio Sample Words	479
Audio Clock Phase Data	479
Error Correction Codes	480
Audio Control Packets	480
Reference Design	481
Inputs and Outputs	483
Modules	485
Data Packet Demultiplexer for C _B C _R Video Stream (hd_audio_demux_data)	486
Data Packet Demultiplexer for Y Video Stream (hd_audio_demux_contr)	486
Audio Packet Parser (aud_det)	486
Data Extraction (data_extr)	488
Audio FIFOs (aud_fifos)	490
FIFO (fifo_28x16)	491
Timing Control (demux_tim)	493
Synchronizing One-Shot (sync_one_shot)	495
Usage Models	496
Synchronous De-embedding	496
Asynchronous De-embedding	496
De-embedding Audio for Separate Video/Audio Processing	497
Performance and Resources	497
Design Files	498
Conclusion	498

Chapter 22: Error Correction for HD-SDI Embedded Audio

Summary	499
Features	499
Introduction	499
Error Correction Theory	500
Finite Fields	501
Error Correction Hardware	505
Encoding	505
Decoding	506
Reference Design	506
Inputs and Outputs	507
Modules	508
Top Level Wrapper (err_corr_top)	508
ECC-Based Error Correction (ecc_correct)	508
Error Look-up Table (syndrome_lut)	510
Audio Packet Parsing (aud_det)	512
FPGA Resources	515

Design Files	515
Conclusion	515

Chapter 23: Audio Demultiplexer for Standard-Definition Digital Video

Specifications	517
Features	517
Usage Models	519
One Video Stream with Video Rate Clock	519
One Video Stream with Faster Clock	519
Multiple Synchronous Video Streams	520
Multiple Asynchronous Video Streams	522
Clock Requirements for Processing Multiple Video Streams	524
Modules	524
I/O Port Description	526
Clock and Control Signals	526
Input Video Streams	527
Audio Sample Output Ports	528
Channel Pair Present Flags	530
Channel Pair Demultiplexer Control Ports	531
Input Packet Error Ports	532
Audio Control Packet Ports	534
Audio Packet Deletion Ports	535
Parameters	535
Sample Buffer Size	536
FPGA Resource Requirements	537
Theory of Operation	537
Overview	537
Input State Machine	539
Output State Machine	543
Audio Packet Deletion	544
Input Stream Control Module	544
Design Files	545
Conclusion	545

Chapter 24: Audio Multiplexer for 3G-SDI, Dual-Link HD-SDI, and HD-SDI Video

Introduction	547
Features	547
Embedded Audio	548
Audio Data Packets	549
32 KHz to 48 KHz Audio Data Packets	549
Audio Sample Words	550
Audio Clock Phase Data	550
Error Correction Codes	550
96 KHz Audio Data Packets	550
Audio Control Packets	551
Reference Design	553
Inputs and Outputs	555

Modules	559
Top-Level Module (hd_audio_mux_top).....	560
Data Packet Multiplexer (hd_audio_mux_data)	560
Control Packet Multiplexer (hd_audio_mux_contr).....	560
Ancillary Data Parsing (anc_det)	561
Audio Packet Multiplexer (packet_mux)	564
Clock Phase Determination (clock_phase)	565
Synchronizing One-Shot (sync_one_shot)	566
Audio Data Packet Construct (aud_data_constr)	567
Audio Control Packet Construct (aud_contr_constr).....	570
Audio FIFOs (aud_fifos).....	572
FIFO (fifo_28x16).....	573
Error Correction Encode (ecc_encode)	575
Format 96 Multiplexer (format96_mux).....	576
Split 96 Deserializer (split96_deser).....	577
Usage Models	578
Embedding Audio from a Synchronous Audio Source	579
Synchronizing Audio via Sample Rate Conversion	580
Embedding Asynchronous Audio.....	580
Replacing Audio Embedded in Received Video	581
Reembedding Audio after Separate Video/Audio Processing.....	581
FPGA Resources	581
Design Files	582
Conclusion	582

Chapter 25: Audio Demultiplexer for 3G-SDI, Dual-Link HD-SDI, and HD-SDI Video

Introduction	583
Features	583
Embedded Audio	584
Audio Data Packets	585
32 KHz to 48 KHz Audio Data Packets.....	585
Audio Sample Words	586
Audio Clock Phase Data.....	586
Error Correction Codes.....	586
96 KHz Audio Data Packets	586
Audio Control Packets	587
Reference Design	589
Inputs and Outputs	591
Modules.....	595
Top-Level Module (hd_audio_demux_top)	596
Data Packet Demultiplexer (hd_audio_demux_data).....	596
Control Packet Demultiplexer (hd_audio_demux_contr)	596
Audio Packet Parser (aud_det).....	596
Data Extraction (data_extr).....	598
Audio FIFOs (aud_fifos).....	600
FIFO (fifo_28x16).....	601
Timing Control (demux_tim).....	603
Synchronizing One-Shot (sync_one_shot)	605
Format 96 Demultiplexer (format96_demux).....	606
Split 96 Serializer (split96_ser).....	607

Usage Models	609
Synchronous De-embedding	611
Asynchronous De-embedding	611
De-embedding Audio for Separate Audio/Video Processing	611
Performance and Resources	612
Design Files	612
Conclusion	612

Section VI:

Miscellaneous Audio and Video Topics

Chapter 26: SMPTE 352M Video Payload Identification Packet Processing

Summary	615
SMPTE 352M Description	615
Byte 1	617
Byte 2	618
Byte 3	618
Byte 4	619
VPID Packet Location	619
Reference Design	620
SMPTE352_vpid_capture	620
SMPTE352_vpid_insert	623
Other Modules Used in the Reference Design	628
Module FPGA Resource Usage	628
Conclusion	628

Section VII:

Appendices

Appendix A: SMPTE Standards Related to SDI

Appendix B: Glossary

About This Guide

This application note describes how to use Virtex®-5 FPGAs to implement various serial digital video interfaces commonly used in the professional video broadcast industry.

Guide Contents

This document contains the following sections and chapters:

Section I: Introduction to the SMPTE Standards

- [Chapter 1, Introduction](#) describes the historical development of video interface standards.
- [Chapter 2, Introduction to the SMPTE Serial Digital Interface Standards](#) describes the SMPTE serial digital interface standards commonly used in the professional video broadcast industry.
- [Chapter 3, Implementing SMPTE Serial Digital Interfaces with RocketIO GTP Transceivers](#) describes how to use RocketIO™ GTP transceivers to implement the video broadcast serial digital interface standards.
- [Chapter 4, Implementing SMPTE Serial Digital Interfaces with RocketIO GTX Transceivers](#) serves as a general introduction to using Virtex-5 FPGA GTX transceivers to implement SMPTE SDI applications.

Section II: Multirate SD/HD/3G-SDI Using Virtex-5 FPGA RocketIO Transceivers

- [Chapter 5, Triple-Rate SDI Receiver for Virtex-5 LXT and SXT Devices](#) describes a triple-rate SDI receiver reference design that supports all three SMPTE serial digital video interface standards.
- [Chapter 6, Triple-Rate SDI Transmitter for Virtex-5 LXT and SXT Devices](#) describes a triple-rate SDI transmitter reference design that supports all three SMPTE serial digital video interface standards.
- [Chapter 7, Triple-Rate SDI Pass-through Design for Virtex-5 LXT and SXT Devices](#) describes a triple-rate SDI pass-through design that supports all three SMPTE serial digital video interface standards.
- [Chapter 8, Triple-Rate SDI for Virtex-5 FXT and TXT Devices](#) describes triple-rate SDI RX and TX reference designs for GTX transceivers.

Section III: SD-SDI Using Virtex-5 FPGA SelectIO LVDS

- [Chapter 9, SD-SDI Receiver](#) describes an SD-SDI receiver reference design for Virtex-5 FPGAs.
- [Chapter 10, SD-SDI Transmitter](#) describes an SD-SDI transmitter reference design for Virtex-5 FPGAs.
- [Chapter 11, SD-SDI Receiver/Transmitter Demonstration Design](#) describes an SD-SDI combined receiver and transmitter demonstration design for Virtex-5 FPGAs.

Section IV: DVB-ASI Using Virtex-5 FPGA SelectIO LVDS

- [Chapter 12, DVB-ASI Introduction and Layer 0 Implementation](#) introduces DVB-ASI and provides an implementation guide for layer 0.
- [Chapter 13, DVB-ASI Layer 1 and 2 Receiver](#) presents a DVB-ASI receiver design for the Virtex-5 FPGA that uses the Xilinx® SelectIO™ technology.
- [Chapter 14, DVB-ASI Layer 1 and 2 Transmitter](#) presents a DVB-ASI transmitter design for the Virtex-5 FPGA that uses Xilinx SelectIO technology.
- [Chapter 15, DVB-ASI Layer 1 and 2 Pass-through Demonstration Design](#) presents a DVB-ASI pass-through design for the Virtex-5 FPGA that uses Xilinx SelectIO technology.

Section V: AES Digital Audio

- [Chapter 16, Introduction to Digital Audio for Video Broadcasting](#) provides an introduction to digital audio standards such as AES3, SMPTE 272M, and SMPTE 299M.
- [Chapter 17, AES3 Serial Digital Audio Interfaces](#) describes how AES3 and S/PDIF receivers and transmitters can be implemented in Xilinx FPGAs.
- [Chapter 18, Asynchronous Sample Rate Converter](#) describes an asynchronous sample rate converter reference design implemented in the Virtex-5 FPGA.
- [Chapter 19, Multi-Channel Asynchronous Sample Rate Converter](#) describes a multi-channel asynchronous sample rate converter reference design implemented in the Virtex-5 FPGA.
- [Chapter 20, Audio Multiplexer for HD-SDI Video](#) describes how audio embedding is done and presents a hardware-verified reference design that implements an audio multiplexer for HD-SDI video.
- [Chapter 21, Audio Demultiplexer for HD-SDI Video](#) describes how audio demultiplexing is done and presents a hardware-verified reference design that implements an audio demultiplexer for HD-SDI video.
- [Chapter 22, Error Correction for HD-SDI Embedded Audio](#) introduces error correction concepts and methods relating to SMPTE 299M, the audio format standard for multiplexed audio in HD-SDI video.
- [Chapter 23, Audio Demultiplexer for Standard-Definition Digital Video](#) describes an audio demultiplexer for SD digital video.
- [Chapter 24, Audio Multiplexer for 3G-SDI, Dual-Link HD-SDI, and HD-SDI Video](#) presents a hardware-verified reference design that implements an audio multiplexer compatible with HD-SDI, 3G-SDI, and dual link HD-SDI video.

- [Chapter 25, Audio Demultiplexer for 3G-SDI, Dual-Link HD-SDI, and HD-SDI Video](#) presents a hardware-verified reference design that implements an audio demultiplexer compatible with HD-SDI, 3G-SDI, and dual link HD-SDI video.

Section VI: Miscellaneous Audio and Video Topics

- [Chapter 26, SMPTE 352M Video Payload Identification Packet Processing](#) describes reference designs that allow VPID packets to be captured from and inserted into video streams.

Section VII: Appendices

- [Appendix A, SMPTE Standards Related to SDI](#) lists some of the SMPTE standards related to SD-SDI, HD-SDI, dual link HD-SDI, and 3G-SDI.
- [Appendix B, Glossary](#) provides definitions of terms used in this application note.

References

This application note uses the following references:

1. [UG196](#), *Virtex-5 FPGA RocketIO GTP Transceiver User Guide*.
2. [DS202](#), *Virtex-5 FPGA Data Sheet: DC and Switching Characteristics*.
3. [UG198](#), *Virtex-5 FPGA RocketIO GTX Transceiver User Guide*.
4. [RPT113](#), *Virtex-5 FPGA RocketIO GTX Transceiver CEI-6G Electrical Specification Characterization Report*
5. [XAPP514](#), *Audio/Video Connectivity Solutions for Virtex-II Pro and Virtex-4 FPGAs*.
6. [XAPP875](#), *Dynamically Programmable DRU for High-Speed Serial I/O*.
7. [XAPP861](#), *Efficient 8X Oversampling Asynchronous Serial Data Recovery Using IDELAY*.
8. EN 50083-9, *Cabled Distribution Systems for Television, Sound and Interactive Multimedia Signals*, <http://www.dvb.org>.
9. Widmer, A. X., and P. A. Franaszek. *A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code* (IBM Journal of Research and Development, Vol. 27, Number 5, 1983).
10. AES3-2003, *AES Recommended Practice for Digital Audio Engineering—Serial transmission format for two-channel linearly represented digital audio data*, Audio Engineering Society, Inc., <http://www.aes.org>.
11. Tech. 3250-E—Third edition, *Specification of the Digital Audio Interface (The AES/EBU interface)*, European Broadcasting Union, <http://www.ebu.ch>.
12. AES18-1996 (r2002), *AES Recommended Practice for Digital Audio Engineering - Format for the user data channel of the AES3 digital audio interface*, Audio Engineering Society, Inc., <http://www.aes.org>.
13. AES5-2003, *AES Recommended Practice for Professional Digital Audio—Preferred sampling frequencies for applications employing pulse-code modulation*, Audio Engineering Society, Inc., <http://www.aes.org>.
14. [XAPP224](#), *Data Recovery*.
15. AES-3id-2001, *AES Information Document—Transmission of AES formatted data by unbalanced coaxial cable*, Audio Engineering Society, Inc., <http://www.aes.org>.
16. AES3-2003, *AES Recommended Practice for Digital Audio Engineering—Serial transmission format for two-channel linearly represented digital audio data*, Audio Engineering Society, Inc., <http://www.aes.org>.

Additional Resources

To find additional documentation, see the Xilinx website at:

<http://www.xilinx.com/support/documentation/index.htm>.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

<http://www.xilinx.com/support/mysupport.htm>.

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
Courier font	Messages, prompts, and program files that the system displays	<code>speed grade: - 100</code>
Courier bold	Literal commands that you enter in a syntactical statement	ngdbuild <i>design_name</i>
Helvetica bold	Commands that you select from a menu	File Open
	Keyboard shortcuts	Ctrl+C
Italic font	Variables in a syntax statement for which you must supply values	ngdbuild <i>design_name</i>
	References to other manuals	See the <i>Development System Reference Guide</i> for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are <i>not</i> connected.
Square brackets []	An optional entry or parameter. However, in bus specifications, such as bus [7:0] , they are required.	ngdbuild [<i>option_name</i>] <i>design_name</i>
Braces { }	A list of items from which you must choose one or more	lowpwr = { on off }
Vertical bar	Separates items in a list of choices	lowpwr = { on off }

Convention	Meaning or Use	Example
Vertical ellipsis . . .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis . . .	Repetitive material that has been omitted	allow block <i>block_name loc1 loc2 ... locn;</i>

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section “ Additional Resources ” for details. Refer to “ Title Formats ” in Chapter 1 for details.
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

Introduction

Television is without doubt one of the truly world-shaping inventions of the 20th century, and quite a number of individual engineers in different countries have contributed to its evolution. Many engineers disagree on the future of television, just as historians might disagree on the exact date the television concept was created. However, no one can argue about its impact on the lives of billions of people in today's world.

The first analog transmission networks for carrying television pictures and sound, just like the technology of television itself, began to come to life in the late 1930s with the invention of coaxial cable. By the 1990s, TV broadcast studios contained miles and miles of analog coaxial cable—but the world was fast becoming a digital landscape. To leverage the existing coaxial cable transport fabric, engineers created a method whereby digital streams of information could be transferred across legacy analog networks, and the Serial Digital Interface (SDI) transmission standard was born. Companies built entire product lines to support the explosive popularity of this new standard. Specialized digital chips called Application-Specific Standard Products (ASSPs) addressed the complex needs of the protocol and found their way into every SDI system where a coaxial cable began or ended.

As entertainment consumers demanded more and more realistic immersion into the television experience, broadcasting movies in wide-screen dimensions promised more income for the television industry. Video resolution would increase, and aging coaxial cables would be forced to carry even more information. Engineers first responded to this demand by extending the bandwidth of the original SDI standards by a factor of 5 ½ times, and a whole new standard for High-Definition SDI (HD-SDI) television transport quickly emerged, followed by a new generation of digital audio and video ASSPs.

The number of audio/video entertainment choices demanded by consumers was growing explosively, and the complexity of the transport networks had to keep up with this demand. Engineers responded by compressing digital video and audio information. The Digital Video Broadcast-Asynchronous Serial Interface (DVB-ASI) standard was developed to carry the newly compressed data over coaxial cables augmented with another new generation of ASSPs.

Modern computer networking technology has provided to the television industry an even lower-cost medium for transferring digital data: Ethernet. It is slowly creeping into broadcast studios and replacing the aging coaxial cables. The emerging standards for transmitting TV audio and video over an Ethernet fabric have been lumped together into the term Video over IP, meaning video and audio information transferred over an Ethernet fabric using the Internet Protocol (IP).

The ASSPs commonly used to implement digital video and audio transport at the hardware level had always been an expensive solution, and the industry needed another world-shaping technology to reduce the costs associated with digital transport. The Field-Programmable Gate Array (FPGA), invented by Xilinx in 1984, met these needs

perfectly. FPGAs are hardware “blank slates,” allowing engineers to innovate and try out new features inexpensively within the bounds of existing communication standards.

Xilinx FPGA designers embed key circuits inside the devices to help audio and video engineers with their creations. For example, the integration into Xilinx® FPGAs of RocketIO™ GTP transceivers and advanced SelectIO™ resources has enabled the support of many more networking standards than previously possible. This now includes HD-SDI, Standard Definition SDI (SD-SDI), and DVB-ASI, as well as others.

In addition, Xilinx application engineers support many audio and video standards with Verilog and VHDL reference designs that can be easily implemented in Xilinx FPGAs. These designs are available to customers at no cost.

This application note describes how to use Xilinx FPGAs to implement various serial digital video interfaces commonly used in the professional video broadcast industry. The serial video interfaces described in this document are:

- **SD-SDI:** Used to transport uncompressed standard-definition digital video.
- **HD-SDI:** Used to transport uncompressed high-definition digital video.
- **3G-SDI:** Used to transport uncompressed 1080p at 60 Hz digital video.
- **DVB-ASI:** Used to transport compressed digital video.
- **Audio Engineering Society (AES):** Used to transport digital audio.

These popular standards offer high-bandwidth transmission between various video applications in the headend, edit suite, or studio, and can represent a significant portion of the bill of materials, particularly when there is a need for multiple channels. Integrating the network protocols into a Xilinx FPGA reduces the overall system cost considerably. Basing a system on Xilinx FPGAs gives the system designer the flexibility to meet exacting requirements, supporting standard interfaces, but still enabling differentiation from competitive offerings.

For each of the serial video interface standards listed above, this application note describes how to implement the various functions needed in the transmitter and receiver. Reference designs are available in both Verilog and VHDL for these functions. Complete demonstration examples that run on Xilinx demonstration boards are also described. The source code for these demonstration examples is also available in both Verilog and VHDL.

Some of the interfaces, such as SD-SDI and DVB-ASI, can be implemented in most recent Xilinx FPGAs, including the Virtex®-4 and Virtex-5 FPGAs, and the extended Spartan®-3A family. The higher-speed serial interfaces of HD-SDI require the GTP transceivers in the Virtex-5 LXT, SXT, and FXT sub-families. GTP transceivers can also support the slower SD-SDI and DVB-ASI standards.

In addition to compression, the 3 Gb/s transceiver technology facilitates doubling the bandwidth of HD-SDI to support the emerging 3 Gb/s SDI (3G-SDI) standards. The primary Xilinx offering is termed “triple-rate SDI.” This code should be used because it has had the most testing to date.

The term “triple rate” means that the design supports SD-SDI, HD-SDI, and 3G-SDI with a single RocketIO Multi-Gigabit Transceiver (MGT). The design also contains code that supports dual link HD-SDI integrated with triple-rate SDI.

The code is offered in “full featured” and “light” versions. Most users will find that the light version fits their needs at a reduced amount of logic. The full-featured version allows easy manipulation of the standards that need more than 10 bits of data.

Section I: Introduction to the SMPTE Standards

***Audio/Video Connectivity Solutions
for Virtex-5 FPGAs***

Introduction to the SMPTE Serial Digital Interface Standards

Introduction

The SDI standards are the predominant standards for uncompressed digital video interfaces in the broadcast studio and video production center. The first SDI standard, SD-SDI, allowed standard-definition digital video to be transported over the coaxial cable infrastructure initially installed in studios to carry analog video. Next, HD-SDI was introduced to support high-definition video. Finally, dual link HD-SDI and 3G-SDI doubled the bandwidth of HD-SDI to support 1080p (50 Hz and 60 Hz) and other video formats requiring more bandwidth than HD-SDI provides.

The Society of Motion Picture and Television Engineers (SMPTE) standards were created to allow SDI interfaces to carry a wide variety of data associated with or in place of uncompressed digital video. Multiple channels of audio are usually carried along with the digital video. Many other types of ancillary data, such as time code and captioning, are also often embedded in the digital video streams carried on SDI interfaces. Compressed video can be mapped onto SDI interfaces. In fact, there are now over 40 SDI-related standards from SMPTE. The International Telecommunication Union (ITU) has equivalents to many of the SMPTE SDI standards. Other organizations, such as the European Broadcasting Union (EBU), have published SDI-related reports and other documents.

This chapter serves as an introduction to SMPTE SDI, providing background information useful for those readers new to these standards.

SD-SDI

SD-SDI interfaces carry uncompressed standard-definition digital video and related ancillary data serially on coaxial cable. SD-SDI is defined by the SMPTE 259M and International Telecommunications Union Radiocommunication Sector (ITU-R) BT.656 standards.

SD-SDI Bit Rates

The SMPTE 259M standard defines four bit rates for SDI ranging from 143 Mb/s to 360 Mb/s. Another standard, SMPTE 344M, adds a 540 Mb/s bit rate for SD-SDI.

The 270 Mb/s bit rate is, by far, the most commonly used SD-SDI bit rate. This bit rate supports both NTSC and PAL resolution 4:2:2 Y'C_B'C_R' digital component video sampled at 13.5 MHz. The 270 Mb/s bit rate is so predominant that most SD-SDI applications only require this bit rate. Supporting only 270 Mb/s simplifies SDI applications because it

reduces the number of reference clock frequencies needed by the SDI receivers and transmitters.

SD-SDI Physical Layer

The SD-SDI signal is a singled-ended (also called unbalanced) signal carried on a single 75Ω coaxial cable terminated with BNC connectors. The signal amplitudes at the transmitter output are typically 800 mV peak-to-peak centered around 0V. Rise and fall times are restricted to be no less than 400 ps and no more than 1.5 ns.

SD-SDI receivers and transmitters are always AC coupled to the BNC connectors using capacitors. These AC coupling capacitors are typically in the range of 1 μF to 10 μF, which is quite large for AC coupling capacitors in serial interfaces.

The large size of the coupling capacitors is dictated by the pathological waveforms and long run lengths found in SD-SDI. Large capacitors are required to prevent droop on the output side of the capacitor during long runs without a bit transition.

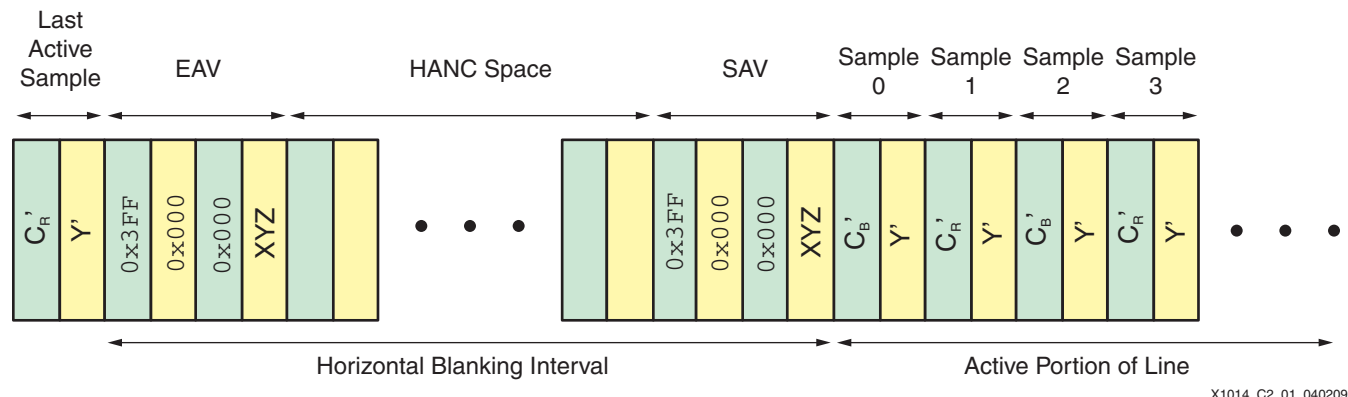
SD-SDI receivers can typically support cable lengths of up to 300m, depending on the quality of the cable. Adaptive cable equalizers are almost always used in the receivers to compensate for the signal attenuation and frequency-dependent phase distortion caused by long cables. With high-quality coaxial cable and modern SD-SDI cable equalizers, SD-SDI receivers can often support cable lengths in excess of 400m.

SD-SDI Data Stream Format

The most common video format carried by SD-SDI is 4:2:2 Y'C_B'C_R' digital component video with 10-bit components (at either PAL or NTSC resolution). The SD-SDI data stream can carry 8-bit video, but the stream has 10-bit words even if 8-bit video components are present.

An SD-SDI transmitter provides a single 10-bit input port with an input data rate of one-tenth the SD-SDI serial bit rate. The input data rate is also equal to twice the video sample rate. Similarly, the output port of an SD-SDI receiver is a single 10-bit data port.

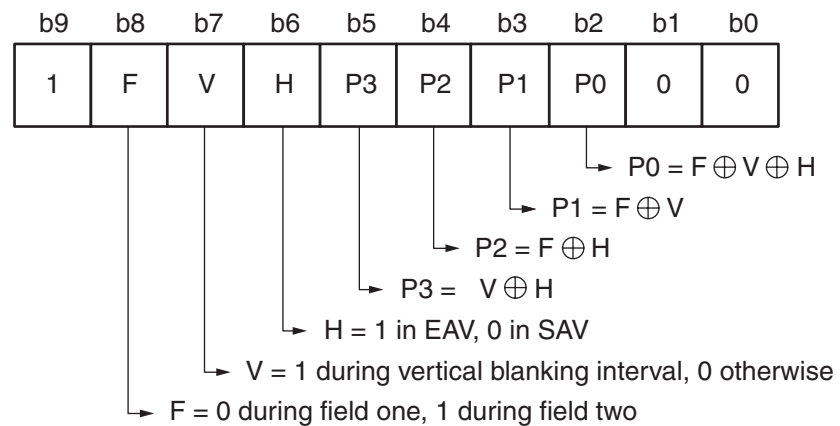
As shown in Figure 2-1, the “active” portion of the data stream contains interleaved Y' and C components, where the C component words alternate between C_B' and C_R'. The active portion begins with the start-of-active-video (SAV) sequence and ends with the end-of-active-video (EAV) sequence. A digital video line is considered to start with the first word of the EAV.



X1014_C2_01_040209

Figure 2-1: SD-SDI Data Stream Format

The SAV and EAV sequences are both four words. The first three words of both sequences are the same: one word of all ones followed by two words of all zeros (0x3FF, 0x000, 0x000). The fourth word is called the XYZ word. The XYZ word contains video timing bits called F, V, and H, as well as some protection bits that allow errors in the F, V, and H bits to be detected. See Figure 2-2 for details of the XYZ word format. The space between the EAV and SAV sequences is called the horizontal ancillary (HANC) space. Ancillary (ANC) data packets containing non-video data are often found in the HANC space. Some types of ancillary data can also be found in the active portion of lines in the vertical blanking interval.



X1014_C2_02_040209

Figure 2-2: XYZ Word Format

SD-SDI Encoding

The SD-SDI transmitter converts the 10-bit data stream into a serial bitstream. The 10-bit words from the data stream are sent LSB first on the serial interface. The transmitter encodes the SD-SDI data stream using a pseudorandom scrambler followed by a non-return-to-zero (NRZ) to non-return-to-zero-interleaved (NRZI) conversion, as shown in Figure 2-3. This figure shows the encoding scheme as if it were implemented in bit-serial fashion, where the \oplus symbol represents XOR gates and the boxes represent flip-flops. The reverse of the encoding scheme is implemented at the receiver to decode the data.

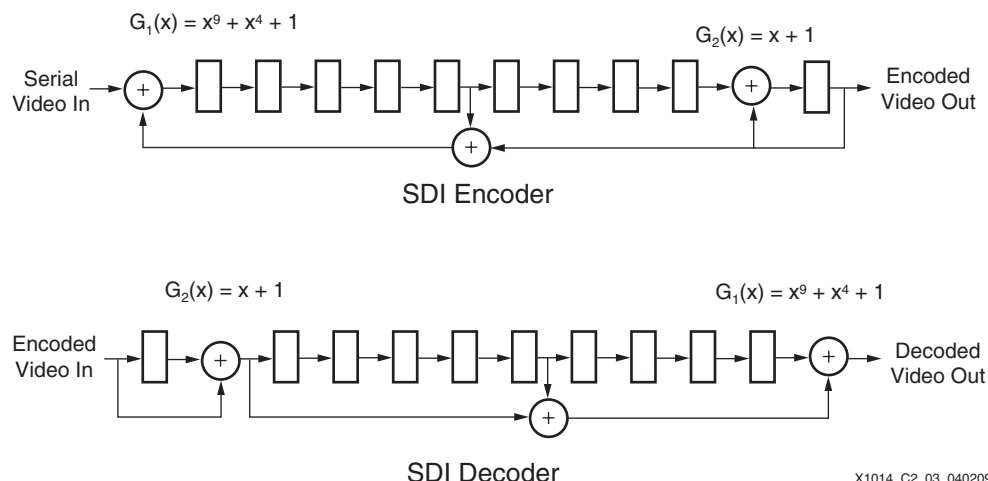


Figure 2-3: SDI Encoding and Decoding

The encoding scheme shown in Figure 2-3 prevents excessively long run lengths without bit transitions. It is an efficient encoding scheme because it carries no overhead, and there is no increase in the amount of data transmitted due to encoding. Because the output of the encoder is an NRZI signal, it is polarity free. This means that it can be inverted between the transmitter and the receiver and the receiver still decodes it correctly.

This scrambling scheme does produce some pathological waveforms. There are two pathological waveforms of interest, as defined in SMPTE RP 178.

One pathological waveform is poorly DC balanced and stresses the cable equalizer. This pattern is the top waveform in Figure 2-4. The waveform has one High bit followed by 19 Low bits. This pattern can be repeated consecutively for the entire active portion of one video line. The opposite polarity is equally possible (one Low bit followed by 19 High bits).

The second pathological waveform, shown at the bottom of Figure 2-4, is a square wave with a period 40 bits long. This square wave can repeat for the entire active portion of one video line. Because it has a low density of transitions, this waveform can cause problems for the clock and data recovery section of the SD-SDI receiver.

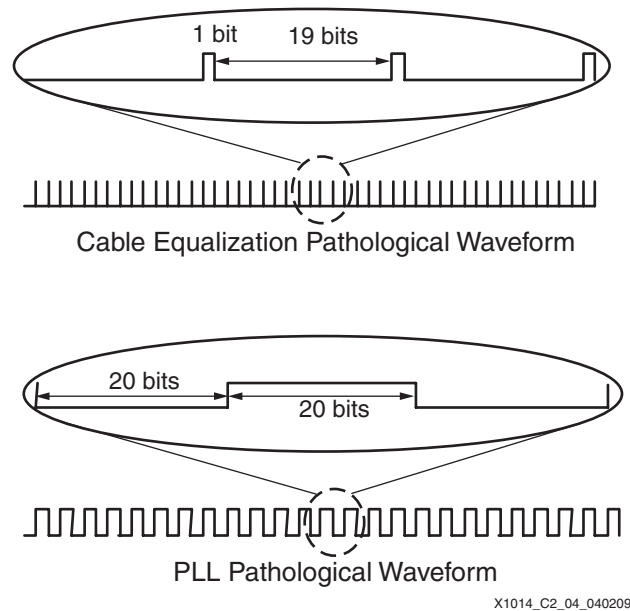


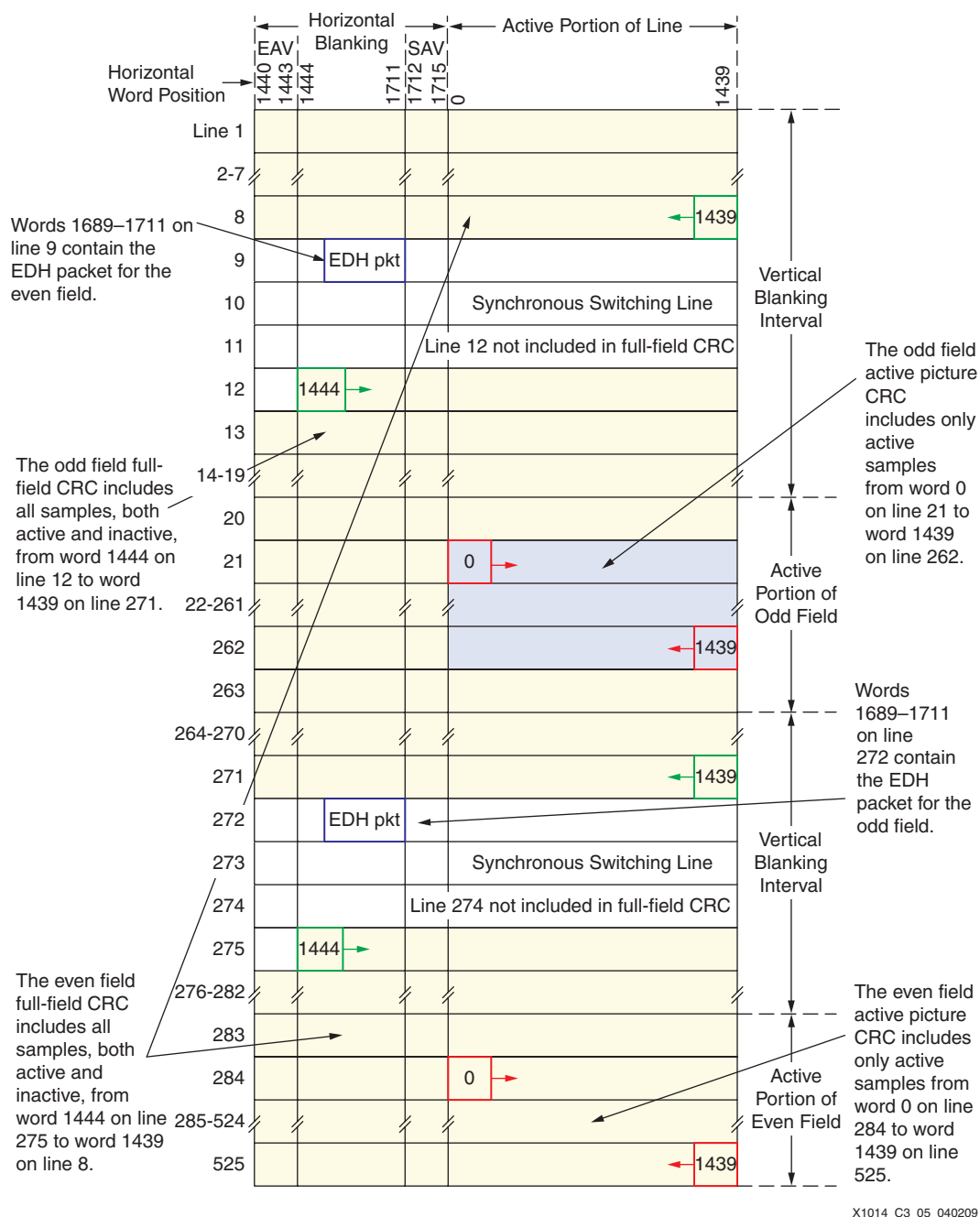
Figure 2-4: SDI Pathological Waveforms

Error Detection for SD-SDI

Error detection in SD-SDI is managed by the error detection and handling (EDH) protocol, as defined in SMPTE RP 165. SD-SDI does not allow retransmission of fields that contain errors, and no error correction codes are present. Thus, EDH is used for error detection only. Use of EDH is common, but not required.

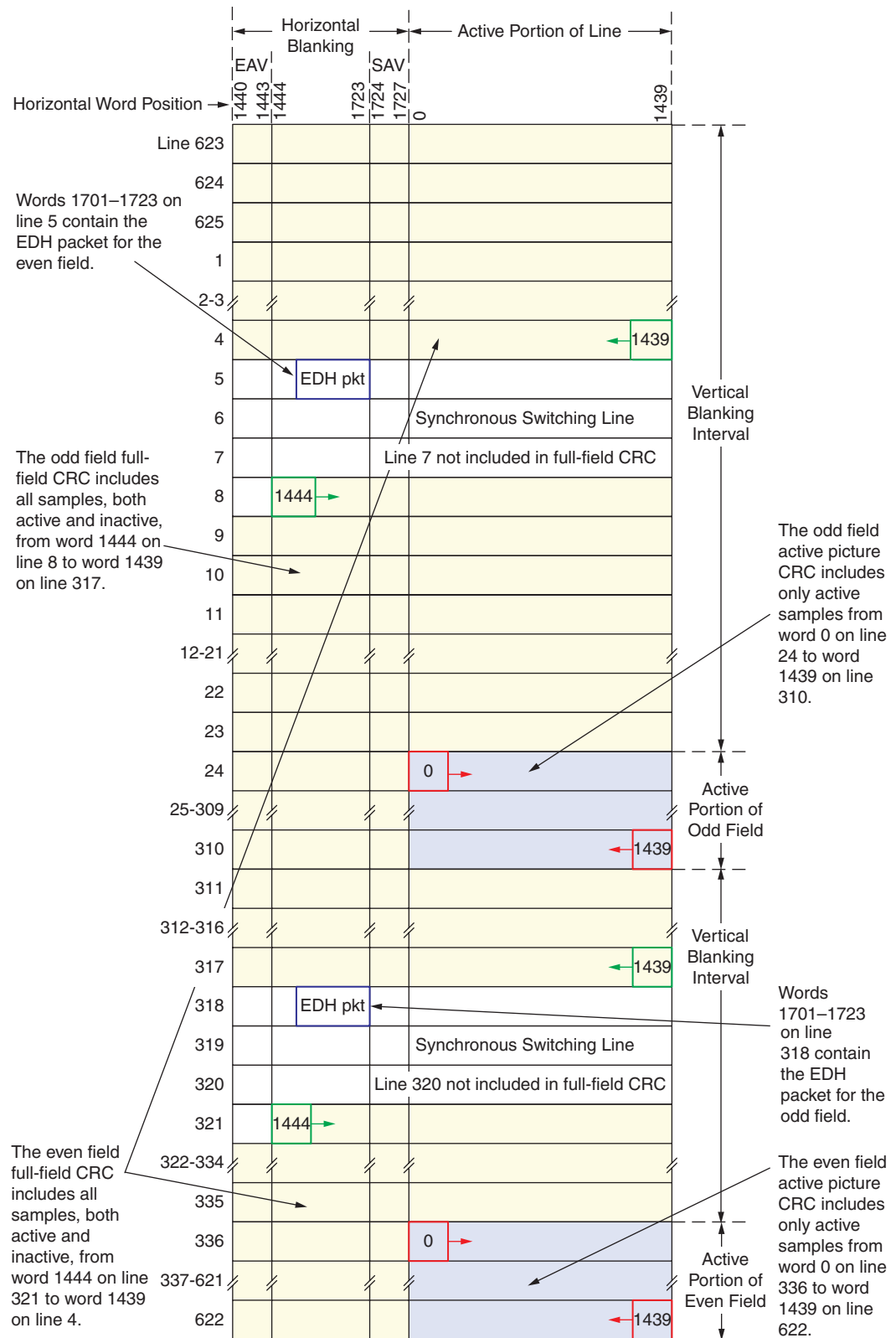
Two cyclic redundancy check (CRC) checkwords are created for each video field. The active picture (AP) checkword is calculated from the active picture area (the active portions of the active video lines). The full-field (FF) checkword is calculated from most, but not all, of the data in the field. The FF checkword omits several lines of video around the synchronous switching line. Figure 2-5 shows the areas included in the FF and AP CRC checkwords for NTSC video. Figure 2-6 shows the same thing for PAL video.

These figures also show the position of the EDH packets. There is an EDH packet for each video field. The EDH packet is formatted as a standard ancillary data packet. However, the EDH packet is unique as its position violates the convention that ancillary data packets in the HANC space of a video line are placed contiguously, starting with the first word of the HANC space. EDH packets must be placed in the HANC space where the last word of the packet is in the last word of the HANC space, right before the SAV sequence.



X1014_C3_05_040209

Figure 2-5: SD-SDI EDH CRC Checkword Calculations for NTSC Video



X1014_C22_06_040209

Figure 2-6: SD-SDI EDH CRC Checkword Calculations for PAL Video

The CRC checkwords are 16-bit values calculated using the CRC-CCITT polynomial generation method, as shown in Figure 2-7.

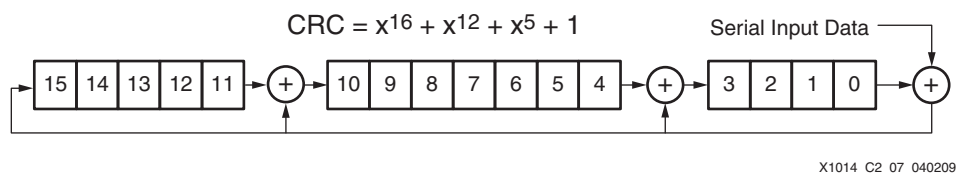


Figure 2-7: SD-SDI EDH CRC Word Generation

In addition to the FF and AP CRC checkwords, the EDH packet includes three sets of error flags. One set is associated with the AP checkword, another set is associated with the FF checkword, and the third set is associated with ANC data packets. Each set of error flags consists of five individual flags:

- **Error Detected Here (edh)**
If the receiving equipment calculates a CRC value from the previous field that differs from the CRC checkword found in the EDH packet, the edh flag must be set to 1. Additionally, if a checksum error was detected in one or more ANC packets in the previous field, the edh flag of the ANC data packet set must be set to 1.
- **Error Detected Already (eda)**
This flag indicates that some upstream piece of equipment has detected an error. A piece of equipment that receives an EDH packet with the edh flag set by the upstream device must set the eda flag to 1 in the packet. It should also clear the edh flag to 0 unless the equipment detects an error in the CRC checkword or ANC checksum, in which case both the eda and edh flags must be set to 1.
- **Internal Error Detected Here (idh)**
The idh flag is provided as a signaling mechanism to allow video equipment to communicate the occurrence of an internal error unrelated to the SDI data stream. For example, the idh flag would be asserted if a piece of equipment detected an overheating condition.
- **Internal Error Detected Already (ida)**
This flag indicates that some upstream piece of equipment has detected an internal error. If a piece of equipment processes an EDH packet with the idh flag set by the upstream device, it must set the ida flag to 1 and clear the idh flag to 0—unless the equipment also detects an internal error, in which case both the ida and idh flags must be set to 1.
- **Unknown Error Status (ues)**
If a video device receives an SD-SDI signal without EDH packets, it can create new EDH packets and insert them into the SD-SDI data stream. In this case, the device can set the ues flag to 1 to indicate to downstream devices that, at some point in the chain, the video signal was not protected by the EDH protocol and could contain undetected errors.

The flag pairs, edh/eda and idh/ida, can be used to track down a faulty device in a chain of video equipment, as shown in Figure 2-8. If the eda flag is set at any point in the chain, it is known that some upstream device detected an error. If the errors occur repeatedly, each device in the equipment chain can be checked to see where the eda flag changes to an edh flag. The piece of equipment or connection proceeding the device asserting the edh flag is likely to have caused the error.

Devices are not required to support the flags from all three sets. Any unsupported flag must be cleared to 0.

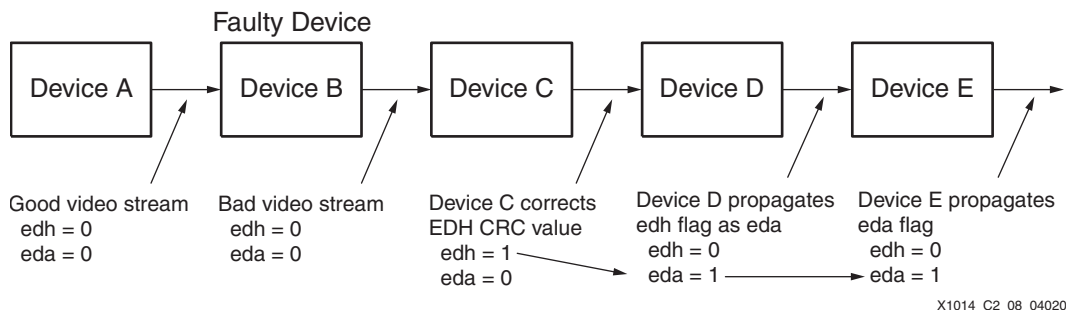


Figure 2-8: SD-SDI EDH Error Flag Forwarding

Figure 2-9 shows the format of the EDH packet. The AP and FF CRC checkwords each have an associated valid bit. The EDH standard allows a piece of equipment to support only one CRC checkword, not both. A CRC checkword that is not calculated must have its valid bit (the V bit in the checkword) cleared to 0.

Word Contents	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Ancillary Data Flag, Word 1 (0x0)	0	0	0	0	0	0	0	0	0	0
Ancillary Data Flag, Word 2 (0x3FF)	1	1	1	1	1	1	1	1	1	1
Ancillary Data Flag, Word 3 (0x3FF)	1	1	1	1	1	1	1	1	1	1
Data ID (0x1F4)	0	1	1	1	1	1	0	1	0	0
Block Number	1	0	0	0	0	0	0	0	0	0
Data Count (16 Words of User Data)	0	1	0	0	0	1	0	0	0	0
Active-picture CRC bits [5:0]	\overline{P}	P	ap5	ap4	ap3	ap2	ap1	ap0	0	0
Active-picture CRC bits [11:6]	\overline{P}	P	ap11	ap10	ap9	ap8	ap7	ap6	0	0
Active-picture CRC bits [15:12]	\overline{P}	P	V	0	ap15	ap14	ap13	ap12	0	0
Full-field CRC bits [5:0]	\overline{P}	P	ff5	ff4	ff3	ff2	ff1	ff0	0	0
Full-field CRC bits [11:6]	\overline{P}	P	ff11	ff10	ff9	ff8	ff7	ff6	0	0
Full-field CRC bits [15:12]	\overline{P}	P	V	0	ff15	ff14	ff13	ff12	0	0
Ancillary Data Error Flags	\overline{P}	P	0	ues	ida	idh	eda	edh	0	0
Active-picture Error Flags	\overline{P}	P	0	ues	ida	idh	eda	edh	0	0
Full-field Error Flags	\overline{P}	P	0	ues	ida	idh	eda	edh	0	0
Reserved Words (7 total)	1	0	0	0	0	0	0	0	0	0
Checksum	$\overline{S8}$	S8	S7	S6	S5	S4	S3	S2	S1	S0

Notes:

- 1) P is an even parity bit for bits b7 through b0 and is located in b8. Words containing a P bit in b8 also have the inverse of b8 located in b9.
- 2) Each CRC value has an associated valid bit (V). If the CRC value is valid, V is set to 1.

X1014_C2_09_040209

Figure 2-9: EDH Packet Format

HD-SDI

HD-SDI is defined by the SMPTE 292M standard. It carries uncompressed high-definition video serially over coaxial cable. It is similar enough to SD-SDI that interfaces can be designed that support both SD-SDI and HD-SDI.

HD-SDI Bit Rates

HD-SDI has only two bit rates. The first is 1.485 Gb/s. This bit rate supports video with field or frame rates of exactly 60 Hz, 50 Hz, 30 Hz, 25 Hz, and 24 Hz.

The second bit rate is 0.1% less than the first bit rate, and is typically written as 1.485/1.001 Gb/s (1.485 divided by 1.001). This bit rate is approximately 1.4835 Gb/s. Because the value 1.001 is used so often, video broadcast technical literature often uses the letter “M” to stand for 1.001. Thus, the second HD-SDI bit rate can also be written as 1.485/M Gb/s. This bit rate supports video with field or frame rates of 60/M Hz, 30/M Hz, and 24/M Hz, or approximately 59.94 Hz, 29.97 Hz, and 23.98 Hz.

SMPTE refers to HD-SDI as a 1.5 Gb/s serial interface. This is only an approximation of the bit rates actually supported by HD-SDI. HD-SDI does not support a bit rate of exactly 1.5 Gb/s.

HD-SDI Physical Layer

The HD-SDI physical layer is almost identical to SD-SDI. Like SD-SDI, it uses 75Ω coaxial cable terminated with BNC connectors.

The electrical characteristics of the HD-SDI signal are almost identical to SD-SDI, with the notable exception of the rise-time and fall-time specifications. For HD-SDI, the rise and fall times must be between 100 ps and 270 ps. This difference in the rise and fall times between HD-SDI and SD-SDI does require that any transmitter supporting both SDI standards have selectable output slew rates.

HD-SDI Data Stream Format

The parallel interface to an HD-SDI receiver or transmitter consists of two 10-bit data streams, as shown in [Figure 2-10](#). Data stream 1 is often called the “Y data stream” and data stream 2 is often called the “C data stream”. This is because when uncompressed 4:2:2 Y'C_B'C_R' digital component video is transported by the interface, the Y' component is carried on data stream 1, and the C_B' and C_R' components are interleaved on data stream 2.

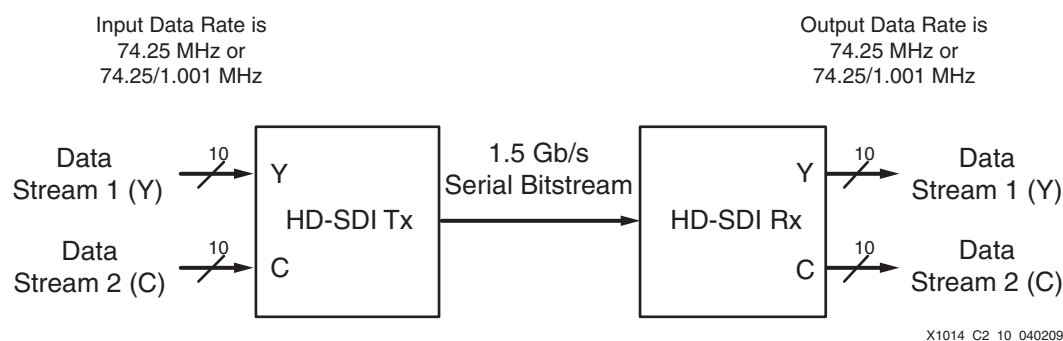
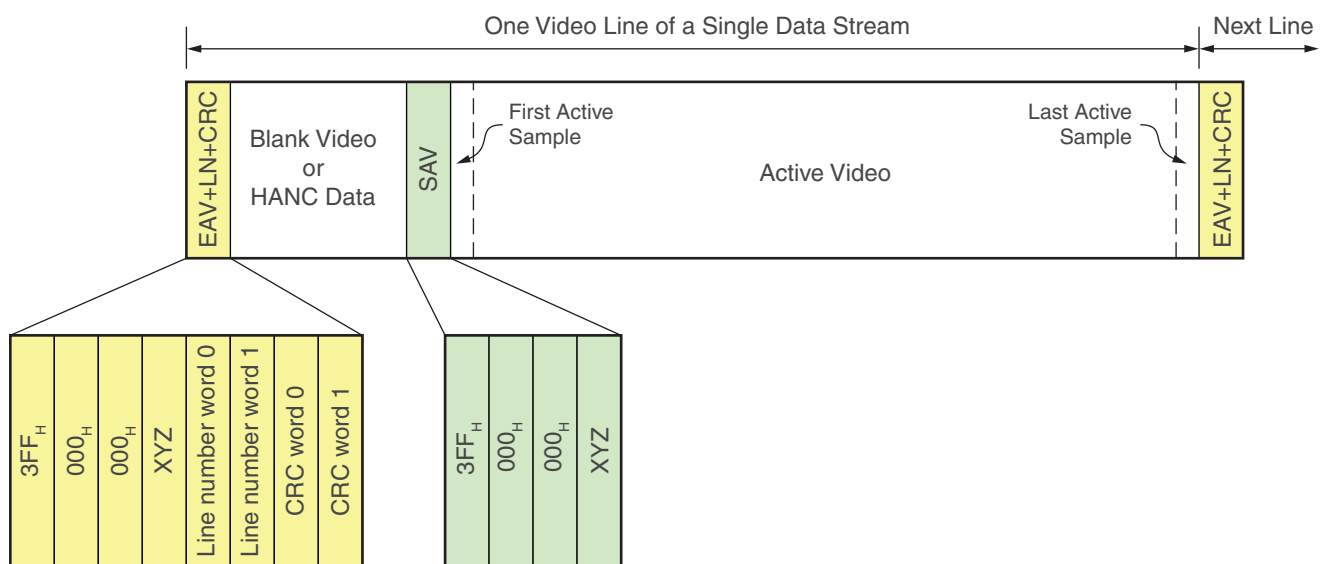


Figure 2-10: HD-SDI Interfaces

Regardless of whether the data mapped onto the HD-SDI interface is uncompressed video or something else, it must always be divided into equal length data structures called lines. A line is the fundamental unit of the HD-SDI transport data stream. When carrying uncompressed video, the transport lines correspond to the horizontal lines of the picture. A line is divided into two main areas: the horizontal blanking interval and the active video as shown in Figure 2-11. The line begins with an EAV timing reference. The EAV also begins the horizontal blanking interval. The horizontal blanking interval ends with the SAV timing reference. Immediately following the SAV is the active video area. The active video area ends immediately before the EAV of the next line.

Figure 2-11 shows the line format for one 10-bit HD-SDI data stream. Both data streams of the HD-SDI interface must have exactly same format. Furthermore, the two data streams must be perfectly synchronized (i.e., the EAV, SAV, blanking interval, and active video areas of the two data streams must be perfectly aligned). No skew is allowed between the two HD-SDI interface data streams.



X1014_C2_11_040209

Figure 2-11: HD-SDI Line Format

The HD-SDI EAV and SAV sequences are formatted identically to the EAV and SAV sequences in SD-SDI. However, the EAV of every line of an HD-SDI data stream is immediately followed by two words carrying the line number (LN) and two words carrying a CRC checkword.

The 11-bit line number is formatted into two LN words as shown in Figure 2-12. Both data streams must have identical line numbers. Line numbers are assigned sequentially beginning with 1 and ranging to the maximum line number for the particular video format.

	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Line number word 0:	Not b8	LN6	LN5	LN4	LN3	LN2	LN1	LN0	0	0
Line number word 1:	1	0	0	0	LN10	LN9	LN8	LN7	0	0

X1014_C2_12_040209

Figure 2-12: HD-SDI Line Number Format

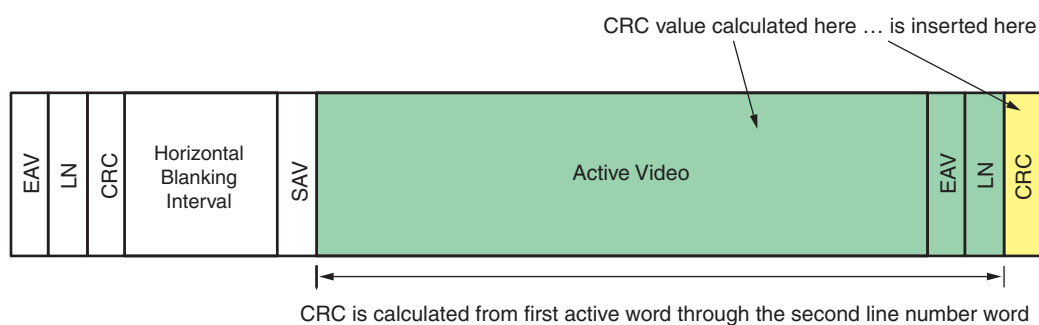
Immediately following the two LN words on each line are two words containing an 18-bit CRC checkword for the previous line. The HD-SDI receiver uses the CRC checkword to detect transmission errors. Because retransmission of a line is not permitted by HD-SDI and the protocol does not include error correction codes, the receiver can do nothing to restore the integrity of the data in the line if it detects an error. Devices with HD-SDI interfaces sometimes report the total number of lines or frames with CRC errors detected by the receiver. Such error reports can be used to detect a faulty link in a chain of equipment so that it can be repaired or replaced.

As with the LN field, the two data streams each have their own CRC values, which are calculated using the CRC polynomial given in Equation 2-1.

$$CRC = x^{18} + x^5 + x^4 + 1 \quad \text{Equation 2-1}$$

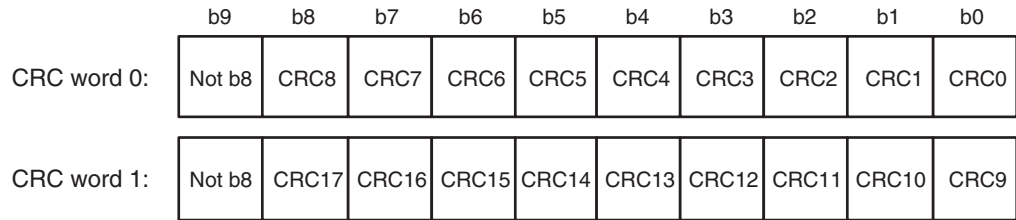
The CRC calculation for a line begins with the first word in the active video and includes all subsequent words up to and including the second LN word on the next line, as shown in Figure 2-13. The SAV and horizontal blanking intervals are not included in the CRC calculation. Ancillary data placed in the horizontal blanking interval usually has its own error detection scheme.

The two CRC words that follow the LN words are formatted as shown in Figure 2-14.



X1014_C2_13_040209

Figure 2-13: HD-SDI CRC Calculation

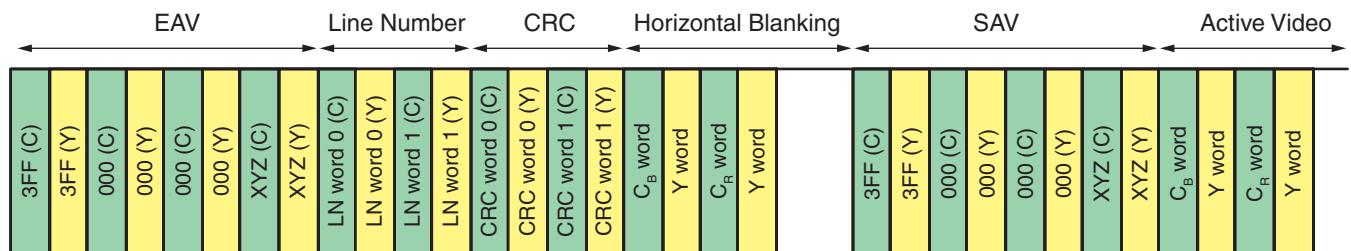


X1014_C2_14_040209

Figure 2-14: HD-SDI CRC Format

As with SD-SDI, the HANC space between the EAV and SAV of each line often contains ancillary data packets. However, with HD-SDI, there are two HANC spaces per line—one in each data stream. The standards for different types of ancillary data packets usually restrict each particular ancillary packet type to either the Y data stream or the C data stream.

Prior to scrambling the data streams for transmission over the HD-SDI serial interface, the two data streams are interleaved together as shown in Figure 2-15. For each video sample, the word from data stream 2 (C) is sent first, followed by the word from data stream 1 (Y). HD-SDI is an LSB-first protocol, so the LSB of each word is sent first by the serializer.



X1014_C2_15_041509

Figure 2-15: HD-SDI Data Stream Interleaving

HD-SDI Encoding

HD-SDI serial bitstreams are encoded in exactly the same manner as SD-SDI bitstreams. HD-SDI uses the same pseudorandom scrambler with the same polynomial as SD-SDI, and this is followed by an NRZ-to-NRZI conversion. Because the same encoding scheme is used, HD-SDI has exactly the same pathological waveforms as SD-SDI.

Video Formats Supported by HD-SDI

The uncompressed HD video formats supported by HD-SDI use the $Y'C_B'C_R'$ color space, 4:2:2 sampling, and have 10 bits per component (20 bits per video sample). The most common of these video formats are defined by these standards:

- SMPTE 274M formats with 1080 active lines
- SMPTE 296M formats with 720 active lines

Other sampling structures such as 4:4:4, and other bit depths such as 12 bits per component require more bandwidth than provided by HD-SDI. These other sampling structures and bit depths are supported by dual link HD-SDI and 3G-SDI.

An older version of the HD-SDI standard also identified SMPTE 295M as a supported video format. However, starting with the 2006 revision of the HD-SDI standard, SMPTE 295M is no longer listed as a supported data format.

The HD-SDI standard does identify SMPTE 260M as a supported video format. Although not common today, this video format once had more support than SMPTE 295M and can be found in some legacy equipment.

SMPTE 274M defines the 1080-line high-definition digital video formats widely used by the broadcast industry. The various formats of 1080i and 1080p video defined by SMPTE 274M are shown in Table 2-1. Systems 1, 2, and 3 are not supported by HD-SDI because they exceed the bandwidth capabilities of HD-SDI. However, they can be carried by dual link HD-SDI or 3G-SDI. Systems 3 through 11 can all be carried by HD-SDI as long as there are only 20 bits per sample.

Table 2-1: SMPTE 274M Video Formats (1920 x 1080 active samples)

System	Format Name	Sample Rate (MHz)	Frame Rate	Total Samples x Lines
1	1080p60	148.5	60	2200 x 1125
2	1080p59.94	148.5/M	60/M	2200 x 1125
3	1080p50	148.5	50	2640 x 1125
4	1080i60	74.25	30	2200 x 1125
5	1080i59.94	74.25/M	30/M	2200 x 1125
6	1080i50	74.25	25	2640 x 1125
7	1080p30	74.25	30	2200 x 1125
8	1080p29.97	74.25/M	30/M	2200 x 1125
9	1080p25	74.25	25	2640 x 1125
10	1080p24	74.25	24	2750 x 1125
11	1080p23.98	74.25/M	24/M	2750 x 1125

Notes:

1. The value M indicates the value 1.001.

SMPTE 274M also allows the 1080p progressive video formats to be transported in a manner that looks like interlaced video, in effect allowing the 1080p video to masquerade as 1080i video. In this case, the source picture is progressive, but it is mapped to an interlaced format for transportation. This is called progressive segmented frame (PsF).

Some early HD video equipment, particularly video tape recorders, could not support 1080p video, but could process 1080i video. By taking each progressive frame and splitting it into two fields, it is possible to make the 1080p video look like 1080i video for equipment that cannot process 1080p video. This is not the same as converting a progressive video frame into two truly interlaced fields, which requires modifying the video samples using a filter. PsF retains the full progressive nature of the video. Converting 1080p to 1080 PsF involves splitting each progressive frame into two fields without modifying the video samples. Again, this is done to make the PsF video masquerade as 1080i video for compatibility with legacy video equipment not capable of supporting 1080p video directly. To be viewed properly, the two PsF fields must be recombined into one progressive frame. The fields cannot be properly displayed as if they were actually two interlaced fields.

SMPTE 274M defines five PsF formats, as shown in Table 2-2. The Progressive Video System Number column indicates which 1080p video system from Table 2-1 is matched by the original progressive video format. Similarly, the Lookalike Interlaced System column indicates which 1080i system from Table 2-1 is mimicked by the PsF video format. Systems D and E do not have corresponding 1080i formats.

Table 2-2: SMPTE 274M PsF Video Formats

System	Format Name	Progressive Video System Number from Table 2-1	Lookalike Interlaced System from Table 2-1
A	1080PsF30	7	4
B	1080PsF29.97	8	5
C	1080PsF25	9	6
D	1080PsF24	10	N/A
E	1080PsF23.98	11	N/A

SMPTE 296M defines 720p video formats commonly used in the broadcast industry today. These formats are shown in Table 2-3. All formats shown are progressive and have 720 active lines per frame and 1280 active samples per line. All SMPTE 296M video formats can be carried by HD-SDI as long as there are only 20 bits per video sample.

Table 2-3: SMPTE 296M Video Formats (1280 x 720 Active Samples)

System	Format Name	Sample Rate (MHz)	Frame Rate (Hz)	Total Samples x Lines
1	720p60	74.25	60	1650 x 750
2	720p59.94	74.25/M	60/M	1650 x 750
3	720p50	74.25	50	1980 x 750
4	720p30	74.25	30	3300 x 750
5	720p29.97	74.25/M	30/M	3300 x 750
6	720p25	74.25	25	3960 x 750
7	720p24	74.25	24	4125 x 750
8	720p23.98	74.25/M	24/M	4125 x 750

Dual Link HD-SDI

SMPTE 372M is the dual link HD-SDI standard. Two HD-SDI interfaces are combined together to form an interface set providing twice the bandwidth of a regular HD-SDI interface. The two links of the dual link HD-SDI interface are referred to as link A and link B.

Dual link HD-SDI can support video formats that require more bandwidth than is available with a single HD-SDI interface. SMPTE 372M defines how to map the 1080i and 1080p video formats of SMPTE 274M to a dual link HD-SDI interface. The 720p video formats of SMPTE 296M are not supported by SMPTE 372M.

SMPTE 372M requires SMPTE 352M video payload ID ancillary data packets on certain lines in each field or frame. These packets not only identify the characteristics of the video

format being transported, but also uniquely identify the A and B links. By observing these packets, a dual link HD-SDI receiver can correctly receive the video signal even if the two links are swapped on the receiver inputs.

When two separate interfaces are bonded together as in dual link HD-SDI, some skew inevitably occurs between the two links. The dual link HD-SDI receiver must allow for this skew and resynchronize the data from the links.

3G-SDI

3G-SDI is very similar to HD-SDI except that the bit rate is twice that of HD-SDI, or approximately 3 Gb/s. Like HD-SDI, two bit rates are supported: 2.97 Gb/s and 2.97/1.001 Gb/s (approximately 2.967 Gb/s). 3G-SDI uses exactly the same electrical signal levels, scrambling, connectors, and cable as HD-SDI. However, the cable drivers and cable equalizers must be designed to support 3 Gb/s signal rates.

The 3G-SDI interface is defined by SMPTE 424M. Another document, SMPTE 425M, defines how various video formats are mapped to the 3G-SDI interface.

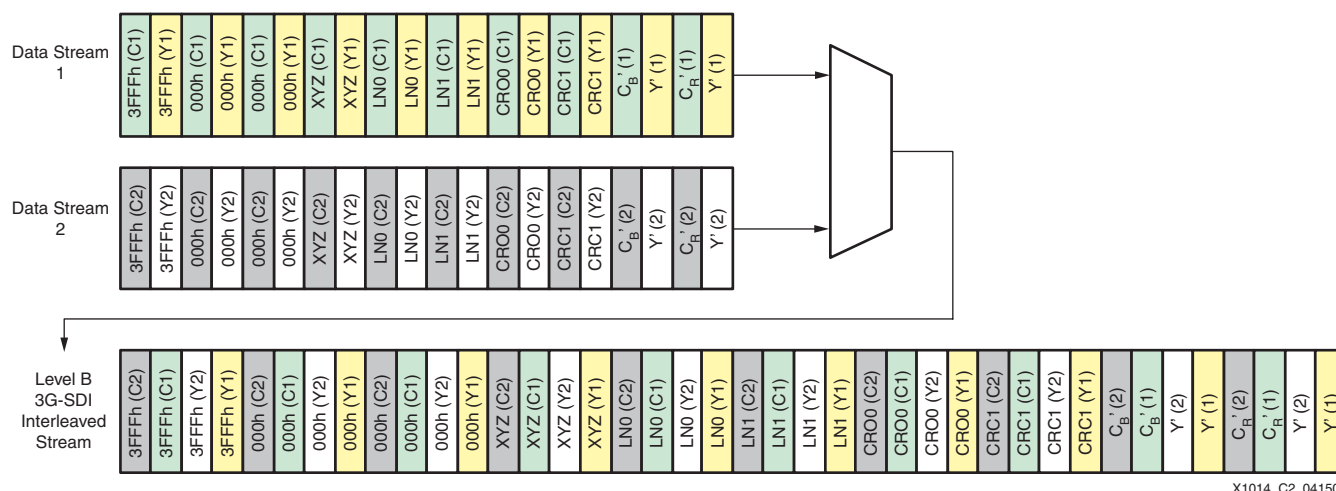
The 3G-SDI source image format mapping document, SMPTE 425M, defines two levels of operation called level A and level B. Level A defines four ways of mapping various video formats to the 3G-SDI interface, depending on the video format, sampling structure, and component bit depth. Level B allows very simple conversions between dual link HD-SDI and 3G-SDI.

The 3G-SDI level A data transport data structure is identical to that of HD-SDI, consisting of two 10-bit data streams. The data carried on each data stream is divided into lines, and each line is further divided into areas by the EAV and SAV timing references. The EAV is followed by two words of line number and two words of CRC. Thus, the format of each 10-bit data stream of the 3G-SDI interface is identical to the HD-SDI data stream shown in [Figure 2-11, page 37](#). The only difference is that the data rate of the 3G-SDI interface is twice that of the HD-SDI interface.

Level B also has two 10-bit data streams. When a dual link HD-SDI signal is carried by a 3G-SDI level B interface, link A of the dual link HD-SDI pair is carried on data stream 1 of the 3G-SDI interface and link B is carried on data stream 2.

Level B also supports the mapping of two independent HD-SDI video signals, rather than a dual link HD-SDI link pair, onto one 3G-SDI interface. However, the two HD-SDI video signals must always have identical timing and be perfectly synchronized (i.e., the EAV and SAV timing references of the two streams must be perfectly aligned).

Level B data streams are somewhat different from level A and HD-SDI data streams. Whereas the EAV and SAV sequences of the 3G-SDI level A interleaved data streams are identical to those of HD-SDI ([Figure 2-15, page 39](#)), the EAV and SAV sequences after interleaving are different in level B. This is because level B interleaves, word for word, two data streams that are themselves interleaved HD-SDI streams. Thus, the interleaved level B data stream is really a four-way interleave, with the two Y and C data streams from the two HD-SDI streams all interleaved together. Two interleaved HD-SDI data streams, identical in format to the interleaved HD-SDI data stream shown in [Figure 2-15, page 39](#), are shown at the top of [Figure 2-16](#). Below them is the level B 3G-SDI data stream that results from interleaving the two HD-SDI data streams.



X1014_C2_041509

Figure 2-16: 3G-SDI Level B Data Stream Interleaving

In an HD-SDI serial bitstream or a 3G-SDI level A serial bitstream, an EAV sequence appears at the receiver as a sequence of twenty consecutive 1 bits followed by forty consecutive 0 bits. However, in a level B 3G-SDI serial bitstream, the EAV sequence appears as forty consecutive 1 bits followed by eighty consecutive 0 bits. The framer function in the 3G-SDI receiver must be able to detect these longer EAV and SAV sequences. Also, the receiver must take care to process the LN and CRC values properly because in level B 3G-SDI, the LN and CRC words are in a four-way interleave, rather than the usual two-way interleave pattern.

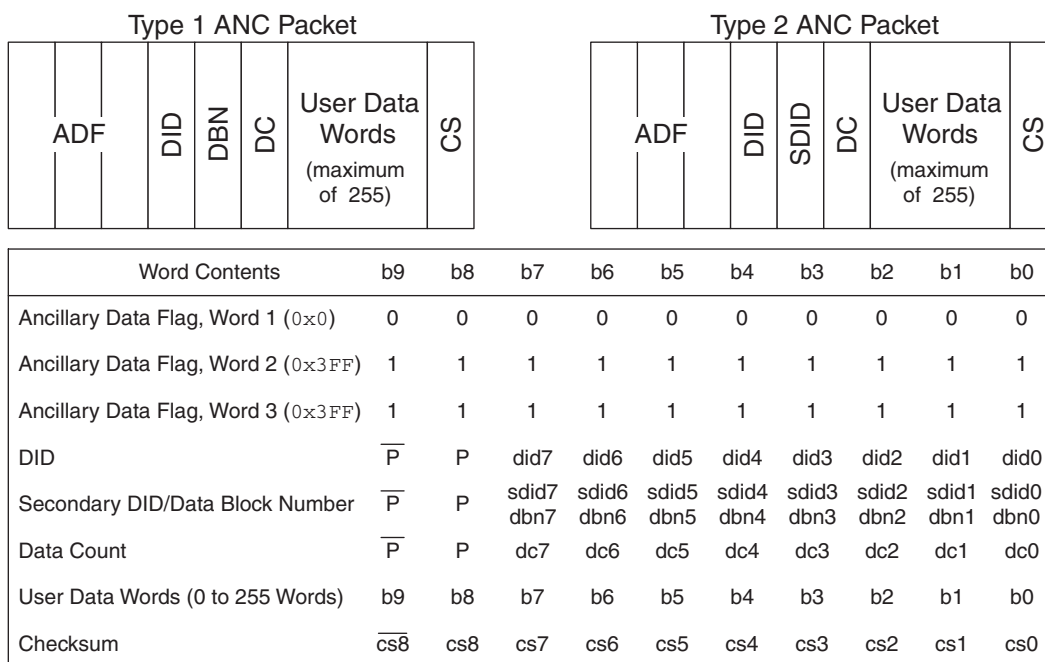
Ancillary Data

Ancillary data is non-video packetized data embedded in the horizontal blanking interval or in the active portions of video lines that are in the vertical blanking interval. If embedded in the horizontal blanking interval, the ancillary data is referred to as HANC data. If embedded in the vertical blanking interval, it is referred to as vertical ANC (VANC). Digital audio packets are a very common type of ANC packet to embed in the video stream.

SMPTE 291M defines the basic format of ANC packets and describes the rules for how the packets are embedded in the HANC and VANC spaces. ANC packets begin with a three-word preamble called the ancillary data flag (ADF). The first word is 0x000 and the second and third words are 0x3FF. Thus, the ADF is essentially the inverse of the first three words of an EAV or SAV sequence. Following the ADF is a data identification word (DID) that uniquely identifies the packet type. Following the DID is either a secondary DID (SDID) to further refine the identity of the packet type, or a data block number (DBN) used to identify a sequence of related packets. Next is the data count (DC) word, indicating how many user data words are present in the payload portion of the packet. The DC word is immediately followed by the first user data word, if any. Following the last user data word is a checksum. If there are no user data words, the checksum follows the DC word.

The length of an ANC packet is variable and is always seven words plus the user data words. The format of the user data words is not defined in SMPTE 291M, other than to specify that they must not be 0x000 or 0x3FF to avoid a sequence of user data words appearing to be an EAV, SAV, or ADF sequence. The actual format of the user data words is defined in each individual standard that defines a particular type of ancillary data packet.

For example, SMPTE 299M defines the format of the user data words for ANC packets containing digital audio for HD-SDI interfaces. Figure 2-17 illustrates the ANC packet format.



Note: P is an even parity bit for bits b7 through b0 and is located in b8. Words containing a P bit in b8 also have the inverse of b8 located in b9.

X1014_C2_17_040209

Figure 2-17: ANC Packet Format

Conclusion

This chapter introduces the basic concepts of the SMPTE SDI standards. Refer to other chapters of this document for details of implementing the various SDI standards with Xilinx® FPGA devices. Refer to [Appendix A, SMPTE Standards Related to SDI](#) for a listing of some of the SMPTE standards related to SD-SDI, HD-SDI, dual-link HD-SDI, and 3G-SDI. A glossary is provided in [Appendix B, Glossary](#).

Implementing SMPTE Serial Digital Interfaces with RocketIO GTP Transceivers

Introduction

Virtex®-5 LXT and SXT devices have RocketIO™ GTP serial transceivers capable of operating at speeds up to 3.75 Gb/s. These transceivers are power-efficient and highly configurable. They can be used to implement the serial digital interface standards commonly used in the video broadcast industry:

- SD-SDI
- HD-SDI
- 3G-SDI
- DVB-ASI

This chapter introduces using the GTP transceivers to implement these video broadcast serial digital interface standards (generically referred to as SDI standards in this chapter). The material in this chapter is common to all of the SDI standards. Details specific to any particular standard are found in other chapters in this application note.

A number of features of the GTP transceiver in the Virtex-5 FPGA are particularly useful when implementing the SDI standards:

- A single reference clock frequency allows reception of both HD-SDI bit rates, both 3G-SDI bit rates, and 270 Mb/s SD-SDI and DVB-ASI.
- All receivers can be completely independent, allowing them to independently switch between HD-SDI, 3G-SDI, SD-SDI, and DVB-ASI.
- Just two reference clock frequencies, 148.5 MHz and 148.5/1.001 MHz (or 74.25 MHz and 74.25/1.001 MHz), allow transmission of both HD-SDI bit rates, both 3G-SDI bit rates, and 270 Mb/s SD-SDI and DVB-ASI.
- The GTP transceivers are low power (less than 100 mW per receiver/transmitter pair at 3 Gb/s).
- A dynamic reconfiguration port (DRP) allows the receiver (RX) and transmitter (TX) units in each GTP transceiver to be dynamically switched between SDI standards and also allows dynamic switching of reference clocks to the GTP transceivers.
- Every GTP_DUAL tile has an external reference clock input and access to reference clocks from tiles above and below it.

Basics of Using the GTP Transceiver to Implement SDI

This section briefly introduces the architecture and features of the GTP transceiver, emphasizing those features used when implementing the SDI standards. Refer to the *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* [Ref 1] for complete details of the GTP transceiver.

GTP_DUAL Tiles

Pairs of GTP transceivers are grouped into tiles called GTP_DUAL tiles. Each of these tiles contains two receivers and two transmitters that all share a common clock multiplication phase-locked loop (PLL). Different Virtex-5 devices have different numbers of GTP_DUAL tiles. However, in all Virtex-5 LXT and SXT devices, the GTP_DUAL tiles are all aligned along one edge of the device.

The clock multiplication PLL is shared by the entire GTP_DUAL tile. Sharing the GTP transceiver PLL places some restrictions on implementing different standards in different units within the tile.

The two receivers in the GTP_DUAL tile can independently receive any of the 3G-SDI and HD-SDI bit rates, and SD-SDI or DVB-ASI at 270 Mb/s. The receivers can switch between the various standards independently, with no restrictions.

The two transmitters in the GTP_DUAL tile must always operate at exact multiples of the same reference clock. So, for example, it is not possible to have one transmitter in the tile operating at 1.485 Gb/s while the other operates at 1.485/1.001 Gb/s. These two bit rates require different reference clock frequencies for the transmitters. Because there is a single reference clock frequency available in the tile at any one time, both transmitters must use this reference clock frequency. This also means that it is not possible to transmit 1.485 Gb/s with one transmitter and 1.485 Gb/s + 10 ppm (for example) with the other transmitter in the same tile. Both transmitters run at exact multiples of the single reference clock frequency. Applications that require all transmitters to be completely independent are restricted to using a single transmitter unit per GTP_DUAL tile.

Note: Changing the GTP_DUAL tile's reference clock frequency (for example, from 74.25 MHz to 74.25/1.001 MHz) to change the transmitter's bit rate usually upsets the operation of the receivers for a short period of time. The duration and severity of this upset has not been characterized. Therefore, Xilinx recommends that SDI transmitters should be isolated in their own GTP_DUAL tile, unless the application can tolerate a brief disturbance of the receivers each time the reference clock frequency to the GTP_DUAL tile is changed.

Applications requiring support for SD-SDI bit rates other than 270 Mb/s are much more restricted than those that only support SD-SDI at 270 Mb/s. The 270 Mb/s rate uses the same reference clock frequency as is used for HD-SDI and 3G-SDI. However, the other SD-SDI bit rates, less commonly used in the broadcast industry, each require different reference clock frequencies, preventing other units in the same tile from simultaneously operating at HD-SDI or 3G-SDI bit rates.

GTP_DUAL Clock Multiplication PLL

The shared PLL in the GTP_DUAL tile is used to multiply the reference clock up to the proper frequency required to support reception and transmission at the desired bit rate. The PLL is not part of the GTP receiver's clock and data recovery (CDR) section. It is used strictly as a clock multiplication PLL to provide a reference clock to the CDR. The GTP RX can receive bitstreams that are over ± 1000 ppm away from the PLL clock frequency. The GTP transmitter, on the other hand, uses the clock produced by the PLL directly as the

clock for the serializer. Thus, the transmitters always run at exact multiples of the reference clock frequency.

For SDI applications, the PLL must be given a reference clock of 148.5 MHz or 148.5/1.001 MHz. Alternatively, 74.25 MHz and 74.25/1.001 MHz can be used. SDI receivers can use any one of these reference clock frequencies to receive SD-SDI at 270 Mb/s, both HD-SDI bit rates, and both 3G-SDI bit rates. Thus, only a single reference clock frequency is required for a triple-rate SDI receiver and it can be any one of 148.5 MHz, 148.5/1.001 MHz, 74.25 MHz, or 74.25/1.001 MHz. SDI transmitters, however, always transmit at a bit rate that is an exact multiple of the reference clock frequency supplied to the PLL. To transmit SD-SDI, the reference clock frequency must be 148.5 MHz (or 74.25 MHz). To transmit HD-SDI at 1.485 Gb/s or 3G-SDI at 2.97 Gb/s, the reference clock frequency must be 148.5 MHz (or 74.25 MHz). To transmit HD-SDI at 1.485/1.001 Gb/s or 3G-SDI at 2.97/1.001 Gb/s, the reference clock frequency must be 148.5/1.001 MHz (or 74.25/1.001 MHz).

Figure 3-1 shows the PLL section as configured for a typical SDI application. The PLL multiplies the reference clock by 10 or 20 to generate a 1.485 GHz PLL clock. When using the /1.001 reference clock frequencies, all frequencies shown in the figure are divided by 1.001.

Note: The reference clock frequencies used for SDI can either be 148.5 MHz and 148.5/1.001 MHz or 74.25 MHz and 74.25/1.001 MHz. The only difference is the setup of the PLL_DIVSEL_FB attribute. However, using 74.25 MHz makes it more difficult to bypass the TX buffer, requiring the use of a digital clock manager (DCM) or PLL with each TX unit. This requirement is eliminated by using reference clock frequencies of 148.5 MHz and 148.5/1.001 MHz instead of 74.25 MHz and 74.25/1.001 MHz, or by enabling the TX buffer. See [GTP Transmitter, page 58](#) for details.

Each receiver unit has a PLL clock divider that divides the PLL clock by 1, 2, or 4. To receive 3G-SDI and 270 Mb/s SD-SDI or DVB-ASI, this divider is set to 1. For HD-SDI, the divider is set to 2. The serial portion of the receiver runs at twice the frequency of the clock output by this divider (symbolically shown in the figure by the x2 block, but actually accomplished by using both edges of the clock). The parallel section of the receiver runs at one-fifth of this clock rate.

The transmitters have a common divider, which for SDI is set to divide by 1. This is followed by individual dividers which, like those for the receivers, are set to divide by 1 for 3G-SDI, SD-SDI, and DVB-ASI, and by 2 for HD-SDI.

The PLL_RXDIVSEL_OUT and PLL_TXDIVSEL_OUT attributes can be changed dynamically through the dynamic reconfiguration port (DRP). Thus, with the PLL section setup as shown in Figure 3-1, it is simple to change the clock dividers for each individual RX or TX unit to change between 3G, HD, and SD modes. Other attributes must also be modified when changing between these modes, but changing the PLL clocks is very straightforward.

The CLKIN signal is output from the GTP_DUAL tile as REFCLKOUT. This output is always equal in frequency to CLKIN and is not affected by any of the multiplier or divider settings of the PLL.

The PLL attributes described in this section are set up in the GTP wrapper files produced by the RocketIO GTP Transceiver Wizard. Furthermore, they are dynamically controlled by the DRP controller located in the `v5gtp_sdi_control` module.

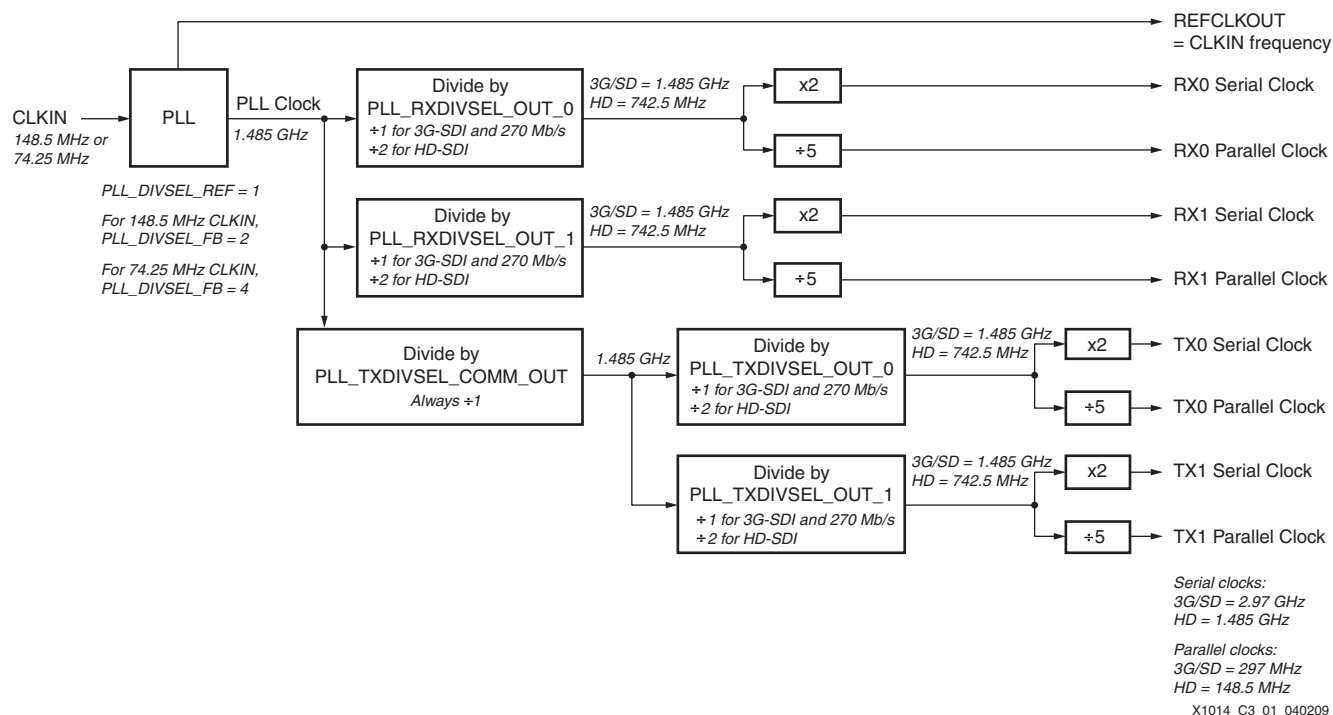


Figure 3-1: PLL Section

GTP Reference Clocks

Each GTP_DUAL tile must be provided with a reference clock. Each tile has a reference clock multiplexer that selects the reference clock from one of four sources:

- An external differential reference clock input (CLKP/N)
- A clock from the tile above (CLKINSOUTH)
- A clock from the tile below (CLKINNORTH)
- A global clock tree (GREFCLK)

Figure 3-2 shows the reference clock selection multiplexer and clock routing resources of a GTP_DUAL tile. The use of GREFCLK as a reference clock to the GTP_DUAL tile for SDI applications has not been characterized. A global clock tree connected as a reference clock to the GTP transceiver through the GREFCLK port has more jitter than a reference clock connected to the FPGA through a dedicated GTP transceiver reference clock external input (CLKP/N). This can make it difficult to meet HD-SDI and 3G-SDI TX output jitter specifications, if GREFCLK is used as the reference clock source.

Each GTP_DUAL tile has an external differential clock input (CLKP/N), which is a dedicated input. This clock input cannot be used as a general-purpose FPGA I/O pin, and it cannot be used to directly provide a clock to the global clock routing resources of the FPGA. However, it is possible to get reference clocks from the GTP_DUAL tile to the global clock resources using the REFCLKOUT port of the GTP_DUAL tile. Whatever clock is selected by a GTP_DUAL tile's reference clock multiplexer is output on that tile's REFCLKOUT port.

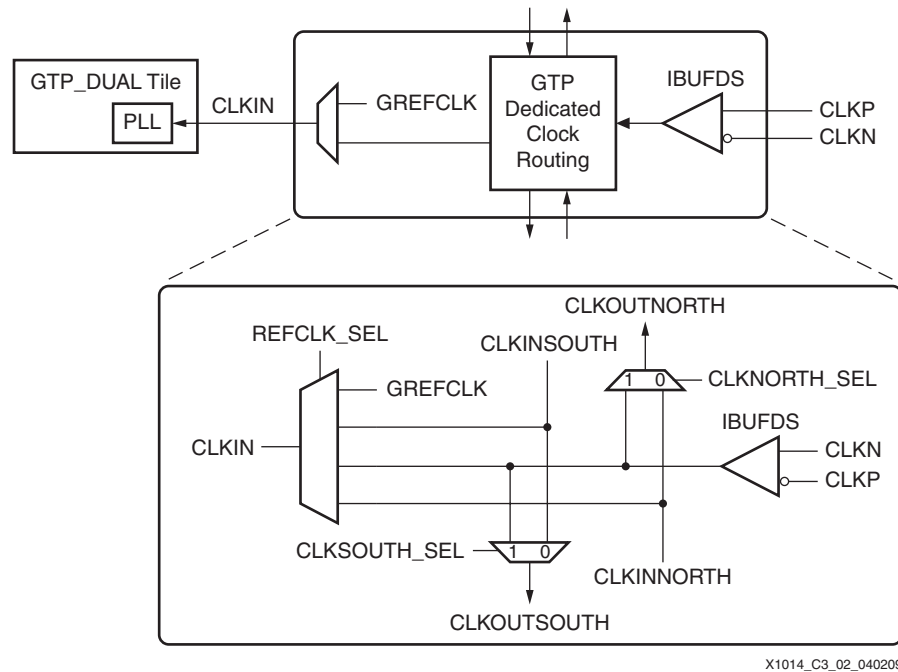


Figure 3-2: GTP Reference Clock Routing and Selection Multiplexer

The CLKP/N external clock input is connected to one input of the tile's reference clock multiplexer. It is also connected to multiplexers that drive the tile's CLKOUTNORTH and CLKOUTSOUTH output ports that provide reference clocks to the tile above and the tile below. The CLKOUTNORTH port drives a reference clock to the tile above and can be driven by either the CLKP/N external clock input or by the CLKINNORTH signal from the tile below. Likewise, the CLKOUTSOUTH port drives a reference clock to the tile below and can either be driven by the CLKP/N external clock input or by the CLKINSOUTH port from the tile above.

The CLKSOUTH and CLKNORTH routing resources provide a flexible way to drive multiple GTP_DUAL tiles from a single external reference clock input. Using the multiplexers in each tile to control the signals output on the CLKOUTSOUTH and CLKOUTNORTH ports of each tile, the CLKSOUTH and CLKNORTH routing resources can be broken up into multiple segments, allowing a high degree of flexibility in clock routing.

There are some limitations on the routing of clocks using the CLKSOUTH and CLKNORTH routing resources. A sourcing GTP_DUAL tile is limited to driving three contiguous GTP_DUAL tiles above it and three contiguous GTP_DUAL tiles below. Thus, a maximum of seven contiguous tiles can use an external reference clock input: the sourcing tile, three tiles above the sourcing tile, and three tiles below the sourcing tile. Figure 3-3 shows how one external reference clock output can source seven tiles. The red lines show how the reference clock entering the FPGA at GTP_DUAL tile D is connected to the three tiles above it using the CLKNORTH routing and the three tiles below it using the CLKSOUTH routing.

In Figure 3-3, there is a single CLKSOUTH and a single CLKNORTH routing resource between any two adjacent tiles. Thus, it is not possible to send two reference clocks in the same direction (north or south) between any two adjacent tiles. For example, there is no way to get a clock entering the FPGA at tile F's CLKP/N input to a tile located below tile G because the CLKOUTSOUTH resource is used between tile F and tile G. However, a clock

entering the FPGA at tile G can drive tiles below tile G because the CLKSOUTH routing resource is free below this tile.

Also, tile D blocks any other reference clocks from being routed through that tile because both its CLKOUTNORTH and CLKOUTSOUTH ports are used. Tile D could, however, receive a reference clock from the tiles above or below it.

These are static restrictions. However, the multiplexers that drive the CLKOUTNORTH and CLKOUTSOUTH ports of each tile and the clock selection multiplexer of each tile can be dynamically switched through the DRP, allowing more complex dynamic clock management strategies, if needed.

It is possible to route a reference clock through an unused GTP_DUAL tile. However, the unused tile MUST be instantiated in the design and powered. If it is not, the tile is powered down and no reference clocks can pass through the tile using the CLKNORTH and CLKSOUTH routing resources. See [GTP Transceiver Reference Clock Connections, page 86](#) for strategies describing how to connect reference clocks to the GTP_DUAL tiles in HDL designs. The *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* [Ref 1] describes the power requirements for an unused GTP_DUAL tile used to route reference clocks. The MGTAVTTRX and MGTAVTTTX pins must be powered, but power filtering is not required.

If a single reference clock source must be connected to more tiles than can be reached via the internal CLKNORTH and CLKSOUTH routing resources (more than seven contiguous tiles), it is necessary to connect that clock source to multiple GTP transceiver external clock inputs.

Any reference clock sources connected to the external CLKP/N differential input of any GTP_DUAL tile must be differential and must meet the specifications and requirements given in the *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* and the *Virtex-5 FPGA Data Sheet: DC and Switching Characteristics* [Ref 2]. In particular, the reference clock source must be AC coupled to the clock input pins. The input buffer provides internal termination and a common mode bias voltage for the AC-coupled input clock signal.

The *Virtex-5 FPGA Data Sheet* specifies a maximum reference clock jitter of 40 ps peak-to-peak. This specification should be met in all SDI applications. However, SDI applications often source GTP reference clocks using genlock solutions that might not meet this jitter requirement. It is important to understand how reference clock jitter affects the performance of the GTP receivers and transmitters.

The GTP receiver is relatively immune to reference clock jitter, up until the point where the tile's clock multiplication PLL cannot lock to the reference clock due to excessive jitter. The GTP transmitter's output jitter is directly impacted by the reference clock jitter. SD-SDI and DVB-ASI transmitters have very large absolute jitter budgets and, therefore, can operate with relatively large amounts of reference clock jitter. HD-SDI transmitters have a much smaller absolute jitter budget, and 3G-SDI transmitters have an even smaller absolute jitter budget. Thus, low-jitter reference clock sources are essential for HD-SDI and, even more so, for 3G-SDI.

All designers planning to exceed the maximum reference clock jitter specification are encouraged to consult with their local Xilinx FAE to determine how exceeding this jitter specification can affect the performance of the GTP receivers and transmitters.

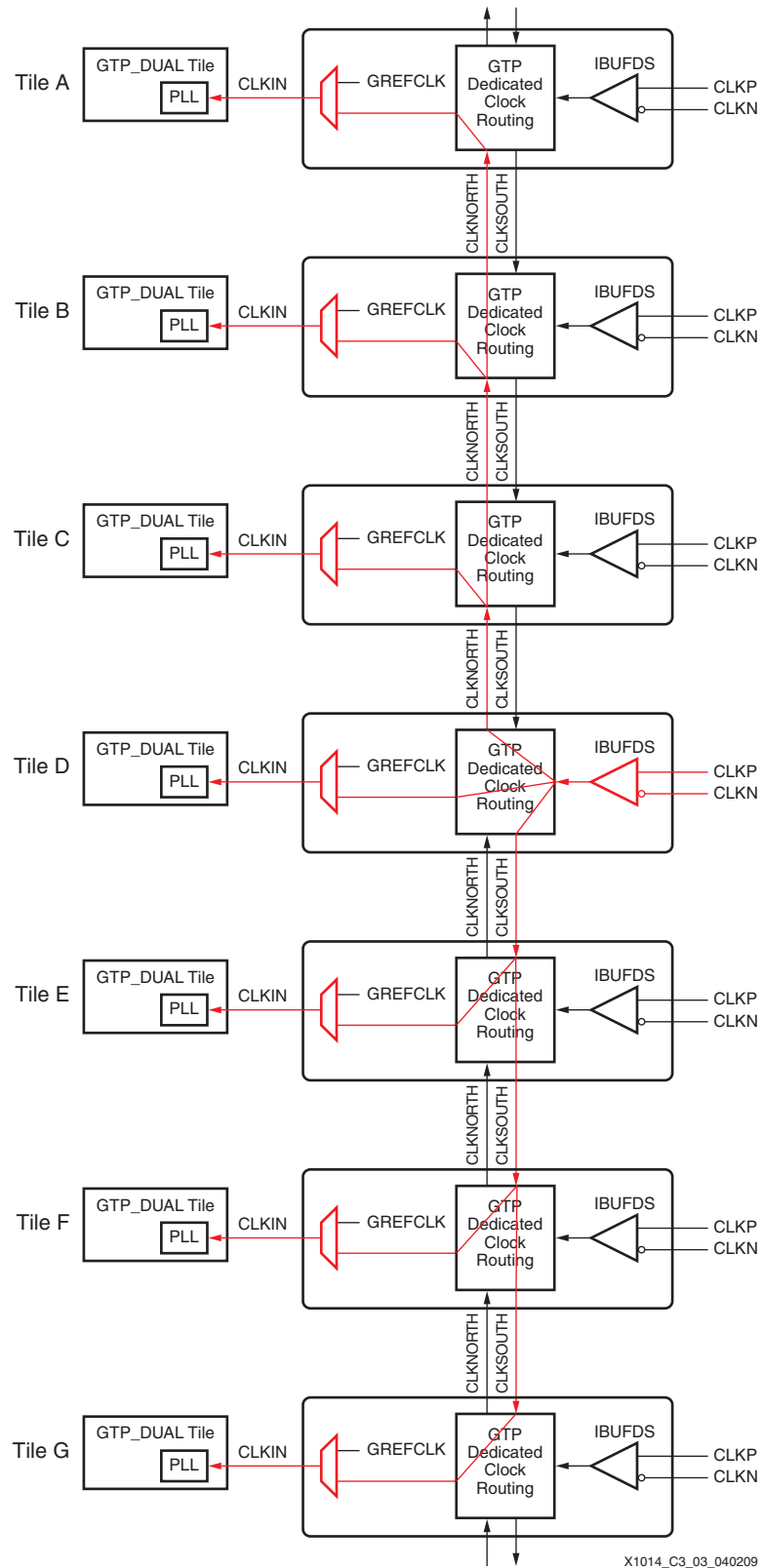


Figure 3-3: Routing a Reference Clock to Seven GTP_DUAL Tiles

GTP Receiver

This section briefly describes how the GTP receiver in the Virtex-5 FPGA is used to receive 3G-SDI, HD-SDI, SD-SDI, and DVB-ASI. Refer to the individual SDI protocol chapters for details of receiving each of these standards with the GTP transceiver.

Receiving 270 Mb/s SD-SDI and DVB-ASI

The GTP receiver implements clock and data recovery for bit rates from 500 Mb/s up to 3.75 Gb/s. It also has a built-in 5X oversampling digital receiver that supports bit rates from 100 Mb/s to 500 Mb/s. However, for several reasons, this built-in 5X oversampling digital receiver is typically not used for SD-SDI and DVB-ASI at 270 Mb/s. Instead, the GTP receiver runs at 11 times the 270 Mb/s bit rate (2.97 Gb/s) and asynchronously oversamples the SD-SDI and DVB-ASI bitstreams. A data recovery unit (DRU), built in the programmable logic of the FPGA, takes the oversampled data from the GTP receiver and recovers the data. The advantages and disadvantages of these two approaches to supporting 270 Mb/s are shown in Table 3-1. The benefits of using the 11X DRU (increased RX jitter tolerance, fewer reference clocks, and complete independence of the two RX units in the same tile) typically outweigh the benefits of using the built-in 5X oversampling digital receiver in SD-SDI and DVB-ASI applications.

Table 3-1: Comparison of Oversampling Techniques Supporting 270 Mb/s Reception

Parameter	Built-in 5X Oversampling Digital RX	Fabric-Based 11X Oversampling DRU
RX Jitter Tolerance	5X oversampling provides marginal tolerance for the long run lengths and pathological patterns of SD-SDI. Xilinx does not recommend using 5X oversampling for SD-SDI applications because of the pathological pattern tolerance.	11X oversampling provides better support for SD-SDI long run lengths and pathological patterns.
Unit Independence	A single enable bit enables the 5X oversampling mode for the entire tile. When this mode is enabled, both receivers in the GTP_DUAL tile run in oversampling mode. Both transmitters also run in oversampling mode and are limited to bit rates between 100 Mb/s and 500 Mb/s.	When using the fabric-based 11X DRU, the two receivers in the GTP_DUAL tile are completely independent. One RX unit can receive SD-SDI or DVB-ASI at 270 Mb/s while the other RX unit in the tile receives HD-SDI or 3G-SDI.
Reference Clock	When using the 5X digital receiver to receive SD-SDI or DVB-ASI, the GTP_DUAL tile's reference clock must be a different frequency than that required for HD-SDI and 3G-SDI. Thus, two reference clock frequencies are required to receive these various bit rates. This further prevents the units in the tile from simultaneous support of different SDI standards.	With the 11X oversampler, the reference clock used for 270 Mb/s SD-SDI and DVB-ASI is the same frequency as that required for HD-SDI and 3G-SDI. Thus, a single reference clock frequency allows both receivers in the GTP_DUAL tile to independently receive 270 Mb/s SD-SDI and DVB-ASI, either HD-SDI bit rate, and either 3G-SDI bit rate.

Table 3-1: Comparison of Oversampling Techniques Supporting 270 Mb/s Reception (Cont'd)

Parameter	Built-in 5X Oversampling Digital RX	Fabric-Based 11X Oversampling DRU
Recovered Clock	The 5X digital receiver recovers a clock.	The 11X oversampling DRU does not recover a clock. The DRU provides a data-ready signal asserted when a 10-bit data word has been recovered. This signal is usually used as a clock enable.
Resource Usage	The 5X digital receiver is built into each GTP transceiver.	The 11X oversampling DRU uses a small amount of FPGA resources.

The biggest downside to using the fabric DRU for SD-SDI is lack of a recovered clock. However, there are techniques available to recover a clock from recovered video. For example, it is possible to use a genlock solution to recover a clock based on the video synchronization signals in the recovered video.

Important RX Ports

Table 3-2 shows the important RX-specific ports of the GTP_DUAL tile used by SDI applications. There are other RX ports used for SDI applications, but they are typically controlled by the `v5gtp_sdi_control` module.

Each RX in the tile has an independent set of the ports shown in Table 3-2. Each signal name contains an italicized *n* that identifies to which RX unit (0 or 1) in the tile the signal belongs. For example, the RXUSRCLK2 port for RX0 is actually called RXUSRCLK20 and for RX1, it is called RXUSRCLK21. The names are also prefixed by a tile number such as TILE0_ to indicate which GTP_DUAL tile the port belongs to when the GTP wrapper file contains multiple GTP_DUAL tiles. For convenience, the ports are described throughout the rest of this application note without the prefixes and suffixes. For example, TILE_x_RXRECCLK_n_OUT is called RXRECCLK.

Table 3-2: Important RX Ports

Port	I/O	Description
TILE _x _RXDATA _n _OUT	Out	This is the receive data bus of the receiver interface to the FPGA logic. For SDI applications, this port can be either 10 or 20 bits wide. However, to minimize global clock resources, a 10-bit RXDATA port is generally preferred.
TILE _x _RXP _n _IN TILE _x _RXN _n _IN	In	This is the differential receive serial data pair.
TILE _x _RXRECCLK _n _OUT	Out	This is the recovered clock from the CDR unit.

Table 3-2: Important RX Ports (Cont'd)

Port	I/O	Description
TILEx_RXUSRCLK2n_OUT	In	This is the input clock used for the RX interface between the FPGA and the GTP transceiver. The frequency of this clock depends on the width of the RXDATA port. When RXDATA is 10 bits wide, this clock is equal in frequency to RXUSRCLK and, in fact, RXUSRCLK and RXUSRCLK2 are driven by exactly the same clock source. When RXDATA is 20 bits wide, this clock must be exactly half the frequency of and phase-aligned with RXUSRCLK.
TILEx_RXUSRCLKn_OUT	In	This is the input clock used for internal RX logic after the RX elastic buffer. This clock must run at 1/10th the bit rate (297 MHz for 3G-SDI and 148.5 MHz for HD-SDI). For SD-SDI, it must run at 297 MHz. This clock is usually derived directly from RXRECCLK, typically by simply buffering RXRECCLK through a BUFG or BUFR.

The RXP/N input signal pair is connected to a current mode logic (CML) receiver that has programmable internal termination. The CML receiver also has built-in AC coupling capacitors that can be bypassed. For SDI applications, these internal AC coupling capacitors are always bypassed (by setting the AC_CAP_DIS attribute of the receiver to TRUE) because they are too small to pass the pathological waveforms present in SDI streams. Instead, external AC coupling capacitors are always used between the external cable equalizer and the receiver input pins.

RX Clocking

For HD-SDI and 3G-SDI, the CDR section recovers data and a clock. The recovered clock is output on the RXRECCLK port. This clock runs at 1/10th the bit rate of the received signal. As shown in Figure 3-4, the RXRECCLK signal is typically connected to a BUFG or BUFR clock buffer. The output of the clock buffer drives the RXUSRCLK and RXUSRCLK2 ports of the RX and also the logic downstream from the RXDATA port. Data is clocked out of the RX on the RXDATA port synchronously with the clock on the RXUSRCLK2 port.

For SD-SDI and DVB-ASI, the CDR section is locked to the reference clock and RXRECCLK is 297 MHz (297/1.001 MHz if the reference clock frequency is 148.5/1.001 MHz or 74.25 MHz/1.001). If a 20-bit RXDATA port is chosen, RXUSRCLK2 must be driven with a clock that is half the frequency of RXUSRCLK. Furthermore, RXUSRCLK and RXUSRCLK2 must be phase aligned, requiring a DCM or PLL to produce these two related clock frequencies. This is shown in Figure 3-5.

Using a 10-bit wide RXDATA port is simpler because RXUSRCLK and RXUSRCLK2 are driven by exactly the same clock signal. As shown in Figure 3-6, it is possible to use a 10-bit RXDATA port with a 20-bit SDI datapath. The differences between Figure 3-4 and Figure 3-6 are the width and the clock frequency of the HD-SDI datapath, which consists primarily of the descrambler and framer.

Note: To minimize the number of clock resources used, all SDI references for the GTP transceiver in the Virtex-5 FPGA use 10-bit RXDATA ports as shown in Figure 3-4 and Figure 3-6. This eliminates the need for a DCM or PLL and requires one global clock instead of two.

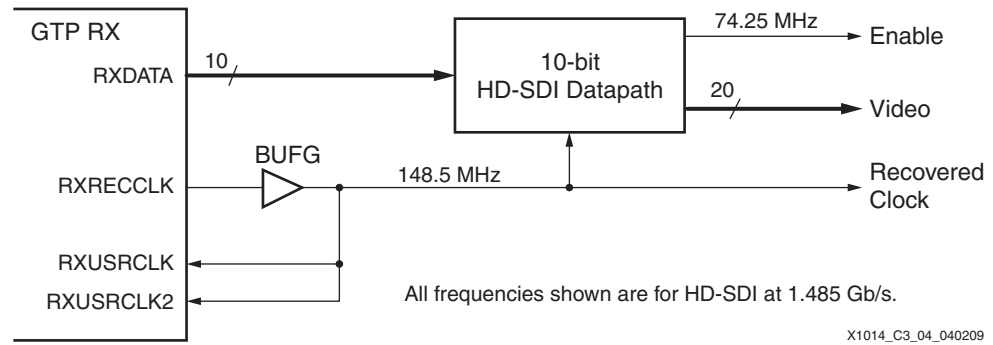


Figure 3-4: HD-SDI RX Clocking with 10-bit RXDATA Port and 10-bit Datapath

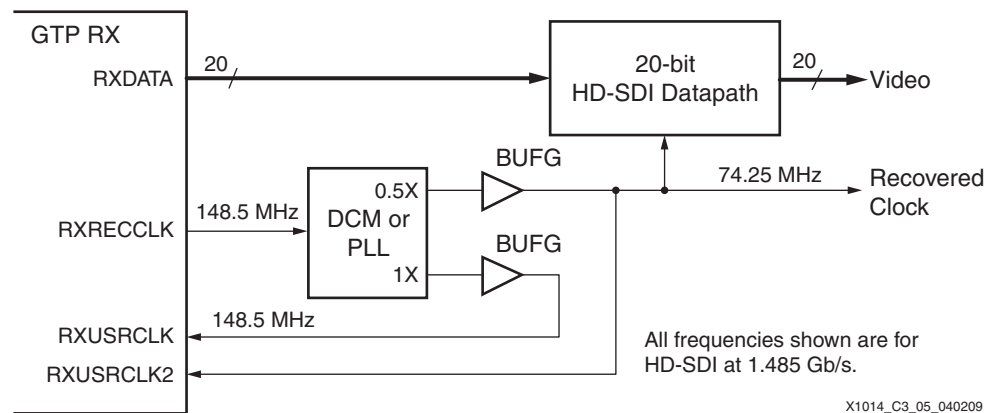


Figure 3-5: HD-SDI RX Clocking with 20-bit RXDATA Port

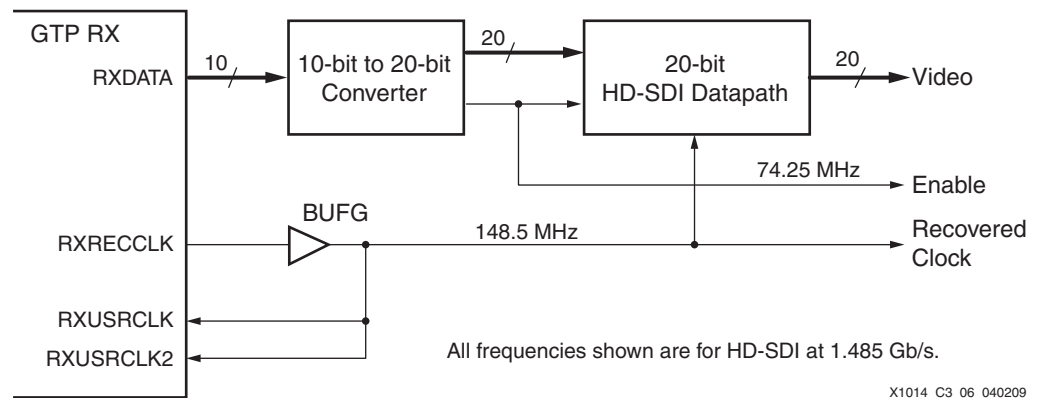


Figure 3-6: HD-SDI RX Clocking with 10-bit RXDATA Port and 20-bit Datapath

Refer to the descriptions of the RX datapaths in the appropriate chapters of this application note for more details about the clocking scheme used for each reference design.

Important RX Attributes

Table 3-3 shows GTP RX attributes that are important for SDI applications. Generally, the settings of these attributes are taken care of through the RocketIO GTP Transceiver Wizard. Those attributes that must be changed dynamically are controlled by the

v5gtp_sdi_control module's DRP controller, although it is possible to override the DRP controller's default values for these attributes using parameters passed to the module. In many cases, there are two of each of these attributes, one for each receiver. In that case, the attribute name shown in the table has "_0" or "_1" appended to the end of the attribute name to indicate to which RX unit it applies. In some cases, however, a single attribute applies to both receivers. Refer to the *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* [Ref 1] for complete details about these attributes.

Table 3-3: Important RX Attributes

Attribute	Description
AC_CAP_DIS	TRUE: Built-in AC coupling capacitors are bypassed. FALSE: Built-in AC coupling capacitors are enabled. For SDI applications, always set this attribute to TRUE to bypass the capacitors.
OVERSAMPLE_MODE	This attribute is always set to FALSE for SDI applications. SD-SDI and DVB-ASI do not use the built-in 5X oversampler, so this attribute is always FALSE.
PLL_RXDIVSEL_OUT	This attribute controls the division factor (1, 2, or 4) applied to the PLL clock to generate the RX serial clock. For SDI applications, this attribute is usually set to 1 for 3G-SDI, SD-SDI, and DVB-ASI, and to 2 for HD-SDI. The v5gtp_sdi_control module dynamically controls this attribute as required to switch the RX between standards.
PMA_CDR_SCAN	This attribute allows direct control of the CDR sampling points. This attribute should always be left at the default value set by the RocketIO GTP Transceiver Wizard.
PMA_RX_CFG	This attribute selects the operating mode of the CDR.
RCV_TERM_GND RCV_TERM_MID RCV_TERM_VTTRX	These attributes control the termination voltage for the receiver. SDI applications bypass the built-in AC coupling capacitors and use external AC coupling capacitors, so all of these attributes must be set to FALSE to set the termination voltage to 2/3 AVTTRX.
RX_BUFFER_USE	TRUE: Use the RX elastic buffer. FALSE: Disable the RX elastic buffer.
TERMINATION_IMP	This attribute selects the termination impedance to be either 50Ω or 75Ω. Although the SDI standards always use 75Ω cables, the RX termination impedance is usually set to 50Ω because an external cable equalizer is used to interface the RX inputs to the cable, and the interface between the cable equalizer and the RX inputs is usually 50Ω.

The PMA_RX_CFG attribute controls the operation of the CDR unit. The value used for this attribute depends upon the operating mode and the setting of the PLL_DIVSEL_OUT attribute. The recommended values for this attribute for the various CDR modes are found in the *Virtex-5 FPGA RocketIO GTP Transceiver User Guide*. The DRP controller in the v5gtp_sdi_control module dynamically controls this attribute to switch the RX between modes. It is possible to override the PMA_RX_CFG values used by the DRP controller for each operating mode through attributes passed to the module.

RX Elastic Buffer

Each GTP receiver has a built-in elastic buffer used to resolve differences between the internal receiver clock domain and the FPGA RX interface clock domain. This buffer can be bypassed to reduce latency, but this requires the application to implement a phase-alignment algorithm.

By disabling the buffer, the latency added by the buffer is eliminated, as is the nondeterministic delay introduced by the RX elastic buffer. However, disabling the RX elastic buffer requires the application to implement a very specific phase-alignment algorithm. This phase-alignment algorithm increases the recovery time of the receiver after operating mode changes. Thus, there is a trade-off between low latency deterministic operation with the buffer bypassed and quicker recovery after mode changes with the buffer enabled.

The *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* [Ref 1] describes in detail the phase-alignment algorithm that must be implemented when the RX elastic buffer is bypassed.

Cable Equalizer

The SDI standards use a single-ended 75Ω cable that can typically be at least 300 meters long for SD-SDI and 100 meters for HD-SDI and 3G-SDI. Such long cable lengths require the use of a cable equalizer. While the GTP receiver has a built-in receiver equalizer, this equalizer is not designed to equalize cables of the lengths used in SDI applications, nor is it designed to handle the pathological patterns present in SDI encoded signals. Furthermore, the GTP receiver requires a differential input, so it cannot be directly connected to the SDI cable.

External SDI-specific cable equalizers are always recommended when implementing SDI applications with the GTP transceiver. These cable equalizers, available from a variety of vendors, are adaptive cable length equalizers that support the cable lengths allowed by the SDI standards. They automatically determine the cable length and adapt the equalization to match. These cable equalizers also change the single-ended cable input to a differential signal that can be connected to the GTP receiver input.

Direct connection of the cable equalizer output to the GTP receiver input is usually not possible. The GTP receiver input is CML, and the SDI cable equalizers typically have LVPECL outputs. AC coupling is usually required between the cable equalizer and the GTP receiver inputs. The value of the AC coupling capacitors should be large enough to support the pathological patterns present in the SDI waveforms. For SDI applications, the AC coupling capacitors are typically in the 1 μF to 10 μF range.

Figure 3-7 shows the connection of an SDI cable equalizer between the BNC connector and the GTP RX inputs. Always refer to the recommendations of the cable equalizer manufacturer for details of the network between the BNC connector and the cable equalizer inputs.

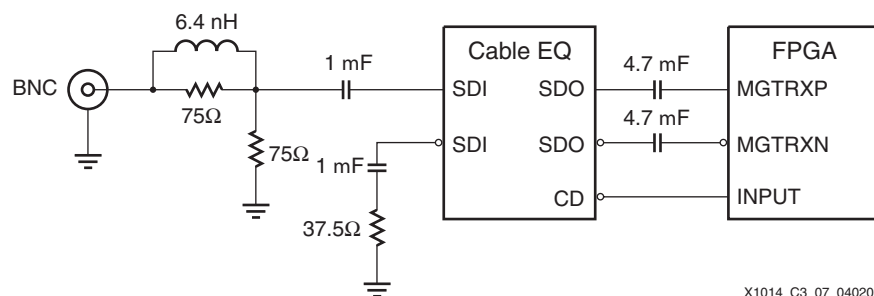


Figure 3-7: Example Cable Equalizer Connections

Figure 3-7 shows a connection from the carrier detect output of the cable equalizer to an FPGA input. This is highly recommended as described next.

For SDI applications, the CDR section is usually configured to use the second-order loop filter to allow reception of the two HD-SDI bit rates (or the two 3G-SDI bit rates) with a single reference clock frequency. When using the second-order loop filter, the CDR section requires a reset whenever the input bitstream stops (called an electrical idle condition). Stoppage of the input bitstream can be caused by the cable being unplugged or the transmitter being turned off. If the CDR section is not reset after the signal is restored, the CDR might not lock properly to the input signal.

The GTP receiver section has a built-in electrical idle detection circuit that is normally used to initiate these electrical idle resets. This electrical idle circuit cannot, however, accurately detect electrical idle conditions when an external cable equalizer is used between the transmitter and the receiver input. An alternative must be provided to reset the CDR section during electrical idle conditions. This alternative is provided by the carrier detect circuit included in the external SDI cable equalizer. It is, therefore, recommended that all SDI applications using the GTP transceiver connect the carrier detect output of each cable equalizer to an FPGA general-purpose input pin so that it can be used to reset the receiver's CDR section during electrical idle conditions.

GTP Transmitter

This section briefly describes how the GTP transmitter of the Virtex-5 FPGA is used to transmit 3G-SDI, HD-SDI, SD-SDI, and DVB-ASI. Refer to the appropriate chapters of this application note for details on transmitting each of these standards with the GTP transceiver.

Transmitting 270 Mb/s SD-SDI and DVB-ASI

The GTP transmitter directly supports transmission of data rates from 500 Mb/s up to 3.75 Gb/s. For data rates slower than 500 Mb/s, an "oversampling" technique is used in which the transmitter is run at some multiple of the bit rate and each bit is sent multiple consecutive times. For SD-SDI and DVB-ASI running at 270 Mb/s, it is best to use 11X oversampling, just as is done for the receiver. This allows the same reference clock frequency to be used to transmit SD-SDI and DVB-ASI as is used for HD-SDI and 3G-SDI. The transmitter runs at a bit rate of 2.97 Gb/s, and each SD-SDI encoded bit is sent 11 consecutive times, resulting in a data rate of 270 Mb/s ($2.97 \text{ Gb/s} \div 11 = 270 \text{ Mb/s}$).

Caution! To transmit at 270 Mb/s, the reference clock to the GTP transmitter must be either 74.25 MHz or 148.5 MHz. It is not possible to use 74.25/1.001 MHz or 148.5/1.001 MHz as the reference clock frequency when transmitting 270 Mb/s because the resulting bit rate would be 270/1.001 Mb/s.

The GTP transmitter has a built-in 5X oversampling mode that can be used to support data rates lower than 500 Mb/s. It is not typically used for SD-SDI and DVB-ASI applications because it requires a different reference clock frequency than that needed for HD-SDI and 3G-SDI. The FPGA resources required to support 11X oversampling with the GTP transmitter are very small, so there is little reason to use the built-in 5X oversampling mode. The reference designs provided in this application note do not use the built-in 5X oversampling TX mode.

Important TX Ports

Table 3-4 describes the important TX-specific ports used by SDI applications. Other TX ports are used for SDI applications, but they are typically controlled by the `v5gtp_sdi_control` module.

Each TX in the tile has an independent set of the ports shown in the table (except for REFCLKOUT) and each signal name has either a 0 or a 1 appended to the end of the name to identify to which TX unit in the tile the signal belongs. The port names are prefixed with the tile number, indicating to which GTP_DUAL tile the port belongs in a GTP wrapper that contains multiple GTP_DUAL tiles. Thus, for example, the TXUSRCLK2 port for TX0 of TILE0 is actually called TILE0_TXUSRCLK20_OUT and for TX1, it is called TILE0_TXUSRCLK21_OUT.

Table 3-4: Important TX Ports

Port	I/O	Description
TILE _x _REFCLKOUT_OUT	Out	This is a copy of the reference clock used to drive TXUSRCLKs when the TX buffer is bypassed. There is only one of these for each GTP_DUAL tile.
TILE _x _TXDATA _n _IN	Out	This is the transmit data bus of the TX interface to the FPGA logic. For SDI applications, this port can be either 10 or 20 bits wide. However, to minimize global clock resources, a 10-bit TXDATA port is generally preferred.
TILE _x _TXOUTCLK _n _OUT	Out	This is the transmitter clock used to drive TXUSRCLKs when the TX buffer is enabled.
TILE _x _TXP _n _OUT TILE _x _TXN _n _OUT	In	This is the differential transmit serial data pair.
TILE _x _TXUSRCLK2 _n _IN	In	This is the input clock used for the TX interface between the FPGA and the GTP transceiver. The frequency of this clock depends on the width of the TXDATA port. When TXDATA is 10 bits wide, this clock is equal in frequency to TXUSRCLK and, in fact, TXUSRCLK and TXUSRCLK2 are driven by exactly the same clock source. When TXDATA is 20 bits wide, this clock must be exactly half the frequency of and phase-aligned with TXUSRCLK.

Table 3-4: Important TX Ports (Cont'd)

Port	I/O	Description
TILEx_TXUSRCLKn_IN	In	This is the input clock used for internal TX logic prior to the TX buffer. This clock must run at 1/10th the bit rate (297 MHz for 3G-SDI and 148.5 MHz for HD-SDI). For SD-SDI, it must run at 297 MHz. This clock is usually derived directly from TXOUTCLK or REFCLKOUT (depending on whether the TX buffer is enabled or not), typically by simply buffering TXOUTCLK or REFCLKOUT through a BUFG or BUFR.

The serial output of the GTP transmitter is a CML differential pair. The CML transmitter has built-in termination that is calibrated to an external precision resistor. Typically, this is set to 50Ω, even though the SDI protocols use 75Ω cables. An external cable driver is required to convert the 50Ω differential CML GTP TX output to a 75Ω single-ended signal suitable for driving SDI cables.

TX Clocking

Like the RX section, each TX unit has two user clock inputs called TXUSRCLK and TXUSRCLK2. These are used to drive different portions of the interface between the GTP TX and the FPGA logic. The source of these clocks usually comes from either TXOUTCLK or REFCLKOUT, depending on whether the TX buffer is enabled or not. If the TX buffer is enabled, then TXOUTCLK is used. If the TX buffer is bypassed, REFCLKOUT must be used.

When using TXOUTCLK to drive the TXUSRCLKs, it is always the correct frequency to drive TXUSRCLK. When the TXDATA port is 10 bits wide, it is also the correct frequency to drive TXUSRCLK2. Typically, TXOUTCLK is buffered by a BUFG or BUFR clock buffer and is then used to drive the TXUSRCLKs and the FPGA resources used to implement the SDI datapath as shown in [Figure 3-8](#).

When using TXOUTCLK with a 20-bit TXDATA port, TXUSRCLK2 must be half the frequency of TXOUTCLK and must be phase-aligned with TXUSRCLK. These requirements are satisfied with a DCM or PLL as shown in [Figure 3-9](#). Because a 20-bit TXDATA port requires more global clocking resources, 10-bit TXDATA ports are typically used. It is possible to use a 20-bit HD-SDI TX datapath with a 10-bit TXDATA port as shown in [Figure 3-10](#).

When the TX buffer is bypassed to minimize latency, TXOUTCLK cannot be used as the clock source for the TXUSRCLKs. Instead, REFCLKOUT must be used. REFCLKOUT is always the same frequency as the reference clock selected to drive the PLL of the GTP_DUAL tile. It might or might not be the correct frequency to drive the TXUSRCLKs, and a DCM or PLL might be required to multiply it up to the correct frequency, even if a 10-bit TXDATA port is used. If the reference clock frequency is 148.5 MHz (or 148.5/1.001 MHz), the DCM or PLL is not required with a 10-bit TXDATA port. However, if the reference clock frequency is 74.25 MHz, then a DCM or PLL is required to double REFCLKOUT to 148.5 MHz to drive the TXUSRCLKs, as shown in [Figure 3-11](#) and [Figure 3-12](#). 20-bit TXDATA ports and datapaths can be used with the TX buffer bypassed, but only the 10-bit paths are shown here. To minimize clock resources required for SDI applications, all GTP transceiver SDI reference designs for Virtex-5 FPGAs in this application note use 10-bit TXDATA ports.

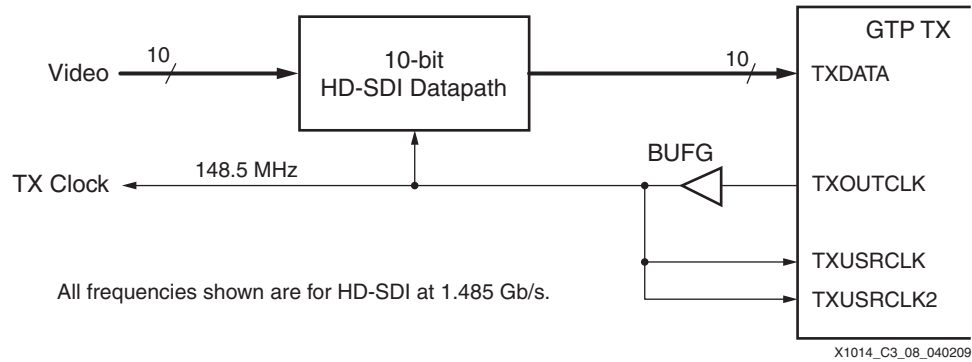


Figure 3-8: HD-SDI TX Clocking, TX Buffer Enabled, 10-Bit TXDATA Port, and 10-Bit Datapath

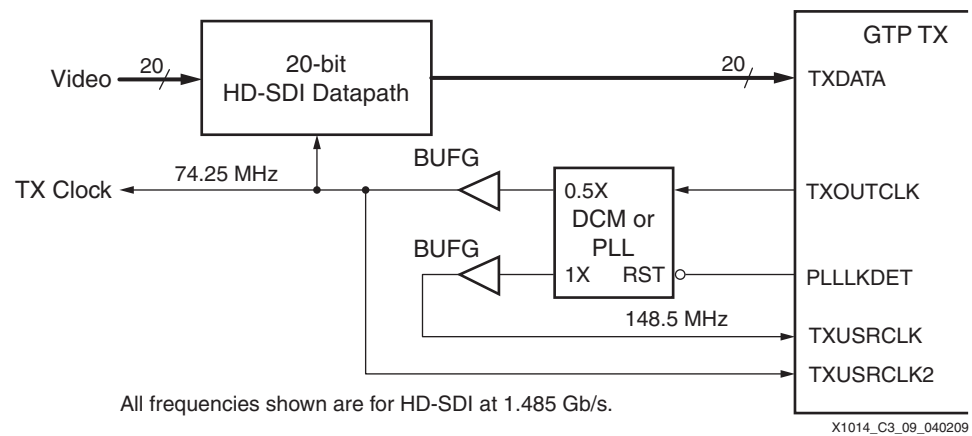


Figure 3-9: HD-SDI TX Clocking, TX Buffer Enabled, 20-Bit TXDATA Port

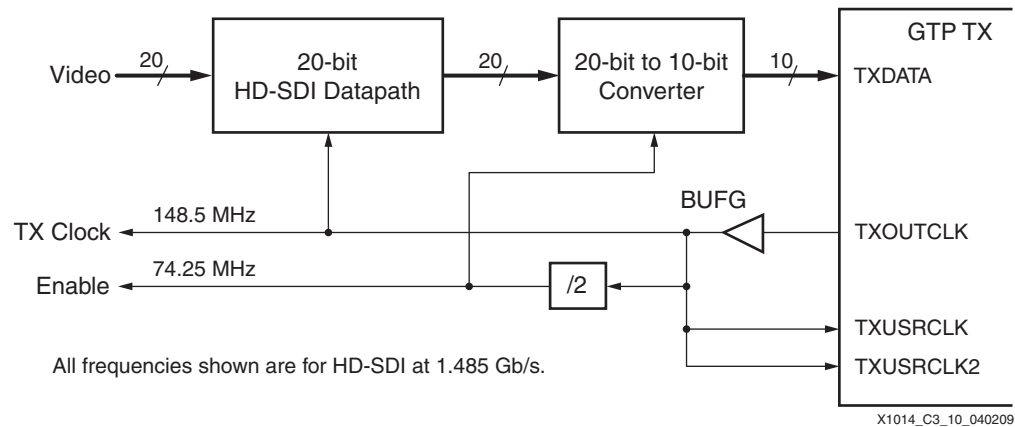


Figure 3-10: HD-SDI TX Clocking, TX Buffer Enabled, 10-Bit TXDATA Port, and 20-Bit Datapath

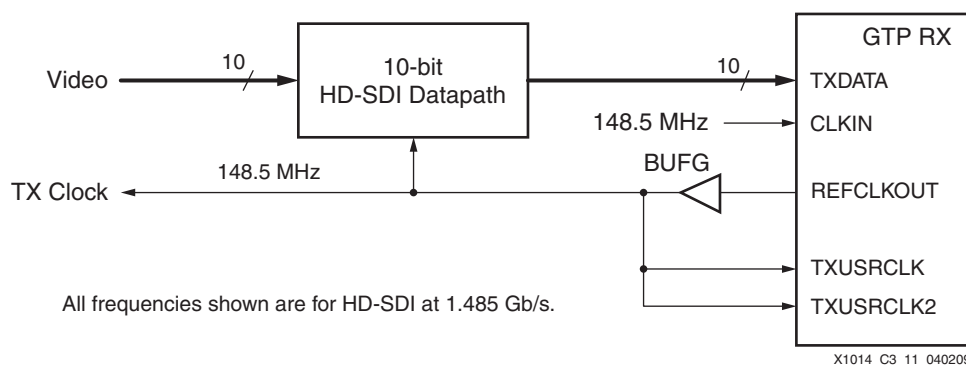


Figure 3-11: HD-SDI TX Clocking, TX Buffer Bypassed, 148.5 MHz Reference Clock

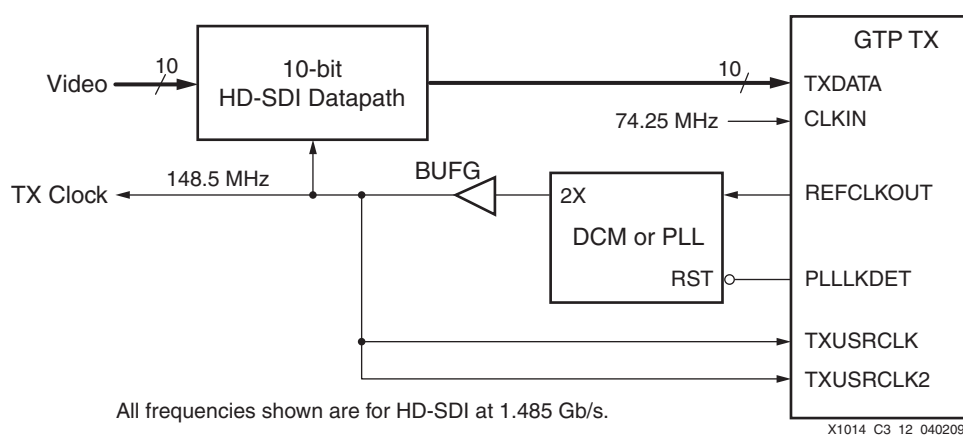


Figure 3-12: HD-SDI TX Clocking, TX Buffer Bypassed, 74.25 MHz Reference Clock

It is also possible to drive the TXUSRCLKs from sources other than TXOUTCLK or REFCLKOUT. For example, in a pass-through HD-SDI application, it can be desirable to drive the TXUSRCLKs with the recovered clock from the receiver. This is possible with the GTP transceiver, but it is recommended that the TX buffer be used for such applications. It is essential that the transmitter's reference clock and the TXUSRCLKs are frequency locked, otherwise the TX buffer can quickly overrun or underrun. Thus, the reference clock of the tile containing the transmitter must be derived from the recovered clock of the receiver. This is typically done by sending the receiver's recovered clock through an external low-bandwidth PLL for jitter reduction and then connecting the output of the PLL to the reference clock input of the tile containing the transmitter, as shown in Figure 3-13. All the variations of 10- or 20-bit TXDATA ports and datapaths are supported, but only the 10-bit case is shown.

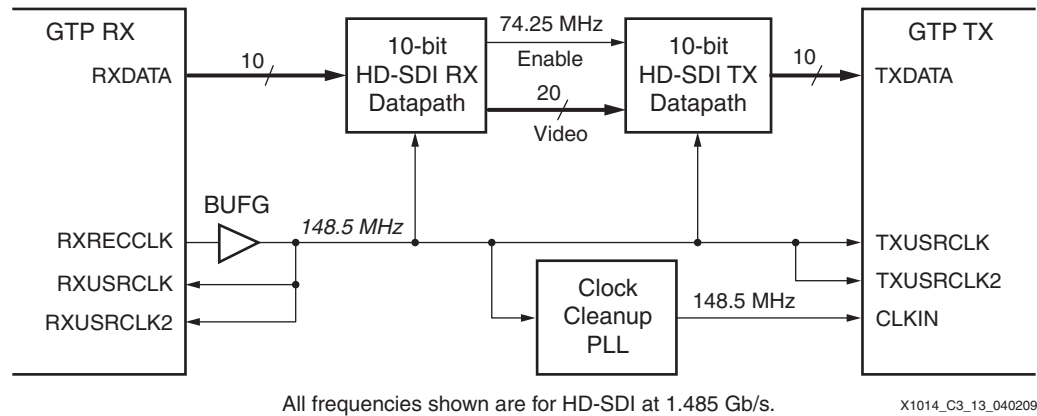


Figure 3-13: HD-SDI TX Clocked by RXRECCLK

Important TX Attributes

Table 3-5 shows GTP TX attributes that are important for SDI applications. Generally, the settings of these attributes are taken care of through the RocketIO GTP Transceiver Wizard. Those attributes that must be changed dynamically are controlled by the DRP controller of the `v5gtp_sdi_control` module. However, it is possible to override the DRP controller's default values for these attributes using parameters passed to the module. In many cases, there are two of each of these attributes, one for each transmitter. In that case, the attribute name shown in the table has “_0” or “_1” appended to the end of the attribute name to indicate to which TX unit it applies. But in some cases, a single attribute applies to both transmitters. Refer to the *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* [Ref 1] for complete details about these attributes.

Table 3-5: Important TX Attributes

Attribute	Description
OVERSAMPLE_MODE	This is always set to FALSE for SDI applications. SD-SDI and DVB-ASI do not use the built-in 5X oversampler, so this attribute is always FALSE.
PLL_TXDIVSEL_COMM_OUT	This attribute controls the division factor applied to the PLL output clock by the divider common to both TX units in the GTP_DUAL tile. For SDI applications, this divider is usually set to divide by 1.
PLL_TXDIVSEL_OUT	This attribute controls the division factor applied to the PLL output clock by the dividers that are unique to each of the TX units in the tile (there are two of these attributes for each tile). For 3G-SDI, SD-SDI, and DVB-ASI, this attribute is set to 1. For HD-SDI, this attribute is set to 2.
TX_BUFFER_USE	TRUE: Use the TX buffer. FALSE: Disable the TX buffer.

TX Buffer

Each GTP transmitter has a built-in buffer used to resolve differences between the internal transmitter clock domain and the FPGA TX interface clock domain. This buffer can be

bypassed to reduce latency, but requires the application to implement a phase-alignment algorithm. The TX buffer is quite shallow, only four words deep.

By disabling the buffer, the latency added by the buffer is eliminated, as is the nondeterministic delay introduced by the TX buffer. However, disabling the TX buffer requires the application to implement a very specific phase-alignment algorithm. This phase-alignment algorithm increases the recovery time of the transmitter after operating mode changes. Thus, there is a trade-off between low-latency deterministic operation with the buffer bypassed and quicker recovery after mode changes with the buffer enabled. The *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* [Ref 1] describes in detail the phase-alignment algorithm that must be implemented when the TX buffer is bypassed.

If the TX buffer is bypassed, the application must use REFCLKOUT as the source of the TXUSRCLKs instead of TXOUTCLK. This might require the use of the DCM or PLL to double REFCLKOUT before it can drive the TXUSRCLKs, if the reference clock is 74.25 MHz. If the TX buffer is enabled, TXOUTCLK can be used to drive the TXUSRCLKs, and it is always the correct frequency for connection to TXUSRCLK.

Cable Driver

An external cable driver designed specifically for SDI applications should be used to interface the GTP transmitter's differential CML output to a single-ended 75Ω signal suitable for driving the cable. Such cable drivers usually have LVPECL interfaces and cannot be directly coupled to the CML output of the GTP TX. AC coupling is typically used between the GTP TX output and the cable driver. The AC coupling capacitors are typically in the 1 μF to 10 μF range in order to pass the SDI pathological patterns.

SD-SDI has different slew rate requirements than HD-SDI and 3G-SDI. SDI cable drivers designed to support both SD-SDI and HD-SDI, or SD-SDI, HD-SDI, and 3G-SDI usually have a slew rate control input. This input must be controlled appropriately based on the SDI mode selected. This is usually done by connecting the slew rate control pin of the cable driver to a general-purpose output of the FPGA. The `v5gtp_sdi_control` module has outputs designed specifically for controlling the slew rate control pins of SDI cable drivers.

Figure 3-14 shows a typical SDI cable driver connected to the outputs of the GTP transmitter in the Virtex-5 FPGA. Refer to the cable driver manufacturer's recommendations for how to properly interface the cable driver to the BNC cable connector.

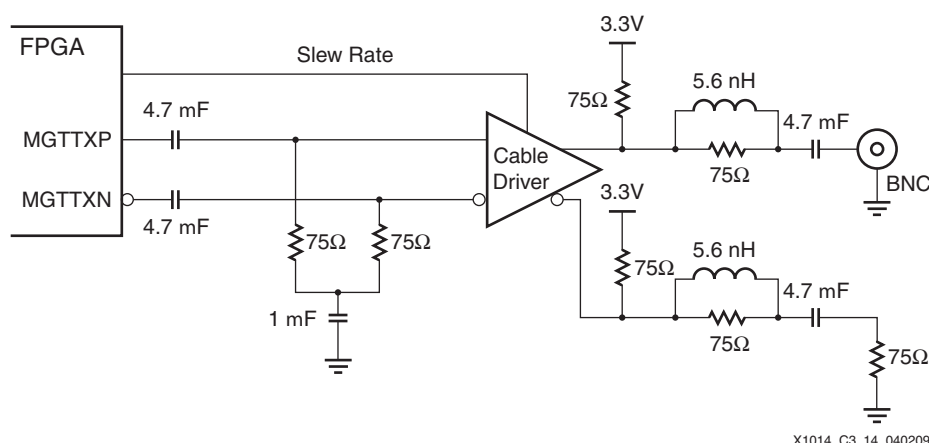


Figure 3-14: Example Cable Driver Connections

Control Module

When using the GTP transceiver of the Virtex-5 FPGA to implement SMPTE SDI standards, a number of functions must be implemented in the programmable logic of the FPGA to support the operation of the GTP transceiver. These functions are not the datapath logic for the SDI transmitters or receivers. Rather, they are support functions such as reset circuits that are required for proper operation of the GTP transceiver. Several of these functions have been gathered together into a module called `v5gtp_sdi_control`, creating a more modular and manageable design. This section describes the `v5gtp_sdi_control` module and how it interacts with the GTP transceiver and the SDI datapath logic.

The `v5gtp_sdi_control` module works with the standard GTP transceiver design flow. The RocketIO GTP Transceiver Wizard creates a wrapper module containing one or more GTP_DUAL tiles. A `v5gtp_sdi_control` module is instantiated for each GTP_DUAL tile used for SDI protocols, providing a number of interrelated functions. Incorporating all these functions into one module hides much of the complexity of the relationships between them. The functions are:

- Reset logic for the GTP transceiver and SDI datapath
- DRP controller to control the GTP transceiver operating modes (SD-SDI, HD-SDI, and 3G-SDI), and reference clock selection and routing
- RX bit rate detection

[Figure 3-15](#) is a diagram of how the `v5gtp_sdi_control` module and the GTP wrapper file interconnect. One `v5gtp_sdi_control` module is required for each GTP_DUAL tile used to implement SDI protocols, regardless of how many of the RX and TX units are actually used in that tile.

The GTP wrapper module is created by the RocketIO GTP Transceiver Wizard. Depending on the parameters given to the Wizard, the wrapper module can contain more than one GTP_DUAL tile. In that case, multiple `v5gtp_sdi_control` modules are connected to the same GTP wrapper module, one for each GTP_DUAL tile that is used for SDI protocols.

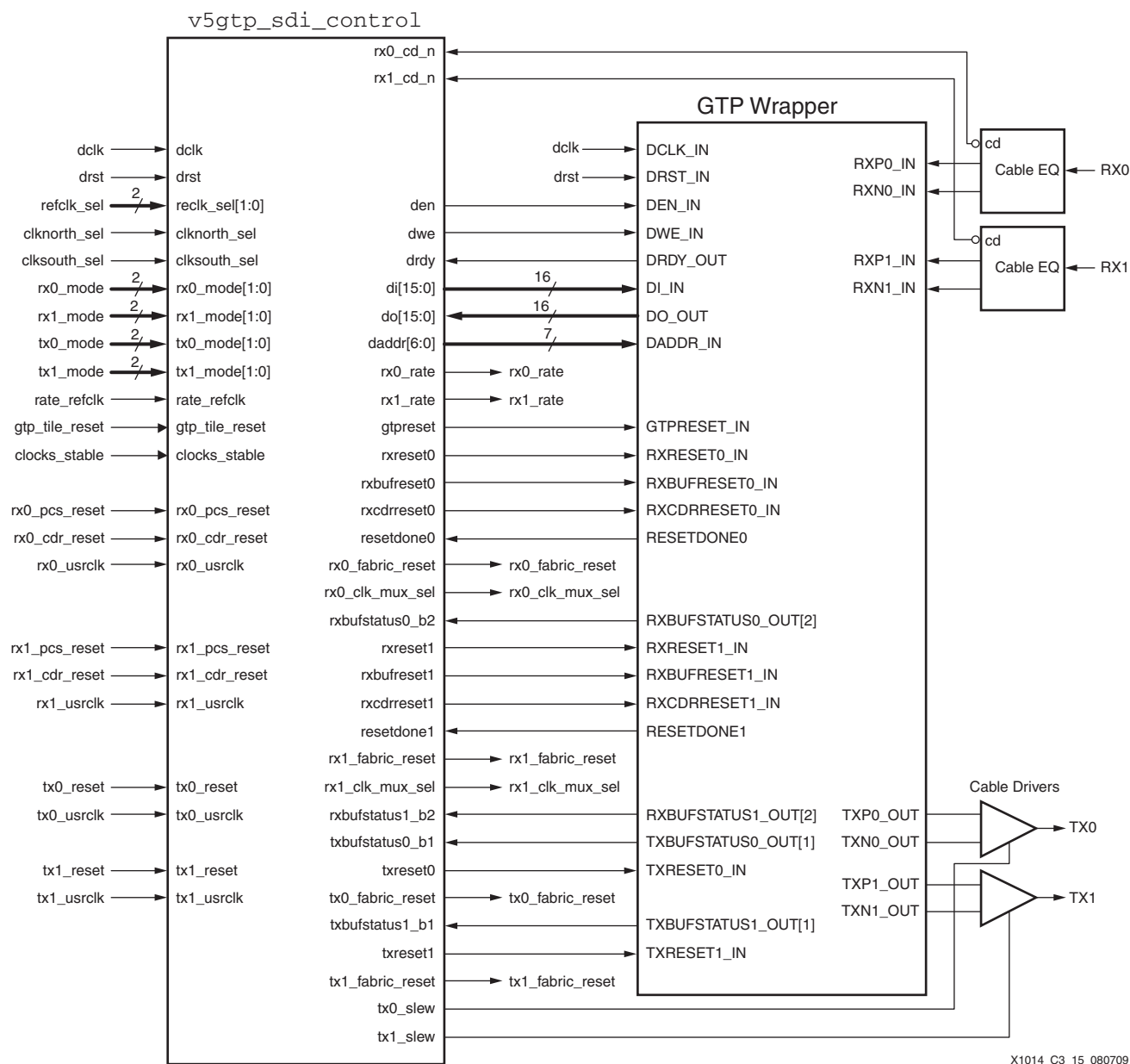


Figure 3-15: Control Module and GTP Wrapper Interconnections

Reset Logic

The GTP transceiver has different reset inputs that must be asserted under various conditions. This section describes the reset signals controlled by the `v5gtp_sdi_control` module.

GTPRESET

GTPRESET is the master reset for the entire tile, and it is the only way to reset the clock multiplication PLL. This PLL is common to all units in the GTP_DUAL tile, and it must be

reset after the reference clock becomes stable. If the reference clock is stopped and then restarted, the PLL should also be reset.

The `v5gtp_sdi_control` module has a `gtpreset` output that drives the `GTPRESET_IN` input of the GTP wrapper to cause a GTPRESET of the GTP_DUAL tile. The module asserts `gtpreset` under three conditions:

- Immediately after the FPGA emerges from configuration.
- When the module's `gtp_tile_reset` input is High.
- When the module's `clocks_stable` input is Low.

The `gtp_tile_reset` and `clocks_stable` inputs provide two different ways to reset the PLL and the rest of the GTP_DUAL tile after the reference clock is started or changes dramatically. Most applications only use one of these inputs to the `v5sdi_gtp_control` module, not both, depending on application-specific requirements.

The `gtp_tile_reset` input generates a long-duration `gtpreset` pulse to the GTP_DUAL tile. The length of this pulse is controlled by the frequency of the `dclk` input (DRP clock) and the parameter/generic `GTP_RESET_CNTR_SIZE`. Asynchronously, on the rising edge of `gtp_tile_reset`, a counter is reset and the `gtpreset` output goes High, resetting the GTP_DUAL tile. As long as the `gtp_tile_reset` input is High, the `gtpreset` output remains High and the counter stays at zero. After `gtp_tile_reset` goes Low, the counter begins counting. While the counter is counting, `gtpreset` remains High. After the counter reaches its terminal count, `gtpreset` goes Low. The terminal count occurs as soon as the most significant bit of the counter becomes 1. Equation 3-1 gives the duration of `gtpreset`.

$$gtpreset\ duration = gtp_tile_reset\ duration + \frac{2^{GTP_RESET_CNTR_SIZE - 1}}{dclk\ frequency} \quad \text{Equation 3-1}$$

Figure 3-16 shows the logic used to generate the GTPRESET. In all figures in this chapter, flip-flops use the same nomenclature as that used by the Xilinx® libraries guide. Asynchronous preset and clear inputs are labeled P and C, respectively. Synchronous set and reset are labeled S and R, respectively. The same nomenclature is also used on the counter in Figure 3-16, where the `gtp_tile_reset` input asynchronously clears the counter and presets the flip-flop.

Note: The libraries guide is automatically installed on the user's PC when the ISE® tools are installed.

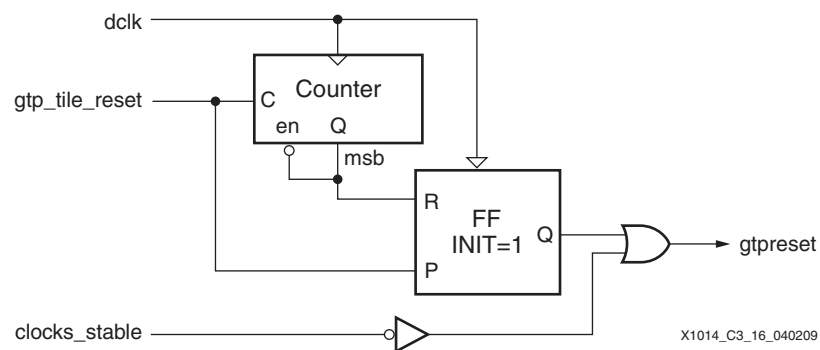


Figure 3-16: **gtpreset Logic**

The `gtp_tile_reset` input and its associated counter are designed for applications that have no inherent method of determining when the reference clock supplied to the GTP_DUAL tile is stable, such as a locked signal from a PLL. In such applications, the `gtpreset` duration

must be set to be longer than the worst-case ramp-up time of the GTP transceiver reference clock.

The `gtpreset` output is asserted immediately after the FPGA comes out of configuration because the INIT value for the flip-flop is 1. It remains asserted for the duration of the counter period described in Equation 3-1 even if `gtp_tile_reset` is not asserted after FPGA configuration. This provides an automatic GTPRESET of the GTP_DUAL tile upon completion of FPGA configuration to allow the reference clock to stabilize.

In some applications, the reference clock source can also supply a signal that indicates when the reference clock is stable. This signal can be used to drive the `clocks_stable` input of the `v5gtp_sdi_control` module. When this signal is Low, the `gtpreset` output is asserted High and remains High until `clocks_stable` goes High. The timer circuit associated with the `gtp_tile_reset` signal is completely bypassed. Thus, the duration of the `gtpreset` signal is equal to the duration that `clocks_stable` is Low.

If the `clocks_stable` input is used and the `gtp_tile_reset` input is not, the counter circuit is still used to assert the `gtpreset` output for the duration of the counter period after FPGA configuration. If the `clock_stable` signal also safely controls `gtpreset` after FPGA configuration, the counter size can be set to something small. However, the counter size must never be set to zero.

If the `clocks_stable` input is not used, it must be tied High. If the `gtp_tile_reset` input is not used, it must be tied Low. Applications must use one or the other of these signals to reset the GTP_DUAL tile if the reference clock stops or changes frequency, other than on a clean transition between the two SDI reference clock frequencies.

Transmitter Resets

The GTP_DUAL tile has a reset input for each transmitter unit. `TXRESET0` resets the PCS section of TX0. `TXRESET1` resets the PCS section of TX1. The `v5gtp_sdi_control` module controls these two GTP transmitter reset signals with its `txreset0` and `txreset1` outputs.

According to the *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* [Ref 1], there are two conditions when the `TXRESET` signals should be asserted:

- When there are disruptions in the `TXUSRCLK` clocks
- When the TX buffer (if used) underflows or overflows

The `v5gtp_sdi_control` module allows the `TXRESET` signals to be asserted under both conditions. It also automatically asserts `TXRESET` for a short period of time if the transmitter mode is changed (between SD-SDI, HD-SDI, or 3G-SDI modes). The TX reset logic is shown in Figure 3-17.

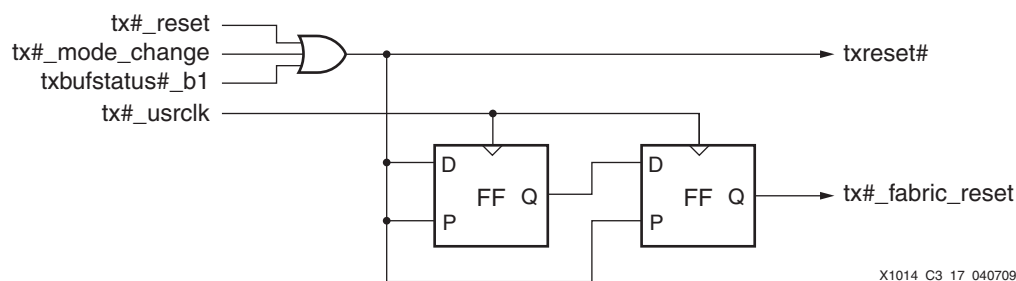


Figure 3-17: TX Reset Logic

To handle TXUSRCLK disruptions, the `v5gtp_sdi_control` module has inputs called `tx0_reset` and `tx1_reset`. These inputs control the `txreset0` and `txreset1` outputs, respectively. In this discussion, the signals for either transmitter are generically called `tx#_reset` for the input signal and `txreset#` for the output signal, where `#` can be 0 or 1. Thus, the `tx0_reset` and `txreset0` signals are used for TX0 of the GTP_DUAL tile and `tx1_reset` and `txreset1` are used for TX1 of the GTP_DUAL tile.

As long as the `tx#_reset` input is High, the `txreset#` output is asserted. If there is any possibility that the TXUSRCLKs might be disrupted, the application must control the `tx#_reset` appropriately to issue a `txreset#` after the TXUSRCLKs have been restored. Disruption of the TXUSRCLKs can occur for application-specific reasons. For example, if a DCM or PLL is used to create the TXUSRCLKs, the `tx#_reset` input of the `v5gtp_sdi_control` module can be driven by the inverted LOCKED output of the DCM or PLL.

The `txreset#` output is also asserted if the TX buffer overflows, as indicated by bit 1 of the `TXBUFSTATUS#_OUT` bit of the GTP_DUAL tile (connected to the `txbufstatus#_b1` input of the `v5gtp_sdi_control` module). It is also asserted when the transmitter operating mode is changed between SDI protocols (3G-SDI, HD-SDI, or SD-SDI). The DRP controller, located inside the `v5gtp_sdi_control` module, switches the modes of the transmitter in response to mode select inputs and asserts the `tx#_mode_change` signal when the transmitter's mode has been changed. Thus, the `tx#_mode_change` signal is an internal signal, not an input to the module.

The transmit reset logic also synchronizes the `txreset#` signal to `tx#_usrclk` to produce the `tx#_fabric_reset` output. This output is asserted asynchronously, as soon as `txreset#` is asserted, but is negated synchronously with the rising edge of `tx#_usrclk`. The `tx#_fabric_reset` signal can be used to reset the FPGA logic of the SDI transmitter, if desired. Because this output is negated synchronously with the rising edge of the `tx#_usrclk`, with proper timing constraints, this signal can be used with both synchronous and asynchronous resets of the various SDI datapath modules.

Receiver Resets

The GTP receiver has several different resets that must be asserted under different conditions. Each GTP receiver in the tile has its own set of reset inputs. The three reset signals controlled by the `v5gtp_sdi_control` module are:

- **RXCDRRESET:** This is also known as the CDR reset. It resets both the CDR and the PCS sections of the receiver. Thus, it resets the entire receiver, except for the clock multiplication PLL common to all units in the tile.
- **RXRESET:** This input resets the PCS section of the receiver and does not affect the CDR section.
- **RXBUFRESET:** This input resets just the RX buffer. The buffer is also reset by **RXRESET** and **RXCDRRESET**; however, if only the buffer needs to be reset, **RXBUFRESET** can be used.

The *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* [Ref 1] states that a CDR reset should be asserted if there is an interruption in the incoming bitstream. The CDR reset must be asserted after the input signal is restored. In SDI, the input signal is interrupted if the transmitter is halted for some reason or if the cable is disconnected.

The *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* states that **RXRESET** should be used when there is an interruption in the RXUSRCLKs. After the RXUSRCLKs have been restored and are stable, **RXRESET** can be safely negated.

RXBUFRESET can be used to reset just the RX buffer, if it is used. The RX buffer should be reset if there is a buffer underflow or overflow. When bit 2 of the RXBUFSTATUS output of the receiver is 1, an RX buffer error has occurred and RXBUFRESET is used to reset the buffer.

Figure 3-18 shows the logic in the `v5gtp_sdi_control` module that controls the various RX resets. There are two sets of this logic, one for each receiver in the GTP_DUAL tile.

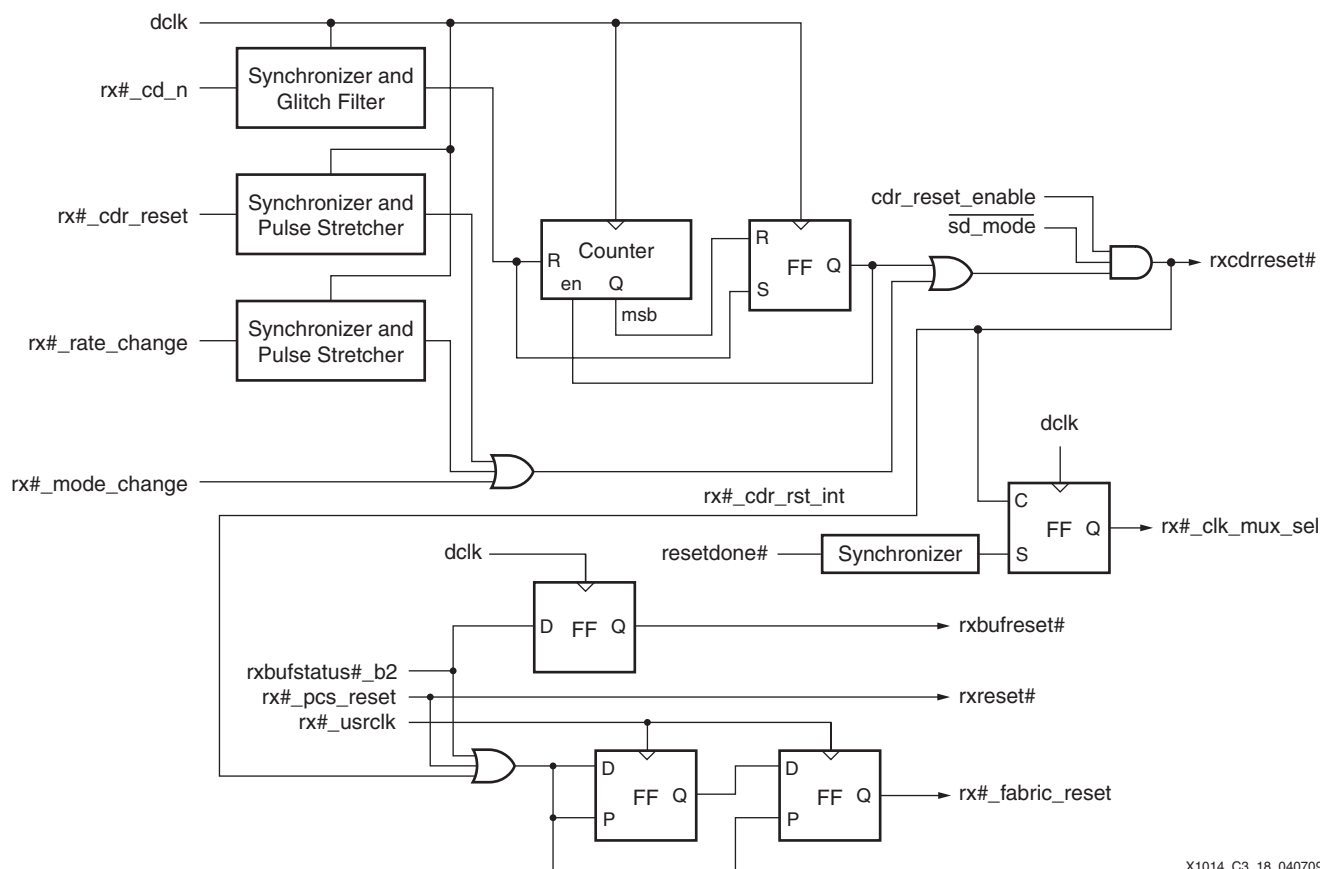


Figure 3-18: RX Reset Logic

The `rxcdrreset#` output is asserted to issue a CDR reset to the GTP RX. This output can be triggered by four different signals:

- A High level on the active-Low carrier detect signal from the external cable equalizer, indicating a loss of input signal either because the cable has been disconnected or the transmitter has been switched off. It is asserted Low when a valid signal is detected and deasserted High when a valid signal is not detected.
- Assertion of the rx#_cdr_reset input to the v5gtp_sdi_control module. This can be used by the application if it needs to reset the entire receiver.
- Assertion of the rx#_rate_change signal from the rate detector. The rate detector asserts this signal if the input bit rate changes. This signal is also asserted if the recovered clock from the receiver is out of range of a valid SDI bit rate, indicating that the CDR section is not locked to the input bitstream. This is an internal signal in the module.

- Assertion of the rx#_mode_change signal from the DRP controller. This signal is asserted when the mode of the receiver is changed between HD-SDI, SD-SDI, and 3G-SDI. This is an internal signal in the module.

The CDR reset is disabled during GTPRESETs because CDR resets are not allowed during GTPRESETs. CDR resets are also disabled when the receiver is in SD-SDI mode because CDR resets interfere with the operation of the receiver when it is locked to the reference clock, as it is during SD-SDI mode.

The carrier detect signal from the cable equalizer tends to be noisy and is used to detect events that are very long in duration. This signal is first processed by a glitch filter and synchronized to the dclk before being applied to the CDR reset logic. Because this signal also tends to bounce for a long period of time on cable disconnects and reconnects, it is used to trigger a long-duration CDR reset pulse on the order of 60 μ s. The duration of this reset signal is determined by the width of the counter, which is set by the CBL_DISCONNECT_RST_CNTR_SIZE parameter/generic and the frequency of dclk, as shown in Equation 3-2.

$$CD \text{ reset duration} = \frac{2^{CBL_DISCONNECT_RESET_CNTR_SIZE - 1}}{dclk \text{ frequency}} \quad \text{Equation 3-2}$$

The rx#_cdr_reset input and rx#_rate_change internal signals are asynchronous to dclk and can have pulses that are shorter in duration than the dclk period. Thus, both signals are synchronized to dclk, and short pulses are detected and stretched out to be at least one dclk cycle long.

Assertion of any of these four signals asserts the rxcdrreset# output. It also causes the rx#_clk_mux_sel signal to go Low. This signal is used to control a global clock multiplexer used to drive the RXUSRCLKs. During CDR resets, the recovered clock from the GTP RX might stop. In many applications, it is desirable to continue to clock the SDI datapath logic, even when no valid data is being received. The global clock multiplexer can select a free-running reference clock to drive the RXUSRCLKs in the absence of a recovered clock during CDR resets. Upon completion of the receiver's internal CDR reset cycle, as indicated by the GTP transceiver's RESETDONE output going High, the rx#_clk_mux_sel goes High again, causing the global clock multiplexer to switch back to RXRECCLK.

Note: Each GTP_DUAL tile has two RESETDONE outputs, one for each RX/TX pair. RESETDONE does not go High until both the RX and the TX in the pair have finished their reset cycles. Furthermore, the reset cycle cannot be completed if the reference clock and the user clocks into the GTP transceiver are halted. Thus, in cases where only the RX of an RX/TX pair is used, it is essential that the TXUSRCLK and TXUSRCLK2 of the TX unit be driven by live clocks and not tied to ground. Otherwise, the RESETDONE signal never goes High and the rx#_clk_mux_sel never selects RXRECCLK as the source of the RX user clock.

Asserting RXCDRRESET to the GTP RX also resets the PCS section and the RX buffer. Thus, it is not necessary to assert either RXRESET or RXBUFRESET when a RXCDRRESET occurs.

The rxreset# output, which drives the GTP receiver's RXRESET input, is only asserted when the rx#_pcs_reset input goes High. The rx#_pcs_reset can be asserted by the application to reset the PCS section of the GTP RX without resetting the CDR section. The only reason for doing this is to reset the PCS section after a disruption of the RXUSRCLKs. This primarily occurs if the RXUSRCLKs are being driven by a DCM or PLL. In this case, the LOCKED output of the DCM or PLL can be inverted and connected to the rx#_pcs_reset input of the v5gtp_sdi_control module to reset the PCS section of the receiver.

The `rxbufreset_#` output drives the `RXBUFRESET` input of the GTP RX to reset the RX buffer in case of an RX buffer error. It is driven by bit 2 of the `RXBUFSTATUS` output of the RX. This signal goes through a flip-flop to make timing easier to achieve.

In the event of a CDR reset, or an `RXRESET` or `RXBUFRESET` assertion, the `v5gtp_sdi_control` module also asserts an output called `rx#_fabric_reset`. This signal is asynchronously asserted as soon as any of the reset conditions become asserted. It is negated synchronously with the rising edge of `rx#_usrclk`. With proper timing constraints, this signal can be used with both synchronous and asynchronous resets of SDI datapath modules.

DRP Control

The `v5gtp_sdi_control` module contains a DRP controller that dynamically controls the operating mode of the GTP transceiver. The DRP controller in the module provides these capabilities:

- Independently sets the operating mode of each RX and TX. Each RX and TX can be set to SD-SDI, HD-SDI, or 3G-SDI mode. However, due to the shared clock multiplication PLL, there are limitations on using both transmitters in the same GTP_DUAL tile.
- Selects the reference clock input to the GTP_DUAL tile's clock multiplication PLL. This allows the transmitters in the tile to be switched between the two HD-SDI bit rates (and between the two 3G-SDI bit rates).
- Selects the source of the `CLKOUTNORTH` and `CLKOUTSOUTH` reference clock outputs of the GTP_DUAL tile to the neighboring tiles.

Mode Selection

Each RX and TX unit can be independently set to 3G-SDI, HD-SDI, or SD-SDI mode. For each RX and TX, there is a 2-bit input port to the `v5gtp_sdi_control` module that selects the SDI mode for the unit, as shown in [Table 3-6](#).

Table 3-6: Mode Selection

Mode Select	Mode
00	HD-SDI
01	SD-SDI
10	3G-SDI
11	Invalid

Whenever the mode of a transmitter is changed, the transmitter PCS section is reset by assertion of the `TXRESET` signal. Whenever the mode of a receiver is changed, the entire receiver is reset by assertion of the `RXCDRRESET` signal.

When the FPGA comes out of configuration, the DRP controller examines the mode select signals and sets each unit to the mode specified. After that, the controller only changes the mode of a unit when the mode select inputs change. Assertion of the `drst` input to the `v5gtp_sdi_control` module also causes the DRP controller to reinitialize the GTP_DUAL tile as soon as `drst` is negated.

Reference Clock Selection and Routing

To support transmission of the two HD-SDI bit rates of 1.485 Gb/s and 1.485/1.001 Gb/s, the reference clock must be switched between 148.5 MHz and 148.5/1.001 MHz (or between 74.25 MHz and 74.25/1.001 MHz). This can be done externally to the FPGA by using a clock multiplexer or a clock source that can be switched between the two reference clock frequencies. It can also be done by changing the clock select multiplexer in the GTP_DUAL tile between the reference clock sources available to it.

The reference clock multiplexer is a 4:1 multiplexer. The inputs are the external differential CLKP/N pair associated with that GTP_DUAL tile, CLKINNORTH from the tile below, CLKINSOUTH from the tile above, and GREFCLK from a global clock net. The clock is selected by the refclk_sel port, as shown in [Table 3-7](#).

Table 3-7: Reference Clock Selection

refclk_sel	Clock Selected
00	External CLKP/N input pair
01	GREFCLK
10	CLKINSOUTH
11	CLKINNORTH

[Figure 3-2, page 49](#) shows the reference clock selection and routing multiplexers inside the GTP_DUAL tile. Every GTP_DUAL tile has a dedicated external differential reference clock input signal shown in the figure as CLKP/CLKN. This reference clock is one input to the tile's reference clock select multiplexer. The clock signal from CLKP/CLKN can also be provided to the GTP_DUAL tile above (on the CLKOUTNORTH port) and the GTP_DUAL tile below (on the CLKOUTSOUTH port). The CLKINSOUTH input port is a reference clock coming from the GTP_DUAL tile above. This signal is connected to an input of the reference clock multiplexer. It also can drive the CLKOUTSOUTH port to the GTP_DUAL tile below. Likewise, the CLKINNORTH input port is a reference clock coming from the GTP_DUAL tile below. It is connected to the reference clock multiplexer and can drive the CLKOUTNORTH port to the GTP_DUAL tile above.

The DRP controller can change the value of the REFCLK_SEL attribute to change the reference clock selection multiplexer for the GTP_DUAL tile, as previously described. It can also change the value of the CLKSOUTH_SEL and CLKNORTH_SEL attributes to change what is routed to the CLKOUTSOUTH and CLKOUTNORTH ports, respectively. This allows the application to dynamically change the routing of the reference clocks through the tile to the neighboring GTP_DUAL tiles.

Rate Detection

The third major function of the v5gtp_sdi_control module is rate detection. It is often necessary in an HD-SDI interface to know whether a bitstream is being received with a rate of 1.485 Gb/s or 1.485/1.001 Gb/s, and also to know which of the two 3G-SDI bit rates are being received.

The rate detection logic distinguishes between these bit rates by comparing the frequency of RXUSRCLK to a known standard reference clock frequency. In addition to the ability to determine which of the two bit rates is being received, this logic can also determine if RXUSRCLK is not a match to a valid SDI bit rate, indicating that the input bitstream has stopped or that the CDR of the receiver has lost lock with the input bitstream. Thus, the rate detection circuit is an essential part of the reset logic for the GTP receiver.

The rate detection circuits require a constant frequency reference clock to compare against the frequency of the RXUSRCLK. This reference clock can be any frequency from 27 MHz up to at least 148.5 MHz. The frequency of the reference clock must be specified using a parameter/generic supplied to the `v5gtp_sdi_control` module called `RATE_REFCLK_FREQ`. For example, to specify a 27 MHz reference clock frequency, this parameter is set to 27000000. The reference clock must be a fixed frequency. It does not have to be a frequency that is in any way related to the video sample rate. It is simply a fixed-frequency reference clock.

For each receiver, the rate detection logic generates an output indicating which bit rate is being received. These outputs are called `rx0_rate` and `rx1_rate`. The value of each output is dependent upon the current operating mode of the receiver, as shown in Table 3-8. When running in SD-SDI mode, the rate output should be ignored. In SD-SDI mode, the recovered clock from the GTP RX is locked to the reference clock. Thus, the RXUSRCLK signal, generated from the recovered clock, can be either 148.5 MHz or 148.5/1.001 MHz. Therefore, for SD-SDI mode, the `rx#_rate` output can be either High or Low.

Table 3-8: Rate Detection Output

Mode	rx#_rate	Bit Rate
SD-SDI	X	270 Mb/s
HD-SDI	0	1.485 Gb/s
	1	1.485/1.001 Gb/s
3G-SDI	0	2.97 Gb/s
	1	2.97/1.001 Gb/s

The `v5gtp_sdi_control` module does not output a signal that specifically indicates when the rate detection logic has determined that the RXUSRCLK frequency is invalid. But, if RXUSRCLK remains at an invalid frequency for a period of time that exceeds an error threshold in the rate detector, a CDR reset is automatically initiated.

Connecting the `v5gtp_sdi_control` Module

Table 3-9 lists the ports of the `v5gtp_sdi_control` module.

Table 3-9: `v5gtp_sdi_control` Module Ports

Port Name	I/O	Width	Description
Cable Driver Slew Rate Control			
tx0_slew	Out	1	This is the slew rate control for the cable driver for TX0. This output is High for SD-SDI and Low for HD-SDI and 3G-SDI.
tx1_slew	Out	1	This is the slew rate control for the cable driver for TX1. This output is High for SD-SDI and Low for HD-SDI and 3G-SDI.

Table 3-9: v5gtp_sdi_control Module Ports (Cont'd)

Port Name	I/O	Width	Description
Clocks and DRP Reset			
rate_refclk	In	1	This port must be driven by a stable, constant frequency reference clock and is used as a comparison frequency by the rate detector. The minimum frequency is 27 MHz. Higher frequencies can be used, up to the point where timing cannot be met. This has been tested up to 148.5 MHz in -1 speed grade Virtex-5 devices. This reference clock frequency does not have to be in any way related to the video clock frequency. It must, however, be a constant frequency, never changing even when the SDI mode changes. If the receivers are not used in the GTP_DUAL tile, this clock input can be tied Low.
rx0_usrclk	In	1	This input connects to the global or regional clock that is also connected to the RXUSRCLK input of RX0 of the GTP_DUAL tile. If RX0 of the GTP_DUAL tile is not used, this clock input can be tied Low.
rx1_usrclk	In	1	This input connects to the global or regional clock that is also connected to the RXUSRCLK input of RX1 of the GTP_DUAL tile. If RX1 of the GTP_DUAL tile is not used, this clock input can be tied Low.
tx0_usrclk	In	1	This input connects to the global or regional clock that is also connected to the TXUSRCLK input of TX0 of the GTP_DUAL tile. If TX0 of the GTP_DUAL tile is not used, this clock input can be tied Low.
tx1_usrclk	In	1	This input connects to the global or regional clock that is also connected to the TXUSRCLK input of TX1 of the GTP_DUAL tile. If TX1 of the GTP_DUAL tile is not used, this clock input can be tied Low.
DRP Signals			
daddr	Out	7	This output connects to the DADDR port of the GTP_DUAL tile.
dclk	In	1	This input must be connected to a free-running clock. The same clock signal must be connected to the DCLK input of the GTP_DUAL tile. This clock input also clocks much of the reset logic, so it must be driven with a valid clock signal even if, for some reason, the DRP controller is not used. It is usually possible for dclk and rate_refclk to be sourced by the same clock signal.
den	Out	1	This output connects to the DEN port of the GTP_DUAL tile.

Table 3-9: v5gtp_sdi_control Module Ports (Cont'd)

Port Name	I/O	Width	Description
di	Out	16	This output connects to the DI port of the GTP_DUAL tile.
do	In	16	This input connects to the DO port of the GTP_DUAL tile.
drdy	In	1	This input connects to the DRDY output of the GTP_DUAL tile.
drst	In	1	This is an asynchronous reset for the DRP controller. If used, it should also be connected to the DRST input of the GTP_DUAL tile. A High pulse on this input causes the DRP controller to reinitialize the GTP transceiver attributes that it controls dynamically.
dwe	Out	1	This output connects to the DWE port of the GTP_DUAL tile.
GTPRESET			
clocks_stable	In	1	If the application detects that the GTP reference clock is not stable, it should drive the clocks_stable input Low to cause the gtpreset output to be asserted. The duration of the gtpreset pulse to the GTP_DUAL tile is equal to the duration that clocks_stable is Low. If unused, this input must be High.
gtp_tile_reset	In	1	When pulsed High, this input initiates a long-duration pulse on the gtpreset output. The duration of the pulse is controlled by the GTP transceiver reset counter. If unused, this input must be Low.
gtpreset	Out	1	This output connects to the GTPRESET input port of the GTP_DUAL tile.
Mode Selection, Reference Clock Selection, and Routing			
clknorth_sel	In	1	This input controls the multiplexer that drives the CLKOUTNORTH port of the GTP_DUAL tile. If this input is Low, the clock from the CLKINNORTH port is passed through to CLKOUTNORTH. If this input is High, the clock from the tile's external CLKP/N input is selected for CLKOUTNORTH.
clksouth_sel	In	1	This input controls the multiplexer that drives the CLKOUTSOUTH port of the GTP_DUAL tile. If this input is Low, the clock from the CLKINSOUTH port is passed through to CLKOUTSOUTH. If this input is High, the clock from the tile's external CLKP/N input is selected for CLKOUTSOUTH.

Table 3-9: v5gtp_sdi_control Module Ports (Cont'd)

Port Name	I/O	Width	Description
refclk_sel	In	2	This input selects the reference clock used by the GTP_DUAL tile. See Table 3-7 for details.
rx0_mode	In	2	This input selects the operating mode for RX0. Refer to Table 3-6 for details.
rx1_mode	In	2	This input selects the operating mode for RX1. Refer to Table 3-6 for details.
tx0_mode	In	2	This input selects the operating mode for TX0. Refer to Table 3-6 for details.
tx1_mode	In	2	This input selects the operating mode for TX1. Refer to Table 3-6 for details.
Receiver Rate Detection			
rx0_rate	Out	1	This output indicates the bit rate being received by RX0. Refer to Table 3-8 for details.
rx1_rate	Out	1	This output indicates the bit rate being received by RX1. Refer to Table 3-8 for details.
RX0 Reset Signals			
resetdone0	In	1	This input must be driven by the RESETDONE0 output of the GTP_DUAL tile.
rx0_cd_n	In	1	This input connects to the carrier detect signal from the cable equalizer of RX0. This input must be Low if a good carrier signal is detected, and High otherwise.
rx0_cdr_reset	In	1	The application can drive this input High to initiate a reset of the CDR and PCS sections of RX0.
rx0_clk_mux_sel	Out	1	This output is used to control the global clock buffer driving RXUSRCLK for RX0. When this output is High, RXRECCLK should be selected. When this output is Low, a reference clock should be selected.
rx0_fabric_reset	Out	1	This output can be used to reset the FPGA logic associated with RX0.
rx0_pcs_reset	In	1	The application can drive this input High to initiate a reset of the PCS section of RX0.
rxbufreset0	Out	1	This output connects to the RXBUFRESET0 port of the GTP_DUAL tile.
rxbufstatus0_b2	In	1	This input must be driven by bit 2 of the RXBUFSTATUS0 port of the GTP_DUAL tile.
rxcdrreset0	Out	1	This output connects to the RXCDRRESET0 port of the GTP_DUAL tile.
rxreset0	Out	1	This output connects to the RXRESET0 port of the GTP_DUAL tile.

Table 3-9: v5gtp_sdi_control Module Ports (Cont'd)

Port Name	I/O	Width	Description
RX1 Reset Signals			
resetdone1	In	1	This input must be driven by the RESETDONE1 output of the GTP_DUAL tile.
rx1_cd_n	In	1	This input connects to the carrier detect signal from the cable equalizer of RX1. This input must be Low if a good carrier signal is detected, and is High otherwise.
rx1_cdr_reset	In	1	The application drives this input High to initiate a reset of the CDR and PCS sections of RX1.
rx1_clk_mux_sel	Out	1	This output controls the global clock buffer driving RXUSRCLK for RX1. When this output is High, RXRECCLK should be selected. When this output is Low, a reference clock should be selected.
rx1_fabric_reset	Out	1	This output can be used to reset the FPGA logic associated with RX1.
rx1_pcs_reset	In	1	The application drives this input High to initiate a reset of the PCS section of RX1.
rxbufreset1	Out	1	This output connects to the RXBUFRESET1 port of the GTP_DUAL tile.
rxbufstatus1_b2	In	1	This input must be driven by bit 2 of the RXBUFSTATUS1 port of the GTP_DUAL tile.
rxcdrreset1	Out	1	This output connects to the RXCDRRESET1 port of the GTP_DUAL tile.
rxreset1	Out	1	This output connects to the RXRESET1 port of the GTP_DUAL tile.
TX0 Reset Signals			
tx0_fabric_reset	Out	1	This output resets the datapath of SDI transmitter TX0.
tx0_reset	In	1	The application drives this input High to initiate a PCS reset of TX0.
txbufstatus0_b1	In	1	This input connects to bit 1 of the TXBUFSTATUS0 port of the GTP_DUAL tile.
txreset0	Out	1	This output must be connected to the TXRESET0 port of the GTP_DUAL tile.
TX1 Reset Signals			
tx1_fabric_reset	Out	1	This output resets the datapath of SDI transmitter TX1.
tx1_reset	In	1	The application drives this input High to initiate a PCS reset of the TX1.

Table 3-9: v5gtp_sdi_control Module Ports (Cont'd)

Port Name	I/O	Width	Description
txbufstatus1_b1	In	1	This input connects to bit 1 of the TXBUFSTATUS1 port of the GTP_DUAL tile.
txreset1	Out	1	This output must be connected to the TXRESET1 port of the GTP_DUAL tile.

Global Clock Multiplexer

GTP transceiver applications require that the RXRECCLK from the GTP receiver be buffered by a global or regional clock. This global or regional clock is called the RXUSRCLK because it must be connected to the RXUSRCLK and RXUSRCLK2 ports of the GTP receiver and to the downstream logic that implements the datapath of the receiver. With a 10-bit RXDATA port, RXUSRCLK and RXUSRCLK2 are always the same frequency, but if the RXDATA port is 20 bits wide, the two ports must be driven by different clock frequencies (see the *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* [Ref 1] for more details).

Under certain conditions, RXRECCLK can stop. This happens when the input bitstream stops toggling. It also happens during CDR resets. Many video applications require that the RXUSRCLK signal be kept running at a valid video frequency, even if the input bitstream stops. The solution to this problem is to use a global clock multiplexer. The multiplexer normally selects RXRECCLK to drive RXUSRCLK. However, under the conditions where RXRECCLK is stopped, the multiplexer selects an alternative reference clock to drive RXUSRCLK. One possible source for such an alternative reference clock is REFCLKOUT of the GTP_DUAL tile.

The v5gtp_sdi_control module produces two signals called rx#_clk_mux_sel, one for each RX in the GTP_DUAL tile. These signals can be used to control the clock multiplexers for the two receivers. The Verilog example below shows how the BUFCTRL primitive in the Virtex-5 FPGA can be instantiated to accomplish this function.

```

BUFCTRL #(
    .INIT_OUT      (1),
    .PRESELECT_I0  ("FALSE"),
    .PRESELECT_I1  ("TRUE"))
CLOCK_MUX_0 (
    .O              (rx0_usrclk),
    .CE0            (1'b1),
    .CE1            (1'b1),
    .I0             (rx0_reccclk),
    .I1             (rx0_refclk_out),
    .IGNORE0        (1'b1),
    .IGNORE1        (1'b1),
    .S0             (rx0_clk_mux_sel),
    .S1             (~rx0_clk_mux_sel));

```

REFCLKOUT from a GTP_DUAL tile can be used directly as the backup free-running clock source only if the reference clock frequency provided to the GTP_DUAL tile is 148.5 MHz or 148.5/1.001 MHz, but not if the reference clock frequency is 74.25 MHz or 74.25/1.001 MHz. REFCLKOUT is always exactly equal to the input reference clock frequency. The rx_usrclk output by the BUFCTRL must be 148.5 MHz or 148.5/1.001 MHz. (It can also be 297 MHz or 297/1.001 MHz.) It cannot be 74.25 MHz as would be the case if the input reference clock frequency is 74.25 MHz. A DCM or PLL can be used to double REFCLKOUT when using a 74.25 MHz reference clock frequency to

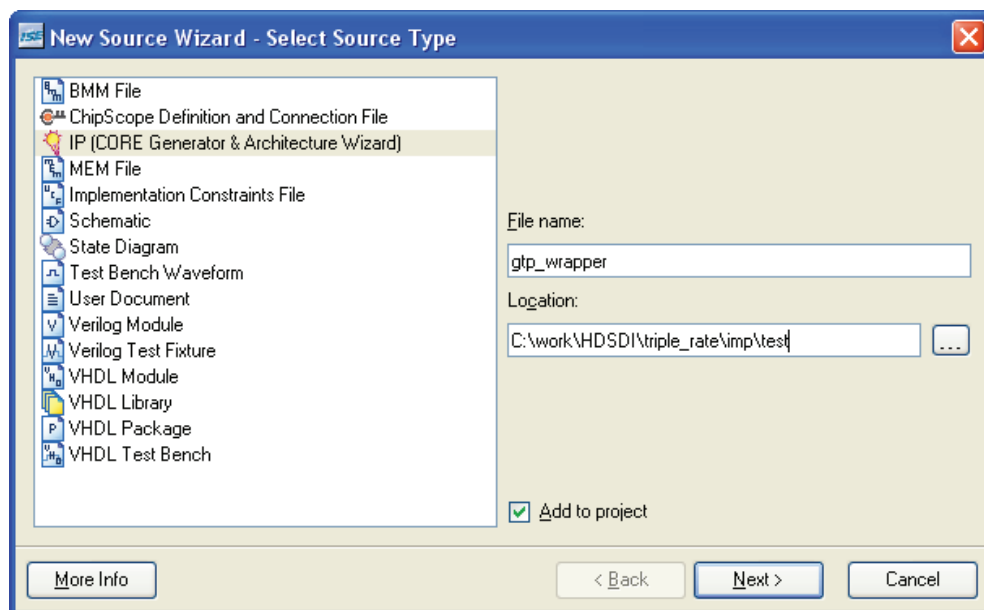
provide a continuous 148.5 MHz clock to the BUFGCTRL. Only one free-running 148.5 MHz clock needs to be created; it can be connected to the BUFGCTRLs for all SDI receivers in the FPGA. Thus, only a single DCM or PLL is required when using a 74.25 MHz REFCLKOUT signal as the source of the 148.5 MHz free-running clock, no matter how many SDI receivers are implemented in the FPGA.

Using the RocketIO GTP Transceiver Wizard

Every SDI application must use the RocketIO GTP Transceiver Wizard to create a GTP wrapper that is customized for the application. The custom GTP wrapper file created by the wizard is specific to the particular FPGA device used and to the particular GTP_DUAL tiles used to implement SDI interfaces in the device. The Wizard is used to specify, among other things, the reference clock frequency used by the GTP_DUAL tile.

When used with the `v5gtp_sdi_control` module, the GTP wrapper module must have certain optional ports enabled. The following description is a step-by-step example of how to create a GTP wrapper file for SDI using the RocketIO GTP Transceiver Wizard (based on version 1.9 of the Wizard):

1. The RocketIO GTP Transceiver Wizard is located in the CORE Generator™ software. One way to launch it is to choose to add a new source in Project Navigator. This opens up the New Source Wizard window, as shown in [Figure 3-19](#).
2. The file name assigned in New Source Wizard window becomes the name of the GTP wrapper module; it is also used to name the files created by the Wizard. Choose IP (CORE Generator & Architecture Wizard) as the source type, as shown in [Figure 3-19](#).
3. Click **Next**.

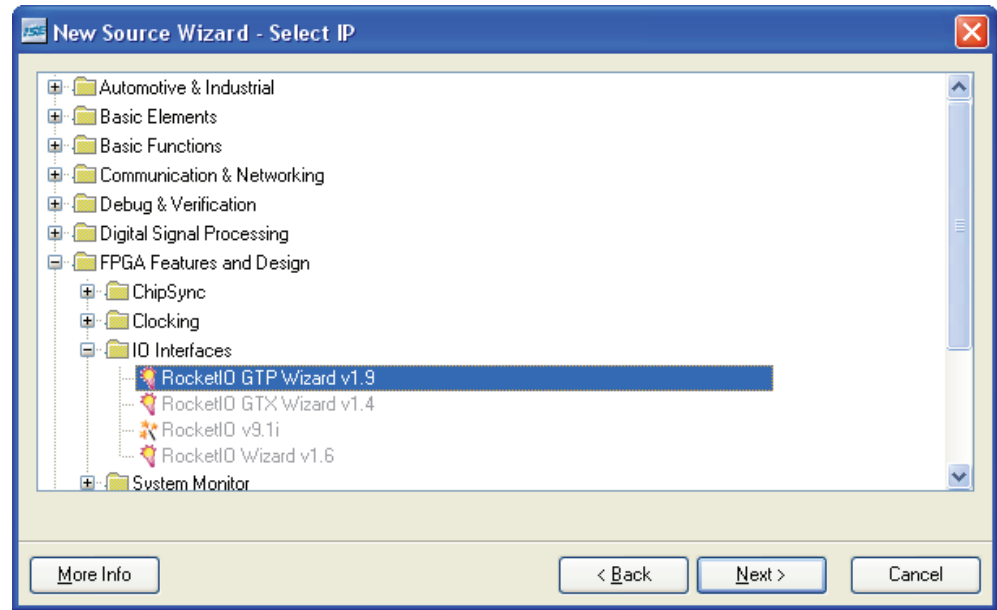


X1014_C3_19_040209

Figure 3-19: New Source Wizard – Source Type Selection

4. The window in [Figure 3-19](#) is replaced with the Select IP window shown in [Figure 3-20](#). In this window, select the RocketIO GTP Transceiver Wizard found in the IO Interfaces folder of the FPGA Features and Design folder. (If the RocketIO GTP

Transceiver Wizard is missing, it is probably because the device type for the project is incorrectly set to a device that does not have GTP transceivers.)



X1014_C3_20_040209

Figure 3-20: New Source Wizard – IP Type Selection

5. Click **Next**.
6. In the next window, click **Finish**.
7. The RocketIO GTP Transceiver Wizard opens. The first page of the Wizard is shown in [Figure 3-21](#). This page allows the selection of the number of GTP_DUAL tiles to be included in the GTP wrapper created by the Wizard. Only those tiles that are used to implement SDI receivers and transmitters should be chosen. If other GTP_DUAL tiles are used to implement other protocols, run the Wizard again to create a separate wrapper file for each additional protocol, creating separate wrappers for the tiles that implement each different protocol. In this example, only one tile is used: tile 112.

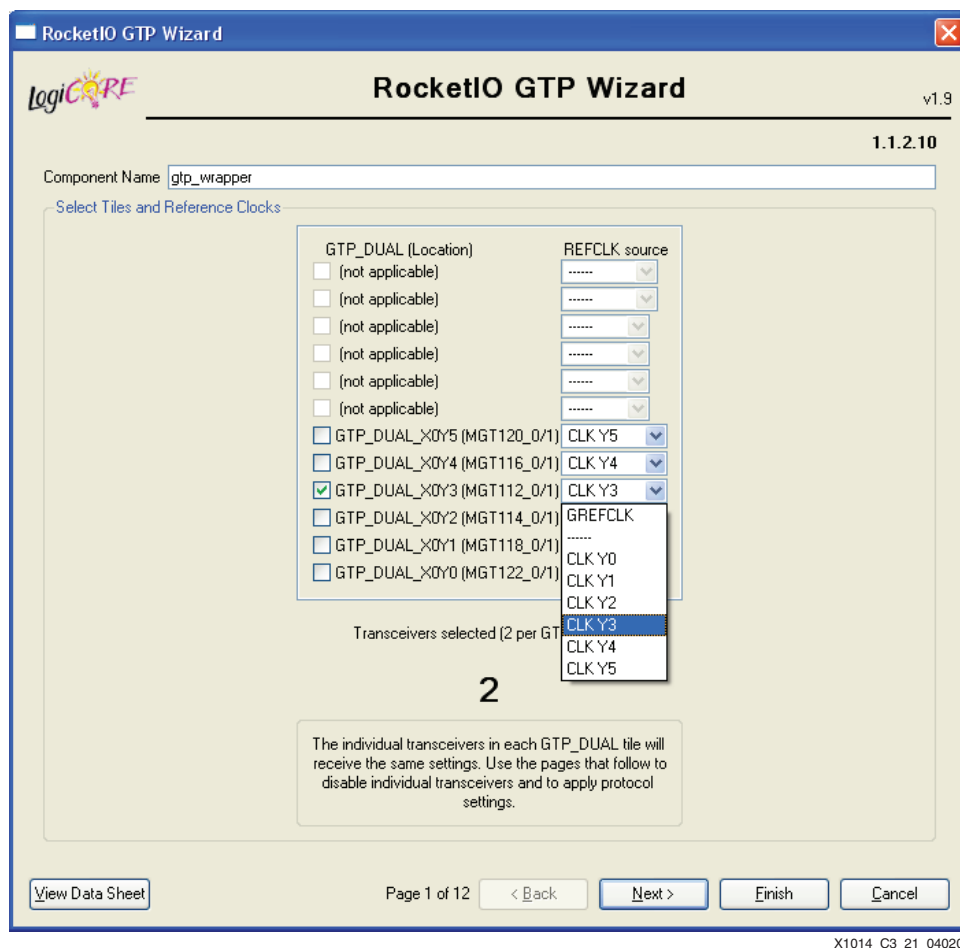


Figure 3-21: RocketIO GTP Transceiver Wizard – Page 1

If multiple tiles are selected, they are all included in one GTP wrapper file. It might be preferable to have an individual wrapper file for each GTP_DUAL tile, even if several of the tiles are used for SDI protocols. In this case, the Wizard should be run for each GTP_DUAL tile separately. It is important that unique GTP wrapper files be created for each GTP_DUAL tile used in the design or that they all be grouped into one GTP wrapper file. A GTP wrapper file must not be instantiated multiple times in a design. The GTP wrapper file can contain tile-specific code and can only be used at the location it was created for.

This page shown in Figure 3-21 also allows the user to select the reference clock for the tile using a pull-down menu next to each tile. The pull-down menu allows for selection of the reference clock from among GREFCLK, the tile's own external reference clock input (listed as CLK Y3 for tile 112), or the external clock from the tiles above and below tile 112. In this case, CLK Y3, the tile's own external reference clock input, is selected. This selection does not affect any connection or attributes within the GTP wrapper file itself. The RocketIO GTP Transceiver Wizard creates an example top-level design with the GTP wrapper file instantiated. The reference clock selections made on this page are used to connect the reference clocks in the example top-level file.

The connection of reference clocks to the GTP wrapper is a critical aspect of successfully implementing an SDI design using the GTP transceiver. This topic is covered in [GTP Transceiver Reference Clock Connections](#), page 86. The reference clock

selection and much of the routing of reference clocks can be overridden dynamically using the DRP controller. Therefore, the reference clock source selected when running the GTP_DUAL tile is not critical.

8. After selecting the tiles to be included and choosing the reference clock source for each one, click **Next** to move to the second page of the Wizard, as shown in Figure 3-22.

X1014_C3_22_040209

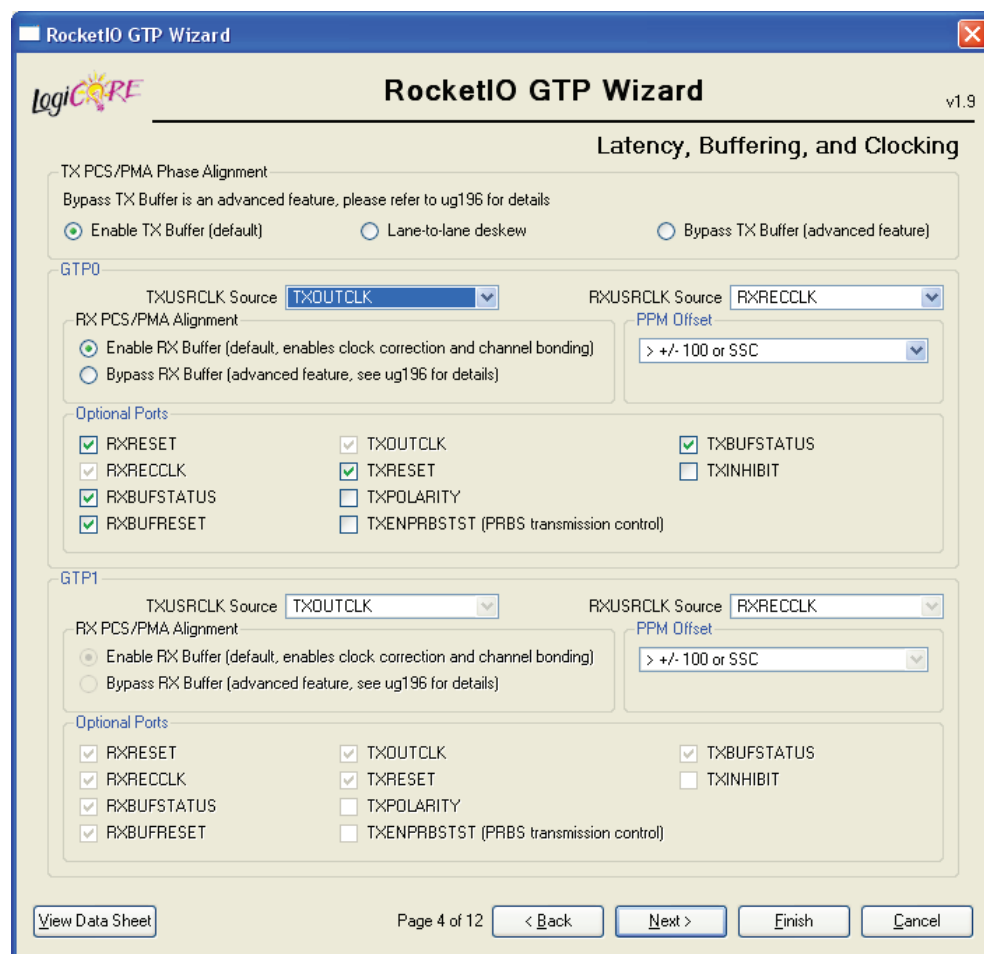
Figure 3-22: RocketIO GTP Transceiver Wizard – Page 2

9. In this window, the line rate and protocol template are selected. Ensure that the internal data width is set to 10, not 8.
10. Choose the silicon version (ES or PRODUCTION), depending on which version of the silicon is to be used. There will be differences in the GTP wrapper based on which silicon version is selected.
11. Set the target line rate to 1.485 Gb/s. Select this line rate even if the target is a 270 Mb/s SD-SDI interface or if the actual target is 3G-SDI at 2.97 Gb/s. This is because the DRP controller in the v5gtp_sdi_control module dynamically sets each unit to the correct line rate, and choosing 1.485 Gb/s as the line rate in the Wizard allows all parameters of the GTP_DUAL tile to be correctly initialized for SDI applications.
12. After the line rate is set, choose the reference clock frequency from the pull-down menu (either 74.25 MHz or 148.5 MHz for SDI).

13. Ensure that:
 - a. **Use Oversampling** is NOT selected.
 - b. **Use Dynamic Reconfiguration Port** is selected.
 - c. **Use REFCLKOUT Port** is selected (it is always selected in this version of the Wizard).

The GTP0 section specifies the protocol for RX0 and TX0. The GTP1 section specifies the protocol for RX1 and TX1.

14. Select **hd sdi** as the protocol template for GTP0 and select **Use GTP0 settings** for GTP1 to default most of the remaining settings on this page and the following pages appropriately for SDI.
15. Click **Next** twice to skip through page 3 and go to page 4, as shown in [Figure 3-23](#).



X1014_C3_23_040209

Figure 3-23: RocketIO GTP Transceiver Wizard – Page 4

Page 4 is mostly concerned with optional ports and whether or not the TX and RX buffers are used or bypassed. There is one global setting at the top of the page for enabling or bypassing the TX buffer that applies to both TX0 and TX1. It is recommended that the TX buffer be enabled. Bypassing the TX buffer requires a TX phase-alignment circuit that must be implemented and added to the design.

16. If the TX buffer is enabled, select **TXOUTCLK** for the TXUSRCLK source. If the TX buffer is bypassed by selecting **Bypass Tx Buffer**, **REFCLKOUT** must be selected as the TXUSRCLK source. RXRECCLK should always be used as the RXUSRCLK source. The RX elastic buffer can be used or bypassed. This selection can be made individually for each receiver in the tile. As with the TX buffer, use of the RX elastic buffer is recommended. If the RX elastic buffer is bypassed, a phase-alignment algorithm must be implemented.
17. Select the optional ports shown in Figure 3-23. The selected ports are all required by the v5gtp_sdi_control module.
18. Click **Next** to move to page 5, shown in Figure 3-24.

RocketIO GTP Wizard v1.9

Preemphasis, Termination, and Equalization

GTP0

Preemphasis and Differential Swing

☐ Pre-emphasis Boost (increase swing by 10%)

Preemphasis level: 000

Main driver differential swing: 000

RX Equalization

☐ Use RX Equalization

High-pass Filter Pole Location

☐ External resistor sets location

☒ Specify individual location: Use RXEQPOLE I

Wideband/Highpass Ratio: Use RXEQMIX Port

RX Termination

☒ Disable internal AC coupling

Termination Voltage: ☒ 2/3 VTTRX* ☐ VTTRX ☐ GND

* recommended value

Optional Ports

☒ RXCDRRESET ☐ RXPOLARITY

GTP1

Preemphasis and Differential Swing

☐ Pre-emphasis Boost (increase swing by 10%)

Preemphasis level: 000

Main driver differential swing: 000

RX Equalization

☐ Use RX Equalization

High-pass Filter Pole Location

☐ External resistor sets location

☒ Specify individual location: Use RXEQPOLE I

Wideband/Highpass Ratio: Use RXEQMIX Port

RX Termination

☒ Disable internal AC coupling

Termination Voltage: ☒ 2/3 VTTRX* ☐ VTTRX ☐ GND

* recommended value

Optional Ports

☒ RXCDRRESET ☐ RXPOLARITY

View Data Sheet Page 5 of 12 < Back Next > Finish Cancel

X1014_C3_24_040209

Figure 3-24: RocketIO GTP Transceiver Wizard – Page 5

The transmitter Pre-emphasis and Differential Swing settings can be set to the normal levels for most HD-SDI applications. They are usually not critical when the SDI cable driver is located fairly close to the FPGA. These settings can be modified, however, to match the requirements of each application.

The internal AC coupling must be disabled and the termination voltage set to $2/3 V_{TTRX}$. This is required because the internal AC coupling capacitors are not large enough to handle the long run lengths of the SDI protocols. External AC coupling capacitors are used between the cable equalizer and the GTP receiver inputs. The optional RXCDRRESET port must also be enabled. This is required by the `v5gtp_sdi_control` module.

None of the other pages of the Wizard are applicable to SDI, so they can either be paged through with the **Next** button until the final screen is reached, or the **Finish** button can be clicked. After **Finish** has been clicked, the Wizard creates the GTP wrapper. It creates two wrapper files in both Verilog and VHDL (a total of four wrapper files). The wrapper files are placed in the project directory into a folder of the same name that was assigned as the file name in Figure 3-19. This folder contains another folder called `src` that contains the actual source code of the wrapper files.

After the steps for the given example are completed, the `src` folder contains four files:

- `gtp_wrapper.v`
- `gtp_wrapper.vhd`
- `gtp_wrapper_tile.v`
- `gtp_wrapper_tile.vhd`

The `gtp_wrapper` file is instantiated inside of `gtp_wrapper` one or more times, once for each tile selected on the first page of the RocketIO GTP Transceiver Wizard. Both the `gtp_wrapper` file and the `gtp_wrapper_tile` file must be added to the project manually. They are not automatically added to the project by running the Wizard.

In addition, the Wizard creates a number of example files. The `ucf` folder includes a file called `example_mgt_top.ucf` containing some constraints that are useful or required. For example, `example_mgt_top.ucf` contains example constraints for the pin locations of the GTP transceiver reference clock inputs. These pins must be constrained, and the example given in the UCF file is a good place to start. The GTP_DUAL tile locations must also be constrained; the UCF file contains example constraints for this, too.

GTP Transceiver Reference Clock Connections

Previous sections of this chapter have described the reference clock routing resources associated with the GTP_DUAL tiles and how these resources can be manipulated dynamically through the DRP. There are other important considerations to understand regarding how to connect the reference clocks in HDL code to ensure that the reference clocks are available in the clock routing structure.

For example, every GTP reference clock that is to be used in the design must be connected to an active GTP_DUAL tile in the HDL design. Otherwise, the reference clock input might be optimized out of the design by the synthesis tools, making it unavailable in the actual implementation.

Every external reference clock input must be connected to an IBUFDS input buffer. This input buffer should be constrained so that it is placed next to the desired GTP_DUAL tile. This is most easily done by applying pin LOC constraints to the signals connected to the IBUFDS inputs.

For example, the following Verilog code instantiates an IBUFDS with input signals called `PAD_mgtclk_hd_p` and `PAD_mgtclk_hd_n`. These two signals are defined as inputs to the top-level module of the design.

```
IBUFDS #(
    .IOSTANDARD ("LVDS_25"))
```

```
MGTCLK122 (
    .I      (PAD_mgtclk_hd_p) ,
    .IB     (PAD_mgtclk_hd_n) ,
    .O      (mgtclk_148_5M) );
```

The following two constraints in the UCF file constrain the two input signals to specific pins that are dedicated to the reference clock input buffer associated with GTP_DUAL tile 112 in an XC5VLX50T-FF1136 device.

```
NET "PAD_mgtclk_hd_p"      LOC = "P4";
NET "PAD_mgtclk_hd_n"      LOC = "P3";
```

These constraints ensure that the reference clock enters the FPGA at the correct place. However, it is not sufficient to ensure that the reference clock is available to the GTP reference clock routing resources. If the output of the IBUFDS is not connected to the CLKIN pin of at least one GTP_DUAL tile that is active in the design, it is optimized out of the design. Thus, in this case, the mgtclk_148_5M signal must be connected the CLKIN_IN port of at least one GTP_DUAL tile. Further, that GTP_DUAL tile must also be constrained to its proper location. Finally, the GTP_DUAL tile must be connected sufficiently so that it is not optimized out of the design. Even if it is not actually used in the design, the GTP_DUAL tile must be minimally connected so as to make the synthesis tools believe that it is used in the design. The easiest way to ensure that the GTP_DUAL tile is not optimized out of the design is to connect at least some of its RXN and TXN ports (differential serial I/O signals) to ports on the top-level design and connect a reference clock to its CLKIN port.

It is not possible to connect the output of the IBUFDS to anything other than CLKIN ports of GTP_DUAL tiles. Because this IBUFDS has been constrained to a location that is a dedicated GTP transceiver reference clock input buffer, it cannot be connected to anything other than the CLKIN ports of GTP_DUAL tiles. The output of the IBUFDS can be connected to the CLKIN ports of more than one GTP_DUAL tile. The tools route the clock through the CLKNORTH and CLKSOUTH routing resources to the various GTP_DUAL tiles that it drives.

Any unused GTP_DUAL tile through which reference clocks are routed must also be instantiated in the design and minimally connected so that it is not optimized out of the design. If a GTP_DUAL tile that is used solely to route reference clocks is not instantiated in the design or gets optimized out of the design because it was not connected to anything, it is powered down and reference clocks cannot be routed through the tile. Unused GTP_DUAL tiles used only for reference clock routing must also have their external power pins connected on the circuit board. If the tile is not powered, reference clocks cannot be routed through it.

With the dynamic reference clock routing control provided by the DRP controller, if the reference clocks are available to the GTP transceiver reference clock routing resources, they can be switched to the required GTP_DUAL tiles. However, this switching occurs some time after the FPGA configuration is complete. It is often useful to ensure that each GTP_DUAL tile has a valid HD-SDI reference clock connected to its CLKIN pin in the HDL design file so that a valid reference clock is provided to the tile's PLL immediately after the FPGA comes out of configuration.

With the dynamic control of the reference clock routing, it is possible to violate the limits placed on the number of GTP_DUAL tiles that can be driven by a single external reference clock input. However, these limits must still be observed, otherwise signal degradation of the reference clock could prevent the design from working properly.

Implementing SMPTE Serial Digital Interfaces with RocketIO GTX Transceivers

Summary

This chapter serves as a general introduction to using Virtex®-5 FPGA GTX transceivers to implement SMPTE SDI applications. The information contained in this chapter is essential to successfully implementing any SDI application with Virtex-5 FPGA GTX transceivers. This chapter should be read and understood before reading one of the target application chapters, such as [Chapter 8, Triple-Rate SDI for Virtex-5 FXT and TXT Devices](#).

An SDI interface generally consists of three main parts: the GTX transceiver, the `v5gtx_sdi_control` module, and the protocol module or modules (such as the triple-rate SDI RX module). The first two of these three parts are described in this chapter. The protocol modules are described in protocol-specific chapters.

The GTX transceivers are instantiated in designs by creating a GTX wrapper module using the RocketIO™ Wizard and instantiating this wrapper into the design. This chapter contains a section that describes how to run the RocketIO Wizard to create a GTX wrapper for an SDI application. The reference designs found in the protocol chapters generally include example GTX wrappers created by the RocketIO Wizard. These wrapper modules are examples only and are specific to the GTX_DUAL tiles used in the Virtex-5 FPGA FX70T on the ML571 SDV demonstration board. The RocketIO Wizard must be used to create custom GTX wrappers for every application.

Introduction

Virtex-5 FXT and TXT devices have RocketIO GTX serial transceivers capable of operating at speeds of up to 6.5 Gb/s. These transceivers can be used to implement the serial digital interface standards commonly used in the video broadcast industry:

- SD-SDI
- HD-SDI
- 3G-SDI
- DVB-ASI

A number of features of the GTX transceiver in the Virtex-5 FPGA are particularly useful when implementing the SDI standards:

- A single reference clock frequency allows reception of both HD-SDI bit rates, both 3G-SDI bit rates, and 270 Mb/s SD-SDI.

- All receivers can be completely independent, allowing them to independently switch between HD-SDI, 3G-SDI, and SD-SDI.
- Just two reference clock frequencies, 148.5 MHz and 148.5/1.001 MHz (or 74.25 MHz and 74.25/1.001 MHz), allow transmission of both HD-SDI bit rates, both 3G-SDI bit rates, and 270 Mb/s SD-SDI.
- A dynamic reconfiguration port (DRP) allows the RX and TX units in each GTX_DUAL tile to be dynamically switched between SDI standards and also allows dynamic switching of reference clocks to the GTX transceivers.
- Every GTX_DUAL tile has an external reference clock input and access to reference clocks from tiles above and below it.

Differences from GTP Transceivers

There are a number of differences between the GTP and GTX transceivers in Virtex-5 devices. The major differences that affect SDI implementations are:

- The Xilinx® SDI reference designs for GTP transceivers use 10-bit RXDATA and TXDATA interfaces because this minimizes clock resources with GTP transceivers and eliminates the need for PLLs or DCMs. With GTX transceivers, clock resource requirements are minimized when 20-bit RXDATA and TXDATA interfaces are used. Because of the use of 20-bit datapaths, the main datapath clock frequencies are half the frequency with GTX transceivers than with GTP transceivers. The datapath clock frequencies are 74.25 MHz for HD-SDI and 148.5 MHz for 3G-SDI. SD-SDI uses a datapath clock of 148.5 MHz with a 27 MHz clock enable.
- The address map of the DRP port is different. Thus, a different control module, `v5gtx_sdi_control`, must be used to properly control the GTX transceiver.

Using the GTX Transceiver to Implement SDI

This section provides a brief introduction to the architecture and features of the GTX transceiver, emphasizing those features used when implementing the SDI standards. Refer to the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide* [Ref 3] for complete details of the GTX transceiver.

GTX_DUAL Tiles

Pairs of GTX transceivers are grouped into tiles called GTX_DUAL tiles. Each of these tiles contains two receivers and two transmitters all sharing a common clock multiplication PLL. Different Virtex-5 devices have different numbers of GTX_DUAL tiles.

The clock multiplication, or Physical Medium Attachment (PMA) PLL is shared by the entire GTX_DUAL tile. Sharing the PMA PLL does place some restrictions on implementing different standards in different units within the tile.

The two receivers in the GTX_DUAL tile can independently receive any of the 3G-SDI and HD-SDI bit rates and SD-SDI or DVB-ASI at 270 Mb/s. They can switch between the various standards independently with no restrictions.

The two transmitters in the GTX_DUAL tile must always operate at exact multiples of the same reference clock. For example, it is not possible to have one transmitter in the tile operating at 1.485 Gb/s while the other operates at 1.485/1.001 Gb/s. These two bit rates require different reference clock frequencies for the transmitters. Because only one reference clock frequency is available in the tile at any one time, both transmitters must use

it. This also means that it is not possible to transmit 1.485 Gb/s with one transmitter and 1.485 Gb/s + 10 ppm (for example) with the other transmitter in the same tile. Both transmitters run at exact multiples of the single reference clock frequency. Applications that require all transmitters to be completely independent are restricted to using a single transmitter unit per GTX_DUAL tile.

Changing the GTX_DUAL tile's reference clock frequency to change the transmitter's bit rate (for example, changing the reference clock frequency from 148.5 MHz to 148.5/1.001 MHz) usually upsets the operation of the receivers for a short period of time. The duration and severity of this upset has not been characterized. Xilinx recommends isolating the SDI transmitters in their own GTX_DUAL tile, unless the application can tolerate a brief disturbance of the receivers each time the reference clock frequency to the GTX_DUAL tile is changed.

GTX Transceiver PMA PLL Section

The shared PMA PLL (Figure 4-1) in the GTX_DUAL tile is used to multiply the reference clock up to the proper frequency required to support reception and transmission at the desired bit rate. The PLL is not part of the GTX receiver's CDR section; it is used only as a clock multiplication PLL to provide a reference clock to the CDR. The GTX RX can receive bitstreams that are over ± 1000 ppm away from the PLL clock frequency. The GTX TX, however, uses the clock produced by the PLL directly as the clock for the serializer. Thus, the transmitters always run at exact multiples of the reference clock frequency.

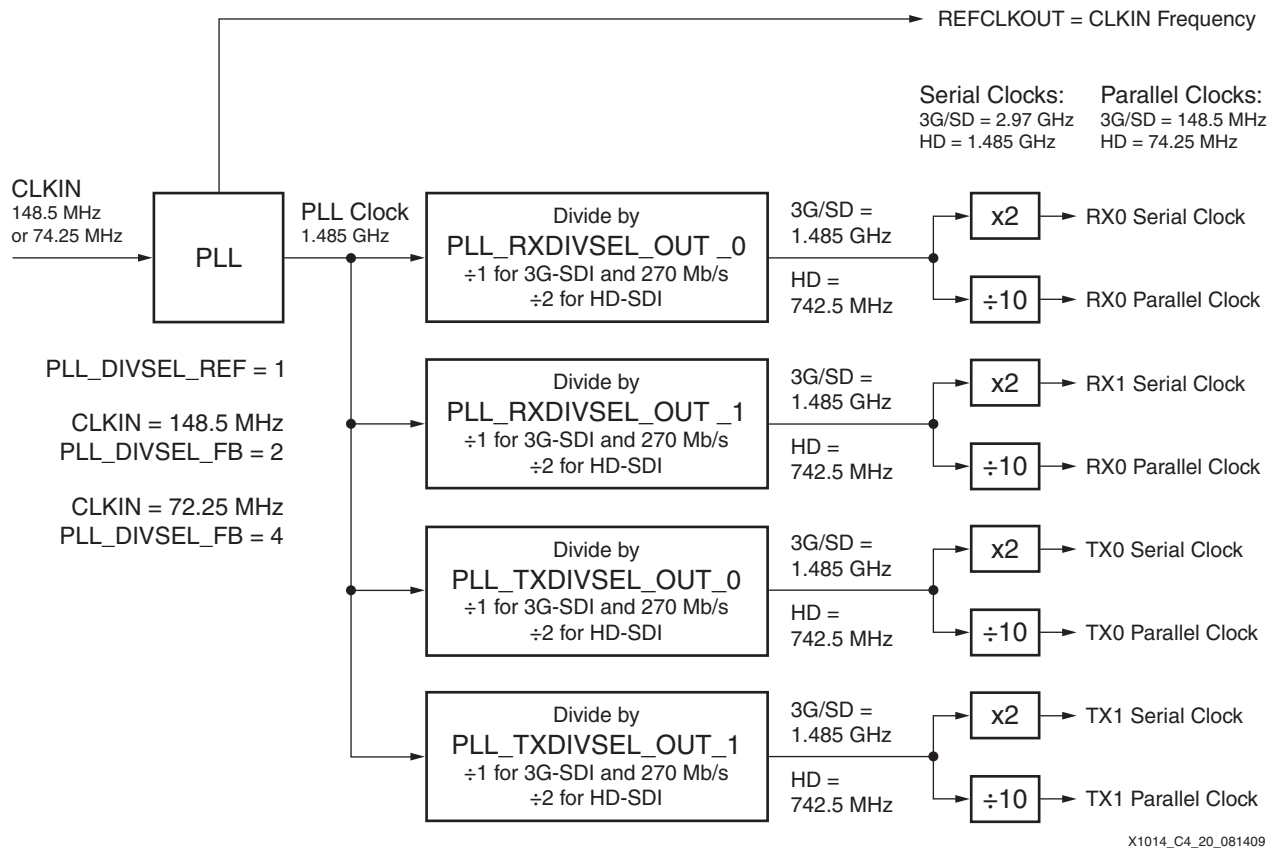


Figure 4-1: PMA PLL Section

For SDI applications, the PLL must be given a reference clock of 148.5 MHz or 148.5/1.001 MHz. Alternatively, 74.25 MHz and 74.25/1.001 MHz can be used. SDI receivers can use any one of these reference clock frequencies to receive SD-SDI at 270 Mb/s, both HD-SDI bit rates, and both 3G-SDI bit rates. Thus, only a single reference clock frequency is required for a triple-rate SDI receiver, and it can be any one of 148.5 MHz, 148.5/1.001 MHz, 74.25 MHz, or 74.25/1.001 MHz. SDI transmitters, however, always transmit at a bit rate that is an exact multiple of the reference clock frequency supplied to the PLL. To transmit SD-SDI, the reference clock frequency must be 148.5 MHz (or 74.25 MHz). To transmit HD-SDI at 1.485 Gb/s or 3G-SDI at 2.97 Gb/s, the reference clock frequency must be 148.5 MHz (or 74.25 MHz). To transmit HD-SDI at 1.485/1.001 Gb/s or 3G-SDI at 2.97/1.001 Gb/s, the reference clock frequency must be 148.5/1.001 MHz (or 74.25/1.001 MHz).

The PLL multiplies the reference clock by 10 or 20 to generate a 1.485 GHz PLL clock. When using 148.5/1.001 MHz or 74.25/1.001 MHz as the reference clock frequency, all frequencies shown in [Figure 4-1](#) are divided by 1.001.

Each receiver unit has a PLL clock divider that can divide the PLL clock by one, two, or four. To receive 3G-SDI and 270 Mb/s SD-SDI or DVB-ASI, this divider is set to one. For HD-SDI, this divider is set to two. The serial portion of the receiver runs at twice the frequency of the clock output by this divider (symbolically shown in [Figure 4-1](#) by the x2 block, but actually accomplished by using both edges of the clock). The parallel section of the receiver runs at one-tenth this clock rate.

The transmitters also have individual dividers which, like those for the receivers, are set to divide by one for 3G-SDI, SD-SDI, DVB-ASI, and divide by two for HD-SDI.

The PLL_RXDIVSEL_OUT and PLL_TXDIVSEL_OUT attributes can be changed dynamically through the DRP. Using the setup as shown in [Figure 4-1](#), the clock dividers can be changed for each individual RX or TX unit to change between 3G-SDI, HD-SDI, and SD-SDI modes. Other attributes must also be modified when changing between these modes, but changing the PLL clocks is very straightforward.

Additionally, the CLKIN signal is output from the GTX_DUAL tile as REFCLKOUT. This output is always equal in frequency to CLKIN and is not affected by any of the multiplier or divider settings of the PLL.

The PLL attributes described in this section are set up in the GTX wrapper files produced by the RocketIO Wizard. Furthermore, the attributes are dynamically controlled by the DRP controller located in the `v5gtx_sdi_control` module.

For HD-SDI, the GTX PMA PLL must run at 1.485 or 1.485/1.001 GHz. Reception of HD-SDI is severely degraded if the PLL frequency is 2.97 GHz or 2.97/1.001 GHz. In reality, it is the PLL_RXDIVSEL_OUT value that is critical. The HD-SDI receiver is severely degraded if PLL_RXDIVSEL_OUT is set to divide the PLL by four. Only a divide by two setting works for HD-SDI, thus requiring a 1.485 GHz PLL frequency for HD-SDI as shown in [Figure 4-1](#).

GTX Transceiver Reference Clocks

Each active GTX_DUAL tile must be provided with a reference clock. Each tile has a reference clock multiplexer that selects the reference clock from one of four sources. The sources are:

- An external differential reference clock input (CLKP/N)
- A clock from the tile above (CLKINSOUTH)
- A clock from the tile below (CLKINNORTH)
- A global clock tree (GREFCLK)

Figure 4-2 shows the reference clock selection multiplexer and clock routing resources of a GTX_DUAL tile.

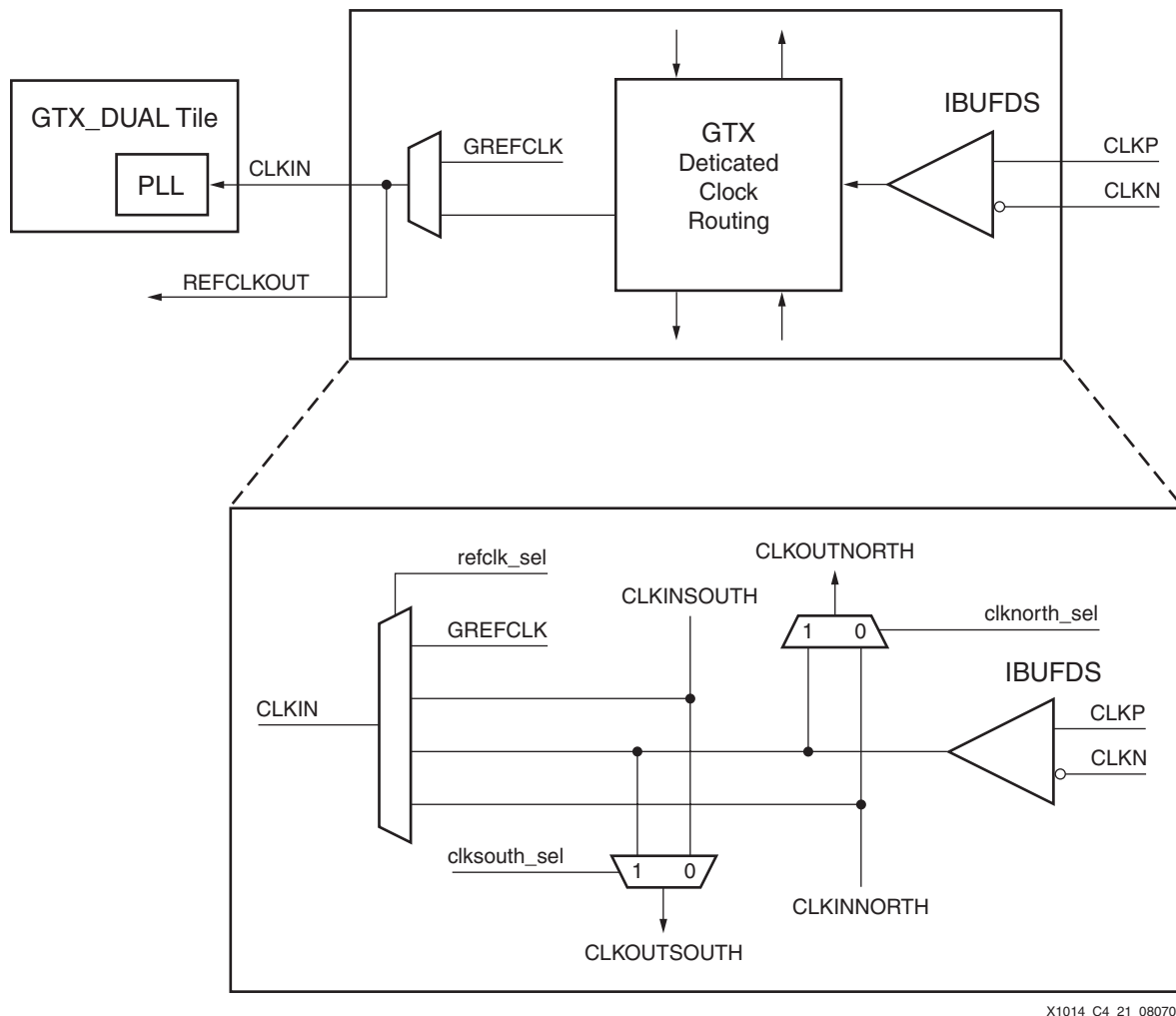


Figure 4-2: GTX Transceiver Reference Clock Routing and Selection Multiplexer

The use of GREFCLK as a reference clock to the GTX_DUAL tile for SDI applications has not been characterized. A global clock tree connected as a reference clock to the GTX transceiver through the GREFCLK port has more jitter than a reference clock connected to the FPGA through a dedicated GTX transceiver reference clock external input (CLKP/N). This makes it difficult to meet HD-SDI and 3G-SDI TX output jitter specifications when

GREFCLK is used as the reference clock source. Xilinx does not recommend the use of GREFCLK as the reference clock source for SDI applications.

Each GTX_DUAL tile has an external differential clock input (CLKP/N). This dedicated input cannot be used as a general-purpose FPGA I/O pin or be used to directly provide a clock to the global clock routing resources of the FPGA. However, reference clocks can be routed from the GTX_DUAL tile to the global clock resources using the REFCLKOUT port of the GTX_DUAL tile. Whatever clock is selected by a GTX_DUAL tile's reference clock multiplexer is output on that tile's REFCLKOUT port.

The CLKP/N external clock input is connected to one input of the tile's reference clock multiplexer. It is also connected to multiplexers that drive the tile's CLKOUTNORTH and CLKOUTSOUTH output ports that provide reference clocks to the tiles above and below. The CLKOUTNORTH port drives a reference clock out to the tile above and can be driven by either the CLKP/N external clock input or by the CLKINNORTH signal from the tile below. Likewise, the CLKOUTSOUTH port drives a reference clock out to the tile below and can either be driven by the CLKP/N external clock input or by the CLKINSOUTH port from the tile above.

The CLKSOUTH and CLKNORTH routing resources provide a flexible way to drive multiple GTX_DUAL tiles from a single external reference clock input. Using the multiplexers in each tile to control the signals output on the CLKOUTSOUTH and CLKOUTNORTH ports of each tile, the CLKSOUTH and CLKNORTH routing resources can be broken up into multiple segments, allowing a high degree of flexibility in clock routing.

There are some limitations on the routing of clocks using the CLKSOUTH and CLKNORTH routing resources. A sourcing GTX_DUAL tile is limited to driving three contiguous GTX_DUAL tiles above and below it. Thus, a maximum of seven contiguous tiles can use an external reference clock input: the sourcing tile, three tiles above the sourcing tile, and three tiles below the sourcing tile. The sourcing tile is the GTX_DUAL tile at which the reference clock enters the FPGA on the tile's external CLKP/N inputs.

[Figure 4-3, page 95](#) shows how one external reference clock input can source seven tiles. The red lines show how the reference clock entering the FPGA at GTX_DUAL tile D is connected to the three tiles above it using the CLKNORTH routing and the three tiles below it using the CLKSOUTH routing.

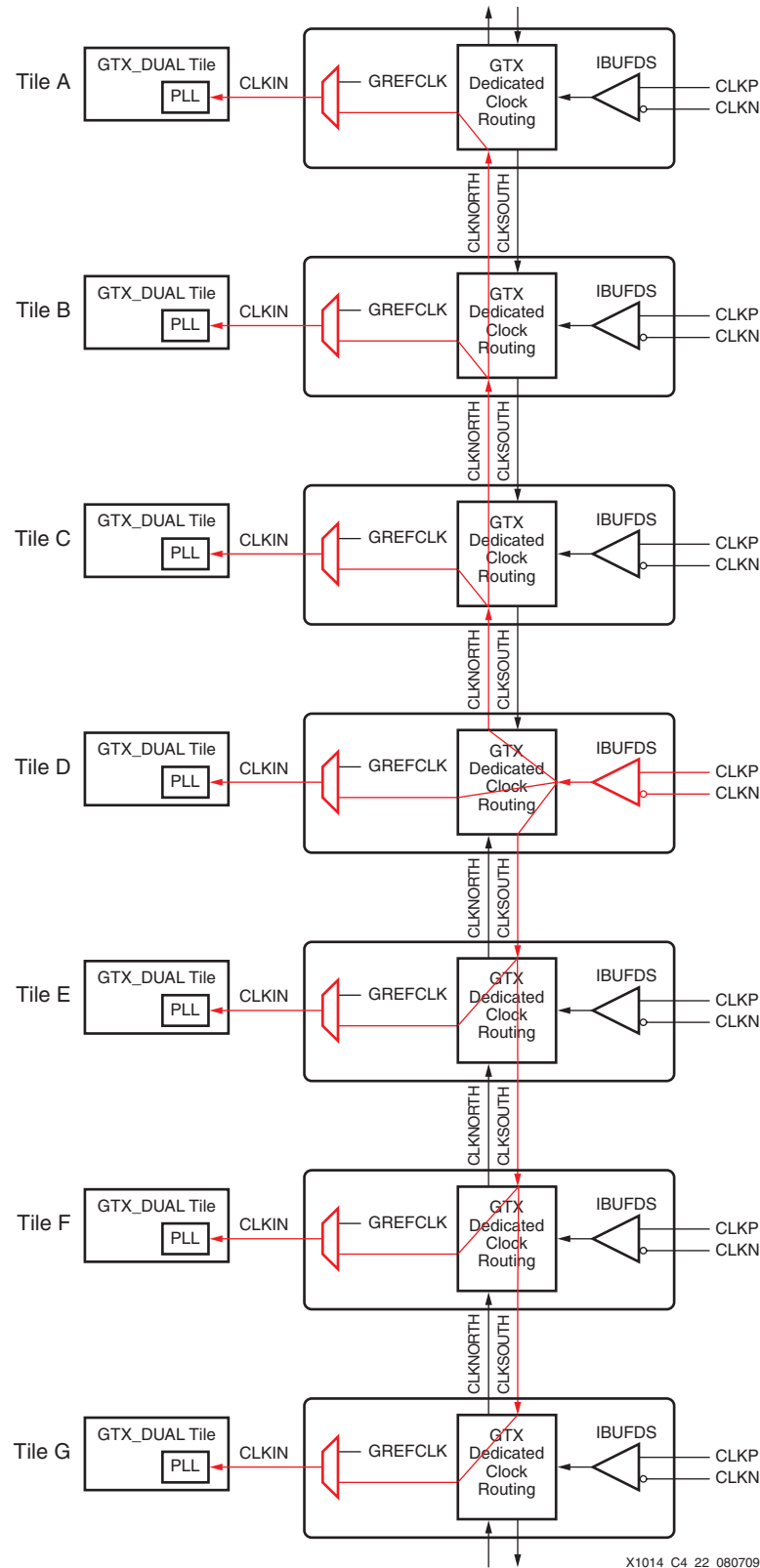


Figure 4-3: Routing a Reference Clock to Seven GTX_DUAL Tiles

In [Figure 4-3](#), there is a single CLKSOUTH and a single CLKNORTH routing resource between any two adjacent tiles. Thus, it is not possible to send two reference clocks in the same direction between any two adjacent tiles. For example, there is no way to get a clock entering the FPGA at tile F's CLKP/N input to a tile located below tile G because the CLKOUTSOUTH resource is used between tile F and tile G. However, a clock entering the FPGA at tile G could drive tiles below this tile because the CLKSOUTH routing resource is free below tile G. Also, tile D blocks any other reference clocks from being routed through that tile because both its CLKOUTNORTH and CLKOUTSOUTH ports are used. Tile D could, however, receive a reference clock from the tiles above or below it.

These are static restrictions. The clock selection multiplexer and the multiplexers that drive the CLKOUTNORTH and CLKOUTSOUTH ports of each tile can be dynamically switched through the DRP, allowing more complex dynamic clock management strategies, if needed.

It is possible to route a reference clock through an unused GTX_DUAL tile. However, the unused tile must be instantiated in the design and powered. If it is not, the tile is powered down and no reference clocks pass through the tile using the CLKNORTH and CLKSOUTH routing resources. See [GTX Transceiver Reference Clock Connections, page 129](#) for strategies describing how to connect reference clocks to the GTX_DUAL tiles in HDL designs. The *Virtex-5 FPGA RocketIO GTX Transceiver User Guide* [Ref 3] describes the power requirements for an unused GTX_DUAL tile used to route reference clocks—the MGTAVTTRX and MGTAVTTTX pins must be powered, but power filtering is not required.

If a single reference clock source must be connected to more tiles than can be reached via the internal CLKNORTH and CLKSOUTH routing resources (more than seven contiguous tiles), that clock source must be connected to multiple GTX transceiver external clock inputs.

Any reference clock sources connected to the external CLKP/N differential input of any GTX_DUAL tile must be a differential and must meet the specifications and requirements given in the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide* [Ref 3] and the *Virtex-5 FPGA Data Sheet: DC and Switching Characteristics* [Ref 2]. In particular, the reference clock source must be AC coupled to the clock input pins. The input buffer provides internal termination and a common mode bias voltage for the AC coupled input clock signal.

The GTX receiver is relatively immune to reference clock jitter, up until the point where the tile's clock multiplication PLL cannot lock to the reference clock due to excessive jitter.

The GTX transmitter's output jitter is directly impacted by the reference clock jitter. SD-SDI and DVB-ASI transmitters have very large absolute jitter budgets and, therefore, are capable of operating with relatively large amounts of reference clock jitter. HD-SDI transmitters have a much smaller absolute jitter budget, and 3G-SDI has an even smaller absolute jitter budget. Thus, low-jitter reference clock sources are essential for HD-SDI, and even more so for 3G-SDI. Refer to the *Virtex-5 FPGA RocketIO GTX Transceiver CEI-6G Electrical Specification* [Ref 4] for more details on how reference clock jitter affects the transmitter output jitter.

GTX Receiver

This section briefly describes how the GTX receiver in the Virtex-5 FPGA is used to receive 3G-SDI, HD-SDI, SD-SDI, and DVB-ASI. Refer to the individual SDI protocol chapters for details of receiving each of these standards with the GTX transceiver.

Receiving 270 Mb/s SD-SDI and DVB-ASI

The GTX receiver implements clock and data recovery for bit rates ranging from 750 Mb/s up to 6.5 Gb/s. It also has a built-in 5X oversampling digital receiver that supports bit rates from 100 Mb/s to 750 Mb/s. However, for several reasons, this built-in 5X oversampling digital receiver is typically not used for SD-SDI and DVB-ASI at 270 Mb/s. Instead, the GTX receiver runs at 11 times the 270 Mb/s bit rate (2.97 Gb/s), is locked to the reference clock frequency, and asynchronously oversamples the SD-SDI and DVB-ASI bit streams. A data recovery unit (DRU) built in the programmable logic of the FPGA takes the oversampled data from the GTX receiver and recovers the data. The advantages and disadvantages of these two approaches to supporting 270 Mb/s are illustrated in [Table 4-1, page 97](#). The benefits of using the 11X DRU (increased RX jitter tolerance, fewer reference clocks, and complete independence of the two RX units in the same tile) outweigh the benefits of using the built-in 5X oversampling digital receiver in SD-SDI and DVB-ASI applications.

Table 4-1: Comparison of Oversampling Techniques Supporting 270 Mb/s Reception

Parameter	Built-in 5X Oversampling Digital RX	FPGA Logic-Based 11X Oversampling DRU
RX Jitter Tolerance	5X oversampling provides marginal tolerance for the long run lengths and pathological patterns of SD-SDI. Xilinx does not recommend using the 5X oversampling for SD-SDI applications because of the pathological pattern tolerance.	11X oversampling provides better support for SD-SDI long run lengths and pathological patterns. It also provides improved RX jitter tolerance in general.
Unit Independence	A single enable bit enables the 5X oversampling mode for the entire tile. When this mode is enabled, both receivers in the GTX_DUAL tile run in oversampling mode. Both transmitters are also in oversampling mode and limited to bit rates between 100 Mb/s and 750 Mb/s.	When using the FPGA logic-based 11X DRU, the two receivers in the GTX_DUAL tile are completely independent. One RX unit can receive SD-SDI or DVB-ASI at 270 Mb/s while the other RX unit in the tile receives HD-SDI or 3G-SDI.
Reference Clock	When using the 5X digital receiver to receive SD-SDI or DVB-ASI, the GTX_DUAL tile's reference clock must be a different frequency than that required for HD-SDI and 3G-SDI. Thus, two reference clock frequencies are required to receive these various bit rates. This further prevents the units in the tile from simultaneous support of different SDI standards.	With the 11X oversampler, the reference clock used for 270 Mb/s SD-SDI and DVB-ASI is the same frequency as that required for HD-SDI and 3G-SDI. Thus, a single reference clock frequency allows both receivers in the GTX_DUAL tile to independently receive 270 Mb/s SD-SDI and DVB-ASI, either HD-SDI bit rate, and either 3G-SDI bit rate.
Recovered Clock	The 5X digital receiver recovers a clock.	The 11X oversampling DRU does not recover a clock. The DRU provides a data ready signal asserted when a 10-bit data word has been recovered. This signal is usually used as a clock enable. The NI-DRU used with the Virtex-5 FPGA GTX transceiver can, however, use an otherwise unused GTX TX unit to synthesize a recovered 270 MHz clock.
Resource Usage	The 5X digital receiver is built into each GTX transceiver.	The 11X oversampling DRU uses a small amount of FPGA resources.

The biggest downside to using the FPGA logic DRU for SD-SDI is the lack of a recovered clock. However, techniques are available to recover a clock from recovered video. For example, it is possible to use a genlock solution to recover a clock based on the video sync signals in the recovered video. As stated in [Table 4-1](#), the NI-DRU used with the

Virtex-5 FPGA GTX transceiver can use an otherwise unused GTX TX to synthesize a recovered 270 MHz clock from the SD-SDI signal.

Important RX Ports

Table 4-2 shows the important RX-specific ports of the GTX_DUAL tile used by SDI applications. There are other RX ports used for SDI applications, but they are typically controlled by the `v5gtx_sdi_control` module.

Each RX in the tile has an independent set of the ports shown in Table 4-2. The italicized *n* in the port name identifies which RX unit in the tile (zero or one) the signal belongs to. For example, the RXUSRCLK2 port for RX0 is called RXUSRCLK20 and the RXUSRCLK2 port for RX1 it is called RXUSRCLK21. The port names are also prefixed by a tile number, TILE*x*, where the *x* indicates which GTX_DUAL tile the port belongs to when the GTX wrapper file contains multiple GTX_DUAL tiles. For convenience, the ports are described throughout the rest of this application note without the prefixes and suffixes. For example, TILE*x*_RXRECCLK*n*_OUT is called RXRECCLK.

Table 4-2: Important RX Ports

Port	I/O	Description
TILE <i>x</i> _RXP <i>n</i> _IN TILE <i>x</i> _RXN <i>n</i> _IN	In	This is the differential receive serial data pair.
TILE <i>x</i> _RXDATA <i>n</i> _OUT	Out	This is the receive data bus of the receiver to the FPGA logic. For SDI applications, this port can be either 10 or 20 bits wide. However, to minimize global clock resources, a 20-bit RXDATA port is generally preferred.
TILE <i>x</i> _RXRECCLK <i>n</i> _OUT	Out	This is the recovered clock from the clock and data recovery unit.
TILE <i>x</i> _RXUSRCLK <i>n</i> _OUT	In	This is the input clock used for internal RX logic after the RX elastic buffer. This clock must run at 1/20th the bit rate (148.5 MHz for 3G-SDI and 74.25 MHz for HD-SDI). For SD-SDI, it must have a frequency of 148.5 MHz. This clock is usually derived directly from RXRECCLK, typically by simply buffering RXRECCLK through a BUFG or BUFR.
TILE <i>x</i> _RXUSRCLK2 <i>n</i> _OUT	In	This is the input clock used for the RX interface between the FPGA and the GTX transceiver. The frequency of this clock depends on the width of the RXDATA port. When RXDATA is 20 bits wide, this clock is equal in frequency to RXUSRCLK and, in fact, RXUSRCLK and RXUSRCLK2 are driven by exactly the same clock source.

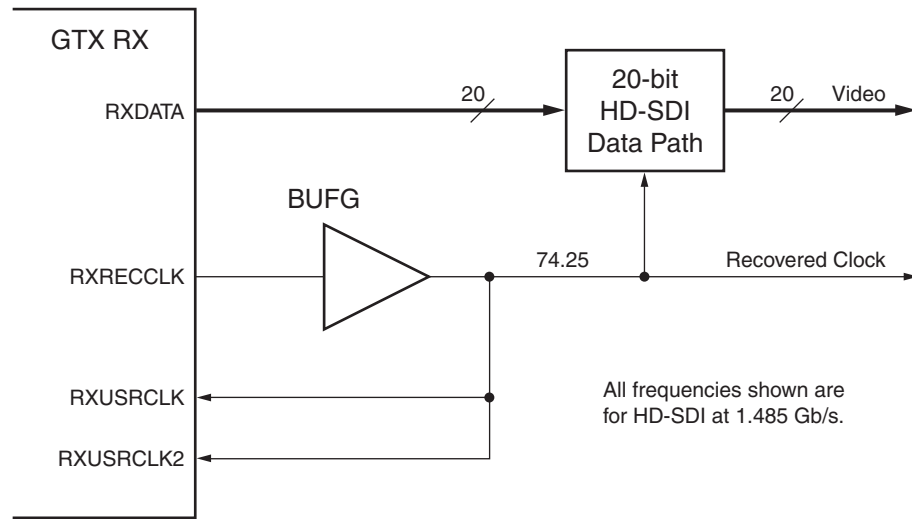
The RXP/N input signal pair is connected to a CML receiver that has programmable internal termination. The CML receiver also has built-in AC coupling capacitors that can be bypassed. For SDI applications, these internal AC coupling capacitors are always bypassed (by setting the AC_CAP_DIS attribute of the receiver to TRUE) because they are too small to pass the pathological waveforms present in SDI streams. Instead, external AC coupling capacitors are always used between the external cable equalizer and the receiver input pins.

RX Clocking

For HD-SDI and 3G-SDI, the CDR section recovers data and a clock. The recovered clock is output on the RXRECCLK port. This clock runs at 1/20th the bit rate of the received signal. As shown in Figure 4-4, the RXRECCLK signal is typically connected to a BUFG or BUFR clock buffer. The output of the clock buffer drives the RXUSRCLK and RXUSRCLK2 ports

of the RX and also the logic downstream from the RXDATA port. Data is clocked out of the RX on the RXDATA port synchronously with the clock on the RXUSRCLK2 port.

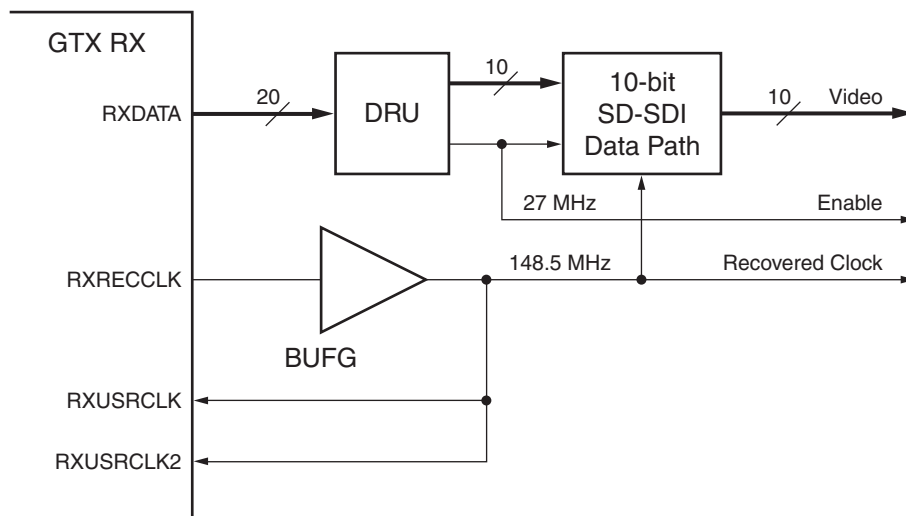
For SD-SDI and DVB-ASI, the CDR section is locked to the reference clock and RXRECCLK is 148.5 MHz. Figure 4-4 shows how the GTX RX is connected to the 20-bit HD-SDI datapath. 3G-SDI is implemented exactly the same way, except that the RXRECCLK has a frequency of 148.5 MHz instead of 74.25 MHz.



X1014_C4_23_080709

Figure 4-4: HD-SDI RX Clocking with 20-Bit RXDATA Port

Figure 4-5 shows how the GTX RX is connected to the 10-bit SD-SDI datapath. The GTX RX oversamples the 270 Mb/s SD-SDI bit stream by 11X. The oversampled data is output on the 20-bit RXDATA port to a DRU that examines the oversampled data and chooses the best samples to use for each bit. The DRU outputs the recovered data 10 bits at a time along with an enable signal that is asserted whenever a data word is ready on the DRU output port. This enable is asserted at a 27 MHz data rate. The SD-SDI RX datapath is 10 bits wide and runs at 27 MHz.



X1014_C4_24_080709

Figure 4-5: SD-SDI RX Clocking with 20-Bit RXDATA Port

Important RX Attributes

Table 4-3 shows GTX RX attributes that are important for SDI applications. Generally, the settings of these attributes are taken care of through the RocketIO Wizard. Those attributes that must be changed dynamically are controlled by the `v5gtx_sdi_control` module's DRP controller (although it is possible to override the DRP controller's default values for these attributes using parameters passed to the module). In many cases, there are two of each of these attributes, one for each receiver. In that case, "_0" or "_1" is appended to the end of the attribute name to indicate which RX unit it applies to. In some cases, however, a single attribute applies to both receivers. Refer to the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide* [Ref 3] for complete details about these attributes.

Table 4-3: Important RX Attributes

Attribute	Description
AC_CAP_DIS	TRUE: Built-in AC coupling capacitors are bypassed. FALSE: Built-in AC coupling capacitors are enabled. For SDI applications, always set this attribute to TRUE to bypass the capacitors.
TERMINATION_IMP	This attribute selects the termination impedance to either 50Ω or 75Ω. Although the SDI standards always use 75Ω cables, the RX termination impedance is usually set to 50Ω because an external cable equalizer is used to interface the RX inputs to the cable, and the interface between the cable equalizer and the RX inputs is usually 50Ω.
PMA_RX_CFG	This attribute selects the operating mode of the CDR. It is modified by the <code>v5gtx_sdi_control</code> module through the DRP when the receiver is switched between SDI modes.
PMA_CDR_SCAN	This attribute allows direct control of the CDR sampling points. It should always be left at the default value set by the RocketIO Wizard.

Table 4-3: Important RX Attributes (Cont'd)

Attribute	Description
PLL_RXDIVSEL_OUT	This attribute controls the division factor (1, 2, or 4) applied to the PLL clock to generate the RX serial clock. For SDI applications, this attribute is usually set to 2 for 3G-SDI, SD-SDI, and DVB-ASI, and to 4 for HD-SDI. The <code>v5gtx_sdi_control</code> module dynamically controls this attribute as required to switch the RX between standards.
RX_BUFFER_USE	TRUE: Use the RX elastic buffer. FALSE: Disable the RX elastic buffer.
OVERSAMPLE_MODE	This attribute is always set to FALSE for SDI applications. This is because SD-SDI and DVB-ASI do not use the built-in 5X oversampler.

The `PMA_RX_CFG` attribute controls the operation of the CDR unit. The value used for this attribute depends upon the operating mode and the setting of the `PLL_DIVSEL_OUT` attribute. The DRP controller in the `v5gtx_sdi_control` module sets this attribute to the correct value for each SDI mode.

RX Elastic Buffer

Each GTX receiver has a built-in elastic buffer used to resolve differences between the internal receiver clock domain and the FPGA logic interface clock domain. This buffer can be bypassed to reduce latency, but this requires the application to implement a phase alignment algorithm.

By disabling the buffer, the latency added by the buffer and the non-deterministic delay introduced by the elastic buffer are eliminated. However, disabling the elastic buffer requires the application to implement a very specific phase-alignment algorithm. This phase-alignment algorithm increases the recovery time of the receiver after operating mode changes. Thus, there is a trade-off between low latency deterministic operation with the buffer bypassed and quicker recovery after mode changes with the buffer enabled.

The *Virtex-5 FPGA RocketIO GTX Transceiver User Guide* [Ref 3] describes in detail the phase-alignment algorithm that must be implemented when the elastic buffer is bypassed.

Cable Equalizer

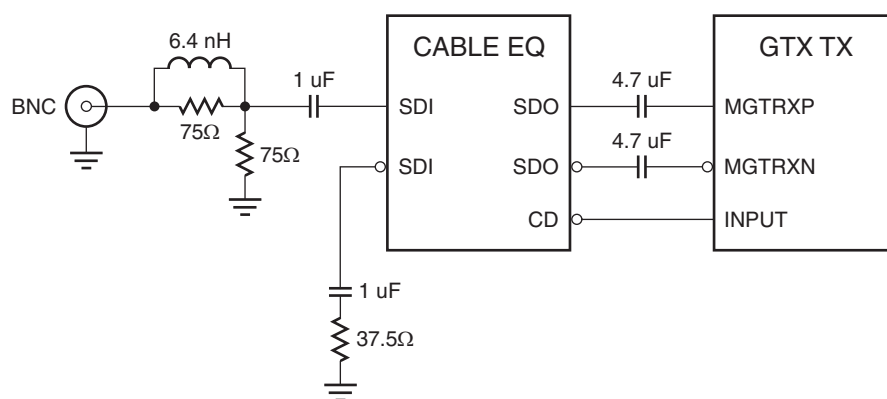
The SDI standards use a single-ended 75Ω cable that can typically be at least 300 meters long for SD-SDI and 100 meters for HD-SDI and 3G-SDI. Such long cable lengths require the use of a cable equalizer. While the GTX transceiver section has a built-in equalizer, this equalizer is not designed to equalize cables of the lengths used in SDI applications. Furthermore, the GTX receiver requires a differential input, so it cannot be directly connected to the SDI cable.

External SDI-specific cable equalizers are always recommended when implementing SDI applications with the GTX transceiver. These cable equalizers (available from a variety of vendors) are adaptive equalizers that support the cable lengths allowed by the SDI standards. They automatically determine the cable length and adapt the equalization to match. These cable equalizers also change the single-ended cable input to a differential signal that can be connected to the GTX receiver input.

Direct connection of the cable equalizer output to the GTX receiver input is usually not possible. The GTX receiver input is CML and the SDI cable equalizers typically have

LVPECL outputs. AC coupling is usually required between the cable equalizer and the GTX receiver inputs. The value of the AC coupling capacitors should be large enough to support the pathological patterns present in the SDI waveforms. For SDI applications, the AC coupling capacitors are typically in the 1 μF to 10 μF range.

Figure 4-6 shows the connection of an SDI cable equalizer between the BNC connector and the GTX RX inputs. Always refer to the recommendations of the cable equalizer manufacturer for details of the network between the BNC connector and the cable equalizer inputs.



X1014_C4_25_080709

Figure 4-6: Example Cable Equalizer Connections

Figure 4-6 shows a connection from the carrier detect output of the cable equalizer to an input of the FPGA. This is highly recommended.

For SDI applications, the CDR section of the GTX RX is usually configured to use the second-order loop filter to allow reception of the two HD-SDI bit rates (or the two 3G-SDI bit rates) with a single reference clock frequency. When using the second-order loop filter, the CDR section requires a reset whenever the input bit stream stops (called an electrical idle condition). Stoppage of the input bit stream can be caused by the cable being unplugged or the transmitter being turned off. If the CDR section is not reset after the signal is restored, the CDR might not lock properly to the input signal.

The GTX receiver section has a built-in electrical idle detection circuit that is normally used to initiate these electrical idle resets. The electrical idle circuit cannot, however, detect electrical idle conditions when an external cable equalizer is used between the transmitter and the receiver input. An alternative must be provided to reset the CDR section during electrical idle conditions. This alternative is provided by the carrier detect circuit included in the external SDI cable equalizer. It is, therefore, recommended that all SDI applications using the GTX transceiver connect the carrier detect output of each cable equalizer to an FPGA general-purpose input pin so that it can be used to reset the receiver's CDR section during electrical idle conditions.

GTX Transmitter

This section briefly describes how the GTX transmitter of the Virtex-5 FPGA is used to transmit 3G-SDI, HD-SDI, SD-SDI, and DVB-ASI. Refer to the individual SDI protocol chapters for details of transmitting each of these standards with the GTX transceivers.

Transmitting 270 Mb/s SD-SDI and DVB-ASI

The GTX transmitter directly supports transmission of data rates from 750 Mb/s up to 3.75 Gb/s. For data rates slower than 750 Mb/s, an “oversampling” technique is used where the transmitter is run at some multiple of the bit rate, and each bit is sent multiple consecutive times. For SD-SDI and DVB-ASI running at 270 Mb/s, it is best to use 11X oversampling, just as is done for the receiver. This allows the same reference clock frequency to be used to transmit SD-SDI and DVB-ASI as is used for HD-SDI and 3G-SDI. The transmitter runs at a bit rate of 2.97 Gb/s and each SD-SDI encoded bit is sent 11 consecutive times, resulting in a data rate of 270 Mb/s ($2.97 \text{ Gb/s} / 11 = 270 \text{ Mb/s}$).

To transmit at 270 Mb/s, the reference clock to the GTX transmitter must be 148.5 MHz (or 74.25 MHz). It is not possible to use 148.5/1.001 MHz (or 74.25/1.001 MHz) as the reference clock frequency when transmitting 270 Mb/s because the resulting bit rate would be 270/1.001 Mb/s.

The GTX transmitter does have a built-in 5X oversampling mode that can be used to support data rates slower than 750 Mb/s. However, it is not typically used for SD-SDI and DVB-ASI applications because it requires a different reference clock frequency than that needed for HD-SDI and 3G-SDI. The FPGA resources required to support 11X oversampling with the GTX transmitter are very small, so there is little reason to use the built in 5X oversampling mode.

Important TX Ports

Table 4-4 shows the important TX-specific ports used by SDI applications. There are other TX ports used for SDI applications, but they are typically controlled by the `v5gtx_sdi_control` module.

Each TX in the tile has an independent set of the ports shown in Table 4-4 (except for REFCLKOUT). Each signal name contains an italicized *n* that identifies to which TX unit (0 or 1) in the tile the signal belongs. The port names are also prefixed with the tile number, `TILEx`, where the *x* indicates which GTX_DUAL tile the port belongs to when the GTX wrapper file contains multiple GTX_DUAL tiles. For example, the TXUSRCLK2 port for TX0 of TILE0 is actually called `TILE0_TXUSRCLK20_OUT` and the port for TX1 of TILE0 is called `TILE0_TXUSRCLK21_OUT`.

Table 4-4: Important TX Ports

Port	I/O	Description
TILE _x _TXP _n _OUT TILE _x _TXN _n _OUT	In	This is the differential transmit serial data pair.
TILE _x _TXDATA _n _IN	Out	This is the transmit data bus of the TX interface to the FPGA logic. For SDI applications, this port can be either 10 or 20 bits wide. However, to minimize global clock resources, a 20-bit TXDATA port is generally preferred.
TILE _x _TXOUTCLK _n _OUT	Out	This is the transmitter clock used to drive TXUSRCLKs when the TX buffer is enabled.
TILE _x _REFCLKOUT_OUT	Out	This is a copy of the reference clock used to drive TXUSRCLKs when the TX buffer is bypassed. There is only one of these for each GTX_DUAL tile.

Table 4-4: Important TX Ports (Cont'd)

Port	I/O	Description
TILEx_TXUSRCLKn_IN	In	This is the input clock used for internal TX logic prior to the TX elastic buffer. This clock must be 1/20th the bit rate (148.5 MHz for 3G-SDI and 74.25 MHz for HD-SDI). For SD-SDI, it must be 148.5 MHz. This clock is usually derived directly from TXOUTCLK or REFCLKOUT (depending on whether the TX buffer is enabled), typically by simply buffering TXOUTCLK or REFCLKOUT through a BUFG or BUFR.
TILEx_TXUSRCLK2n_IN	In	This is the input clock used for the TX interface between the FPGA and the GTX transceiver. The frequency of this clock depends on the width of the TXDATA port. When TXDATA is 20 bits wide, this clock is equal in frequency to TXUSRCLK and, in fact, TXUSRCLK and TXUSRCLK2 are driven by exactly the same clock source.

The serial output of the GTX transmitter is a CML differential pair. The CML transmitter has built in termination that is calibrated to an external precision resistor. Typically, this is set to 50Ω, even though the SDI protocols use 75Ω cables. An external cable driver is required to convert the 50Ω differential CML GTX TX output to a 75Ω single-ended signal suitable for driving SDI cables.

TX Clocking

Like the RX section, each TX unit has two user clock inputs called TXUSRCLK and TXUSRCLK2. These are used to drive different portions of the interface between the GTX TX and the FPGA logic. The source of these clocks usually comes from either TXOUTCLK or REFCLKOUT, depending on whether the TX buffer is enabled or not. If the TX buffer is enabled, TXOUTCLK is used. If the TX buffer is bypassed, REFCLKOUT must be used.

When using TXOUTCLK to drive the TXUSRCLKs, it is always the correct frequency to drive TXUSRCLK. When the TXDATA port is 20 bits wide, it is also the correct frequency to drive TXUSRCLK2. Typically, TXOUTCLK is buffered by a BUFG or BUFR clock buffer and then used to drive the TXUSRCLKs and the FPGA resources used to implement the SDI datapath, as shown in [Figure 4-7](#).

When the TX buffer is bypassed to minimize latency, TXOUTCLK cannot be used as the clock source for the TXUSRCLKs. Instead, REFCLKOUT must be used. REFCLKOUT is always the same frequency as the reference clock selected to drive the tile's PLL. When the reference clock frequency is 148.5 MHz, REFCLKOUT is also 148.5 MHz. Thus, REFCLKOUT is not the correct frequency to drive the TXUSRCLKs for HD-SDI, but it is correct for 3G-SDI and SD-SDI. Refer to the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide* [Ref 3] for more information about bypassing the TX buffer.

To minimize clock resources required for SDI applications, all Virtex-5 FPGA GTX SDI reference designs in this application note use 20-bit TXDATA ports. [Figure 4-7](#) shows how the clocking works for HD-SDI. It works the same way for 3G-SDI, except that the TXOUTCLK frequency is 148.5 MHz instead of 74.25 MHz.

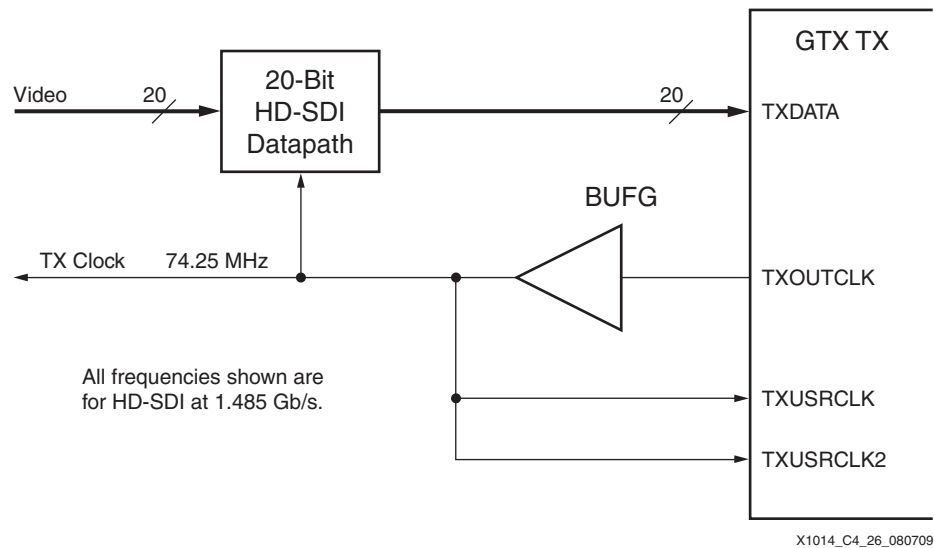


Figure 4-7: HD-SDI TX Clocking

Figure 4-8 shows the clocking for an SD-SDI transmitter. The frequency of TXOUTCLK is 148.5 MHz. The main SD-SDI datapath is 10 bits wide and runs at 27 MHz, enabled by a 27 MHz clock enable signal. Just prior to the GTX TXDATA port input, the data is replicated by 11X, making 11 bits for each bit to be sent. This data is sent 20 bits at a time at a 148.5 MHz rate (for a total data rate of 2.97 Gb/s) to the GTX TXDATA port. Each original bit is sent 11 consecutive times at the GTX transmitter's 2.97 Gb/s line rate, producing a 270 Mb/s SD-SDI data stream ($2.97 \text{ Gb/s} / 11 = 270 \text{ Mb/s}$).

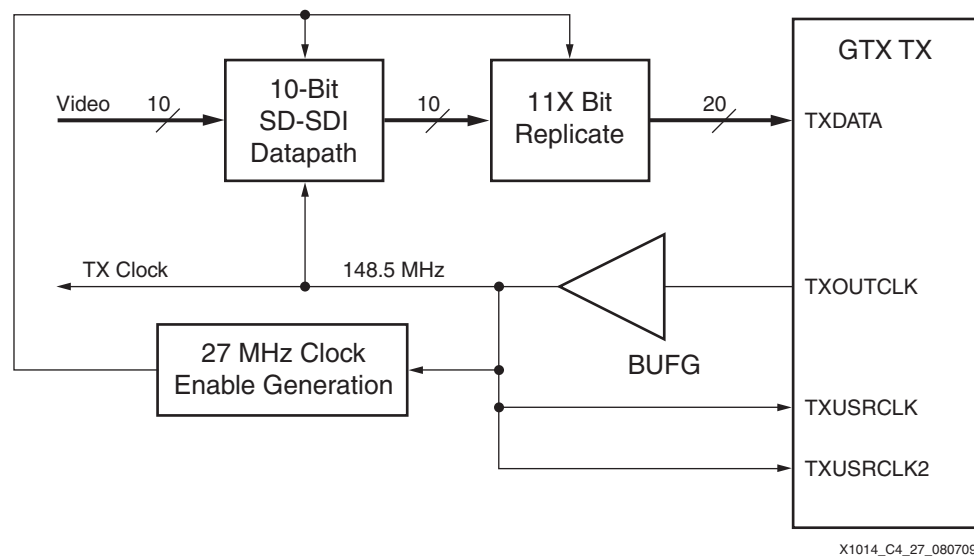
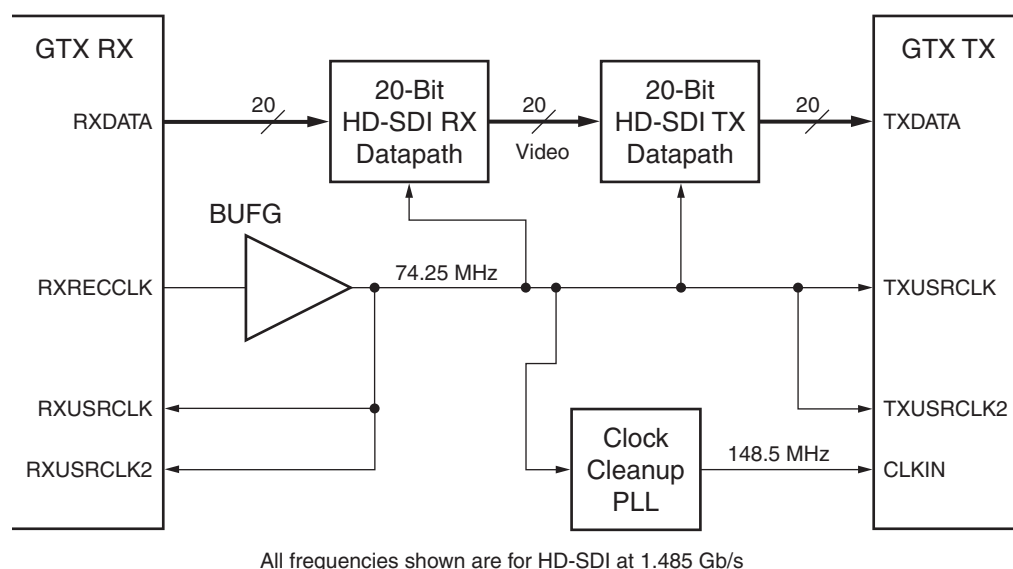


Figure 4-8: SD-SDI TX Clocking

It is also possible to drive the TXUSRCLKs from sources other than TXOUTCLK or REFCLKOUT. For example, in a pass-through HD-SDI application, it can be desirable to drive the TXUSRCLKs with the recovered clock from the receiver. This is possible with the GTX transmitter, but it is recommended that the TX buffer be used for such applications. It is essential that the transmitter's reference clock and the TXUSRCLKs are frequency

locked. Otherwise, the TX buffer quickly overruns or underruns. Thus, the reference clock of the GTX_DUAL tile containing the transmitter must be derived from the recovered clock of the receiver. This is typically done by sending the receiver's recovered clock through an external low-bandwidth PLL for jitter reduction and then connecting the output of the PLL to the reference clock input of the tile containing the transmitter, as shown in Figure 4-9. This example takes advantage of the TX buffer built into the GTX TX to compensate for phase differences or short-term frequency drift between the TXUSRCLKs and the TX reference clock from the clock cleaner.



X1014_C4_28_080709

Figure 4-9: HD-SDI TX Clocked by RXRECCLK

Important TX Attributes

Table 4-5 shows GTX TX attributes that are important for SDI applications. Generally, the settings of these attributes are taken care of through the RocketIO Wizard. Those attributes that must be changed dynamically are controlled by the `v5gtx_sdi_control` module's DRP controller (although it is possible to override the DRP controller's default values for these attributes using parameters passed to the module). In many cases, there are two of each of these attributes, one for each transmitter. In that case, the attribute name shown in the table has "_0" or "_1" appended to the end of the attribute name to indicate to which TX unit it applies. In some cases, however, a single attribute applies to both transmitters. Refer to the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide* [Ref 3] for complete details about these attributes.

Table 4-5: Important TX Attributes

Attribute	Description
PLL_TXDIVSEL_OUT	This attribute controls the division factor applied to the PLL output clock by the dividers that are unique to each of the TX units in the tile (there are two of these attributes for each tile). For 3G-SDI, SD-SDI, and DVB-ASI, this attribute is set to 1. For HD-SDI, this attribute is set to 2.

Table 4-5: Important TX Attributes (Cont'd)

Attribute	Description
TX_BUFFER_USE	TRUE: Use the TX elastic buffer. FALSE: Disable the TX elastic buffer.
OVERSAMPLE_MODE	This attribute is always set to FALSE for SDI applications. This is because SD-SDI and DVB-ASI do not use the built-in 5X oversampler.

TX Elastic Buffer

Each GTX transmitter has a built-in elastic buffer used to resolve differences between the internal transmitter clock domain and the FPGA logic interface clock domain. This buffer can be bypassed to reduce latency, but this requires the application to implement a phase-alignment algorithm. The TX buffer is quite shallow (only four words deep).

By disabling the buffer, the latency added by the buffer and the non-deterministic delay introduced by the elastic buffer are eliminated. However, disabling the elastic buffer requires the application to implement a very specific phase-alignment algorithm. This phase-alignment algorithm increases the recovery time of the transmitter after operating mode changes. Thus, there is a trade-off between low-latency deterministic operation with the buffer bypassed and quicker recovery after mode changes with the buffer enabled.

The *Virtex-5 FPGA RocketIO GTX Transceiver User Guide* [Ref 3] describes in detail the phase-alignment algorithm that must be implemented when the elastic buffer is bypassed.

If the TX buffer is bypassed, the application must use REFCLKOUT as the source of the TXUSRCLKs instead of TXOUTCLK. Because REFCLKOUT is not always the correct frequency to drive the TXUSRCLKs, a DCM or PLL is typically required to multiply REFCLKOUT to the correct frequency.

Cable Driver

An external cable driver, designed specifically for SDI applications, should be used to interface the GTX transmitter's differential CML output to a single-ended 75Ω signal suitable for driving the cable. Such cable drivers usually have LVPECL interfaces and cannot be directly coupled to the CML output of the GTX TX. AC coupling is typically used between the GTX TX output and the cable driver. The AC coupling capacitors are typically in the 1 μF to 10 μF range to pass the SDI pathological patterns.

SD-SDI has different slew rate requirements than HD-SDI and 3G-SDI. SDI cable drivers designed to support both SD-SDI and HD-SDI (or SD-SDI, HD-SDI, and 3G-SDI) usually have a slew rate control input. This input must be controlled appropriately based on the SDI mode selected. This is usually done by connecting the slew rate control pin of the cable driver to a general-purpose output of the FPGA. The `v5gtx_sdi_control` module has outputs designed specifically for controlling the slew rate control pins of SDI cable drivers.

Figure 4-10 shows a typical SDI cable driver connected to the outputs of the Virtex-5 FPGA GTX transmitter. Always follow the cable driver manufacturer's recommendations for interfacing the cable driver to the BNC cable connector.

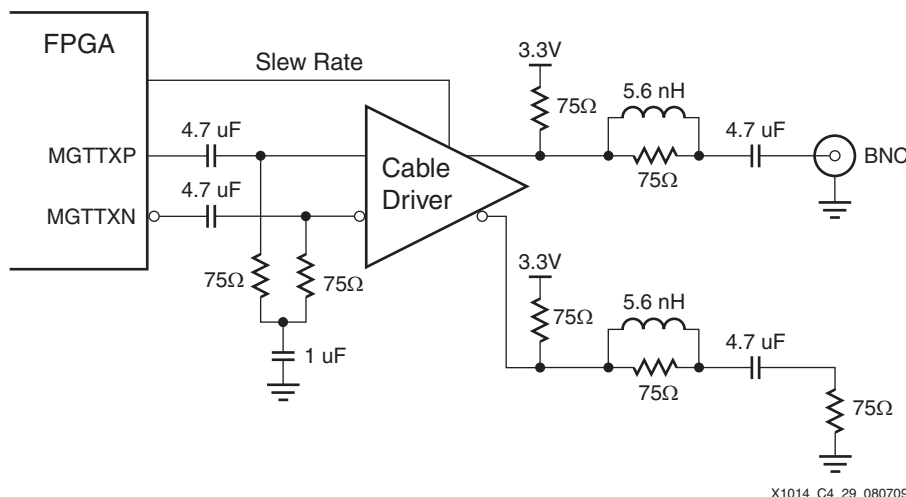


Figure 4-10: Example Cable Driver Connections

Control Module

When using the GTX transceiver of the Virtex-5 FPGA to implement SMPTE SDI standards, a number of functions must be implemented in the programmable logic of the FPGA to support the operation of the GTX transceiver. These functions are not the datapath logic for the SDI transmitters or receivers. Rather, they are support functions such as reset circuits that are required for proper operation of the GTX transceiver. Several of these functions have been gathered together into a module called the `v5gtx_sdi_control` module. This creates a more modular and manageable design. This section describes the `v5gtx_sdi_control` module and how it interacts with the GTX transceiver and the SDI datapath logic.

The `v5gtx_sdi_control` module works with the standard GTX transceiver design flow. The RocketIO GTX Transceiver Wizard is used to create a wrapper module containing one or more GTX_DUAL tiles. A `v5gtx_sdi_control` module is instantiated for each GTX_DUAL tile used for SDI protocols, providing a number of interrelated functions. Incorporating all these functions into one module hides much of the complexity of the relationships between them. The functions are:

- Reset logic for the GTX transceiver and SDI datapath
- DRP controller to control the GTX transceiver operating modes (SD-SDI, HD-SDI, and 3G-SDI) and reference clock selection and routing
- RX bit rate detection

Figure 4-11 is a diagram of how the `v5gtx_sdi_control` module and the GTX wrapper file interconnect. One `v5gtx_sdi_control` module is required for each GTX_DUAL tile used to implement SDI protocols, regardless of how many of the RX and TX units are actually used in that tile.

The GTX wrapper module is created by the RocketIO Wizard. Depending on the parameters given to the wizard, the wrapper module might contain more than one GTX_DUAL tile. In that case, multiple `v5gtx_sdi_control` modules are connected to the same GTX wrapper module, one for each GTX_DUAL tile that is used for SDI protocols. A single `v5gtx_sdi_control` module should not be used to control multiple

GTX_DUAL tiles. A different v5gtx_sdi_control module should be used for each GTX_DUAL tile.

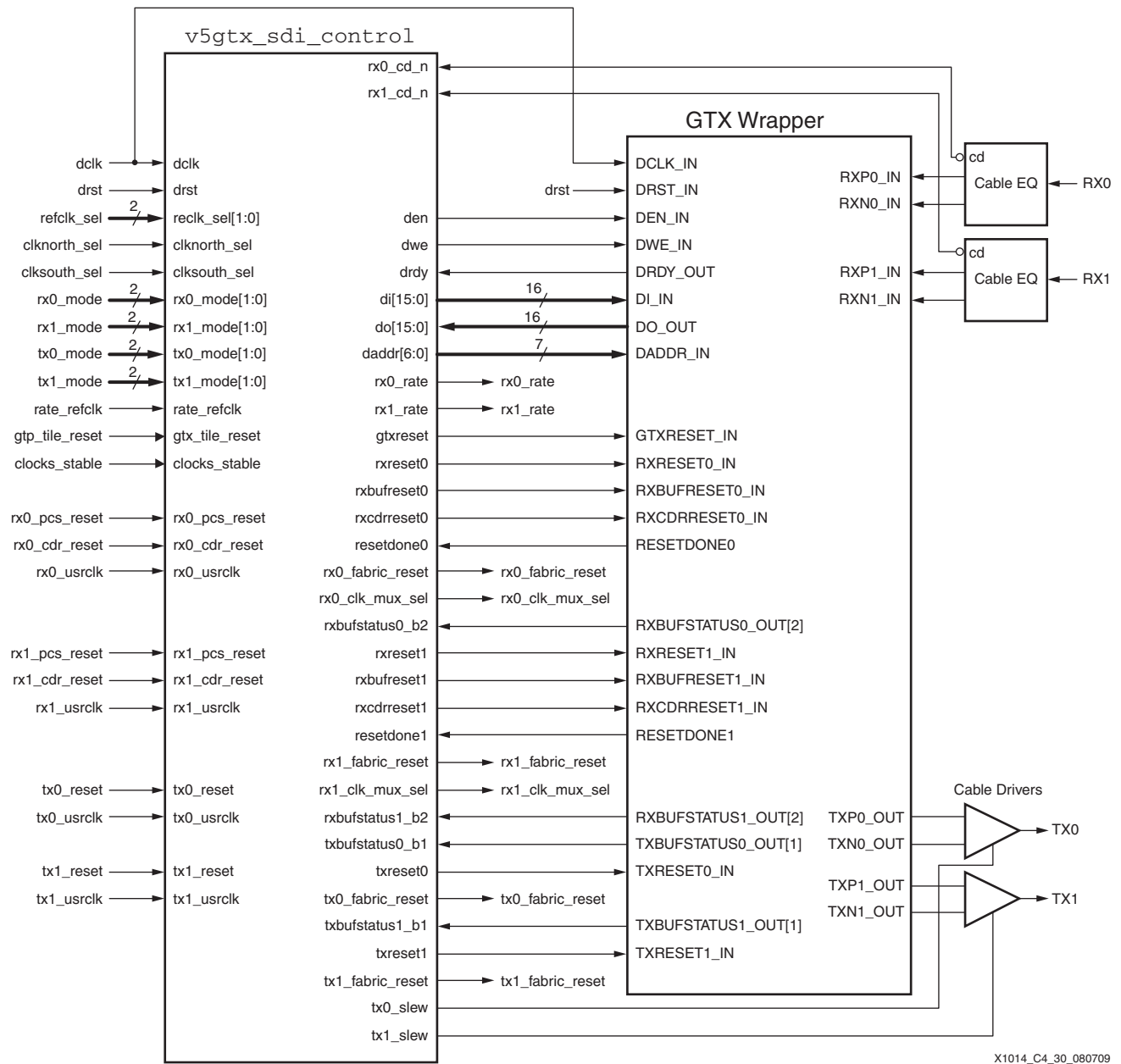


Figure 4-11: Control Module and GTX Wrapper Interconnections

Reset Logic

The GTX transceiver has several reset inputs that must be asserted under various conditions. This section describes the reset signals controlled by the `v5gtx_sdi_control` module.

GTXRESET

GTXRESET is the master reset for the entire tile, and it is the only way to reset the clock multiplication PLL. This PLL is common to all units in the GTX_DUAL tile. The PLL must be reset after the reference clock becomes stable. If the reference clock is stopped and then restarted, the PLL should also be reset.

The `v5gtx_sdi_control` module has a `gtxreset` output that drives the GTXRESET_IN input of the GTX wrapper to issue a GTXRESET to the GTX_DUAL tile. The module asserts `gtxreset` under three conditions:

- Immediately after the FPGA emerges from configuration
- When the module's `gtx_tile_reset` input is High
- When the module's `clocks_stable` input is Low

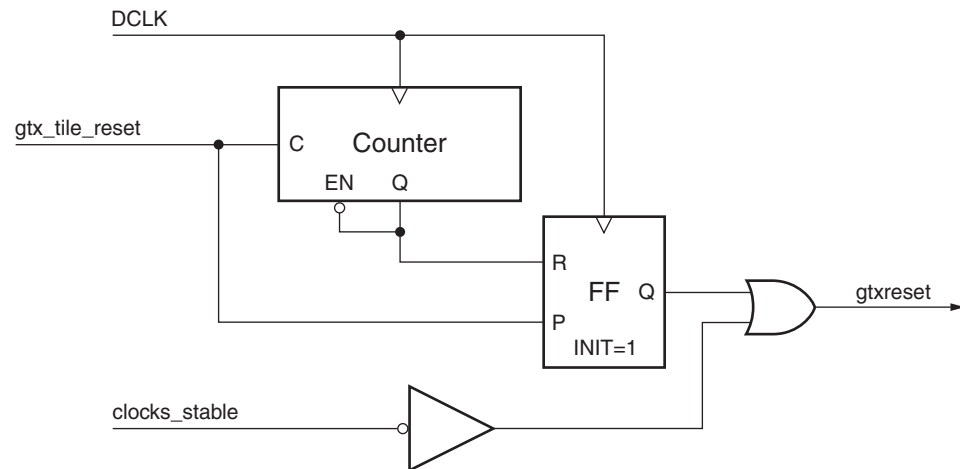
The `gtx_tile_reset` and `clocks_stable` inputs provide two different ways to reset the GTX transceiver's PLL and the rest of the GTX_DUAL tile after the reference clock is started or changes dramatically. Most applications only use one of these inputs to the `v5sdi_gtx_control` module, not both, depending on application-specific requirements.

The `gtx_tile_reset` input generates a long-duration GTXRESET pulse to the GTX_DUAL tile. The length of this pulse is controlled by the frequency of the `dclk` input (DRP clock) and the parameter/generic `GTX_RESET_CNTR_SIZE`. Asynchronously, on the rising edge of `gtx_tile_reset`, a counter is reset and the `gtxreset` output goes High, resetting the GTX_DUAL tile. As long as the `gtx_tile_reset` input is High, the `gtxreset` output remains High and the counter stays at zero. After `gtx_tile_reset` goes Low, the counter begins counting. While the counter is counting, `gtxreset` remains High. After the counter reaches its terminal count, `gtxreset` goes Low. The terminal count occurs as soon as the most significant bit of the counter becomes a 1. Equation 4-1 gives the duration of `gtxreset`.

$$gtxreset\ duration = gtx_tile_reset\ duration + \frac{2^{GTX_RESET_CNTR_SIZE} - 1}{dclk\ frequency} \quad \text{Equation 4-1}$$

Figure 4-12 shows the logic used to generate the GTXRESET. In all figures in this chapter, flip-flops use the same nomenclature as that used by the Xilinx libraries guide.

Asynchronous preset and clear inputs are labeled P and C, respectively. Synchronous set and reset are labeled S and R, respectively. The same nomenclature is also used on the counter in Figure 4-12, where the `gtx_tile_reset` input asynchronously clears the counter and presets the flip-flop.



X1014_C4_31_080709

Figure 4-12: GTXRESET Logic

The `gtx_tile_reset` and its associated counter are designed for applications that have no inherent method of determining when the reference clock supplied to the GTX_DUAL tile is stable, such as a locked signal from a PLL. In such applications, the `gtxreset` duration must be set to be longer than the worst-case start-up time of the GTX transceiver reference clock.

The `gtxreset` output is asserted immediately after the FPGA comes out of configuration because the INIT value for the flip-flop is 1. It remains asserted for the duration of the counter period described in Equation 4-1 even if `gtx_tile_reset` is not asserted after FPGA configuration. This provides an automatic GTXRESET to the tile upon completion of FPGA configuration to allow the reference clock to stabilize.

In some applications, the reference clock source can also supply a signal that indicates when the reference clock is stable. This signal can be used to drive the `clocks_stable` input of the `v5gtx_sdi_control` module. When this signal is Low, the `gtxreset` output is asserted High and remains High until `clocks_stable` goes High. The timer circuit associated with the `gtx_tile_reset` signal is completely bypassed. Thus, the duration of the `gtxreset` signal is equal to the duration that `clocks_stable` is Low.

If the `clocks_stable` input is used and the `gtx_tile_reset` input is not, the counter circuit is still used to assert the `gtxreset` output for the duration of the counter period after FPGA configuration. If the `clock_stable` signal also safely controls the `gtxreset` after FPGA configuration, the counter size can be set to something small. However, the counter size must never be set to zero.

If the `clocks_stable` input is not used, it must be tied High. If the `gtx_tile_reset` input is not used, it must be tied Low. Except on a clean transition between the two SDI reference clock frequencies, applications must use one or the other of these signals to reset the GTX_DUAL tile if the reference clock stops or changes frequency.

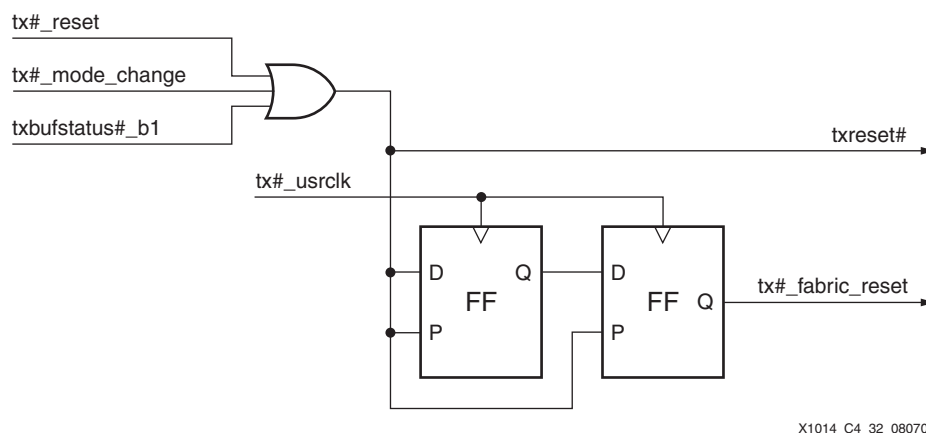
Transmitter Resets

The GTX_DUAL tile has a reset input for each transmitter unit. TXRESET0 resets the PCS section of TX0. TXRESET1 resets the PCS section of TX1. The `v5gtx_sdi_control` module controls these two GTX transmitter reset signals with its `txreset0` and `txreset1` outputs.

According to the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide* [Ref 3], there are two conditions when the TXRESET signals should be asserted:

- When there are disruptions in the TXUSRCLK clocks
- When the TX buffer (if used) underflows or overflows

The `v5gtx_sdi_control` module allows the TXRESET signals to be asserted under both conditions. It also automatically asserts TXRESET for a short period of time if the transmitter mode is changed (between SD-SDI, HD-SDI, or 3G-SDI modes). The TX reset logic is shown in Figure 4-13.



X1014_C4_32_080709

Figure 4-13: TX Reset Logic

To handle TXUSRCLK disruptions, the `v5gtx_sdi_control` module has inputs called `tx0_reset` and `tx1_reset`. These inputs control the `txreset0` and `txreset1` outputs, respectively. In this discussion, the signals for either transmitter are generically called `tx#_reset` for the input signal and `txreset#` for the output signal, where # can be 0 or 1. Thus, the `tx0_reset` and `txreset0` signals are used for TX0 of the GTX_DUAL tile and `tx1_reset` and `txreset1` are used for TX1 of the GTX_DUAL tile.

As long as the `tx#_reset` input is High, the `txreset#` output is asserted to the GTX transceiver. If there is any possibility that the TXUSRCLKs might be disrupted, the application must control the `tx#_reset` appropriately to issue a `txreset#` after the TXUSRCLKs have been restored.

Disruption of the TXUSRCLKs can occur for application-specific reasons. For example, if a DCM or PLL is used to create the TXUSRCLKs, then the `tx#_reset` input of the `v5gtx_sdi_module` can be driven by the inverted LOCKED output of the DCM or PLL.

The `txreset#` output is also asserted if the TX buffer overflows, as indicated by bit 1 of the `TXBUFSTATUS#_OUT` bit of the GTX_DUAL tile (connected to the `txbufstatus#_b1` input of the `v5gtx_sdi_control` module). It is also asserted when the transmitter operating mode is changed between SDI protocols (3G-SDI, HD-SDI, or SD-SDI). The DRP controller, located inside the `v5gtx_sdi_control` module switches the modes of the transmitter in response to mode select inputs and asserts the `tx#_mode_change` signal when the transmitter's mode has been changed. Thus, the `tx#_mode_change` signal is an internal signal, not an input to the module.

The transmit reset logic also synchronizes the `txreset#` signal to `tx#_usrclk` to produce the `tx#_fabric_reset` output. This output is asserted asynchronously, as soon as `txreset#` is asserted, but is negated synchronously with the rising edge of `tx#_usrclk`. The `tx#_fabric_reset` signal can be used to reset the FPGA logic of the SDI transmitter, if desired.

Because this output is negated synchronously with the rising edge of the tx#_usrclk, with proper timing constraints, this signal can be used with both synchronous and asynchronous resets of the various SDI datapath modules.

Receiver Resets

The GTX receiver has several different resets that must be asserted under different conditions. Each receiver in the tile has its own set of reset inputs. The three reset signals controlled by the v5gtx_sdi_control module are:

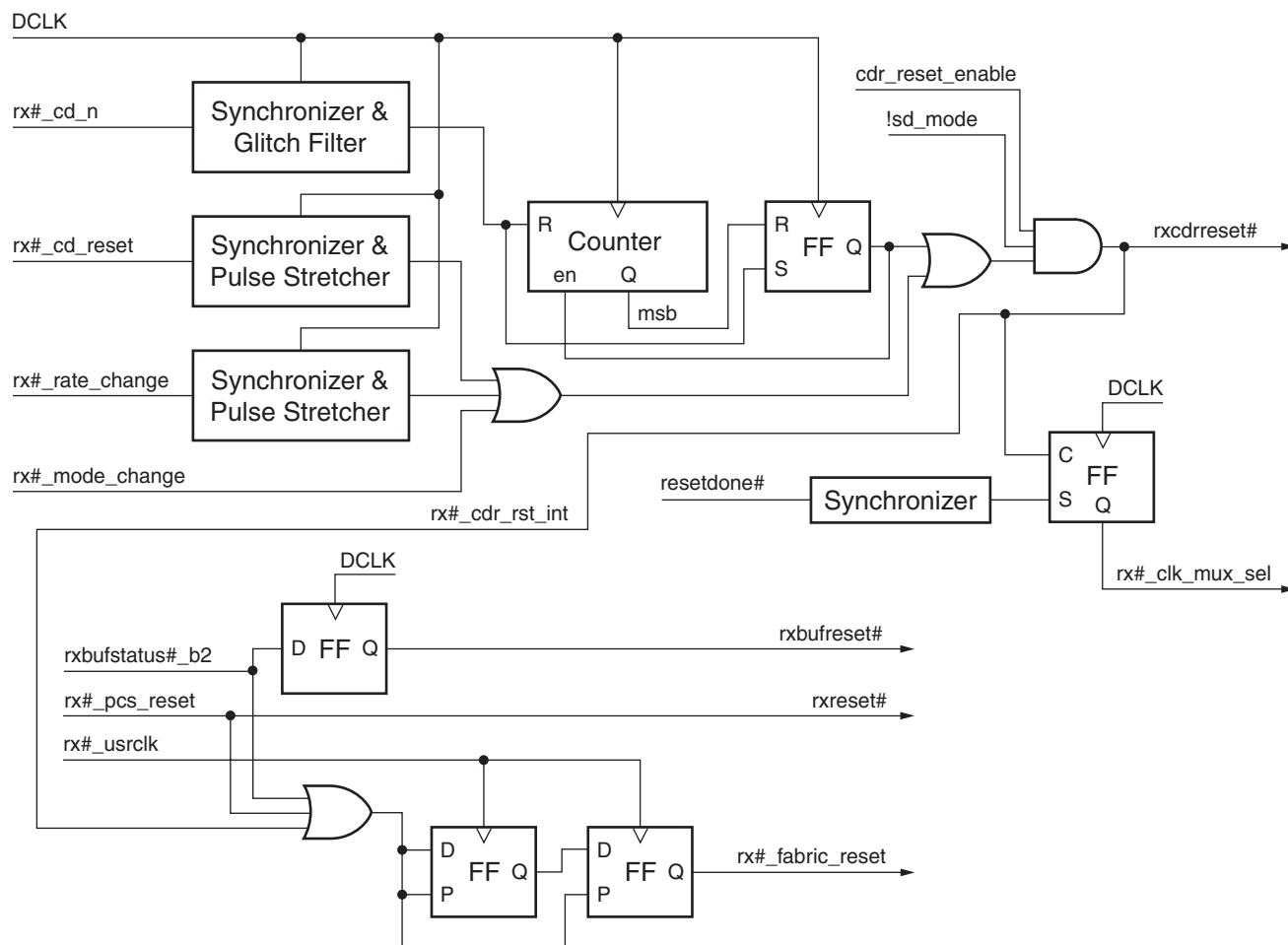
- **RXCDRRESET:** This is also known as the CDR reset. It resets both the CDR and the PCS sections of the receiver. Thus, it resets the entire receiver, except for the clock multiplication PLL common to all units in the tile.
- **RXRESET:** This input resets the PCS section of the receiver and does not affect the CDR section.
- **RXBUFRESET:** This input resets just the RX buffer. The buffer is also reset by RXRESET and RXCDRRESET, but if only the buffer needs to be reset, RXBUFRESET can be used.

The *Virtex-5 FPGA RocketIO GTX Transceiver User Guide* [Ref 3] states that a CDR reset should be asserted if there is an interruption in the incoming bitstream. The CDR reset must be asserted after the input signal is restored. In SDI, the input signal is interrupted if the transmitter is halted for some reason or if the cable is disconnected.

The *Virtex-5 FPGA RocketIO GTX Transceiver User Guide* states that RXRESET should be used when there is an interruption in the RXUSRCLKs. After the RXUSRCLKs have been restored and are stable, RXRESET can be safely negated.

RXBUFRESET can be used to reset just the RX buffer, if it is used. The RX buffer should be reset if there is a buffer underflow or overflow. When bit two of the RXBUFSTATUS output of the receiver is 1, an RX buffer error has occurred and RXBUFRESET is used to reset the buffer.

Figure 4-14 shows the logic in the v5gtx_sdi_control module that controls the various RX resets. There are two sets of this logic, one for each receiver in the GTX_DUAL tile.



X1014_C4_33_080709

Figure 4-14: RX Reset Logic

The rxcdrreset# output is asserted to issue a CDR reset to the RX. This output can be triggered by four different signals:

- A High level on the active-Low carrier detect signal from the external cable equalizer indicating a loss of input signal, either because the cable has been disconnected or the transmitter has been switched off. It is asserted Low when a valid signal is detected and deasserted High when a valid signal is not detected.
- Assertion of the rx#_cd_rst input to the v5gtx_sdi_control module. This can be used by the application if it needs to reset the entire receiver.
- Assertion of the rx#_rate_change signal from the rate detector. The rate detector asserts this signal if the input bit rate changes. This signal is also asserted if the recovered clock from the receiver is out of range of a valid SDI bit rate, indicating that the CDR section is not locked to the input bitstream. This is an internal signal in the module.
- Assertion of the rx#_mode_change signal from the DRP controller. This signal is asserted when the mode of the receiver is changed between HD-SDI, SD-SDI, and 3G-SDI. This is an internal signal in the module.

The CDR reset is disabled during GTXRESETS because CDR resets are not allowed during GTXRESETS. CDR resets are also disabled when the receiver is in SD-SDI mode because

CDR resets interfere with the operation of the receiver when it is locked to the reference clock, as it is during SD-SDI mode.

The carrier detect signal from the cable equalizer tends to be noisy and is intended to be used to detect events that are very long in duration. This signal is first processed by a glitch filter and synchronized to the dclk before being applied to the CDR reset logic. Because this signal also tends to bounce for a long period of time on cable disconnects and reconnects, it is used to trigger a long-duration CDR reset pulse (on the order of 60 μ s). The duration of this reset signal is determined by the width of the counter, which is set by the CBL_DISCONNECT_RST_CNTR_SIZE parameter/generic and the frequency of dclk, as shown in Equation 4-2.

$$CD \text{ reset duration} = \frac{2^{CBL_DISCONNECT_RESET_CNTR_SIZE - 1}}{dclk \text{ frequency}} \quad \text{Equation 4-2}$$

The rx#_cdr_reset input and rx#_rate_change internal signal are asynchronous to dclk and can have pulses that are shorter in duration than the dclk period. Thus, both signals are synchronized to dclk, and short pulses are detected and stretched out to be at least one dclk cycle long.

Assertion of any of these four signals asserts the rxcdrreset# output. It also causes the rx#_clk_mux_sel signal to go Low. This signal is used to control a global clock multiplexer used to drive the RXUSRCLKs. During CDR resets, the recovered clock from the GTX RX can stop. In many applications, it is desirable to continue to clock the SDI datapath logic, even when no valid data is being received. The global clock multiplexer can select a free-running reference clock to drive the RXUSRCLKs in the absence of a recovered clock during CDR resets. Upon completion of the GTX receiver's internal CDR reset cycle, indicated by the GTX receiver output RESETDONE going High, the rx#_clk_mux_sel goes High again, causing the global clock multiplexer to switch back to RXRECCLK.

Note: Each GTX_DUAL tile has two RESETDONE outputs, one for each RX/TX pair. RESETDONE does not go High until both the RX and the TX in the pair have finished their reset cycles. Furthermore, the reset cycle cannot be completed if the reference clock and the user clocks into the GTX transceiver are halted. Thus, in cases where only the RX of a RX/TX pair is used, it is essential that the TXUSRCLK and TXUSRCLK2 of the TX unit be driven by live clocks and not tied to ground. Otherwise, the RESETDONE signal never goes High and the rx#_clk_mux_sel never selects the RXRECCLK as the source of the RX user clock.

Asserting RXCDRRESET to the GTX RX also resets the PCS section and the RX buffer. Thus, it is not necessary to assert either RXRESET or RXBUFRESET when a RXCDRRESET occurs.

The rxreset# output, which drives the receiver's RXRESET input, is only asserted when the rx#_pcs_reset input goes High. The rx#_pcs_reset can be asserted by the application to reset the PCS section of the GTX RX without resetting the CDR section. The only reason for doing this is to reset the PCS section after a disruption of the RXUSRCLKs. This primarily occurs if the RXUSRCLKs are being driven by a DCM or PLL. In this case, the LOCKED output of the DCM or PLL can be inverted and connected to the rx#_pcs_reset input of the v5gtx_sdi_control module to reset the PCS section of the receiver.

The rxbufreset_# output drives the RXBUFRESET input of the GTX RX to reset the RX buffer in case of an RX buffer error. It is driven by bit two of the RXBUFSTATUS output of the RX. This signal goes through a flip-flop to make timing easier to achieve.

In the event of a CDR reset, or an RXRESET or RXBUFRESET assertion, the v5gtx_sdi_control module also asserts an output called rx#_fabric_reset. This signal is asynchronously asserted as soon as any of the reset conditions become asserted. It is negated synchronously with the rising edge of rx#_usrclk. With proper timing constraints,

this signal can be used with both synchronous and asynchronous resets of SDI datapath modules.

DRP Control

The `v5gtx_sdi_control` module contains a DRP controller that dynamically controls the operating mode of the GTX transceiver. The DRP controller in the module provides these capabilities:

- Independently sets the operating mode of each RX and TX. Each RX and TX can be set to SD-SDI mode, HD-SDI mode, or 3G-SDI mode. Due to the shared clock multiplication PLL, there are limitations on using both transmitters in the same GTX_DUAL tile.
- Selects the reference clock input to the GTX_DUAL tile's clock multiplication PLL. This allows the transmitters in the tile to be switched between the two HD-SDI (and 3G-SDI) bit rates.
- Selects the source of the CLKOUTNORTH and CLKOUTSOUTH reference clock outputs of the GTX_DUAL tile to the neighboring GTX_DUAL tiles.

Mode Selection

Each RX unit and each TX unit can be independently set to 3G-SDI, HD-SDI, or SD-SDI mode. For each RX and TX, there is a 2-bit input port to the `v5gtx_sdi_control` module that selects the operating mode for the unit, as shown in [Table 4-6](#).

Table 4-6: Mode Selection

Mode Select	Mode
00	HD-SDI
01	SD-SDI
10	3G-SDI
11	Invalid

Whenever the transmitter mode is changed, the transmitter PCS section resets by assertion of the TXRESET signal. Whenever the mode of a receiver is changed, the entire receiver is reset by assertion of the RXCDRRESET signal.

When the FPGA comes out of configuration, the DRP controller examines the mode select signals and sets each unit to the mode specified. After that, the controller only changes the mode of a unit when the mode select inputs change. Assertion of the `drst` input to the `v5gtx_sdi_control` module also causes the DRP controller to reinitialize the GTX_DUAL tile as soon as `drst` is negated.

Reference Clock Selection & Routing

To support transmission of the two HD-SDI bit rates of 1.485 Gb/s and 1.485/1.001 Gb/s, the reference clock must be switched between 148.5 MHz and 148.5/1.001 MHz. This can be done externally to the FPGA by using a clock multiplexer or a clock source that can be switched between the two reference clock frequencies. It can also be done by changing the clock select multiplexer in the GTX_DUAL tile between the reference clock sources available to it.

The reference clock multiplexer is a 4:1 multiplexer. The inputs are the external differential CLKP/N pair associated with that GTX_DUAL tile, CLKINNORTH from the tile below, CLKINSOUTH from the tile above, and GREFCLK from a global clock net. The clock is selected by refclk_sel port, as shown in [Table 4-7](#).

Table 4-7: Reference Clock Selection

refclk_sel	Clock Selected
00	External CLKP/N input pair
01	GREFCLK
10	CLKINSOUTH
11	CLKINNORTH

[Figure 4-2, page 93](#) shows the reference clock selection and routing multiplexers inside the GTX_DUAL tile. Every GTX_DUAL tile has a dedicated external differential reference clock input signal shown in the figure as CLKP/N. This reference clock is one input to the tile's reference clock select multiplexer. The clock signal from CLKP/N can also be provided to the GTX_DUAL tile above (on the CLKOUTNORTH port) and the GTX_DUAL tile below (on the CLKOUTSOUTH port). The CLKINSOUTH input port is a reference clock coming from the GTX_DUAL tile above. This signal is connected to an input of the reference clock multiplexer. It also can drive the CLKOUTSOUTH port to the GTX_DUAL tile below. Likewise, the CLKINNORTH input port is a reference clock coming from the GTX_DUAL tile below. It is connected to the reference clock multiplexer and can drive the CLKOUTNORTH port to the GTX_DUAL tile above.

The DRP controller can change the settings of the REFCLK_SEL attribute to change the reference clock selection multiplexer for the GTX_DUAL tile, as previously described. It can also change the value of the CLKSOUTH_SEL and CLKNORTH_SEL attributes to change what is routed to the CLKOUTSOUTH and CLKOUTNORTH ports, respectively. This allows the application to specify and dynamically change the routing of the reference clocks through the tile to the neighboring GTX_DUAL tiles.

Rate Detection

The third major function of the v5gtx_sdi_control module is rate detection. It is often necessary for an HD-SDI interface to know whether it is receiving a bit stream with a rate of 1.485 Gb/s or 1.485/1.001 Gb/s, and also to know which of the two 3G-SDI bit rates are being received.

The rate detection logic distinguishes between these bit rates by comparing the frequency of RXUSRCLK to a known standard reference clock frequency. In addition to the ability to determine which of the two bit rates is being received, the rate detection logic can also determine if RXUSRCLK is not a match to a valid SDI bit rate, indicating that the input bit stream has stopped or that the CDR of the receiver has lost lock with the input bit stream. Thus, the rate detection circuit is an essential part of the reset logic for the GTX receiver.

The rate detection circuit requires a constant frequency reference clock to compare against the frequency of the RXUSRCLK. This reference clock can be any frequency from 27 MHz up to at least 148.5 MHz. The frequency of the reference clock must be specified using a parameter/generic supplied to the v5gtx_sdi_control module called RATE_REFCLK_FREQ. To specify a 27 MHz reference clock frequency, for example, this parameter is set to **27000000**. The reference clock must be a fixed frequency that does not change, even when the SDI mode changes. It does not have to be a frequency that is in any way related to the video sample rate. It is simply a fixed-frequency reference clock.

For each receiver, the rate detection logic generates an output indicating which bit rate is being received. These outputs are called rx0_rate and rx1_rate. The value of each output is dependent upon the current operating mode of the receiver, as shown in Table 4-8. When running in SD-SDI mode, the rate output should be ignored. In SD-SDI mode, the recovered clock from the GTX RX is locked to the reference clock. Thus, for SD-SDI mode, the rx#_rate output can be either High or Low.

Table 4-8: Rate Detection Output

Mode	rx#_rate	Bit Rate
SD-SDI	X	270 Mb/s
HD-SDI	0	1.485 Gb/s
	1	1.485/1.001 Gb/s
3G-SDI	0	2.97 Gb/s
	1	2.97/1.001 Gb/s

The v5gtx_sdi_control module does not output a signal that specifically indicates when the rate detection logic has determined that the RXUSRCLK frequency is invalid. But, if RXUSRCLK remains at an invalid frequency for a period of time that exceeds an error threshold in the rate detector, a CDR reset is automatically initiated.

Connecting the v5gtx_sdi_control Module

Table 4-9 lists the ports of the v5gtx_sdi_control module.

Table 4-9: v5gtx_sdi_control Module Ports

Port Name	I/O	Width	Description
Clocks and DRP Reset			
rate_refclk	In	1	This is used as a comparison frequency by the rate detector. It must be driven by a stable, constant frequency reference clock. The minimum frequency is 27 MHz. Any higher frequency can be used, up to the point where timing cannot be met. This has been tested up to 148.5 MHz in -1 speed grade Virtex-5 devices. This reference clock frequency does not have to be related, in any way, to the video clock frequency. It must, however, be a constant frequency, never changing even when the SDI mode changes. It can be difficult to meet timing with faster rate_refclk frequencies. It is recommended, therefore, that the frequency of rate_refclk be less than 50 MHz.
rx0_usrclk	In	1	This input connects to the global or regional clock that is also connected to the RXUSRCLK input of RX0 of the GTX_DUAL tile.
rx1_usrclk	In	1	This input connects to the global or regional clock that is also connected to the RXUSRCLK input of RX1 of the GTX_DUAL tile.
tx0_usrclk	In	1	This input connects to the global or regional clock that is also connected to the TXUSRCLK input of TX0 of the GTX_DUAL tile.

Table 4-9: v5gtx_sdi_control Module Ports (Cont'd)

Port Name	I/O	Width	Description
tx1_usrclk	In	1	This input connects to the global or regional clock that is also connected to the TXUSRCLK input of TX1 of the GTX_DUAL tile.
Mode Selection, Reference Clock Selection, and Routing			
refclk_sel	In	2	This selects the reference clock used by the tile. See Table 4-7, page 117 for details.
clknorth_sel	In	1	This controls the multiplexer that drives the CLKOUTNORTH port of the GTX_DUAL tile. If this input is zero, the clock from the CLKINNORTH port is passed through to CLKOUTNORTH. If this input is one, the clock from the tile's external CLKP/N input is selected for CLKOUTNORTH.
clksouth_sel	In	1	This controls the multiplexer that drives the CLKOUTSOUTH port of the GTX_DUAL tile. If this input is zero, the clock from the CLKINSOUTH port is passed through to CLKOUTSOUTH. If this input is one, the clock from the tile's external CLKP/N input is selected for CLKOUTSOUTH.
rx0_mode	In	2	This selects the operating mode for RX0. Refer to Table 4-6, page 116 for details.
rx1_mode	In	2	This selects the operating mode for RX1. Refer to Table 4-6, page 116 for details.
tx0_mode	In	2	This selects the operating mode for TX0. Refer to Table 4-6, page 116 for details.
tx1_mode	In	2	This selects the operating mode for TX1. Refer to Table 4-6, page 116 for details.
Cable Driver Slew Rate Control			
tx0_slew	Out	1	This is the slew rate control for the cable driver for TX0. This output is High for SD-SDI and Low for HD-SDI and 3G-SDI.
tx1_slew	Out	1	This is the slew rate control for the cable driver for TX1. This output is High for SD-SDI and Low for HD-SDI and 3G-SDI.
Receiver Rate Detection			
rx0_rate	Out	1	This indicates the bit rate being received by RX0. Refer to Table 4-8, page 118 for details.
rx1_rate	Out	1	This indicates the bit rate being received by RX1. Refer to Table 4-8, page 118 for details.
GTXRESET			
gtx_tile_reset	In	1	This input, when pulsed High, initiates a long-duration GTXRESET pulse to the tile. The duration of the pulse is controlled by the GTX reset counter. If unused, this input must be Low.

Table 4-9: v5gtx_sdi_control Module Ports (Cont'd)

Port Name	I/O	Width	Description
clocks_stable	In	1	This input can be used to generate a GTXRESET if the reference clock is not stable. The duration of the GTXRESET pulse to the GTX_DUAL tile is equal to the duration that clocks_stable is Low. If unused, this input must be High.
gtxreset	Out	1	This connects to the GTXRESET input port of the GTX_DUAL tile.
RX0 Reset Signals			
rx0_cd_n	In	1	This connects to the carrier detect signal from the cable equalizer of RX0. This input must be Low if a good carrier signal is detected, and High if it is not.
rx0_pcs_reset	In	1	The application can drive this input High to initiate a reset of the PCS section of RX0.
rx0_cdr_reset	In	1	The application can drive this input High to initiate a reset of the CDR and PCS sections of RX0.
resetdone0	In	1	This input must be driven by the RESETDONE0 output of the GTX_DUAL tile.
rxbufstatus0_b2	In	1	This input must be driven by bit 2 of the RXBUFSTATUS0 port of the GTX_DUAL tile.
rx0_clk_mux_sel	Out	1	This output is used to control the global clock buffer driving RXUSRCLK for RX0. When this output is High, RXRECCLK should be selected. When this output is Low, a reference clock should be selected.
rx0_fabric_reset	Out	1	This output can be used to reset the FPGA logic associated with RX0.
rxreset0	Out	1	This output should be connected to the RXRESET0 port of the GTX_DUAL tile.
rxbufreset0	Out	1	This output connects to the RXBUFRESET0 port of the GTX_DUAL tile.
rxcdrreset0	Out	1	This output connects to the RXCDRRESET0 port of the GTX_DUAL tile.
RX1 Reset Signals			
rx1_cd_n	In	1	This input connects to the carrier detect signal from the cable equalizer of RX1. This input must be Low if a good carrier signal is detected, and High if it is not.
rx1_pcs_reset	In	1	The application can drive this input High to initiate a reset of the PCS section of RX1.
rx1_cdr_reset	In	1	The application can drive this input High to initiate a reset of the CDR and PCS sections of RX1.
resetdone1	In	1	This input must be driven by the RESETDONE1 output of the GTX_DUAL tile.

Table 4-9: v5gtx_sdi_control Module Ports (Cont'd)

Port Name	I/O	Width	Description
rxbufstatus1_b2	In	1	This input must be driven by bit 2 of the RXBUFSTATUS1 port of the GTX_DUAL tile.
rx1_clk_mux_sel	Out	1	This output is used to control the global clock buffer driving RXUSRCLK for RX1. When this output is High, RXRECCLK should be selected. When this output is Low, a reference clock should be selected.
rx1_fabric_reset	Out	1	This output can be used to reset the FPGA logic associated with RX1.
rxreset1	Out	1	This output connects to the RXRESET1 port of the GTX_DUAL tile.
rxbufreset1	Out	1	This output connects to the RXBUFRESET1 port of the GTX_DUAL tile.
rxcdrreset1	Out	1	This output connects to the RXCDRRESET1 port of the GTX_DUAL tile.
TX0 Reset Signals			
tx0_reset	In	1	The application can drive this input High to initiate a PCS reset of TX0.
txbufstatus0_b1	In	1	This input connects to bit 1 of the TXBUFSTATUS0 port of the GTX_DUAL tile.
tx0_fabric_reset	Out	1	This output can be used to reset the datapath of the SDI transmitter TX0.
txreset0	Out	1	This output connects to the TXRESET0 port of the GTX_DUAL tile.
TX1 Reset Signals			
tx1_reset	In	1	The application can drive this input High to initiate a PCS reset of TX1.
txbufstatus1_b1	In	1	This input connects to bit 1 of the TXBUFSTATUS1 port of the GTX_DUAL tile.
tx1_fabric_reset	Out	1	This output can be used to reset the datapath of the SDI transmitter TX1.
txreset1	Out	1	This output connects to the TXRESET1 port of the GTX_DUAL tile.
DRP Signals			
dclk	In	1	This input must be connected to a free-running clock. The same clock signal must be connected to the DCLK input of the GTX_DUAL tile. This clock input also clocks much of the reset logic. It must be driven with a valid clock signal even if, for some reason, the DRP controller is not used. It is usually possible for dclk and rate_refclk to be sourced by the same clock signal.

Table 4-9: v5gtx_sdi_control Module Ports (Cont'd)

Port Name	I/O	Width	Description
drst	In	1	This is an asynchronous reset for the DRP controller. If used, it should also be connected to the DRST input of the GTX_DUAL tile. A High pulse on this input causes the DRP controller to reinitialize the GTX transceiver attributes that it controls dynamically.
drdy	In	1	This input connects to the DRDY output of the GTX_DUAL tile.
do	In	16	This input connects to the DO port of the GTX_DUAL tile.
daddr	Out	7	This output connects to the DADDR port of the GTX_DUAL tile.
den	Out	1	This output connects to the DEN port of the GTX_DUAL tile.
di	Out	16	This output connects to the DI port of the GTX_DUAL tile.
dwe	Out	1	This output connects to the DWE port of the GTX_DUAL tile.

Global Clock Multiplexer

GTX transceiver applications require that the RXRECCLK from the GTX receiver be buffered by a global or regional clock. This global or regional clock is called RXUSRCLK because it must be connected to the RXUSRCLK and RXUSRCLK2 ports of the GTX receiver and to the downstream FPGA logic that implements the datapath of the receiver. With a 20-bit RXDATA port, RXUSRCLK and RXUSRCLK2 are always the same frequency and can be connected to the same clock source.

Under certain conditions, RXRECCLK can stop. This happens when the input bit stream stops toggling. It also happens during CDR resets. Many video applications require that the RXUSRCLK signal be kept running at a valid video frequency, even if the input bit stream stops. The solution to this problem is to use a global clock multiplexer. The multiplexer normally selects RXRECCLK to drive RXUSRCLK. But, under the conditions where RXRECCLK is stopped, the multiplexer selects an alternative reference clock to drive RXUSRCLK. One possible source for such an alternative reference clock is REFCLKOUT of the GTX_DUAL tile.

The v5gtx_sdi_control module produces two signals called rx#_clk_mux_sel, one for each RX in the GTX_DUAL tile. These signals can be used to control the clock multiplexers for the two receivers. The Verilog example below shows how the Virtex-5 FPGA BUFCTRL primitive can be instantiated to accomplish this function.

```

BUFCTRL #(
    .INIT_OUT (1),
    .PRESELECT_I0 ("FALSE"),
    .PRESELECT_I1 ("TRUE"))
CLOCK_MUX_0 (
    .O (rx0_usrclk),
    .CE0 (1'b1),
    .CE1 (1'b1),
    .I0 (rx0_recclk),
    .I1 (rx0_refclk_out),
    .IGNORE0 (1'b1),

```

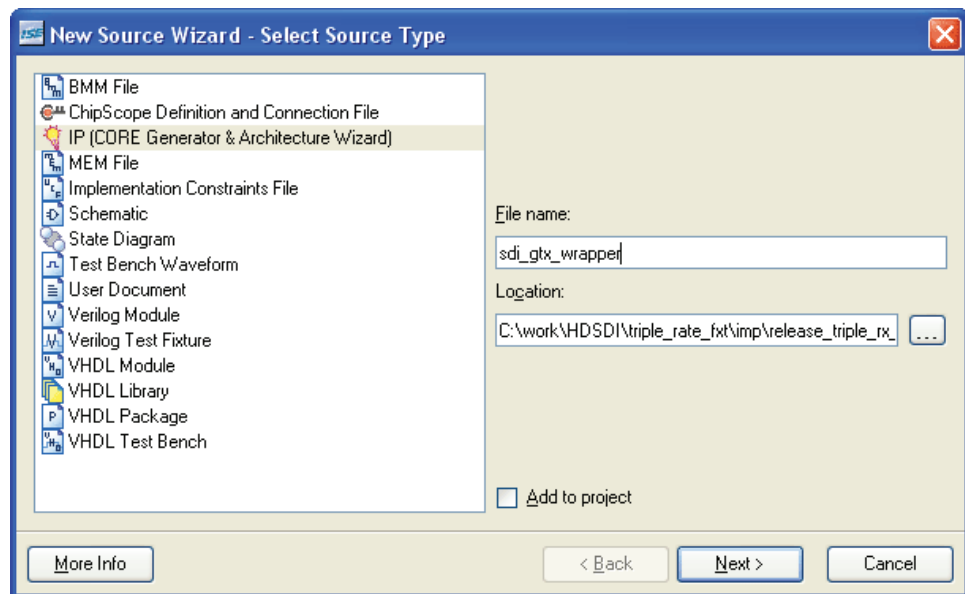
```
.IGNORE1 (1'b1),
.S0 (rx0_clk_mux_sel),
.S1 (~rx0_clk_mux_sel));
```

Using the RocketIO GTX Wizard

The RocketIO GTX Wizard is used to create the GTX wrapper module. When used with the `v5gtx_sdi_control` module, The GTX wrapper module must have certain optional ports enabled. This section gives a step-by-step example of how to create a GTX wrapper file for SDI using the RocketIO wizard. The example is based on version 1.5 of the RocketIO GTX transceiver wizard. The instructions also show how to work around the RocketIO Wizard's checking of legal frequencies for the PMA PLL, allowing the PLL to be run at slightly less than the previously specified minimum clock frequency of 1.5 GHz in order to support HD-SDI.

Every application must run the RocketIO GTX Wizard to create unique GTX wrappers tailored for that application. GTX wrapper files should not be reused from other applications or from the Xilinx SDI reference designs.

1. The RocketIO wizard is located in CORE Generator™ software. One way to launch the RocketIO wizard is to choose to add a new source in Project Navigator. This opens up the New Source Wizard window, as shown in [Figure 4-15](#).
2. The file name assigned in the New Source Wizard window becomes the name of the GTX wrapper module; it is also used to name the files created by the Wizard. Select **IP (CORE Generator & Architecture Wizard)** as the source type, as shown in [Figure 4-15](#).
3. Click **Next**.

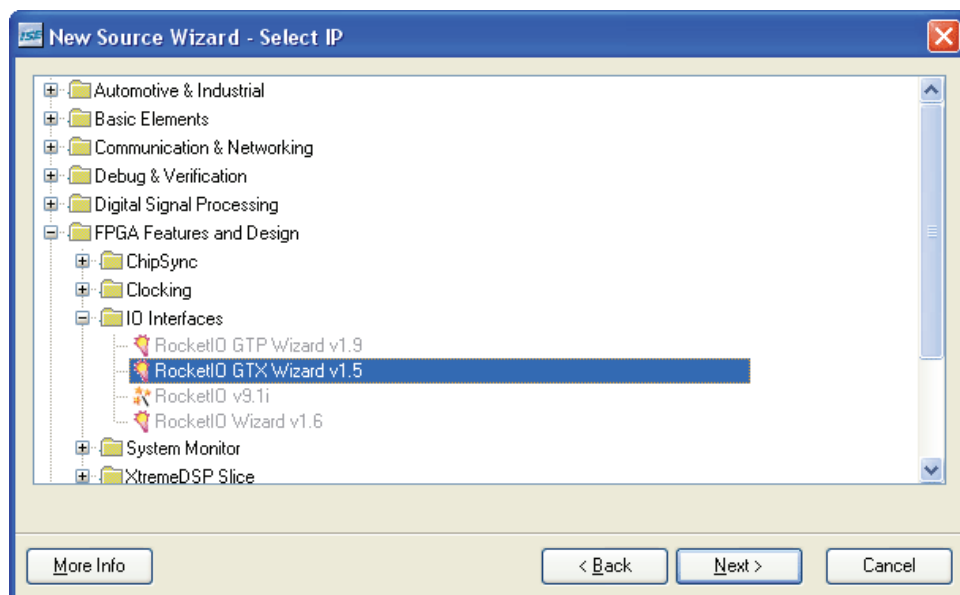


X1014_C4_34_080709

Figure 4-15: New Source Wizard – Source Type Selection

4. The window in [Figure 4-15](#) is replaced with the Select IP window shown in [Figure 4-16](#). In this window, select the RocketIO GTX Transceiver Wizard found in the IO Interfaces folder of the FPGA Features and Design folder. (If the RocketIO GTX

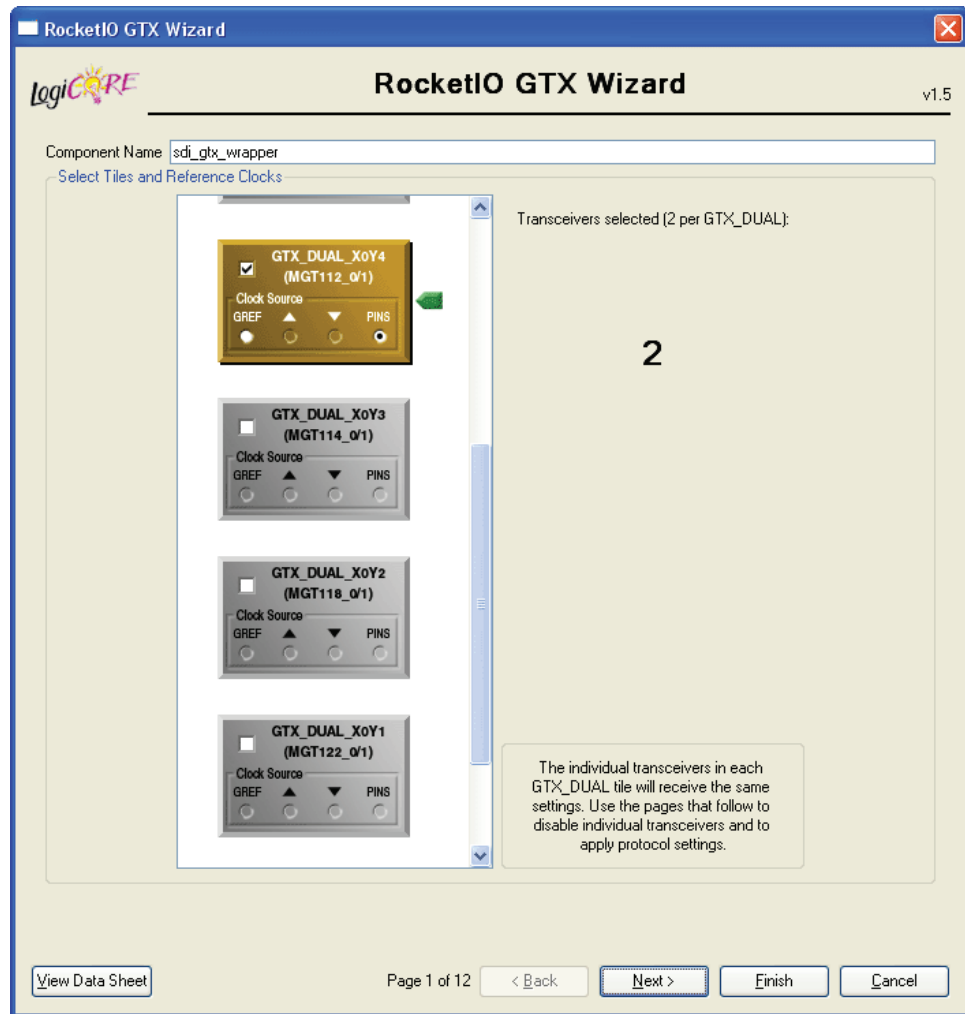
Wizard is missing, it is probably because the device type for the project is incorrectly set to a device that does not have GTX transceivers.)



X1014_C4_35_080709

Figure 4-16: New Source Wizard - IP Type Selection

5. Click **Next**.
6. In the next window, click **Finish**.
7. The RocketIO GTX Transceiver Wizard opens. The first page of the Wizard is shown in [Figure 4-17](#). This page allows the selection of the number of GTX_DUAL tiles to be included in the GTX wrapper created by the Wizard. Only those tiles that are used to implement SDI receivers and transmitters should be chosen. If other GTX_DUAL tiles are used to implement other protocols, run the Wizard again to create a separate wrapper file for each additional protocol, creating separate wrappers for the tiles that implement each different protocol. In this example, four tiles are used: tiles 112, 114, 118, and 122.



X1014_C4_36_080709

Figure 4-17: RocketIO GTX Transceiver Wizard, Page 1

If multiple tiles are selected, they are all included in one GTX wrapper file. It might be preferable to have an individual wrapper file for each GTX_DUAL tile, even if several of the tiles are used for SDI protocols. In this case, the Wizard should be run for each GTX_DUAL tile separately. It is important that unique GTX wrapper files be created for each GTX_DUAL tile used in the design or that they all be grouped into one GTX wrapper file. A GTX wrapper file must not be instantiated multiple times in a design. The GTX wrapper file can contain tile-specific code and can only be used at the location it was created for.

The page shown in [Figure 4-17](#) also allows the user to select the reference clock for the tile using a pull-down menu next to each tile. This selection does not affect any connection or attributes within the GTX wrapper file itself. The RocketIO GTX Transceiver Wizard creates an example top-level design with the GTX wrapper file instantiated. The reference clock selections made on this page are used to connect the reference clocks in the example top-level file.

The connection of reference clocks to the GTX wrapper is a critical aspect of successfully implementing an SDI design using the GTX transceiver. This topic is covered in [GTX Transceiver Reference Clock Connections](#), [page 129](#). The reference

clock selection and much of the routing of reference clocks can be overridden dynamically using the DRP controller.

8. After selecting the tiles to be included and choosing the reference clock source for each one, click **Next** to move to the second page of the Wizard as shown in Figure 4-18.

X1014_C4_37_080709

Figure 4-18: RocketIO GTX Transceiver Wizard, Page 2

9. In this window, the line rate and protocol template are selected. Select **hd sdi** from the Protocol Template pull-down menu for GTX0. This sets many of the default values—not all of them correctly, however. The Protocol Template for GTX1 should be set to **Use GTX0 settings**. This duplicates all GTX0 settings for GTX1. Changing the protocol template changes many settings. Therefore, the protocol template must always be set first, before making any other changes on this or subsequent pages.

Note: At the time of publication, the RocketIO Wizard does not allow creation of a wrapper to run the GTX transceiver PLL below 1.5 GHz. However, for HD-SDI, the GTX transceiver should be run at 1.485 GHz. Thus, this workaround creates a proper GTX wrapper for an SDI application.

10. Set the target line rate to 1.5 Gb/s. This is not the exact HD-SDI bit rate. However, if the bit rate is not set to at least 1.5 Gb/s, the RocketIO Wizard configures the PMA PLL to

- 2.97 GHz and HD-SDI does not work. Choose this line rate even if the target is SD-SDI or 3G-SDI.
11. After the line rate is set, choose the reference clock frequency from the pull-down menu. For SDI applications, the reference clock frequency is typically either 150 MHz or 75 MHz. These are not the real reference clock frequencies that are used, but match the target line rate setting of 1.5 Gb/s.
12. Ensure that:
 - a. **Use Oversampling** is NOT selected.
 - b. **Use Dynamic Reconfiguration Port** is selected.
13. The Datapath Widths for the RX and TX are set to 10 when HD-SDI is set as the protocol template. Change both of these 20 bits.
14. After the page looks exactly as shown in Figure 4-18 (with the only exception that a different reference clock frequency can be chosen), click **Next** twice to skip through page 3 and go to page 4, as shown in Figure 4-19.

RocketIO GTX Wizard v1.5

Latency, Buffering, and Clocking

PCS/PMA Phase Alignment

Bypass TX/RX Buffer is an advanced feature, please refer to ug198 for details
RX Buffer use is required to enable clock correction and channel bonding

GTX0

TXUSRCLK(2) Source: TXOUTCLK RXUSRCLK(2) Source: RXRECLK

TX PCS/PMA Alignment

☒ Enable TX Buffer (default)
☐ Lane-to-lane deskew
☐ Bypass TX Buffer (advanced feature)

RX PCS/PMA Alignment

☒ Enable RX Buffer (default)
☐ Bypass RX Buffer (advanced feature)

PPM Offset

> +/- 100 or SSC

Optional Ports

☒ RXRESET ☒ TXOUTCLK ☒ TXBUFSTATUS
☒ RXRECLK ☒ TXRESET ☐ TXINHIBIT
☒ RXBUFSTATUS ☐ TXPOLARITY
☒ RXBUFRESET ☐ TXENPRBSTST (PRBS transmission control)

GTX1

TXUSRCLK(2) Source: TXOUTCLK RXUSRCLK(2) Source: RXRECLK

TX PCS/PMA Alignment

☐ Enable TX Buffer (default)
☐ Lane-to-lane deskew
☐ Bypass TX Buffer (advanced feature)

RX PCS/PMA Alignment

☐ Use RX Buffer (default)
☐ Bypass RX Buffer (advanced feature)

PPM Offset

> +/- 100 or SSC

Optional Ports

☒ RXRESET ☒ TXOUTCLK ☒ TXBUFSTATUS
☒ RXRECLK ☒ TXRESET ☐ TXINHIBIT
☒ RXBUFSTATUS ☐ TXPOLARITY
☒ RXBUFRESET ☐ TXENPRBSTST (PRBS transmission control)

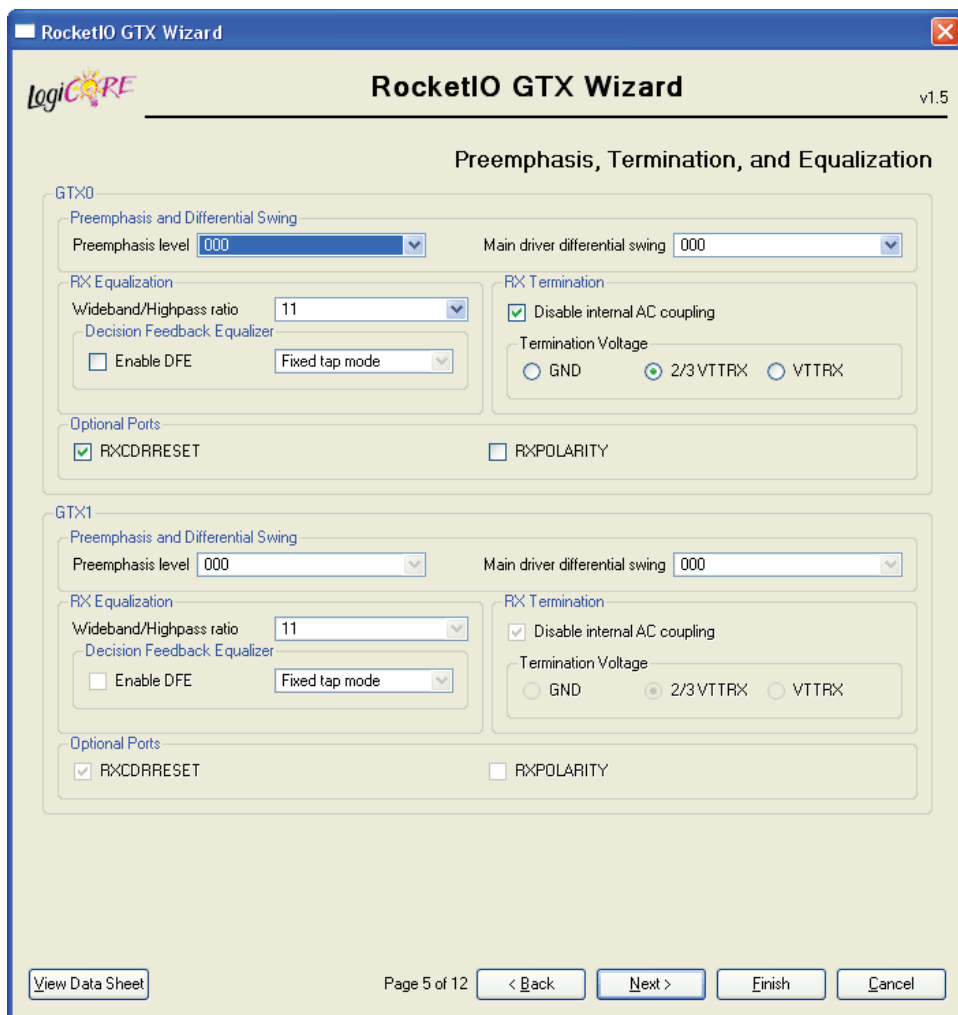
[View Data Sheet](#) Page 4 of 12 < Back Next > Finish Cancel

X1014_C4_38_080709

Figure 4-19: RocketIO GTX Transceiver Wizard, Page 4

Page 4 is mostly concerned with optional ports and whether or not the TX buffer and RX buffer are used or bypassed. It is recommended that the TX buffer be enabled. Bypassing the TX buffer requires a TX phase-alignment circuit that must be implemented and added to the design.

15. If the TX buffer is enabled, select **TXOUTCLK** for the TXUSRCLK(2) source. If the TX buffer is bypassed by selecting **Bypass Tx Buffer**, **REFCLKOUT** must be selected as the TXUSRCLK source. RXRECCLK should always be used as the RXUSRCLK source. For SDI applications, the PPM Offset must be $> \pm 100$ or SSC.
16. Select the optional ports shown in Figure 4-19. The ports selected are all required by the `v5gtx_sdi_control` module.
17. Click **Next** to move to page 5, shown in Figure 4-20.



The screenshot shows the 'RocketIO GTX Wizard' window, version 1.5, titled 'Preemphasis, Termination, and Equalization'. It contains two identical sections for GTX0 and GTX1. Each section has the following settings:

- Preemphasis and Differential Swing:** Preemphasis level is set to 000, and Main driver differential swing is set to 000.
- RX Equalization:** Wideband/Highpass ratio is set to 11. Under 'Decision Feedback Equalizer', 'Enable DFE' is unchecked and 'Fixed tap mode' is selected.
- RX Termination:** 'Disable internal AC coupling' is checked. 'Termination Voltage' is set to 2/3 VTTRX.
- Optional Ports:** 'RXCDRRESET' is checked, and 'RXPOLARITY' is unchecked.

At the bottom of the window, there is a 'View Data Sheet' button, a page indicator 'Page 5 of 12', and navigation buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

X1014_C4_39_080709

Figure 4-20: RocketIO GTX Transceiver Wizard, Page 5

The transmitter Pre-emphasis and Differential Swing settings can be set to the normal levels for most SDI-SDI applications. They are usually not critical when the SDI cable driver is located fairly close to the FPGA. These settings can be modified, however, to match the requirements of each application.

The internal AC coupling must be disabled and the Termination Voltage set to 2/3 VTTRX. This is required because the internal AC coupling capacitors are not large enough to handle the long run lengths of the SDI protocols. External AC coupling capacitors are used between the cable equalizer and the GTX receiver inputs. The optional RXCDRRESET port must also be enabled. This is required by the `v5gtx_sdi_control` module.

None of the other pages of the wizard are applicable to SDI, so they can either be paged through with the **Next** button until the final screen is reached, or the **Finish** button can be clicked. After **Finish** has been clicked, the Wizard creates the GTX wrapper. It creates two wrapper files in both Verilog and VHDL (a total of 4 wrapper files). The wrapper files are placed in the project directory into a folder of the same name that was assigned as the file name in [Figure 4-15, page 123](#). This folder contains another folder called `src` that contains the actual source code of the wrapper files.

After the steps for the given example are completed, the `src` folder contains four files:

- `sdi_gtx_wrapper.v`
- `sdi_gtx_wrapper.vhd`
- `sdi_gtx_wrapper_tile.v`
- `sdi_gtx_wrapper_tile.v`

The `sdi_gtx_wrapper` is instantiated inside of `sdi_gtx_wrapper` one or more times, once for each tile selected on the first page of the RocketIO wizard.

In addition, the wizard creates a number of example files. The `ucf` folder includes a file called `example_mgt_top.ucf` containing some constraints that are useful or required. It contains example constraints for constraining the pin locations of the GTX transceiver reference clock inputs. These pins must be constrained, and the example given in the UCF file is a good place to start. The GTX_DUAL tile locations must also be constrained; the UCF file contains example constraints for this, too.

GTX Transceiver Reference Clock Connections

Previous sections of this chapter have described the reference clock routing resources associated with the GTX_DUAL tiles and how these resources can be manipulated dynamically through the DRP. There are other important considerations to understand regarding how to connect the reference clocks in HDL code to make sure the reference clocks are available in the clock routing structure.

For example, every reference clock used in the design must be connected to an active GTX_DUAL tile in the HDL design. Otherwise, the clock input can be optimized out of the design by the synthesis tools, making it unavailable in the actual implementation.

Every external reference clock input must be connected to an IBUFDS input buffer. This input buffer should be constrained so that it is placed next to the desired GTX_DUAL tile. The easiest way to do this is by applying pin LOC constraints to the signals connected to the IBUFDS inputs.

For example, the following Verilog code instantiates an IBUFDS with input signals called `PAD_mgtclk_hd_p` and `PAD_mgtclk_hd_n`. These two signals are defined as inputs to the top-level module of the design.

```
IBUFDS #(
    .IOSTANDARD ("LVDS_25"))
MGTCLK112 (
    .I (PAD_mgtclk_hd_p),
    .IB (PAD_mgtclk_hd_n),
```

```
.O (mgtclk_148_5M));
```

These two constraints in the UCF file constrain the two input signals to specific pins that are dedicated to the reference clock input buffer associated with GTX_DUAL tile 112 in an XC5VFX70T-FF1136 device.

```
NET "PAD_mgtclk_hd_p" LOC = "P4";
NET "PAD_mgtclk_hd_n" LOC = "P3";
```

This ensures that the reference clock enters the FPGA at the correct place. However, it is not sufficient to ensure that the reference clock is available to the GTX transceiver reference clock routing resources. If the output of the IBUFDS is not connected to the CLKIN pin of at least one GTX transceiver that is active in the design, it is optimized out of the design. In this case, the mgtclk_148_5M signal must be connected to the CLKIN_IN port of at least one GTX_DUAL tile. That GTX_DUAL tile must also be constrained to its proper location. Finally, the GTX_DUAL tile must be connected sufficiently so that it is not optimized out of the design. Even if it is not actually used in the design, the GTX_DUAL tile must be minimally connected so the synthesis tools believe it is used in the design. The easiest way to ensure that the GTX_DUAL tile is not optimized out of the design is to connect at least some of its RXN and TXN ports (differential serial I/O signals) to ports on the top-level design, and connect a reference clock to the tile's CLKIN port.

It is not possible to connect the output of the IBUFDS to anything other than CLKIN ports of GTX_DUAL tiles because this IBUFDS is constrained to a location that is a dedicated GTX transceiver reference clock input buffer. The output of the IBUFDS can be connected to the CLKIN ports of more than one GTX_DUAL tile, and the tools route the clock through the CLKNORTH and CLKSOUTH routing resources to the various GTX_DUAL tiles that it drives.

Any unused GTX_DUAL tile through which reference clocks are routed must also be instantiated in the design. They must also be minimally connected so that they are not optimized out of the design. If a GTX_DUAL tile that is used solely to route reference clocks is not instantiated in the design or gets optimized out of the design because it was not connected to anything, it is powered down and reference clocks cannot be routed through the tile. Also, unused GTX_DUAL tiles used only for reference clock routing must have their external power pins connected on the circuit board to power the tile. If the tile is not powered, reference clocks cannot be routed through it.

With the dynamic reference clock routing control provided by the DRP controller, if the reference clocks are available to the GTX transceiver reference clock routing resources, they can be switched to the required GTX_DUAL tiles. However, this switching occurs some time after the FPGA configuration is complete. It is often useful to ensure that each GTX_DUAL tile has a valid SDI reference clock connected to its CLKIN pin in the HDL design file. This provides a valid reference clock to the tile's PLL immediately after the FPGA comes out of configuration.

With the dynamic control of the reference clock routing, it is possible to violate the limits placed on the number of GTX_DUAL tiles that can be driven by a single external reference clock input. However, these limits must still be observed, otherwise signal degradation of the reference clock could prevent the design from working properly.

Section II:

Multirate SD/HD/3G-SDI

Using Virtex-5 FPGA

RocketIO Transceivers

*Audio/Video Connectivity Solutions
for Virtex-5 FPGAs*

Triple-Rate SDI Receiver for Virtex-5 LXT and SXT Devices

Summary

The RocketIO™ GTP transceivers available in Virtex®-5 LXT and SXT devices can be used to implement SDI receivers that support all three SMPTE serial digital video interface standards (SD-SDI, HD-SDI, and 3G-SDI), and can automatically switch between these three standards as the input signal changes. This chapter describes such a “triple-rate” SDI receiver reference design. The reference design also supports dual link HD-SDI using two combined triple-rate SDI receivers.

This triple-rate SDI receiver reference design has been designed specifically for the GTP transceiver available in Virtex-5 LXT and SXT devices. It cannot be used directly with other Xilinx® FPGAs.

Introduction

There are many options when implementing a triple-rate SDI interface. Some applications require full unpacking of the 3G-SDI video streams into 1080p 50 Hz or 60 Hz native video and other video formats that have more than 20 bits per sample. Other applications might not need to unpack the 3G-SDI streams into native video, either because the application does not need to support those particular video formats or because the unpacking function is implemented elsewhere in the application. Some applications require EDH error checking for SD-SDI, while others might omit this function to reduce space.

In recognition of the many implementation options available, the triple-rate SDI receiver reference design is designed in a modular manner. This allows it to be flexible enough to be useful for a wide range of application requirements, from the simplest to the most complex. Two versions of the triple-rate SDI receiver reference design are provided in this chapter: a light version and a full-featured version. The light version is designed for less demanding applications, and provides a simpler, less complicated interface. The full-featured version is larger and more complex, and supports full unpacking into native video for every video format compatible with the 3G-SDI standard, level A and level B, as well as all video formats supported by dual link HD-SDI.

[Chapter 3, Implementing SMPTE Serial Digital Interfaces with RocketIO GTP Transceivers](#) contains information vital to implementing triple-rate SDI receivers using the Virtex-5 FPGA GTP transceiver, and is prerequisite to reading this chapter.

Supported Video Formats

The triple-rate SDI receiver supports the video formats shown in Table 5-1. The full-featured version can unpack all of these into native video. The term “native video” is used in this document to mean digital video that conforms to the SMPTE digital video standards such as SMPTE 274M and SMPTE 296M.

The 3G-SDI and dual link HD-SDI standards require many of the supported native video formats to be mapped prior to transmission into SDI streams before they can be transported on an SDI interface. To convert the SDI data streams back into native video, a complementary unpacking function must be implemented. The light version supports all of these formats, but does not provide unpacking functions for all video formats.

Table 5-1: Video Formats Supported by Triple-Rate SDI Receiver

Interface	Video Standard	Sampling Structure/Bit Depth	Frame/Field Rate (Hz)	Native Video Unpacking
SD-SDI SMPTE 259M-C	PAL	4:2:2 Y'C _B 'C _R ' 10-bit	50	Not required
	NTSC	4:2:2 Y'C _B 'C _R ' 10-bit	59.94	Not required
HD-SDI SMPTE 292M	SMPTE 274M	4:2:2 Y'C _B 'C _R ' 10-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	Not required
	SMPTE 296M	4:2:2 Y'C _B 'C _R ' 10-bit	720p: 23.98, 24, 25, 29.97, 30, 50, 59.94, 60	Not required
	SMPTE 260M	4:2:2 Y'C _B 'C _R ' 10-bit	1035i: 59.94, 60	Not required
	SMPTE 295M	4:2:2 Y'C _B 'C _R ' 10-bit	1080i: 50	Not required
3G-SDI Level A SMPTE 425M-A	SMPTE 274M	4:2:2 Y'C _B 'C _R ' 10-bit	1080p: 50, 59.94, 60	Light: Yes Full: Yes
		4:4:4 or 4:4:4:4 Y'C _B 'C _R ' or RGB 10-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	Light: No Full: Yes
		4:4:4 Y'C _B 'C _R ' or RGB 12-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	Light: No Full: Yes
		4:2:2 Y'C _B 'C _R ' 12-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	Light: No Full: Yes
	SMPTE 296M	4:4:4 or 4:4:4:4 Y'C _B 'C _R ' or RGB 10-bit	720p: 23.98, 24, 25, 29.97, 30, 50, 59.94, 60	Light: No Full: Yes
	SMPTE 428-9	4:4:4 X'Y'Z' 12-bit	2048 X 1080p: 24	Light: No ⁽¹⁾ Full: No ⁽¹⁾

Table 5-1: Video Formats Supported by Triple-Rate SDI Receiver (Cont'd)

Interface	Video Standard	Sampling Structure/Bit Depth	Frame/Field Rate (Hz)	Native Video Unpacking
3G-SDI Level B SMPTE 425M-B	SMPTE 372M	See Dual Link HD-SDI SMPTE 372M in this table.		See Dual Link HD-SDI SMPTE 372M
	2 X HD-SDI streams	See Dual Link HD-SDI SMPTE 372M in this table.		Light: Yes Full: Yes
Dual Link HD-SDI SMPTE 372M	SMPTE 274M	4:2:2 Y'C _B 'C _R ' 10-bit	1080p: 50, 59.94, 60	Light: No Full: Yes
		4:4:4 or 4:4:4:4 Y'C _B 'C _R ' or RGB 10-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	Light: No Full: Yes
		4:4:4 Y'C _B 'C _R ' or RGB 12-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	Light: No Full: Yes
		4:2:2 Y'C _B 'C _R ' 12-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	Light: No Full: Yes
	SMPTE 428-9	4:4:4 X'Y'Z' 12-bit	2048 X 1080p: 24	Light: No ⁽¹⁾ Full: No ⁽¹⁾

Notes:

- For SMPTE 428-9 video, the light and full-featured triple-rate SDI receivers can receive the SMPTE 428-9 container but cannot unpack it into native SMPTE 428-1 video.

Triple-Rate SDI Receiver Features

One important feature of the triple-rate SDI receiver is automatic switching between video standards. As the input signal to the receiver switches between SD-SDI, HD-SDI, and 3G-SDI, the receiver automatically detects when the SDI bit rate changes and reconfigures itself to correctly receive the signal. Both HD-SDI bit rates, both 3G-SDI bit rates, and the 270 Mb/s bit rate SD-SDI can be received by the GTP receiver with only a single reference clock frequency. The reference clock frequency can be any of the following frequencies: 148.5 MHz, 148.5/1.001 MHz, 74.25 MHz, or 74.25/1.001 MHz. The two receivers in the GTP_DUAL tile share this same reference clock input, but because they can receive any of these SDI bit rates using the common reference clock, the two receivers in the same tile can act independently. For example, one RX unit in the GTP_DUAL tile can receive 3G-SDI at 2.97/1.001 Gb/s, while the other RX unit in the same tile receives SD-SDI at 270 Mb/s, HD-SDI at either bit rate, or 3G-SDI at either bit rate.

The triple-rate SDI receiver reference design is primarily designed to support only the 270 Mb/s SD-SDI bit rate. It is possible to receive the other SD-SDI bit rates with modifications to this reference design. Bit rates other than 270 Mb/s require different reference clock frequencies for the GTP receiver. This requires a reference clock switching solution external to the FPGA and is beyond the scope of this chapter. It would also require changes to the bit rate detection module (`triple_sdi_rx_autorate`) to support the additional bit rates. Additionally, the reference clock switching solution would limit the ability of the two receivers in the same GTP_DUAL tile to act independently because there

is only one reference clock input per tile. Therefore, for example, if the reference clock frequency is changed so that one receiver in the tile can receive 360 Mb/s SD-SDI, both receivers in the tile would be unable to receive any bit rate except 360 Mb/s.

Using a GTP transmitter in the same GTP_DUAL tile with receivers presents some challenges. The transmitter requires that the reference clock frequency be switched between 148.5 MHz and 148.5/1.001 MHz (or between 74.25 MHz and 74.25/1.001 MHz) when the transmit bit rate needs to change. This changing of the reference clock frequency for the transmitter briefly upsets the CDR units of the receivers in the same tile, resulting in reception errors.

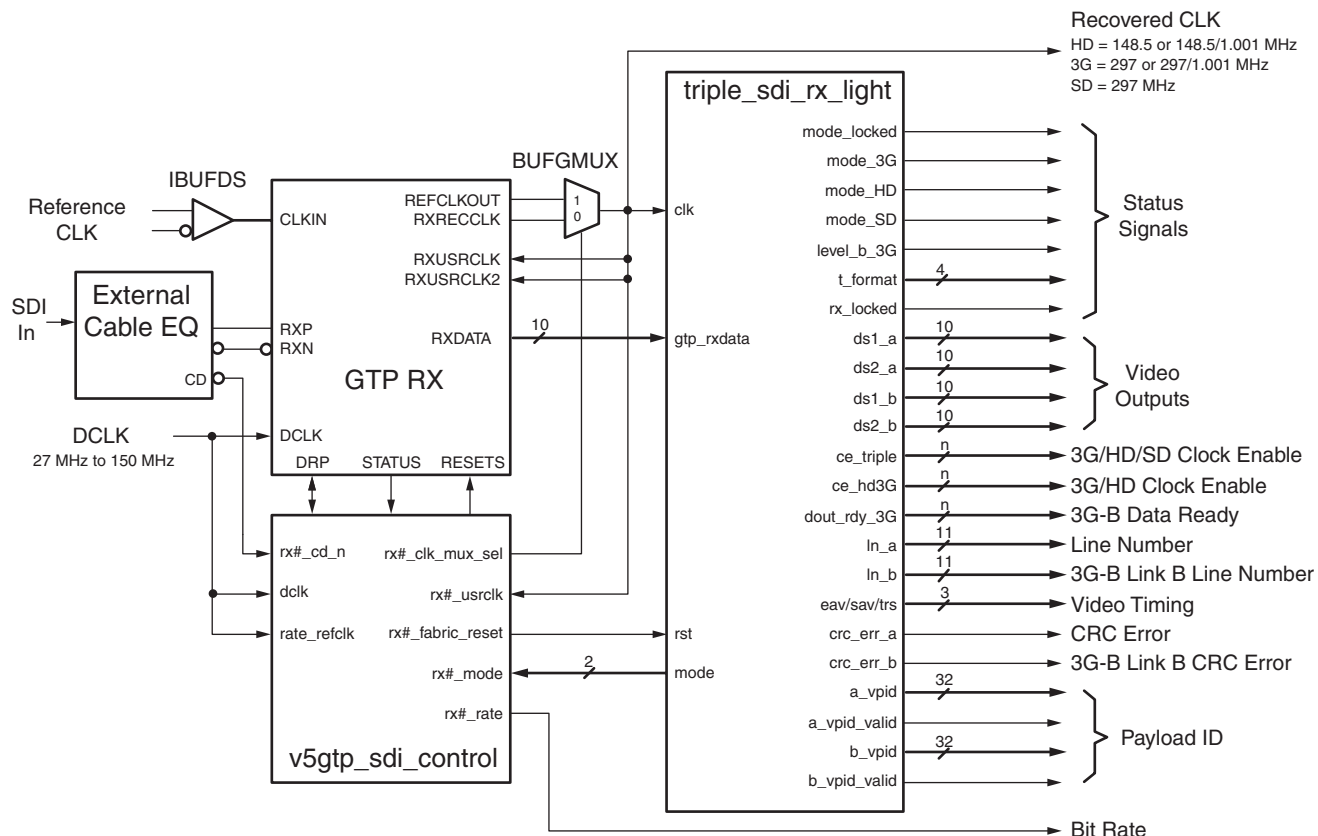
The triple-rate SDI receiver can capture SD-SDI EDH packets and use them to detect receive errors. It can also check HD-SDI and 3G-SDI data streams for receive errors using the CRC words embedded in each line. The receiver can also capture SMPTE 352M video payload ID (VPID) packets. The full-featured triple-rate SDI receiver must capture these VPID packets to properly unpack the 3G-SDI and dual link HD-SDI streams into native video.

Light Triple-Rate SDI Receiver Reference Design

The `triple_sdi_rx_light` module, combined with a GTP receiver and a `v5gtp_sdi_control` module, provides a basic triple-rate receiver, as shown in [Figure 5-1](#). It has these features:

- Only one reference clock frequency is required to receive the five supported bit rates: 270 Mb/s, 1.485 Gb/s, 148.5/1.001 Gb/s, 2.97 Gb/s, and 2.97/1.001 Gb/s.
- The receiver has a bit rate detector capable of distinguishing between the 1.485 Gb/s and 1.485/1.001 Gb/s HD-SDI bit rates, and between the two 3G-SDI bit rates. An output signal from the receiver indicates to the application which bit rate is being received.
- The receiver can automatically detect the SDI standard (3G-SDI, HD-SDI, SD-SDI, or dual link HD-SDI) of the input signal. The current mode is reported on an output port.
- Detection and reporting of the video transport format (i.e., 1080p 30 Hz, 1080i 50 Hz, etc.) is supported.
- The receiver supports all 3G-SDI level A video formats (SMPTE 425M-A) and all 3G-SDI level B video formats (SMPTE 425M-B), with automatic detection of level A or level B. However, not all video formats are unpacked into native video.
- CRC error checking is supported for HD-SDI and 3G-SDI.
- SMPTE 352M VPID packets are captured for all SDI standards. All captured SMPTE 352M packet data is available on output ports for one stream, or two streams for those formats that require SMPTE 352M packets in both streams.
- The receiver has a flexible set of clock enable outputs. The number of each type of clock enable is determined by a parameter (or generic) passed to the module.
- The receiver datapath is designed to interface with a 10-bit GTP RXDATA port to minimize global clocking resources. Only a single global clock is required for the receiver, which is driven by the RXRECCLK of the GTP receiver. No DCMs or PLLs are required for most implementations.
- A BUFGMUX switches the main clock to a backup reference clock when the recovered clock stops due to interruptions of the input serial bitstream.

The `triple_sdi_rx_light` module does not provide EDH error checking for SD-SDI. EDH error checking can be added externally to the `triple_sdi_rx_light` module. The EDH processor provided in *Audio/Video Connectivity Solutions for Virtex-II Pro and Virtex-4 FPGAs* [Ref 5] can be connected directly to the outputs of the `triple_sdi_rx_light` module.



Note: The BUFGMUX allows REFCLKOUT to be used as a backup, free-running clock when RXRECCLK stops. This is optional and RXRECCLK can instead go to a BUFG to drive the RX clock. REFCLKOUT cannot be connected directly to the BUFGMUX if the reference clock frequency is 74.25 MHz. This frequency must first be doubled with a PLL or DCM.

X1014_C4_01_040909

Figure 5-1: Block Diagram of Light Triple-Rate SDI Receiver

Table 5-2 describes the I/O ports of the `triple_sdi_rx_light` module.

Table 5-2: Ports of `triple_sdi_rx_light` Module

Port Name	I/O	Width (Bits)	Description
Inputs			
clk	In	1	This port must be connected to the global or regional clock that also drives the RXUSRCLK and RXUSRCLK2 ports of the GTP RX. This clock is usually driven by the RXRECCLK output of the GTP RX. The clock frequency must be 297 MHz (or 297/1.001 MHz) for 3G-SDI. It must be 148.5 MHz (or 148.5/1.001 MHz) for HD-SDI. It must be 297 MHz for SD-SDI.

Table 5-2: Ports of `triple_sdi_rx_light` Module (Cont'd)

Port Name	I/O	Width (Bits)	Description
rst	In	1	The falling edge of the asynchronous reset signal must meet the reset recovery time of all flip-flops relative to the next rising edge of clk. This input can be driven by the rx#_fabric_reset output of the v5gtp_sdi_control module, which, when combined with proper timing constraints, results in proper timing of the reset signal to the synchronous elements it resets.
gtp_rxdata	In	10	This input connects to the 10-bit RXDATA port of the GTP wrapper module.
frame_en	In	1	The framer function of the triple_sdi_rx_light module automatically readjusts the word alignment to match the alignment of each TRS (EAV or SAV) when this input is High. This input can be used to implement TRS alignment filtering. For example, if the nsp output is connected to the frame_en input, the framer ignores a single misaligned TRS, keeping the existing word alignment until the new word alignment is confirmed by a second matching TRS. If TRS filtering is not desired, this input must be tied High. Also, it is important to turn off any TRS filtering on the synchronous switching line by driving the frame_en input High on the synchronous switching lines.
Outputs			
mode	Out	2	<p>This port indicates the current SDI mode of the receiver:</p> <ul style="list-style-type: none"> 00 = HD-SDI 01 = SD-SDI 10 = 3G-SDI <p>The mode port changes value as the receiver searches for the correct mode. During this time, the mode_locked output is Low. When the receiver detects the correct SDI mode matching the input data stream, the mode_locked output switches to High.</p> <p>Note: When using the v5gtp_sdi_control module, this output port must drive the rx#_mode input of the v5gtp_sdi_control module for the particular RX unit in the GTP_DUAL tile used to implement this SDI receiver.</p>
mode_HD mode_SD mode_3G	Out	1	These three output ports are decoded unary versions of the mode port. They are provided for convenience. Unlike the mode output, which changes continuously as the receiver seeks to identify and lock to the incoming signal, these outputs are all forced Low when the receiver is not locked. The mode output matching the current operating mode of the receiver is High when mode_locked is High. It is essential, however, that these outputs not be used to control any data or control path essential for locking the receiver to the incoming signal. Those paths must be controlled directly by decoding the mode port.
mode_locked	Out	1	When this output is Low, the receiver is actively searching for the SDI mode that matches the input data stream. During this time, the mode output port changes frequently. When the module locks to the correct SDI mode, the mode_locked output goes High.

Table 5-2: Ports of triple_sdi_rx_light Module (Cont'd)

Port Name	I/O	Width (Bits)	Description
t_format	Out	4	This output port indicates the video transport timing format of the SDI signal in HD and 3G modes only. See Table 5-3 for encoding. The output port is only valid when rx_locked is High. This output port is not valid in SD mode.
level_b_3G	Out	1	In 3G-SDI mode, this output is asserted High when the input signal is level B compliant and Low when it is level A compliant. This output is only valid in 3G-SDI mode.
rx_locked	Out	1	This output is High when the receiver is locked to the incoming signal. In HD-SDI and 3G-SDI modes, this output is driven by the transport format detector that generates the t_format output. In SD-SDI mode, this output is identical to the mode_locked signal. Because this output is driven by the transport format detector, there can be more than one video frame time of delay from when the receiver is actually locked to the input signal until rx_locked is asserted. During this time, the transport format detector is determining the transport format.
ce_triple	Out	NUM_TRIPLE_CE	These clock enables are valid for all three SDI modes, although for some modes, these clock enables are not asserted at the sample rate. These clock enables are asserted at half the clk frequency for HD-SDI and 3G-SDI and at 27 MHz for SD-SDI. The number of identical clock enables provided on this port is specified by the NUM_TRIPLE_CE parameter/generic.
ce_hd3G	Out	NUM_HD3G_CE	These clock enables are valid for HD-SDI and 3G-SDI, but not for SD-SDI. The number of identical clock enables provided on this port is specified by the NUM_HD3G_CE parameter/generic.
nsp	Out	1	When this output is High, it indicates that the framer has detected a TRS at a new word alignment. If frame_en is High, this output is only asserted briefly. If frame_en is Low, this output remains asserted until the framer is allowed to readjust to the new TRS alignment.
ln_a	Out	11	This output provides the current line number captured from the LN words of the Y data stream. This output is valid in HD-SDI and 3G-SDI modes, but not in SD-SDI mode. In 3G-SDI level B mode, it represents the line number from the Y data stream of link A. For any case where the interface line number is not the same as the picture line number, such as for 1080p 60 Hz carried on 3G-SDI level B or dual link HD-SDI, this output equals the interface line number.
a_vpid	Out	32	This port outputs all four user data bytes of the SMPTE 352M packet from data stream 1 in the following format: byte4, byte3, byte2, byte1 (MSB to LSB). This output is valid only when a_vpid_valid is High and works in any SDI mode. In 3G-SDI level A mode, it outputs the VPID data captured from data stream 1 (Y). In 3G-SDI level B mode, it outputs the VPID data captured from data stream 1 of link A (dual link streams) or HD-SDI stream 1 (dual streams).
a_vpid_valid	Out	1	This output is High if a_vpid is valid.

Table 5-2: Ports of `triple_sdi_rx_light` Module (Cont'd)

Port Name	I/O	Width (Bits)	Description
b_vpid	Out	32	This port outputs all four user data bytes of the SMPTE 352M packet from data stream 2 in the following format: byte4, byte3, byte2, byte1 (MSB to LSB). This output is valid only in 3G-SDI mode and only when b_vpid_valid is High. In 3G-SDI level A mode, it outputs the VPID data captured from data stream 2 (C). In 3G-SDI level B mode, it outputs the VPID data captured from data stream 1 of link B (dual link streams) or HD-SDI stream 2 (dual streams).
b_vpid_valid	Out	1	This output is High if b_vpid is valid.
crc_err_a	Out	1	This output is asserted High for one sample period when a CRC error is detected on the previous video line. For 3G-SDI level B, this output indicates CRC errors on data stream 1 only. A second output called crc_err_b indicates CRC errors on data stream 2 for 3G-SDI level B. This output is not valid in SD-SDI mode.
ds1_a	Out	10	The output of this port is determined by the video standard: <ul style="list-style-type: none"> SD-SDI: Multiplexed Y/C data stream. HD-SDI: Y data stream. 3G-SDI level A: Data stream 1 (Y). 3G-SDI level B: Data stream 1 (Y) of link A or HD-SDI stream 1.
ds2_a	Out	10	The output of this port is determined by the video standard: <ul style="list-style-type: none"> SD-SDI: Not used. HD-SDI: C data stream. 3G-SDI level A: Data stream 2 (C). 3G-SDI level B: Data stream 2 (C) of link A or HD-SDI stream 2.
eav	Out	1	This output is asserted High for one sample time when the XYZ word of an EAV is present on the data stream output ports.
sav	Out	1	This output is asserted High for one sample time when the XYZ word of an SAV is present on the data stream output ports.
trs	Out	1	This output is asserted High for four sample times as all four words of an EAV or SAV are output on the data stream output ports.
Outputs Used Only in 3G-SDI Level B Mode			
dout_rdy_3G	Out	NUM_3G_DRDY	In 3G-SDI level B mode, the output data rate is 74.25 MHz, but the ce_triple and ce_hd3G outputs are asserted at 148.5 MHz. The dout_rdy_3G outputs are asserted at a 74.25 MHz rate in level B mode. In 3G-SDI level B mode, this output is a 74.25 MHz square wave. In other modes, this output is always High. Data must only be taken by downstream devices when both dout_rdy_3G and one of the clock enables (ce_triple or ce_hd3G) are High. Datapaths that are not used in 3G-SDI level B mode can use the clock enables directly without qualifying them with dout_rdy_3G.
ds1_b	Out	10	This output is the data stream 1 (Y) of link B or HD-SDI stream 2.
ds2_b	Out	10	This output is the data stream 2 (C) of link B or HD-SDI stream 2.

Table 5-2: Ports of `triple_sdi_rx_light` Module (Cont'd)

Port Name	I/O	Width (Bits)	Description
crc_err_b	Out	1	This output is the CRC error for link B or HD-SDI stream 2.
ln_b	Out	11	This line number output is only valid in 3G-SDI level B mode and represents the line number from the Y data stream of link B or HD-SDI stream 2. For any case where the interface line number is not the same as the picture line number, this output equals the interface line number.

Table 5-3 shows the encoding of the `t_format` port. This port indicates the transport format (not always the same as the picture format) that the receiver calculates by counting words per line and active lines per field or frame. The `t_format` port can uniquely identify most video transport formats, but cannot distinguish between transport formats that have identical timing. For example, it cannot differentiate between 1080i 60 Hz and 1080PsF 30 Hz (1080p 30 Hz video carried on a 1080i 60 Hz transport) because there is no difference in the transport timing.

In dual link HD-SDI and 3G-SDI level B modes, the 1080p 50 Hz and 60 Hz video formats are carried on an interlaced transport. Therefore, the `triple_sdi_rx_light` module reports them as being 1080i 50 Hz and 1080i 60 Hz, respectively. However, in 3G-SDI level A mode, these two video formats are carried progressively and are uniquely identified as 1080p 50 Hz and 1080p 60 Hz (codes 1110 and 1101, respectively).

Table 5-3: `t_format` Output Port Encoding for `triple_sdi_rx_light` Module

<code>t_format</code>	Standard	Video Format (Frame Rate for p and Field Rate for i)
0000	SMPTE 260M	1035i 49.94 Hz and 60 Hz
0001	SMPTE 295M	1080i 50 Hz
0010	SMPTE 274M	1080i 59.94 Hz and 60 Hz 1080PsF 29.97 Hz and 30 Hz
0011	SMPTE 274M	1080i 50 Hz 1080PsF 25 Hz
0100	SMPTE 274M	1080p 29.97 Hz and 30 Hz 1080p 59.94 Hz and 60 Hz (3G-SDI level B only)
0101	SMPTE 274M	1080p 25 Hz 1080p 50 Hz (3G-SDI level B only)
0110	SMPTE 274M	1080p 23.98 Hz and 24 Hz
0111	SMPTE 296M	720p 59.94 Hz and 60 Hz
1000	SMPTE 274M	1080PsF 23.98 Hz and 24 Hz
1001	SMPTE 296M	720p 50 Hz
1010	SMPTE 296M	720p 29.97 Hz and 30 Hz
1011	SMPTE 296M	720p 25 Hz
1100	SMPTE 296M	720p 23.98 Hz or 24 Hz
1101	SMPTE 274M	1080p 59.94 Hz or 60 Hz (3G-SDI level A only)

Table 5-3: t_format Output Port Encoding for triple_sdi_rx_light Module (Cont'd)

t_format	Standard	Video Format (Frame Rate for p and Field Rate for i)
1110	SMPTE 274M	1080p 50 Hz (3G-SDI level A only)
1111	Reserved	

Table 5-4 describes the Verilog parameters or VHDL generics that can be supplied to the triple_sdi_rx_light module. If a parameter/generic is not supplied, it defaults to the value given in the table.

Table 5-4: Parameters of triple_sdi_rx_light Module

Parameter	Default Value	Description
NUM_TRIPLE_CE	2	This parameter specifies the number of identical ce_triple clock enable signals provided by the triple_sdi_rx module. This parameter should never be set to less than 1.
NUM_HD3G_CE	2	This parameter specifies the number of identical ce_hd3G clock enable signals provided by the triple_sdi_rx module. This parameter should never be set to less than 1.
NUM_3G_DRDY	2	This parameter specifies the number of identical dout_rdy_3G data ready signals provided by the triple_sdi_rx module. This parameter should never be set to less than 1.
ERRCNT_WIDTH	4	This is a parameter of the SDI mode detection module. It specifies the number of bits used in the error counter. This counter must be wide enough to support counting up to the values specified by MAX_ERRS_LOCKED and MAX_ERRS_UNLOCKED.
MAX_ERRS_LOCKED	15	This is a parameter of the SDI mode detection module. It specifies the maximum number of consecutive lines with errors allowed while the receiver is locked to the SDI mode before the receiver moves to the unlocked state and begins searching for the correct SDI mode.
MAX_ERRS_UNLOCKED	2	This is a parameter of the SDI mode detection module. It specifies the maximum number of consecutive lines with errors allowed while the receiver is searching for the correct SDI mode before it continues the search by moving to the next mode.

Light Triple-Rate Receiver Clocking

The clock input (clk) to the triple_sdi_rx_light module must come from a global (or regional) clock buffer driven by the recovered clock from the GTP receiver (RXRECCLK). As described in [Chapter 3, Implementing SMPTE Serial Digital Interfaces with RocketIO GTP Transceivers](#), this clock might stop if the input SDI serial bitstream stops. The user might therefore want to use a BUFGMUX as the global clock buffer. The v5gtp_sdi_control module has an output to control a BUFGMUX so that a backup, free-running reference clock can be supplied in place of RXRECCLK when RXRECCLK is stopped. This is entirely optional and is not required for all applications.

Note: If the reference clock frequency is 74.25 MHz or 74.25/1.001 MHz, REFCLKOUT from the GTP transceiver will not be the correct frequency to be used directly as an input to the BUFGMUX, as shown in [Figure 5-1, page 137](#). This is because REFCLKOUT is always exactly the same frequency as the reference clock input. The frequency of the backup clock into the BUFGMUX must be either

148.5 MHz or 297 MHz. Thus, a DCM or PLL would be required to double a 74.25 MHz REFCLKOUT to 148.5 MHz, unless another suitable backup clock is available in the FPGA. However, a single 74.25 MHz REFCLKOUT from any GTP RX in the FPGA, doubled to 148.5 MHz, can be used as the backup clock for any number of SDI receivers by connecting it to the BUFGMUX of each receiver.

The frequency of the recovered clock depends on the current SDI mode of the receiver. In HD-SDI mode, it is 148.5 MHz (or 148.5/1.001 MHz). In 3G-SDI mode, it is 297 MHz or 297/1.001 MHz, and for SD-SDI, it is 297 MHz.

Internally, most of the `triple_sdi_rx_light` module runs at half the recovered clock frequency for HD-SDI and 3G-SDI (some portions run at one quarter of the recovered clock frequency in 3G-SDI mode, depending on the video format). For SD-SDI, the internal data rate is 27 MHz. Clock enables are used in the `triple_sdi_rx_light` module to run the various sections at the proper rates.

The `triple_sdi_rx_light` module also supplies various clock enables on output ports. These can be used, in conjunction with the global recovered clock, to clock logic downstream from the `triple_sdi_rx_light` module at the correct data rate. There are three such signals. The number of each type is controlled by individual parameters (generics). These parameters must never be set to less than 1. Because the timing of these signals must meet 297 MHz timing, routing these signals to a large number of flip-flop clock enables might make it difficult to meet timing. By using multiple identical clock enables, it is easier for the designer to meet timing on the clock enable signals.

The three different types of clock enable signals are described here. The timing diagrams for the various different SDI modes and video formats also show how these signals are asserted relative to the data supplied on the outputs of the receiver module:

- **ce_triple:** These clock enables are valid for all three SDI modes:
 - 3G-SDI: outputs are asserted every other clock cycle (148.5 MHz rate).
 - HD-SDI: outputs are asserted every other clock cycle (74.25 MHz rate).
 - SD-SDI: outputs are asserted, on average, once every 11 clock cycles (27 MHz rate).

Note: For 3G-SDI level B, the `ce_triple` output is asserted at 148.5 MHz, but the output data rate is 74.25 MHz.

- **ce_hd3G:** These clock enables are valid only for HD-SDI and 3G-SDI. For these two modes, the `ce_hd3G` outputs have the same timing as `ce_triple`. There is little reason to use these clock enables instead of the `ce_triple` clock enables, but they are retained for compatibility with older versions of the reference design.
- **dout_rdy_3G:** For 3G-SDI level B, the output data rate is always 74.25 MHz. The `dout_rdy_3G` outputs are asserted High for two clock cycles and Low for two clock cycles in 3G-SDI level B mode, providing a 74.25 MHz output data rate. Downstream devices must take data only when both `dout_rdy_3G` and one of the clock enables (either `triple_ce` or `hd3G_ce`) are both High. The `dout_rdy_3G` outputs are always High in all modes except 3G-SDI level B.

SD-SDI Output Timing

In SD-SDI mode, the RXRECCLK from the GTP RX runs at 297 MHz. The recovered SD-SDI data stream is output on the `ds1_a` port with the Y and C components interleaved at a 27 MHz data rate. The timing signals `trs`, `eav`, and `sav` are also valid.

In SD-SDI mode, the 297 MHz RXRECCLK from the GTP RX is not a recovered clock because the GTP RX is locked to the reference clock and asynchronously samples the input bitstream. RXRECCLK is, therefore, an exact multiple of the reference clock supplied to the

GTP transceiver. If the GTP reference clock is generated by a genlock circuit and is therefore frequency locked to the video signal, RXRECCLK could be considered to be frequency locked to the video as well.

The GTP RX provides the oversampled SD data to the `triple_sdi_rx_light` module, where a DRU recovers the data. The DRU provides a data strobe that is asserted when it has a 10-bit data word ready. The `triple_sdi_rx_light` module outputs this data strobe on the `ce_triple` output in SD-SDI mode. On average, this output is asserted once every 11 clock cycles. The output cadence is not, however, regular in nature. Due to a variety of factors, the number of clock cycles between assertions of the data strobe varies considerably and can be much less or much more than 11 clock cycles. Applications must not rely on the data ready signal being regular in nature. For example, it is quite difficult to directly convert this signal into a 27 MHz clock without using a very low bandwidth PLL.

Figure 5-2 shows the timing of the SD video and timing signals produced by the `triple_sdi_rx_light` module. The outputs only change on the rising edge of `clk` when `ce_triple` is High. The line number output on the `ln_a` port changes to the new line number during the first word of the EAV.

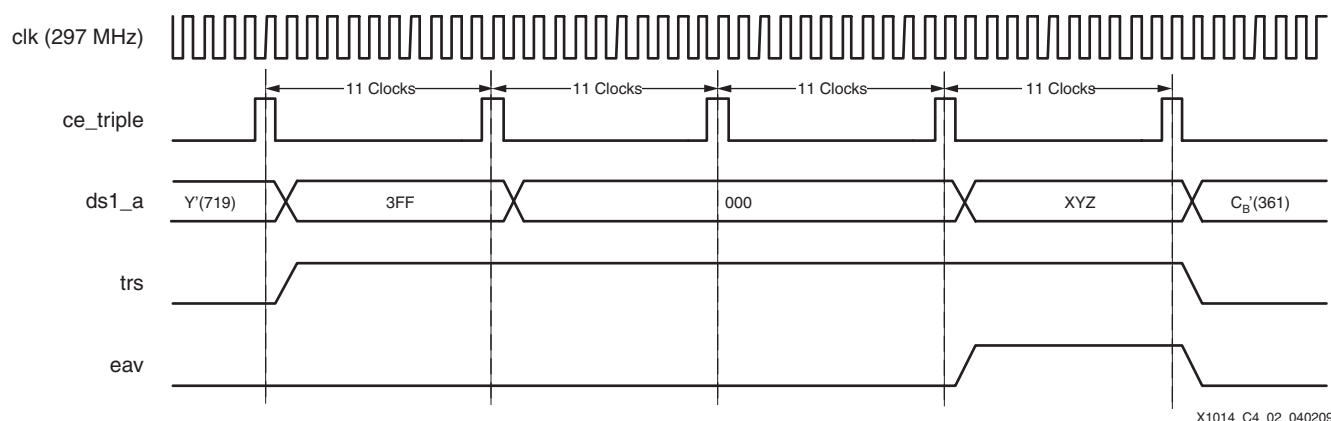


Figure 5-2: SD-SDI Output Timing of `triple_sdi_rx_light` Module

HD-SDI Output Timing

In HD-SDI mode, the RXRECCLK recovered clock from the GTP RX is a true recovered clock and runs at 148.5 MHz (or 148.5/1.001 MHz). The Y and C data streams of the HD-SDI signal are output on the `ds1_a` and `ds2_a` ports, respectively, along with the timing signals `trs`, `eav`, and `sav`. The line number for HD-SDI is output on the `ln_a`. In HD-SDI mode, the line number changes during the CRC1 word.

The `ce_triple` and `ce_hd3G` clock enables are asserted every other clock cycle, producing an output data rate of 74.25 MHz (or 74.1758 MHz). The output ports only change when `ce_triple` and `ce_hd3G` are High.

Figure 5-3 shows the timing of the HD-SDI outputs.

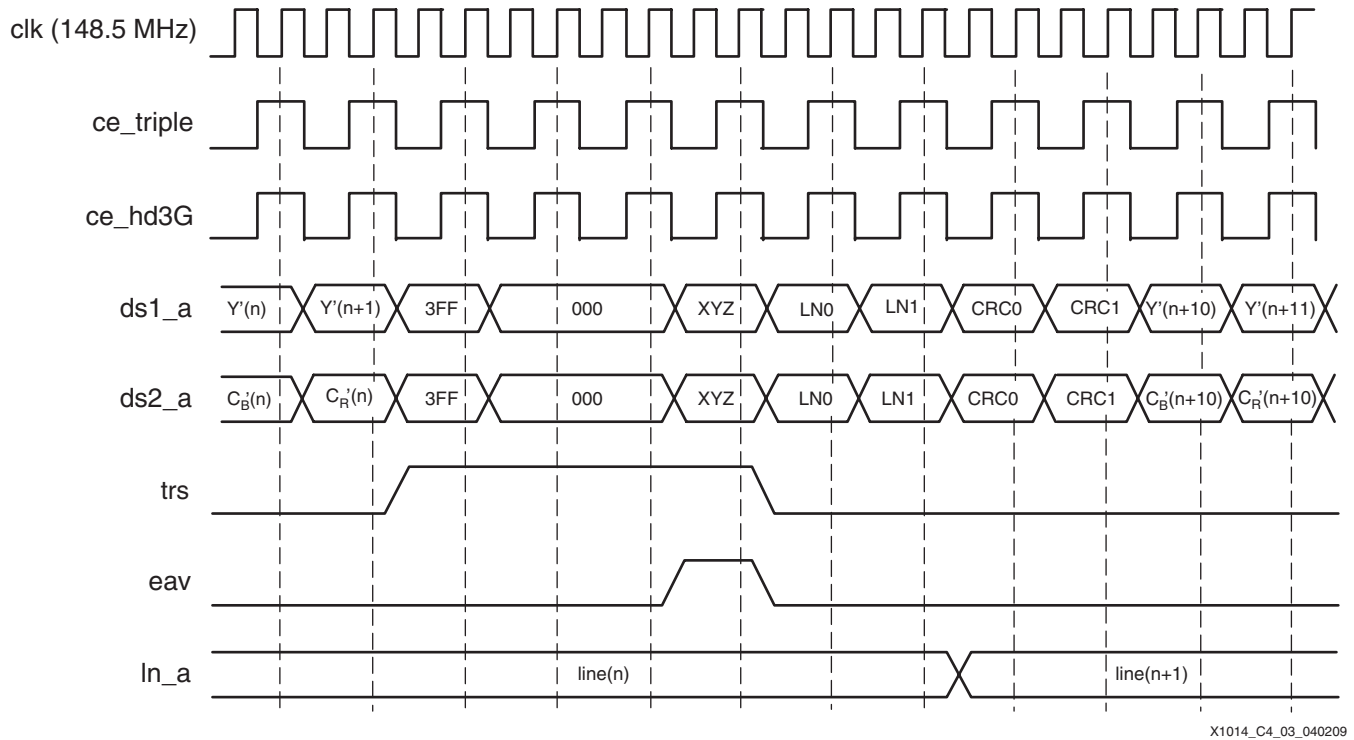
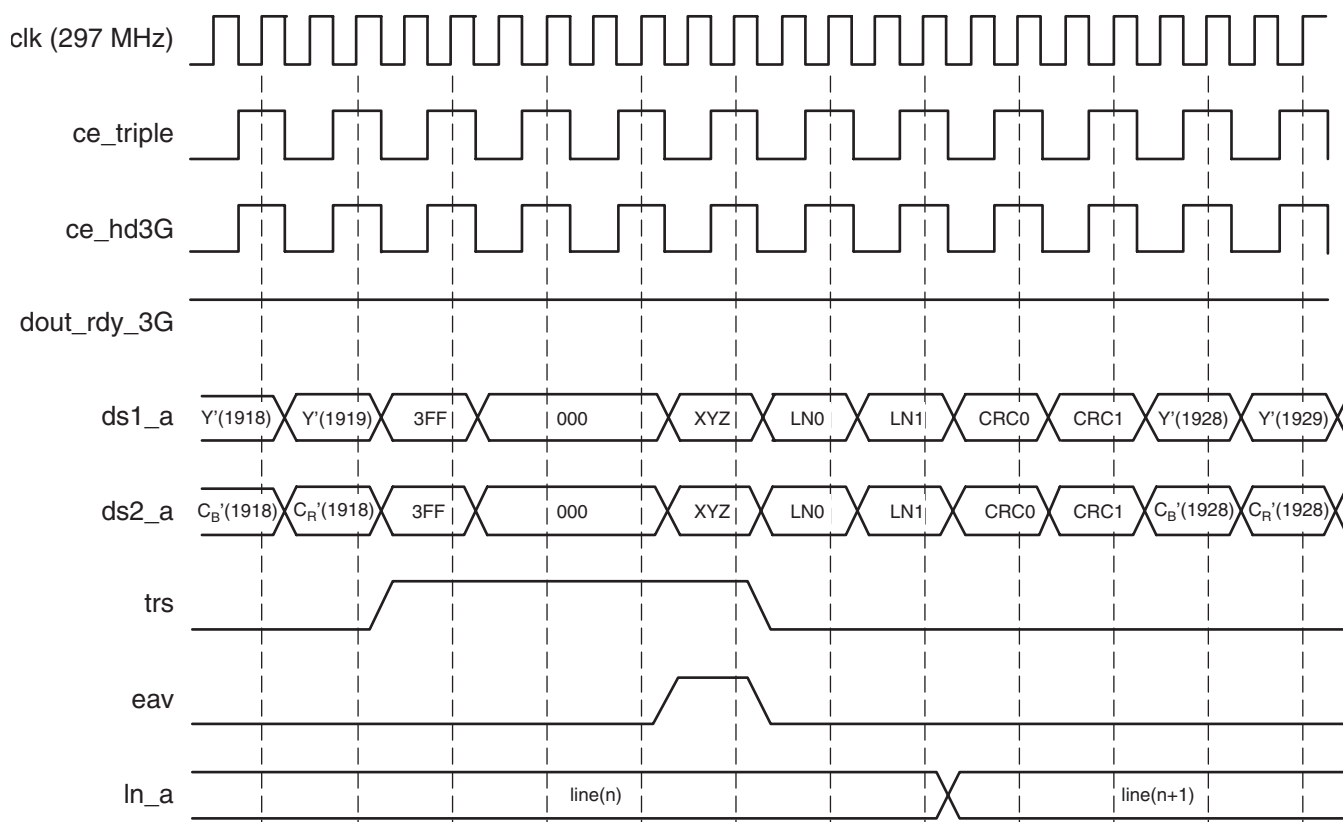


Figure 5-3: HD-SDI Output Timing of `triple_sdi_rx_light` Module

3G-SDI Output Timing

In 3G-SDI mode, the RXRECCLK frequency is 297 MHz (or 297/1.001 MHz). This is a true recovered clock and runs at either two times the video sample rate in level A mode or four times the video sample rate in level B mode. The `ce_triple` and `ce_hd3G` clock enables always run at 148.5 MHz (or 148.5/1.001 MHz) in 3G-SDI mode.

Figure 5-4 shows the output timing of the `triple_sdi_rx_light` module when receiving a 1080p 50, 59.94, or 60 Hz signal in 3G-SDI level A mode. Other video formats such as 4:4:4 10-bit or 12-bit have identical timing, but the video samples are packed per the SMPTE 425M standard, such that it takes two words on each data stream to carry a single video sample. The `triple_sdi_rx_light` module does not unpack the other video formats, but simply outputs them in their packed format. Therefore, the output timing for all video formats in 3G-SDI level A mode is identical to that shown in Figure 5-4, with the two data streams containing packed 3G-SDI level A data streams for all video formats except 1080p 50 Hz, 59.94 Hz, and 60 Hz.



X1014_C4_04_040709

Figure 5-4: 3G-SDI Level A Output Timing (1080p 50 Hz or 60 Hz) of `triple_sdi_rx_light` Module

Figure 5-5 shows the output timing of the `triple_sdi_rx_light` module when receiving a signal in 3G-SDI level B mode. The `dout_rdy_3G` signal runs at 74.25 MHz, indicating when data words are available on the four 10-bit data stream output ports. The outputs change when the clock enables (`ce_triple` and `ce_hd3G`) are High and `dout_rdy_3G` is also High.

Both line number output ports are active. When the level B signal is carrying SMPTE 372M dual link data, the values on both `ln_a` and `ln_b` are identical and indicate the interface line number, not the picture line number. When carrying two independent HD-SDI signals, the two line number ports are not necessarily the same, depending on whether the two HD-SDI signals are vertically synchronized or not. The `ln_a` and `ln_b` ports, while not shown on the diagram, change at the same relative position as they do for level A and HD-SDI—right after the LN0 word.

When SMPTE 372M dual link data is carried on the 3G-SDI level B interface, link A is output on `ds1_a` and `ds2_a`, and link B is output on `ds1_b` and `ds2_b`. These four links are formatted as per SMPTE 372M. The `triple_sdi_rx_light` module does not contain the logic to unpack the SMPTE 372M data streams into native video.

When two independent HD-SDI streams are carried on the 3G-SDI level B interface, the first HD-SDI stream is output on `ds1_a` (Y) and `ds2_a` (C). The second HD-SDI stream is output on `ds1_b` (Y) and `ds2_b` (C). These two HD-SDI streams are horizontally synchronized so that their EAVs and SAVs line up exactly.

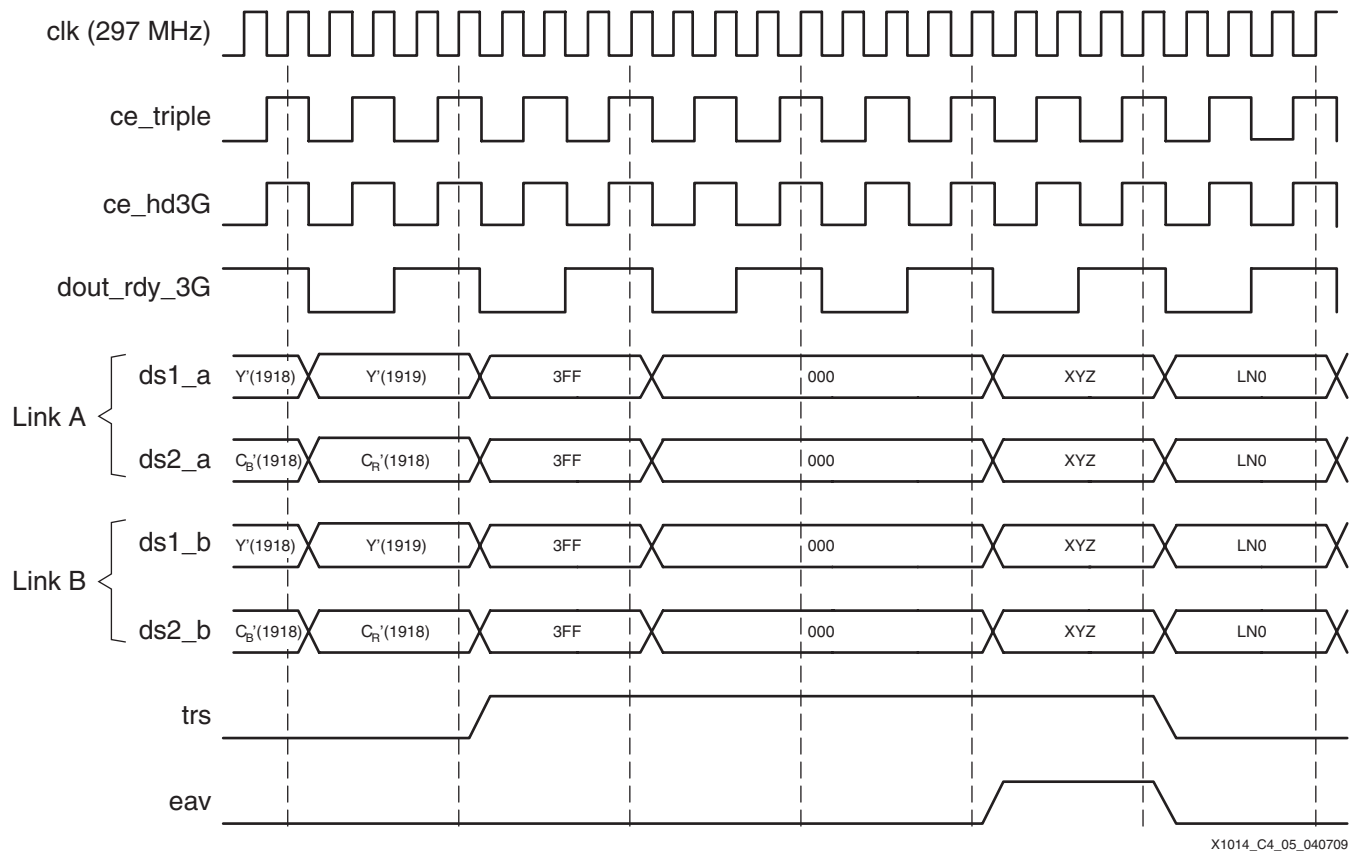


Figure 5-5: 3G-SDI Level B Output Timing of `triple_sdi_rx_light` Module

Full-Featured Triple-Rate SDI Receiver

The `triple_sdi_rx` module contains all of the necessary logic to take the data stream from the GTP receiver and process it into native video for SD-SDI, HD-SDI, 3G-SDI, and dual link HD-SDI. It has all of the features of the `triple_sdi_rx_light` module plus these features:

- 3G-SDI level A is unpacked into native video with full support for all video formats specified by SMPTE 425M.
- 3G-SDI level B carrying SMPTE 372M dual link HD-SDI formatted data streams is unpacked into native video with full support for all video formats specified by SMPTE 372M.
- EDH error checking and video flywheel for SD-SDI are supported.
- Ancillary data stream outputs with timing signals for HD-SDI, 3G-SDI, and dual link HD-SDI are supported. Dual link HD-SDI and 3G-SDI data unpacking can destroy ANC packets, so these ancillary data ports provide the preserved data streams prior to unpacking so that the ancillary data can be captured. These ports are valid for any SDI mode.

The unpacking process used to recover the native video that has been mapped according to the rules of SMPTE 372M and SMPTE 425M into 3G-SDI (both level A and level B) and dual link HD-SDI data streams can result in illegal video values in the blanking intervals on the native video outputs of the triple-rate SDI receiver. This applies to both the horizontal blanking interval and the active portion of lines in the vertical blanking interval.

This is the normal result of the unpacking process specified by the SMPTE standards. This does not apply to 1080p 50 Hz, 59.94 Hz, and 60 Hz video, but does apply to the other video formats that have more than 20 bits per video sample. For example, when the video format is 4:4:4 12-bit $Y'C_B'C_R'$ carried on 3G-SDI level A, the transmitter might have filled the blanking intervals of the formatted data streams with values of 0x040 and 0x200 (black-level video) anywhere that ANC packets are not present to make these blank samples contain legal SDI values. However, when these data streams are unpacked into native video, the result is values of 0x000 on the Y' and C_B' components, and 0x208 on the C_R' component. These are obviously neither black level values nor legal values. If the application requires legal black level values in the blanking intervals, it must force the components to the desired values after the triple-rate SDI receiver module.

For the same reason, any ANC packets in the blanking intervals do not survive the 3G-SDI and dual link HD-SDI unpacking processes for video formats with more than 20 bits per video sample. This is why the triple-rate SDI receiver has separate ANC output ports where the ANC packets are intact.

[Figure 5-6](#) shows a simplified top-level block diagram of a full-featured triple-rate SDI receiver not configured for dual link HD-SDI.

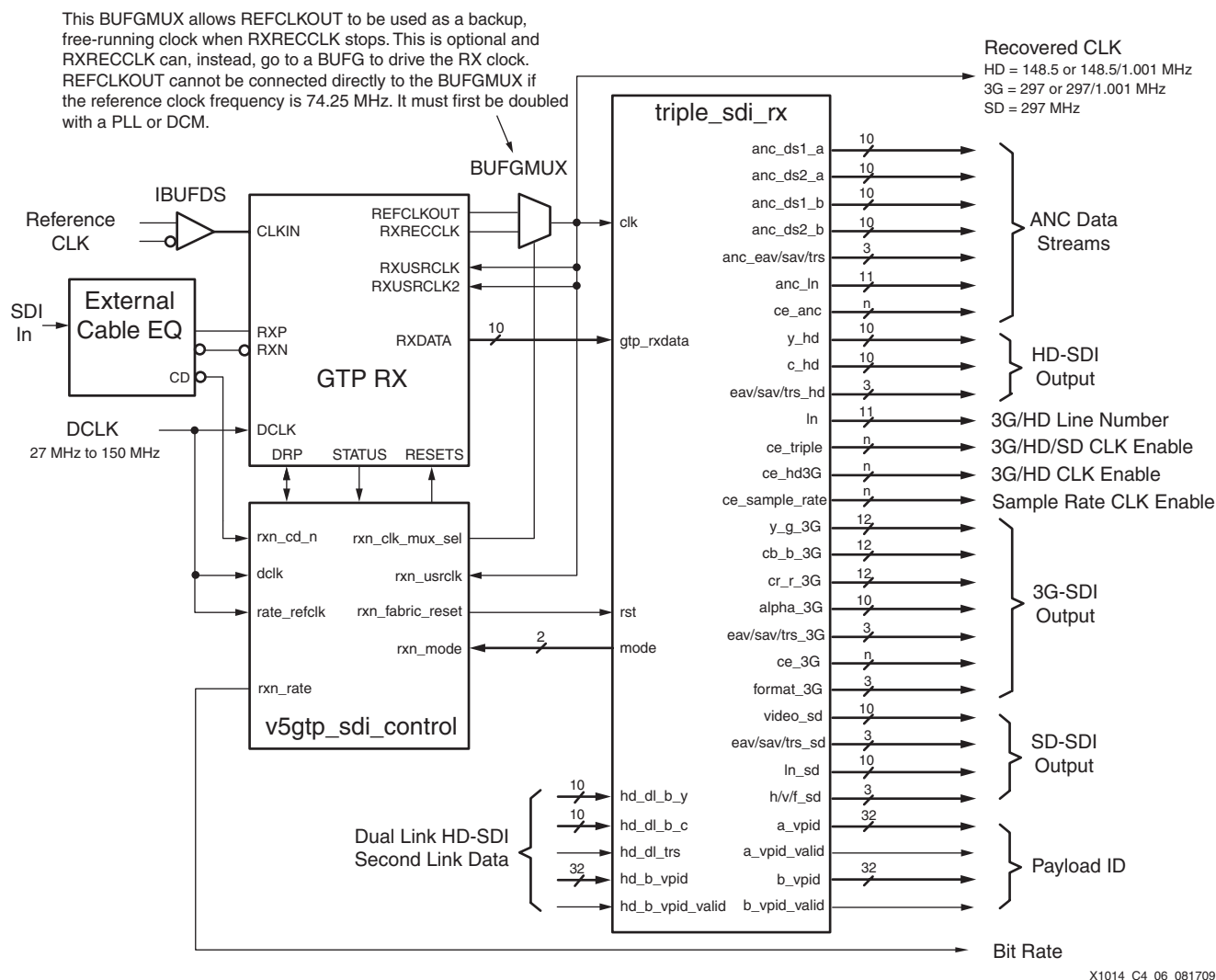


Figure 5-6: Block Diagram of Full-Featured Triple-Rate SDI Receiver

When combined with a GTP RX and the v5gtp_sdi_control module, the triple_sdi_rx module implements a full-featured triple-rate SDI receiver. It supports all video formats and both levels of 3G-SDI with full unpacking of all 3G-SDI level A and B video formats into native video. It also includes a full SD-SDI EDH processor. The SD-SDI feature set is configurable through the SD_MODE parameter, allowing trade-offs to be made between functionality and design size. The video components and main video timing signals (TRS, EAV, and SAV) are output from the module on separate sets of outputs—one set for each SDI standard (SD, HD, and 3G). Some applications might need these signals combined into one common set of signals. This can be done externally to the module with multiplexers or the module can be customized for such applications.

Table 5-5 describes the ports of the triple_sdi_rx module.

Table 5-5: Ports of `triple_sdi_rx` Module

Port Name	I/O	Width	Description
Inputs			
clk	In	1	The recovered clock input connects to the global or regional clock that drives the RXUSRCLK and RXUSRCLK2 clock inputs of the GTP RX. This clock is usually driven by the RXRECCLK output of the GTP RX. The clock frequency must be 297 MHz (or 297/1.001 MHz) for 3G-SDI, 148.5 MHz (or 148.5/1.001 MHz) for HD-SDI, and 297 MHz for SD-SDI.
rst	In	1	This input is an asynchronous reset. The falling edge of this reset signal must meet the reset recovery time of all flip-flops relative to the next rising edge of clk. This input can be driven by the rx#_fabric_reset output of the v5gtp_sdi_control module, which when combined with proper timing constraints, results in proper timing of the reset signal to the synchronous elements it resets.
gtp_rxdata	In	10	This input connects to the 10-bit RXDATA port of the GTP wrapper module.
frame_en	In	1	The framer automatically readjusts the word alignment to match the alignment of each TRS (EAV or SAV) when this input is High. This input can be used to implement TRS alignment filtering. For example, if the nsp output is connected to the frame_en input, the framer ignores a single misaligned TRS, keeping the existing word alignment until the new word alignment is confirmed by a second matching TRS. If TRS filtering is not desired, this input must be tied High. Also, it is important to turn off any TRS filtering on the synchronous switching line by driving the frame_en input High on the synchronous switching lines.
force_dl_mode	In	1	<p>When the current mode is 3G-SDI level B or dual link HD-SDI, this input, when High, forces the data streams to be evaluated as SMPTE 372M dual link HD-SDI data streams, even if the SMPTE 352M packets are missing or indicate data streams that are not SMPTE 372M streams. When this input is High, the dl_default_mapping port determines how the streams are unpacked. For 3G-SDI level B, the SMPTE 352M packets are required and should always be present. SMPTE 372M also requires these packets for dual link HD-SDI, but they are often omitted by older equipment.</p> <p>Note: When the SDI mode is HD-SDI and force_dl_mode is High, the module runs in dual link mode. It is therefore important to only assert this input High in HD-SDI mode if dual link mode is the intended mode of operation.</p>

Table 5-5: Ports of triple_sdi_rx Module (Cont'd)

Port Name	I/O	Width	Description
dl_default_mapping	In	2	<p>Normally, the triple-rate SDI receiver uses the information from the SMPTE 352M VPID packets to determine which unpacking algorithm to use. However, if VPID packets are not found, the receiver is not able to correctly unpack the data streams into native video unless the mapping algorithm is specified on the dl_default_mapping input port and force_dl_mode is High. The value on this port is used to specify the SMPTE 372M mapping algorithm regardless of whether the SMPTE 372M data streams are carried by 3G-SDI level B or regular dual link HD-SDI interfaces. Legal 3G-SDI data streams always have SMPTE 352M packets, but dual link HD-SDI data streams often do not have these packets—although the packets are required by the SMPTE 372M standard. The encoding of this port is:</p> <ul style="list-style-type: none"> 00 = 4:2:2 10-bit Y'C_B'C_R' 1080p 50 Hz, 59.94 Hz, or 60 Hz 01 = 4:4:4 or 4:4:4:4 10-bit Y'C_B'C_R' or R'G'B' 10 = 4:4:4 12-bit Y'C_B'C_R' or R'G'B' 11 = 4:2:2 or 4:2:2:4 12-bit Y'C_B'C_R' (optional 10-bit alpha component)
dl_crc_ln_blank	In	1	<p>When this input of the port is High, the CRC and LN words of each line are replaced with blanking level data on the native video component outputs. This applies only to SMPTE 372M data streams when carried by either 3G-SDI level B or dual link HD-SDI. The video unpacking process scrambles the CRC and LN words, so blanking them ensures that legal values are present in the video component output data streams. Normally, this input should be tied High.</p>
clr_edh_err	In	1	<p>For SD-SDI only, this input, when asserted High, clears the EDH error counter in the EDH processor to zero, thereby clearing the edh_err output. This is a synchronous input and only clears the EDH error counter on the rising edge of clk when the ce_triple output is High.</p>
Inputs From Paired Dual Link RX in SMPTE 372M Dual Link HD-SDI Mode			
hd_dl_b_y	In	10	<p>The master RX of a dual link HD-SDI RX pair must have this port driven by the y_hd port of the other RX of the pair.</p>
hd_dl_b_c	In	10	<p>The master RX of a dual link HD-SDI RX pair must have this port driven by the c_hd port of the other RX of the pair.</p>
hd_dl_trs	In	10	<p>The master RX of a dual link HD-SDI RX pair must have this port driven by the trs_hd port of the other RX of the pair.</p>
hd_b_vpid	In	32	<p>The master RX of a dual link HD-SDI RX pair must have this port driven by the b_vpid port of the other RX of the pair.</p>
hd_b_vpid_valid	In	1	<p>The master RX of a dual link HD-SDI RX pair must have this port driven by the b_vpid_valid port of the other RX of the pair.</p>

Table 5-5: Ports of `triple_sdi_rx` Module (Cont'd)

Port Name	I/O	Width	Description
Clock Enable Outputs			
ce_triple	Out	NUM_TRIPLE_CE	These clock enables are valid for all three SDI modes, although for some video formats these clock enables are not asserted at the video sample rate. These clock enables are asserted at half the clk frequency for HD-SDI and 3G-SDI, and at 27 MHz for SD-SDI. The number of identical clock enables provided on this port is specified by the NUM_TRIPLE_CE parameter/generic.
ce_sample_rate	Out	NUM_SR_CE	These clock enables are valid for all three SDI modes. They always run at the output video sample rate of the unpacked native video. The number of identical clock enables provided on this port is specified by the NUM_SR_CE parameter/generic.
ce_hd3G	Out	NUM_HD3G_CE	These clock enables are valid for HD-SDI and 3G-SDI, but not for SD-SDI. These outputs are always asserted at half the clk frequency. The number of identical clock enables provided on this port is specified by the NUM_HD3G_CE parameter/generic.
ce_3G	Out	NUM_3G_CE	These clock enables run at the 3G-SDI native video output sample rate. The number of these identical data ready signals is controlled by the NUM_3G_CE parameter/generic.
ce_anc	Out	NUM_ANC_CE	These clock enables run at the ancillary data stream rate. The number of these identical clock enables is controlled by the NUM_ANC_CE parameter/generic.
Outputs Valid for All Modes			
mode	Out	2	<p>This port indicates the current SDI mode of the receiver.</p> <ul style="list-style-type: none"> 00 = HD-SDI 01 = SD-SDI 10 = 3G-SDI <p>The mode port changes value as the receiver searches for the correct mode. During this time, the mode_locked output is Low. When the receiver detects the correct SDI mode matching the input data stream, the mode_locked output is set High.</p> <p>Note: When using the <code>v5gtp_sdi_control</code> module, this output port must drive the <code>rx#_mode</code> input of the <code>v5gtp_sdi_control</code> module for the particular RX unit in the GTP_DUAL tile used to implement this SDI receiver.</p>
mode_HD mode_SD mode_3G	Out	1	These three output ports are decoded unary versions of the mode port and are provided for convenience. Unlike the mode output that changes continuously as the receiver seeks to identify and lock to the incoming signal, these outputs are all forced Low when the receiver is not locked. The mode output matching the current operating mode of the receiver is High when mode_locked is High.
mode_locked	Out	1	When this output is Low, the receiver is actively searching for the SDI mode that matches the input data stream. During this time, the mode output port changes frequently. When the module locks to the correct SDI mode, the mode_locked output goes High.

Table 5-5: Ports of `triple_sdi_rx` Module (Cont'd)

Port Name	I/O	Width	Description
t_format	Out	4	This output port indicates the video transport timing format of the SDI signal. See Table 5-6, page 159 for encoding. The output port is only valid when rx_locked is High. If the SD_MODE parameter is set to MINIMAL, this output is not valid in SD-SDI mode.
rx_locked	Out	1	This output is High when the receiver is locked to the incoming signal. This output is driven by the transport format detector that generates the t_format output. If the SD_MODE parameter is set to MINIMAL, there is no SD-SDI transport format detector so rx_locked is set to be equal to the mode_locked output in SD-SDI mode. Because this output is driven by the transport format detector, there can be more than one video frame time of delay from when the receiver is actually locked to the input signal until rx_locked is asserted. During this time, the transport format detector is determining the transport format.
nsp	Out	1	When this output is High, it indicates that the framer has detected a TRS at a new word alignment. If frame_en is High, this output is only asserted briefly. If frame_en is Low, this output remains asserted until the framer is allowed to readjust to the new TRS alignment.
SMPTE 352M Payload ID Outputs Valid in All Modes			
a_vpid	Out	32	All four user data bytes of the SMPTE 352M packet from data stream 1 are output on this port in the following format: byte4, byte3, byte2, byte1 (MSB to LSB.)
a_vpid_valid	Out	1	This output is High if a_vpid is valid.
p_transport	Out	1	This output is High if the transport is progressive and Low if it is interlaced. This output is generated from the a_vpid data and is valid only when a_vpid_valid is High. The transport is interlaced while the picture is progressive for progressive segmented frame video formats and for 1080p 50 Hz or 60 Hz video carried in SMPTE 372M data streams.
p_picture	Out	1	This output is High if the picture is progressive and Low if it is interlaced. This output is generated from the a_vpid data and is valid only when a_vpid_valid is High.

Table 5-5: Ports of `triple_sdi_rx` Module (Cont'd)

Port Name	I/O	Width	Description
sample_struct	Out	4	<p>This is the video component sampling structure. This output is generated from the <code>a_vpid</code> data and is valid only when <code>a_vpid_valid</code> is High.</p> <p>0x0 = 4:2:2 (Y'C_B'C_R') 0x1 = 4:4:4 (Y'C_B'C_R') 0x2 = 4:4:4 (R'G'B') 0x3 = 4:2:0 0x4 = 4:2:2:4 (Y'C_B'C_R'A) 0x5 = 4:4:4:4 (Y'C_B'C_R'A) 0x6 = 4:4:4:4 (R'G'B'A) 0x7 = Reserved 0x8 = 4:2:2:4 (Y'C_B'C_R'D) 0x9 = 4:4:4:4 (Y'C_B'C_R'D) 0xA = 4:4:4:4 (R'G'B'D) 0xB = Reserved 0xC = Reserved 0xD = Reserved 0xE = 4:4:4 (X'Y'Z') 0xF = Reserved</p>
p_rate	Out	4	<p>This is the picture update rate (Hz). This output is generated from the <code>a_vpid</code> data and is valid only when <code>a_vpid_valid</code> is High.</p> <p>0x0 = No defined value 0x1 = Reserved 0x2 = 24/M 0x3 = 24 0x4 = Reserved 0x5 = 25 0x6 = 30/M 0x7 = 30 0x8 = Reserved 0x9 = 50 0xA = 60/M 0xB = 60 0xC through 0xF = Reserved</p>
dynamic_range	Out	2	<p>This is the video component dynamic range code. This output is generated from the <code>a_vpid</code> data and is valid only when <code>a_vpid_valid</code> is High.</p> <ul style="list-style-type: none"> 00 = 100% 01 = 200% 10 = 400% 11 = Reserved

Table 5-5: Ports of triple_sdi_rx Module (Cont'd)

Port Name	I/O	Width	Description
bit_depth	Out	1	This output is asserted High for one sample period when a CRC error is detected on the previous video line. For 3G-SDI level B, this output indicates CRC errors on data stream 1 only. There is a second output called crc_err2 that indicates CRC errors on data stream 2 for 3G-SDI level B.
Outputs Valid for HD-SDI and 3G-SDI Modes			
crc_err	Out	1	This output is asserted High for one sample period when a CRC error is detected on the previous video line. For 3G-SDI level B, this output indicates CRC errors on data stream 1 only. There is a second output called crc_err2 that indicates CRC errors on data stream 2 for 3G-SDI level B.
ln	Out	11	This output port always indicates the current picture line number. For those interfaces where the line number carried on the interface differs from the picture line number, the picture line number (not the interface line number) is provided on this port. The line number changes at some point between the first word of the EAV and the last word of the CRC that follows the EAV.
HD-SDI Outputs			
y_hd	Out	10	This port outputs the Y data stream from the HD-SDI receiver. It carries the Y' component for HD video.
c_hd	Out	10	This port outputs the C data stream from the HD-SDI receiver. It carries the multiplexed C _B ' and C _R ' components for HD video.
eav_hd	Out	1	This output is asserted High for one sample time when the XYZ word of an EAV is present on the y_hd and c_hd ports.
sav_hd	Out	1	This output is asserted High for one sample time when the XYZ word of an SAV is present on the y_hd and c_hd ports.
trs_hd	Out	1	This output is asserted High for four sample times as all four words of an EAV or SAV are output on the y_hd and c_hd ports.
3G-SDI Outputs (Some also Used for Dual Link HD-SDI)			
y_g_3G	Out	12	Level A (all video formats): Y' or G' component Level B and dual link HD-SDI: Y' or G' component Level B (non dual link data): Y component of HD-SDI stream 1
cb_b_3G	Out	12	Level A (4:4:4 formats): C _B ' or B' component Level A (4:2:2 formats): C _B ' and C _R ' components interleaved Level B and dual link HD-SDI (372M with 4:4:4 formats): C _B ' or B' component Level B and dual link HD-SDI (372M with 4:2:2 formats): C _B ' and C _R ' components interleaved Level B (non dual link): C component of HD-SDI stream 1

Table 5-5: Ports of `triple_sdi_rx` Module (Cont'd)

Port Name	I/O	Width	Description
cr_r_3G	Out	12	Level A (4:4:4 formats): C_R' or R' component Level A (4:2:2 formats): not used Level B and dual link HD-SDI (372M with 4:4:4 formats): C_R' or R' component Level B and dual link HD-SDI (372M with 4:2:2 formats): not used Level B (non dual link): C component of HD-SDI stream 2
alpha_3G	Out	10	Level A (4:4:4 formats): alpha (if present) Level A (4:2:2 formats): not used Level B and dual link HD-SDI (372M with 4:4:4 formats): alpha (if present) Level B and dual link HD-SDI (372M with 4:2:2 formats): not used Level B (non dual link): Y component of HD-SDI stream 2
level_b_3G	Out	1	This output is asserted High when the input signal is level B compliant and Low when it is level A compliant. This output is only valid in 3G-SDI mode.
format_3G	Out	3	This output indicates the video format determined from the SMPTE 352M payload ID packets. This output port can be used to determine how to interpret the data on the y_g_3G, cb_b_3G, cr_r_3G, and alpha_3G ports. <ul style="list-style-type: none"> 000 = Invalid video format 001 = 4:4:4 $R'G'B'$ or 4:4:4 $R'G'B'A$ 10-bit 010 = 4:4:4 $R'G'B'$ 12-bit 011 = 3G-SDI level B carrying two independent HD-SDI streams 100 = 4:2:2 $Y'C_B'C_R'$ 1080p 60Hz, 60/M Hz, or 50 Hz 101 = 4:4:4 $Y'C_B'C_R'$ or 4:4:4 $Y'C_B'C_R'A$ 10-bit 110 = 4:4:4 $Y'C_B'C_R'$ or $X'Y'Z'$ 12-bit 111 = 4:2:2 $Y'C_B'C_R'$ or 4:2:2 $Y'C_B'C_R'A$ 12-bit
trs_3G	Out	1	This output is asserted High for four sample times as all four words of an EAV or SAV are output. This output is also active for dual link HD-SDI.
eav_3G	Out	1	This output is asserted High for one sample time as the XYZ word of an EAV is output. This output is also active for dual link HD-SDI.
sav_3G	Out	1	This output is asserted High for one sample time as the XYZ word of an SAV is output. This output is also active for dual link HD-SDI.
alpha_act_3G	Out	1	This output is High if the alpha component is active. This output is also valid for dual link HD-SDI.
skew_error_lv1b	Out	1	When SMPTE 372M streams are carried on 3G-SDI level B or dual link HD-SDI, this output is High if the skew between the two streams exceeds the capabilities of the receiver.

Table 5-5: Ports of triple_sdi_rx Module (Cont'd)

Port Name	I/O	Width	Description
dl_active	Out	1	This output is High if the 3G-SDI level B data streams contain SMPTE 372M dual link HD-SDI data or when the module is receiving SMPTE 372M streams by dual link HD-SDI. This output is generated by decoding the SMPTE 352M VPID data. In HD-SDI mode, if force_dl_mode is High, this output is driven High.
crc_err2	Out	1	For 3G-SDI level B only, this output indicates CRC errors on data stream 2. The crc_err output indicates CRC errors on data stream 1. This output is not valid in dual link HD-SDI mode.
b_vpid	Out	32	All four bytes of VPID data from data stream 2 are output on this port in the following format: byte4, byte3, byte2, byte1 (MSB to LSB). This output is not valid in dual link HD-SDI mode.
b_vpid_valid	Out	1	This output is High if the b_vpid data is valid. This output is not valid in dual link HD-SDI mode.
SD-SDI Outputs			
video_sd	Out	10	Multiplexed Y/C components of the SD video signal are output on this port.
eav_sd	Out	1	This output is asserted High for one sample time as the XYZ word of an EAV is output on video_sd.
sav_sd	Out	1	This output is asserted High for one sample time as the XYZ word of an SAV is output on video_sd.
trs_sd	Out	1	This output is asserted High for four sample times as all four words of an EAV or SAV are output on video_sd.
ln_sd	Out	10	This output is the current video line number. This output is only valid if SD_MODE is set to EDH, which enables the video flywheel that generates the SD line numbers.
edh_err	Out	1	This output is asserted High whenever the error counter in the EDH processor is not equal to zero. The error counter is cleared to zero when the clr_edh_err input is asserted High. This output is only valid if SD_MODE is set to EDH.
f_sd	Out	1	This output is the field indicator. It is Low for field 1 and High for field 2. This output is generated by the flywheel in the EDH processor and is only valid if SD_MODE is set to EDH.
v_sd	Out	1	This is the vertical sync signal. It is High during the vertical blanking interval. This output is generated by the flywheel in the EDH processor and is only valid if SD_MODE is set to EDH.
h_sd	Out	1	This is the horizontal sync signal. It is High during the horizontal blanking interval. This output is generated by the flywheel in the EDH processor and is only valid if SD_MODE is set to "EDH".
Raw Data Streams for ANC Processing			
anc_ds1_a	Out	10	This is the data stream 1 (Y) for HD-SDI and 3G-SDI level A. It is the data stream 1 of link A for 3G-SDI level B and dual link HD-SDI. It is also the multiplexed Y/C data stream for SD-SDI.

Table 5-5: Ports of `triple_sdi_rx` Module (Cont'd)

Port Name	I/O	Width	Description
anc_ds2_a	Out	10	This is the data stream 2 (C) for HD-SDI and 3G-SDI level A. It is the data stream 2 of link A for 3G-SDI level B and dual link HD-SDI. This output is not valid for SD-SDI.
anc_ds1_b	Out	10	This output is valid only for 3G-SDI level B and dual link HD-SDI. It is the data stream 1 (Y) of link B.
anc_ds2_b	Out	10	This output is valid only for 3G-SDI level B and dual link HD-SDI. It is the data stream 2 (C) of link B.
anc_trs	Out	1	This output is asserted High for four sample times as all four words of an EAV or SAV are output on the ANC data stream ports.
anc_eav	Out	1	This output is asserted High for one sample time when the XYZ word of an EAV is output on the ANC data stream ports.
anc_sav	Out	1	This output is asserted High for one sample time when the XYZ word of an SAV is output on the ANC data stream ports.
anc_b_active	Out	1	This output is asserted High when the anc_ds1_b and anc_ds2_b carry valid data. This output is only asserted High in SMPTE 372M dual link HD-SDI mode and in 3G-SDI level B mode.
anc_ln	Out	11	This output specifies the line number for ANC processing. This output is only guaranteed to be valid during the horizontal blanking interval, starting with the first HANC word following the last CRC word after the EAV, and continuing through the last word of the horizontal blanking interval. For those 3G-SDI level B and dual link HD-SDI formats where the interface line number is different from the picture line number, this value is the interface line number, not the picture line number. This output is only valid in SD-SDI mode if the EDH processor is included in the design.

Table 5-6 shows the coding of the `t_format` field. This field indicates the transport format (not always the same as the picture format) calculated by the receiver by counting words per line and active lines per field or frame. The `t_format` field can uniquely identify most video transport formats, but cannot distinguish between transport formats that have identical timing. For example, it cannot differentiate between 1080i 60 Hz and 1080PsF 30 Hz (1080p 30 Hz video carried on a 1080i 60 Hz transport) because there is no difference in the timing. For a complete interpretation of the video format, the SMPTE 352M VPID information must be consulted when the transport timing is ambiguous.

Table 5-6: **t_format** Output Port Encoding for **triple_sdi_rx** Module

SDI Mode	t_format	Standard	Video Format
SD-SDI	0000	SMPTE 125M	NTSC 4:2:2
	0001		Invalid
	0010	SMPTE 267M	NTSC 4:2:2 16 x 9 widescreen
	0011	SMPTE RP 174	NTSC 4:4:4 13.5 MHz sample rate
	0100	ITU-R BT.656	PAL 4:2:2
	0101		Invalid
	0110	ITU-R BT.601	PAL 4:2:2 16 x 9 widescreen
	0111	ITU-R BT.799	PAL 4:4:4 13.5 MHz sample rate
HD-SDI and 3G-SDI	0000	SMPTE 260M	1035i 59.94 Hz and 60 Hz
	0001	SMPTE 295M	1080i 50 Hz
	0010	SMPTE 274M	1080i 59.94 Hz and 60 Hz 1080PsF 29.97 Hz and 30 Hz
	0011	SMPTE 274M	1080i 50 Hz 1080PsF 25 Hz
	0100	SMPTE 274M	1080p 29.97 Hz and 30 Hz 1080p 59.94 Hz and 60 Hz (3G-SDI level B only)
	0101	SMPTE 274M	1080p 25 Hz 1080p 50 Hz (3G-SDI level B only)
	0110	SMPTE 274M	1080p 23.98 Hz and 24 Hz
	0111	SMPTE 296M	720p 59.94 Hz and 60 Hz
	1000	SMPTE 274M	1080PsF 23.98 Hz and 24 Hz
	1001	SMPTE 296M	720p 50 Hz
	1010	SMPTE 296M	720p 29.97 Hz and 30 Hz
	1011	SMPTE 296M	720p 25 Hz
	1100	SMPTE 296M	720p 23.98 Hz or 24 Hz
	1101	SMPTE 274M	1080p 59.94 Hz and 60 Hz (3G-SDI level A only)
	1110	SMPTE 274M	1080p 50 Hz (3G-SDI level A only)

In dual link HD-SDI and 3G-SDI level B modes, the 1080p 50 and 60 Hz video formats are carried on an interlaced transport. Therefore, the module reports them as being 1080i 50 Hz and 1080i 60 Hz, respectively. However, in 3G-SDI level A mode, these two video formats are carried on a progressive transport and are uniquely identified as 1080p 50 Hz and 1080p 60 Hz (codes 1110 and 1101 respectively).

Table 5-7 describes the parameters/generics of the `triple_sdi_rx` module.

Table 5-7: Parameters of triple_sdi_rx Module

Parameter	Default Value	Description
NUM_TRIPLE_CE	2	This parameter specifies the number of identical ce_triple clock enable signals provided by the module. This parameter should never be set to less than 1.
NUM_SR_CE	2	This parameter specifies the number of identical ce_sample_rate clock enable signals provided by the module. This parameter should never be set to less than 1.
NUM_HD3G_CE	2	This parameter specifies the number of identical ce_hd3G clock enable signals provided by the module. This parameter should never be set to less than 1.
NUM_3G_CE	2	This parameter specifies the number of identical ce_3G clock enable signals provided by the module. This parameter should never be set to less than 1.
NUM_ANC_CE	2	This parameter specifies the number of identical ce_anc clock enable signals provided by the module. This parameter should never be set to less than 1.
ERRCNT_WIDTH	4	This is a parameter of the SDI mode detection module. It specifies the number of bits used in the error counter. This counter must be wide enough to support counting up to the values provided by the MAX_ERRS_LOCKED and MAX_ERRS_UNLOCKED parameters. This error counter is internal to the SDI mode detector and is not the same as the SD-SDI EDH error counter.
MAX_ERRS_LOCKED	15	This is a parameter of the SDI mode detection module. It specifies the maximum number of consecutive lines with errors allowed while the receiver is locked to the SDI mode before the receiver moves to the unlocked state and begins searching for the correct SDI mode.
MAX_ERRS_UNLOCKED	2	This is a parameter of the SDI mode detection module. It specifies the maximum number of consecutive lines with errors allowed while the receiver is searching for the correct SDI mode before it continues the search by moving to another mode.

Table 5-7: Parameters of `triple_sdi_rx` Module (Cont'd)

Parameter	Default Value	Description
SD_MODE	"EDH"	<p>This parameter specifies what type of SD-SDI processing is included in the <code>triple_sdi_rx</code> module. The options are:</p> <ul style="list-style-type: none"> "EDH": This includes the full EDH processor and flywheel. All SD outputs are valid. "AUTODETECT": This includes just the video format detector, allowing the <code>t_format</code> and <code>rx_locked</code> outputs of the receiver to work for SD mode. The <code>edh_err</code>, <code>ln_sd</code>, <code>f_sd</code>, <code>v_sd</code>, and <code>h_sd</code> outputs are not valid in this mode. "MINIMAL": This includes no additional SD processing. The <code>t_format</code> and <code>rx_locked</code> outputs are not valid for SD-SDI in this mode. The <code>edh_err</code>, <code>ln_sd</code>, <code>f_sd</code>, <code>v_sd</code>, and <code>h_sd</code> outputs are not valid in this mode.

Full-Featured Triple-Rate Receiver Clocking

The clock input (`clk`) to the `triple_sdi_rx` module must come from a global (or regional) clock buffer driven by the recovered clock from the GTP receiver (`RXRECCLK`). As described in [Chapter 3, Implementing SMPTE Serial Digital Interfaces with RocketIO GTP Transceivers](#), this clock might stop if the SDI serial input bitstream stops, so the designer might want to use a `BUFGMUX` as the global clock buffer. The `v5gtp_sdi_control` module has an output to control a `BUFGMUX` so that a backup, free-running reference clock can be supplied in place of `RXRECCLK` when `RXRECCLK` is stopped. This is entirely optional and is not required for all applications.

Note: If the reference clock frequency is 74.25 MHz or 74.25/1.001 MHz, the `REFCLKOUT` in the GTP transceiver will not be the correct frequency to be used directly as an input to the `BUFGMUX` (as shown in [Figure 5-1, page 137](#)). This is because `REFCLKOUT` is always exactly the same frequency as the reference clock input. The frequency of the backup clock into the `BUFGMUX` must be either 148.5 MHz or 297 MHz. Thus, a DCM or PLL would be required to double a 74.25 MHz `REFCLKOUT` to 148.5 MHz, unless another suitable backup clock is available in the FPGA. However, a single 74.25 MHz `REFCLKOUT` from any GTP RX in the FPGA, doubled to 148.5 MHz, can be used as the backup clock for any number of SDI receivers by connecting it to the `BUFGMUX` of each receiver.

The frequency of the recovered clock depends on the current SDI mode of the receiver. In HD-SDI mode, the frequency is 148.5 MHz (or 148.5/1.001 MHz). For 3G-SDI, it is 297 MHz or 297/1.001 MHz, and for SD-SDI, it is always 297 MHz.

Internally, most of the `triple_sdi_rx` module runs at half the recovered clock frequency for HD-SDI and 3G-SDI (some portions run at one quarter the recovered clock frequency in 3G-SDI mode, depending on the video format). For SD-SDI, the internal data rate is 27 MHz. Clock enables are used in the `triple_sdi_rx` module to run the various sections at the proper rates.

The `triple_sdi_rx` module also supplies various clock enables on output ports. These can be used, in conjunction with the recovered clock, to clock logic downstream from the `triple_sdi_rx` module at the correct data rate. There are five such signals. The number of each type is controlled by individual parameters (generics). These parameters must never be set to less than 1. Because the timing of these signals must meet 297 MHz timing, routing these signals to a large number of flip-flop clock enables might make it difficult to

meet timing. By using multiple identical clock enables, it is easier to meet timing on the clock enable signals.

A wide variety of clock enable outputs are produced by the module and are shown in [Table 5-8](#). This table lists all of the possible operating modes of the receiver, the frequency of the RX_USRCLK clock in each mode, and the behavior of each clock enable in each mode. In the table, 1080p refers only to 1080p 50 Hz, 59.94 Hz, or 60 Hz. The clock enable columns, such as ce_triple, show the effective clock enable assertion rate relative to USRCLK. For example, 1/11 (27 MHz) means that the clock enable is asserted one out of every 11 USRCLK cycles for an effective rate of 27 MHz.

Table 5-8: Full-Featured Receiver Clock Enables

SDI Mode	Video Format	USRCLK (MHz)	ce_triple (MHz)	ce_sample_rate (MHz)	ce_hd3G (MHz)	ce_3G (MHz)	ce_anc M(Hz)
SD-SDI	All	297	1/11 (27)	1/11 (27)	1/2 (148.5)	Always 1	1/11 (27)
HD-SDI	All	148.5	1/2 (74.25)	1/2 (74.25)	1/2 (74.25)	Always 1	1/2 (74.25)
Dual Link HD-SDI	1080p	148.5	1/2 (74.25)	Always 1 (148.5)	1/2 (74.25)	Always 1	1/2 (74.25)
	> 20b/sample ⁽¹⁾	148.5	1/2 (74.25)	1/2 (74.25)	1/2 (74.25)	Always 1	1/2 (74.25)
3G-SDI Level A	1080p	297	1/2 (148.5)	1/2 (148.5)	1/2 (148.5)	1/2 (148.5)	1/2 (148.5)
	> 20b/sample	297	1/2 (148.5)	1/4 (74.25)	1/2 (148.5)	1/4 (74.25)	1/2 (148.5)
3G-SDI Level B	SMPTE 372M 1080p	297	1/2 (148.5)	1/2 (148.5)	1/2 (148.5)	1/2 (148.5)	1/4 (74.25)
	SMPTE 372M > 20b/sample	297	1/2 (148.5)	1/4 (74.25)	1/2 (148.5)	1/4 (74.25)	1/4 (74.25)
	2X HD-SDI	297	1/2 (148.5)	1/4 (74.25)	1/2 (148.5)	1/4 (74.25)	1/4 (74.25)

Notes:

1. The "> 20b/sample" video format refers to any video format with more than 20 bits per sample.

Video and Timing Output Signals

The triple_sdi_rx module has many video and timing outputs. Some sets of outputs are only valid in certain modes, while others are valid in multiple modes. [Table 5-9](#) shows which outputs are valid in each mode of operation and what type of data is available on them.

Table 5-9: Outputs vs. Operating Mode of triple_sdi_rx Module

SDI Mode	Sub-Mode	HD Outputs	3G Outputs	SD Outputs	ANC Outputs	Notes
HD-SDI	Single Link	Y and C component streams	Not valid	Not valid	Y and C component streams	HD and ANC outputs are identical.
	SMPTE 372M Dual Link	Data streams from the single link received by this receiver	Native video	Not valid	Link A on anc_ds1_a and anc_ds2_a. Link B on anc_ds1_b and anc_ds2_b.	HD-SDI outputs are the single link data from the module's HD-SDI input. The 3G-SDI outputs are the native video unpacked from the dual link streams of both receivers.
3G-SDI	Level A	3G-SDI data streams (video not unpacked)	Native video	Not valid	3G-SDI data streams	HD and ANC outputs are identical.
	Level B with SMPTE 372M	Not valid	Native video	Not valid	Four SMPTE 372M data streams (video is not unpacked)	
	Level B with 2 X HD-SDI streams	Not valid	HD-SDI #1 is output on y_g_3G and cb_b_3G. HD-SDI #2 is output on alpha_3G and cr_r_3G.	Not valid	HD-SDI stream A on anc_ds1_a and anc_ds2_a. HD-SDI stream B on anc_ds1_b and anc_ds2_b.	The two HD-SDI signals are output on the 3G-SDI ports and on the ANC outputs (although they are not necessarily synchronous on these two sets of outputs). The line number from HD-SDI #1 is output on both the ln and anc_ln ports. The line number from HD-SDI #2 is normally the same and is not output from the module but is present in the data streams following the EAV.

Table 5-9: Outputs vs. Operating Mode of `triple_sdi_rx` Module (Cont'd)

SDI Mode	Sub-Mode	HD Outputs	3G Outputs	SD Outputs	ANC Outputs	Notes
SD-SDI	EDH	Multiplexed Y/C data on <code>y_hd</code> (not corrected by flywheel)	Not valid	Multiplexed Y/C data on <code>video_sd</code> with timing errors corrected by flywheel	Multiplexed Y/C data on <code>anc_ds1_a</code>	All SD outputs are valid. The SD output ports are driven by the video flywheel and are slightly delayed from the raw SD data stream present on the HD outputs.
	AUTODETECT	Multiplexed Y/C data on <code>y_hd</code>	Not valid	Multiplexed Y/C data on <code>video_sd</code>	Multiplexed Y/C data on <code>anc_ds1_a</code>	The <code>edh_err</code> , <code>ln_sd</code> , <code>f_sd</code> , <code>v_sd</code> , and <code>h_sd</code> outputs are not valid.
	MINIMAL	Multiplexed Y/C data on <code>y_hd</code>	Not valid	Multiplexed Y/C data on <code>video_sd</code>	Multiplexed Y/C data on <code>anc_ds1_a</code>	The <code>edh_err</code> , <code>ln_sd</code> , <code>f_sd</code> , <code>v_sd</code> , and <code>h_sd</code> outputs are not valid. The <code>t_format</code> output is not valid for SD-SDI.

SD-SDI Output Timing

In SD-SDI mode, the `RXRECCLK` from the GTP RX runs at 297 MHz. The recovered SD-SDI data stream is output on the `video_sd` port with the Y and C components interleaved at a 27 MHz data rate.

In SD-SDI mode, the 297 MHz `RXRECCLK` from the GTP RX is not a recovered clock because the GTP RX is locked to the reference clock and asynchronously samples the input bitstream. `RXRECCLK` is therefore an exact multiple of the reference clock supplied to the GTP transceiver. If the GTP reference clock is generated by a genlock circuit, and therefore frequency locked to the video signal, `RXRECCLK` could then be considered to be frequency locked to the video, as well.

The GTP RX provides the oversampled SD data to the `triple_sdi_rx` module where a DRU recovers the data. The DRU provides a data strobe that is asserted when it has a 10-bit word of data ready. The `triple_sdi_rx` module outputs this data strobe on the `ce_triple` output in SD-SDI mode. On average, this output is asserted once every 11 clock cycles. The output cadence is not, however, regular in nature. Due to a variety of factors, the number of clock cycles between assertions of the data strobe varies considerably and can be much less or much more than 11 clock cycles. Applications must not rely on the data ready signal being regular in nature. For example, it is quite difficult to directly convert this signal into a 27 MHz clock without using a very low bandwidth PLL.

Figure 5-7 shows the timing of the SD video and timing signals produced by the `triple_sdi_rx` module. The outputs only change on the rising edge of the `clk` signal when `ce_triple` (and `ce_sample_rate`) is High. The line number output on the `ln_sd` port changes to the new line number during the first word of the EAV.

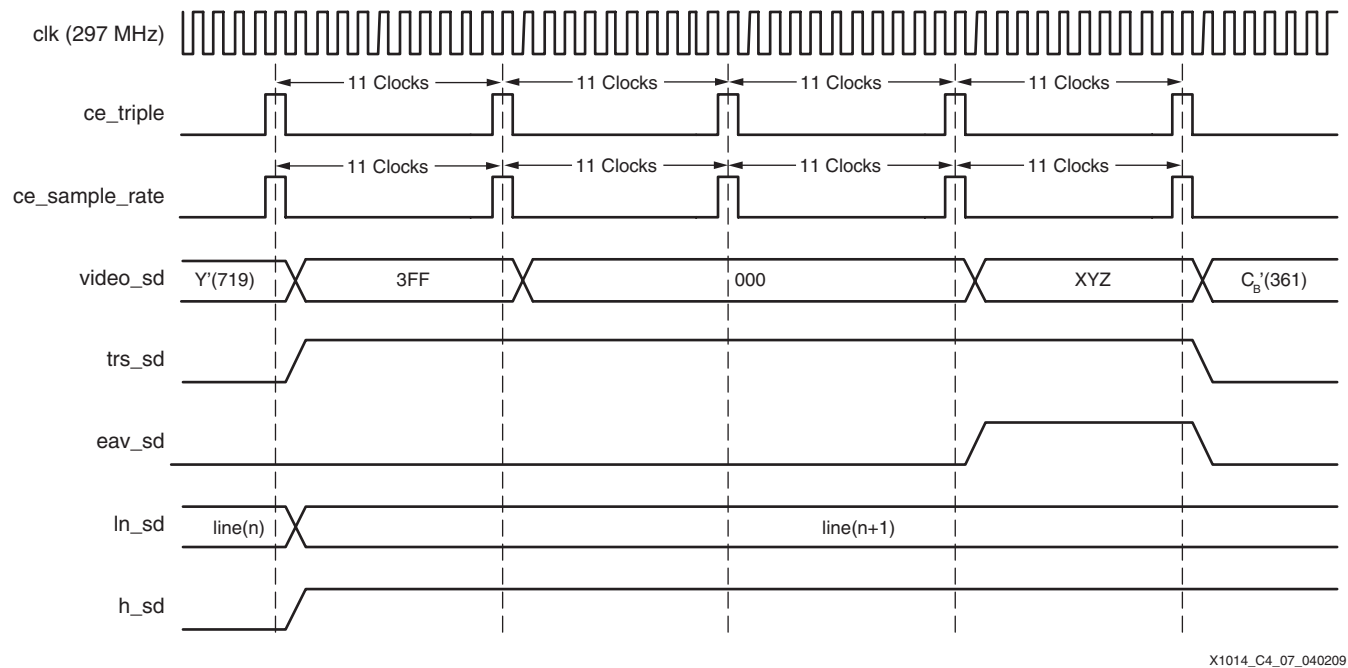


Figure 5-7: SD-SDI Output Timing of `triple_sdi_rx` Module

The `f_sd` (field indicator) and `v_sd` (vertical sync) outputs always change during the first word of the EAV. The `h_sd` (horizontal sync) output rises during the first word of the EAV and falls after the last word of the SAV. Therefore, `h_sd` is high during the entire horizontal blanking interval, including all words of the EAV and SAV.

The `ln_sd`, `f_sd`, `v_sd`, `h_sd`, and `edh_err` outputs are only valid if the `SDI_MODE` parameter/generic is set to "EDH". The `t_format` output is only valid in SD-SDI mode if the `SDI_MODE` parameter/generic is "EDH" or "AUTODETECT".

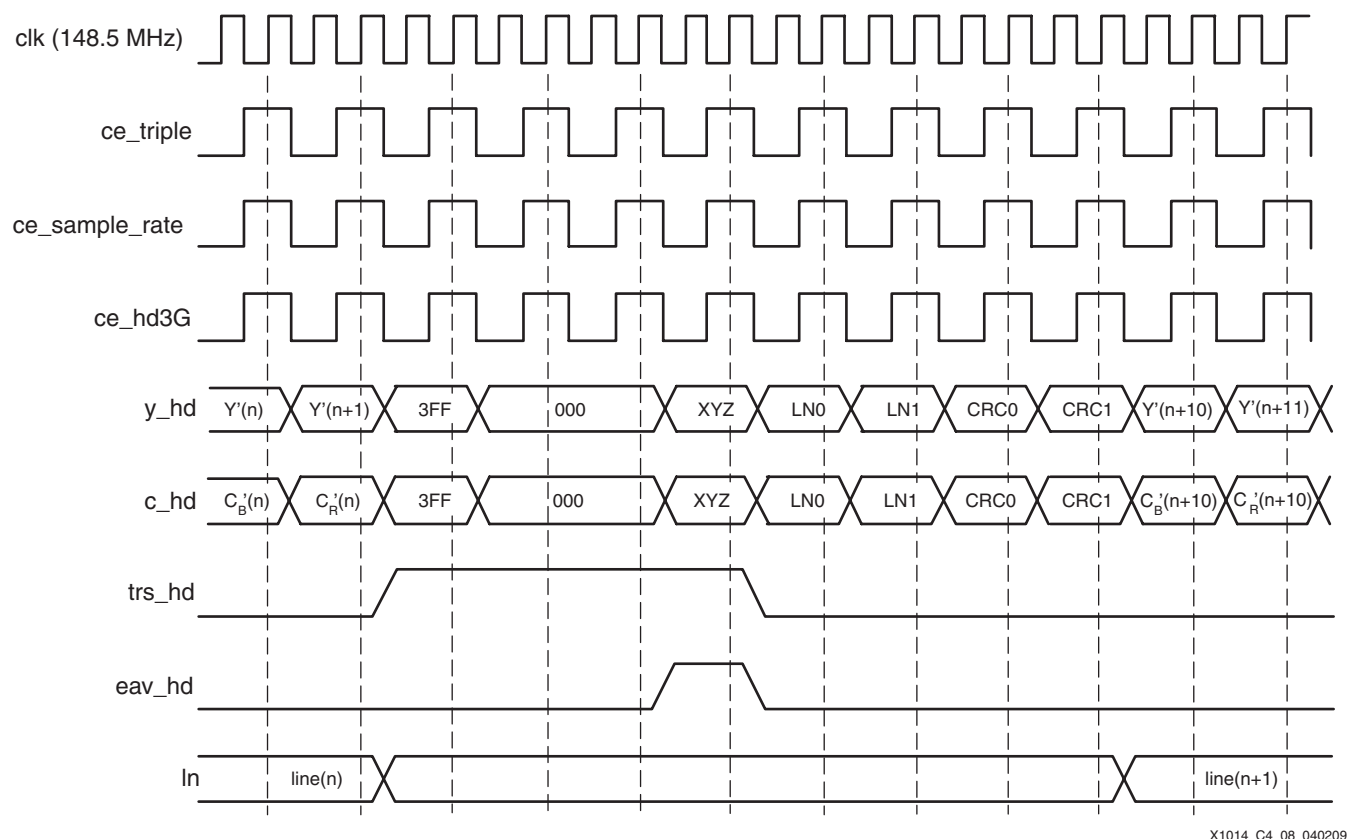
When `SD_MODE` is set to "EDH", the SD video output and timing signals are processed by the video flywheel. This flywheel corrects misplaced EAV and SAV sequences. Because of this, the video does not always match the original video received by the triple-rate receiver when the flywheel detects timing errors in the video. The flywheel adds a small amount of latency. The original, uncorrected video is output on the `y_hd` output along with matching timing on the `eav_hd`, `sav_hd`, and `trs_hd` outputs. The video output on the `y_hd` output does not have the latency introduced by the flywheel, so it precedes the video output on the `video_sd` output by a few clock cycles.

If `SD_MODE` is "AUTODETECT" or "MINIMAL", the video output on the `video_sd` output is identical to and in sync with the video output on the `y_hd` output.

HD-SDI Output Timing

In HD-SDI mode, the `RXRECCLK` recovered clock from the GTP RX is a true recovered clock and runs at 148.5 MHz (or 148.5/1.001 MHz). The Y and C data streams of the HD-SDI signal are output on the `y_hd` and `c_hd` ports along with the timing signals `trs_hd`, `eav_hd`, and `sav_hd`. The line number for HD-SDI is output on the `ln` port.

The `ce_triple`, `ce_sample_rate`, and `ce_hd3G` clock enables are asserted every other clock cycle, producing an output data rate of 74.25 MHz (or 74.1758 MHz). The output ports only change when these clock enables are High. Figure 5-8 shows the timing of the HD-SDI outputs.



X1014_C4_08_040209

Figure 5-8: HD-SDI Output Timing of `triple_sdi_rx` Module

3G-SDI Output Timing

3G-SDI output timing varies depending on the video format and the level (A or B). Also, depending on the video format, the active set of video component output ports changes. Table 5-10 shows, for each video format, what is present on the various 3G-SDI video component outputs.

Table 5-10: `SMPTE425_rx` Output Video Component Ports for each Video Format

format_3G	Video Format	y_g_3G	cb_b_3G	cr_r_3G	alpha_3G	Output Sample Rate (MHz)
000	Not valid					
001	4:4:4 10-bit or 4:4:4:4 10-bit	G' on bits [11:2]	B' on bits [11:2]	R' on bits [11:2]	Alpha if alpha_act_3G = 1	74.25
010	4:4:4 12-bit	G'	B'	R'		74.25
011	Not valid					
100	4:2:2 10-bit	Y' on bits [11:2]	C _{B'} C _{R'} interleaved on bits [11:2]			148.5
101	4:4:4 10-bit or 4:4:4:4 10-bit	Y' on bits [11:2]	C _{B'} on bits [11:2]	C _{R'} on bits [11:2]	Alpha if alpha_act_3G = 1	74.25

Table 5-10: SMPTE425_rx Output Video Component Ports for each Video Format (Cont'd)

format_3G	Video Format	y_g_3G	cb_b_3G	cr_r_3G	alpha_3G	Output Sample Rate (MHz)
110	4:4:4 12-bit	Y'	C _B '	C _R '		74.25
111	4:2:2 12-bit	Y'	C _B 'C _R ' interleaved			74.25

As shown in Table 5-10, 10-bit components output on the y_g_3G, cb_b_3G, and cr_r_3G ports are always output on the most significant 10 bits of the port (bits 11 through 2).

In 3G-SDI mode, the RXRECCLK from the GTP RX runs at 297 MHz (or 297/1.001 MHz). RXRECCLK is a true recovered clock and is locked to the input data rate.

There are two different output sample rates: 74.25 MHz (or 74.25/1.001 MHz) and 148.5 MHz (or 148.5/1.001 MHz). The 1080p 50 Hz, 59.94 Hz, and 60 Hz video formats are the only ones with 148.5 MHz output sample rates. All other video formats have 74.25 MHz output sample rates.

When the output sample rate is 148.5 MHz, the ce_triple, ce_sample_rate, ce_hd3G, and ce_3G signals are all identical and are asserted every other clock cycle. For the 74.25 MHz sample rate, the ce_triple and ce_hd3G signals are still asserted every other clock cycle (148.5 MHz), but ce_sample_rate and ce_3G are asserted at the 74.25 MHz sample rate. Thus, for 3G-SDI, ce_triple and ce_hd3G are always asserted at 148.5 MHz, but ce_sample_rate and ce_3G are either 148.5 MHz or 74.25 MHz, which matches the sample rate of the video format.

The current picture line number is output on the ln port. The line number changes sometime between the start of the EAV and four clock cycles after the XYZ word of the EAV. The line numbers output on this port are always "picture" line numbers and not "interface" line numbers. Interface line numbers differ from picture line numbers only for SMPTE 352M data streams carried by 3G-SDI level B or dual link HD-SDI. In these cases, the interface line number is available on the anc_ln port.

For level B, the triple_sdi_rx module unpacks SMPTE 372M data streams to native video. This does require valid SMPTE 352M VPID packets to identify the video format. Otherwise, the format is specified by the lvlb_default_mapping input port. In some applications, it might not be desirable to unpack level B SMPTE 372M data streams to native video. In those cases, the raw SMPTE 372 data streams are available on the ancillary data output ports, with the link A data streams on anc_ds1_a (Y) and anc_ds2_a (C), and the link B data streams on anc_ds1_b (Y) and anc_ds2_b (C).

When dual HD-SDI streams are carried on a 3G-SDI level B interface (not dual link), those streams are available on the 3G-SDI output ports. HD-SDI stream 1 is available on ports y_g_3G (Y) and cb_b_3G (C). HD-SDI stream 2 is available on ports alpha_3G (Y) and cr_r_3G (C). The two HD-SDI streams are also output on the ancillary data ports. HD-SDI stream 1 is output on ports anc_ds1_a (Y) and anc_ds2_a (C), and HD-SDI stream 2 is output on ports anc_ds1_b (Y) and anc_ds2_b (C). The output streams on the 3G-SDI output ports might not be synchronous with the streams output on the ancillary ports due to differences in the data processing path latencies. The ln and anc_ln ports both output the line number captured from HD-SDI signal 1. The line number of HD-SDI signal 2 is not output from the module but can be captured from the LN words embedded in the data streams immediately after the EAV.

Figure 5-9 shows the 3G-SDI output timing for video with a sample rate of 148.5 MHz. Figure 5-10 shows the 3G-SDI output timing for video with a sample rate of 74.25 MHz.

Figure 5-11 shows the 3G-SDI level B timing when two HD-SDI streams are carried on the interface. The timing of the In output port is only shown in Figure 5-9, but it is the same for Figure 5-10 and Figure 5-11.

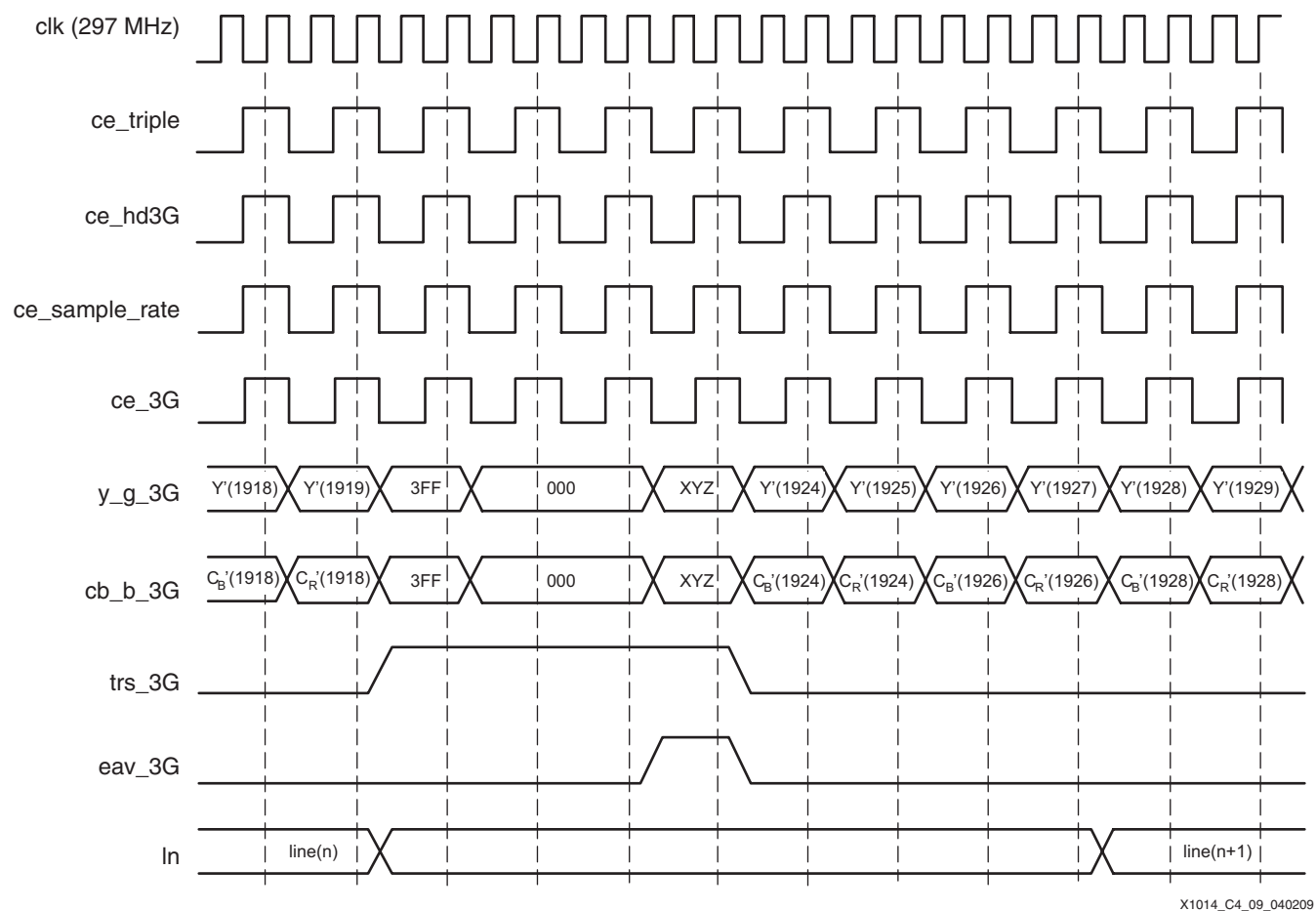
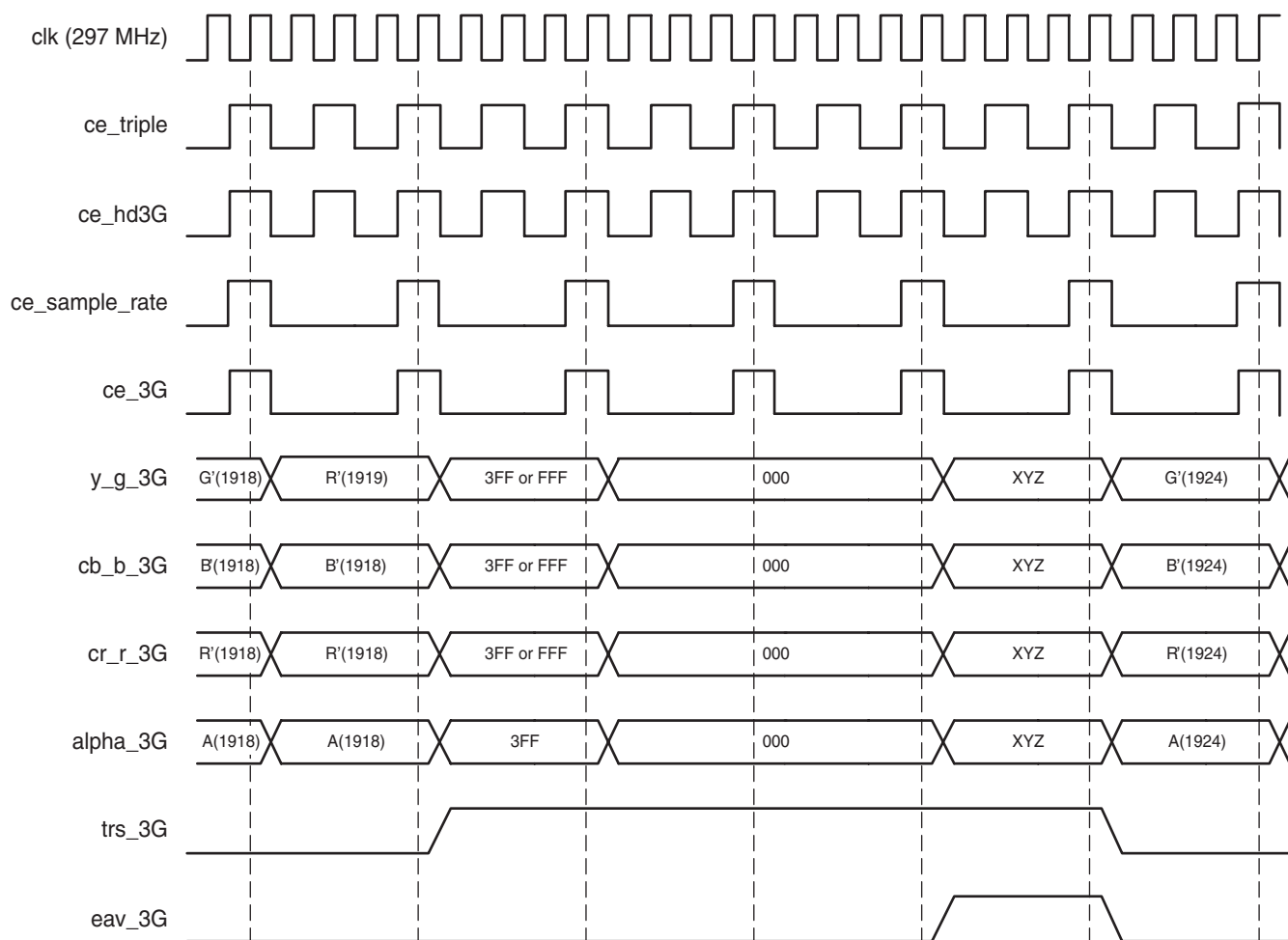


Figure 5-9: 3G-SDI Timing of `triple_sdi_rx` Module for 148.5 MHz Sample Rate (1080p 50 Hz, 59.94 Hz, and 60 Hz)



X1014_C4_10_040209

Figure 5-10: 3G-SDI Timing of `triple_sdi_rx` Module for 74.25 MHz Sample Rate

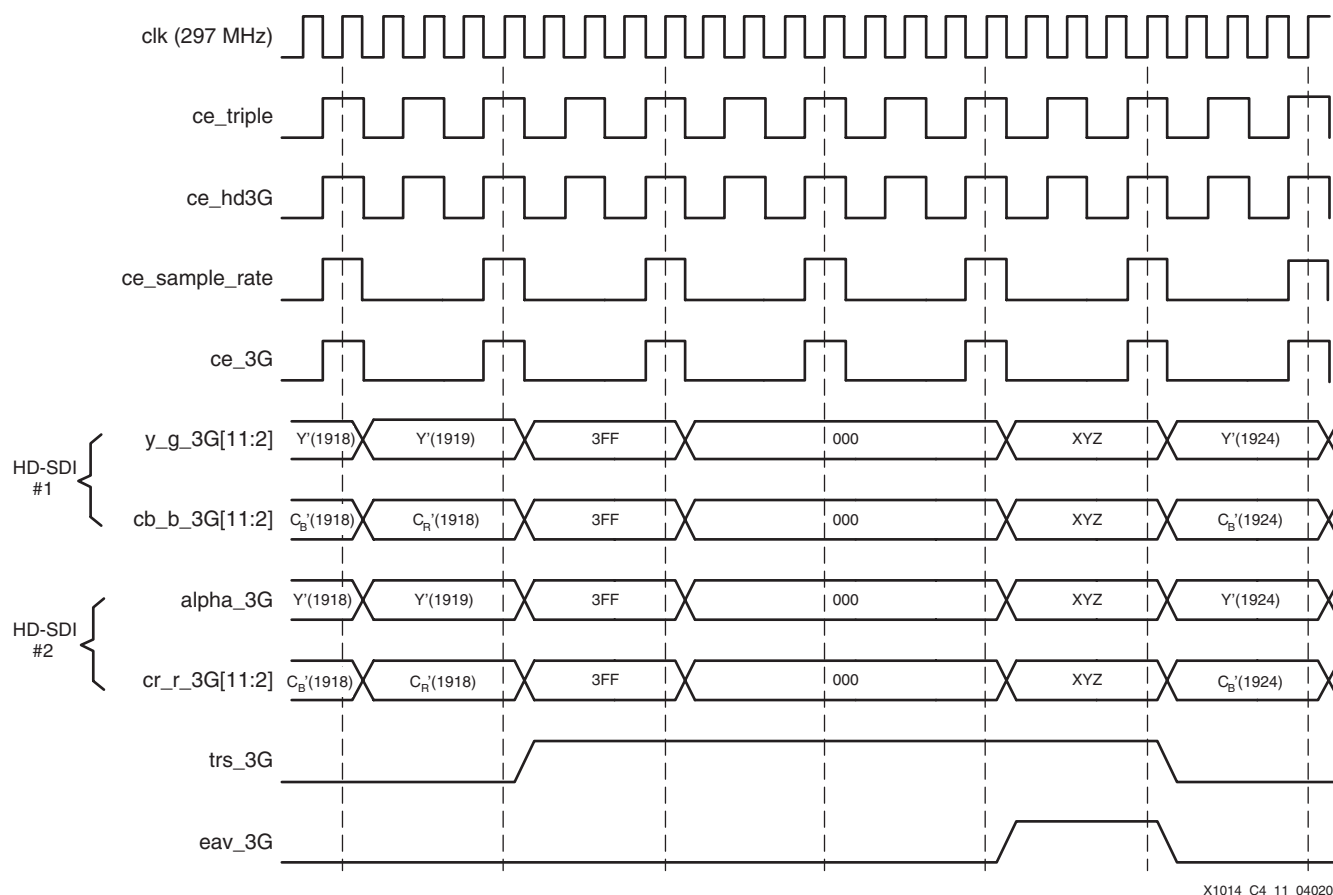


Figure 5-11: 3G-SDI Level B Dual Stream Timing of `triple_sdi_rx` Module

Ancillary Data Outputs

The `triple_sdi_rx` module provides a set of data stream and timing outputs intended to be used for processing ancillary data embedded in SDI data streams. For SD-SDI and HD-SDI, embedded ancillary data can be processed from the regular SD or HD output streams. However, this is not always true for 3G-SDI and dual link HD-SDI. The unpacking process used to reformat most 3G-SDI and dual link HD-SDI data formats into native video scrambles and destroys the embedded ancillary data packets. Therefore, the raw data streams that include the intact ancillary data packets are provided as outputs from the module. The ancillary data output ports provide the correct data streams and timing for processing ancillary data for all SDI modes. The `ce_anc` data ready signals produce the correct timing for all cases of SD, HD, 3G, and dual link HD-SDI.

In 3G-SDI level B mode with SMPTE 372M data streams, or in dual link HD-SDI mode, native video is output on the regular 3G-SDI video output ports. In some cases, applications might prefer to have the four raw SMPTE 372M data streams. These are available on the ancillary output ports. When running in 3G-SDI level B mode with dual HD-SDI streams (not dual link), the two HD-SDI streams are output on both the 3G-SDI output ports and on the ancillary data output ports.

Four 10-bit ancillary data output ports plus timing signals are provided by the module. For HD-SDI and 3G-SDI level A, only two data ports are used: `anc_ds1_a` and `anc_ds2_a`. The data rate for HD-SDI is 74.25 MHz. For 3G-SDI, it is 148.5 MHz. For 3G-SDI level B and

dual link HD-SDI, all four data ports are active and the data rate is 74.25 MHz. For SD-SDI, only one 10-bit data port, anc_ds1_a, is used.

Figure 5-12 shows the timing of the ancillary data ports in 3G-SDI level A mode. The ancillary data timing is the same regardless of whether the video sample rate is 148.5 MHz or 74.25 MHz.

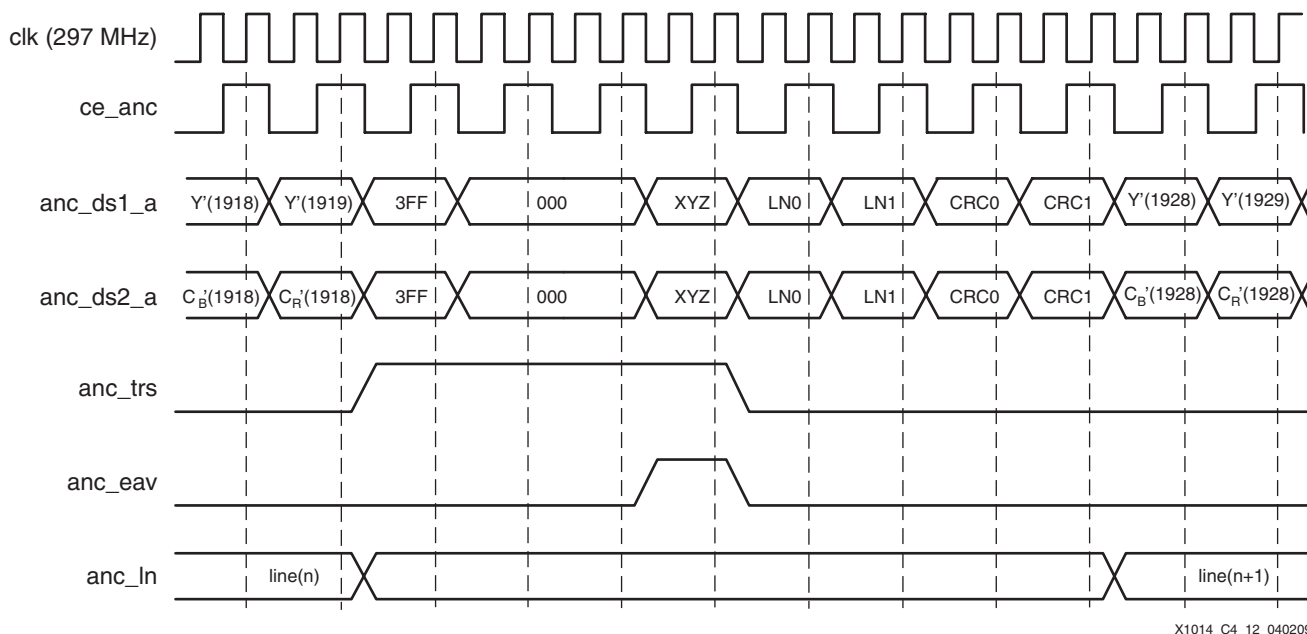


Figure 5-12: 3G-SDI Level A ANC Data Port Timing of triple_sdi_rx Module

Figure 5-13 shows the timing of the ancillary data ports for level B, regardless of whether the data streams are SMPTE 372M dual link data streams or dual HD-SDI data streams. The exact timing of the line number output on anc_ln varies slightly depending on the mode and video format, but this output is always valid by the start of the HANC space, after the last CRC word following the EAV. The line number output by the anc_ln port is an interface line number and not a picture line number. Picture line numbers are output on the ln port. In most cases, picture and interface line numbers are the same. However, for SMPTE 372M data streams, interface line numbers are different from picture line numbers. The interface line number is required when processing ancillary data, not the picture line number.

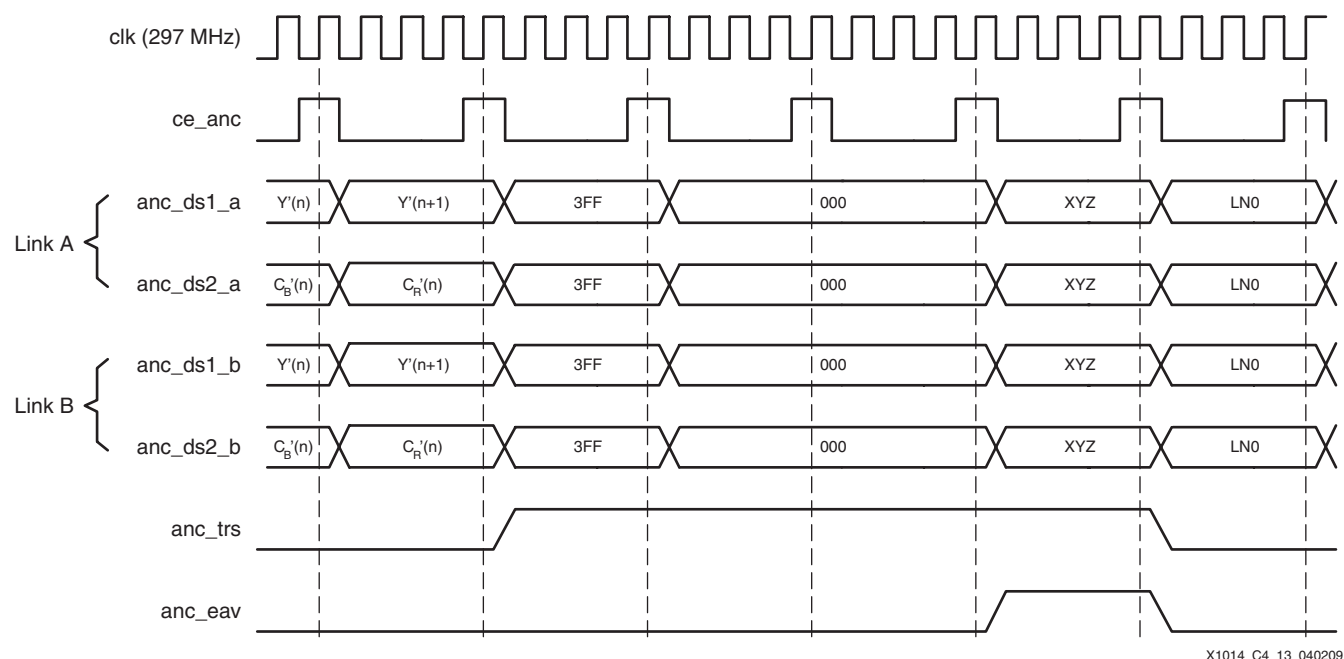


Figure 5-13: 3G-SDI Level B ANC Data Port Timing of `triple_sdi_rx` Module

Figure 5-14 shows the timing of the data on the ancillary data ports for HD-SDI. The second set of ANC output data streams are only active in SMPTE 327M dual link HD-SDI mode.

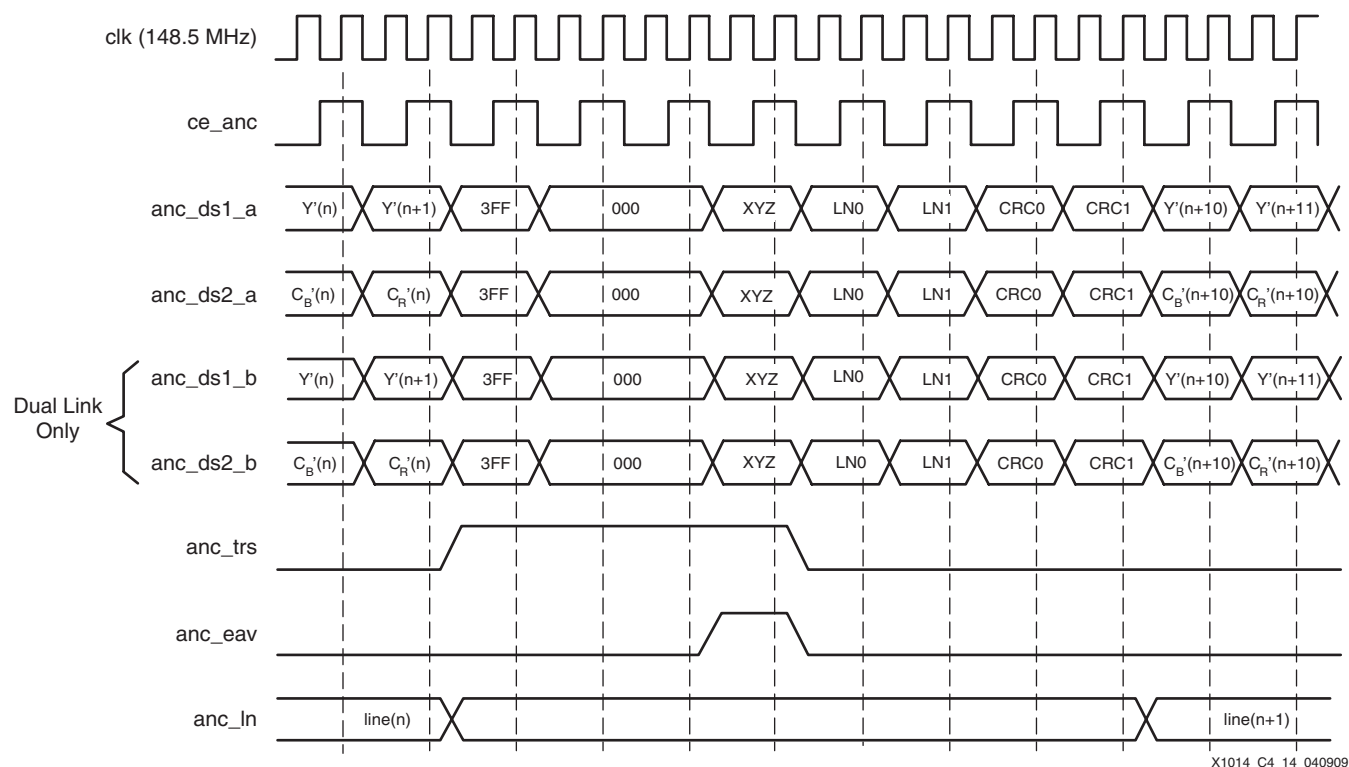


Figure 5-14: HD-SDI ANC Data Port Timing of `triple_sdi_rx` Module

Figure 5-15 shows the timing of the ancillary data ports for SD-SDI. The anc_In output is only valid in SD-SDI mode if the SD_MODE parameter is set to EDH. The ce_anc clock enable output is typically asserted once every 11 clock cycles, but can change to a 10 or 12 clock cycle period when the DRU needs to make up the difference between the local reference clock frequency and the input SDI data rate.

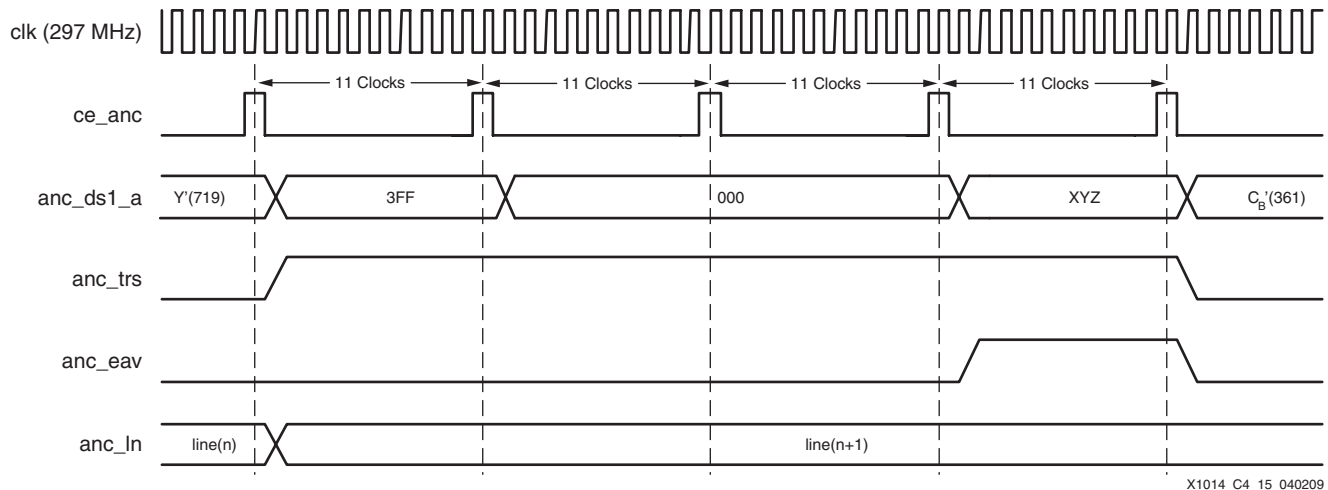


Figure 5-15: SD-SDI ANC Data Port Timing of `triple_sdi_rx` Module

Dual Link HD-SDI Mode

The full-featured triple-rate SDI receiver fully supports SMPTE 372M dual link HD-SDI. Two `triple_sdi_rx` modules are used, one for each link of the dual link HD-SDI pair, with each module connected to its own GTP RX. One `triple_sdi_rx` module runs in normal HD-SDI mode and the other (the master) runs in dual link mode. The data streams from the module running in normal HD-SDI mode are connected to the special dual link data stream input ports of the master module. The master module unpacks the dual link streams into native video and outputs them on its 3G-SDI video output ports—just as it does with SMPTE 372M data streams carried on a 3G-SDI level B interface. Either `triple_sdi_rx` module can be the master module. The master is determined solely by the interconnections between the two `triple_sdi_rx` modules.

Figure 5-16 shows an example dual link HD-SDI-capable receiver. Two GTP RX units (either in the same GTP_DUAL tile as shown in Figure 5-16 or in different GTP_DUAL tiles) feed two `triple_sdi_rx` modules. The HD-SDI Y and C data streams, along with the TRS signal and VPID data, are connected from one `triple_sdi_rx` module to the special dual link input ports of the other `triple_sdi_rx` module (the master module). The master `triple_sdi_rx` module then looks at the SMPTE 352M VPID data from both receivers and determines if it should run in dual link HD-SDI mode. The master `triple_sdi_rx` module can also be forced to run in dual link HD-SDI mode by driving its `force_dl_mode` input port High and specifying the unpacking algorithm to use on the `dl_default_mapping` port. These two ports are not shown in the simplified block diagram of Figure 5-16.

When running in dual link HD-SDI mode, the master `triple_sdi_rx` module outputs native video and timing on its 3G output ports: `y_g_3G`, `cb_b_3G`, `cr_r_3G`, `alpha_3G`, `trs_3G`, `eav_3G`, and `sav_3G`. The value on the `ln` output port is the current picture line number, not the interface line number.

The `format_3G` output port indicates the dual link HD-SDI mapping used and is useful in determining how to interpret the data streams on the component output ports. The output sample rate is either 148.5 MHz (for 1080p 50 Hz, 59.94 Hz, and 60 Hz video) or 74.25 MHz for video formats with more than 20 bits per sample. An appropriate clock enable, such as `ce_sample_rate`, should be used with the video outputs. [Figure 5-17, page 176](#) and [Figure 5-18, page 177](#) show the timing diagrams for the two sample rates in dual link HD-SDI mode.

The four raw data streams from the two receivers are output on the ancillary data output ports of the master `triple_sdi_rx` module in dual link HD-SDI mode. Any skew present on the data streams from the two links has been removed by the time the data is output on the ancillary data ports. Ancillary data can also be extracted from the regular HD-SDI streams from both receiver modules if desired. The advantage of using the ANC output ports of the master receiver in dual link HD-SDI mode is that the data streams from the two receivers have been de-skewed relative to each other.

It is essential that the HD-SDI data streams from the second `triple_sdi_rx` module be synchronous with the clock used by the master `triple_sdi_rx` module. There are several ways to do this. [Figure 5-16](#) shows one example of how clocking can be implemented in a dual link HD-SDI system. In this example, the recovered clock (RXRECLK1) from the bottom RX is used to clock the RX when the system is not running in dual link HD-SDI mode. When running in dual link HD-SDI mode, the BUFGMUX switches the clock for the bottom RX unit over to RXRECCLK0 from the top GTP RX. In dual link HD-SDI mode, the bit rates of both links are, by definition, identical. The RX buffer in the bottom GTP RX compensates for any short-term fluctuations between the received data rate and the rate at which the data is being read using the recovered clock from the top GTP RX. By clocking the data out of the bottom GTP RX with the recovered clock from the other GTP RX, the data streams from the bottom RX are synchronous with the clock used to process the dual link data in the master `triple_sdi_rx` module. The master `triple_sdi_rx` module can handle the phase of the data streams from the other receiver being out of phase with its clock enables as long as these data streams are synchronous to its clock.

The BUFGMUX can be controlled with the `dl_active` output of the master `triple_sdi_rx` module ANDed with the `mode_HD` outputs of both `triple_sdi_rx` modules. The `dl_active` output is High both in dual link HD-SDI mode and in 3G-SDI level B mode when SMPTE 372M data streams are present. The BUGMUX should only be switched to drive the RXUSRCLK of the second RX in dual link HD-SDI mode, not in 3G-SDI level B mode. Therefore, it is necessary to qualify `dl_active` with the `mode_HD` signals from both receivers.

The clocking scheme shown in [Figure 5-16](#) does lack one feature, which is the ability to use a local reference clock to keep the units running if the recovered clocks stop due to a stoppage of the input bitstream. If this feature is required, a different clocking structure is required. This structure would require a 3:1 clock multiplexer for the second receiver unit. Virtex-5 devices only have 2:1 clock multiplexers. Cascading 2:1 clock multiplexers would not be a good implementation because it would introduce significant clock skew on a very fast clock.

The preferred implementation would be to clock each RX with its own recovered clock. Then, using a very shallow asynchronous FIFO, the data streams and the TRS signal would be synchronized from the second RX to the master RX unit clock. The VPID data can be synchronized through a dual-rank synchronizer.

The FIFO would only need to be a few locations deep because the two receivers would be running at the same average frequency in dual link mode. A deep FIFO would increase the

skew between the two links to the extent that the dual link HD-SDI logic in the master RX could not deskew the two links.

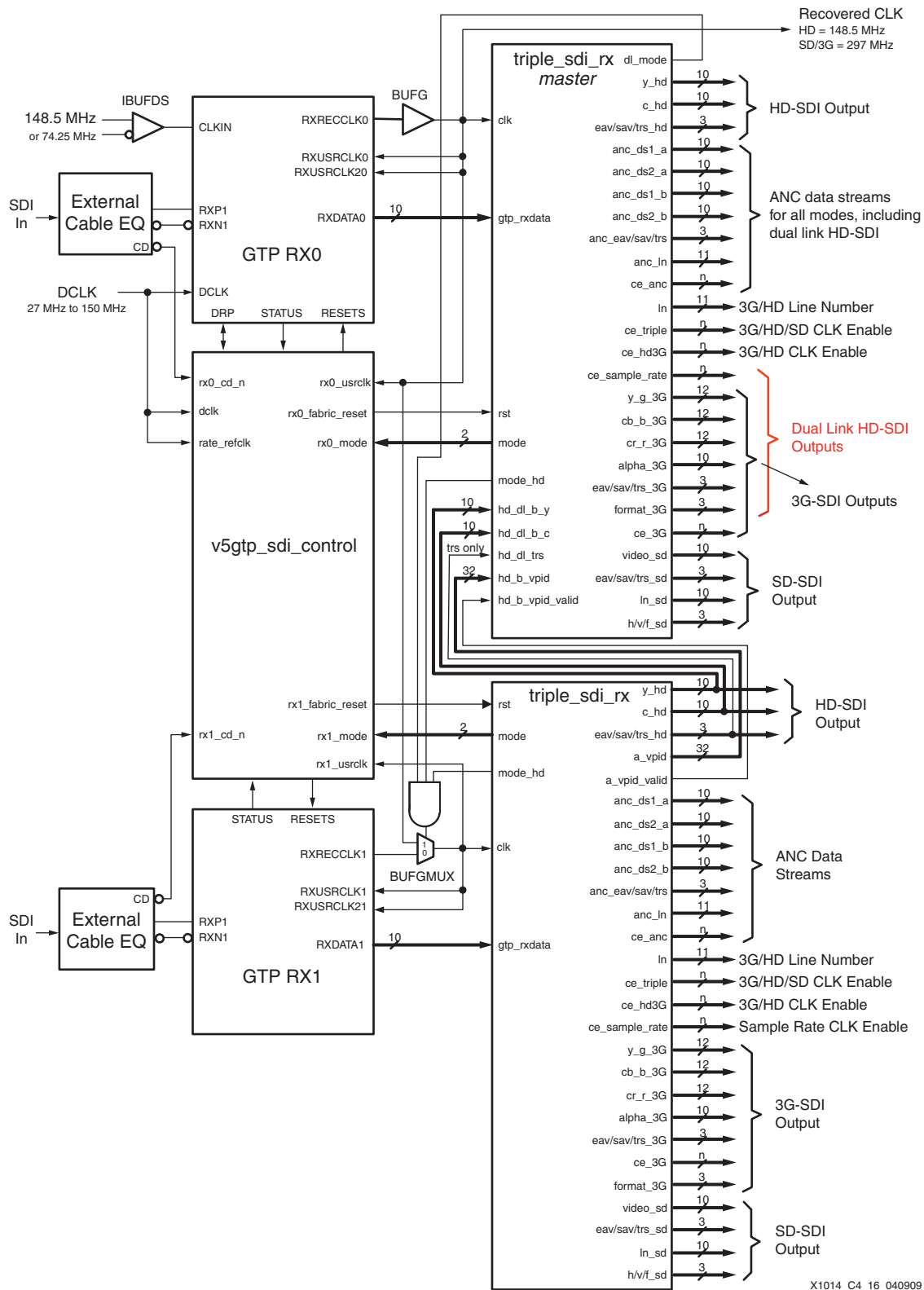


Figure 5-16: Dual Link HD-SDI Mode Block Diagram of triple_sdi_rx Module

Figure 5-17 shows the timing of the output native video in dual link HD-SDI mode when the video is 1080p 50 Hz, 59.94 Hz, or 60 Hz. The clock frequency is 148.5 MHz. The `ce_triple` clock enable is asserted every other clock cycle. This is because the data rate of the single-link HD-SDI data streams output on the HD-SDI ports is 74.25 MHz. The `ce_sample_rate` clock enable, however, is asserted High all the time, indicating that the data rate of the unpacked video from the dual link HD-SDI streams is 148.5 MHz. The native video is output on the 3G-SDI video ports of the master `triple_sdi_rx` module in dual link HD-SDI mode.

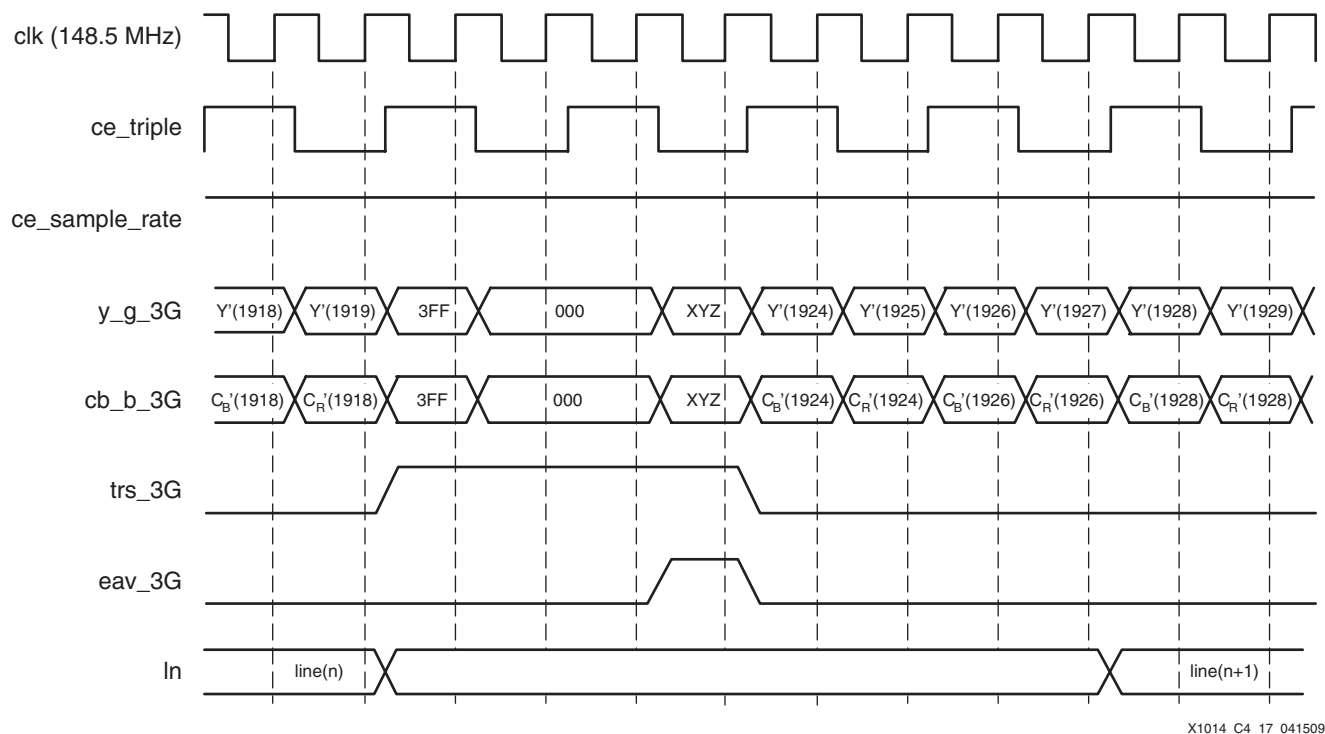
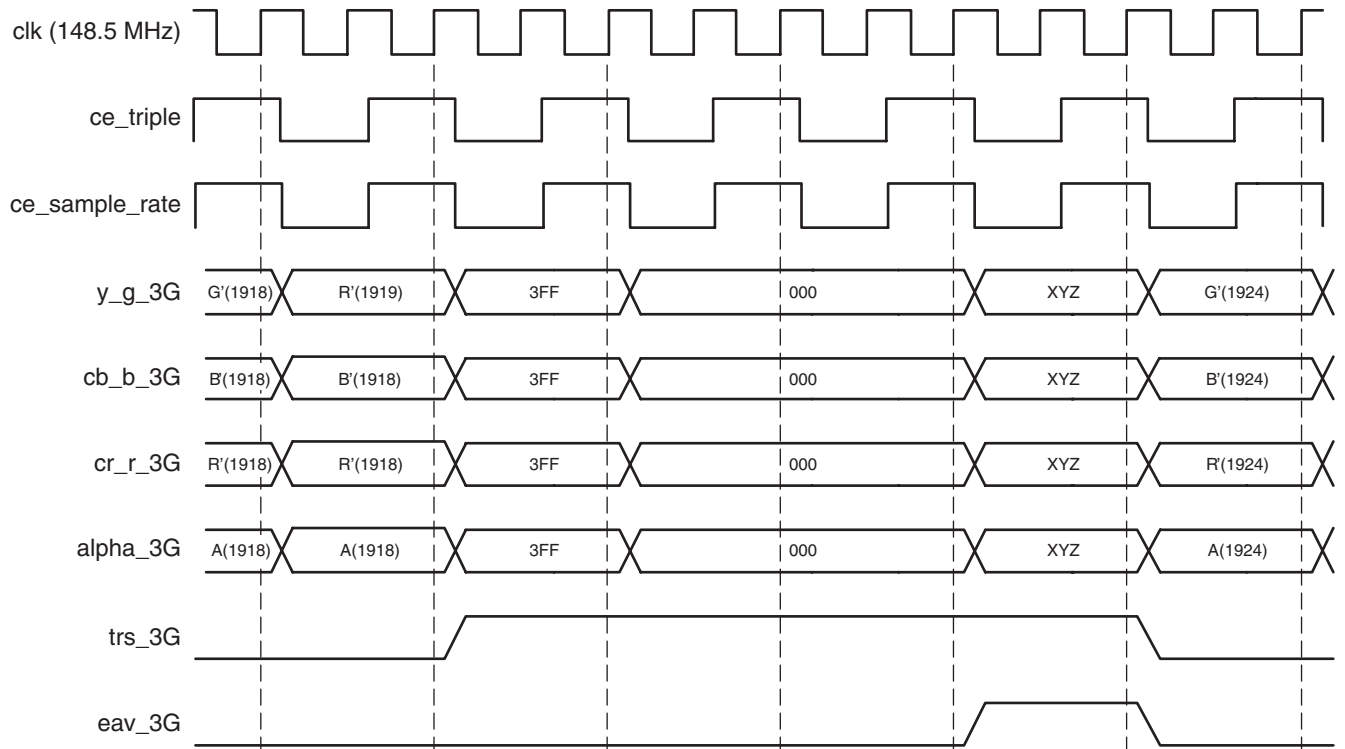


Figure 5-17: Dual Link HD-SDI Timing for 148.5 MHz Sample Rate of `triple_sdi_rx` Module

Figure 5-18 shows the timing of the native video in dual link HD-SDI mode when the video has more than 20 bits per video sample (74.25 MHz sample rate). The clock frequency is 148.5 MHz. The `ce_triple` signal is asserted every other clock cycle, which is a 74.25 MHz rate. The `ce_sample_rate` signal is also asserted every other clock cycle because the output data rate is 74.25 MHz. The native video is output on the 3G-SDI video ports of the master `triple_sdi_rx` module in dual link HD-SDI mode. The `In` port timing is not shown in Figure 5-18, but this timing is the same as shown in Figure 5-17, i.e., it changes between the first word of the EAV and four clock cycles after the XYZ word of the EAV.

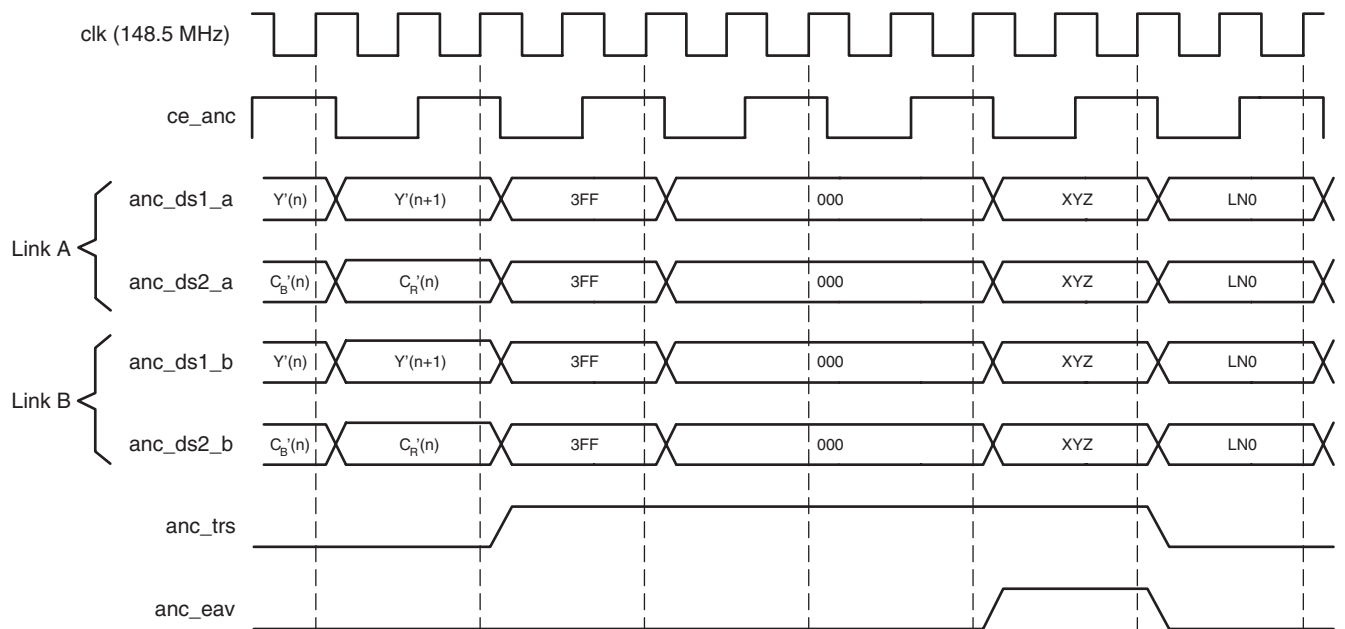
Figure 5-19, page 177 shows the timing of the ANC output ports when the receiver is running in dual link HD-SDI mode. The clock frequency is 148.5 MHz. The `ce_anc` signal runs at the basic HD-SDI data rate of 74.25 MHz (asserted every other clock cycle). The four data streams are output on the four ANC data stream output ports. The data streams from link A and link B are deskewed before being output on the ANC ports. The data streams might also have been swapped. This happens if SMPTE 372M packets are present in the input data streams and indicate that the links need to be swapped. The dual link HD-SDI ANC data streams are only output on the ANC ports of the master `triple_sdi_rx` module. The timing of the `anc_in` port is not shown, but as with the `In`

port timing, it changes between the first word of the EAV and three clock cycles after the XYZ word of the EAV.



X1014_C4_18_041509

Figure 5-18: Dual Link HD-SDI Timing for 74.25 MHz Sample Rate of **triple_sdi_rx** Module



X1014_C4_19_040209

Figure 5-19: Dual Link HD-SDI ANC Data Port Timing of **triple_sdi_rx** Module

FPGA Resource Usage

Table 5-11 shows the FPGA resources required for each of the triple-rate SDI interface reference designs described in this chapter. The resource usage includes all the modules required to implement the interface, including the `v5gtp_sdi_control` module. The resources reported are based on using one RX unit in a GTP_DUAL tile. Some savings over the listed resource usage can be achieved when multiple units in a GTP_DUAL tile are used. This is because logic in the `v5gtp_sdi_control` module can be shared. The results shown were achieved using the ISE® Design Suite 10.1 SP2, and Xilinx Synthesis Technology (XST) and MAP set to optimize for speed. The XST SafeImplementation property was set to **Yes**.

Table 5-11: Triple-Rate SDI Interface FPGA Resource Usage

Reference Design	Flip-Flops	LUTs	Block RAMs
Light RX	1100	1248	None
Full-featured RX (SDI_MODE = "MINIMAL")	1855	2127	3 x RAMB36 3 x RAMB18
Full-featured RX (SDI_MODE = "AUTODETECT")	1898	2219	3 x RAMB36 3 x RAMB18
Full-featured RX (SDI_MODE = "EDH")	2277	2792	3 x RAMB36 3 x RAMB18

As shown in **Table 5-11**, the full-featured RX uses three 36K block RAMs and three 18K block RAMs. The block RAMs are all used by the line buffers required for 1080p 50 Hz and 60 Hz line sequencing when receiving this video format with 3G-SDI level B and dual link HD-SDI.

Timing

Because the basic clock frequency used for 3G-SDI and for SD-SDI is 297 MHz, some paths of the design need to run at this frequency. However, the modules of the triple-rate SDI reference design have been designed so that only a few paths actually run at this frequency. The bulk of the paths run at one half this frequency or less, making it easier to achieve timing. This is done by using clock enables. Those paths that do not run at the full data rate are given a multi-cycle timing constraint so that the timing analyzer knows they are not required to run at 297 MHz. However, the clock enables themselves must meet setup and hold time to all flip-flops that they enable at the full 297 MHz clock frequency. The multi-cycle constraints are shown in the UCF files of the demonstration projects included with the reference design.

It is possible to meet timing with the slowest speed grade Virtex-5 LXT and SXT devices. However, this typically requires optimizing the design for speed in both XST and MAP. Floor planning might be required to meet timing in larger designs or designs that consume a significant portion of the FPGA resources.

To make it possible to place timing constraints on certain signals internal to the reference design, these signals have XST KEEP constraints placed on them in the Verilog and VHDL code. This is an XST-specific constraint that has to be manually converted to the equivalent constraint if Synplify is used. Similarly, some registers in the design have the XST-specific "equivalent register removal" constraint applied, and these constraints also have to be converted to the Synplify equivalent.

All of the modules in the triple-rate SDI have been tested with the slowest speed grade Virtex-5 devices, and it is possible to meet timing in these slowest speed grade parts.

It would be possible to use a basic frequency of 148.5 MHz if a 20-bit RXDATA port was used on the GTP transceiver. However, the triple-rate SDI RX reference design for the Virtex-5 FPGA GTP transceiver was specifically designed to work with 10-bit GTP RXDATA ports. This was done to reduce global clocking resource requirements. Using 20-bit RXDATA paths requires the use of a DCM or PLL to produce two phase-aligned clock frequencies, and requires two global clock trees. In general, these additional clock resources are required per triple-rate SDI receiver and could quickly exhaust the available clocking resources if many SDI interfaces are implemented in a single FPGA. By using 10-bit RXDATA ports and dealing with the 297 MHz clock, the global clocking resources per triple-rate SDI RX have been reduced to one global clock tree with no DCMs or PLLs required.

The RXRECCLK generated by the GTP transceiver can, under some conditions, have erratic timing when the input SDI bitstream stops and restarts, or when the GTP transceiver is switched between SDI modes. These periods of erratic timing can cause problems for sequential control logic such as finite state machines (FSM). The erratic timing can cause FSMs to go into illegal states. Thus, it is highly recommended that the synthesis tool be forced to generate “safe” implementations of FSMs that include illegal state recovery. Most synthesis tools, by default, optimize away illegal state recovery for FSMs.

The clock enable signals are particularly critical timing paths because they must meet setup and hold times of the various synchronous elements at the full 297 MHz clock speed. The reference design uses multiple identical copies of clock enables to reduce loading on the clock enables. This is better than attempting to use a global clock tree to distribute the clock enables because the global clock tree has a large delay almost equal to or exceeding the clock period of a 297 MHz clock (depending on device size). Care must be taken to properly constrain the clock enables as shown in the example UCF files in the reference design.

The clock enables and write enables of the block RAMs used to implement the line buffers in the SMPTE372_rx_1080p_mem_RAMB36 modules (used only in the full-featured RX) are particularly critical and require special timing constraints. These signals are the AND of address bits with the main clock enable.

The address bits are generated by counters in the multi-cycle datapath. The address bits used for the enable and write enables must be fast enough to meet full 297 MHz timing when combined with the clock enable signal to drive the block RAMs. FROM-THRU-TO constraints are used to properly constrain these particular signals as shown in the example UCF files in the reference design.

XST sometimes implements a clock enable using a look-up table (LUT) driving the D input of a flip-flop, rather than using the flip-flop clock enable input. This makes it more difficult to meet timing because it adds the delay of the LUT into the critical clock enable timing path. XST has a constraint called Use Clock Enable. Setting this constraint to **Yes** forces XST to use clock enables rather than implementing the clock enable function through logic driving the D input of the flip-flop. It is recommended that for designs that heavily use clock enables with a high-speed clock, this constraint should be set to **Yes** globally. This constraint is set using the XST properties in Project Navigator, or by a command line flag. It is possible to set this constraint locally on certain modules.

Reference Design

The reference design for the triple-rate SDI receiver is available at <https://secure.xilinx.com/webreg/clickthrough.do?cid=119758>. Open the ZIP archive and extract the file `xapp1014_c5_GTP_SDI_RX.zip`.

Conclusion

The reference designs presented in this chapter provide two distinct methods of implementing triple-rate SDI receivers in Virtex-5 LXT and SXT devices. The full-featured receiver provides a complete triple-rate SDI receiver solution, including the logic to unpack all supported video formats into native video. The simpler light version of the triple-rate SDI receiver reference design is better suited for applications that do not need the unpacking functions.

In either case, the triple-rate SDI receiver reference designs fully support SD-SDI, HD-SDI, dual link HD-SDI, and 3G-SDI level A and level B, with dynamic switching between any of these SDI standards. They do so requiring a minimum number of reference clocks and global clocking resources.

Triple-Rate SDI Transmitter for Virtex-5 LXT and SXT Devices

Summary

The RocketIO™ GTP transceivers available in Virtex®-5 LXT and SXT devices can be used to implement SDI transmitters that support all three SMPTE serial digital video interface standards (SD-SDI, HD-SDI, and 3G-SDI) and can dynamically switch between these standards. This chapter describes such a triple-rate SDI transmitter reference design. The reference design also supports dual link HD-SDI using two triple-rate SDI transmitters paired together.

The triple-rate SDI TX reference design has been designed specifically for the GTP transceiver found in Virtex-5 LXT and SXT devices. It cannot be used directly with other Xilinx® FPGAs.

Introduction

There are many options when implementing a triple-rate SDI interface. Some applications require full mapping of video formats that have more than 20 bits per sample into 3G-SDI data streams. Other applications might not need the mapping function either because the application does not need to support those particular video formats or because the mapping function is implemented elsewhere in the application. Some applications require EDH packet generation for SD-SDI, while others might omit this function to reduce space.

The triple-rate SDI TX reference design is, therefore, designed in a modular manner to allow it to be flexible enough to suit a wide variety of application requirements from the simple to the most complex. Two versions of the triple-rate SDI TX reference designs are provided in this chapter: a light version and a full-featured version. The light version is designed for less demanding applications and provides a simpler, less complicated interface. The full-featured version is a no-compromise solution. It supports every video format compatible with 3G-SDI level A and level B, as well as all video formats supported by dual link HD-SDI. As a result, it is a larger and more complex design.

[Chapter 3, Implementing SMPTE Serial Digital Interfaces with RocketIO GTP Transceivers](#) contains information vital to implementing triple-rate SDI receivers using the Virtex-5 FPGA GTP transceiver, and is prerequisite to reading this chapter.

Supported Video Formats

The triple-rate SDI TX reference design supports the video formats shown in [Table 6-1](#).

Table 6-1: Video Formats Supported by Triple-Rate SDI Transmitter

Interface	Video Standard	Sampling Structure/Bit Depth	Frame/Field Rate (Hz)	Native Video Packing
SD-SDI SMPTE 259M-C	PAL	4:2:2 Y'C _B 'C _R ' 10-bit	50	Not required
	NTSC	4:2:2 Y'C _B 'C _R ' 10-bit	59.94	Not required
HD-SDI SMPTE 292M	SMPTE 274M	4:2:2 Y'C _B 'C _R ' 10-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	Not required
	SMPTE 296M	4:2:2 Y'C _B 'C _R ' 10-bit	720p: 23.98, 24, 25, 29.97, 30, 50, 59.94, 60	Not required
	SMPTE 260M	4:2:2 Y'C _B 'C _R ' 10-bit	1035i: 59.94, 60	Not required
	SMPTE 295M	4:2:2 Y'C _B 'C _R ' 10-bit	1080i: 50	Not required
3G-SDI Level A SMPTE 425M-A	SMPTE 274M	4:2:2 Y'C _B 'C _R ' 10-bit	1080p: 50, 59.94, 60	Light: Yes Full: Yes
		4:4:4 or 4:4:4:4 Y'C _B 'C _R ' or RGB 10-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	Light: No Full: Yes
		4:4:4 Y'C _B 'C _R ' or RGB 12-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	Light: No Full: Yes
		4:2:2 Y'C _B 'C _R ' 12-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	Light: No Full: Yes
	SMPTE 296M	4:4:4 or 4:4:4:4 Y'C _B 'C _R ' or RGB 10-bit	720p: 23.98, 24, 25, 29.97, 30, 50, 59.94, 60	Light: No Full: Yes
	SMPTE 428-9	4:4:4 X'Y'Z' 12-bit	2048 X 1080p: 24	Light: No ⁽¹⁾ Full: No ⁽¹⁾
3G-SDI Level B SMPTE 425M-B	SMPTE 372M	See Dual Link HD-SDI SMPTE 372M in this table.		See Dual Link HD-SDI SMPTE 372M.
	2 X HD-SDI streams	See HD-SDI SMPTE 292M in this table.		Light: Yes Full: Yes

Table 6-1: Video Formats Supported by Triple-Rate SDI Transmitter (Cont'd)

Interface	Video Standard	Sampling Structure/Bit Depth	Frame/Field Rate (Hz)	Native Video Packing
Dual Link HD-SDI SMPTE 372M	SMPTE 274M	4:2:2 Y'C _B 'C _R ' 10-bit	1080p: 50, 59.94, 60	Light: No Full: Yes
		4:4:4 or 4:4:4:4 Y'C _B 'C _R ' or RGB 10-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	Light: No Full: Yes
		4:4:4 Y'C _B 'C _R ' or RGB 12-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	Light: No Full: Yes
		4:2:2 Y'C _B 'C _R ' 12-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	Light: No Full: Yes
	SMPTE 428-9	4:4:4 X'Y'Z' 12-bit	2048 X 1080p: 24	Light: No ⁽¹⁾ Full: No ⁽¹⁾

Notes:

- For SMPTE 428-9 video, the light and full-featured triple-rate SDI receivers can receive the SMPTE 428-9 container but cannot unpack it into native SMPTE 428-1 video.

The triple-rate SDI TX reference design is primarily designed to support only the 270 Mb/s SD-SDI bit rate. It is possible to transmit the other SD-SDI bit rates with this reference design. However, bit rates other than 270 Mb/s require different reference clock frequencies for the GTP transmitter. This requires a reference clock switching solution external to the FPGA and is beyond the scope of this chapter.

Triple-Rate SDI Transmitter Features

The triple-rate SDI transmitter can dynamically switch between SD-SDI, HD-SDI, and 3G-SDI under control of the application. Both HD-SDI bit rates, both 3G-SDI bit rates, and 270 Mb/s SD-SDI are all supported by only two reference clock frequencies.

The two transmitters in a GTP_DUAL tile share a common reference clock. Therefore, they are not totally independent of each other. It is not possible, for example, to transmit SD-SDI with one TX while the other transmits HD-SDI at 1.485/1.001 Gb/s or 3G-SDI at 2.97/1.001 Gb/s because this would require the two transmitters in the same tile to have different reference frequencies. It is possible, however, to transmit SD-SDI and HD-SDI at 1.485 Gb/s or 3G-SDI at 2.97 Gb/s simultaneously using the two transmitters in one GTP_DUAL tile because these bit rates use the same reference clock frequency.

The triple-rate SDI transmitter can generate and insert EDH packets for SD-SDI and CRC values for HD-SDI, dual link HD-SDI, and 3G-SDI. The transmitters can insert line numbers for HD-SDI, dual link HD-SDI, and 3G-SDI. The transmitters can also generate and insert SMPTE 352M VPID packets into SD-SDI, HD-SDI, dual link HD-SDI, and 3G-SDI data streams. These packets are required for 3G-SDI and dual link HD-SDI.

Light versus Full-Featured

The triple-rate SDI TX is provided in both light and full-featured versions. Both versions can transmit all video formats compatible with SD-SDI, HD-SDI, dual link HD-SDI, and 3G-SDI (level A and level B). The main difference between the two is that the full-featured TX contains all the necessary mapping functions to support all video formats for all SDI modes while the light version does not contain the mapping functions for all video formats. Some video formats, such as 1080p 50 Hz and 60 Hz on 3G-SDI level A, do not require mapping, so the light version supports those formats directly. In some applications, the mapping function might be done at some other point in the datapath. The video formats that do not require mapping and can be handled directly with the light version of the reference design are:

- SD-SDI: NTSC and PAL
- HD-SDI: all formats
- 3G-SDI level A: 1080p 50 Hz, 59.94 Hz, and 60 Hz
- 3G-SDI level B: two independent HD-SDI streams

All other formats require either the full-featured triple-rate SDI TX or mapping implemented prior to the light triple-rate SDI TX.

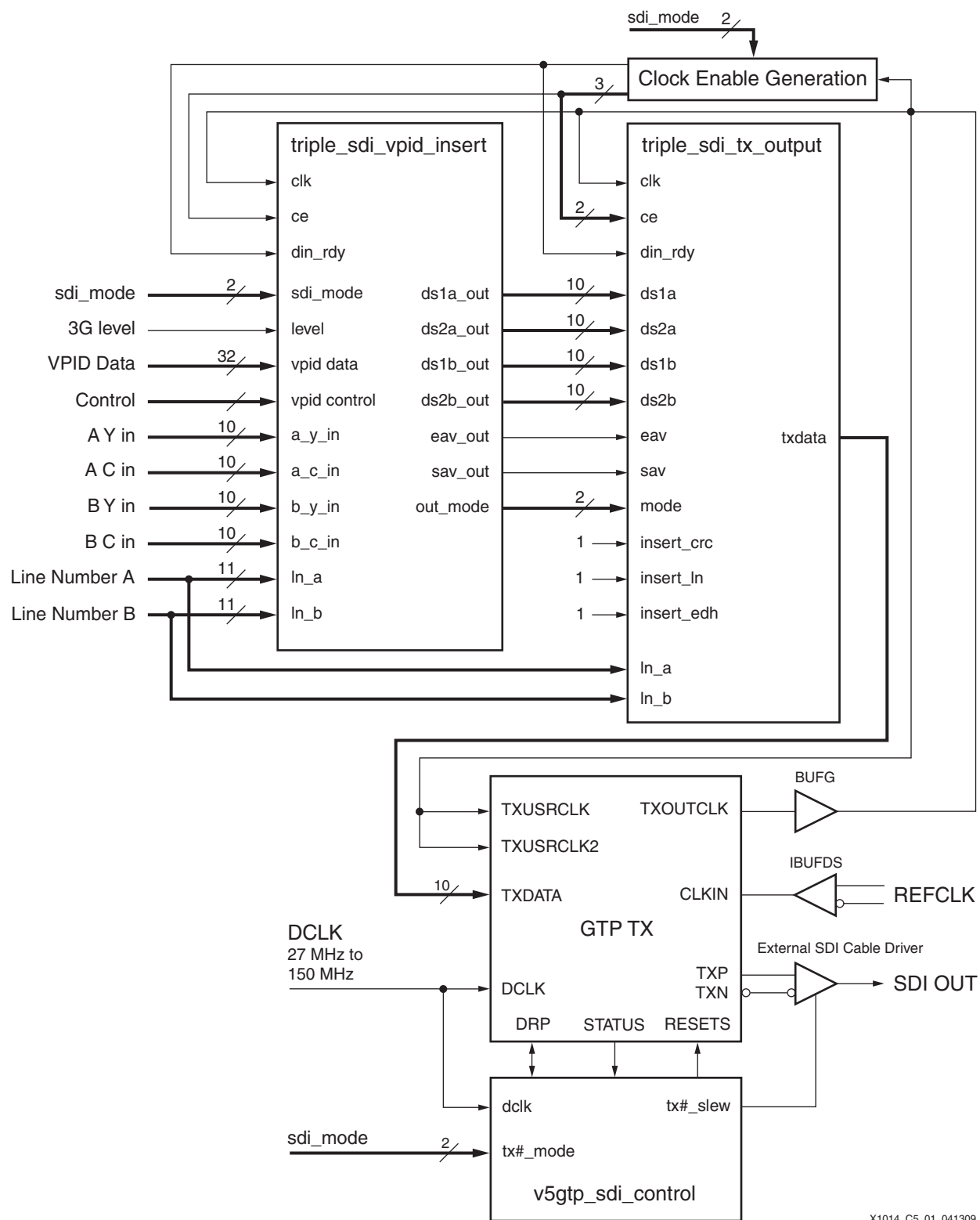
Note: The full-featured TX does not contain the mapping function to map SMPTE 428-1 video into a serial digital container, as described in SMPTE 428-9, for transport via 3G-SDI or dual link HD-SDI.

Light Triple-Rate SDI Transmitter Reference Design

The light version of the triple-rate SDI transmitter provides the basic operations necessary to support SD-SDI, HD-SDI, dual link HD-SDI, and 3G-SDI transmission. It does not do video mapping for 3G-SDI or dual link HD-SDI. With the exception of 1080p 50 Hz, 59.94 Hz, and 60 Hz video carried on 3G-SDI level A, the data streams entering the transmitter must already be formatted into 3G-SDI data streams or dual link HD-SDI streams when running in 3G-SDI and dual link HD-SDI modes. The light triple-rate SDI transmitter has these features:

- Only two reference clock frequencies required to support all SDI modes:
 - 148.5 MHz or 74.25 MHz for SD-SDI at 270 Mb/s, HD-SDI at 1.485 Gb/s, and 3G-SDI at 2.97 Gb/s.
 - 148.5/1.001 MHz or 74.25/1.001 MHz for HD-SDI at 1.485/1.001 Gb/s and 3G-SDI at 2.97/1.001 Gb/s.
- Directly supports 3G-SDI level A transmission of 1080p 50 Hz, 59.94 Hz, and 60 Hz video.
- Transmits pre-formatted dual link HD-SDI streams via dual link HD-SDI or 3G-SDI level B.
- Supports all 3G-SDI level A compatible video formats with the addition of a 3G-SDI level A mapping module.
- Directly supports transmission of two independent HD-SDI streams via 3G-SDI level B.
- Optional EDH packet generation/updating for SD-SDI.
- Optional CRC generation and insertion for HD-SDI and 3G-SDI.
- SMPTE 352M VPID insertion can be enabled for 3G-SDI, HD-SDI, dual link HD-SDI, and SD-SDI.
- Designed to interface with a 10-bit GTP RXDATA port to minimize global clocking resources. Only a single global clock is required for the transmitter. No DCMs or PLLs are required.

Figure 6-1 is a block diagram of the light version of the triple-rate SDI TX.



X1014_C5_01_041309

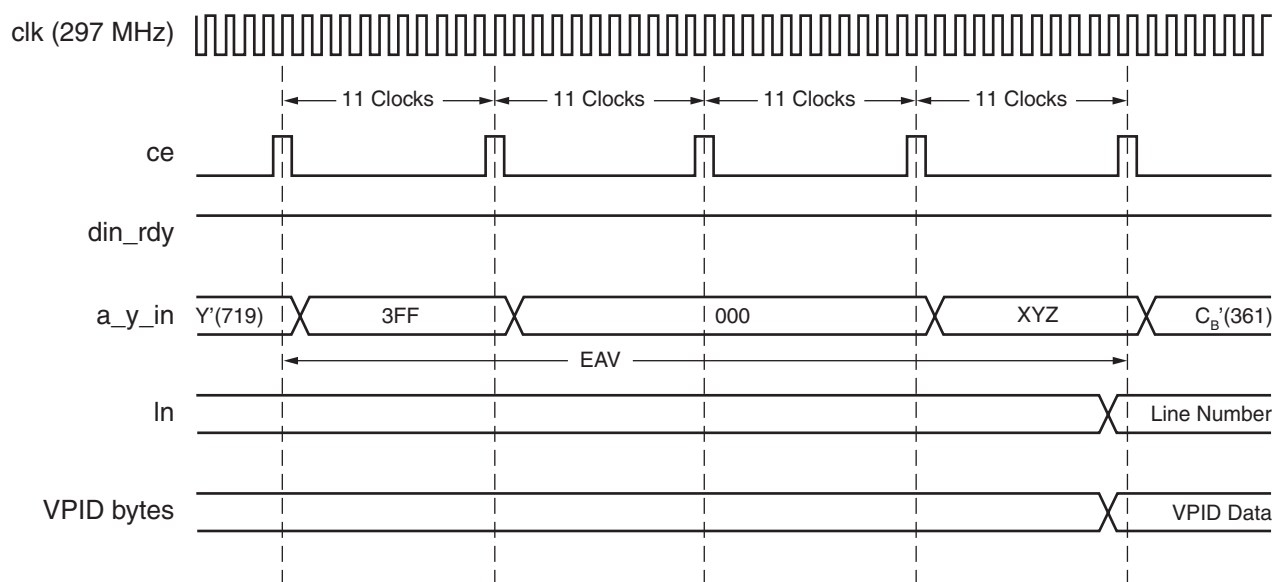
Figure 6-1: Light Triple-Rate SDI Transmitter Block Diagram

The light version of the triple-rate SDI TX does not do video mapping for 3G-SDI or dual link HD-SDI. The data streams entering the transmitter in 3G-SDI mode must already be formatted into legal 3G-SDI or dual link HD-SDI data streams, with the exception that the transmitter can insert SMPTE 352M VPID packets into the data streams. 1080p 50 Hz, 59.94 Hz, and 60 Hz video do not require any mapping for 3G-SDI level A and can be input directly to the `triple_sdi_vpid_insert` module.

The transmitter datapath consists of two modules: `triple_sdi_vpid_insert` and `triple_sdi_tx_output`. These two modules, combined with the GTP wrapper and `v5gtp_sdi_control` modules, form the light triple-rate SDI TX reference design, as shown in [Figure 6-1](#). The `v5gtp_sdi_control` module is described in [Chapter 3, Implementing SMPTE Serial Digital Interfaces with RocketIO GTP Transceivers](#). [Chapter 3](#) also describes how to use the RocketIO wizard to create the GTP wrappers.

Light Triple-Rate Transmitter SD-SDI Operation

When running in SD-SDI mode, the TXOUTCLK from the GTP TX has a frequency of 297 MHz because the transmitter runs at 11 times the 270 Mb/s bit rate. The interleaved Y/C data stream is connected to the `a_y_in` port of the `triple_sdi_vpid_insert` module. The clock enable input to both modules (`triple_sdi_vpid_insert` and `triple_sdi_tx_output`) must be asserted for one clock cycle out of every eleven to clock the datapath at 27 MHz, as shown in [Figure 6-2](#). The `din_rdy` ports of both modules must always be held High in SD-SDI mode.



Notes:

The clk, ce, din_rdy, and In inputs are connected to the corresponding ports of both the triple_sdi_vpid_insert and triple_sdi_tx_output modules. All inputs except ce have an entire 27 MHz sample time in which to become stable. Data is only sampled on the inputs on the rising edge of clk when ce is High.

In SD-SDI mode, the line number value on the In input port is only used by the triple_sdi_vpid_insert module and must be stable starting with the XYZ word of the EAV and must remain valid through the entire HANC interval.

The VPID input bytes on the input of the triple_sdi_vpid_insert module must be stable beginning with the XYZ word of the EAV and must remain stable during the entire HANC interval on lines where SMPTE 352M packets are inserted.

X1014_C5_02_041309

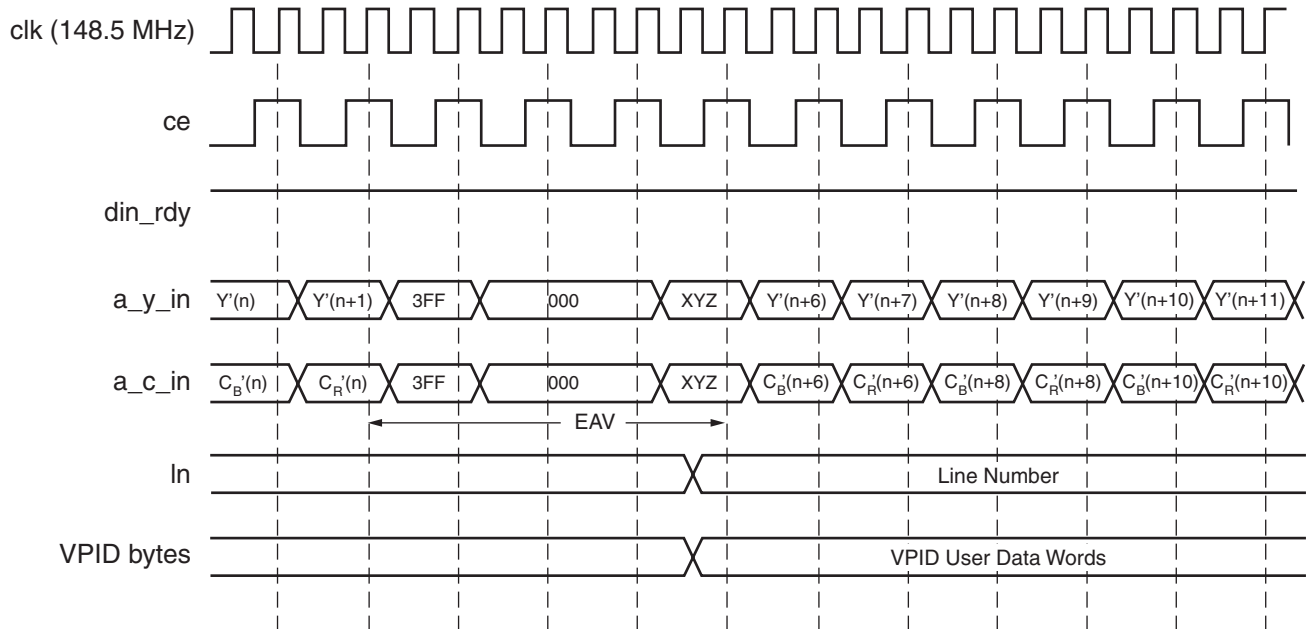
Figure 6-2: SD-SDI Input Timing for Light Triple-Rate SDI Transmitter

The triple_sdi_vpid_insert module optionally inserts SMPTE 352M VPID packets and outputs the modified data stream to the triple_sdi_tx_output module on the ds1a connection.

The triple_sdi_tx_output module inserts EDH packets if insert_edh is High. It then scrambles the data and replicates each scrambled bit eleven times. The scrambled and replicated data is output on the txdata port at 297 MHz. The GTP TX serializes the data for transmissions over the serial interface.

Light Triple-Rate Transmitter HD-SDI Operation

When running in HD-SDI mode, the TXOUTCLK from the GTP TX has a frequency of 148.5 MHz (or 148.5/1.001 MHz). The clock enable input to the datapath modules must be asserted every other clock cycle to produce a 74.25 MHz data rate, as shown in Figure 6-3. The din_rdy ports of both datapath modules must always be held High in HD-SDI mode.



Notes:

The clk, ce, din_rdy, and ln inputs are connected to the corresponding ports of both the triple_sdi_vpid_insert and triple_sdi_tx_output modules. All inputs except ce have a 74.25 MHz sample time in which to become stable. Data is only sampled on the inputs on the rising edge of clk when ce is High.

In HD-SDI mode, the line number value on the ln input port is used by both the triple_sdi_vpid_insert and the triple_sdi_tx_output modules and must be stable starting with the XYZ word of the EAV and must remain valid through the entire HANC interval.

The VPID input bytes on the input of the triple_sdi_vpid_insert module must be stable beginning with the XYZ word of the EAV and must remain stable during the entire HANC interval on lines where SMPTE 352M packets are inserted.

X1014_C5_03_041309

Figure 6-3: HD-SDI Input Timing for Light Triple-Rate SDI Transmitter

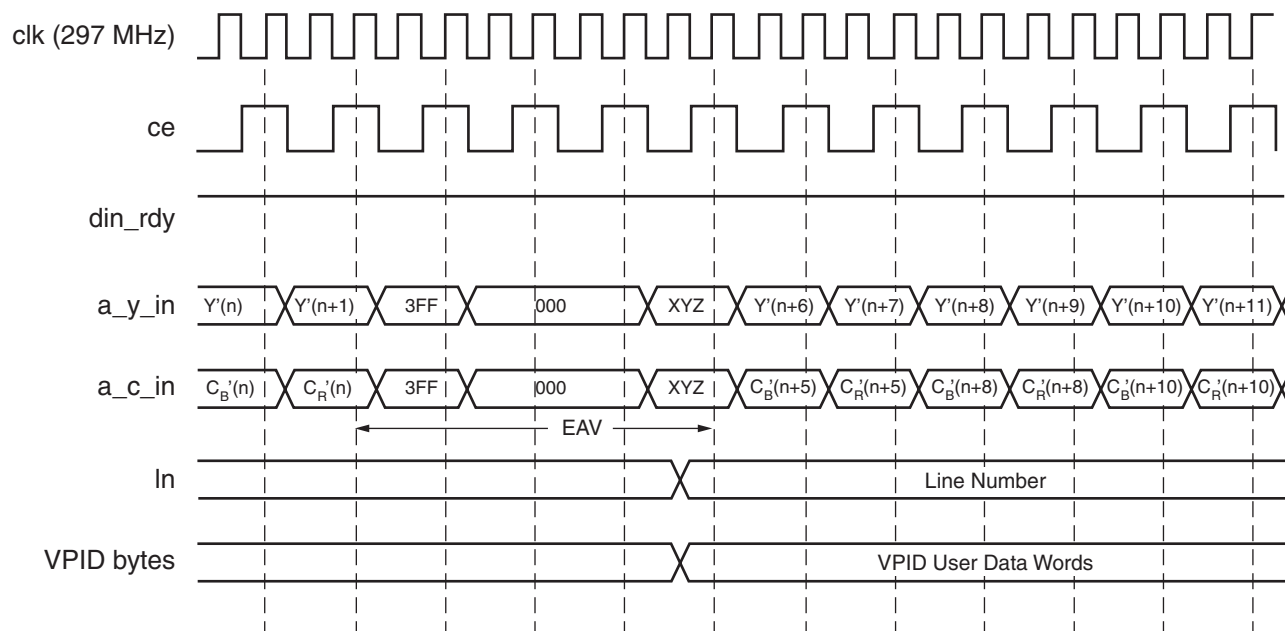
Data enters the triple_sdi_vpid_insert module as two 10-bit data streams with the Y data stream on the a_y_in port and the C data stream on the a_c_in port. The input data rate is 74.25 MHz. The triple_sdi_vpid_insert module optionally inserts SMPTE 352M VPID packets into the Y data stream before outputting the two data streams on the ds1a and ds2a ports to the triple_sdi_tx_output module.

The triple_sdi_tx_output module inserts line numbers into both data streams immediately after the EAV if insert_ln is High. If insert_crc is High, the module calculates and inserts CRC values immediately after the line numbers. The triple_sdi_tx_output module scrambles the data streams and outputs one 10-bit data stream running at 148.5 MHz on the txdata output port. The GTP TX serializes the data for transmissions over the serial interface.

Light Triple-Rate Transmitter 3G-SDI Operation

When running in 3G-SDI mode, the datapath modules can run in either level A mode or level B mode, as selected by the triple_sdi_vpid_insert module's level input (level A = Low, level B = High). The triple_sdi_vpid_insert module tells the triple_sdi_tx_output module which level is selected through the out_mode port.

In 3G-SDI level A mode, the `triple_sdi_vpid_insert` module accepts two 10-bit data streams on its `a_y_in` (data stream 1) and `a_c_in` (data stream 2) input ports. The clock frequency is 297 MHz (or 297/1.001 MHz). The clock enable to the datapath modules must be asserted every other clock cycle, producing a 148.5 MHz data rate, as shown in [Figure 6-4](#). The `din_rdy` ports of the datapath modules must be held High.



Notes:

The clk, ce, din_rdy, and ln inputs are connected to the corresponding ports of both the triple_sdi_vpid_insert and triple_sdi_tx_output modules. All inputs except ce have a 148.5 MHz sample time in which to become stable. Data is only sampled on the inputs on the rising edge of clk when ce is High.

In 3G-SDI mode, the line number value on the In input port is used by both the triple_sdi_vpid_insert and the triple_sdi_tx_output modules and must be stable starting with the XYZ word of the EAV and must remain valid through the entire HANC interval.

The VPID input bytes on the input of the triple_sdi_vpid_insert module must be stable beginning with the XYZ word of the EAV and must remain stable during the entire HANC interval on lines where SMPTE 352M packets are inserted.

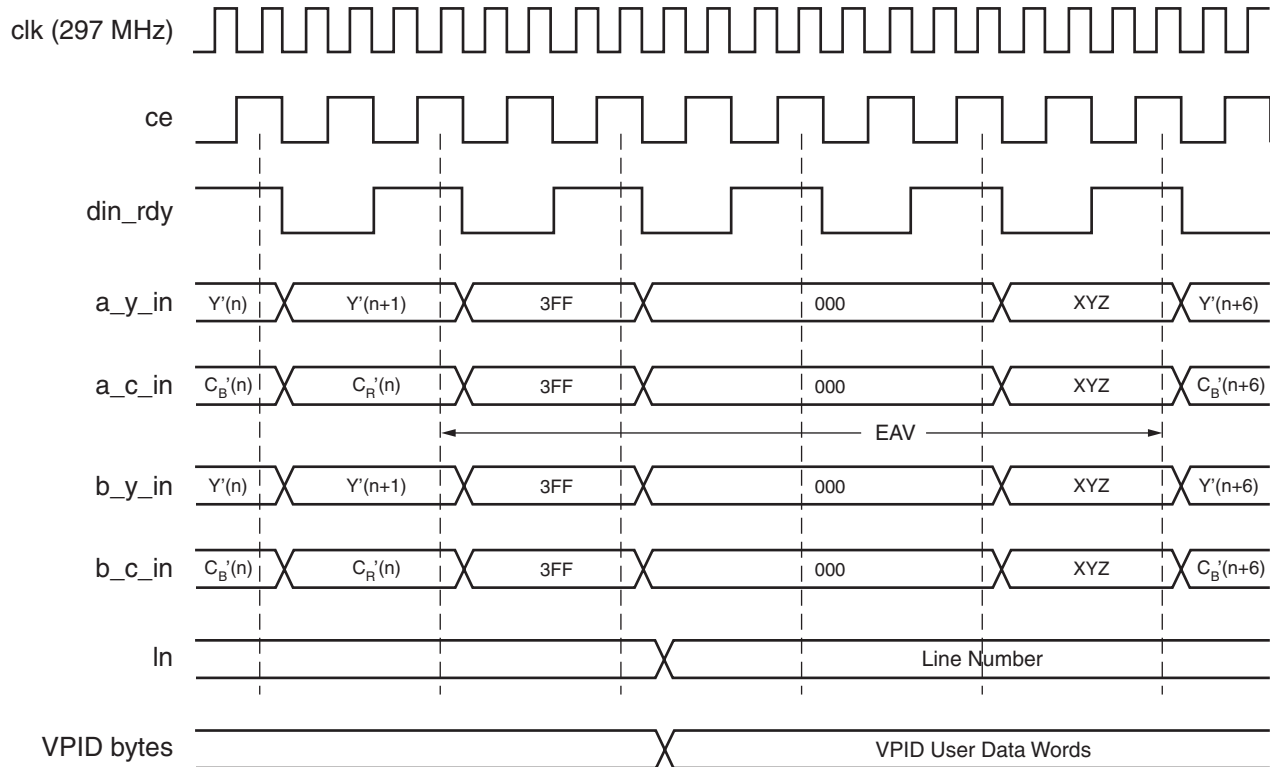
X1014 C5 04 041309

Figure 6-4: 3G-SDI Level A Input Timing for Light Triple-Rate SDI Transmitter

The `triple_sdi_vpid_insert` module inserts SMPTE 352M packets into both data streams before outputting the data streams on the `ds1a` and `ds2a` output ports to the `triple_sdi_tx_output` module. The `triple_sdi_tx_output` module inserts line numbers into both data streams immediately after the EAV if `insert_ln` is High. If `insert_crc` is High, the module calculates and inserts CRC values immediately after the line numbers. The `triple_sdi_tx_output` module scrambles the data streams and outputs one 10-bit data stream running at 297 MHz on the `txdata` output port. The GTP transmitter serializes the data for transmission over the serial interface.

In 3G-SDI level B mode, the `triple_sdi_vpid_insert` module requires four 10-bit data streams on its `a_y_in` (Y channel of link A), `a_c_in` (C channel of link A), `b_y_in` (Y channel of link B), and `b_c_in` (C channel of link B). These four data streams can be any SMPTE 372M dual link HD-SDI signal pre-formatted according to the SMPTE 372M

specification, or they can be two independent HD-SDI streams that are to be combined onto a single 3G-SDI level B link. The frequency of the GTP transceiver's TXOUTCLK is 297 MHz (or 297/1.001 MHz). The clock enable to the datapath modules must be asserted every other clock cycle to provide a basic data rate of 148.5 MHz. However, as shown in Figure 6-5, the input data rate is actually 74.25 MHz. The `din_rdy` input ports of the datapath module are used to control the 74.25 MHz input data rate. Data is only taken by the modules when both the clock enable and `din_rdy` are High. The AND of the `din_rdy` and clock enable signal must, therefore, only be asserted for one clock cycle out of every four. This is typically done by asserting `din_rdy` High for two clock cycles and then Low for two clock cycles – a 74.25 MHz square wave.



Notes:

The `clk`, `ce`, `din_rdy`, and `ln` inputs are connected to the corresponding ports of both the `triple_sdi_vpid_insert` and `triple_sdi_tx_output` modules. All inputs except `ce` and `din_rdy` have a 74.25 MHz sample time in which to become stable. Data is only sampled on the inputs on the rising edge of `clk` when `ce` and `din_rdy` are both High.

In 3G-SDI mode, the line number value on the `ln` input port is used by both the `triple_sdi_vpid_insert` and the `triple_sdi_tx_output` modules and must be stable starting with the XYZ word of the EAV and must remain valid through the entire HANC interval.

The VPID input bytes on the input of the `triple_sdi_vpid_insert` module must be stable beginning with the XYZ word of the EAV and must remain stable during the entire HANC interval on lines where SMPTE 352M packets are inserted.

X1014_C5_05_041309

Figure 6-5: 3G-SDI Level B Input Timing for Light Triple-Rate SDI Transmitter

The `triple_sdi_vpid_insert` module inserts SMPTE 352M VPID packets in the Y data streams of both link A and link B, as required by the SMPTE 425M specification. The

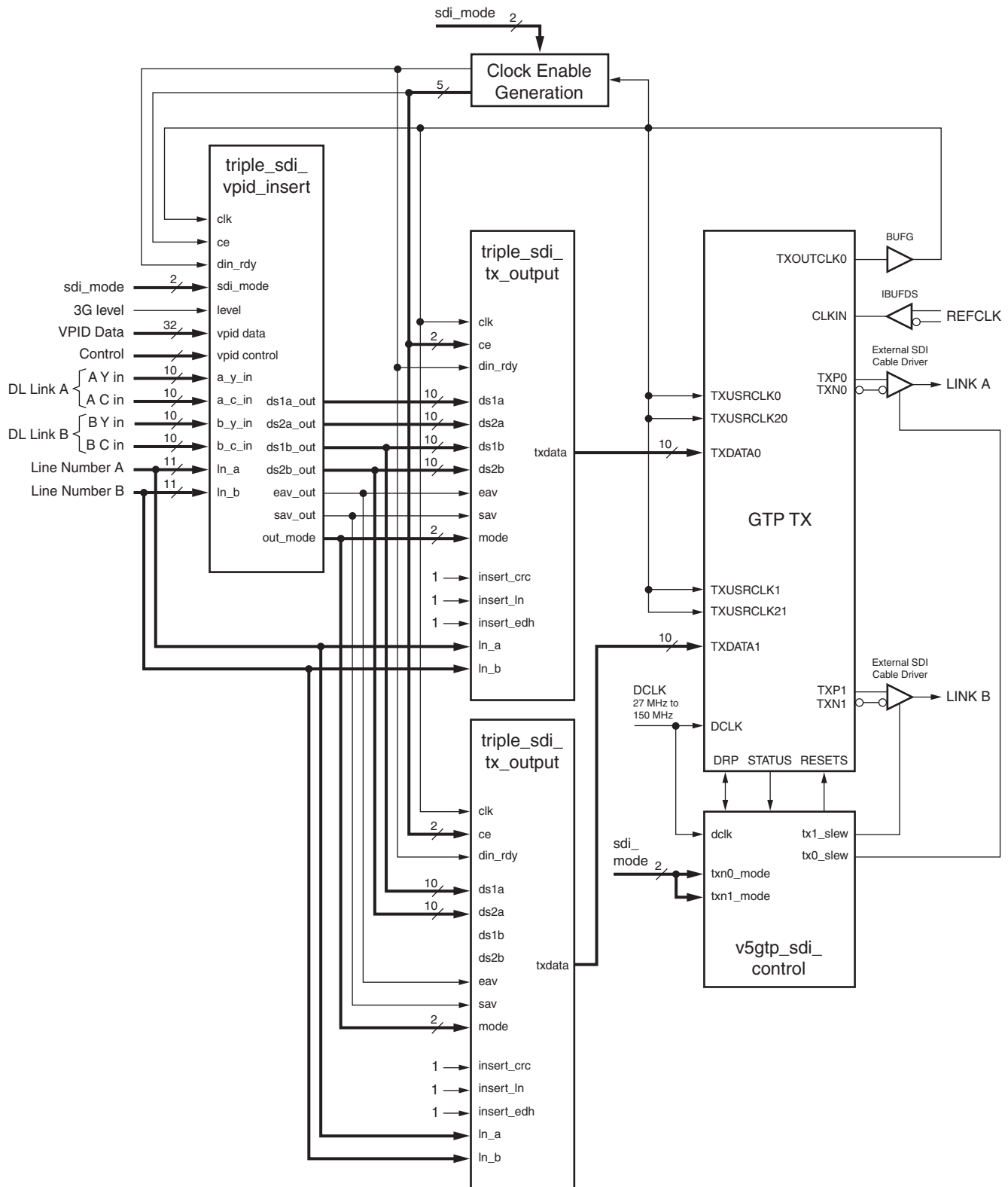
four data streams are output to the `triple_sdi_tx_output` module on the `ds1a`, `ds2a`, `ds1b`, and `ds2b` ports.

The `triple_sdi_tx_output` module inserts line numbers into all four data streams if `insert_ln` is High. The module calculates and inserts CRC values immediately after the line numbers in all four data streams if `insert_crc` is High. The data streams are then four-way interleaved to produce a single 10-bit data stream running at 297 MHz on the `txdata` output port. The GTP transmitter serializes the data for transmissions over the serial interface.

Light Triple-Rate Transmitter Dual Link HD-SDI Operation

The light triple-rate SDI transmitter does not contain the formatting logic to convert various video formats into SMPTE 372M dual link HD-SDI streams. However, preformatted dual link HD-SDI streams can be transmitted by a pair of light triple-rate SDI transmitters.

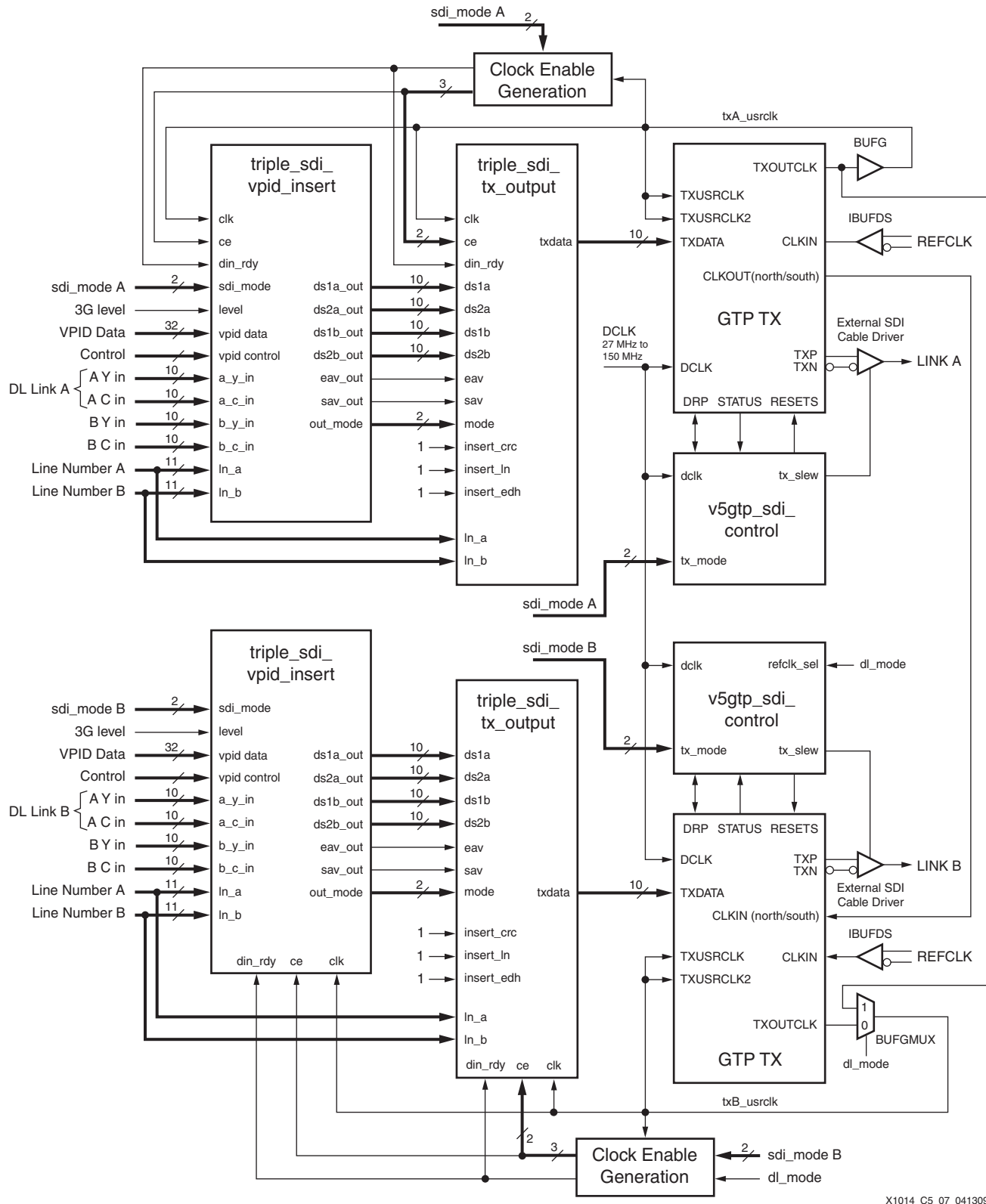
There are several ways to build a dual link HD-SDI transmitter using the modules that make up the light triple-rate SDI transmitter. The first method is best suited for applications in which the two transmitters are always paired as a dual link HD-SDI pair. In this method, shown in [Figure 6-6](#), the link A transmitter can be used in any SDI mode, but the link B transmitter is only used in dual link HD-SDI mode. A single `triple_sdi_vpid_insert` module inserts SMPTE 352M packets into the Y stream of both HD-SDI links of the dual link pair. These four data streams out of the VPID inserter are connected to two `triple_sdi_tx_output` modules with the A data streams going to one output module and the B data streams going to the other, as shown in [Figure 6-6](#). The B data streams also go to the link A output module because this is required for 3G-SDI level B operation. Because this method is best suited for applications where the link B transmitter is only used in dual link HD-SDI mode, both transmitters are shown in the same `GTP_DUAL` tile. It is also possible to use this method with GTP transmitters in separate tiles.



X1014_C5_06_041309

Figure 6-6: Example Dual Link HD-SDI Transmitter With Light Triple-Rate SDI Transmitter

The second method uses two independent triple-rate SDI transmitters, each with their own `triple_sdi_vpid_insert` module, that are paired to form a dual link HD-SDI pair only in dual link HD-SDI mode. In dual link HD-SDI mode, each transmitter runs as if it were a normal HD-SDI transmitter with one transmitter handling link A and the other handling link B. This method is shown in [Figure 6-7](#).



X1014_C5_07_041309

Figure 6-7: Example Dual Link HD-SDI Transmitter With Light Triple-Rate SDI Transmitter (Alternative Method)

There are several ways to handle clocking in with this method. The two transmitters must be driven by the same reference clock frequency because they must run at exactly the same bit rate. This can be done either by providing the same reference clock to the external reference clock inputs of the two GTP_DUAL tiles or by routing one reference clock between the two GTP_DUAL tiles using the dedicated GTP transceiver clock north/south routing resources. Because the two GTP transmitters are driven by the same reference clock, their TXOUTCLKs are frequency locked, although not necessarily in phase. Thus, the link A data streams must be synchronized to the tx_usrclk of link A, and the link B data streams must be synchronized to the tx_usrclk of link B. This synchronization must not introduce excessive skew between the two links, otherwise the maximum link skew between the two transmitters, as specified by SMPTE 372M, could be exceeded.

Alternatively, one of the transmitters could drive its tx_usrclk with a BUFGMUX so that, in dual link mode, its tx_usrclk is derived from the TXOUTCLK of the other GTP TX. This is shown in [Figure 6-7](#) where txB_usrclk is driven by a BUFGMUX. The advantage of this clock scheme is that the data streams do not need to be synchronized as they enter the two triple-rate SDI transmitters if the data stream source is driven by either one of the global tx_usrclk signals. The reference clock from the link A GTP_DUAL tile is routed through the GTP transceiver north/south clock routing structure to the other GTP_DUAL tile. The link B GTP transceiver can then use either the external reference clock connected to the FPGA at that tile or the reference clock from the link A GTP_DUAL tile. The reference clock used by the link B GTP TX is dynamically selected by the v5gtp_sdi_control module through the DRP port under control of the dl_mode signal (dual link mode), which is connected to the v5gtp_sdi_control module's refclk_sel port. One other consideration for this method is that the link B clock enables must be in phase with the link A clock enables when running in dual link HD-SDI mode.

Regardless of which method is used to construct the dual link HD-SDI transmitter, the input timing is identical to normal HD-SDI mode, as shown in [Figure 6-3, page 189](#). The ln_b line number port of the triple_sdi_vpid_insert module(s) is ignored in dual link HD-SDI mode. The ln_b port is only used in 3G-SDI level B mode.

As required by the SMPTE 372M standard, the two HD-SDI links must be identified using bit 6 of byte 4 of the SMPTE 352M packets. This bit must be 0 in link A and 1 in link B. If a single VPID insert module is used as shown in [Figure 6-6](#), the triple_sdi_vpid_insert module has separate input ports for byte 4 for the two links, only for this purpose.

[Table 6-2](#) summarizes the input data rates and connections for all SDI modes. The last row shows an alternative mode of operation for dual link HD-SDI mode. Normally, in dual link HD-SDI mode, the transmitters operate in a manner identical to normal HD-SDI mode with the din_rdy held High and ce asserted at a 74.25 MHz rate. However, it is also possible in dual link HD-SDI mode to hold ce High and toggle din_rdy at a 74.25 MHz rate as shown in the bottom row of [Table 6-2](#).

Table 6-2: Light Triple-Rate Transmitter Modes

sdi_mode	Level	TXOUTCLK (MHz)	ce (MHz)	din_rdy	Input Data Rate (MHz)	a_y_in	a_c_in	b_y_in	b_c_in
00 HD-SDI and dual link HD-SDI	X	148.5 or 148.5/1.001	1 out of 2 (74.25)	High	74.25 or 74.25/1.001	Y	C		
01 SD-SDI	X	297	1 out of 11 (27)	High	27	Y/C			

Table 6-2: Light Triple-Rate Transmitter Modes (Cont'd)

sdi_mode	Level	TXOUTCLK (MHz)	ce (MHz)	din_rdy	Input Data Rate (MHz)	a_y_in	a_c_in	b_y_in	b_c_in
10 3G-SDI A	0	297 or 297/1.001	1 out 2 (148.5)	High	148.5 or 148.5/1.001	Data stream 1	Data stream 2		
10 3G-SDI B	1	297 or 297/1.001	1 out 2 (148.5)	74.25 MHz Square Wave	74.25 or 74.25/1.001	Link A Y	Link A C	Link B Y	Link B C
00 Alternative dual link HD-SDI mode	X	148.5 or 148.5/1.001	High	1 out of 2 (74.25 MHz)	74.25 or 74.25/1.001	Y	C		

Table 6-3 lists the ports of the `triple_sdi_vpid_insert` module. This module is the front end of the light triple-rate SDI transmitter. Its main function is to insert SMPTE 352M VPID packets into the data streams, but it also serves some additional functions required by the `triple_sdi_tx_output` module. These functions include detecting and identifying EAV and SAV sequences and generating the `out_mode` signals that connect to the mode port of the output module.

Table 6-3: Ports of `triple_sdi_vpid_insert` Module

Port Name	I/O	Width	Description
Inputs			
clk	In	1	This clock input must be driven by the same clock that drives the TXUSRCLK and TXUSRCLK2 ports of the GTP transmitter. It must have a frequency of 148.5 MHz or 148.5/1.001 MHz for HD-SDI, 297 MHz or 297/1.001 MHz for 3G-SDI, and 297 MHz for SD-SDI.
ce	In	1	The clock enable must be asserted at a 27 MHz rate for SD-SDI (one clock cycle out of 11), 74.25 MHz rate for HD-SDI (every other clock cycle), and 148.5 MHz rate for 3G-SDI (every other clock cycle), as shown in Table 6-2.
din_rdy	In	1	For SD-SDI, HD-SDI, and level A 3G-SDI, this input should be kept High at all times. For level B 3G-SDI, an input data sample is taken when both <code>ce</code> and <code>din_rdy</code> are High. Typically, this input is driven with a 74.25 MHz square wave (High for two clock cycles and Low for two clock cycles) in 3G-SDI level B mode.
rst	In	1	This is an asynchronous reset. The falling edge of this reset signal must meet the reset recovery time of all flip-flops relative to the next rising edge of <code>clk</code> . This input can be driven by the <code>tx#_fabric_reset</code> output of the <code>v5gtp_sdi_control</code> module that, when combined with proper timing constraints, results in proper timing of the reset signal to the synchronous elements it resets.
sdi_mode	In	2	This input port is used to select the SDI mode: 00 = HD-SDI (including dual link HD-SDI) 01 = SD-SDI 10 = 3G-SDI 11 = Invalid

Table 6-3: Ports of `triple_sdi_vpid_insert` Module (Cont'd)

Port Name	I/O	Width	Description
level	In	1	In 3G-SDI mode, this input determines whether the modules insert SMPTE 352M packets per level A (level = Low) or level B (level = High). Because these two 3G-SDI levels have quite different requirements for placement of SMPTE 352M packets, this input must be properly controlled, otherwise the data streams generated by the module will not be legal.
enable	In	1	When this input is High, SMPTE 352M packets are inserted into the data stream, otherwise the packets are not inserted.
overwrite	In	1	If this input is High, SMPTE 352M packets already present in the data streams are overwritten. If this input is Low, existing SMPTE 352M packets are not overwritten. Caution! When transmitting SMPTE 372M dual link using 3G-SDI level B, existing SMPTE 352M packets in the data streams must be updated to indicate that the transport interface is 3G-SDI rather than HD-SDI. The <code>triple_sdi_vpid_insert</code> module only updates the SMPTE 352M packets if <code>overwrite</code> is High.
byte1	In	8	This value is inserted as the first user data word of the SMPTE 352M packet. It must be valid during the entire HANC interval.
byte2	In	8	This value is inserted as the second user data word of the SMPTE 352M packet. It must be valid during the entire HANC interval.
byte3	In	8	This value is inserted as the third user data word of the SMPTE 352M packet. It must be valid during the entire HANC interval.
byte4a	In	8	This value is inserted as the fourth user data word of the SMPTE 352M packet. This word is used for the SMPTE 352M packets for SD-SDI, HD-SDI, and 3G-SDI level A. For 3G-SDI level B and dual link HD-SDI, this value is used for the SMPTE 352M packet inserted into the Y channel of link A only. The byte4a input must be valid during the entire HANC interval.
byte4b	In	8	This value is inserted as the fourth user data word of the SMPTE 352M packet that is inserted in the Y channel of link B for 3G-SDI level B and dual link HD-SDI only. This input value is not used for SD-SDI, HD-SDI, or 3G-SDI level A. The byte4b input must be valid during the entire HANC interval.
ln_a	In	11	The current line number must be provided to the module through this port. SD-SDI only uses 10-bit line numbers, so the MSB of the port must be 0 in SD-SDI mode. The line number must be valid at least one clock cycle before the start of the HANC space (by the XYZ word of the EAV) and must remain valid during the entire HANC space. This input is the only line number input used for SD-SDI, HD-SDI, and 3G-SDI level A. For 3G-SDI level B, a second line number input port, <code>ln_b</code> , is also provided.
ln_b	In	11	This is the second line number input port used only for 3G-SDI level B. This additional line number port allows the two unrelated (but horizontally synchronized) HD-SDI signals to be vertically unsynchronized when level B carries two independent HD-SDI signals. Because <code>ln_b</code> is always used in 3G-SDI level B mode, when the input data streams are SMPTE 372M dual link HD-SDI streams being transported on a 3G-SDI level B interface, <code>ln_b</code> must be equal to <code>ln_a</code> .

Table 6-3: Ports of `triple_sdi_vpid_insert` Module (Cont'd)

Port Name	I/O	Width	Description
line_f1	In	11	The SMPTE 352M packet for field 1 is inserted on the line indicated by this value. For progressive video, the SMPTE 352M packet for the frame is inserted on this line. The line_f1 input must be valid during the entire HANC interval.
line_f2	In	11	The SMPTE 352M packet for field 2 is inserted on the line indicated by this value. For progressive video, this input port is ignored (and the line_f2_en port must be held Low). The line_f2 input must be valid during the entire HANC interval.
line_f2_en	In	1	This input controls whether or not SMPTE 352M packets are inserted on the line indicated by line_f2. For interlaced video, this input must be High. For progressive video, this input must be Low. The line_f2_en input must be valid during the entire HANC interval. For progressive video transported on an interlaced transport, such as 1080p 60 Hz transported by either 3G-SDI level B or dual link HD-SDI, SMPTE 352M packets must be inserted into both fields of the interlaced transport streams, so this input must be High.
a_y_in	In	10	SD-SDI: The multiplexed Y/C data stream enters the module on this port. HD-SDI and 3G-SDI level A: The Y data stream enters the module on this port. 3G-SDI level B and dual link HD-SDI: The Y data stream of link A enters the module on this port.
a_c_in	In	10	HD-SDI and 3G-SDI level A: The C data stream enters the module on this port. 3G-SDI level B and dual link HD-SDI: The C data stream of link A enters the module on this port.
b_y_in	In	10	For 3G-SDI level B and dual link HD-SDI, the Y data stream of link B enters the module on this port. When two <code>triple_sdi_vpid_insert</code> modules are used to process dual link HD-SDI, this input is not used.
b_c_in	In	10	For 3G-SDI level B and dual link HD-SDI, the C data stream of link B enters the module on this port. When two <code>triple_sdi_vpid_insert</code> modules are used to process dual link HD-SDI, this input is not used.
Outputs			
ds1a_out	Out	10	This data stream output is the same as the a_y_in data stream, but with SMPTE 352M packets inserted.
ds2a_out	Out	10	This data stream output is the same as the a_c_in data stream, but with SMPTE 352M packets inserted.
ds1b_out	Out	10	This data stream output is the same as the b_y_in data stream, but with SMPTE 352M packets inserted.
ds2b_out	Out	10	This data stream output is the same as the b_c_in data stream, but with SMPTE 352M packets inserted.
eav_out	Out	1	This output is High when the XYZ word of an EAV is output on the data stream outputs.
sav_out	Out	1	This output is High when the XYZ word of an SAV is output on the data stream outputs.
out_mode	Out	2	This output port must be connected to the mode input port of the <code>triple_sdi_tx_output</code> module.

Table 6-4 lists the ports of the `triple_sdi_tx_output` module. This module is the back end of the triple-rate SDI transmitter (both light and full-featured versions). It takes in one, two, or four data streams, depending on the SDI mode, from the `triple_sdi_vpid_insert` module, optionally generates and inserts EDH packets (SD-SDI only) or CRC and LN words (HD-SDI and 3G-SDI only), and then scrambles the data and outputs a single 10-bit data stream that is ready to be serialized by the GTP TX.

Table 6-4: Ports of `triple_sdi_tx_output` Module

Port Name	I/O	Width	Description
Inputs			
clk	In	1	This clock input must be driven by the same clock that drives the TXUSRCLK and TXUSRCLK2 ports of the GTP TX. The clock must have a frequency of 148.5 MHz or 148.5/1.001 MHz for HD-SDI, 297 MHz or 297/1.001 MHz for 3G-SDI, and 297 MHz for SD-SDI.
ce	In	2	These two identical clock enables must be asserted at a 27 MHz rate for SD-SDI (one cycle out of 11), a 74.25 MHz rate for HD-SDI (every other cycle), and a 148.5 MHz rate for 3G-SDI (every other cycle), as shown in Table 6-2.
din_rdy	In	1	For SD-SDI, HD-SDI, and level A 3G-SDI, this input should be kept High at all times. For level B 3G-SDI, an input data sample is taken when both ce and din_rdy are High. Typically, this input is driven with a 74.25 MHz square wave (High for two clock cycles and Low for two clock cycles) in 3G-SDI level B mode.
rst	In	1	This is an asynchronous reset. The falling edge of this reset signal must meet the reset recovery time of all flip-flops relative to the next rising edge of clk. This input can be driven by the tx#_fabric_reset output of the v5gtp_sdi_control module which, when combined with proper timing constraints, results in proper timing of the reset signal to the synchronous elements it resets.
mode	In	2	This input port is used to select the mode of operation of the <code>triple_sdi_tx_output</code> module. It is usually driven by the out_mode port of the <code>triple_sdi_vpid_insert</code> module. The operation modes are: 00 = HD-SDI, 3G-SDI level A, and dual link HD-SDI 01 = SD-SDI 10 = 3G-SDI level B 11 = Invalid
ds1a	In	10	This input is driven by the ds1a_out port of the <code>triple_sdi_vpid_insert</code> module. It carries the multiplexed Y/C data stream for SD-SDI, the Y data stream for HD-SDI, dual link HD-SDI, and 3G-SDI level A, and the Y data stream of link A for 3G-SDI level B.
ds2a	In	10	This input is driven by the ds2a_out port of the <code>triple_sdi_vpid_insert</code> module. It carries the C data stream for HD-SDI, dual link HD-SDI, and 3G-SDI level A, and the C data stream of link A for 3G-SDI level B.
ds1b	In	10	This input is driven by the ds1b_out port of the <code>triple_sdi_vpid_insert</code> module. It carries the Y data stream of link B for 3G-SDI level B.
ds2b	In	10	This input is driven by the ds2b_out port of the <code>triple_sdi_vpid_insert</code> module. It carries the C data stream of link B for 3G-SDI level B.
insert_crc	In	1	When this input is High, CRC values are calculated and inserted into all data streams when running in HD-SDI and 3G-SDI modes. This input is ignored when running in SD-SDI mode.

Table 6-4: Ports of `triple_sdi_tx_output` Module (Cont'd)

Port Name	I/O	Width	Description
insert_ln	In	1	When this input is High, LN words are inserted after the EAVs in all data streams when running in HD-SDI and 3G-SDI modes. This input is ignored when running in SD-SDI mode.
insert_edh	In	1	When this input is High, EDH packets are generated and inserted into the SD-SDI data stream. This input is ignored when running in HD-SDI and 3G-SDI modes.
ln_a	In	1	This is the line number input and is required for HD-SDI and 3G-SDI when insert_ln is High. The line number input must be setup prior to the XYZ word of the EAV and must remain stable for at least two sample times. For HD-SDI, dual link HD-SDI, and level A 3G-SDI, this is the only line number input port used. For level B 3G-SDI, a second line number port, ln_b, is provided.
ln_b	In	11	When transmitting level B 3G-SDI and when the insert_ln signal is High, the line number on this port is inserted into the Y and C channels of link B. This allows the two independent HD-SDI signals carried by level B to be vertically unsynchronized. Because ln_b is always used in 3G-SDI level B mode, when the input data streams are SMPTE 372M dual link HD-SDI streams, ln_b must be equal to ln_a.
eav	In	1	This input must be High when the XYZ word of each EAV enters the module on the data stream inputs. The eav input is not required for SD-SDI. This port is typically driven by the eav_out port of the <code>triple_sdi_vpid_insert</code> module.
sav	In	1	This input must be High when the XYZ word of each SAV enters the module on the data stream inputs. This input is not required for SD-SDI. This port is typically driven by the sav_out port of the <code>triple_sdi_vpid_insert</code> module.
Output			
txdata	Out	10	The scrambled SDI data stream is output on this port. The data should be directly connected to the TXDATA port of the GTP TX. For SD-SDI, this data is 11X oversampled data running at 297 MHz. For HD-SDI, the data rate is 148.5 MHz, and for 3G-SDI, the data rate is 297 MHz.

Light Triple-Rate Transmitter Details

This section provides details about the operation of the light triple-rate SDI transmitter reference design.

SMPTE 352M Packet Insertion

The `triple_sdi_vpid_insert` module can insert SMPTE 352M VPID packets into SD-SDI, HD-SDI, dual link HD-SDI, and 3G-SDI (both level A and level B) streams. The dual link HD-SDI and 3G-SDI standards require SMPTE 352M packets to be inserted into the data streams. The packets are optional in SD-SDI and HD-SDI.

The `triple_sdi_vpid_insert` module is a wrapper around a pair of `SMPTE352_vpid_insert` modules from the SMPTE 352M reference design. The behavior of this module is the same as the behavior of the underlying `SMPTE352_vpid_insert` module. The behavior of this module should be reviewed, as described in [Chapter 26, SMPTE 352M Video Payload Identification Packet Processing](#).

[Chapter 26](#) also describes the required values for the user data words of the SMPTE 352M packets.

SMPTE 352M packets are inserted on one designated video line in each frame if the transport is progressive and one designated video line in each field if the transport is interlaced. The designated video lines that carry SMPTE 352M packets vary depending on the SDI mode and the video format. These video lines are detailed in [Chapter 26](#). The line_f1 input port determines the line in the progressive frame or in the first interlaced field where the module inserts a SMPTE 352M packet. The line_f2 input port determines the line in the second interlaced field where the module inserts the SMPTE 352M packet. The line_f2_en input determines whether packets are inserted in the line specified by line_f2. This input must be Low for a progressive transport and High for an interlaced transport.

Note: It is very important that the line_f2_en input to the `triple_sdi_vpid_insert` module be controlled correctly. This input must be High for interlaced video and Low for progressive video. However, this input must actually be controlled based on whether the transport is interlaced or progressive, not the picture. The 1080p 50 Hz and 60 Hz 4:2:2 10-bit video formats, when carried by dual link HD-SDI or 3G-SDI level B (but not 3G-SDI level A) are transported in an interlaced manner, even though the picture is progressive. The same is true for progressive segmented frame transport. It is essential that the SMPTE 352M packets be inserted into both fields of the transport data streams for these formats. Some receiving equipment fails to lock properly if the SMPTE 352M packets are only present in one field rather than in both fields of interlaced transports.

Dual link HD-SDI data streams coming from a dual link SDI receiver might already have SMPTE 352M packets in the Y data streams of each link. SMPTE 372M mandates these packets, although not all equipment on the market properly inserts these required packets. If the packets are present, the first data word is probably 0x87, indicating that the data streams are carried on a dual link HD-SDI interface. When sending this dual link data on a 3G-SDI level B interface, the first word must be replaced with a value of 0x8A, indicating a SMPTE 372M signal carried on a 3G-SDI level B interface. Thus, the SMPTE 352M packets must be modified when sending the dual link HD-SDI streams on a 3G-SDI level B interface. Only the first byte of the VPID data and the CRC word of the packet must be modified. The values of the other bytes do not need to change, but must be captured first and applied to the VPID byte inputs of the `triple_sdi_vpid_insert` module so that they are reinserted when the packet is overwritten by the `triple_sdi_vpid_insert` module.

In 3G-SDI level B mode, the two independent HD-SDI signals carried by level B are allowed to be vertically unsynchronized if they are independent HD-SDI signals and not a dual link HD-SDI pair. If this is the case, SMPTE 352M packets are inserted independently on the two HD-SDI signals carried by the 3G-SDI level B interface. The second line number input port on the `triple_sdi_vpid_insert` module allows two separate line numbers to be provided for level B, one for each HD-SDI signal. However, because there is only one set of VPID data inputs to the insertion module, the SMPTE 352M packets, with the exception of byte 4, are identical. This means that the two HD-SDI streams must carry identical video formats. This is normally the case due to the restrictions in 3G-SDI level B requiring the two HD-SDI streams to be of the same format. If an application requires insertion of different VPID packets into the two HD-SDI signals carried by a 3G-SDI level B signal, the `triple_sdi_vpid_insert` module could easily be modified to provide two completely independent sets of VPID input data ports.

Line Number Insertion

3G-SDI and HD-SDI both require that line numbers be present in the two words that follow each EAV. The `triple_sdi_tx_output` module inserts those line numbers appropriately for HD-SDI and both levels of 3G-SDI (inserting them into all four data streams for level B 3G-SDI) if the insert_ln input is High. The line numbers to be inserted

are provided to the `triple_sdi_tx_output` module on the `ln_a` and `ln_b` inputs. In some cases, it may not be necessary or desirable to overwrite line numbers that are already present in the data streams. In that case, the `insert_ln` input can be driven Low and no line numbers are inserted. If the `insert_ln` input is hard-wired Low in the source code, the line number insertion logic is optimized out of the design by the synthesis tool.

For HD-SDI, dual link HD-SDI, and 3G-SDI level A, only the line number on `ln_a` is used. For 3G-SDI level B, the line number on `ln_a` is inserted into the Y and C data streams of link A and the line number on `ln_b` is inserted into the Y and C data streams of link B (again, only if `insert_ln` is asserted). For dual link HD-SDI, two `triple_sdi_tx_output` modules are used, one for each link. Each `triple_sdi_tx_output` module processes its HD-SDI data streams through the `ds1a` and `ds2a` import ports, just like normal HD-SDI mode. Thus, the `ln_b` input port is not used in dual link HD-SDI mode.

The SMPTE 425M specification does not specifically state that the two HD-SDI signals carried in level B mode have to be vertically synchronized (except that, when SMPTE 372M dual link HD-SDI is being carried, the two HD-SDI signals must be vertically synchronized). The transmitter reference design allows the two independent HD-SDI signals to be vertically unsynchronized by providing the second (`ln_b`) line number input port.

CRC Generation and Insertion

3G-SDI and HD-SDI both require that CRC values be present in the two words that follow the line numbers after the EAV. The `triple_sdi_tx_output` module calculates and inserts CRC values for each line for HD-SDI and both levels of 3G-SDI (inserting them into all four data streams for level B 3G-SDI) if the `insert_crc` input is High. In some cases, it might not be necessary or desirable to overwrite the CRC values already present in the data stream. In that case, the `insert_crc` input can be driven Low and no CRC values are inserted. If the `insert_crc` input is hard-wired Low, the CRC generation and insertion logic is optimized out of the design by the synthesis tool.

EDH Generation and Insertion

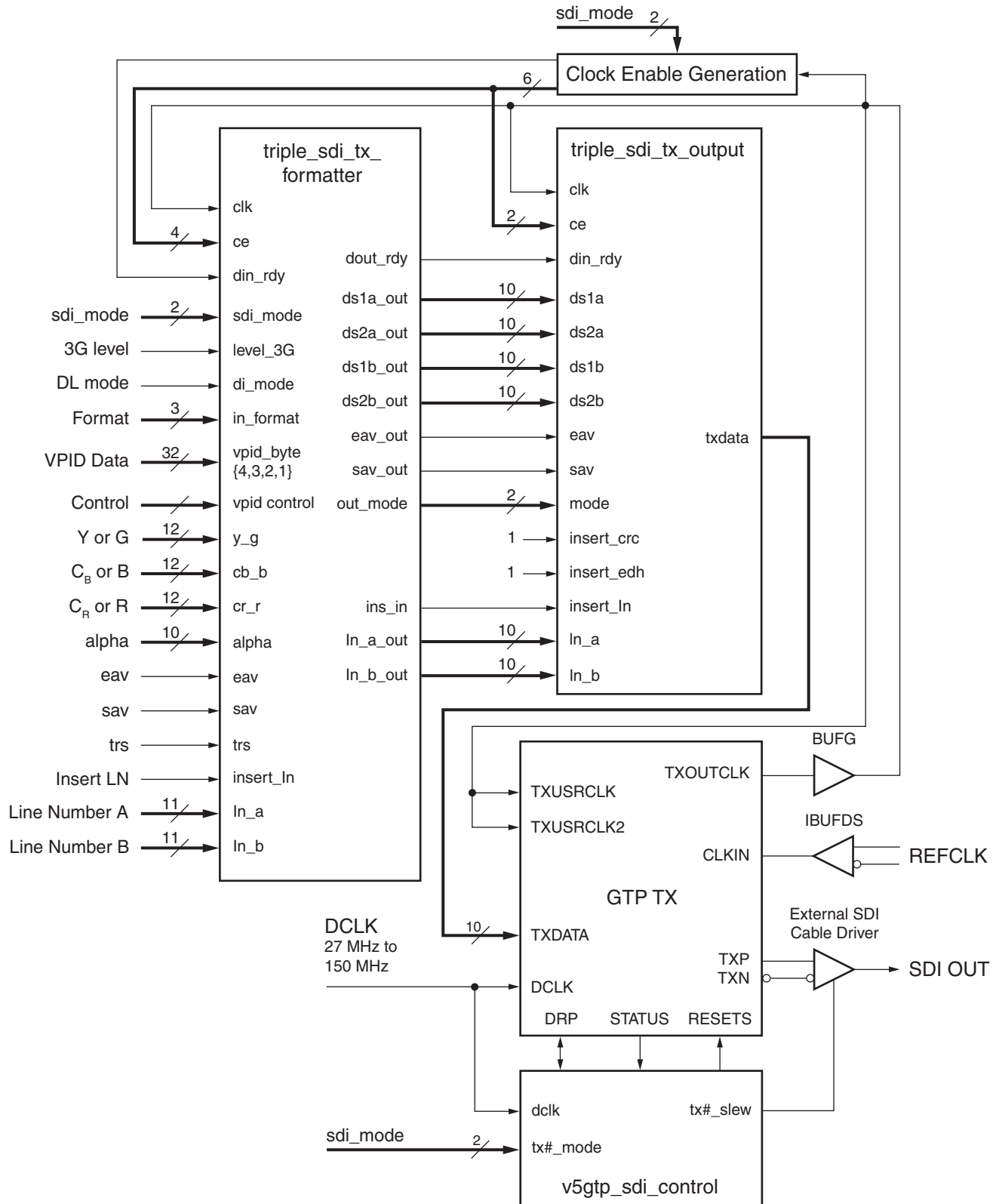
EDH packets are an option in SD-SDI and are never used for HD-SDI and 3G-SDI. The EDH packets contain CRC values that can be used to detect errors in the SD-SDI data stream. When running in SD-SDI mode, the `triple_sdi_tx_output` module generates and inserts EDH packets when `insert_edh` is High. If EDH is hard-wired Low, the synthesis tool optimizes the EDH generator out of the design. The EDH generator used in this reference design is the version that does not use the PicoBlaze™ processor, as described in the “SD-SDI Ancillary Data and EDH Processors” chapter of *Audio/Video Connectivity Solutions for Virtex-II Pro and Virtex-4 FPGAs* [Ref 5].

Full-Featured Triple-Rate SDI Transmitter Reference Design

The full-featured version of the triple-rate SDI transmitter provides a full set of features for a triple-rate SDI transmitter. It provides mapping functions to map native video into dual link HD-SDI, 3G-SDI level A, and 3G-SDI level B data streams. Native video, in this context, means digital video as formatted per SMPTE 274M or SMPTE 296M. The full-featured transmitter maps the native video according to the SMPTE 372M and SMPTE 425M standards to create data streams that can be transported on the dual link HD-SDI and 3G-SDI interfaces. This is the primary difference between the full-featured TX and the light TX. The light TX does not contain the modules that map the native video into dual link HD-SDI and 3G-SDI data streams, but the full-featured TX does. The full-featured triple-rate SDI transmitter has these features:

- Only two reference clock frequencies required:
 - 148.5 MHz or 74.25 MHz for SD-SDI at 270 Mb/s, HD-SDI at 1.485 Gb/s, and 3G-SDI at 2.97 Gb/s.
 - 148.5/1.001 MHz or 74.25/1.001 MHz for HD-SDI at 1.485/1.001 Gb/s and 3G-SDI at 2.97/1.001 Gb/s.
- Maps all native video formats supported by SMPTE 425M for 3G-SDI level A or level B transport.
- Maps all native video formats supported by SMPTE 372M for dual link HD-SDI transport.
- EDH packet insertion/updating for SD-SDI.
- CRC generation and insertion for HD-SDI and 3G-SDI.
- SMPTE 352M VPID insertion can be enabled for 3G-SDI, HD-SDI, dual link HD-SDI, and SD-SDI.
- Designed to interface with a 10-bit GTP RXDATA port to minimize global clocking resources. Only a single global clock is required for the transmitter. No DCMs or PLLs are required for implementations that use the built in GTP TX buffer.

Figure 6-8 is a block diagram of the full-featured triple-rate SDI TX. The full-featured triple-rate SDI transmitter looks very similar to the light triple-rate SDI transmitter. The primary difference is that the `triple_sdi_tx_formatter` module replaces the `triple_sdi_vpid_insert` module. The `triple_rate_vpid_insert` module is actually contained inside the `triple_sdi_tx_formatter` module.



X1014_C5_08_041309

Figure 6-8: Full-Featured Triple-Rate SDI Transmitter Block Diagram

The full-featured transmitter runs in five primary modes: SD-SDI, HD-SDI, dual link HD-SDI, 3G-SDI level A, and 3G-SDI level B. These modes are selected by the `sdi_mode`, `level_3G`, and `dl_mode` input ports, as shown in [Table 6-5](#). Some of these modes support many different input video formats, and the `in_format` port selects the proper video mapping algorithm in the `triple_sdi_tx_formatter` module.

[Table 6-5](#) shows the general input requirements for the various operating modes of the full-featured triple-rate SDI transmitter.

Table 6-5: Full Featured Triple-Rate Transmitter Modes

Mode	<code>sdi_mode</code>	<code>level_3G</code>	<code>dl_mode</code>	TXOUTCLK (MHz)	<code>ce</code> (MHz)	<code>din_rdy</code> (MHz)	Input Data Format and Sample Rate
HD-SDI	00	X	0	148.5 or 148.5/1.001	1 out of 2 (74.25)	Always 1	Y on <code>y_g[11:2]</code> C on <code>cb_b[11:2]</code> 74.25 MHz sample rate
Dual link HD-SDI with 1080p 50 Hz or 60 Hz	00	X	1	148.5 or 148.5/1.001	Always 1	Always 1	See Table 6-6 148.5 MHz sample rate
Dual link HD-SDI with other video formats	00	X	1	148.5 or 148.5/1.001	1 out of 2 (74.25)	Always 1	See Table 6-6 74.25 MHz sample rate
SD-SDI	01	X	X	297	1 out of 11 (27)	Always 1	Y/C interleaved on <code>y_g[11:2]</code> 27 MHz sample rate
3G-SDI A with 1080p 50 Hz or 60 Hz	10	0	X	297 or 297/1.001	1 out of 2 (148.5)	Always 1	See Table 6-6 148.5 MHz sample rate
3G-SDI A with other video formats	10	0	X	297 or 297/1.001	1 out of 2 (148.5)	74.25 square wave	See Table 6-6 74.25 MHz sample rate
3G-SDI B with 1080p 50 Hz or 60 Hz	10	1	1	297 or 297/1.001	1 out of 2 (148.5)	Always 1	See Table 6-6 148.5 MHz sample rate
3G-SDI B with other video formats	10	1	1	297 or 297/1.001	1 out of 2 (148.5)	74.25 square wave	See Table 6-6 74.25 MHz sample rate

[Table 6-6](#) shows more detailed input requirements for the various video formats supported by 3G-SDI and dual link HD-SDI.

Table 6-6: Full-Featured TX 3G-SDI and Dual Link HD-SDI Input Video Format Requirements

Video Format	<code>in_format</code>	Input Sample Rate (MHz)	<code>y_g</code>	<code>cb_b</code>	<code>cr_r</code>	<code>alpha</code>
4:2:2 10b 1080p 50 or 60 Hz	000	148.5	Y on [11:2]	C_B/C_R on [11:2]		
4:4:4 10b or 4:4:4:4 10b	001	74.25	Y on [11:2]	C_B on [11:2]	C_R on [11:2]	alpha 4:4:4:4 only

Table 6-6: Full-Featured TX 3G-SDI and Dual Link HD-SDI Input Video Format Requirements (Cont'd)

Video Format	in_format	Input Sample Rate (MHz)	y_g	cb_b	cr_r	alpha
4:4:4 12b	010	74.25	Y	C _B	C _R	
4:2:2 12b or 4:2:2:4 12b ⁽¹⁾	011	74.25	Y	C _B /C _R		alpha 4:2:2:4 only
Preformatted data streams (bypass video mapping)	1X0	74.25	Link A Y	Link A C	Link B C	Link B Y
Two independent HD-SDI signals (3G-SDI level B only)	1X1	74.25	HD-SDI #1 Y	HD-SDI #1 C	HD-SDI #2 Y	HD-SDI #2 C

Notes:

1. The 4:2:2:4 12b mode is supported by dual link HD-SDI in the SMPTE 372M document but is not supported by 3G-SDI level A in the SMPTE 425M document. The reference design supports 4:2:2:4 (4:2:2 12-bit video with a 10-bit alpha component) only in dual link HD-SDI mode.

Table 6-7 describes the ports of the `triple_sdi_tx_formatter` module. This module is the front end of the full-featured triple-rate SDI transmitter. This module maps the various video formats into SDI data streams and inserts SMPTE 352M VPID packets into the data streams.

Table 6-7: Ports of `triple_sdi_tx_formatter` Module

Port Name	I/O	Width	Description
Inputs			
clk	In	1	This clock input must be driven by the same clock that drives the TXUSRCLK and TXUSRCLK2 ports of the GTP TX. The clock must have a frequency of 148.5 MHz or 148.5/1.001 MHz for HD-SDI, 297 MHz or 297/1.001 MHz for 3G-SDI, and 297 MHz for SD-SDI.
ce	In	4	These four identical clock enables must be asserted at a 27 MHz rate for SD-SDI, a 74.25 MHz rate for HD-SDI, and a 148.5 MHz rate for 3G-SDI, as shown in Table 6-5. For dual link HD-SDI with 1080p 50 Hz or 60 Hz video, the ce input is held High. For other dual link HD-SDI video formats, the input is asserted at 74.25 MHz.
din_rdy	In	1	For all modes in which the clock enable frequency is the same as the input video rate, this input must be held High. For modes in which the clock enable frequency is different from the input video rate, this input must be asserted at the video data rate. See Table 6-5 for details.
rst	In	1	This is an asynchronous reset. The falling edge of this reset signal must meet the reset recovery time of all flip-flops relative to next rising edge of clk. This input can be driven by the tx#_fabric_reset output of the v5gtp_sdi_control module that, when combined with proper timing constraints, results in proper timing of the reset signal to the synchronous elements it resets.

Table 6-7: Ports of `triple_sdi_tx_formatter` Module (Cont'd)

Port Name	I/O	Width	Description
sdi_mode	In	2	<p>This input port is used to select the SDI mode:</p> <p>00 = HD-SDI 01 = SD-SDI 10 = 3G-SDI 11 = Invalid</p> <p>The <code>v5gtp_sdi_control</code> module also takes in these same two <code>sdi_mode</code> bits. Changing the <code>sdi_mode</code> value causes the <code>v5gtp_sdi_control</code> module to assert the <code>tx_fabric_reset</code> signal, thereby resetting major portions of the TX datapath. The GTP transceiver might also switch modes or reference clocks based on this value.</p>
level_3G	In	1	<p>In 3G-SDI mode (<code>sdi_mode</code> = 10), this input determines whether the transmitter generates level A (<code>level_3G</code> = Low) data streams or level B (<code>level_3G</code> = High) data streams. When running in 3G-SDI mode, changing this input immediately changes the video mapping algorithm in use.</p>
dl_mode	In	1	<p>In HD-SDI mode, this input determines whether the module runs in normal HD-SDI mode (<code>dl_mode</code> = Low) or dual link HD-SDI mode (<code>dl_mode</code> = High). When running in HD-SDI mode, changing this input immediately changes the video mapping algorithm in use.</p>
in_format	In	3	<p>In the dual link HD-SDI and 3G-SDI modes, this input port identifies the video format, as shown in Table 6-6, and selects the video mapping algorithm. When running in dual link HD-SDI and 3G-SDI modes, changing this input immediately changes the video mapping algorithm in use.</p>
color_space	In	2	<p>This input port specifies the color space of the video. This information is only used in 3G-SDI level A mode. For some 3G-SDI level A mapping modes, new 3G-SDI level A SAV sequences must be created and inserted into the data streams. This requires filling several words in the HANC space with blank video. The <code>color_space</code> input is used to control the value of those filler words, depending on which color space is currently being used. The encoding of this port is:</p> <p>00 = Y'C_B'C_R' 01 = R'G'B' 10 = XYZ 11 = Invalid</p> <p>This input port should be set up prior to the SAV sequence and remain valid until after the SAV sequence is complete. It is best to change this input only during the active portion of the line and not during the horizontal blanking interval.</p>

Table 6-7: Ports of `triple_sdi_tx_formatter` Module (Cont'd)

Port Name	I/O	Width	Description
alpha_act	In	1	<p>This input port indicates whether the alpha channel is active (alpha_act = High) or inactive (alpha_act = Low). The input port is only used in dual link HD-SDI and 3G-SDI modes.</p> <p>For some video formats, a fourth video component called alpha is permitted. This optional alpha component is always present in the dual link HD-SDI and 3G-SDI output data streams for those video formats that support alpha. This alpha component must be filled with legal values, even if the alpha component is inactive.</p> <p>For video formats supporting the alpha component, if the alpha_act input is Low, the alpha samples in the output video streams are automatically filled with values of 0x040, except where EAV and SAV sequences, including LN and CRC words, are inserted automatically by the module. If this input is High, valid alpha values must be supplied to the alpha input port. However, EAV and SAV sequences are not required to be present in the alpha data stream because the module creates and inserts them automatically, deriving the XYZ word from the XYZ word present on the y_g port. Thus, it is easy to fill alpha with a value other than the default value of 0x040 by supplying a constant fill value to the alpha input port and asserting the alpha_act input High.</p> <p>The alpha_act input is sampled on the rising edge of clock cycles when ce is High. After it is sampled by the module, alpha_act immediately takes effect.</p>
insert_ln	In	1	<p>This input enables insertion of line numbers into the data streams. Normally, line number insertion should always be enabled (High), but can be disabled (Low) for preformatted data streams that already contain line numbers.</p>
insert_vpid	In	1	<p>When this input is High, SMPTE 352M packets are inserted into the data stream, otherwise the packets are not inserted.</p>
overwrite_vpid	In	1	<p>If this input is High, SMPTE 352M packets already present in the data streams are overwritten. If this input is Low, existing SMPTE 352M packets are not overwritten.</p> <p>Caution! When transmitting SMPTE 372M dual link HD-SDI streams on a 3G-SDI level B transport, existing SMPTE 352M packets in the data streams must be updated to indicate that the transport interface is 3G-SDI rather than HD-SDI. The <code>triple_sdi_tx_formatter</code> module only updates the SMPTE 352M packets if overwrite is High.</p>
vpid_byte1	In	8	<p>This value is inserted as the first user data word of the SMPTE 352M packet. It must be valid during the entire HANC interval.</p>
vpid_byte2	In	8	<p>This value is inserted as the second user data word of the SMPTE 352M packet. It must be valid during the entire HANC interval.</p>
vpid_byte3	In	8	<p>This value is inserted as the third user data word of the SMPTE 352M packet. It must be valid during the entire HANC interval.</p>
vpid_byte4a	In	8	<p>This value is inserted as the fourth user data word of the SMPTE 352M packet. This word is used for the SMPTE 352M packets for SD-SDI, HD-SDI, and 3G-SDI level A. For 3G-SDI level B and dual link HD-SDI, this value is used for the SMPTE 352M packet inserted into the Y channel of link A only. The vpid_byte4a value must be valid during the entire HANC interval.</p>

Table 6-7: Ports of `triple_sdi_tx_formatter` Module (Cont'd)

Port Name	I/O	Width	Description
<code>vpid_byte4b</code>	In	8	This value is inserted as the fourth user data word of the SMPTE 352M packet that is inserted in the Y channel of link B for 3G-SDI level B and dual link HD-SDI. This input value is not used for SD-SDI, HD-SDI, or 3G-SDI level A. The <code>vpid_byte4b</code> value must be valid during the entire HANC interval.
<code>line_f1</code>	In	11	The SMPTE 352M packet for field 1 is inserted on the line indicated by this value. For progressive video, the SMPTE 352M packet for the frame is inserted on this line. The <code>line_f1</code> input must be valid during the entire HANC interval.
<code>line_f2</code>	In	11	The SMPTE 352M packet for field 2 is inserted on the line indicated by this value. For progressive video, this input port is ignored (and the <code>line_f2_en</code> port must be held Low). The <code>line_f1</code> input must be valid during the entire HANC interval.
<code>line_f2_en</code>	In	1	This input controls whether or not SMPTE 352M packets are inserted on the line indicated by <code>line_f2</code> . If the transport is interlaced, this input must be High. If the transport is progressive, this input must be Low. The <code>line_f2_en</code> input must be valid during the entire HANC interval.
<code>y_g</code>	In	12	SD-SDI: The interleaved Y/C components enter the module on this port. HD-SDI: The Y component enters the module on this port. Native video with 3G-SDI and dual link HD-SDI: The Y component enters the module on this port. Pre-formatted dual link HD-SDI data streams: The Y data stream of link A enters the module on this port. 3G-SDI level B carrying two independent HD-SDI signals: The Y component of the first HD-SDI signal enters the module on this port. 10-bit inputs must be connected to bits [11:2] of this port.
<code>cb_b</code>	In	12	SD-SDI: This port is unused. HD-SDI: The multiplexed C_B/C_R components enter the module on this port. Native video with 3G-SDI and dual link HD-SDI: For 4:4:4 video formats, the C_B component enters the module on this port. For 4:2:2 video formats, the interleaved C_B/C_R components enter the module on this port. Pre-formatted dual link HD-SDI data streams: The interleaved C_B/C data stream of link A enters the module on this port. 3G-SDI level B carrying two independent HD-SDI signals: The interleaved C_B/C_R components of the first HD-SDI signal enter the module on this port. 10-bit inputs must be connected to bits [11:2] of this port.
<code>cr_r</code>	In	12	SD-SDI: This port is unused. HD-SDI: This port is unused. Native video with 3G-SDI and dual link HD-SDI: For 4:4:4 video formats, the C_R component enters the module on this port. For 4:2:2 video formats, this port is unused. Pre-formatted dual link HD-SDI data streams: The interleaved C_B/C data stream of link B enters the module on this port. 3G-SDI level B carrying two independent HD-SDI signals: The interleaved C_B/C_R components of the second HD-SDI signal enter the module on this port. 10-bit inputs must be connected to bits [11:2] of this port.

Table 6-7: Ports of `triple_sdi_tx_formatter` Module (Cont'd)

Port Name	I/O	Width	Description
alpha	In	10	SD-SDI: This port is unused. HD-SDI: This port is unused. Native video with 3G-SDI and dual link HD-SDI: For video formats with an alpha component, the alpha component enters the module on this port. For video formats without an alpha component, this port is unused. Pre-formatted dual link HD-SDI data streams: The Y data stream of link B enters the module on this port. 3G-SDI level B carrying two independent HD-SDI signals: The Y component of the second HD-SDI signal enters the module on this port.
trs	In	1	This input must be High because all four words of each EAV or SAV enter the module on the video input ports. The trs input is ignored in SD-SDI mode, but is required in all other modes.
eav	In	1	This input must be High because the XYZ word of each EAV enters the module on the video input ports. The eav input is ignored in SD-SDI mode, but is required in all other modes.
sav	In	1	This input must be High because the XYZ word of each SAV enters the module on the video input ports. The sav input is ignored in SD-SDI mode, but is required in all other modes.
ln_a	In	11	The current line number must be provided to the module through this port. SD-SDI only uses 10-bit line numbers, so the MSB of the port must be 0 in SD-SDI mode. The line number must be valid at least one clock cycle before the start of the HANC space and must remain valid during the entire HANC interval. This input is the only line number input used for SD-SDI, HD-SDI, dual link HD-SDI, and 3G-SDI level A. For 3G-SDI level B, a second line number input port, ln_b, is also provided.
ln_b	In	11	This is the second line number input port used only for 3G-SDI level B. This additional line number port allows the two unrelated (but horizontally synchronized) HD-SDI signals to be vertically unsynchronized when level B carries two independent HD-SDI signals. This input port is used only when in_format is 101 or 111.
Outputs			
ds1a	Out	10	This output connects to the ds1a input port of the <code>triple_sdi_tx_output</code> module.
ds1b	Out	10	This output connects to the ds1b input port of the <code>triple_sdi_tx_output</code> module.
ds2a	Out	10	This output connects to the ds2a input port of the <code>triple_sdi_tx_output</code> module.
ds2b	Out	10	This output connects to the ds2b input port of the <code>triple_sdi_tx_output</code> module.
eav_out	Out	1	This output connects to the eav input port of the <code>triple_sdi_tx_output</code> module.
sav_out	Out	1	This output connects to the sav input port of the <code>triple_sdi_tx_output</code> module.

Table 6-7: Ports of `triple_sdi_tx_formatter` Module (Cont'd)

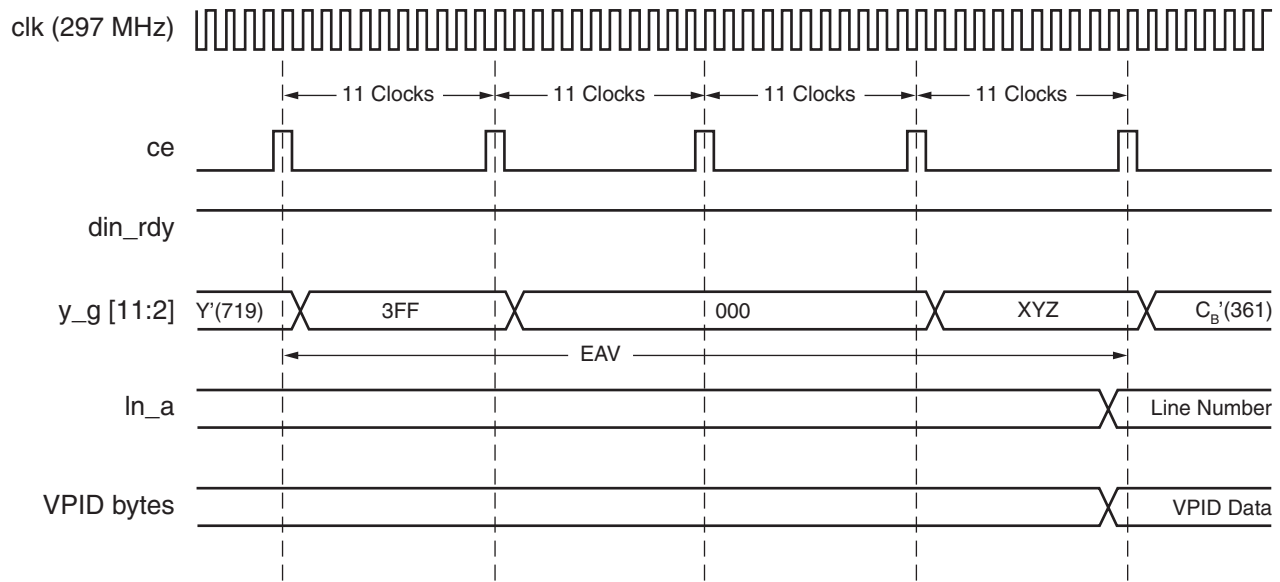
Port Name	I/O	Width	Description
out_mode	Out	2	This output connects to the mode input port of the <code>triple_sdi_tx_output</code> module.
ln_a_out	Out	11	This output connects to the ln_a input port of the <code>triple_sdi_tx_output</code> module.
ln_b_out	Out	11	This output connects to the ln_b input port of the <code>triple_sdi_tx_output</code> module.
ins_ln	Out	1	This output connects to the insert_ln input port of the <code>triple_sdi_tx_output</code> module.
dout_rdy	Out	1	This output connects to the din_rdy input port of the <code>triple_sdi_tx_output</code> module.

Table 6-4 in [Light Triple-Rate Transmitter SD-SDI Operation](#), page 187, describes the ports of the `triple_sdi_tx_output` module. This module is the back end of the triple-rate SDI transmitter for both the full-featured and light versions. This module takes the data streams from the `triple_sdi_tx_formatter` module, optionally generates and inserts EDH packets (SD-SDI only) or CRC and LN words (HD-SDI and 3G-SDI only), and then scrambles the data and outputs a single 10-bit data stream that is ready to be serialized. In some cases, the output module is not allowed to insert line numbers. In these cases, the `triple_sdi_tx_formatter` module inserts the line numbers itself and prevents the output module from inserting line numbers by driving its `ins_ln` output Low.

Full-Featured Triple-Rate Transmitter SD-SDI Operation

[Figure 6-9](#) shows the timing of the input signals to the full-featured triple-rate SDI TX running in SD-SDI mode. In SD-SDI mode, the clock frequency is 297 MHz. The clock enable must be asserted one clock cycle out of every 11, producing a 27 MHz data rate. This must always be exactly one clock cycle out of 11, not an average of one clock cycle out of 11.

The multiplexed Y/C components enter the `triple_sdi_tx_formatter` module on the 10 MSBs of the `y_g` input port. The line number must enter the `triple_sdi_tx_formatter` module on the 10 LSBs of its `ln_a` port. SMPTE 352M VPID insertion is optional and can be disabled by driving `insert_vpid` Low.



Notes:

The clk and ce inputs are connected to the corresponding ports of both the triple_sdi_tx_formatter and triple_sdi_tx_output modules. All inputs except ce have an entire 27 MHz sample time in which to become stable. Data is only sampled on the inputs on the rising edge of clk when ce is High.

In SD-SDI mode, the line number value on the ln_a input port is only used by the triple_sdi_tx_formatter module and must be stable starting with the XYZ word of the EAV and must remain valid through the entire HANC interval.

The VPID input bytes on the input of the triple_sdi_tx_formatter module must be stable beginning with the XYZ word of the EAV and must remain stable during the entire HANC interval on lines where SMPTE 352M packets are inserted.

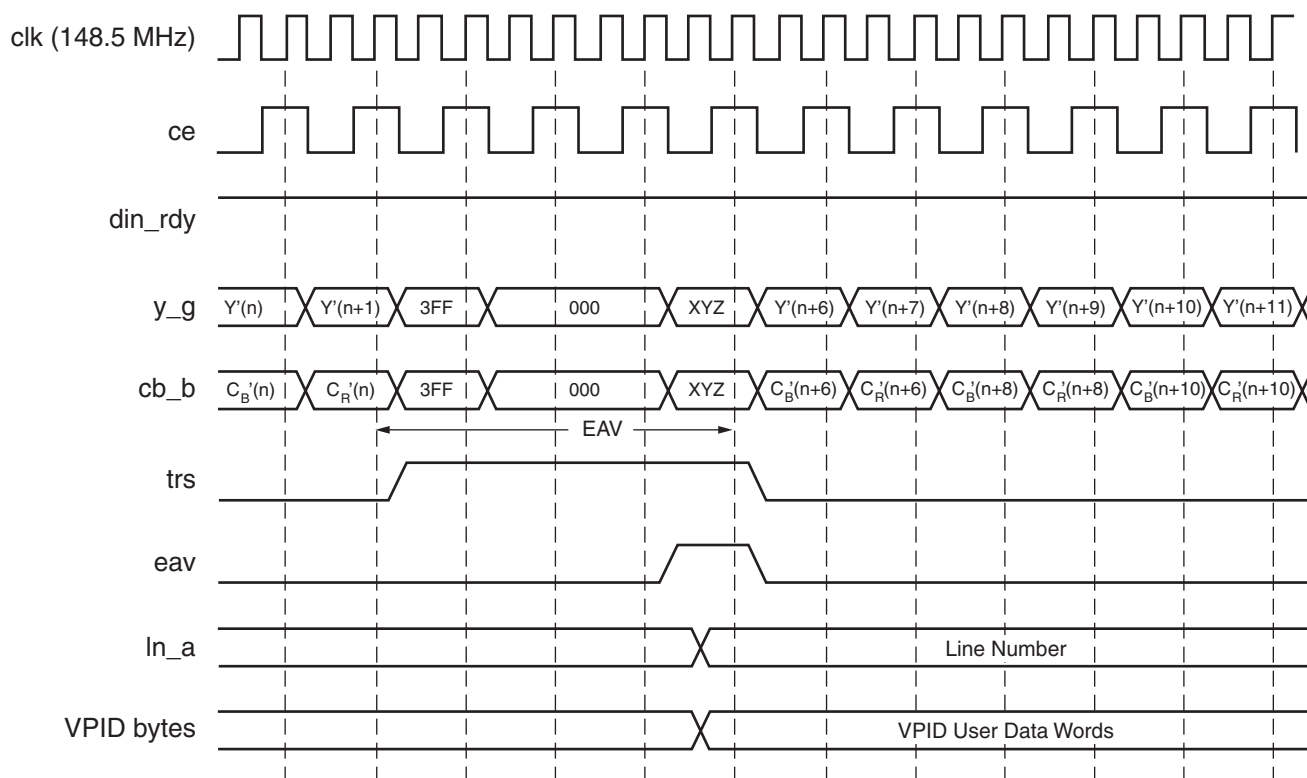
X1014_C5_09_041309

Figure 6-9: SD-SDI Input Timing for Full-Featured Triple-Rate SDI Transmitter

Full-Featured Triple-Rate Transmitter HD-SDI Operation

Figure 6-10 shows the timing of the input signals to the full-featured triple-rate SDI TX running in HD-SDI mode. In HD-SDI mode, the clock frequency is 148.5 MHz. The clock enable is asserted every other clock cycle, producing a data rate of 74.25 MHz. The din_rdy signal must be High all the time.

The Y component enters the triple_sdi_tx_formatter module on the 10 MSBs of the y_g input port. The multiplexed C_B and C_R components enter the triple_sdi_tx_formatter module on the 10 MSBs of the cb_b input port. Line number insertion should be enabled by driving insert_ln High and supplying the line number on the ln_a port. The trs, eav, and sav timing inputs must be driven appropriately. VPID insertion is optional and is disabled by driving the vpid_insert input Low



Notes:

The **clk** and **ce** inputs are connected to the corresponding ports of both the `triple_sdi_tx_formatter` and `triple_sdi_tx_output` modules. All inputs except **ce** have a 74.25 MHz sample time in which to become stable. Data is only sampled on the inputs on the rising edge of **clk** when **ce** is High.

In HD-SDI mode, the line number value on the **ln_a** input port is used by both the `triple_sdi_tx_formatter` and the `triple_sdi_tx_output` modules and must be stable starting with the XYZ word of the EAV and must remain valid through the entire HANC interval.

The VPID input bytes on the input of the `triple_sdi_tx_formatter` module must be stable beginning with the XYZ word of the EAV and must remain stable during the entire HANC interval on lines where SMPTE 352M packets are inserted.

Timing of an EAV sequence is shown in the diagram. The timing for an SAV sequence is the same except that the **sav** input must be asserted High instead of the **eav** input during the XYZ word of the SAV sequence.

X1014_C5_10_041309

Figure 6-10: HD-SDI Input Timing for Full-Featured Triple-Rate SDI Transmitter

Full-Featured Triple-Rate Transmitter Dual Link HD-SDI Operation

The `triple_rate_tx_formatter` module contains the necessary logic to map all SMPTE 372M compatible video formats into dual link HD-SDI data streams. This same mapping logic is used for 3G-SDI level B. In dual link HD-SDI mode, a single `triple_rate_tx_formatter` module provides data streams to two SDI TX output paths, each consisting of a `triple_sdi_tx_output` module and a GTP transmitter. The link B transmitter can either be used exclusively in dual link HD-SDI mode (in which case it can be placed in the same GTP_DUAL tile as the link A transmitter), or it can be a completely independent transmitter that is paired with the link A transmitter only in dual link HD-SDI mode.

Figure 6-11 shows how various modules are used to build an application in which the two triple-rate SDI transmitters can be used independently and paired to format a dual link HD-SDI transmitter. Two complete triple-rate SDI transmitters are used, each with its own GTP transmitter and `triple_sdi_tx_formatter` and `triple_sdi_tx_output` modules.

In modes other than dual link HD-SDI mode, the two transmitters work independently. They have their own TXUSRCLK driven by the TXOUTCLK of their respective GTP TX unit. However, in dual link HD-SDI mode, the `triple_sdi_tx_formatter` module for link A generates the streams for both transmitters and the `triple_sdi_tx_formatter` module for link B is not used. Multiplexers on certain input ports to the link B `triple_sdi_tx_output` module select whether the module is driven by the link A or link B `triple_sdi_tx_formatter` module. In dual link HD-SDI mode, the `ds1a` and `ds2a` data streams from the link A `triple_sdi_tx_formatter` module are processed by the link A `triple_sdi_tx_output` module while the `ds1b` and `ds2b` streams are processed by the link B `triple_sdi_tx_output` module.

A BUFGMUX is used to drive the TXUSRCLK for the link B transmitter. In dual link HD-SDI mode, the BUFGMUX selects the TXOUTCLK from the link A GTP TX unit. In other modes, when the transmitters are operating independently, the BUFGMUX selects the TXOUTCLK from the link B GTP TX unit.

Each transmitter must have its own clock enable generator so that the generators can operate independently. However, the clock enable generator for link B must synchronize with the link A clock enable generator in dual link HD-SDI mode so that the clock enables applied to the link B `triple_sdi_tx_output` modules are aligned with the link A clock enables. To reduce power consumption, the clock enables controlling the link B `triple_sdi_tx_formatter` module can be held Low in dual link HD-SDI mode, disabling this module when not in use.

Figure 6-12 shows the input timing for 1080p 50 Hz, 60 Hz, or 60/1.001 Hz video when running in dual link HD-SDI mode. In this mode, the frequency of TXUSRCLK is 148.5 MHz (or 148.5/1.001 MHz). The input data is two 10-bit components with a sample rate of 148.5 MHz. The clock enable is held High all the time. The `din_rdy` signal to the `triple_sdi_tx_formatter` module is also held High. This module produces four data streams, two for each link of the dual link pair. Thus, the output data rate of the `triple_sdi_tx_formatter` module is 74.25 MHz. The two `triple_sdi_tx_output` modules have their clock enable inputs held High, but their `din_rdy` inputs are driven at a 74.25 MHz rate by the `dout_rdy` port of the `triple_sdi_tx_formatter` module.

Figure 6-13 shows the input timing for 4:2:2 12-bit video for dual link HD-SDI mode. This video format has two 12-bit components and an input sample rate of 74.25 MHz. The TXUSRCLK frequency is 148.5 MHz. The clock enables to the `triple_sdi_tx_formatter` and `triple_sdi_tx_output` modules are asserted at a 74.25 MHz rate. The `triple_sdi_tx_formatter` module produces four data streams, two for each link. The `dout_rdy` signal from the `triple_sdi_tx_formatter` module is always High because the output data rate is 74.25 MHz – the same as the clock enable rate.

Note: SMPTE 372M also allows a fourth 10-bit alpha component to be included with this video format and this is supported by the module. This alpha component is not shown in Figure 6-13.

Figure 6-14 shows the input timing for 4:4:4 video with the dual link HD-SDI transmitter. The components can be either 10-bit or 12-bit. With 10-bit components, an optional fourth alpha component can be included, although it is not shown. The input sample rate is 74.25 MHz. The TXUSRCLK frequency is 148.5 MHz. The clock enables to the `triple_sdi_tx_formatter` and `triple_sdi_tx_output` modules are asserted at a 74.25 MHz rate. The `triple_sdi_tx_formatter` module produces four data streams,

two for each link. The dout_rdy signal from the triple_sdi_tx_formatter module is held High because the output data rate is 74.25 MHz – the same as the clock enable rate.

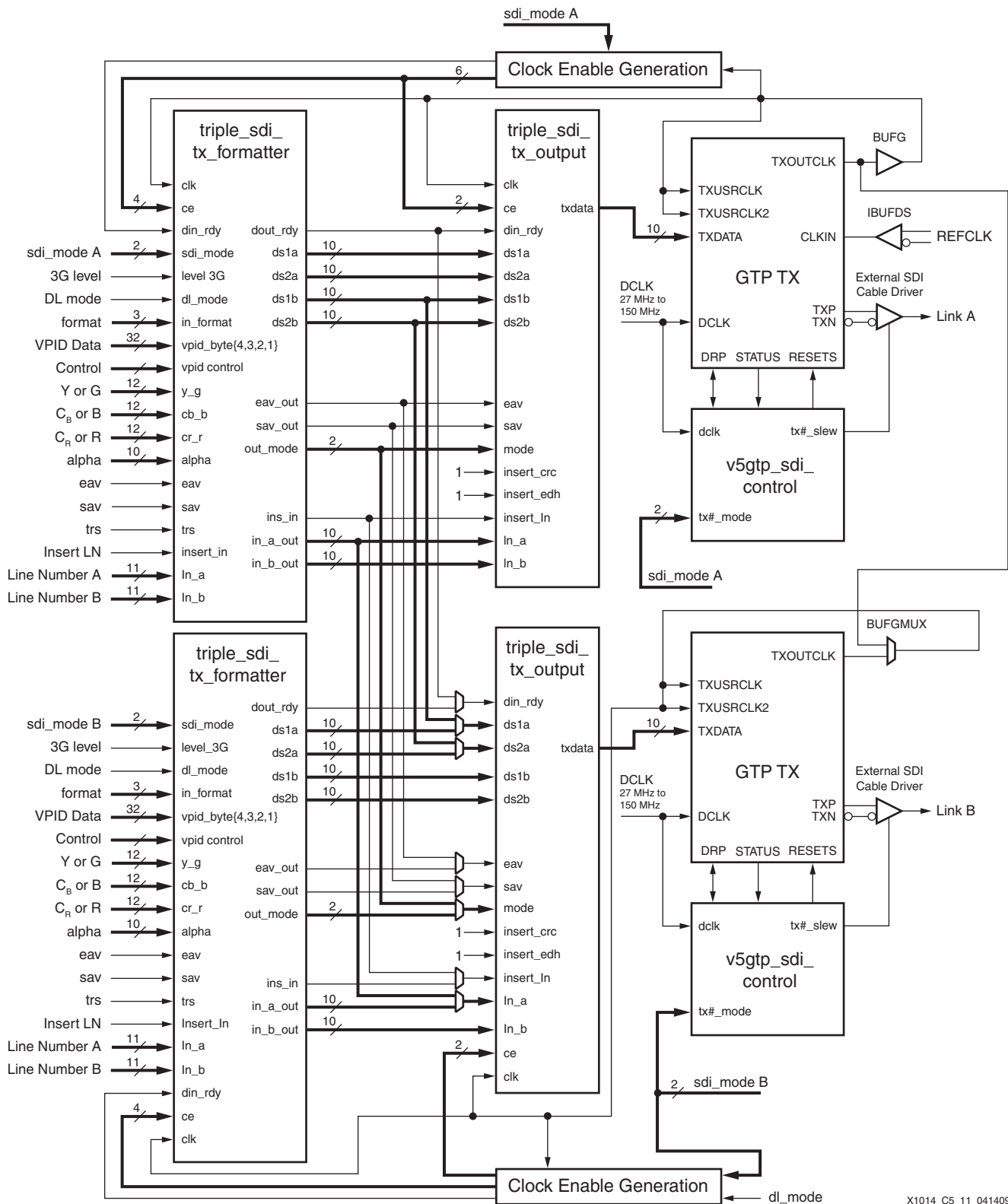
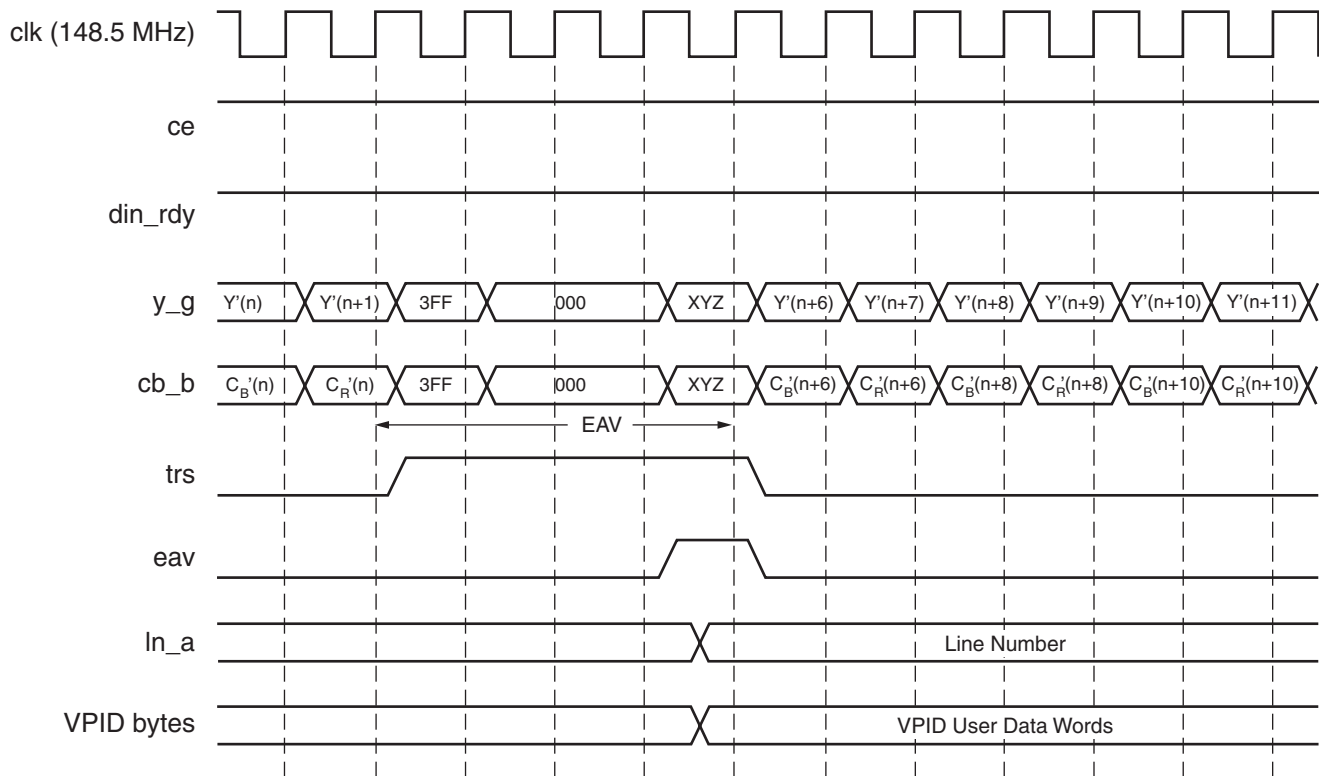


Figure 6-11: Dual Link HD-SDI Transmitter Using Full-Featured Triple-Rate SDI Transmitter

X1014_C05_11_041409



Notes:

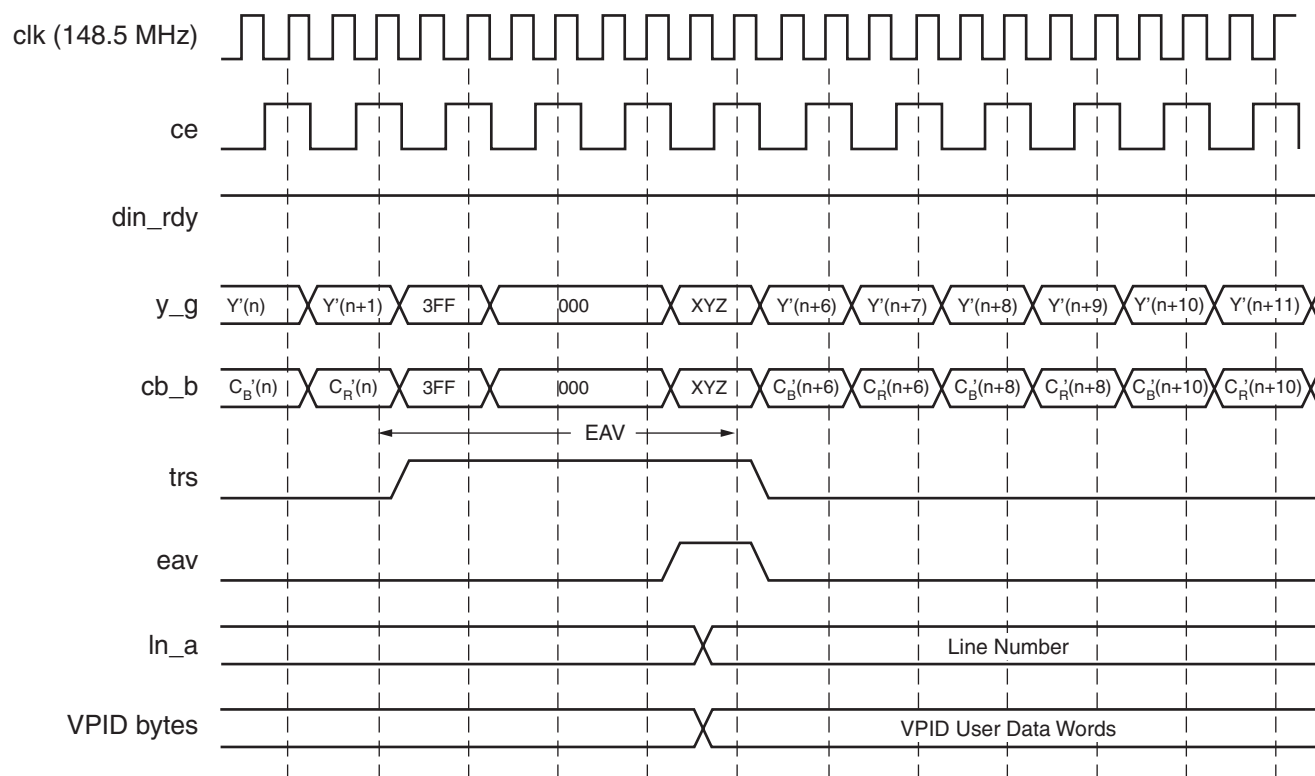
In dual link HD-SDI mode, the line number value on the `ln_a` input port is used by both the `triple_sdi_tx_formatter` and the `triple_sdi_tx_output` modules and must be stable starting with the XYZ word of the EAV and must remain valid through the entire HANC interval.

The VPID input bytes on the input of the `triple_sdi_tx_formatter` module must be stable beginning with the XYZ word of the EAV and must remain stable during the entire HANC interval on lines where SMPTE 352M packets are inserted.

The timing of an EAV sequence is shown in the diagram. The timing for an SAV sequence is the same except that the sav input must be asserted High instead of the eav input during the XYZ word of the SAV sequence.

X1014_C5_12_041309

Figure 6-12: Dual Link HD-SDI Input Timing for Full-Featured Triple-Rate SDI Transmitter (148.5 MHz Sample Rate)



Notes:

The clk and ce inputs are connected to the corresponding ports of both the triple_sdi_tx_formatter and triple_sdi_tx_output modules. All inputs except ce have a 74.25 MHz sample time in which to become stable. Data is only sampled on the inputs on the rising edge of clk when ce is High.

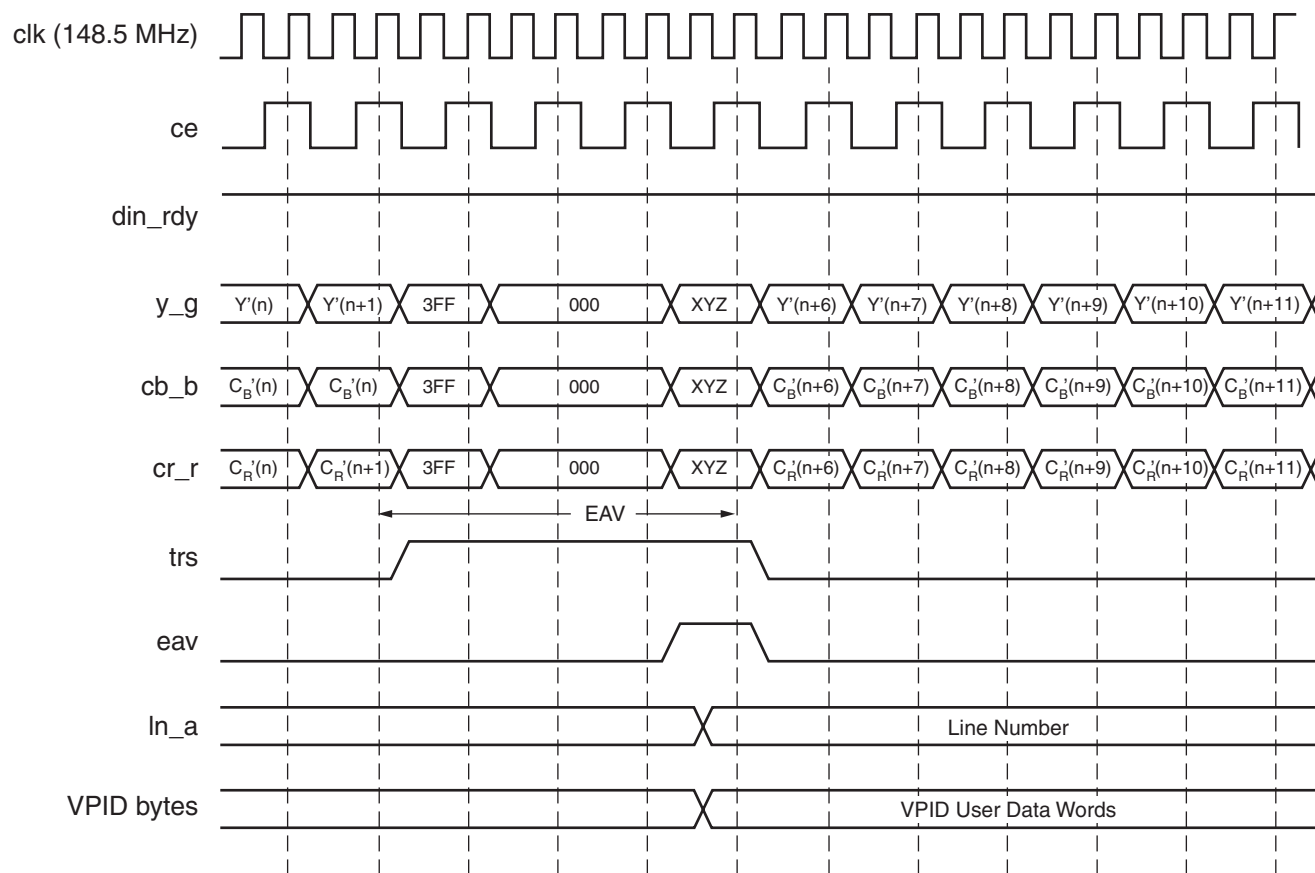
In HD-SDI mode, the line number value on the ln_a input port is used by both the triple_sdi_tx_formatter and the triple_sdi_tx_output modules and must be stable starting with the XYZ word of the EAV and must remain valid through the entire HANC interval.

The VPID input bytes on the input of the triple_sdi_tx_formatter module must be stable beginning with the XYZ word of the EAV and must remain stable during the entire HANC interval on lines where SMPTE 352M packets are inserted.

The timing of an EAV sequence is shown in the diagram. The timing for an SAV sequence is the same except that the sav input must be asserted High instead of the eav input during the XYZ word of the SAV sequence.

X1014_CS_13_041309

Figure 6-13: Dual Link HD-SDI Input Timing for Full-Featured Triple-Rate SDI Transmitter (4:2:2 12-Bit)



Notes:

The clk and ce inputs are connected to the corresponding ports of both the triple_sdi_tx_formatter and triple_sdi_tx_output modules. All inputs except ce have a 74.25 MHz sample time in which to become stable. Data is only sampled on the inputs on the rising edge of clk when ce is High.

In HD-SDI mode, the line number value on the ln_a input port is used by both the triple_sdi_tx_formatter and the triple_sdi_tx_output modules and must be stable starting with the XYZ word of the EAV and must remain valid through the entire HANC interval.

The VPID input bytes on the input of the triple_sdi_tx_formatter module must be stable beginning with the XYZ word of the EAV and must remain stable during the entire HANC interval on lines where SMPTE 352M packets are inserted.

The timing of an EAV sequence is shown in the diagram. The timing for an SAV sequence is the same except that the sav input must be asserted High instead of the eav input during the XYZ word of the SAV sequence.

X1014_C5_14_041309

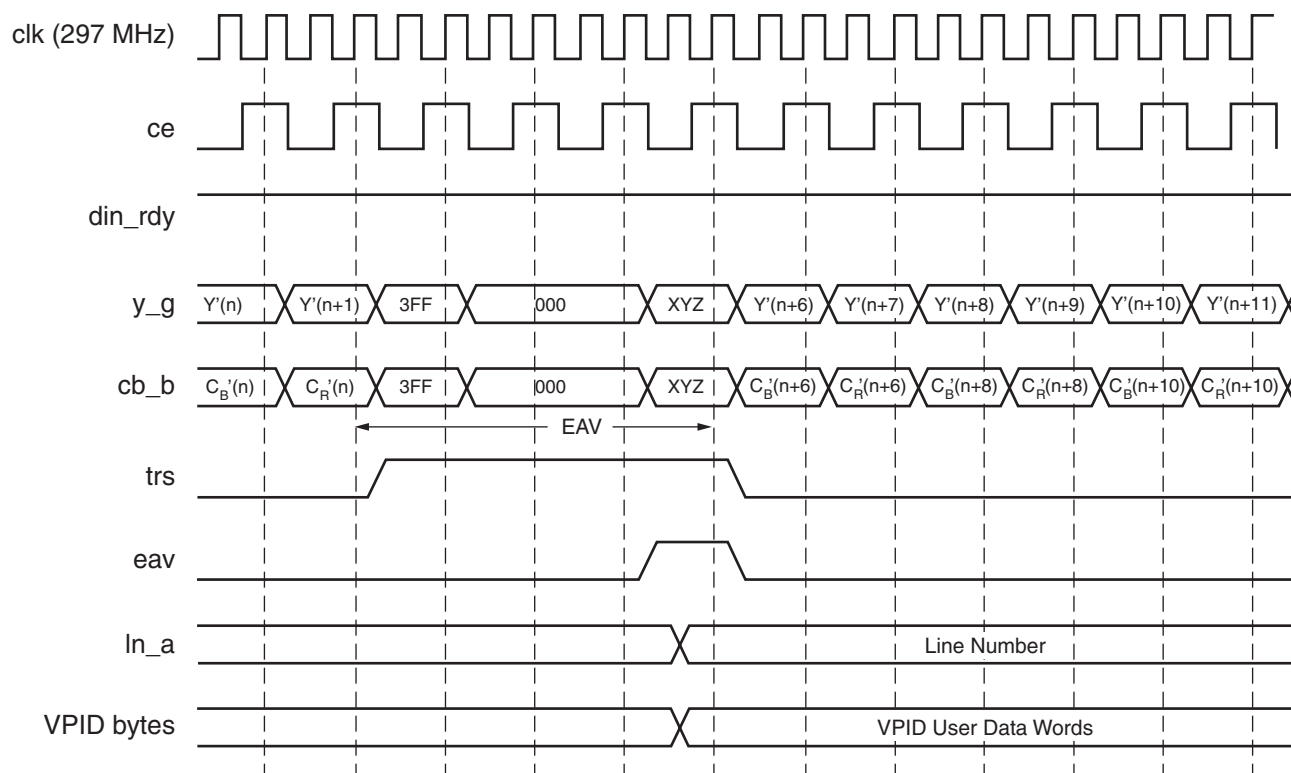
Figure 6-14: Dual Link HD-SDI Input Timing for Full-Featured Triple-Rate SDI Transmitter (4:4:4)

Full-Featured Triple-Rate Transmitter 3G-SDI Level A Operation

Several different input video formats are supported in 3G-SDI level A mode, each with different timing requirements. Figure 6-15 shows the input video timing of 4:2:2 10-bit 1080p 50 Hz, 59.94 Hz, and 60 Hz video. The Y component is provided on the most significant 10 bits of the y_g port and the interleaved C_B/C_R components are provided on the most significant 10 bits of the cb_b port. In this mode, valid EAV and SAV sequences must be present in both input component streams, as shown in Figure 6-15. The CRC and

LN words are generated and inserted by the output module and are not required in the input streams.

The clock rate is 297 MHz and ce is asserted every other clock cycle to provide the 148.5 MHz data rate. The din_rdy input must be High.



Notes:

The clk and ce inputs are connected to the corresponding ports of both the triple_sdi_tx_formatter and triple_sdi_tx_output modules. All inputs except ce have a 148.5 MHz sample time in which to become stable. Data is only sampled on the inputs on the rising edge of clk when ce is High.

In 3G-SDI mode, the line number value on the ln input port must be stable starting with the XYZ word of the EAV and must remain valid through the entire HANC interval.

The VPID input bytes on the input of the triple_sdi_vpid_insert module must be stable beginning with the XYZ word of the EAV and must remain stable during the entire HANC interval on lines where SMPTE 352M packets are inserted.

The timing of the EAV sequence is shown. The timing of the SAV sequence is the same, except that the sav input must be asserted during the XYZ word of each SAV instead of the eav input.

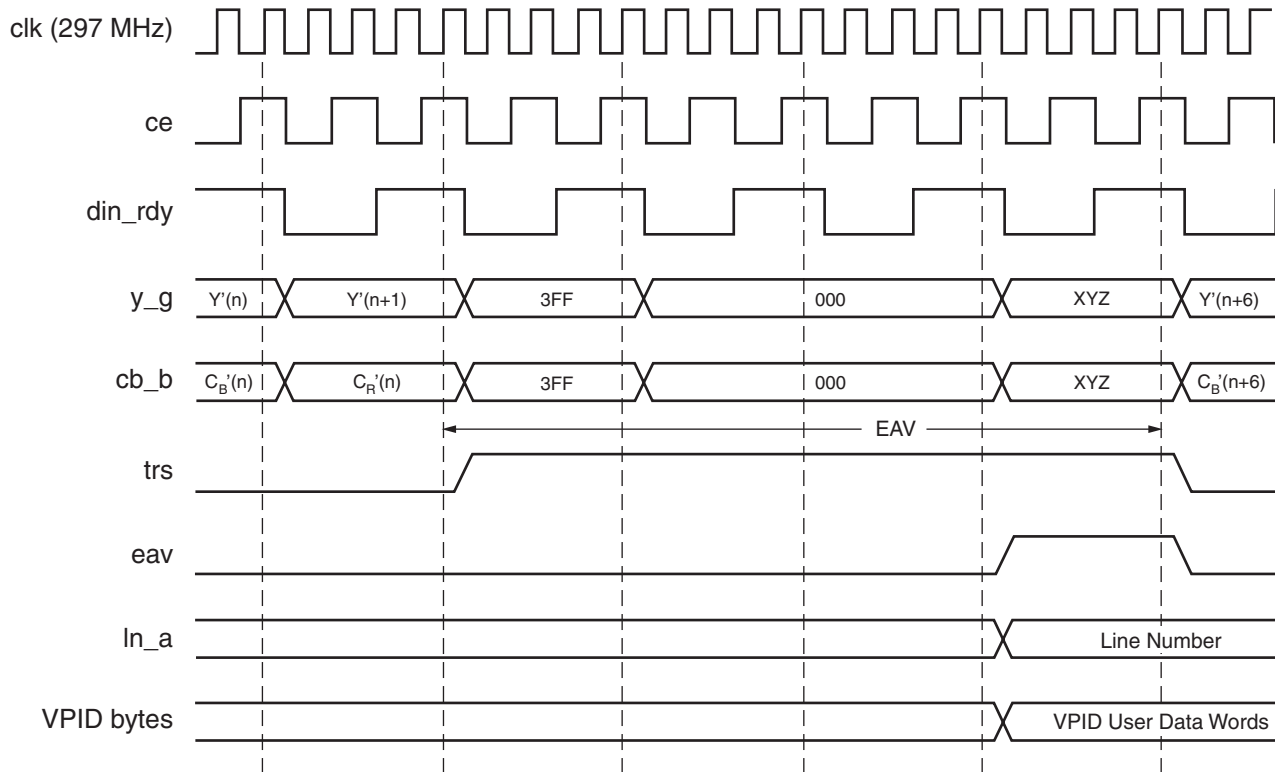
X1014_CS_15_041309

Figure 6-15: 3G-SDI Input Timing for Full-Featured Triple-Rate SDI Transmitter (148.5 MHz Sample Rate)

Figure 6-16 shows the input signal timing for 4:2:2 12-bit video. The Y component is provided on the y_g port and the interleaved C_B/C_R components are provided on the cb_b port. As specified in the SMPTE 425M standard, new EAV and SAV sequences are generated and inserted into the data streams by the triple_sdi_tx_formatter module as it maps the video into 3G-SDI data streams. The XYZ words of these new EAV and SAV sequences are identical to the XYZ word of the EAV and SAV sequences of the input Y component on the y_g port. Thus, the Y component data stream entering the

`triple_sdi_tx_formatter` module must have EAV and SAV sequences with valid XYZ words. The EAV and SAV sequences on the `cb_b` port are ignored.

The input video sample rate in this mode is 74.25 MHz. The clock runs at 297 MHz, and `ce` must be asserted every other clock cycle (148.5 MHz). The `din_rdy` port must be asserted at a 74.25 MHz rate, indicating when video samples are ready on the input ports of the `triple_sdi_tx_formatter` module. The input data words are sampled on the rising edge of `clk` when both `ce` and `din_rdy` are High.



Notes:

The `clk` and `ce` inputs are connected to the corresponding ports of both the `triple_sdi_tx_formatter` and `triple_sdi_tx_output` modules. All inputs except `ce` have a 74.25 MHz sample time in which to become stable. Data is only sampled on the inputs on the rising edge of `clk` when both `ce` and `din_rdy` are High.

In 3G-SDI mode, the line number value on the `ln_a` input port must be stable starting with the XYZ word of the EAV and must remain valid through the entire HANC interval.

The VPID input bytes on the input of the `triple_sdi_tx_formatter` module must be stable beginning with the XYZ word of the EAV and must remain stable during the entire HANC interval on lines where SMPTE 352M packets are inserted.

The timing of an EAV sequence is shown in the diagram. The timing for an SAV sequence is the same except that the `sav` input must be asserted High instead of the `eav` input during the XYZ word of the SAV sequence.

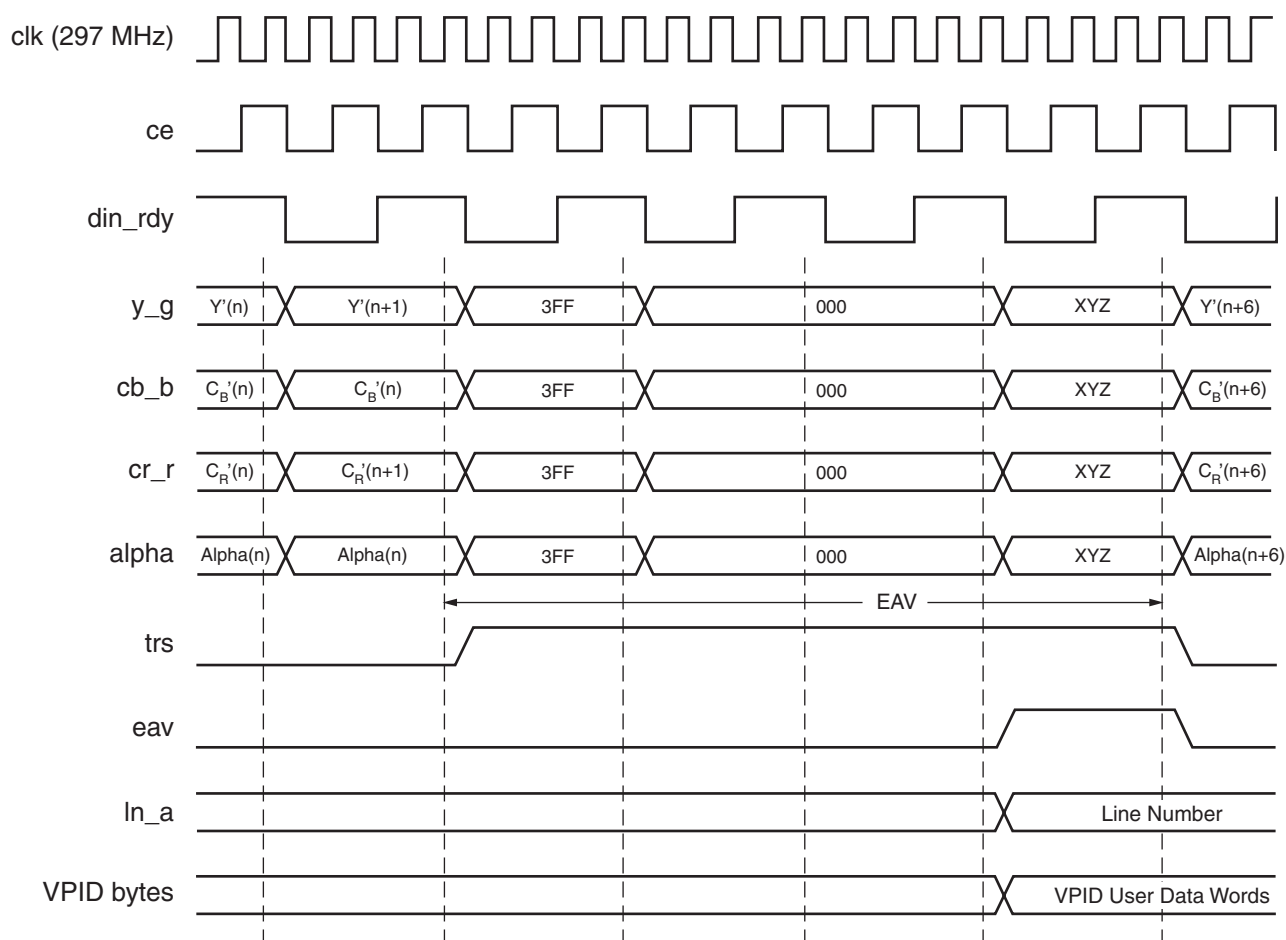
X1014_C5_16_041309

Figure 6-16: 3G-SDI Input Timing for Full-Featured Triple-Rate SDI Transmitter (4:2:2 12-Bit)

Figure 6-17 shows the input signal timing for the 4:4:4 10-bit or 12-bit video formats. Three input components are supplied to the `y_g`, `cb_b`, and `cr_r` input ports. In 10-bit mode, an alpha component is always present in the output video stream. The alpha samples are filled with 0x040 if `alpha_act` is Low or with the values entering the alpha input port if `alpha_act` is High. EAV and SAV sequences, including CRC and LN words, are generated and inserted into the output 3G-SDI data streams by the `triple_sdi_tx_formatter`

module. The XYZ words of the EAV and SAV sequences match the XYZ word of the EAV and SAV sequences of the Y' or G' component entering the module on the y_g port. Thus, the input Y' or G' component data stream must have valid EAV and SAV sequences. The EAV and SAV sequences of the other input components are ignored.

The video sample rate in this mode is 74.25 MHz. The clock runs at 297 MHz and ce must be asserted every other clock cycle (148.5 MHz). The din_rdy port must be asserted at a 74.25 MHz rate, indicating when video samples are ready on the input ports of the triple_sdi_tx_formatter module. The input data words are sampled on the rising edge of clk when both ce and din_rdy are High.



Notes:

The clk and ce inputs are connected to the corresponding ports of both the triple_sdi_tx_formatter and triple_sdi_tx_output modules. All inputs except ce have a 74.25 MHz sample time in which to become stable. Data is only sampled on the inputs on the rising edge of clk when both ce and din_rdy are High.

In 3G-SDI mode, the line number value on the ln_a input port must be stable starting with the XYZ word of the EAV and must remain valid through the entire HANC interval.

The VPID input bytes on the input of the triple_sdi_tx_formatter module must be stable beginning with the XYZ word of the EAV and must remain stable during the entire HANC interval on lines where SMPTE 352M packets are inserted.

The timing of an EAV sequence is shown in the diagram. The timing for an SAV sequence is the same except that the sav input must be asserted High instead of the eav input during the XYZ word of the SAV sequence.

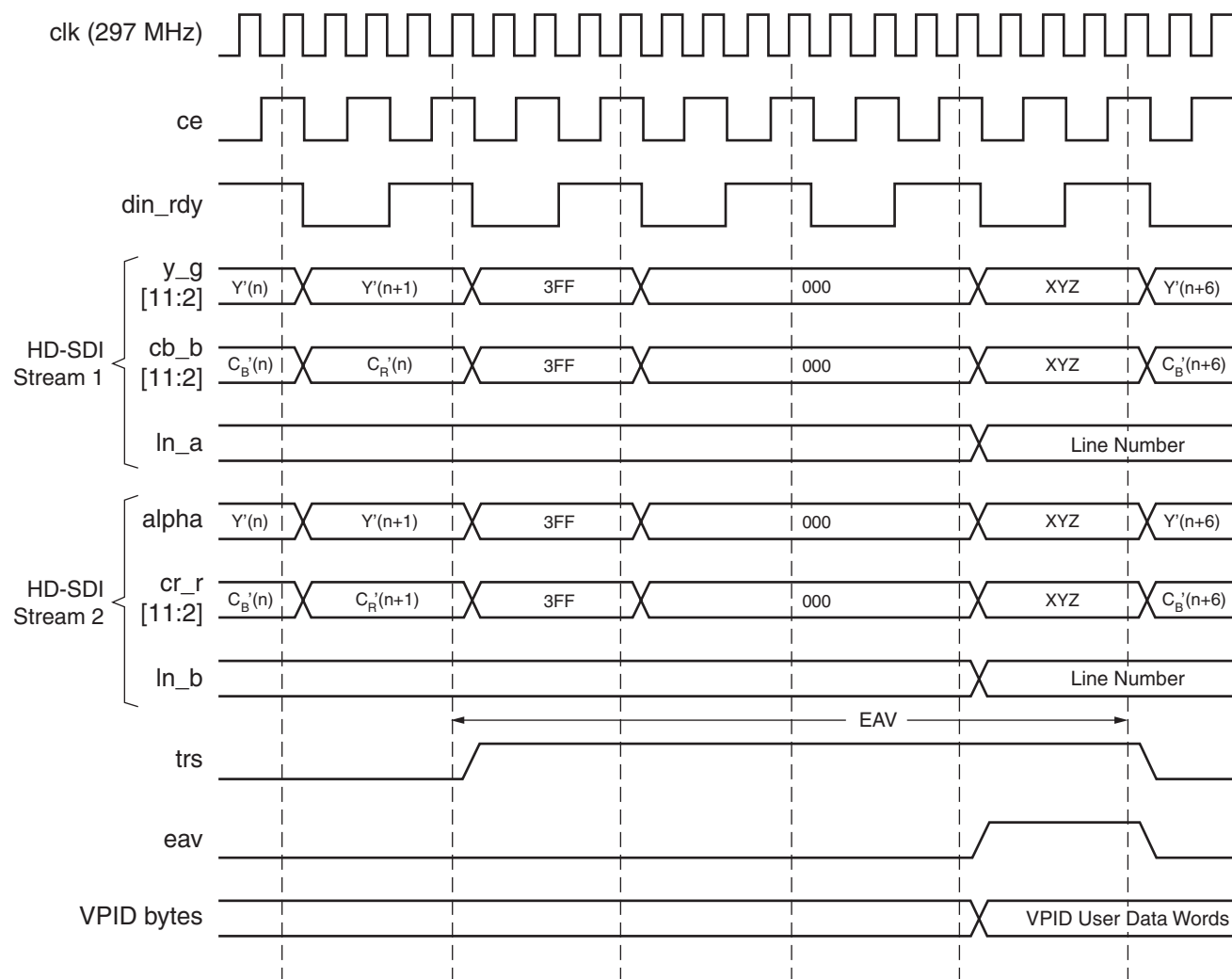
X1014_C5_17_041309

Figure 6-17: 3G-SDI Input Timing for Full-Featured Triple-Rate SDI Transmitter (4:4:4)

Full-Featured Triple-Rate Transmitter 3G-SDI Level B Operation

In 3G-SDI level B mode, the same video formats described for level A are all supported. The timing diagrams in [Figure 6-15](#), [Figure 6-16](#), and [Figure 6-17](#) also apply to 3G-SDI level B. However, the 4:2:2 12-bit mode shown in [Figure 6-16](#) might also contain a 10-bit alpha component in 3G-SDI level B mode. This is because level B mode carries SMPTE 372M dual link HD-SDI data streams. SMPTE 372M permits 4:2:2 12-bit video to have a 10-bit alpha component, whereas 4:2:2 12-bit video is not permitted to have an alpha component in 3G-SDI level A. Thus, when running in 3G-SDI level B mode and providing 4:2:2 12-bit video to the `triple_sdi_tx_formatter` module, if `alpha_act` is High, valid alpha component values must be provided on the alpha port. If `alpha_act` is Low, the `triple_sdi_tx_formatter` module fills the alpha samples in the output data streams with a default value of 0x040 (and valid EAV and SAV sequences).

In 3G-SDI level B mode, it is possible to transport two independent, but synchronous, HD-SDI streams. This mode is shown in [Figure 6-18](#) and is selected by setting the `triple_sdi_tx_formatter` module's `in_format` port to 111 or 101.



Notes:

The clk and ce inputs are connected to the corresponding ports of both the triple_sdi_tx_formatter and triple_sdi_tx_output modules. All inputs except ce have a 74.25 MHz sample time in which to become stable. Data is only sampled on the inputs on the rising edge of clk when both ce and din_rdy are High.

The line number values on the In_a and In_b input ports must be stable starting with the XYZ word of the EAV and must remain valid through the entire HANC interval.

The VPID input bytes on the input of the triple_sdi_tx_formatter module must be stable beginning with the XYZ word of the EAV and must remain stable during the entire HANC interval on lines where SMPTE 352M packets are inserted.

The timing of an EAV sequence is shown in the diagram. The timing for an SAV sequence is the same except that the sav input must be asserted High instead of the eav input during the XYZ word of the SAV sequence.

X1014_C5_18_041309

Figure 6-18: 3G-SDI Level B Timing for Full-Featured Triple-Rate SDI Transmitter with 2X HD-SDI

Full-Featured Triple-Rate Transmitter Details

This section describes various details of the operation of the full-featured triple-rate SDI TX reference design.

SMPTE 352M Packet Insertion

The `triple_sdi_tx_formatter` module can insert SMPTE 352M VPID packets into SD-SDI, HD-SDI, dual link HD-SDI, and 3G-SDI (both level A and level B) streams. The 3G-SDI and dual link HD-SDI standards require SMPTE 352M packets to be inserted into the data streams. The packets are optional in SD-SDI and HD-SDI.

The `triple_sdi_tx_formatter` module uses a pair of `SMPTE352_vpid_insert` modules from the SMPTE 352M reference design to insert the VPID packets. The behavior of this module is the same as the behavior of the underlying `SMPTE352_vpid_insert` module. The behavior of this module should be reviewed, as described in [Chapter 26, SMPTE 352M Video Payload Identification Packet Processing](#). [Chapter 26](#) also describes the required values for the user data words of the SMPTE 352M packets.

SMPTE 352M packets are inserted on one designated line in each frame for progressive video and one designated line in each field for interlaced video. The lines designated to carry SMPTE 352M packets vary depending on the SDI mode and the video format and are described in [Chapter 26](#). The `line_f1` input port determines the line in the progressive frame or in the first interlaced field where the module inserts the SMPTE 352M packet. The `line_f2` input port determines the line in the second interlaced field where the module inserts the SMPTE 352M packet. The `line_f2_en` input determines whether or not packets are inserted in the line specified by `line_f2`. This input must be Low for a progressive transport and High for an interlaced transport.

Note: It is very important that the `line_f2_en` input to the `triple_sdi_vpid_insert` module be controlled correctly. This input must be High for interlaced video and Low for progressive video. However, this input must actually be controlled based on whether the transport is interlaced or progressive, not the source picture. The 1080p 50 Hz and 60 Hz 4:2:2 10-bit video formats, when carried by dual link HD-SDI or 3G-SDI level B, are transported in an interlaced manner, even though the picture format is progressive. The same is true for the progressive segmented frame formats. It is essential that the SMPTE 352M packets be inserted into both fields of the transport data streams for these formats. Some receiving equipment fails to lock properly if the SMPTE 352M packets are only present in one field rather than in both fields.

Dual link HD-SDI data streams coming from a dual link SDI receiver might already have SMPTE 352M packets in the Y data streams of each link because dual link SDI requires these packets. However, not all equipment on the market properly inserts these required packets. If the packets are present, the first data byte is 0x87, indicating that the data streams are carried on a dual link HD-SDI interface. When sending this dual link data on a 3G-SDI level B interface, the first byte must be replaced with a value of 0x8A, indicating SMPTE 372M streams carried on a 3G-SDI level B interface. Thus, the first byte of the VPID data must be modified. The values of the other three user data words do not need to change, but must be captured first and applied to the VPID byte inputs of the `triple_sdi_tx_formatter` module so that they are reinserted when the packet is overwritten.

In 3G-SDI level B mode, the two independent HD-SDI signals carried by level B are allowed to be vertically unsynchronized. If this is the case, SMPTE 352M packets are inserted independently on the two HD-SDI signals carried by the 3G-SDI level B interface. The second line number input port on the `triple_sdi_vpid_insert` module allows two separate line numbers to be provided for level B, one for each HD-SDI signal. However, because there is only one set of VPID data inputs to the insertion module, the SMPTE 352M packets, with the exception of byte 4, are identical. This means that the two HD-SDI streams must be carrying identical video formats. This is normally the case due to the restrictions in 3G-SDI level B requiring the two HD-SDI streams to be of the same format. If an application requires insertion of different VPID packets into the two HD-SDI signals carried by a 3G-SDI level B signal, the `triple_sdi_tx_formatter` module

could easily be modified to provide two completely independent sets of VPID input data ports.

Line Number Insertion

3G-SDI and HD-SDI both require that line numbers be present in the two words that follow each EAV. The `triple_sdi_tx_output` module inserts those line numbers appropriately for HD-SDI and both levels of 3G-SDI (inserting them into all four data streams for level B 3G-SDI) if the `insert_ln` input is High. The line numbers to be inserted are provided to the `triple_sdi_tx_formatter` module on the `ln_a` and `ln_b` inputs. In some cases, it might not be necessary or desirable to overwrite line numbers that are already present in the data streams. In that case, the `insert_ln` input can be driven Low and no line numbers are inserted. If the `insert_ln` input is hard-wired Low in the source code, the line number insertion logic is optimized out of the design by the synthesis tool.

For HD-SDI, dual link HD-SDI, and 3G-SDI level A, only the line number on `ln_a` is used. For 3G-SDI level B, the line number on `ln_a` is inserted into the Y and C data streams of link A, and the line number on `ln_b` is inserted into the Y and C data streams of link B (again, only if `insert_ln` is asserted).

The SMPTE 425M specification does not specifically state that the two HD-SDI signals carried in level B mode have to be vertically synchronized (except that, when SMPTE 372M dual link HD-SDI is being carried, they must be vertically synchronized). The transmitter reference design supports vertically unsynchronized HD-SDI signals by providing the two separate line number input ports.

CRC Generation and Insertion

3G-SDI and HD-SDI both require that CRC values be present in the two words that follow the line numbers after the EAV. The `triple_sdi_tx_output` module calculates and inserts CRC values for each line for HD-SDI and both levels of 3G-SDI (inserting them into all four data streams for level B 3G-SDI and dual link HD-SDI) if the `insert_crc` input is High. In some cases, it might not be necessary or desirable to overwrite the CRC values already present in the data stream. In that case, the `insert_crc` input can be driven Low and no CRC values are inserted. If the `insert_crc` input is hardwired Low in the source code, the CRC generation and insertion logic is optimized out of the design by the synthesis tool.

EDH Generation and Insertion

EDH packets are an option in SD-SDI and are never used for HD-SDI and 3G-SDI. These packets provide CRC values that can be used to detect errors in the SD-SDI data stream. When running in SD-SDI mode, the `triple_sdi_tx_output` module generates and inserts EDH packets when `insert_edh` is High. If EDH is hard-wired Low, the synthesis tool optimizes the EDH generator out of the design. The EDH generator used in this reference design is the version that does not use the PicoBlaze processor, as described in the “SD-SDI Ancillary Data and EDH Processors” chapter of *Audio/Video Connectivity Solutions for Virtex-II Pro and Virtex-4 FPGAs* [Ref 5].

FPGA Resource Usage

Table 6-8 shows the FPGA resources required for several different implementations of the triple-rate SDI TX interface reference design described in this chapter. The resource usage includes all the modules required to implement the interface, including the `v5gtp_sdi_control` module, and is based on using one TX unit in a GTP_DUAL tile.

The results shown in Table 6-8 were achieved using ISE® Design Suite 10.1 SP2, and XST and MAP set to optimize for speed. The XST Safe Implementation property was set to **Yes**.

Table 6-8: Triple-Rate SDI Transmitter FPGA Resource Usage

Reference Design	Flip-Flops	LUTs	Block RAMs
Light TX without EDH	621	754	None
Light TX with EDH	993	1293	None
Full-featured TX without EDH	948	1151	3 x RAMB36 3 x RAMB18
Full-featured TX with EDH	1316	1739	3 x RAMB36 3 x RAMB18

The full-featured TX uses three 36K block RAMs and three 18K block RAMs for line buffers to support 1080p 50 Hz and 60 Hz when transmitting this video format with level B 3G-SDI and dual link HD-SDI. If this mode is not required, these block RAMs can be eliminated.

Timing

Because the basic clock frequency used for 3G-SDI and for SD-SDI is 297 MHz, some paths of the design need to run at this frequency. However, the modules of the triple-rate SDI reference design have been designed so that only a few paths actually run at this frequency. The bulk of the paths run at one half this frequency or less, making it easier to achieve timing. This is done by using clock enables. Those paths that do not run at the full data rate are given a multi-cycle timing constraint so that the timing analyzer knows that the paths do not have to run at 297 MHz. However, the clock enables themselves must meet the setup and hold times to all flip-flops that they enable at the full 297 MHz clock frequency. The multi-cycle constraints are shown in the UCF files of the demonstration projects included with the reference design.

To make it possible to place timing constraints on certain signals internal to the reference design, these signals have had XST KEEP constraints placed on them in the Verilog and VHDL code. This is an XST-specific constraint and has to be manually converted to the equivalent constraint if Synplify is used. Similarly, some registers in the design have the XST-specific “equivalent register removal” constraint applied. These constraints also have to be converted to the Synplify equivalent.

All of the modules in the triple-rate SDI have been tested with the slowest speed grade Virtex-5 devices, thus allowing timing to be met in these slowest speed grade parts.

It would be possible to use a basic frequency of 148.5 MHz if a 20-bit TXDATA port was used on the GTP transceiver. However, the triple-rate SDI TX reference design for the Virtex-5 GTP transceiver was specifically designed to work with 10-bit GTP TXDATA ports. This was done to reduce global clocking resource requirements. Using 20-bit TXDATA paths requires the use of a DCM or PLL to produce two phase-aligned clock frequencies and requires two global clock trees. These additional clock resources are required, in general, per triple-rate SDI transmitter and could quickly exhaust the available clocking resources in the FPGA if many SDI interfaces are implemented in a single FPGA. By using 10-bit TXDATA ports and dealing with the 297 MHz clock, the global clocking resources per triple-rate SDI TX have been reduced to one global clock tree with no DCMs or PLLs required.

The TXOUTCLK generated by the GTP can, under some conditions, have erratic timing. When a GTP TX is switched between HD-SDI mode and either SD-SDI or 3G-SDI modes, the frequencies of the TXOUTCLKs change between 148.5 MHz and 297 MHz. The timing of these clocks can be erratic for a short period of time immediately following the mode change. These brief periods of erratic timing can cause problems for sequential control logic such as FSMs. Most synthesis tools, by default, optimize away illegal state recovery for FSMs. The erratic timing of the GTP clocks can cause FSMs to go into illegal states. Thus, it is highly recommended that the synthesis tool be forced to generate “safe” implementations of FSMs that include illegal state recovery.

The clock enable signals are particularly critical timing paths because they must meet the setup and hold times of the various synchronous elements at the full 297 MHz clock speed. The reference design uses multiple identical copies of clock enables to reduce loading on the clock enables. This is better than attempting to use a global clock tree to distribute the clock enables because the global clock tree has a large delay almost equal to or exceeding the clock period of a 297 MHz clock, depending on the size of the device. Care must be taken to properly constrain the clock enables as shown in the example UCF files in the reference design.

The clock enables and write enables of the block RAMs used to implement the line buffers in the SMPTE372_tx_1080p_mem_RAMB36 modules (used only in the full-featured RX and TX designs) are particularly critical and require special timing constraints. These signals are the AND of address bits with the main clock enable. The address bits are generated by counters in the multi-cycle datapath, but the address bits used for the clock enable and write enables must be fast enough to meet the full 297 MHz timing when combined with the clock enable signal to drive the block RAMs. FROM-THRU-TO constraints are used to properly constrain these particular signals, as shown in the example UCF files in the reference design.

XST sometimes implements a clock enable using a LUT driving the D input of a flip-flop, rather than using the flip-flop’s clock enable input. This makes it more difficult to meet timing because it adds the delay of the LUT into the critical clock enable timing path. XST has a constraint called Use Clock Enable. Setting this constraint to **Yes** forces XST to use clock enables rather than implementing the clock enable function through logic driving the D input of the flip-flop. It is recommended that, for these designs that heavily use clock enables with a high-speed clock, this constraint be set to **Yes** globally using the XST properties in Project Navigator, or by a command line flag. It is possible to set this constraint locally on certain modules.

Reference Design

The reference design for the triple-rate SDI transmitter is available at <https://secure.xilinx.com/webreg/clickthrough.do?cid=119758>. Open the ZIP archive and extract the file xapp1014_c6_GTP_SDI_TX.zip.

Conclusion

The reference designs presented in this chapter provide two distinct methods of implementing triple-rate SDI transmitters in the Virtex-5 LXT and SXT devices. The full-featured transmitter provides a complete triple-rate SDI transmitter solution, including the logic to map all supported native video formats into 3G-SDI and dual link HD-SDI data streams. On the other hand, some applications do not need the mapping functions, either because the video formats being handled do not need to be mapped or because the transmitter always takes in preformatted data streams. Such applications can

take advantage of the simpler light version of the triple-rate SDI transmitter reference design.

In either method, the triple-rate SDI transmitter reference designs fully support SD-SDI, HD-SDI, dual link HD-SDI, and 3G-SDI level A and level B with dynamic switching between any of these SDI standards and requiring a minimum number of reference clocks and global clocking resources.

Triple-Rate SDI Pass-through Design for Virtex-5 LXT and SXT Devices

Summary

The RocketIO™ GTP transceivers available in Virtex®-5 LXT and SXT devices can be used to implement SDI receivers and transmitters that support all three SMPTE serial digital video interface standards: SD-SDI, HD-SDI, and 3G-SDI. This chapter describes an example triple-rate SDI pass-through design using the “light” versions of the triple-rate SDI receiver and transmitter reference designs. This pass-through design supports SD-SDI, HD-SDI, and 3G-SDI (both level A and level B) with the data received by the triple-rate SDI receiver retransmitted by the triple-rate SDI transmitter. This example design also shows how low-jitter reference clocks for the SDI transmitter are generated from the recovered clock or video sync signals.

This triple-rate SDI reference design has been designed specifically for the GTP transceiver available in Virtex-5 LXT and SXT devices. It cannot be used directly with other Xilinx® FPGAs.

Introduction

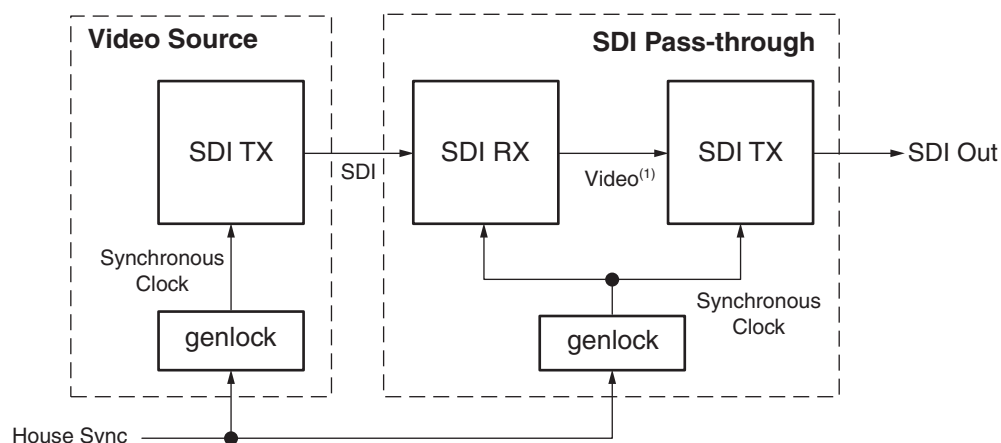
Many items of video equipment receive video and audio data streams through SDI receivers, process the data streams in some fashion, and then output the data streams through SDI transmitters. The example design described in this chapter receives and retransmits data streams using triple-rate SDI interfaces built with the GTP transceivers in Virtex-5 LXT and SXT devices. The example design does not process the audio and video data streams in any way. Instead, it simply retransmits the data streams. However, it does illustrate the techniques used to build more complex applications.

The most important considerations when retransmitting data received by an SDI interface are how to provide a low-jitter transmitter clock and how to synchronize the received data with this low-jitter transmitter clock. Three common techniques for doing this are synchronous systems, frame sync, and asynchronous systems. These are described in the following sections.

Synchronous Systems

Some video equipment is designed to work in a synchronous environment. This type of equipment receives a video sync signal, commonly called “house sync,” and derives local clocks from the sync signal using genlock techniques. All video equipment in the studio uses house sync to derive local clocks. Therefore, video moving between pieces of equipment over SDI interfaces is synchronous with the local clocks found in each piece of equipment.

Figure 7-1 is a simple block diagram of a synchronous system. Two pieces of equipment, the video source and the SDI pass-through, are both genlocked to house sync. The SDI output of the video source is synchronous with house sync. Therefore, the video output by the SDI RX in the SDI pass-through equipment is also synchronous with house sync. Furthermore, the local clock produced by the genlock circuit in the pass-through equipment is used to clock the SDI transmitter. The video can move synchronously from the SDI RX to the SDI TX in the pass-through equipment.



Note: 1) The video from the receiver is synchronous with the clock from the genlock circuit

X1014_C6_01_033009

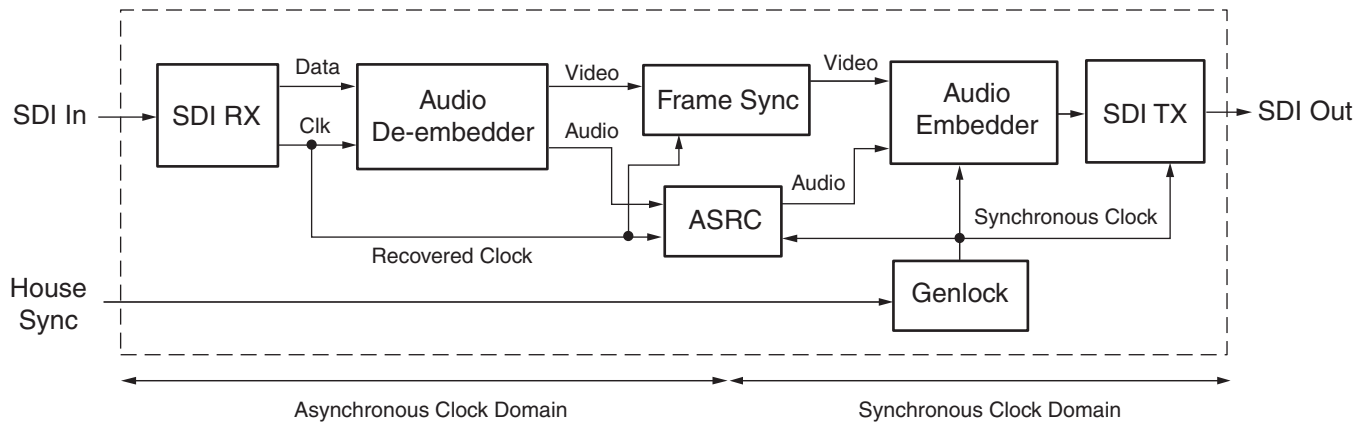
Figure 7-1: Synchronous System

Between house sync pulses, the local clocks of the video source and the pass-through equipment drift. Therefore, a small elastic buffer or FIFO is needed in the SDI RX to compensate for the drift. The GTP RX in the Virtex-5 FPGA has such an elastic buffer built in.

Frame Sync

To synchronize video and audio received from the SDI RX to a local clock (usually derived from the house sync signal), some video equipment contains a frame sync and an audio asynchronous sample rate converter (ASRC). The video input to the frame sync and the audio input to the ASRC are asynchronous to house sync. The video output from the frame sync and the audio output from the ASRC are synchronous with house sync. The local clock derived from house sync clocks the outputs of the frame sync, ASRC, and the SDI TX.

Figure 7-2 is a simple block diagram of a frame sync system. The SDI input is asynchronous to house sync. The SDI RX recovers a clock that is also asynchronous to house sync. An audio de-embedder takes the data stream from the SDI RX and separates the audio from the video. The datapaths from the SDI RX to the inputs of the frame sync and ASRC are clocked by the recovered clock from the SDI RX. The synchronous clock from the genlock circuit clocks data out of the frame sync and the ASRC, through the audio embedder where the audio is re-embedded into the video stream, and into the SDI TX.



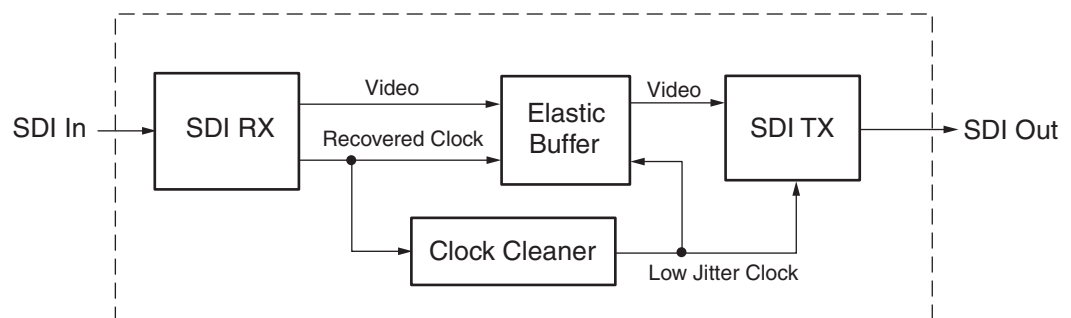
X1014_C6_02_041309

Figure 7-2: Frame Sync System

Asynchronous Systems

It is also possible for video equipment to operate asynchronously to any external reference. Such equipment does not depend on house sync or generate local clocks derived from a house sync input. Instead, it uses the recovered clock from the SDI receiver to clock the SDI transmitter. Typically, jitter reduction techniques are required to reduce the jitter on the recovered clock before it can be used as a reference clock to drive the SDI transmitters. The reference design described in this chapter works in this way.

Figure 7-3 is a simple block diagram of an asynchronous system. The recovered clock from the SDI RX is the reference clock used by the entire system. However, it usually has too much jitter to be used directly. Using the recovered clock as the reference clock to the SDI TX serializer would result in too much transmitter output jitter. To reduce jitter to an acceptable level, the recovered clock is passed through a clock cleaner PLL. The low-jitter clock output from the clock cleaner is then used as the reference clock to the serializer. While the recovered clock and the low-jitter clock are frequency locked, there will be phase differences between the clocks. A shallow elastic buffer or asynchronous FIFO is used to move the video from the recovered clock domain to the low-jitter clock domain. The GTP transmitters in Virtex-5 devices contain a shallow elastic buffer suitable for this purpose.



X1014_C6_03_033009

Figure 7-3: Asynchronous System

Design Description

The pass-through example design presented in this chapter is an asynchronous system. The recovered clock from the SDI receiver is sent through a jitter cleanup circuit and then used as the reference clock for the SDI transmitter. On the Xilinx ML571 serial digital video (SDV) board, a clock module is added that has a clock cleaner used to remove the jitter from the recovered clock of the SDI receiver. The output of the clock cleaner is used as a reference clock to two SDI transmitters. The two SDI transmitters are identical—both retransmit the data received by the single SDI receiver.

After jitter is removed, the recovered clock from the GTP RX can be used as a reference clock for the GTP TX in HD-SDI and 3G-SDI modes. However, in SD-SDI mode, the GTP RX does not recover a clock. Instead, the 297 MHz RXRECCLK from the GTP RX is derived from a local reference clock and is asynchronous to the actual recovered video data rate. A clock enable signal is generated by the triple-rate SDI receiver that is synchronous with RXRECCLK. This clock enable throttles the SDI datapath to the actual 27 MHz data rate of the recovered video. This presents a problem for retransmitting SD-SDI because the SDI transmitter requires a 148.5 MHz or 74.25 MHz reference clock that is an exact 5.5X or 2.75X multiple of the 27 MHz video data rate.

The clock enable from the SDI RX has a frequency of 27 MHz and runs at exactly the input video data rate. However, it has low-frequency jitter that is very difficult to remove using PLL-based clock cleaners.

In the example design, a genlock circuit takes the video sync signals recovered by the SD-SDI receiver and produces a 148.5 MHz clock, which is an exact 5.5X multiple of the 27 MHz video data rate. This clock, generated by the genlock circuit, passes through a clock cleaner for jitter reduction. The resulting 148.5 MHz clock is used as the reference clock for the SDI transmitter in SD-SDI mode.

Figure 7-4 is a block diagram of the triple-rate SDI pass-through design. It has one triple-rate SDI receiver, implemented with a GTP RX and a `triple_sdi_rx_light` module. The `triple_sdi_rx_light` module does not contain an EDH processor. In this example design, an EDH processor is connected to the output of the `triple_sdi_rx_light` module to check the SD-SDI data streams for errors, reported via an LED, and to generate the F, V, and H sync signals that are fed to the genlock circuit.

The received data streams pass through a `triple_sdi_tx_output` module and then to two GTP TX modules to drive the two SDI TX outputs. It is not necessary to drive two transmitters as in this example design. This was done only for demonstration purposes.



the data and clock frequencies and paths change depending on the SDI mode. The following four sections explain how these paths are configured for each SDI mode.

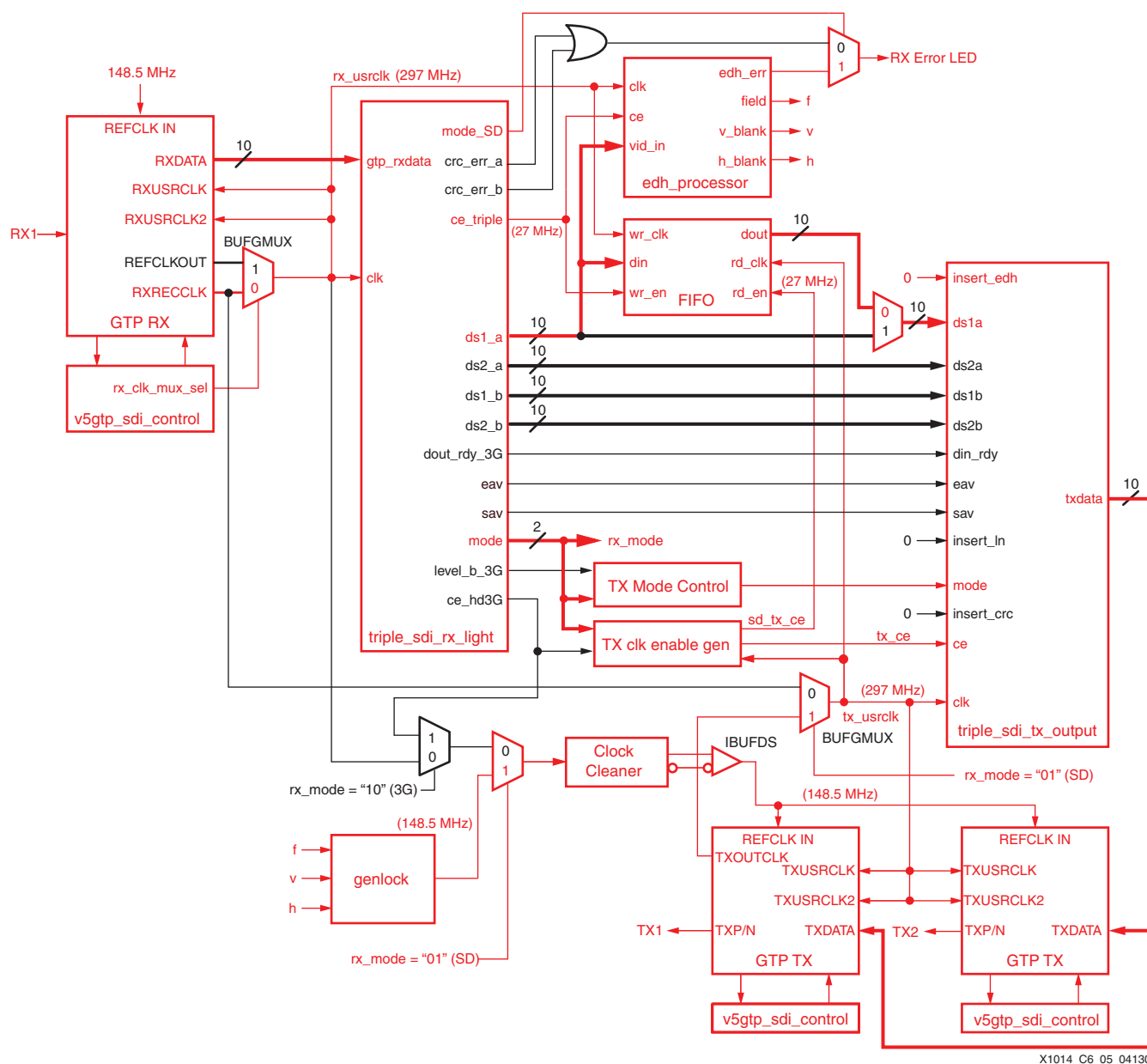
The design uses two global clocks called rx_usrclk and tx_usrclk. In 3G-SDI and HD-SDI modes, both of these clocks are driven by the RXRECCLK from the GTP RX and are identical. In SD-SDI mode, rx_usrclk is driven by RXRECCLK, an asynchronous 297 MHz clock, while tx_usrclk is driven by TXOUTCLK from the GTP TX and is synchronous to the video rate.

SD-SDI Mode

Figure 7-5 is the same block diagram as Figure 7-4, but with the clock and datapaths used in SD-SDI mode colored red. A 148.5 MHz reference clock is provided to the GTP_DUAL tile used to implement the SDI RX. The reference clock can also be 74.25 MHz (see [Other Considerations, page 242](#) for more information on this configuration). In SD-SDI mode, the CDR circuit is locked to this reference clock, and the GTP RX asynchronously oversamples the input bit stream by 11X. The 297 MHz clock output by the GTP RX (RXRECCLK) is not really a recovered clock because it is asynchronous to the SD video data rate. However, the oversampled data stream output on RXDATA is synchronous with RXRECCLK. Therefore, the RX datapath is clocked by rx_usrclk, which is the global version of RXRECCLK.

The `triple_sdi_rx_light` receiver module contains a DRU to recover the SD video from the 11X oversampled data stream. The module also generates a 27 MHz clock enable on the `ce_triple` output. This clock enable is asserted High one clock cycle out of every 11 cycles, on average, of the 297 MHz `rx_usrclk`, providing a 27 MHz data rate. However, the clock enable is erratic and not consistently asserted every 11 clock cycles.

The recovered data stream is output from the `triple_sdi_rx_light` module on the `dsl_a` port. This is a 10-bit wide data stream with a 27 MHz data rate. The data stream is connected to the input of the EDH processor. It is also written into an asynchronous FIFO. This FIFO is used (only in SD-SDI mode) to move the data from the RX clock domain to the TX clock domain. The FIFO does not need to be deep because the RX and TX clock domains are frequency locked. However, it should be sufficiently deep to allow for frequency drift of the genlock circuit between sync pulses.



X1014_C6_05_041309

Figure 7-5: SD-SDI Clock and Datapaths

Because the 27 MHz clock enable does not have a consistent 11-cycle period, it is difficult to use this as an actual recovered clock to drive the transmitter. Reducing the jitter of the clock enable is quite difficult to do. Furthermore, even if this clock enable had low jitter, it is the wrong frequency to be used as a reference clock to the GTP TX in SD-SDI mode. The GTP TX needs a reference clock with a frequency of 74.25 MHz or 148.5 MHz, not 27 MHz.

A genlock circuit is used to generate the required transmitter reference clock in SD-SDI mode. The F, V, and H sync signals from the EDH processor are driven out of the FPGA to the genlock circuit. The genlock circuit locks to these sync signals and produces a 148.5 MHz clock. This clock goes through the clock cleaner circuit to remove jitter. The output of the clock cleaner is a low-jitter, 148.5 MHz clock. It is connected to the reference clock inputs of the GTP transmitters. The 148.5 MHz reference clock produced by the

genlock and clock cleaner is frequency locked to the recovered video, not to the asynchronous RXRECCLK output by the GTP RX.

The clock multiplication PLL in the GTP TX multiplies the 148.5 MHz reference clock from the genlock and clock cleaner up to 2.97 GHz to drive the serializer. This 2.97 GHz serial clock is divided back down to 297 MHz and output on the GTP TXOUTCLK port. TXOUTCLK from one of the GTP transmitters is connected to a BUFGMUX that drives tx_usrclk. The 297 MHz tx_usrclk clocks the TX datapaths from the output of the FIFO, through the triple_sdi_tx_output module, to the TXDATA input ports of the GTP transmitters.

The data is read out of the asynchronous FIFO using tx_usrclk and a 27 MHz clock enable (tx_ce) generated by the TX clock enable generator. In SD-SDI mode, tx_ce is always asserted High one clock cycle out of every 11. Thus, the received data passes from the jittery clock domain—where the receiver clock enable is usually asserted once every 11 clock cycles (but not always)—to the TX clock domain, where the clock enable is always, without variation, asserted High once every 11 cycles. The data from the FIFO goes to the triple_sdi_tx_output module where it is scrambled and replicated 11 times, thereby creating a suitable data stream for the two GTP transmitters.

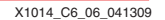
HD-SDI Mode

Figure 7-6 shows the clock and datapaths used in HD-SDI mode. The CDR unit in the GTP RX locks to the incoming HD-SDI signal and recovers both clock and data. The 148.5 MHz recovered clock is output on the GTP RXRECCLK port and, after passing through the BUFGMUX, drives rx_usrclk. The RXRECCLK also drives the tx_usrclk BUFGMUX. Therefore, rx_usrclk and tx_usrclk are identical copies of the 148.5 MHz recovered clock. The entire RX and TX datapath, up to and including the TXDATA ports of the GTP transmitters, is synchronous and is clocked by RXRECCLK.

The 148.5 MHz rx_usrclk is driven out the FPGA to the clock cleaner where jitter on this clock is reduced. The 148.5 MHz low-jitter clock from the clock cleaner is used as the reference clock to the GTP transmitters. This clock is frequency locked to RXRECCLK, rx_usrclk, and tx_usrclk. A shallow elastic buffer inside each GTP TX compensates for any phase difference between tx_usrclk (connected to the GTP TXUSRCLK inputs) and the reference clock.

Because the entire datapath is synchronous up to the GTP transmitter TXDATA ports, the two data streams from the ds1_a and ds2_a ports of the triple_sdi_rx_light receiver module can be connected directly to the ds1a and ds2a inputs of the triple_sdi_tx_output module. The data rate of these data streams is 74.25 MHz and is controlled by the ce_hd3G clock enable output of the triple_sdi_rx_light module. In HD-SDI mode, tx_ce also has a frequency of 74.25 MHz and is phase aligned with ce_hd3G. These clock enables are asserted every other cycle of rx_usrclk and tx_usrclk.

TXOUTCLK from the GTP TX is not used in this mode. Instead, the clock into the TXUSRCLKs of the GTP transmitters is derived from the RXRECCLK of the GTP receiver.



3G-SDI Level A Mode

The RXRECCLK from the GTP is 297 MHz in 3G-SDI mode. As with HD-SDI mode, RXRECCLK drives both rx_usrclk and tx_usrclk. The entire datapath up to the TXDATA ports of the GTP transmitters is synchronous with RXRECCLK. The actual data rate of the two data streams output by the triple_sdi_rx_light module is 148.5 MHz, and, as in HD-SDI mode, is regulated by the ce_hd3G output of the triple_sdi_rx_light

module. This clock enable and the tx_ce clock enable (identical in phase and frequency to ce_hd3G) are asserted every other clock cycle, thus controlling the 148.5 MHz data rate.

Besides the frequency of the clocks and clock enables, the primary difference between 3G-SDI level A mode and HD-SDI mode is the source of the input to the clock cleaner. The reference clock to the GTP transmitters must still be 148.5 MHz, as it was for HD-SDI mode. Because `rx_usrclk` is 297 MHz in 3G-SDI mode, it cannot be used directly as the source to the clock cleaner. Instead, the `ce_hd3G` output of the `triple_sdi_rx_light` module is connected to the clock cleaner input. This clock enable has the correct frequency of 148.5 MHz. It would also be simple to generate the 148.5 MHz clock to the clock cleaner by dividing `rx_usrclk` in half with a toggle flip-flop, but this is exactly how `ce_hd3G` is generated in the `triple_sdi_rx_light` module.

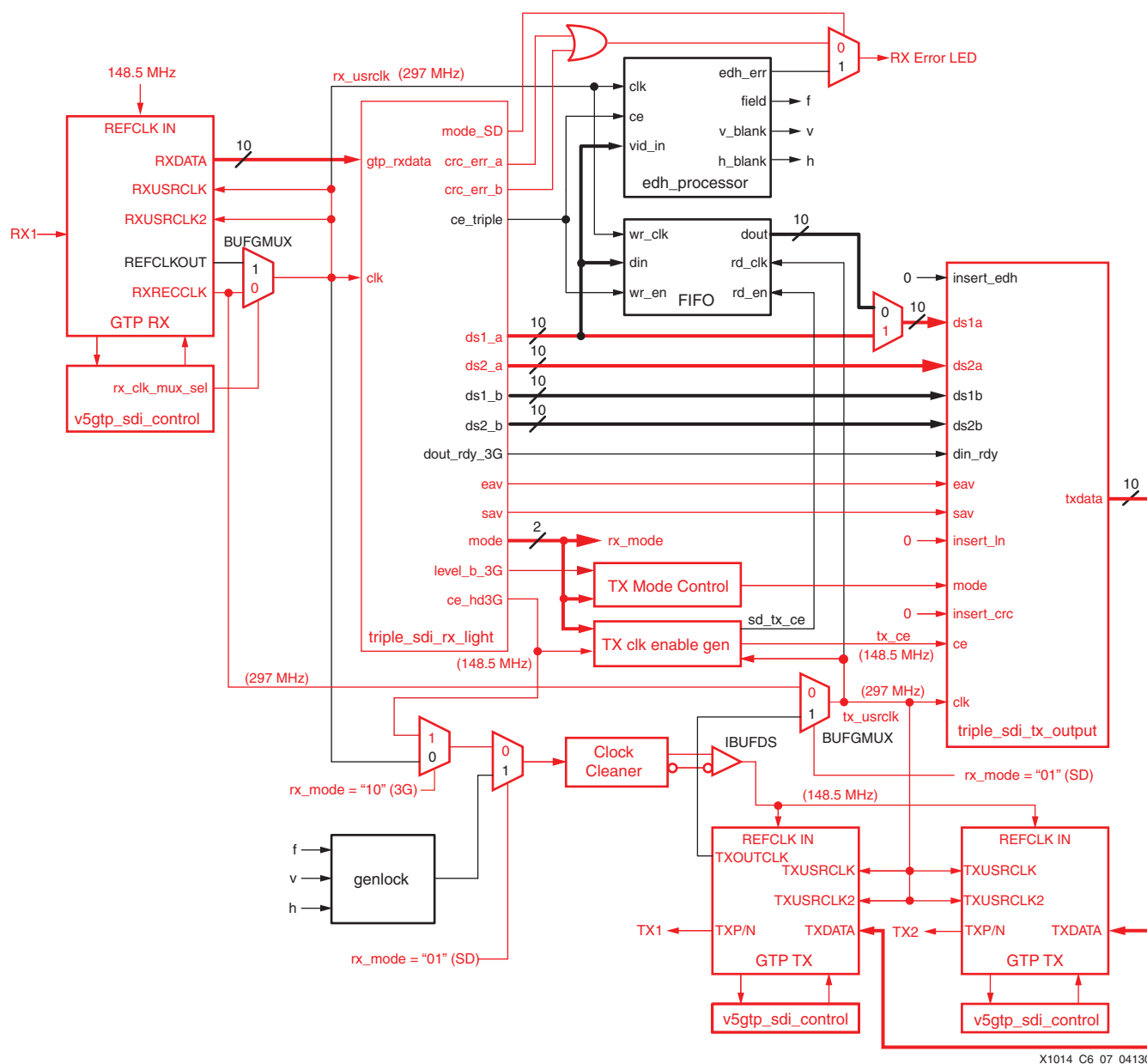


Figure 7-7: 3G-SDI Level A Clock and Datapaths

3G-SDI Level B Mode

Figure 7-8 shows the clock and datapaths when running in 3G-SDI level B mode. This mode is almost identical to 3G-SDI level A mode. However, in 3G-SDI level B mode, the `triple_sdi_rx_light` module produces four 10-bit data streams with a data rate of 74.25 MHz, rather than the two 148.5 MHz 10-bit data streams produced in 3G-SDI level A mode. All clocks and clock enables still run at the same frequencies as 3G-SDI level A mode. Even the `ce_hd3G` and `tx_ce` clock enables still have a frequency of 148.5 MHz. However, in 3G-SDI level B mode only, the `dout_rdy_3G` output of the `triple_sdi_rx_light` module, connected to the `din_rdy` input of the `triple_sdi_tx_output` module, toggles at a 74.25 MHz rate, throttling the data transfer rate between the receiver and the transmitter modules to 74.25 MHz instead of 148.5 MHz. In the other modes, `dout_rdy_3G` is held High all of the time. The `triple_sdi_rx_output` modules takes data from its inputs only when both its `ce` and `din_rdy` inputs are both High.

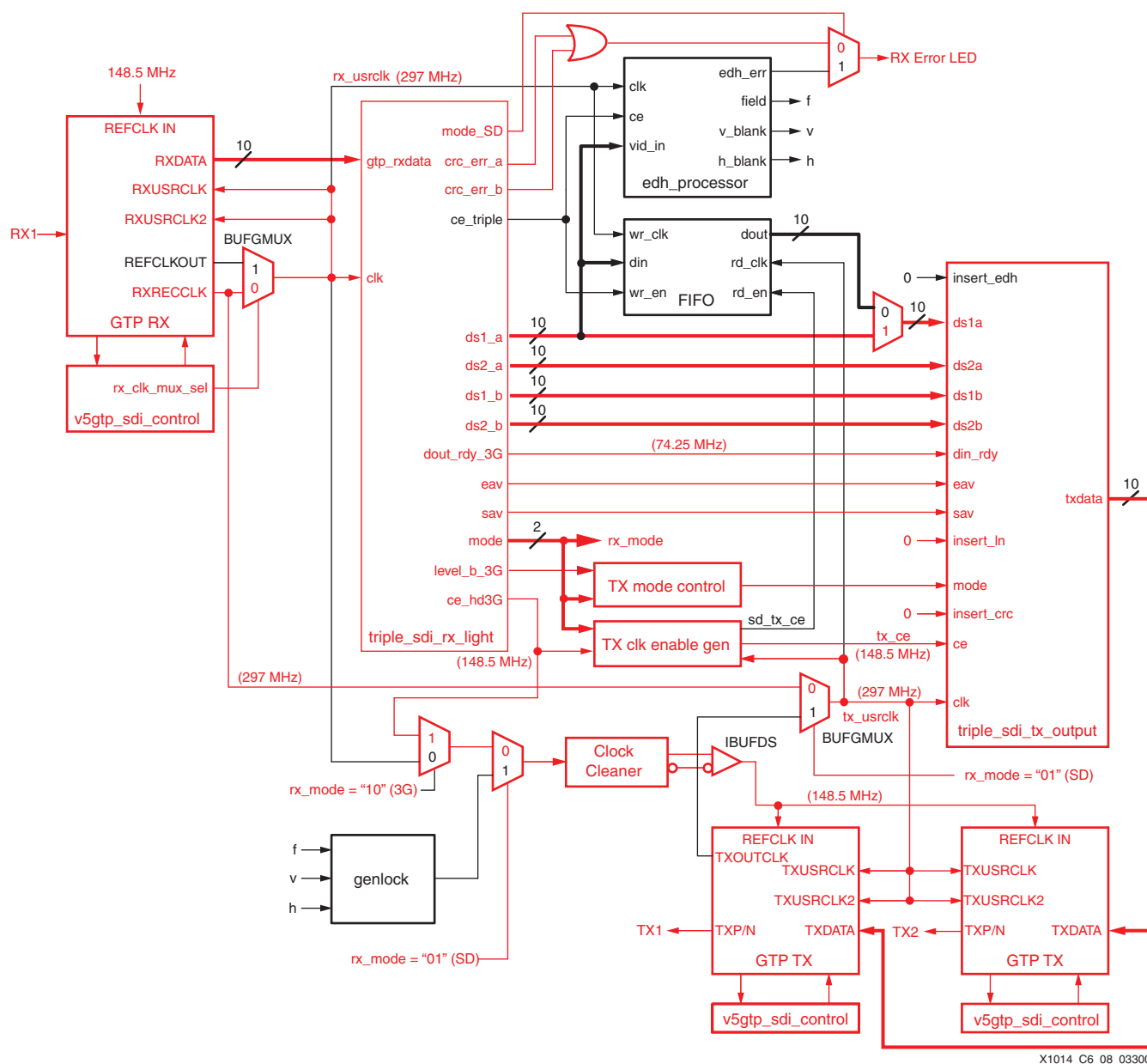


Figure 7-8: 3G-SDI Level B Clock and Datapaths

Other Considerations

Block diagrams [Figure 7-5](#) to [Figure 7-8](#) show a BUFGMUX driving rx_usrclk. The inputs to the BUFGMUX are REFCLKOUT and RXRECCLK. Normally, RXRECCLK is selected. However, when the input SDI bitstream stops for an extended period of time, RXRECCLK also stops. In these cases, the v5gtp_sdi_control module switches the BUFGMUX to allow REFCLKOUT to be used as a backup, free-running clock. REFCLKOUT can only be directly used for this function if the reference clock into the GTP_DUAL tile containing the RX is 148.5 MHz. If it is 74.25 MHz, REFCLKOUT would need to be doubled to 148.5 MHz before being connected to the input of the BUFGMUX. Any other free-running source of a 148.5 MHz or 297 MHz clock could be used instead of REFCLKOUT. This is an optional

feature and the BUFGMUX can be replaced with a BUFG driven only by the RXRECCLK output of the GTP RX.

Figure 7-5 to Figure 7-8 show three `v5gtp_sdi_control` modules, one for each GTP RX and TX. However, the real requirement is one `v5gtp_sdi_control` module per GTP_DUAL tile used to implement SDI interfaces. For example, if the two transmitters were in the same tile (which, in this case, is possible because they are always transmitting at the same bit rate), then one `v5gtp_sdi_control` module would suffice for both transmitters in the same tile. On the ML571 board, the GTP transceivers used for the SDI receivers and SDI transmitters are all in different GTP_DUAL tiles, so the example design uses three separate `v5gtp_sdi_control` modules.

The genlock device used on the ML571 board clock module is a GS4911B. It has a serial interface to the FPGA for control and status. A module in the example design is used to configure the genlock device to produce the 148.5 MHz rate reference clock needed to transmit SD-SDI from either PAL or NTSC sync signals. The EDH processor determines whether the video being received is PAL or NTSC. The GS4911B control module uses the video format signal from the EDH processor to determine how to configure the GS4911B. Applications that need the SD video format detection function provided by the EDH processor but do not need to do EDH error checking can use the autodetect module rather than the full EDH processor. The clock cleaner device used on the ML571 board clock module is a GS4915.

The example design is based on the “light” triple-rate SDI RX and TX modules. It is certainly possible to build a pass-through design using the full-featured triple-rate SDI RX and TX modules. However, for a simple pass-through system like the one presented in this chapter, the unpacking and packing logic included in the full-featured modules is not needed. For applications that process the video between the RX and TX, however, and require native video rather than raw SDI data streams, the full-featured RX and TX would be advantageous. The techniques for clocking a pass-through design built with the full-featured RX and TX modules are essentially the same as those described here.

This design does not modify EDH packets or CRC words as they pass through the system because the data streams are not modified. However, for more complex applications that modify the audio or video, the transmitter must update the EDH packets in SD-SDI mode and the CRC words in HD-SDI or 3G-SDI modes. This is done by asserting the `insert_crc` and `insert_edh` input ports of the `triple_sdi_tx_output` module High.

Reference Design

The triple-rate SDI pass-through reference design is available at <https://secure.xilinx.com/webreg/clickthrough.do?cid=119758>. Open the ZIP archive and extract the file `xapp1014_c7_GTP_SDI_PassThru.zip`.

Conclusion

The complexities of a triple-rate SDI pass-through application are primarily in the clocking schemes required to support all the different SDI modes. The example design and the block diagrams of this chapter illustrate how to handle these clocking issues. The example design can serve as a starting point for building any SDI applications that must receive SDI signals, process the video and audio, and then retransmit the video and audio via SDI transmitters.

Triple-Rate SDI for Virtex-5 FXT and TXT Devices

Summary

The RocketIO™ GTX transceivers available in Virtex®-5 FXT and TXT devices can be used to implement SDI receivers and transmitters that support all three SMPTE serial digital video interface standards: SD-SDI, HD-SDI, and 3G-SDI (triple-rate SDI). This chapter describes triple-rate SDI RX and TX reference designs for GTX transceivers.

These triple-rate SDI reference designs have been designed specifically for the GTX transceivers available in Virtex-5 FXT and TXT devices. It cannot be used directly with other Xilinx® FPGAs.

Introduction

While similar to the GTP transceivers in Virtex-5 LXT and SXT devices, the Virtex-5 FPGA GTX transceivers are slightly different. The triple-rate SDI reference designs for GTP transceivers cannot be used directly with GTX transceivers. Thus, triple-rate SDI reference designs have been created specifically for Virtex-5 FPGA GTX transceivers.

All triple-rate SDI interfaces using the GTX transceiver consist of three main parts:

- The GTX transceiver (instantiated in the design using a wrapper created by the RocketIO GTX Transceiver Wizard)
- The `v5gtx_sdi_control` module
- The triple-rate SDI RX or TX datapath modules

Refer to [Chapter 4, Implementing SMPTE Serial Digital Interfaces with RocketIO GTX Transceivers](#) for details about using the GTX transceivers in SDI applications. This chapter describes how to use the RocketIO GTX Wizard to create the GTX wrapper used to instantiate GTX transceivers in the user application. The chapter also describes, in detail, the `v5gtx_sdi_control` module that provides essential functions for both the GTX transceiver and the triple-rate SDI RX and TX modules.

The triple-rate SDI reference designs for the Virtex-5 FPGA GTP transceiver are available in two varieties: “light” and “full-featured.” The primary difference between the light and full-featured versions is that the full-featured versions provide all the packing and unpacking functions needed to convert between native video and SDI data streams for dual-link HD-SDI and 3G-SDI. The triple-rate SDI RX and TX reference designs for the Virtex-5 FPGA GTX transceiver are only available in one version. This version is generally equivalent to the GTP transceiver “light” version.

The other major difference between the GTX and GTP transceiver triple-rate SDI reference designs is that the triple-rate SDI RX and TX datapaths for GTX transceivers are designed to interface to the GTX transceiver with 20-bit data ports instead of the 10-bit data ports used with the GTP transceiver. In each case, the reference designs are designed to minimize the clocking resources needed to support SDI interfaces. Minimizing clock resources requires 10-bit interfaces with the GTP transceiver, but requires 20-bit interfaces with the GTX transceiver. Because the GTX transceiver SDI datapath modules are designed for 20-bit interfaces, the clock frequencies required for the datapaths are half of those used with the GTP transceiver SDI datapath modules.

Supported Video Formats

The triple-rate SDI receiver and transmitter support the video formats shown in [Table 8-1](#). The receiver and transmitter do not convert between native video and SDI data streams for all video formats. The formats that require conversion between SDI data streams and native video using external formatting modules are marked as “External” in the Native Video Packing column. Formats that are directly supported are marked as “Not required” in the Native Video Unpacking column.

Table 8-1: Supported Video Formats

Interface	Video Standard	Sampling Structure/ Bit Depth	Frame/Field Rate (Hz)	Native Video Unpacking
SD-SDI SMPTE 259M-C (other bit rates possible)	PAL	4:2:2 Y'C _B 'C _R ' 10-bit or 8-bit	50	Not required
	NTSC	4:2:2 Y'C _B 'C _R ' 10-bit or 8-bit	59.94	Not required
HD-SDI SMPTE 292M	SMPTE 274M	4:2:2 Y'C _B 'C _R ' 10-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	Not required
	SMPTE 296M	4:2:2 Y'C _B 'C _R ' 10-bit	720p: 23.98, 24, 25, 29.97, 30, 50, 59.94, 60	Not required
	SMPTE 260M	4:2:2 Y'C _B 'C _R ' 10-bit	1035i: 59.94, 60	Not required
	SMPTE 295M	4:2:2 Y'C _B 'C _R ' 10-bit	1080i: 50	Not required

Table 8-1: Supported Video Formats (Cont'd)

Interface	Video Standard	Sampling Structure/ Bit Depth	Frame/Field Rate (Hz)	Native Video Unpacking
3G-SDI Level A SMPTE 425M-A	SMPTE 274M	4:2:2 Y'C _B 'C _R ' 10-bit	1080p: 50, 59.94, 60	Not required
		4:4:4 or 4:4:4:4 Y'C _B 'C _R ' or RGB 10-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	External
		4:4:4 Y'C _B 'C _R ' or RGB 12-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	External
		4:2:2 Y'C _B 'C _R ' 12-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	External
	SMPTE 296M	4:4:4 or 4:4:4:4 Y'C _B 'C _R ' or RGB 10-bit	720p: 23.98, 24, 25, 29.97, 30, 50, 59.94, 60	External
	SMPTE 428-9	4:4:4 X'Y'Z' 12-bit	2048 X 1080p: 24	External
3G-SDI Level B SMPTE 425M-B	SMPTE 372M	See Dual Link HD-SDI SMPTE 372M in this table.		
	2 X HD-SDI streams	See HD-SDI SMPTE 292M in this table.		
Dual Link HD-SDI SMPTE 372M	SMPTE 274M	4:2:2 Y'C _B 'C _R ' 10-bit	1080p: 50, 59.94, 60	External
		4:4:4 or 4:4:4:4 Y'C _B 'C _R ' or RGB 10-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	External
		4:4:4 Y'C _B 'C _R ' or RGB 12-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	External
		4:2:2 Y'C _B 'C _R ' 12-bit	1080p: 23.98, 24, 25, 29.97, 30 1080i: 50, 59.94, 60 1080PsF: 23.98, 24, 25, 29.97, 30	External
	SMPTE 428-9	4:4:4 X'Y'Z' 12-bit	2048 X 1080p: 24	External

GTX Triple-Rate SDI RX Reference Design

The `triple_sdi_rx_light_20b` module, combined with a GTX receiver and a `v5gtx_sdi_control` module, implements a triple-rate receiver as shown in [Figure 8-1](#). It has these features:

- Only a single reference clock frequency is required to receive the five supported bit rates:
 - 270 Mb/s
 - 1.485 Gb/s
 - 148.5/1.001 Gb/s
 - 2.97 Gb/s
 - 2.97/1.001 Gb/s
- The receiver has a bit rate detector capable of distinguishing between the two HD-SDI or 3G-SDI bit rates. An output signal from the receiver indicates to the application which bit rate is being received.
- The receiver automatically detects the SDI standard of the input signal (3G-SDI, HD-SDI, or SD-SDI) and reports the current mode on an output port.
- The receiver detects and reports the video transport format (i.e., 1080p 30 Hz, 1080i 50 Hz, etc.).
- The receiver supports both 3G-SDI level A and level B, and automatically detects whether the 3G-SDI data streams are level A or B. The 3G-SDI level is reported on an output port.
- CRC error checking is supported for HD-SDI and 3G-SDI.
- SMPTE 352M video payload ID (VPID) packets are captured for all SDI standards. All captured SMPTE 352M packet data is available on output ports for one or two streams (for those formats that require SMPTE 352M packets in both streams).
- The receiver datapath is designed to interface with a 20-bit GTX RXDATA port to minimize global clocking resources. Only a single global clock is required for the receiver, driven by the RXRECCLK of the GTX receiver. No DCMs or PLLs are required for most implementations.
- A BUFGMUX switches the main clock to a backup reference clock when the recovered clock stops due to interruptions of the input serial bitstream.

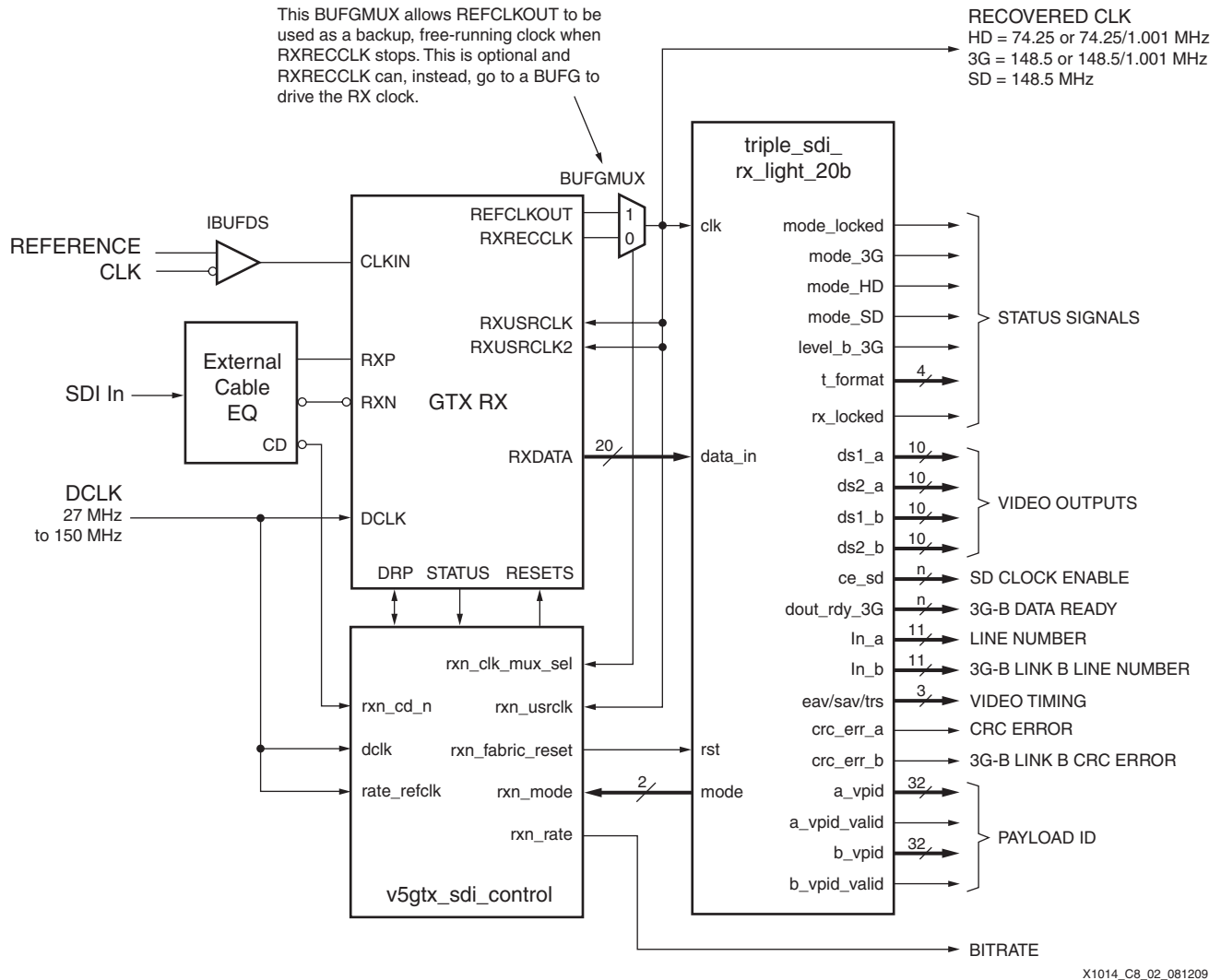


Figure 8-1: Block Diagram of the Triple-Rate SDI RX

Table 8-2 lists the ports of the triple_sdi_rx_light_20b module.

Table 8-2: Ports of triple_sdi_rx_light_20b Module

Port Name	I/O	Width	Description
Inputs			
clk	In	1	This is the recovered clock. It should be connected to the global or regional clock that drives the RXUSRCLK and RXUSRCLK2 clock inputs of the GTX RX. This clock is usually driven by the RXRECCLK output of the GTX RX. The clock frequency must be 148.5 MHz (or 148.5/1.001 MHz) for 3G-SDI, 74.25 MHz (or 74.25/1.001 MHz) for HD-SDI, and 148.5 MHz for SD-SDI.

Table 8-2: Ports of `triple_sdi_rx_light_20b` Module (Cont'd)

Port Name	I/O	Width	Description
rst	In	1	This is the asynchronous reset. The falling edge of this reset signal must meet the reset recovery time of all flip-flops relative to next rising edge of clk. This input can be driven by the rxn_fabric_reset output of the v5gtx_sdi_control module which, when combined with proper timing constraints, results in proper timing of the reset signal to the synchronous elements it resets.
data_in	In	20	This input should be connected to the 20-bit RXDATA port of the GTX wrapper module.
frame_en	In	1	This is the framer enable signal. The framer automatically readjusts the word alignment to match the alignment of each TRS (EAV or SAV) when this input is High. This input can be used to implement TRS alignment filtering. For example, if the nsp output is connected to the frame_en input, the framer ignores a single misaligned TRS, keeping the existing word alignment until the new word alignment is confirmed by a second matching TRS. If TRS filtering is not desired, this input must be tied High. Also, it is important to turn off any TRS filtering on the synchronous switching line by driving the frame_en input High on the synchronous switching lines.
Outputs			
ce_sd	Out	NUM_SD_CE	This output port consists of NUM_SD_CD identical copies of the 27 MHz data ready signal generated by the SD-SDI data recovery unit in SD-SDI mode. In HD-SDI and 3G-SDI modes, this output is always High.
mode	Out	2	<p>This port indicates the current SDI mode of the receiver:</p> <ul style="list-style-type: none"> 00 = HD-SDI 01 = SD-SDI 10 = 3G-SDI <p>The mode port changes value as the receiver searches for the correct mode. During this time, the mode_locked output is Low. When the receiver detects the correct SDI mode matching the input data stream, the mode_locked output goes High.</p> <p>This output port must drive the v5gtx_sdi_control module's rxn_mode input for the particular RX unit in the GTX_DUAL tile used to implement this SDI receiver.</p>
mode_HD mode_SD mode_3G	Out	1	These three output ports are decoded unary versions of the mode port and are provided for convenience. Unlike the mode output that changes continuously as the receiver seeks to identify and lock to the incoming signal, these outputs are all forced Low when the receiver is not locked. The mode output matching the current operating mode of the receiver is High when mode_locked is High. It is essential, however, that these outputs not be used to control any data or control paths essential for locking the receiver to the incoming signal; those paths must be controlled directly by decoding the mode port.
mode_locked	Out	1	When this output is Low, the receiver is actively searching for the SDI mode that matches the input data stream. During this time, the mode output port changes frequently. When the module locks to the correct SDI mode, the mode_locked output goes High.

Table 8-2: Ports of triple_sdi_rx_light_20b Module (Cont'd)

Port Name	I/O	Width	Description
t_format	Out	4	This output port indicates the video transport timing format of the SDI signal in HD and 3G modes only. See Table 8-3, page 253 for encoding. The output port is only valid when rx_locked is High. This output port is not valid in SD mode.
level_b_3G	Out	1	In 3G-SDI mode, this output is asserted High when the input signal is level B, and Low when it is level A. This output is only valid in 3G-SDI mode.
rx_locked	Out	1	This output is High when the receiver is locked to the incoming signal. In HD and 3G modes, this output is driven by the transport format detector that generates the t_format output. In SD mode, this output is identical to the mode_locked signal. Because this output is driven by the transport format detector, there can be more than one video frame time of delay from when the receiver is actually locked to the input signal until rx_locked is asserted. During this time, the transport format detector determines the transport format. There is no locked indicator for SD-SDI.
nsp	Out	1	When this output is High, it indicates that the framer has detected a TRS at a new word alignment. If frame_en is High, this output is only asserted briefly. If frame_en is Low, this output remains asserted until the framer is allowed to readjust to the new TRS alignment.
ln_a	Out	11	This is the current line number captured from the LN words of the Y data stream. This output is valid in HD and 3G modes, but not in SD mode. In 3G level B mode, it represents the line number from the Y data stream of link A. For any case where the interface line number is not the same as the picture line number, such as for 1080p 60 Hz carried on 3G-SDI level B or dual-link HD-SDI, this output equals the interface line number.
a_vpid	Out	32	All four user data bytes of the SMPTE 352M packet from data stream 1 are output on this port in the following format: MSB to LSB (byte4, byte3, byte2, byte1). This output is valid only when a_vpid_valid is High and works in any SDI mode. In 3G-SDI level A mode, this port outputs the VPID data captured from data stream 1 (Y). In 3G-SDI level B mode, it outputs the VPID data captured from data stream 1 of link A (dual-link streams) or HD-SDI stream 1 (dual streams).
a_vpid_valid	Out	1	This output is High if a_vpid is valid.
b_vpid	Out	32	All four user data bytes of the SMPTE 352M packet from data stream 2 are output on this port in the following format: MSB to LSB (byte4, byte3, byte2, byte1). This output is valid only in 3G-SDI mode and only when b_vpid_valid is High. In 3G-SDI level A mode, this port outputs the VPID data captured from data stream 2 (C). In 3G-SDI level B mode, it outputs the VPID data captured from data stream 1 of link B (dual-link streams) or HD-SDI stream 2 (dual streams).
b_vpid_valid	Out	1	This output is High if b_vpid is valid.
crc_err_a	Out	1	This output is asserted High for one sample period when a CRC error is detected on the previous video line. For 3G-SDI level B, this output indicates CRC errors on data stream 1 only. A second output called crc_err2 indicates CRC errors on data stream 2 for 3G-SDI level B. This output is not valid in SD-SDI mode.

Table 8-2: Ports of `triple_sdi_rx_light_20b` Module (Cont'd)

Port Name	I/O	Width	Description
ds1_a	Out	10	This output outputs these data streams, depending on the video standard: <ul style="list-style-type: none"> SD-SDI: Multiplexed Y/C data stream. HD-SDI: Y data stream. 3G-SDI level A: Data stream 1 (Y). 3G-SDI level B: Data stream 1 (Y) of link A or HD-SDI stream 1.
ds2_a	Out	10	This output outputs these data streams, depending on the video standard: <ul style="list-style-type: none"> SD-SDI: Not used. HD-SDI: C data stream. 3G-SDI level A: Data stream 2 (C). 3G-SDI level B: Data stream 2 (C) of link A or HD-SDI stream 2.
eav	Out	1	This output is asserted High for one sample time when the XYZ word of an EAV is present on the data stream output ports.
sav	Out	1	This output is asserted High for one sample time when the XYZ word of an SAV is present on the data stream output ports.
trs	Out	1	This output is asserted High for four sample times as all four words of an EAV or SAV are output on the data stream output ports.
recclk_txdata	Out	20	This output can be connected to a GTX TXDATA port to use the GTX TX to synthesize a 270 MHz recovered clock for SD-SDI.
Outputs Used Only in 3G-SDI Level B Mode			
dout_rdy_3G	Out	NUM_3G_DRDY	In 3G-SDI level B mode, the output data rate is 74.25 MHz, but the clock frequency is 148.5 MHz. The dout_rdy_3G outputs are asserted at a 74.25 MHz rate in level B mode. This output is always High in all other modes, allowing it to be used as a clock enable to downstream modules.
ds1_b	Out	10	This is data stream 1 (Y) of link B or HD-SDI stream 2.
ds2_b	Out	10	This is data stream 2 (C) of link B or HD-SDI stream 2.
crc_err_b	Out	1	This is the CRC error for link B or HD-SDI stream 2.
ln_b	Out	11	This line number output is only valid in 3G-SDI level B mode. It represents the line number from the Y data stream of link B or HD-SDI stream 2. For any case where the interface line number is not the same as the picture line number, this output equals the interface line number.

Table 8-3 shows the encoding of the `t_format` port. This port indicates the transport format (not always the same as the picture format) calculated by the receiver by counting words per line and active lines per field or frame. The `t_format` port can uniquely identify most video transport formats, but cannot distinguish between transport formats that have identical timing. For example, it cannot differentiate between 1080i 60 Hz and 1080PsF 30 Hz (1080p 30 Hz video carried on a 1080i 60 Hz transport) because there is no difference in the transport timing.

In dual-link HD-SDI and 3G-SDI level B modes, the 1080p 50 Hz and 60 Hz video formats are carried on an interlaced transport. Thus, the module reports them as being 1080i 50 Hz and 1080i 60 Hz, respectively. However, in 3G-SDI level A mode, these two video formats are carried progressively and are uniquely identified as 1080p 50 Hz and 1080p 60 Hz (codes 1110 and 1101 respectively).

Table 8-3: **t_format** Output Port Encoding for **triple_sdi_rx_light_20b** Module

t_format	Standard	Video Format (Frame Rate for p and Field Rate for i)
0000	SMPTE 260M	1035i 49.94 Hz and 60 Hz
0001	SMPTE 295M	1080i 50 Hz
0010	SMPTE 274M	1080i 59.94 Hz and 60 Hz 1080PsF 29.97 Hz and 30 Hz
0011	SMPTE 274M	1080i 50 Hz 1080PsF 25 Hz
0100	SMPTE 274M	1080p 29.97 Hz and 30 Hz 1080p 59.94 Hz and 60 Hz (3G-SDI level B only)
0101	SMPTE 274M	1080p 25 Hz 1080p 50 Hz (3G-SDI level B only)
0110	SMPTE 274M	1080p 23.98 Hz and 24 Hz
0111	SMPTE 296M	720p 59.94 Hz and 60 Hz
1000	SMPTE 274M	1080PsF 23.98 Hz and 24 Hz
1001	SMPTE 296M	720p 50 Hz
1010	SMPTE 296M	720p 29.97 Hz and 30 Hz
1011	SMPTE 296M	720p 25 Hz
1100	SMPTE 296M	720p 23.98 Hz or 24 Hz
1101	SMPTE 274M	1080p 59.94 Hz or 60 Hz (3G-SDI level A only)
1110	SMPTE 274M	1080p 50 Hz (3G-SDI level A only)
1111	Reserved	

Table 8-4 describes the parameters of the `triple_sdi_rx_light_20b` module.

Table 8-4: **Parameters of triple_sdi_rx_light_20b** Module

Parameter	Default Value	Description
NUM_SD_CE	2	This parameter specifies the number of identical <code>ce_sd</code> clock enable outputs provided by the module. It should never be set to less than 1.
NUM_3G_DRDY	2	This parameter specifies the number of identical <code>dout_rdy_3G</code> data ready outputs provided by the module. It should never be set to less than 1.
ERRCNT_WIDTH	4	This is a parameter of the SDI mode detection module. It specifies the number of bits used in the error counter. This counter must be wide enough to support counting to <code>MAX_ERRS_LOCKED</code> and <code>MAX_ERRS_UNLOCKED</code> .

Table 8-4: Parameters of `triple_sdi_rx_light_20b` Module (Cont'd)

Parameter	Default Value	Description
MAX_ERRS_LOCKED	15	This is a parameter of the SDI mode detection module. It specifies the maximum number of consecutive lines with errors allowed while the receiver is locked to the SDI mode before the receiver moves to the unlocked state and begins searching for the correct SDI mode.
MAX_ERRS_UNLOCKED	2	This is a parameter of the SDI mode detection module. It specifies the maximum number of consecutive lines with errors allowed while the receiver is searching for the correct SDI mode before it continues the search by moving to another mode.

Triple-Rate SDI RX Modes

The triple-rate SDI receiver automatically determines the standard of the incoming SDI mode (SD-SDI, HD-SDI, 3G-SDI level A, or 3G-SDI level B). The recovered clock frequency from the GTX transceiver and the number of data streams output by the triple-rate SDI receiver depend on the SDI mode.

Triple-Rate SDI Clocking

The GTX transceiver CDR unit requires a reference clock, which can be either 148.5 MHz or 148.5/1.001 MHz (or 74.25 MHz or 74.25/1.001 MHz). These reference clock frequencies support all supported SDI bit rates for SD-SDI, HD-SDI, and 3G-SDI.

The clock input (clk) to the `triple_sdi_rx_light_20b` module must come from a global (or regional) clock buffer driven by the recovered clock from the GTX receiver (RXRECCLK). As described in [Chapter 4, Implementing SMPTE Serial Digital Interfaces with RocketIO GTX Transceivers](#), this clock can stop if the input SDI serial bitstream stops. Thus, it might be desirable to use a BUFGMUX as the global clock buffer. The `v5gtx_sdi_control` module has an output to control a BUFGMUX so that a backup free-running reference clock can be supplied in place of RXRECCLK when RXRECCLK is stopped. This is entirely optional and is not required for all applications.

The frequency of the recovered clock depends on the current SDI mode of the receiver. In HD-SDI mode, it is 74.25 MHz (or 74.25/1.001 MHz). In 3G-SDI mode, it is 148.5 MHz or 148.5/1.001 MHz. For SD-SDI, it is equal to the reference clock frequency, usually 148.5 MHz.

Internally, most of the logic in the `triple_sdi_rx_light_20b` module runs at the recovered clock frequency for HD-SDI and 3G-SDI (some portions run at half the recovered clock frequency in 3G-SDI level B mode). For SD-SDI, the internal data rate is 27 MHz. Clock enables are used in the `triple_sdi_rx_light_20b` module to run the various sections at the proper rates.

The `triple_sdi_rx_light_20b` module also supplies one or more copies of the SD-SDI clock enable on the `ce_sd` output port. These clock enables can be used, in conjunction with the global recovered clock, to clock logic downstream from the `triple_sdi_rx_light_20b` module at the 27 MHz SD-SDI data rate. The `NUM_SD_CE` parameter/generic specifies the number of identical copies of `ce_sd` output by the module. `NUM_SD_CE` must never be set to less than one.

The `ce_sd` clock enable is generated by the SD-SDI data recovery unit (DRU) located inside the `triple_sdi_rx_light_20b` module. Typically, the cadence of the `ce_sd` signal is 5/6/5/6 cycles of the clock. However, the cadence varies occasionally to make up for differences between the actual recovered data rate and the frequency of the GTX transceiver reference clock. The number of clock cycles between assertions of `ce_sd` can be more than six and less than five at times. In all modes except SD-SDI, `ce_sd` is always High.

Similarly, the module also outputs a `dout_rdy_3G` data ready signal. In 3G-SDI mode, `dout_rdy_3G` toggles at half the clock frequency in 3G-SDI mode because the output data rate is 74.25 MHz, while the clock frequency is 148.5 MHz. In all modes except 3G-SDI level B, `dout_rdy_3G` is always High. The `NUM_3G_DRDY` parameter/generic specifies the number of identical copies of the `dout_rdy_3G` signal output by the module. `NUM_3G_DRDY` should never be set to less than one.

SD-SDI Output Timing

In SD-SDI mode, the frequency of the `RXRECCLK` from the GTX RX is 148.5 MHz. The recovered SD-SDI data stream is output on the `ds1_a` port with the Y and C components interleaved at a 27 MHz data rate. The `trs`, `eav`, and `sav` timing signals are also valid.

In SD-SDI mode, the 148.5 MHz `RXRECCLK` from the GTX RX is not a recovered clock because the GTX RX is locked to the reference clock and asynchronously samples the input bitstream. `RXRECCLK` is, therefore, an exact multiple of the reference clock supplied to the GTX RX.

The GTX RX provides the oversampled SD data to the `triple_sdi_rx_light_20b` module where a DRU recovers the data. The DRU provides a data ready signal that is asserted when it has a 10-bit data word ready. The `triple_sdi_rx_light_20b` module outputs this data ready signal on the `ce_sd` output. On average, this output is asserted once every 5.5 clock cycles by using a 5/6/5/6 cadence. The output cadence is occasionally altered when the DRU needs to catch up to the actual data rate. This occurs because the GTX RX reference clock is a local clock and is asynchronous to the actual timing of the incoming SD-SDI bitstream.

Figure 8-2 shows the timing of the SD video and timing signals produced by the `triple_sdi_rx_light_20b` module. The outputs only change on the rising edge of `clk` when `ce_sd` is High. The timing of the EAV sequence is shown. The `sav` output signal has the same timing as the `eav` signal during SAV sequences.

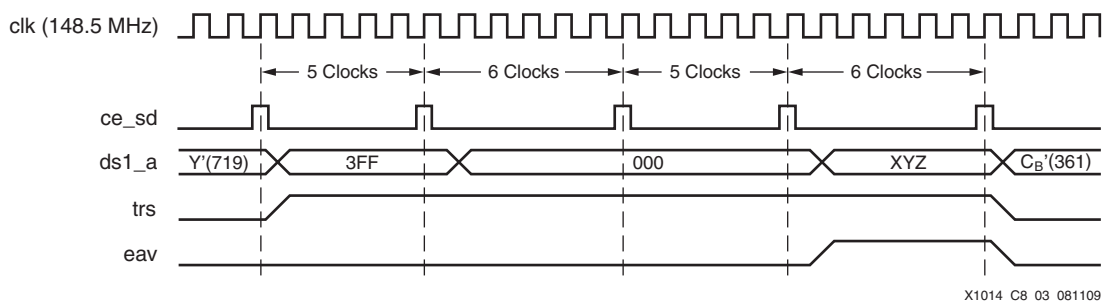


Figure 8-2: SD-SDI Output Timing of `triple_sdi_rx_light_20b` Module

HD-SDI Output Timing

In HD-SDI mode, the RXRECCLK recovered clock from the GTX RX is a true recovered clock and runs at 74.25 MHz (or 74.25/1.001 MHz). The Y and C data streams of the HD-SDI signal are output on the ds1_a and ds2_a ports, respectively, along with the timing signals trs, eav, and sav. The line number is output on the ln_a port. In HD-SDI mode, the line number changes during the CRC0 word. Figure 8-3 shows the timing of the HD-SDI outputs.

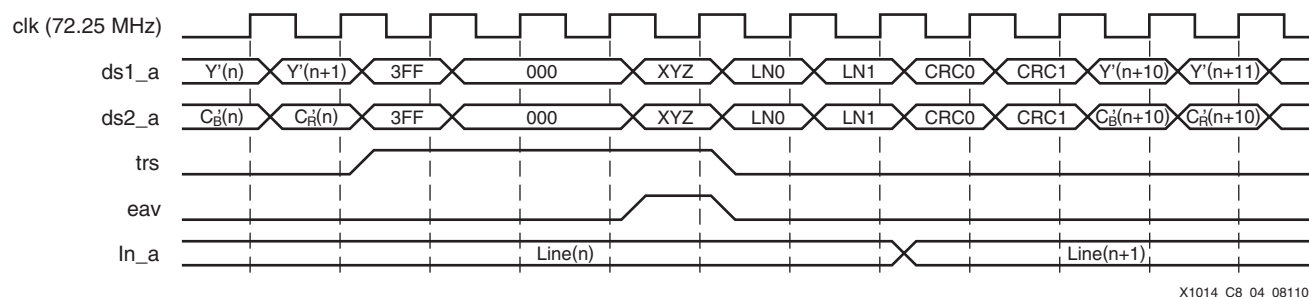


Figure 8-3: HD-SDI Output Timing of `triple_sdi_rx_light_20b` Module

3G-SDI Output Timing

In 3G-SDI mode, the RXRECCLK frequency is 148.5 MHz (or 148.5/1.001 MHz). This is a true recovered clock and runs at either the video sample rate in level A mode or twice the video sample rate in level B mode.

Figure 8-4 shows the output timing of the `triple_sdi_rx_light_20b` module when receiving a 1080p 50 Hz, 59.94 Hz, or 60 Hz signal in 3G-SDI level A mode. Other video formats, such as 4:4:4 10-bit or 12-bit, have identical timing, but the video samples are packed per SMPTE 425M such that it takes two words on each data stream to carry a single video sample. The `triple_sdi_rx_light_20b` module does not unpack the other video formats, but outputs them in their packed format. Thus, the output timing for all video formats in 3G-SDI level A mode is identical to that shown in Figure 8-4, with the video mapped onto the two data streams per the SMPTE 425M specification.

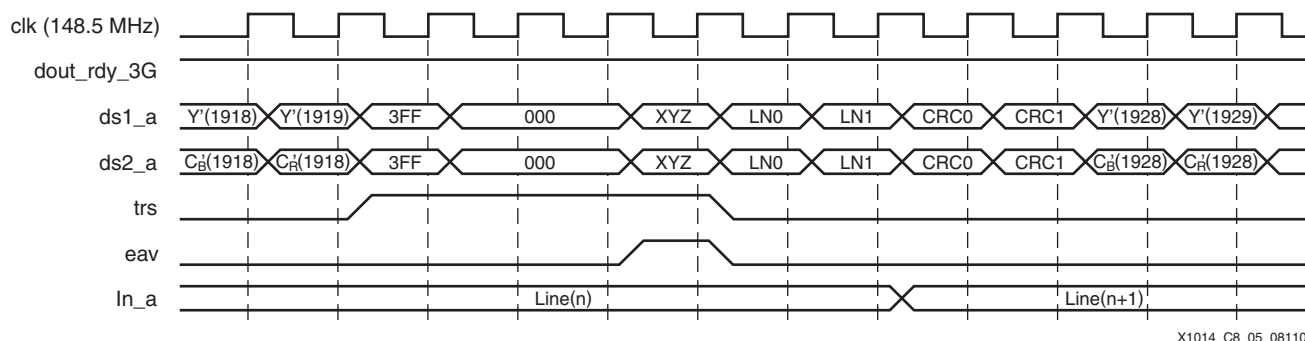


Figure 8-4: 3G-SDI Level A Output Timing (1080p 50 Hz or 60 Hz) of `triple_sdi_rx_light_20b` Module

Figure 8-5 shows the output timing of the `triple_sdi_rx_light_20b` module when receiving a signal in 3G-SDI level B mode. The `dout_rdy_3G` signal is asserted every other clock cycle indicating when data words are available on the four 10-bit data stream output ports.

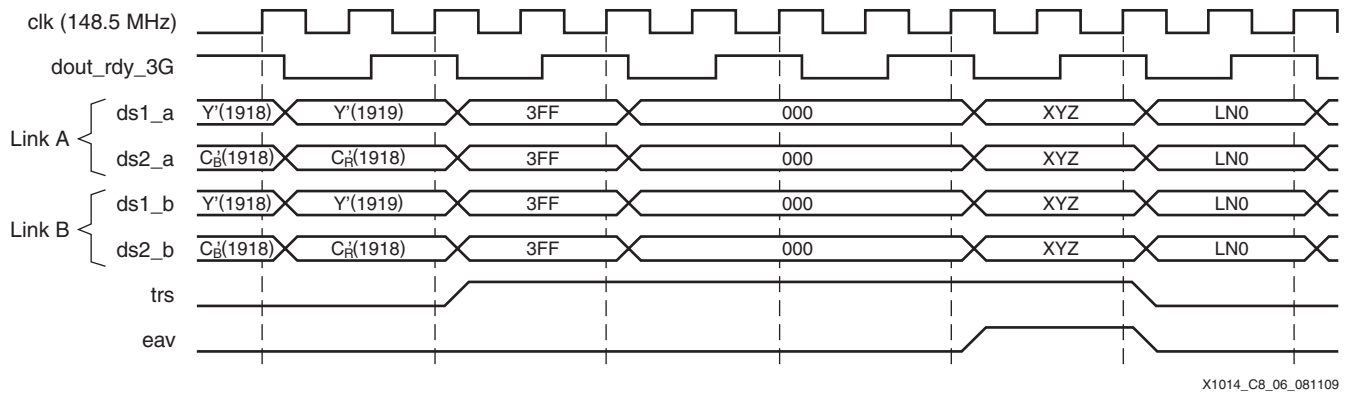


Figure 8-5: 3G-SDI Level B Output Timing of `triple_sdi_rx_light_20b` Module

Both line number output ports are active. When the level B signal is carrying SMPTE 372M dual-link data, the values on both `ln_a` and `ln_b` are identical and indicate the interface line number, not the picture line number. When carrying two independent HD-SDI signals, the two line number ports are not necessarily the same, depending on whether the two HD-SDI signals are vertically synchronized or not. The `ln_a` and `ln_b` ports, while not shown on the diagram, change at the same relative position as they do for level A and HD-SDI—right after the LN1 word.

When SMPTE 372M dual-link data is carried on the 3G-SDI level B interface, link A is output on `ds1_a` and `ds2_a` and link B is output on `ds1_b` and `ds2_b`. These four links carry video mapped per SMPTE 372M. The `triple_sdi_rx_light_20b` module does not contain the necessary logic to unpack the SMPTE 372M data streams into native video.

When two independent HD-SDI streams are carried on the 3G-SDI level B interface, the first HD-SDI stream is output on `ds1_a` (Y) and `ds2_a` (C). The second HD-SDI stream is output on `ds1_b` (Y) and `ds2_b` (C). These two HD-SDI streams are horizontally synchronized so that their EAVs and SAVs line up exactly.

Other Triple-Rate SDI RX Design Considerations

Dual-Link HD-SDI

To implement a dual-link HD-SDI receiver, two triple-rate SDI receivers are paired with one receiving link A and the other receiving link B. Typically, the received data streams for the two links are skewed. Therefore, the skew must be removed. After deskewing, the data streams can be unpacked into native video, if desired.

Processing Embedded Audio and Other Ancillary Data

The data streams output from the triple-rate SDI receiver always have all ancillary data, including embedded audio packets, intact. Modules designed to process ancillary data can be connected to the data streams and timing signals output by the triple-rate SDI receiver.

SMPTE 352M VPID Packets

The triple-rate SDI receiver module captures SMPTE 352M packets present in the data streams for all SDI modes. For SD-SDI and HD-SDI, the four data bytes of the SMPTE 352M packet are output on the `a_vpid` port. The `a_vpid_valid` port indicates when

valid SMPTE 352M packets have been captured. This output has some hysteresis so that SMPTE 352M packets can be missing from a few fields or frames before the valid signal is negated. During the time that the valid output is asserted and new SMPTE 352M packets are not found, the data from the last valid SMPTE 352M packet received is output on the a_vpid port.

The SMPTE 425M 3G-SDI standard requires SMPTE 352M packets in both data streams. The triple-rate SDI receiver captures the SMPTE 352M packets from both streams, outputting the data from the packet in data stream 1 on a_vpid and the data from the packet in C data stream 2 on b_vpid. There are individual a_vpid_valid and b_vpid_valid outputs, too. The way that SMPTE 352M packets are inserted into the data streams for 3G-SDI level B changed with the 2008 revision of SMPTE 425M. This reference design expects SMPTE 352M packets in 3G-SDI level B to be located per the SMPTE 425M-2008 revision. The design does not detect and capture SMPTE 352M packets in 3G-SDI level B if they have been inserted per the earlier revision of SMPTE 425M.

SD-SDI EDH Error Detection

While the triple-rate SDI receiver detects CRC errors in HD-SDI and 3G-SDI modes, it does not contain an EDH error checker. However, either of the two EDH processors described in *Audio/Video Connectivity Solutions for Virtex-II Pro and Virtex-4 FPGAs* [Ref 5] can be connected downstream from the triple_sdi_rx_light_20b module to check the SD-SDI data stream for EDH errors.

SD-SDI Data Recovery Unit

The Virtex-5 FPGA GTX transceiver triple-rate SDI receiver uses a DRU based on the reference design described in *Dynamically Programmable DRU for High-Speed Serial I/O* [Ref 6]. The DRU provided in this reference design is an optimized version of the DRU described in that application note. It is optimized for use with 11X oversampled 270 Mb/s data. It is also optimized so that the reference clock used for SD-SDI reception can be either 148.5 MHz or 148.35 MHz (it performs equally well with either reference clock frequency). The optimized DRU is also much smaller than the general-purpose DRU implementation described in *Dynamically Programmable DRU for High-Speed Serial I/O*. The DRU in this reference design offers a number of advantages over the DRUs used in previous Xilinx SDI receivers:

- The DRU in this reference design offers a significant improvement in receiver jitter tolerance. With the optimized DRU, the Virtex-5 FPGA GTX SD-SDI receiver jitter tolerance exceeds 0.75 UI at 10 MHz jitter frequency.
- Because it is optimized for 11X oversampled 270 Mb/s data, the reference design presented only supports 270 Mb/s. However, the full DRU described in *Dynamically Programmable DRU for High-Speed Serial I/O* can also be used. This design can support non-integer oversampling rates. This allows support for other SD-SDI bit rates without requiring other GTX transceiver reference clock frequencies. For example, to support the 360 Mb/s SD-SDI bit rate, the DRU described in *Dynamically Programmable DRU for High-Speed Serial I/O* can be set to oversample at 8.25X ($360 \text{ Mb/s} \times 8.25 = 2.97 \text{ Gb/s}$).
- The optimized DRU provided with this reference design has the ability to use a GTX TX to synthesize a recovered clock for SD-SDI. The triple_rate_sdi_rx module has a 20-bit data port called recclk_txdata that can be connected to the TXDATA port of a GTX transceiver. The GTX TX reference clock must be exactly the same frequency as the reference clock used by the GTX RX that implements the

SD-SDI receiver. The serial output of the GTX TX is then a 270 MHz clock that is frequency locked to the SD-SDI signal.

GTX Transceiver Triple-Rate SDI TX Reference Design

The SDI transmitter for the Virtex-5 FPGA GTX transceiver provides the basic operations necessary to support SD-SDI, HD-SDI, dual-link HD-SDI, and 3G-SDI transmission. It does not do video mapping for 3G-SDI or dual-link HD-SDI. Video formats that require mapping (all 3G-SDI level A formats except 1080p 50 Hz, 59.94 Hz, and 60 Hz, all 3G-SDI level B formats, and all dual-link HD-SDI formats) must be mapped into SDI data streams prior to the triple-rate SDI TX module. The GTX triple-rate SDI transmitter has these features:

- Only two reference clock frequencies are required to support all SDI modes:
- It supports 148.5 MHz for SD-SDI at 270 Mb/s, HD-SDI at 1.485 Gb/s, and 3G-SDI at 2.97 Gb/s.
- It supports bit rates of 148.5/1.001 MHz for HD-SDI at 1.485/1.001 Gb/s and 3G-SDI at 2.97/1.001 Gb/s.
- It directly supports 3G-SDI level A transmission of 1080p 50 Hz, 59.94 Hz, and 60 Hz video.
- It transmits pre-formatted dual-link HD-SDI streams via either dual-link HD-SDI or 3G-SDI level B.
- With the addition of a 3G-SDI level A mapping module, it supports all 3G-SDI level A compatible video formats.
- It directly supports transmission of two independent HD-SDI streams via 3G-SDI level B.
- It supports EDH packet generation and updating for SD-SDI.
- It supports CRC generation and insertion for HD-SDI and 3G-SDI.
- SMPTE 352M video payload ID insertion can be enabled for all SDI modes.
- It is designed to interface with a 20-bit GTX transmitter TXDATA port to minimize global clocking resources. Only a single global or regional clock is required for the transmitter. No DCMs or PLLs are required.

Figure 8-6 shows a block diagram of the GTX triple-rate SDI TX.

The transmitter datapath consists of two modules: the `triple_sdi_vpid_insert` module and the `triple_sdi_tx_output_20b` module. These two modules, combined with the GTX wrapper module and the `v5gtx_sdi_control` module, form the triple-rate SDI TX reference design, as shown in Figure 8-6. The `v5gtx_sdi_control` module is described in Chapter 4, [Implementing SMPTE Serial Digital Interfaces with RocketIO GTX Transceivers](#). Also described in this chapter is the process of creating the necessary GTX wrapper in the RocketIO GTX Transceiver Wizard.

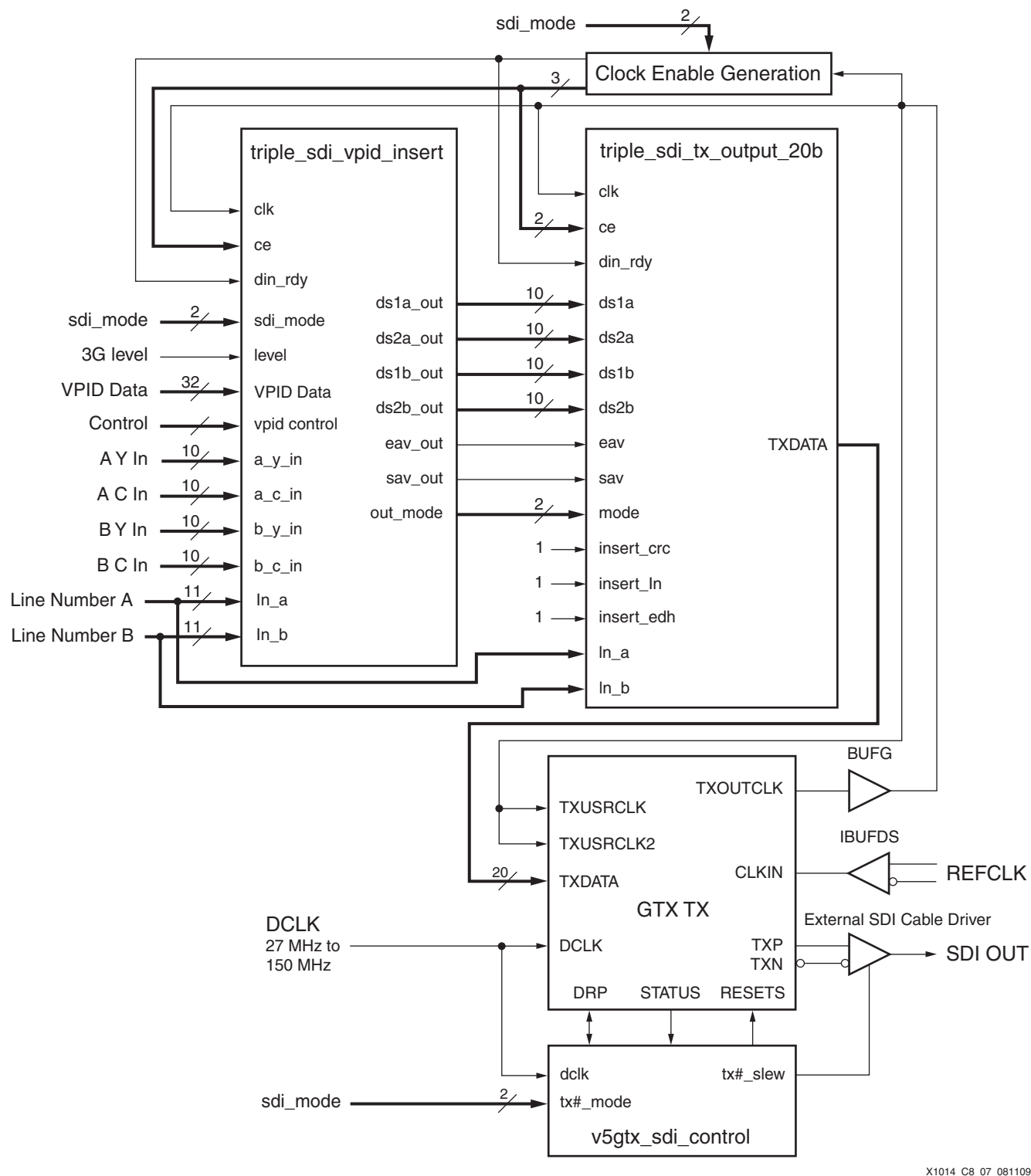


Figure 8-6: GTX Transceiver Triple-Rate TX Block Diagram

Triple-Rate SDI TX Datapath Modules

The triple-rate SDI TX datapath consists of two datapath modules, the `triple_sdi_vpid_insert` module, which inserts SMPTE 352M VPID packets into the data streams and the `triple_sdi_tx_output_20b` module, which creates the final output data stream that is suitable for input to the TXDATA port of the GTX transmitter.

Triple-Rate SDI VPID Insert Module

Table 8-5 describes the ports of the `triple_sdi_vpid_insert` module. This module is the front-end of the triple-rate SDI transmitter. Its main function is to insert SMPTE 352M VPID packets into the data streams, but it also serves some additional functions required by the `triple_sdi_tx_output_20b` module, such as detecting and identifying EAV and SAV sequences and generating the `out_mode` signals that connect to the `mode` port of the output module.

Table 8-5: Ports of `triple_sdi_vpid_insert` Module

Port Name	I/O	Width	Description
Inputs			
clk	In	1	This clock input must be driven by the same clock that drives the TXUSRCLK and TXUSRCLK2 ports of the GTX transmitter. It must have a frequency of 74.25 MHz or 74.25/1.001 MHz for HD-SDI, 148.5 MHz or 148.5/1.001 MHz for 3G-SDI, and 148.5 MHz for SD-SDI.
ce	In	1	The clock enable must be asserted at a 27 MHz rate for SD-SDI (with a mandatory 5/6/5/6 clock-cycle cadence). For other SDI modes, the clock enable is always High.
din_rdy	In	1	For SD-SDI, HD-SDI, and level A 3G-SDI, this input must be kept High at all times. For level B 3G-SDI, this input must be asserted every other clock cycle.
rst	In	1	This is an asynchronous reset. The falling edge of this reset signal must meet the reset recovery time of all flip-flops relative to the next rising edge of clk. This input can be driven by the <code>txn_fabric_reset</code> output of the <code>v5gtx_sdi_control</code> module which, when combined with proper timing constraints, results in proper timing of the reset signal to the synchronous elements it resets.
sdi_mode	In	2	This input port is used to select the SDI mode: <ul style="list-style-type: none"> 00 = HD-SDI (including dual-link HD-SDI) 01 = SD-SDI 10 = 3G-SDI 11 = Invalid
level	In	1	In 3G-SDI mode, this input determines whether the modules insert SMPTE 352M packets per level A (level = Low) or level B (level = High). Because these two 3G-SDI levels have quite different requirements for placement of SMPTE 352M packets, this input must be properly controlled. Otherwise, the data streams generated by the module are not legal.

Table 8-5: Ports of `triple_sdi_vpid_insert` Module (Cont'd)

Port Name	I/O	Width	Description
enable	In	1	When this input is High, SMPTE 352M packets are inserted into the data stream. If this input is Low, the packets are not inserted.
overwrite	In	1	If this input is High, SMPTE 352M packets already present in the data streams are overwritten. If this input is Low, existing SMPTE 352M packets are not overwritten. When transmitting SMPTE 372M dual link using 3G-SDI level B, existing SMPTE 352M packets in the data streams must be updated to indicate that the transport is 3G-SDI rather than HD-SDI. This module only updates these packets if overwrite is High.
byte1	In	8	This value is inserted as the first user data word of the SMPTE 352M packet. It must be valid during the entire HANC interval.
byte2	In	8	This value is inserted as the second user data word of the SMPTE 352M packet. It must be valid during the entire HANC interval.
byte3	In	8	This value is inserted as the third user data word of the SMPTE 352M packet. It must be valid during the entire HANC interval.
byte4a	In	8	This value is inserted as the fourth user data word of the SMPTE 352M packet. This word is used for the SMPTE 352M packets for SD-SDI, HD-SDI, and 3G-SDI level A. For 3G-SDI level B and dual-link HD-SDI, this value is used for the SMPTE 352M packet inserted into the Y channel of link A only. It must be valid during the entire HANC interval.
byte4b	In	8	This value is inserted as the fourth user data word of the SMPTE 352M packet that is inserted in the Y channel of link B (for 3G-SDI level B and dual-link HD-SDI only). This input value is not used for SD-SDI, HD-SDI, or 3G-SDI level A. It must be valid during the entire HANC interval.
ln_a	In	11	The current line number must be provided to the module through this port. SD-SDI only uses 10-bit line numbers, so the MSB of the port must be 0 in SD-SDI mode. The line number must be valid at least one clock cycle before the start of the HANC space (by the XYZ word of the EAV) and must remain valid during the entire HANC space. This input is the only line number input used for SD-SDI, HD-SDI, and 3G-SDI level A. For 3G-SDI level B, a second line number input port, <code>ln_b</code> , is also provided. The line numbers input on this port are used only to identify the line for the purposes of inserting SMPTE 352M packets. If SMPTE 352M packet insertion is disabled in a particular SDI mode, SD-SDI for example, valid line numbers do not need to be supplied on this port in that SDI mode.

Table 8-5: Ports of `triple_sdi_vpid_insert` Module (Cont'd)

Port Name	I/O	Width	Description
<code>ln_b</code>	In	11	This is the second line number input port used only for 3G-SDI level B. This additional line number port allows the two unrelated (but horizontally synchronized) HD-SDI signals to be vertically unsynchronized when level B carries two independent HD-SDI signals. Because <code>ln_b</code> is always used in 3G-SDI level B mode, when the input data streams are SMPTE 372M dual-link HD-SDI streams being transported on a 3G-SDI level B interface, <code>ln_b</code> must be equal to <code>ln_a</code> .
<code>line_f1</code>	In	11	The SMPTE 352M packet for field 1 is inserted on the line indicated by this value. For progressive video, the SMPTE 352M packet for the frame is inserted on this line. It must be valid during the entire HANC interval.
<code>line_f2</code>	In	11	The SMPTE 352M packet for field 2 is inserted on the line indicated by this value. For progressive video, this input port is ignored (and the <code>line_f2_en</code> port must be held Low). This value must be valid during the entire HANC interval.
<code>line_f2_en</code>	In	1	This input controls whether or not SMPTE 352M packets are inserted on the line indicated by <code>line_f2</code> . For interlaced video, this input must be High. For progressive video, this input must be Low. This input must be valid during the entire HANC interval. For progressive video transported on an interlaced transport, such as 1080p 60 Hz transported by either 3G-SDI level B or dual-link HD-SDI, SMPTE 352M packets must be inserted into both fields of the interlaced transport streams. In this case, the input must be High.
<code>a_y_in</code>	In	10	<ul style="list-style-type: none"> SD-SDI: The multiplexed Y/C data stream enters the module on this port. HD-SDI and 3G-SDI level A: The Y data stream enters the module on this port. 3G-SDI level B and dual-link HD-SDI: The Y data stream of link A enters the module on this port.
<code>a_c_in</code>	In	10	<ul style="list-style-type: none"> HD-SDI and 3G-SDI level A: The C data stream enters the module on this port. 3G-SDI level B and dual-link HD-SDI: The C data stream of link A enters the module on this port.
<code>b_y_in</code>	In	10	For 3G-SDI level B and dual-link HD-SDI, the Y data stream of link B enters the module on this port. When two VPID insert modules are used to process dual-link HD-SDI, this input is not used.
<code>b_c_in</code>	In	10	For 3G-SDI level B and dual-link HD-SDI, the C data stream of link B enters the module on this port. When two VPID insert modules are used to process dual-link HD-SDI, this input is not used.
Outputs			
<code>ds1a_out</code>	Out	10	This data stream output is the same as the <code>a_y_in</code> data stream, but with SMPTE 352M packets inserted.

Table 8-5: Ports of `triple_sdi_vpid_insert` Module (Cont'd)

Port Name	I/O	Width	Description
ds2a_out	Out	10	This data stream output is the same as the a_c_in data stream, but with SMPTE 352M packets inserted.
ds1b_out	Out	10	This data stream output is the same as the b_y_in data stream, but with SMPTE 352M packets inserted.
ds2b_out	Out	10	This data stream output is the same as the b_c_in data stream, but with SMPTE 352M packets inserted.
eav_out	Out	1	This output is High when the XYZ word of an EAV is output on the data stream outputs.
sav_out	Out	1	This output is High when the XYZ word of an SAV is output on the data stream outputs.
out_mode	Out	2	This output port must be connected to the mode input port of the <code>triple_sdi_tx_output_20b</code> module.

Triple-Rate SDI Output Module

Table 8-6 describes the ports of the `triple_sdi_tx_output_20b` module. This module is the back end of the triple-rate SDI transmitter. It takes in one, two, or four data streams (depending on the SDI mode) from the `triple_sdi_vpid_insert` module. The `triple_sdi_tx_output_20b` module optionally generates and inserts EDH packets (SD-SDI only) or CRC and LN words (HD-SDI and 3G-SDI only), scrambles the data, and outputs a single 20-bit data stream that is ready to be serialized by the GTX TX.

Table 8-6: Ports of `triple_sdi_tx_output_20b` Module

Port Name	I/O	Width	Description
Inputs			
clk	In	1	This clock input must be driven by the same clock that drives the TXUSRCLK and TXUSRCLK2 ports of the GTX transmitter. It must have a frequency of 74.25 MHz or 74.25/1.001 MHz for HD-SDI, 148.5 MHz or 148.5/1.001 MHz for 3G-SDI, and 148.5 MHz for SD-SDI.
ce	In	2	The clock enable must be asserted at a 27 MHz rate for SD-SDI (with a mandatory 5/6/5/6 clock cycle cadence). For other SDI modes, the clock enable is always High. This module takes two identical copies of the clock enable signal on its ce port. For most applications, both bits of the ce port can be driven with the same clock enable signal. The ce[0] bit controls everything except the EDH processor and the ce[1] bit controls only the EDH processor. If EDH packet insertion or updating is not required, ce[1] can be tied Low and the EDH processor is optimized out of the design by the synthesis tool.
din_rdy	In	1	For SD-SDI, HD-SDI, and level A 3G-SDI, this input must be kept High at all times. For level B 3G-SDI, this input must be asserted every other clock cycle.

Table 8-6: Ports of `triple_sdi_tx_output_20b` Module (Cont'd)

Port Name	I/O	Width	Description
rst	In	1	This is an asynchronous reset. The falling edge of this reset signal must meet the reset recovery time of all flip-flops relative to the next rising edge of clk. This input can be driven by the <code>txn_fabric_reset</code> output of the <code>v5gtx_sdi_control</code> module which, when combined with proper timing constraints, results in proper timing of the reset signal to the synchronous elements it resets.
mode	In	2	This input port is used to select the mode of operation of the <code>triple_sdi_tx_output_20b</code> module. The encoding on this port is not the same as the encoding on the mode input of the <code>triple_sdi_vpid_insert</code> module. The <code>triple_sdi_vpid_insert</code> module's <code>out_mode</code> port is properly encoded to drive the mode input port of the <code>triple_sdi_tx_output_20b</code> module. <ul style="list-style-type: none"> 00 = HD-SDI, 3G-SDI level A, and dual-link HD-SDI 01 = SD-SDI 10 = 3G-SDI level B 11 = Invalid
ds1a	In	10	This input is driven by the <code>ds1a_out</code> port of the <code>triple_sdi_vpid_insert</code> module. It carries the multiplexed Y/C data stream for SD-SDI, the Y data stream for HD-SDI, dual-link HD-SDI, and 3G-SDI level A, and the Y data stream of link A for 3G-SDI level B.
ds2a	In	10	This input is driven by the <code>ds2a_out</code> port of the <code>triple_sdi_vpid_insert</code> module. It carries the C data stream for HD-SDI, dual-link HD-SDI, 3G-SDI level A, and the C data stream of link A for 3G-SDI level B.
ds1b	In	10	This input is driven by the <code>ds1b_out</code> port of the <code>triple_sdi_vpid_insert</code> module. It carries the Y data stream of link B for 3G-SDI level B.
ds2b	In	10	This input is driven by the <code>ds2b_out</code> port of the <code>triple_sdi_vpid_insert</code> module. It carries the C data stream of link B for 3G-SDI level B.
insert_crc	In	1	When this input is High, CRC values are calculated and inserted into all data streams when running in HD-SDI and 3G-SDI modes. This input is ignored when running in SD-SDI mode.
insert_ln	In	1	When this input is High, LN (line number) words are inserted after the EAVs in all data streams when running in HD-SDI and 3G-SDI modes. This input is ignored when running in SD-SDI mode.
insert_edh	In	1	When this input is High, EDH packets are generated and inserted into the SD-SDI data stream. This input is ignored when running in HD-SDI and 3G-SDI modes.

Table 8-6: Ports of `triple_sdi_tx_output_20b` Module (Cont'd)

Port Name	I/O	Width	Description
ln_a	In	1	This is the line number input. This is required for HD-SDI and 3G-SDI when insert_ln is High. The line number input must be stable by the XYZ word of the EAV and must remain stable for at least two sample times. For HD-SDI, dual-link HD-SDI, and level A 3G-SDI, this is the only line number input port used. For level B 3G-SDI, a second line number port, ln_b, is provided. This input port is not used for SD-SDI.
ln_b	In	11	When transmitting level B 3G-SDI and the insert_ln signal is High, the line number on this port is inserted into the Y and C channels of link B. This allows the two independent HD-SDI signals carried by level B to be vertically unsynchronized. Because ln_b is always used in 3G-SDI level B mode, when the input data streams are SMPTE 372M dual-link HD-SDI streams, ln_b must be equal to ln_a. This input port is not used for SD-SDI, HD-SDI, or 3G-SDI level A.
eav	In	1	This input must be High when the XYZ word of each EAV enters the module on the data stream inputs. This input is not required for SD-SDI. This port is typically driven by the eav_out port of the <code>triple_sdi_vpid_insert</code> module.
sav	In	1	This input must be High when the XYZ word of each SAV enters the module on the data stream inputs. This input is not required for SD-SDI. This port is typically driven by the sav_out port of the <code>triple_sdi_vpid_insert</code> module.
Outputs			
txdata	Out	20	The scrambled SDI data stream is output on this port. The data should be directly connected to the TXDATA port of the GTX wrapper module. For SD-SDI, this data is 11X oversampled data running at 148.5 MHz. For HD-SDI, the data rate is 74.25 MHz. For 3G-SDI, the data rate is 148.5 MHz.
ce_align_err	Out	1	The SD-SDI bit replication logic requires that the ce signal have an exact 5/6/5/6 clock cycle cadence. If the ce cadence ever varies from this cadence, the bit replication logic underflows or overflows and the ce_align_err output is asserted High until the cadence returns to normal.

Triple-Rate SDI TX Modes

The SDI mode (SD-SDI, HD-SDI, 3G-SDI level A or level B) of the triple-rate SDI transmitter is determined by the mode and level inputs to the transmitter datapath modules. The clock frequency requirements and the number of data streams expected by the triple-rate SDI transmitter datapath depend on the SDI mode.

Triple-Rate SDI TX SD-SDI Operation

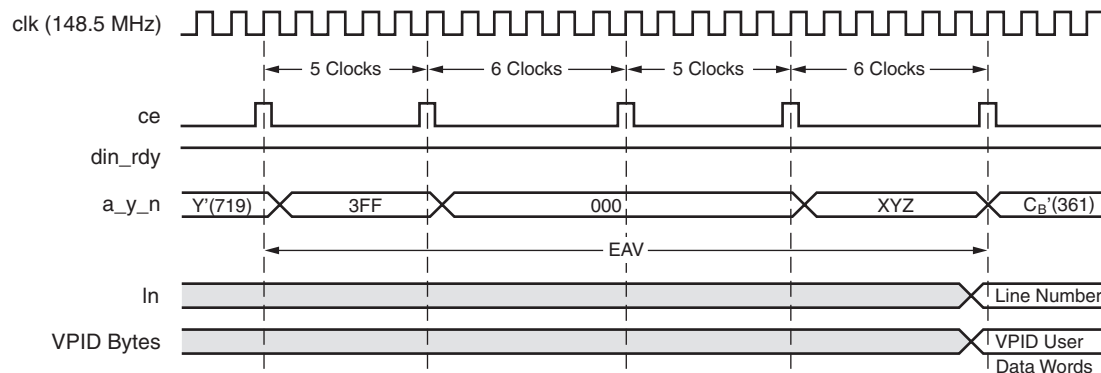
When running in SD-SDI mode, the TXOUTCLK from the GTX TX has a frequency of 148.5 MHz. This is because the transmitter runs at 11 times the 270 Mb/s bit rate ($270 \text{ Mb/s} \times 11/20b = 148.5 \text{ MHz}$). The interleaved Y/C data stream is connected to the

a_y_in port of the triple_sdi_vpid_insert module. The clock enable input to both modules (triple_sdi_vpid_insert and triple_sdi_tx_output_20b) must be asserted at a 27 MHz rate, as shown in Figure 8-7. This means that it must be asserted with a 5/6/5/6 clock cycle cadence. Any other cadence of the ce signal causes the SD-SDI bit replication logic to underflow or overflow. The din_rdy ports of both modules must always be held High in SD-SDI mode.

The triple_sdi_vpid_insert module optionally inserts SMPTE 352M VPID packets and outputs the modified data stream to the triple_sdi_tx_output_20b module on the dsla connection.

If insert_edh is High, the triple_sdi_tx_output_20b module inserts EDH packets, scrambles the data, and replicates each scrambled bit 11 times. The scrambled and replicated data is output on the txdata port at 148.5 MHz. The GTX transmitter serializes the data for transmissions over the serial interface.

Line number values input on the ln_a port of the triple_sdi_vpid_insert module are only required if SMPTE 352M packets are inserted in SD-SDI mode. Line number values are not required to be input on the ln_a port of the triple_sdi_tx_output_20b module in SD-SDI mode.



Notes:

The clk, ce, din_rdy, and ln inputs are connected to the corresponding port of both the triple_sdi_vpid_insert and triple_sdi_output_20b modules. All inputs except ce have an entire sample time in which to become stable. Data is only sampled on the inputs on the rising edge of clk when ce is High. The sample time is the time between rising edges of clk when ce is High. As shown in the figure, the ce signal must always have a 5/6/5/6 clock cycle cadence. Any other cadence under- or overflows the SDI bit-replication logic. Even though the figure shows that the first word of the EAV is a 5 clock-cycle sample, there is no such relationship required. The first word of the EAV could, instead, be a 6 clock sample.

In SD-SDI mode, the line number value on the ln input port is only used by the triple_sdi_vpid_insert module. It must be stable starting with the XYZ word of the EAV and must remain valid through the entire HANC interval.

The VPID input bytes on the input of the triple_sdi_vpid_insert module must be stable beginning with the XYZ word of the EAV and must remain stable during the entire HANC interval on lines where SMPTE 352M packets are inserted.

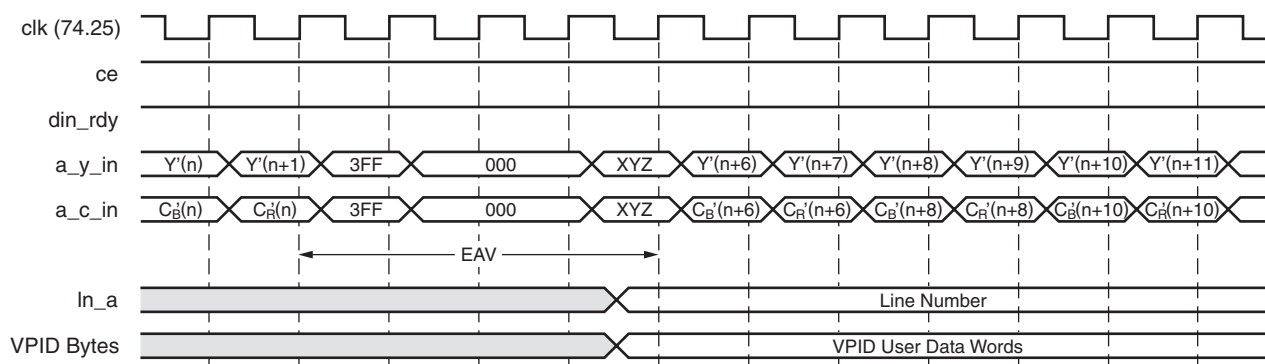
X1014_C8_08_081109

Figure 8-7: SD-SDI Input Timing for GTX Triple-Rate SDI TX

Triple-Rate SDI TX HD-SDI Operation

When running in HD-SDI mode, the frequency of the TXOUTCLK from the GTX TX is 74.25 MHz (or 74.25/1.001 MHz). The clock enable input to the datapath modules must be

High. Figure 8-8 shows the timing of the input signals for HD-SDI mode. The `din_rdy` ports of both datapath modules must always be held High in HD-SDI mode.



Notes:

In HD-SDI mode, the line number value on the `In_a` input port is used by both the `triple_sdi_vpid_insert` and the `triple_sdi_tx_output_20b` modules and must be stable starting with the XYZ word of the EAV and must remain valid through the entire HANC interval.

The VPID input bytes on the input of the `triple_sdi_vpid_insert` module must be stable beginning with the XYZ word of the EAV and must remain stable during the entire HANC interval on lines where SMPTE 352M packets are inserted.

X1014_C8_09_081109

Figure 8-8: HD-SDI Input Timing for Triple-Rate SDI TX

Data enters the `triple_sdi_vpid_insert` module as two 10-bit data streams, with the Y data stream on the `a_y_in` port and the C data stream on the `a_c_in` port. The input data rate is 74.25 MHz. The `triple_sdi_vpid_insert` module optionally inserts SMPTE 352M VPID packets into the Y data stream before outputting the two data streams on the `ds1a` and `ds2a` ports to the `triple_sdi_tx_output_20b` module.

If `insert_ln` is High, the `triple_sdi_tx_output_20b` module inserts line numbers into both data streams immediately after the EAV. If `insert_crc` is High, it calculates and inserts CRC values immediately after the line numbers. The `triple_sdi_tx_output_20b` module scrambles the data streams and outputs one 20-bit data stream running at 74.25 MHz on the `txdata` output port. The GTX transmitter serializes the data for transmissions over the serial interface.

Line numbers must be supplied on the `In_a` port of the `triple_sdi_vpid_insert` module only if SMPTE 352M packet insertion is enabled in HD-SDI mode. Line numbers must be supplied on the `In_a` port of the `triple_sdi_tx_output_20b` module only if line number insertion is enabled.

Triple-Rate SDI TX 3G-SDI Operation

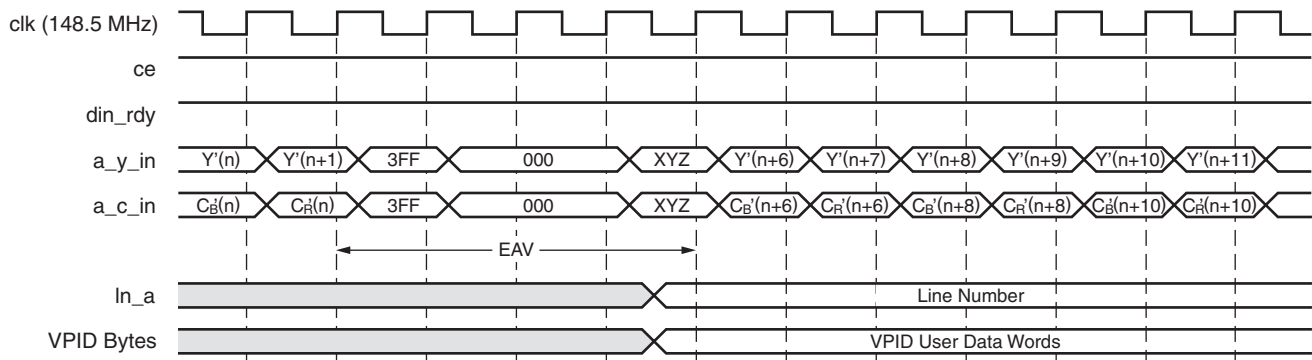
When running in 3G-SDI mode, the datapath modules can run in either level A or level B mode, as selected by the `triple_sdi_vpid_insert` module's level input (level A = Low, level B = High). The `triple_sdi_vpid_insert` module communicates to the `triple_sdi_tx_output_20b` module which level is selected through the `out_mode` port.

In 3G-SDI level A mode, the `triple_sdi_vpid_insert` module accepts two 10-bit data streams on its `a_y_in` (data stream 1) and `a_c_in` (data stream 2) input ports. The clock frequency is 148.5 MHz (or 148.5/1.001 MHz). The clock enable and `din_rdy` inputs to the

datapath modules must be High. Figure 8-9 shows the timing of the input signals for 3G-SDI level A mode.

Before outputting the data streams on the ds1a and ds2a output ports to the triple_sdi_tx_output_20b module, the triple_sdi_vpid_insert module inserts SMPTE 352M packets into both data streams. If insert_ln is High, the triple_sdi_tx_output_20b module inserts line numbers into both data streams immediately after the EAV. If insert_crc is High, it calculates and inserts CRC values immediately after the line numbers. The triple_sdi_tx_output_20b module scrambles the data streams and outputs one 20-bit data stream running at 148.5 MHz on the txdata output port. The GTX transmitter serializes the data for transmission over the serial interface.

Line numbers are required on the ln_a port of the triple_sdi_vpid_insert module only if SMPTE 352M packet insertion is enabled. However, SMPTE 352M packets are required in 3G-SDI mode. Line numbers are required on the ln_a port of the triple_sdi_tx_output_20b module if line number insertion is enabled.



Notes:

In 3G-SDI level A mode, the line number value on the ln_a input port is used by both the triple_sdi_vpid_insert and the triple_sdi_tx_output_20b modules. It must be stable starting with the XYZ word of the EAV and must remain valid through the entire HANC interval.

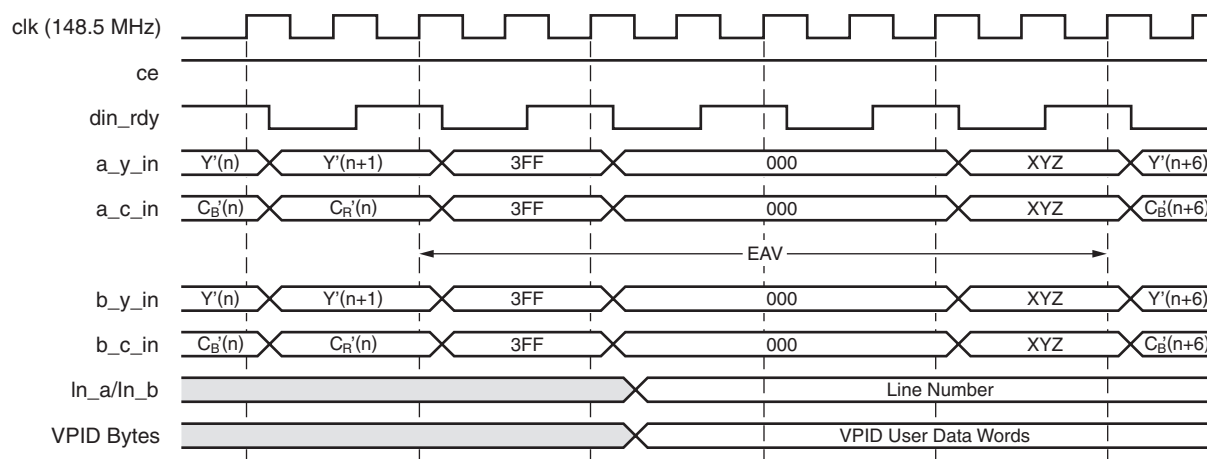
The VPID input bytes on the input of the triple_sdi_vpid_insert module must be stable beginning with the XYZ word of the EAV. They must remain stable during the entire HANC interval on lines where SMPTE 352M packets are inserted.

X1014_C8_10_081109

Figure 8-9: 3G-SDI Level A Input Timing for Triple-Rate SDI TX

In SG-SDI level B mode, the triple_sdi_vpid_insert module requires four 10-bit data streams on its a_y_in (Y channel of link A), a_c_in (C channel of link A), b_y_in (Y channel of link B), and b_c_in (C channel of link B). These four data streams can be any SMPTE 372M dual-link HD-SDI signal pre-formatted according to the SMPTE 372M specification. They can also be two independent HD-SDI streams that are combined onto a single 3G-SDI level B link. The frequency of the GTX transmitter TXOUTCLK is 148.5 MHz (or 148.5/1.001 MHz). Thus, the datapath modules must be clocked every other clock cycle in 3G-SDI level B mode, as shown in Figure 8-10. For the triple_sdi_vpid_insert module, either the clock enable or the din_rdy signal can be used to enable this module every other clock cycle. However, for the triple_sdi_output_20b module, the clock enable cannot be used and din_rdy must be used. This is because the clock enable controls the entire module, including the back end, which must run at 148.5 MHz. The din_rdy however, controls only the input timing to the module (with the exception of SD-SDI mode, where ce must be used to provide a 27 MHz data rate to the

`triple_sdi_tx_output_20b` module and `din_rdy` must be High). Thus, in 3G-SDI level B mode, the normal way to control these modules is to keep the clock enable High and toggle the `din_rdy` signal every other clock cycle, as shown in Figure 8-10.



Notes:

In 3G-SDI mode, the line number value on the `ln_a` and `ln_b` input port is used by both the `triple_sdi_vpid_insert` and the `triple_sdi_tx_output_20b` modules. It must be stable starting with the XYZ word of the EAV and must remain valid through the entire HANC interval.

The VPID input bytes on the input of the `triple_sdi_vpid_insert` module must be stable beginning with the XYZ word of the EAV. They must remain stable during the entire HANC interval on lines where SMPTE 352M packets are inserted.

X1014_C8_11_081109

Figure 8-10: 3G-SDI Level B Input Timing for Triple-Rate SDI TX

The `triple_sdi_vpid_insert` module inserts SMPTE 352M VPID packets in the Y data streams of both link A and link B, as required by the SMPTE 425M-2008 specification. The four data streams are output to the `triple_sdi_tx_output_20b` module on the `ds1a`, `ds2a`, `ds1b`, and `ds2b` ports.

If `insert_ln` is High, the `triple_sdi_tx_output_20b` module inserts line numbers into all four data streams. If `insert_crc` is High, it calculates and inserts CRC values immediately after the line numbers in all four data streams. The data streams are then interleaved to produce a single 20-bit data stream running at 148.5 MHz on the `txdata` output port. The GTX transmitter serializes the data for transmissions over the serial interface.

If SMPTE 352M packets are to be inserted, line numbers are required on both the `ln_a` (for link A) and `ln_b` (for link B) input ports of the `triple_sdi_vpid_insert` module. Also, SMPTE 352M packets are required for 3G-SDI. Line numbers are required for the `ln_a` and `ln_b` ports of the `triple_sdi_tx_output_20b` module when line number insertion is enabled. Two different line number ports are provided for the case where two independent HD-SDI signals are to be carried by 3G-SDI level B. When dual-link HD-SDI is being carried by 3G-SDI level B, `ln_a` and `ln_b` must be driven with identical line numbers.

Triple-Rate SDI TX Dual-Link HD-SDI Operation

The triple-rate SDI transmitter does not contain the formatting logic to convert various video formats into SMPTE 372M dual-link HD-SDI streams. However, preformatted dual-link HD-SDI streams can be transmitted by a pair of light triple-rate SDI transmitters.

There are several ways to build a dual-link HD-SDI transmitter using the modules that make up the light triple-rate SDI transmitter. The first method is best suited for applications where the two transmitters are always paired as a dual-link HD-SDI pair. In this method, shown in [Figure 8-11](#), the link A transmitter can be used in any SDI mode, but the link B transmitter is only used in dual-link HD-SDI mode. A single `triple_sdi_vpid_insert` module inserts SMPTE 352M packets into the Y stream of both HD-SDI links of the dual-link pair. These four data streams out of the VPID inserter are connected to two `triple_sdi_tx_output_20b` modules with the A data streams going to one output module and the B data streams going to the other, as shown in [Figure 8-11](#). The B data streams also go to the link A output module because this is required for 3G-SDI level B operation. Because this method is best suited for applications where the link B transmitter is only used in dual-link HD-SDI mode, both transmitters are shown in the same GTX_DUAL tile. However, it is also possible to use this method with GTX transmitters in separate tiles.

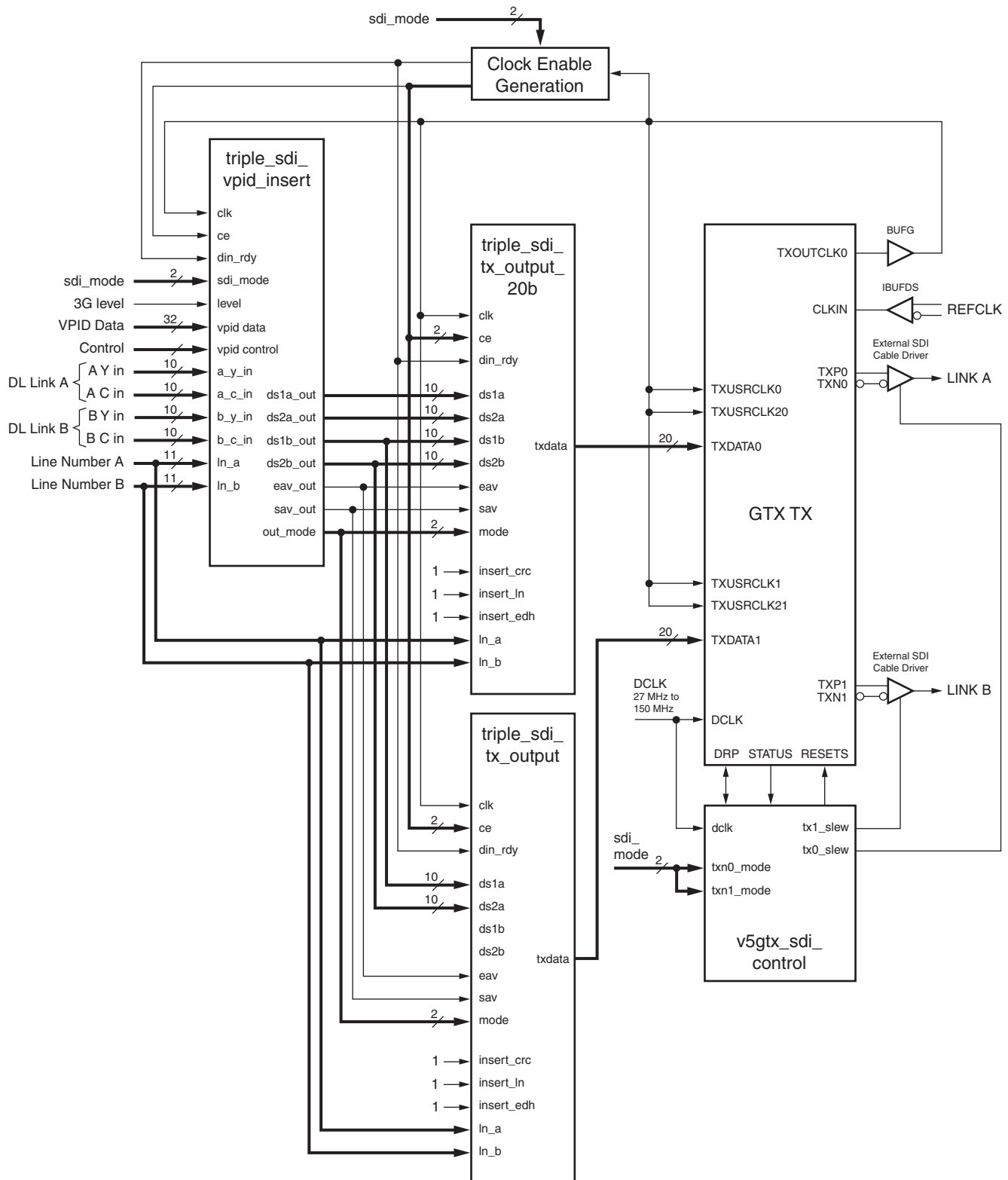
The second method uses two independent triple-rate SDI transmitters, each with its own `triple_sdi_vpid_insert` module. In dual-link HD-SDI mode only, these two transmitters are paired to form a dual-link HD-SDI pair. In dual-link HD-SDI mode, each transmitter runs as if it were a normal HD-SDI transmitter with one transmitter handling link A and the other handling link B. This method is shown in [Figure 8-12](#).

There are several ways to handle clocking with the method shown in [Figure 8-12](#). The two transmitters must be driven by the same reference clock frequency because they must run at exactly the same bit rate. This can be done either by providing the same reference clock to the external reference clock inputs of these two GTX_DUAL tiles or by routing one reference clock between the two GTX_DUAL tiles using the dedicated GTX transmitter clock north/south routing resources. Because the two GTX transmitters are driven by the same reference clock, their TXOUTCLKs are frequency locked, although not necessarily in phase. Therefore, the link A data streams must be synchronized to the `tx_usrclk` of link A and the link B data streams must be synchronized to the `tx_usrclk` of link B. This synchronization must not introduce excessive skew between the two links. Otherwise, the maximum link skew between the two transmitters, as specified by SMPTE 372M, could be exceeded.

Alternatively, one of the transmitters could drive its `tx_usrclk` with a BUFGMUX so that, in dual-link mode, its `tx_usrclk` is derived from the TXOUTCLK of the other GTX TX. This is shown in [Figure 8-12](#), where `txB_usrclk` is driven by a BUFGMUX. The advantage of this clock scheme is that no synchronization of the data streams is needed because the data streams enter the two triple-rate SDI transmitters if the data stream source is driven by either one of the global `tx_usrclk` signals. The reference clock from the link A GTX_DUAL tile is routed through the GTX transmitter north/south clock routing structure to the other GTX_DUAL tile. The link B GTX TX can then use either the external reference clock connected to the FPGA at that tile or the reference clock from the link A GTX_DUAL tile. The reference clock used by the link B GTX TX is dynamically selected by the `v5gtx_sdi_control` module through the DRP port under control of the `dl_mode` signal (dual-link mode) that is connected to the `v5gtx_sdi_control` module's `refclk_sel` port. Another consideration for this method is that the link B clock enables should be in phase with the link A clock enables when running in dual-link HD-SDI mode.

Regardless of which method is used to construct the dual-link HD-SDI transmitter, the input timing is identical to normal HD-SDI mode, as shown in [Figure 8-8](#). Also, the `ln_b` line number port of the `triple_sdi_vpid_insert` module(s) is ignored in dual-link HD-SDI mode. The `ln_b` port is only used in 3G-SDI level B mode.

As required by the SMPTE 372M standard, the two HD-SDI links must be identified using bit 6 of byte 4 of the SMPTE 352M packets. This bit must be 0 in link A and 1 in link B. If a single VPID insert module is used, as shown in [Figure 8-11](#), the `triple_sdi_vpid_insert` has separate input ports for byte 4 for the two links, just for this purpose.



X1014_C8_12_081109

Figure 8-11: Example Dual-Link HD-SDI TX (Transmitters Always Paired)

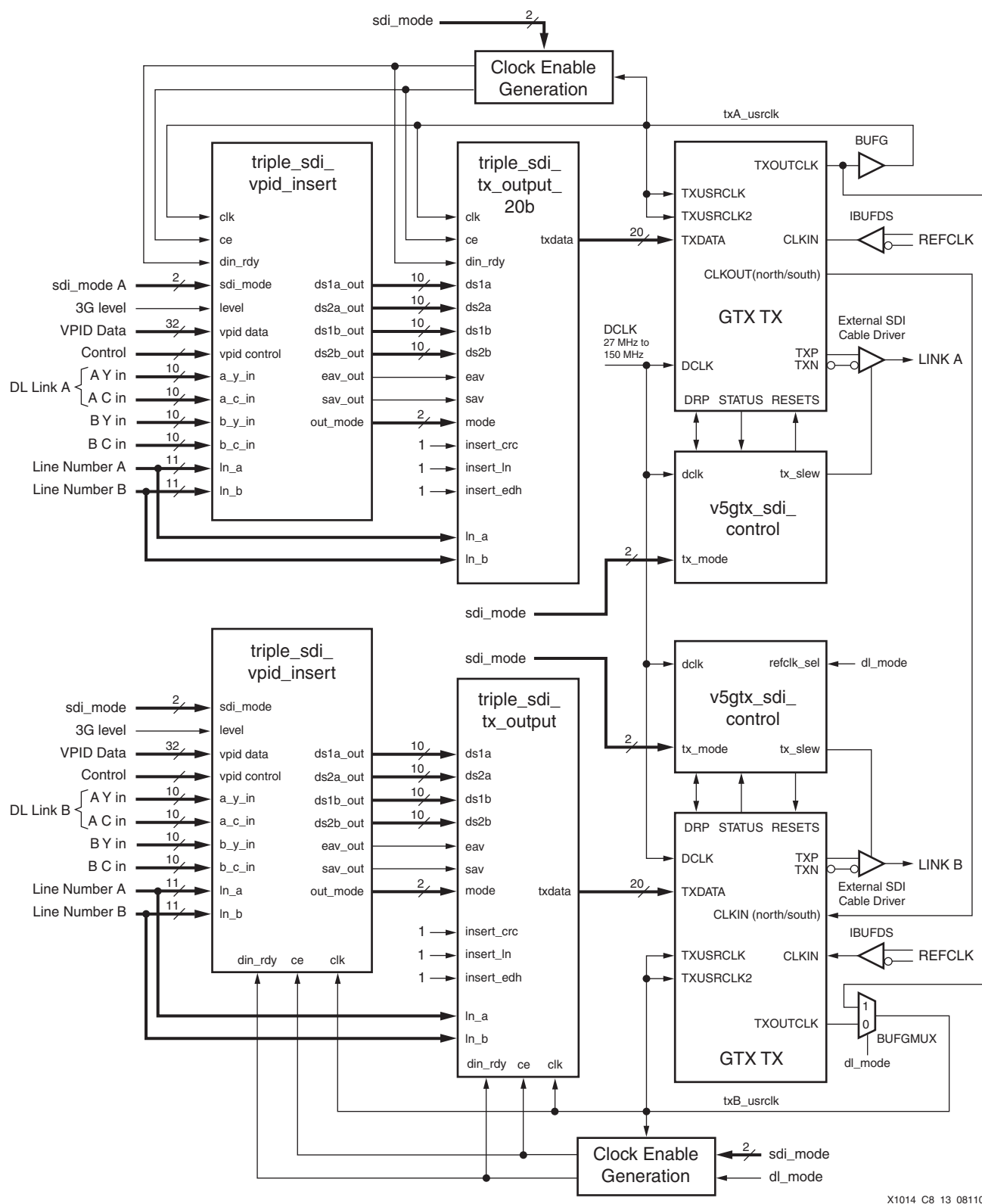


Figure 8-12: Example Dual-Link HD-SDI TX (Transmitters Can Operate Independently)

Summary of Triple-Rate TX Modes

Table 8-7 summarizes the input data rates and connections for all SDI modes.

Table 8-7: Triple-Rate SDI TX Modes

SDI Mode	Level	TXOUTCLK	ce	din_rdy	input data rate	a_y_in	a_c_in	b_y_in	b_c_in
HD-SDI and Dual Link HD-SDI	X	74.25 MHz or 74.25/1.00 MHz	High	High	74.25 MHz or 74.25/1.001 MHz	Y	C		
SD-SDI	X	148.5 MHz	5/6/5/6 Cadence	High	27 MHz	Y/C			
3G-SDI A	0	148.5 MHz or 148.5/1.00 MHz	High	High	148.5 MHz or 148.5/1.001 MHz	Data stream 1	Data stream 2		
3G-SDI B	1	148.5 MHz or 148.5/1.001 MHz	High	Asserted every other clock cycle	74.25 MHz or 74.25/1.001 MHz	link A Y	link A C	link B Y	link B C

Triple-Rate TX Details

This section provides some details about the operation of the GTX triple-rate SDI transmitter reference design.

SMPTE 352M Packet Insertion

The `triple_sdi_vpid_insert` module can insert SMPTE 352M VPID packets into SD-SDI, HD-SDI, dual-link HD-SDI, and 3G-SDI (both level A and level B) streams. The dual-link HD-SDI and 3G-SDI standards require SMPTE 352M packets in the data streams. The packets are optional in SD-SDI and HD-SDI.

The `triple_sdi_vpid_insert` module is a wrapper around a pair of `SMPTE352_vpid_insert` modules from the SMPTE 352M reference design. The behavior of this module is the same as that of the underlying `SMPTE352_vpid_insert` module. This module is described further in [Chapter 26, SMPTE 352M Video Payload Identification Packet Processing](#). Chapter 26 also describes the required values for the user data words of the SMPTE 352M packets.

SMPTE 352M packets are inserted on one designated video line in each frame if the transport is progressive, and one designated video line in each field if the transport is interlaced. The designated video lines that carry SMPTE 352M packets vary depending on the SDI mode and the video format. They are detailed in [Chapter 26](#). The `line_f1` input port determines the line in the progressive frame or in the first interlaced field where the module inserts a SMPTE 352M packet. The `line_f2` input port determines the line in the second interlaced field where the module inserts the SMPTE 352M packet. The `line_f2_en` input determines whether packets are inserted in the line specified by `line_f2`. This input must be Low for a progressive transport and High for an interlaced transport.

It is very important that the `line_f2_en` input to the `triple_sdi_vpid_insert` module be controlled correctly. This input must be High for interlaced video and Low for progressive video. However, this input must actually be controlled based on whether the transport is interlaced or progressive, not whether the picture is interlaced or progressive. The 1080p 50 Hz and 60 Hz 4:2:2 10-bit progressive video formats, when carried by dual-link HD-SDI or 3G-SDI level B (but not 3G-SDI level A), are transported in an

interlaced manner. The same is true for progressive segmented frame transport. It is essential that the SMPTE 352M packets be inserted into both fields of the transport data streams for these formats. Some receiving equipment fails to lock properly if the SMPTE 352M packets are only present in one field rather than in both fields of interlaced transports.

Dual-link HD-SDI data streams output from a dual-link SDI receiver might already have SMPTE 352M packets in the Y data streams of each link. This is because SMPTE 372M mandates these packets, although not all equipment on the market properly inserts these required packets. If the packets are present, the first data word should be 87 hex, indicating that the data streams are carried on a dual-link HD-SDI interface. When sending this dual-link data on a 3G-SDI level B interface, the first word must be replaced with a value of 8A hex, indicating an SMPTE 372M signal carried on a 3G-SDI level B interface. Thus, the SMPTE 352M packets must be modified when sending the dual-link HD-SDI streams on a 3G-SDI level B interface. Only the first byte of the VPID data must be modified. The values of the other bytes do not need to change, but must be captured first and applied to the VPID byte inputs of the `triple_sdi_vpid_insert` module so they can be reinserted when the packet is overwritten by the `triple_sdi_vpid_insert` module.

In 3G-SDI level B mode, the two independent HD-SDI signals carried by level B are allowed to be vertically unsynchronized. In this case, SMPTE 352M packets are inserted independently on the two HD-SDI signals carried by the 3G-SDI level B interface. The second line number input port on the `triple_sdi_vpid_insert` module allows two separate line numbers to be provided for level B, one for each HD-SDI signal. However, because there is only one set of VPID data inputs to the insertion module, the SMPTE 352M packets, with the exception of byte 4, are identical. This means that the two HD-SDI streams must be carrying identical video formats. This is normally the case due to restrictions in 3G-SDI level B requiring the two HD-SDI streams be of the same format. If an application requires insertion of different VPID packets into the two HD-SDI signals carried by a 3G-SDI level B signal, the `triple_sdi_vpid_insert` module can be modified to provide two completely independent sets of VPID input data ports.

Line Number Insertion

3G-SDI and HD-SDI both require that line numbers be present in the two words that follow each EAV. If the `insert_in` input is High, the `triple_sdi_tx_output_20b` module inserts those line numbers appropriately for HD-SDI and both levels of 3G-SDI (inserting them into all four data streams for level B 3G-SDI). The line numbers to be inserted are provided to the `triple_sdi_tx_output_20b` module on the `ln_a` and `ln_b` inputs. In some cases, it might not be necessary or desirable to overwrite line numbers that are already present in the data streams. In that case, the `insert_in` input can be driven Low and no line numbers are inserted. If the `insert_in` input is hard-wired Low in the source code, the line number insertion logic is optimized out of the design by the synthesis tool.

For dual-link HD-SDI data streams carried either by dual-link HD-SDI or 3G-SDI level B, if the video format is 1080p 50 Hz, 59.94 Hz, or 60 Hz, the line numbers are required to be transport line numbers, not video line numbers. The dual-link HD-SDI formatting module supplied by Xilinx for 1080p 50 Hz, 59.94 Hz, and 60 Hz video creates the transport line numbers and inserts them into the data streams. Thus, when this module is used and the video format is 1080p 50 Hz, 59.94 Hz, and 60 Hz (transported via dual-link HD-SDI or 3G-SDI level B), line number insertion by the `triple_sdi_tx_output_20b` module must be disabled.

For HD-SDI, dual-link HD-SDI, and 3G-SDI level A, only the line number on `ln_a` is used. For 3G-SDI level B, the line number on `ln_a` is inserted into the Y and C data streams of link

A and the line number on `ln_b` is inserted into the Y and C data streams of link B (again, only if `insert_ln` is asserted). For dual-link HD-SDI, two `triple_sdi_tx_output_20b` modules are used, one for each link. Each `triple_sdi_tx_output_20b` module processes its HD-SDI data streams through the `ds1a` and `ds2a` import ports, just like normal HD-SDI mode. Thus, the `ln_b` input port is not used in dual-link HD-SDI mode.

The SMPTE 425M specification does not specifically state that the two HD-SDI signals carried in level B mode have to be vertically synchronized (except for when SMPTE 372M dual-link HD-SDI is being carried). SMPTE 425M does not expect the case where the two HD-SDI signals are vertically unsynchronized, but has stated that this is a legal case. The transmitter reference design allows this case by providing the second (`ln_b`) line number input port.

CRC Generation and Insertion

3G-SDI and HD-SDI both require that CRC values be present in the two words that follow the line numbers after the EAV. If the `insert_crc` input is High, the `triple_sdi_tx_output_20b` module calculates and inserts CRC values for each line for HD-SDI and both levels of 3G-SDI (inserting them into all four data streams for level B 3G-SDI). In some cases, it may not be necessary or desirable to overwrite the CRC values already present in the data stream. In that case, the `insert_crc` input can be driven Low and no CRC values are inserted. If the `insert_crc` input is hard-wired Low, the CRC generation and insertion logic is optimized out of the design by the synthesis tool.

EDH Generation and Insertion

EDH packets are optional (but usually present) in SD-SDI and are never used for HD-SDI and 3G-SDI. The EDH packets contain CRC values that can be used to detect errors in the SD-SDI data stream. When running in SD-SDI mode, the `triple_sdi_tx_output_20b` module generates and insert EDH packets when `insert_edh` is High. If EDH is hard-wired Low, the synthesis tool optimizes the EDH generator out of the design. The EDH processor used in this reference design is found in the “SD-SDI Ancillary Data and EDH Processors” chapter of *Audio/Video Connectivity Solutions for Virtex-II Pro and Virtex-4 FPGAs* [Ref 5].

Ancillary Data Insertion

Ancillary data packets can be inserted prior to the `triple_sdi_vpid_insert` module or between the `triple_sdi_vpid_insert` module and the `triple_sdi_tx_output_20b` module. Because the SMPTE 352M packets inserted by the `triple_sdi_vpid_insert` module should, according to the SMPTE 352M standard, be placed at the beginning of the HANC space (immediately after the CRC words following the EAV), if ancillary data packets get inserted on the same lines as the SMPTE 352M packets, it would be better to insert the ancillary data packets after the `triple_sdi_vpid_insert` module.

The `triple_sdi_tx_output_20b` module inserts EDH packets in SD-SDI mode, if enabled. Any ancillary data packets that conflict with the standard EDH packet locations are partially or completely overwritten when the EDH packets are inserted.

FPGA Resource Usage

[Table 8-8](#) shows the FPGA resources required for each of the triple-rate SDI interface reference designs described in this chapter. The resource usage includes all the modules required to implement the interface, including the `v5gtx_sdi_control` module. The resources reported are based on using one RX unit or one TX unit in a GTX_DUAL tile. Some savings over the listed resource usage are achieved when multiple units in a GTX_DUAL tile are used. This is because logic in the `v5gtx_sdi_control` module can be shared.

The GTX triple-rate SDI RX module does not include an EDH processor for SD-SDI. [Table 8-8](#) shows the size of the receiver both with and without an EDH processor. The GTX SDI TX does include an EDH processor. It can be optimized out of the design by connecting the `insert_edh` input port of the `triple_sdi_tx_output_20b` module Low. [Table 8-8](#) shows the size of the transmitter with and without the EDH processor included. In both the RX and TX cases, the EDH processor used is from the “SD-SDI Ancillary Data and EDH Processors” chapter of *Audio/Video Connectivity Solutions for Virtex-II Pro and Virtex-4 FPGAs* [Ref 5].

The results shown were achieved using the ISE® Design Suite 11.1, and XST and MAP set to optimize for speed. The XST “Safe Implementation” property was set to **yes**.

The RX and TX reference designs do not use any DCMs or PLLs from the clock-management tile. They do not require any block RAMs.

Table 8-8: Triple-Rate SDI Interface FPGA Resource Usage

Reference Design	Flip-Flops	LUTs
GTX Triple-Rate RX (no EDH processor)	1,412	1,452
GTX Triple-Rate RX + EDH processor	1,793	2,057
GTX Triple-Rate TX (with EDH processor)	924	1,327
GTX Triple-Rate TX (EDH processor optimized away)	558	744

Timing

Because of the use of 20-bit RX and TX datapaths, the maximum clock frequency used in these designs is 148.5 MHz. It should not be difficult to meet timing in the slowest speed grade devices.

The RXRECCLK generated by the GTX transceiver can, under some conditions, have erratic timing when the input SDI bitstream stops and restarts or when the GTX transceiver is switched between SDI modes. These periods of erratic timing can cause problems for sequential control logic such as finite state machines (FSM). Most synthesis tools, by default, optimize away illegal state recovery for FSMs. The erratic timing of the GTX transceiver clocks can cause FSMs to go into illegal states. Thus, it is highly recommended that the synthesis tool be forced to generate “safe” implementations of FSMs that include illegal state recovery.

Reference Design

The reference design for the triple-rate SDI receiver and transmitter is available at <https://secure.xilinx.com/webreg/clickthrough.do?cid=119758>. Open the ZIP archive and extract the file xapp1014_c8_GTX_SDI.zip.

Table 8-9 gives an overview of the reference design presented in this chapter.

Table 8-9: Reference Design Checklist

Reference Design Attributes	
General	
Developer name	Xilinx
Target devices	Virtex-5 FXT and TXT FPGAs, all speed grades
Source code provided	Yes
Source code format	VHDL, Verilog
Design uses code/IP from an existing reference design or application note, third party, CORE Generator™ software	Yes
Simulation	
Functional simulation performed	No
Timing simulation performed	No
Testbench used for functional and timing simulation provided	No
Testbench format	
Simulator software used/version	
SPICE/IBIS simulations	No
Hardware verification	
Hardware verified	Yes
Hardware platform used for verification	ML571 SDV board

Conclusion

This chapter describes the triple-rate SDI receiver and transmitter reference designs for the GTX transceivers found in Virtex-5 FXT and TXT devices. The reference designs are optimized to minimize the number of global clock resources required.

The reference designs support SD-SDI, HD-SDI, 3G-SDI level A and level B, and dual-link HD-SDI. Packing and unpacking of native video to and from SDI data streams is not provided for 3G-SDI and dual-link HD-SDI.

Section III: SD-SDI using Virtex-5 FPGA SelectIO LVDS

***Audio/Video Connectivity Solutions
for Virtex-5 FPGAs***

SD-SDI Receiver

Summary

The standard-definition serial digital interface (SD-SDI) standard describes how to transport standard-definition (SD) digital video serially over coaxial cable. SD-SDI is commonly used to connect SD video equipment in broadcast studios and video production centers. Refer to *Audio/Video Connectivity Solutions for Virtex®-II Pro and Virtex-4 FPGAs* [Ref 5] for detailed information on the related standards for SD-SDI.

The reference design presented in this chapter comprises a complete SD-SDI receiver with error detection and handling (EDH). This implementation uses the low-voltage differential signaling (LVDS) I/O standards from Xilinx® SelectIO™ technology, as opposed to other Xilinx implementations that use the GTP transceiver I/Os. Summaries for each functional block are given in this chapter. The details of each functional block are presented in the individual SD-SDI chapters for the video encoder, video decoder, video flywheel, ancillary data, and EDH processors. Refer to the respective chapters in this application note for further information regarding these functional blocks.

The SD-SDI application example in this chapter is designed specifically for the Xilinx ML571 SDV demonstration board. However, the bulk of the application code is generic and can be easily adapted to customer needs.

LVDS SD-SDI Receiver (SDI Receiver)

This section describes the components that make up the LVDS SD-SDI receiver. Figure 9-1 shows the SDI receiver architecture.

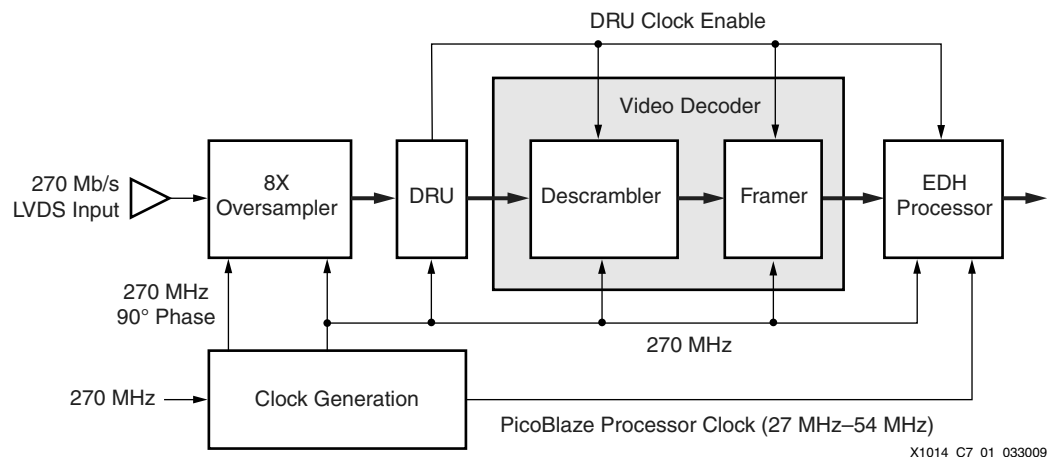


Figure 9-1: SDI Receiver Architecture

8X Oversampler

The oversampler asynchronously samples the incoming serial data and provides multiple sample points to the DRU. An 8X oversampling factor has been shown to be sufficient for providing good receiver jitter tolerance and is practical to implement using the Virtex-5 FPGA. The advanced capabilities of the ILOGIC, IDELAY, and clock management tile (CMT) in the Virtex-5 FPGA make implementing an 8X oversampler efficient and practical.

The oversampler uses a combination of two clock phases, fixed data delays, and a special inverting output of the LVDS input buffer to provide eight sample points per clock cycle. This is accomplished by creating two sampling clocks 90 degrees apart in phase using a DCM or PLL from the CMT. The incoming data is then delayed by two different amounts using the IDELAY, one for the p-side positive data and another for the n-side negative data, resulting in a zero-degree data stream and another data stream delayed by 45 degrees. By sampling these two different data streams on the four clock edges (using the two phase-shifted clocks sampling on both edges), a total of eight sample points per clock cycle is achieved. For details on the oversampler block used in this reference design, refer to *Efficient 8X Oversampling Asynchronous Serial Data Recovery Using IDELAY* [Ref 7].

Data Recovery Unit

The DRU takes the results from the oversampler and uses a sophisticated algorithm to determine how to recover stable data from multiple sample points. The algorithm is contained in a state machine implemented in block RAM within the FPGA. In summary, the DRU examines the sample points from the oversampler and looks for transitions in the data points. These transitions indicate the edges of the data valid window for the received data. After the edges are determined, the DRU selects the sample points that are closest to the center of the data valid window. This combination of 8X oversampling and sample point selection are crucial to providing a robust receiver with good jitter tolerance.

The outputs of the DRU are a 10-bit word that consists of the 10 recovered data bits and a clock enable that is asserted at an average rate of 1/10th of the bit (word) rate. The 10-bit recovered data word is sourced to the video decoder block for descrambling and framing. The clock enable output is used throughout most of the receiver blocks downstream from the DRU because these blocks function at the word rate rather than the bit rate. This clock enable has an average rate of 1/10th of the bit rate (27 MHz in this case), but it should not be directly used as a clock. It is not a periodic signal in the same way that a clock is. Rather, it is a pulse with a width of one sample clock cycle that is asserted, on average, every 10 cycles of the sample clock. In some cases, it might be asserted after 11 or 12 sample clock cycles. This variation is normal and is a direct result of the asynchronous nature of the input data relative to the sample clock and the DRU algorithm.

Video Decoder

Prior to processing, video data must be encoded in accordance with the SDI standard. This encoding process ensures that the data stream is polarity free, and that the receiver can lock onto and extract a clock and data. The data stream is encoded in such a way that sufficient level transitions are present to allow a PLL or other circuit to lock onto the signal and extract a clock, if desired.

After oversampling and recovering the data from the encoded serial stream, the receiver must reverse the encoding process to recover the original raw video data and timing signals. The descrambler provides this function by reversing the original encoder polynomial process to decode the original data. For details on the descrambler

implementation, refer to the “SD-SDI Video Decoder” chapter of *Audio/Video Connectivity Solutions for Virtex-II Pro and Virtex-4 FPGAs* [Ref 5].

After the data has been descrambled, it must be framed to align on the correct 10-bit or 8-bit word boundaries. The framer provides this function. For this framing process to work, a unique, repeating pattern called a timing reference symbol (TRS) is provided that is guaranteed not to occur in the data itself. The framer looks for this pattern and rotates the 10-bit parallel word from the DRU to align the 10-bit words correctly. Refer to the aforementioned “SD-SDI Video Decoder” chapter of *Audio/Video Connectivity Solutions for Virtex-II Pro and Virtex-4 FPGAs* for details on the framer implementation.

Error Detection and Handling Processor

The Error Detection and Handling (EDH) processor provides a simple error detection mechanism suitable for a basic SDI receiver. The EDH protocol is an optional but commonly used addition to the SD-SDI standard. This protocol allows an SD-SDI receiver to verify that each field of video is received correctly. The EDH standard provides only error detection, not correction.

The EDH processor provides several functions in addition to error detection and handling, such as TRS detection, decoding of the field, and vertical and horizontal timing signals. This feature is useful for genlock applications in which the user might wish to use an external device to create a new pixel clock based on the decoded timing signals. The EDH processor also detects and identifies the incoming video standard for the six supported SD standards. Lastly, it can identify when the current video position is in the synchronous switching interval.

While an EDH processor can be implemented in pure gates, the EDH protocol requires that such an implementation be an inefficient use of FPGA resources. A more efficient approach uses a simple microprocessor implemented in the FPGA logic. The PicoBlaze™ processor provides the perfect vehicle to reduce the design size while providing the needed functionality. For details on this EDH processor implementation, refer to the “Reducing the Size of SD-SDI EDH Processing Using the PicoBlaze Processor” chapter of *Audio/Video Connectivity Solutions for Virtex-II Pro and Virtex-4 FPGAs*.

Clocking

The receiver requires a minimum of two clocks: a 270 MHz, zero-degree phase and a 270 MHz, 90-degree phase. These two clocks must be related by a 90-degree phase shift. They are used by the 8X oversampler to create the necessary sample points, as described in [8X Oversampler, page 284](#). The 270 MHz, zero-degree phase clock is used for the remainder of the receiver blocks, including the DRU and all downstream logic (decoder, framer, etc.). The DRU generates a clock enable equivalent to a word rate that is 1/10th the bit rate or 27 MHz. This clock enable is used by the decoder, framer, and parts of the EDH processor such that these blocks run at the 27 MHz word rate rather than the 270 MHz bit rate, which saves one clock buffer.

The EDH processor also requires a clock in the range of 27 MHz to 54 MHz for the PicoBlaze processor. Because the PicoBlaze processor does not support a clock enable, the 270 MHz clock and DRU clock enable cannot be used. The PicoBlaze processor can, but need not be, synchronous to the 270 MHz video clock. Complete synchronization is provided where these two clock domains are crossed. The speed of the PicoBlaze processor clock is somewhat dependent on the video standards being processed. If only 4:2:2 SD video is being handled, 27 MHz is sufficient. In the worst case, for 4:4:4 SD video, a clock frequency of 54 MHz should be used.

Module I/O

Table 9-1 describes the input and output ports of the receiver module.

Table 9-1: SDI Receiver Ports

Port	I/O	Width	Description
vidclk	In	1	This is a video bit-rate clock input of 270 MHz.
vidclk_90	In	1	This is a video bit-rate clock input of 270 MHz, 90-degree phase shifted from vidclk.
cpucclk	In	1	This is a 27 MHz–54 MHz clock for the PicoBlaze processor in the EDH processor. Need not be synchronous to vidclk.
ce	In	1	This is a clock enable for vidclk domain only. Not applicable to cpucclk.
rst	In	1	This is an asynchronous reset for all logic elements except the PicoBlaze processor.
cpu_rst	In	1	This is a synchronous reset for the PicoBlaze processor only.
PAD_sdi_rxp	In	1 (of pair)	This is the P-side of the LVDS SDI data input. (PAD and buffer are instantiated at a lower level.)
PAD_sdi_rxn	In	1 (of pair)	This is the N-side of the LVDS SDI data input. (PAD and buffer are instantiated at a lower level.)
err_flg_en	In	16	This is an error flag enable for the EDH processor. It enables particular error flags.
en_sync_switch	In	1	This input enables fast synchronizing to the EAV during synchronous switching interval.
hd_sd	Out	1	This output indicates whether the standard is high definition (HD) or SD. For this reference design, only SD is supported.
std	Out	3	This output indicates the current detected standard. It is only valid when std_locked is asserted.
std_locked	Out	1	This output is asserted when the EDH processor standard detector is locked to the incoming standard.
vid_out	Out	10	This output is a decoded and delayed video out. Video out is delayed to match all timing signal outputs.
sync_switch	Out	1	This output indicates that the video stream is currently in the synchronous switching interval.
f	Out	1	This is a field bit timing signal. It indicates which video field is currently active.
v	Out	1	This is a vertical blanking interval timing signal. It is asserted when the vertical blanking interval is active.

Table 9-1: SDI Receiver Ports (Cont'd)

Port	I/O	Width	Description
h	Out	1	This is a horizontal blanking interval timing signal. It is asserted when horizontal blanking interval is active.
trs	Out	1	This output is asserted when the current video output is the TRS.
nsp	Out	1	This output is asserted when the framer has detected a framing position different from the original starting framing position.
h_pos	Out	12	This is the current horizontal position of the video.
v_pos	Out	10	This is the current vertical (line) position of the video.
packet_flags	Out	4	These are the EDH output flags as enabled by err_flg_en.
edh_ap_err	Out	1	This output is asserted when an EDH error in the active picture portion of the video is detected.
edh_ff_err	Out	1	This output is asserted when a full field EDH error has occurred.
ap_flags	Out	5	These are active picture flags.
ff_flags	Out	5	These are full field flags.
anc_flags	Out	5	These are ancillary data packet flags.
err_detected	Out	1	This output indicates that an EDH error is detected, as defined by enabled err flags.

Reference Design

Together with the SDI receiver modules described in this chapter, additional modules are provided as part of the reference design for design robustness and demonstration purposes. These additional modules are instantiated in the top level of the design. The individual receiver can be instantiated by using the `sdi_rx_softedh.v/.vhd` modules. Special reset blocks are included to ensure that the logic is reset properly upon startup after the clocks have become stable. The reference design files can be downloaded at <https://secure.xilinx.com/webreg/clickthrough.do?cid=119758>. Open the ZIP archive and extract the file `xapp1014_sec3_SDI_LVDS.zip`.

SD-SDI Transmitter

Summary

The SD-SDI standard describes how to transport standard-definition digital video serially over coaxial cable. SD-SDI is commonly used to connect SD video equipment in broadcast studios and video production centers. Refer to *Audio/Video Connectivity Solutions for Virtex®-II Pro and Virtex-4 FPGAs* [Ref 5] for detailed information on the related standards for SD-SDI.

The reference design presented in this chapter comprises a SD-SDI transmitter with error detection and handling (EDH) generation, ancillary data multiplexing, and internal video pattern generation. This implementation uses the LVDS I/O standards from Xilinx® SelectIO™ technology, as opposed to other Xilinx implementations that use the GTP transceiver I/Os. Summaries for each functional block are given in this chapter. The details of each functional block are presented in the individual SD-SDI chapters for the video encoder, video decoder, video flywheel, ancillary data, and EDH processors. Refer to the respective chapters in this application note for further information regarding these functional blocks.

SD-SDI Transmitter

Figure 10-1 illustrates the SDI transmitter block diagram. The transmitter includes a more full-featured video processor and EDH processor than the ones used in the receiver. This enhancement provides additional capabilities and shows how to apply these additional modules that are part of the overall suite of Xilinx video connectivity reference designs.

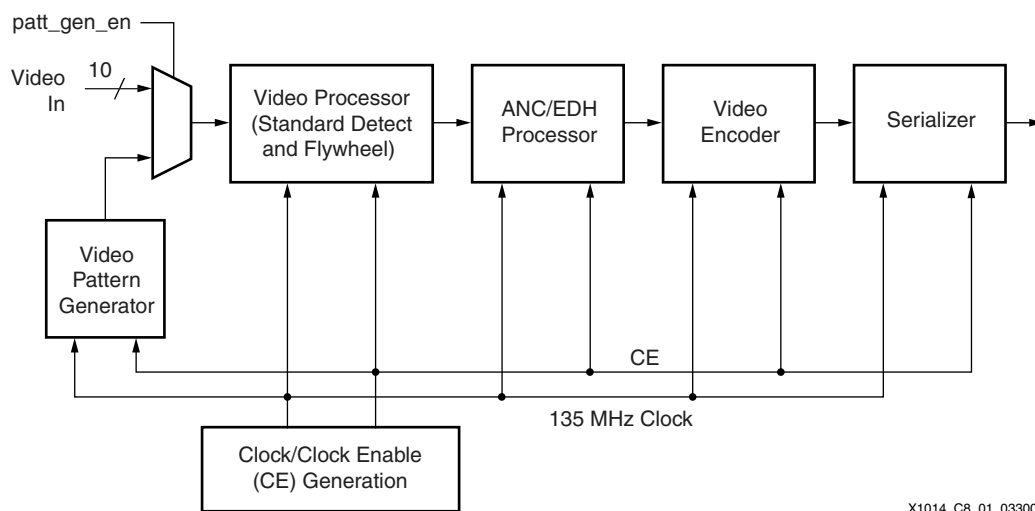


Figure 10-1: SDI Transmitter

The SDI transmitter includes an internal pattern generator, a flywheel for noise immunity, an ancillary and EDH processor that provides ancillary data multiplexing and EDH generation, an SDI video encoder (also known as a scrambler), video standard detection, and a 10:1, 135 MHz/270 Mb/s dual data rate (DDR) serializer. A multiplexed video input port allows the user to choose between the internal pattern generator and another video source for the transmitter.

Video Pattern Generator

Video pattern generators are often included with modern video test equipment as a means to provide internal diagnostics or known good video test data. From this perspective, it is useful for a transmitter to have an embedded video pattern generator.

The included video pattern generator is a simple NTSC video generator that generates SMPTE EG-1 color bars or the RP178 check field. The SMPTE EG-1 color bar pattern is the preferred pattern for image generation while the included RP178 check field generator can be used to stress the PLL and cable equalizer of attached video equipment. Detailed information on this pattern generator as well as other video pattern generators (e.g., PAL) is available in the “SDTV Video Pattern Generators” chapter of *Audio/Video Connectivity Solutions for Virtex-II Pro and Virtex-4 FPGAs* [Ref 5]. The pattern desired (EG-1 or RP178) is selectable via an input to the SDI transmitter module.

Standard Detect and Flywheel

Prior to processing, SDI streams are often modified by inserting ancillary data and the EDH packet for error handling. Furthermore, the location of this data (both ancillary and EDH) varies with the video standard. Consequently, any video processor must first detect the video standard to properly locate and modify the ancillary data (if needed), and check or modify the EDH packet. Standard detection, in its simplest form, involves detecting and decoding the timing reference symbol (TRS) word in the video stream. The TRS word contains timing information regarding the current field as well as vertical and horizontal blanking. Video processors use this information to determine the start of a new video line, calculate how many video words are present on a line, and compare this number to the known standards.

In addition to correctly detecting the standard, the video processor must also synchronize to the incoming video stream such that it knows the horizontal and vertical positions of the current video sample. Using this information, the processor can insert, modify, or delete ancillary data packets and EDH check words as needed. For this reference design, a special type of video processor, called a flywheel, provides not only these vital functions, but some measure of noise immunity as well. A video flywheel, like its mechanical counterpart, can recover easily from noisy or briefly interrupted video streams while providing consistent video timing data to the downstream video equipment, thus making the overall system more reliable.

Like a standard video processor, the flywheel first attempts to synchronize itself to the incoming video stream by decoding the TRS symbols. However, unlike a standard video processor, after the flywheel is locked to the incoming source, it generates its own video timing signals. The flywheel also continuously monitors the incoming video timing signals and compares them to its internal timing signals. When a difference in timing occurs (due to noise or switching), the flywheel does not immediately lose lock and try to reacquire the video signal as a standard processor would. Rather, it continues to generate correct video timing signals (analogous to momentum in a mechanical flywheel). In most cases, the noisy input video corrects itself. However, if the signal continues to be noisy or has switched to a different standard, the flywheel eventually drops its lock and reacquires the input timing. An input to the transmitter is also provided that, when asserted, forces the flywheel to reacquire the input signal.

In addition to noise immunity, the video flywheel can also be used to correct invalid TRS symbols. Because the flywheel continues to generate correct TRS symbols even when the video source is disconnected, downstream video equipment can remain synchronized even though the visual information in the data is invalid. The standard detection and flywheel are described in detail in the “SD-SDI Video Flywheel” chapter of *Audio/Video Connectivity Solutions for Virtex-II Pro and Virtex-4 FPGAs* [Ref 5].

Ancillary Data/EDH Processor

Ancillary and EDH processing can involve several different steps. In general, ancillary data (ANC) can be multiplexed (inserted) into the video stream, demultiplexed (extracted) from the video stream, or both. EDH processing can involve checking and generation. The SDI TX implementation in this reference design implements ancillary data multiplexing and EDH generation.

Ancillary data multiplexing can be used to place a wide variety of “non-video” information into the encoded video stream. It is useful to understand that EDH packets are just a form of ancillary data with special identifiers that denote them as EDH packets.

The included EDH processor provides an input named `receive_mode` to disable the EDH processor receiver functions. This input is useful in a transmitter-only application because errors would be flagged on the incoming video if no EDH packet were present. For example, when using the internal pattern generators as the video source, the EDH packet is not present. The `receive_mode` input should thus be deasserted to disable the receiver functions to avoid false errors in the EDH packet. If `receive_mode` is enabled, the EDH processor performs CRC checks on the incoming EDH packet. See [Table 10-1](#) for more information about the `receive_mode` input.

The ancillary data multiplexer in the EDH processor allows the user to insert any ancillary data desired into the video stream. The multiplexer properly formats the user-defined input data into a new ANC packet, and inserts it into available ANC space.

The EDH generator inserts new EDH packets into the video stream by either generating a new packet (if one was not already present) or by modifying the existing EDH packet. New

CRC values are calculated and any internal error flags are set. Several inputs are provided for the purpose of inserting internal errors. These are documented in [Module I/O](#), page 292.

Video Encoder

Prior to transmission, the raw video data must be encoded to the SDI standard. This encoding process ensures a polarity-free, transition-rich data stream for the SDI receiver to lock onto. The encoding process uses a simple polynomial scrambler for this purpose. There are several methods to implement the encoder, but this design uses a simple parallel architecture. The details of the encoder can be found in the “SD-SDI Video Encoder” chapter of *Audio/Video Connectivity Solutions for Virtex-II Pro and Virtex-4 FPGAs* [Ref 5].

Serializer

The final logic block of the transmitter is the serializer. The serializer provides a 10:1 serialization ratio. To make the serializer more portable to other Xilinx FPGAs, it is designed as a DDR serializer, allowing the design to use a half bit-rate clock instead of a full bit-rate clock. Current Xilinx FPGAs include DDR registers in the input/output blocks. These blocks make it easier to implement DDR functionality. The logic shifts two bits per clock cycle to the output DDR register, which then shifts the final output data. Thus, the serialization takes place in two stages. The first stage slices the 10-bit input word into five sequential 2-bit slices. The DDR register then provides the final 2:1 serialization.

Clocking

The transmitter requires only a single clock and clock enable to function. The serializer uses a half bit rate clock. Thus, the transmitter’s half bit rate clock rate is 135 MHz. This clock can be supplied to all the transmitter functions if a suitable word-rate clock enable is also generated. In this reference design, a simple clock enable generator provides clock enable pulses at the rate of 1/5th the 135 MHz clock, which is equivalent to a 27 MHz video rate. Only a small portion of the serializer uses the 135 MHz clock, which makes timing closure much easier.

In all cases, if desired, the clock enables can be eliminated and a synchronous word-rate clock can be used instead. However, this is generally an inefficient use of clocking resources.

Module I/O

[Table 10-1](#) describes the input and output ports of the transmitter module.

Table 10-1: SDI Transmitter Ports

Port	I/O	Width	Description
clk	In	1	This is the half bit rate clock, or word-rate clock. If a word-rate clock is used, it must be synchronous with the serclk_0 input.
ce	In	1	This is the clock enable input for the clk domain. It should be at 1/5th of the serclk rate if clk is the same as serclk_0.

Table 10-1: SDI Transmitter Ports (Cont'd)

Port	I/O	Width	Description
rst	In	1	This is an asynchronous reset input. It resets all logic.
vid_in	In	10	This is the unencoded raw video input.
reacquire	In	1	This input is asserted to force the video flywheel to reacquire lock to the input video.
en_sync_switch	In	1	This input is asserted for fast resynchronization to the EAV symbol at the end of the switching interval.
en_trs_blank	In	1	This input enables TRS blanking.
serclk_0	In	1	This is the half bit rate clock input.
serclk_180	In	1	This is an optional 180-degree phase-shifted serclk_0. It is not used for the Virtex-5 FPGA design.
trs_clip_en	In	1	This input is asserted to enable TRS clipping.
patt_gen_en	In	1	This input is asserted to select between the internal pattern generator or vid_in as a video source.
tst_patt_sel	In	1	This input selects either SMPTE EG-1 color bars or RP178 check field from the internal pattern generator. 0 = EG-1 color bars 1 = RP178 check field
anc_idh_local	In	1	This input is asserted to set the ANC internal error detected here (IDH) flag bit in the EDH packet.
anc_ues_local	In	1	This input is asserted to set the ANC unknown error status (UES) flag bit in the EDH packet.
ap_idh_local	In	1	This input is asserted to set the active picture (AP) IDH bit in the EDH packet.
ff_idh_local	In	1	This input is asserted to set the full field (FF) IDH bit in EDH packet.
errcnt_flag_en	In	16	This output enables various EDH flag checks for the error counter in receive mode.
clr_errcnt	In	1	This input is asserted to clear the error counter.
receive_mode	In	1	This input enables or disables receiver checks in the EDH processor. It should be driven Low for the transmitter only.
ancdm_*	In	Various	These are the ancillary demultiplexer inputs. Not currently used by the transmitter. Tie off as shown in the HDL reference design files.

Table 10-1: SDI Transmitter Ports (Cont'd)

Port	I/O	Width	Description
ancm_hanc_pkt	In	1	This input is asserted to insert ANC data into the horizontal ANC space.
ancm_vanc_pkt	In	1	This input is asserted to insert ANC data into the vertical ANC space.
ancm_pkt_rdy_in	In	1	This input is asserted to indicate that the ANC packet is ready for insertion.
ancm_cal_udw_par	In	1	This input is asserted to enable automatic user data word (UDW) parity calculation.
ancm_din	In	10	This is the ANC data input.
ancm_ld_did	In	1	This input is asserted to load the DID value into the ANC buffer space for packet assembly.
ancm_ld_dbn	In	1	This input is asserted to load the data block number (DBN) value into the ANC buffer space for packet assembly.
ancm_ld_dc	In	1	This input is asserted to load the data count (DC) value into the ANC buffer space for packet assembly.
ancm_ld_udw	In	1	This input is asserted to load the UDW into the ANC buffer. It is asserted once for each UDW, up to the DC total.
ancm_udw_wr_adr	In	8	This is the UDW write address.
std	Out	3	This is the decoded video standard. It is only valid when std_locked is asserted.
std_locked	Out	1	This output is asserted to indicate that the flywheel decoder is locked to the video input.
trs	Out	1	This output is asserted when the current video word is TRS.
field	Out	1	This output indicates the current active field.
v_blank	Out	1	This output is asserted during the vertical blanking interval.
h_blank	Out	1	This output is asserted during the horizontal blanking interval.
horz_count	Out	12	This is the current horizontal position.
vert_count	Out	10	This is the current vertical (line) position.
sync_switch	Out	1	This output is asserted to indicate that video is in the synchronous switching interval.
locked	Out	1	This output indicates that the EDH processor is locked to the video.
eav_next	Out	1	This output marks the beginning of the EAV.
sav_next	Out	1	This output marks the beginning of the SAV.

Table 10-1: SDI Transmitter Ports (Cont'd)

Port	I/O	Width	Description
xyz_word	Out	1	This output marks the XYZ word in the video input.
anc_next	Out	1	This output marks the ANC space.
edh_next	Out	1	This output marks the EDH space.
EDH Flag Outputs	Out	Various	This output is not used in the TX design.
ANC Demux Outputs	Out	Various	These ancillary demultiplexer outputs are not used in the transmitter design
ancm_pkt_in_empty	Out	1	This output is asserted by the EDH processor to indicate that it is ready to receive ANC packet data on the ANC multiplexer inputs.
adi_txout	Out	1	This is a 270 Mb/s serialized, encoded video data output.

Reference Design

Together with the SDI transmitter modules described in this chapter, additional modules are provided as part of the reference design for design robustness and demonstration purposes. These additional modules are instantiated in the top level of the design. The individual transmitter can be instantiated by using the `sdi_tx.v/.vhd` modules directly.

Special reset blocks are included to ensure that the logic is reset properly upon start-up after the clocks have become stable. In addition, a module called `ancdata_gen` is included to demonstrate how to use the ancillary data-multiplexing feature of the included transmitter. This module interfaces to the SDI transmitter and inserts predefined UDWs into the video stream that can be shown using standard video test equipment such as the Tektronix WFM700 Multi-format Waveform Monitor. The reference design files can be downloaded from <https://secure.xilinx.com/webreg/clickthrough.do?cid=119758>. Open the ZIP archive and extract the file `xapp1014_sec3_SDI_LVDS.zip`.

SD-SDI Receiver/Transmitter Demonstration Design

Summary

The SD-SDI standard describes how to transport standard-definition digital video serially over coaxial cable. SD-SDI is commonly used to connect SD video equipment in broadcast studios and video production centers. Refer to *Audio/Video Connectivity Solutions for Virtex®-II Pro and Virtex-4 FPGAs* [Ref 5] for detailed information on the related standards for SD-SDI.

The reference design presented in this chapter comprises an integration example SD-SDI receiver with checks for error detection and handling (EDH), and a complete SD-SDI transmitter with EDH generation, ancillary data multiplexing, and internal video pattern generation. This implementation uses the LVDS I/O standards from Xilinx® SelectIO™ technology, as opposed to other Xilinx implementations that use the gigabit transceiver I/Os. This chapter focuses on the integration and demonstration of the SDI receiver and transmitter using the Xilinx ML571 SDV demonstration board. Refer to [Chapter 9, SD-SDI Receiver](#) and [Chapter 10, SD-SDI Transmitter](#) for more information on the receiver and transmitter designs, respectively.

The SD-SDI application example in this chapter is designed specifically for the ML571 board. However, the bulk of the application code is generic and can be easily adapted to customer needs.

Overview

The SDI receiver/transmitter design provides a complete demonstration of basic SDI receiver and transmitter functions using standard video test equipment and the Virtex-5 FPGA. The demonstration system consists of a complete SDI receiver with an 8X oversampler and DRU, an SDI framer, a decoder, and a simple PicoBlaze™ processor EDH checker. The system also includes a complete SDI transmitter with SMPTE pattern generation, ancillary data (ANC) multiplexing, and EDH generation. The RX and TX are separate functional blocks that are integrated together into a single, top-level design. These functional blocks are described individually in the sections that follow.

SD-SDI Receiver Demonstration

This section describes the SDI receiver demonstration design for the Virtex-5 FPGA.

Required Equipment

These items are needed for the SDI receiver demonstration system:

- ML571 SDV demonstration board with power supply
- Platform USB cable for bitstream loading
- 75Ω coaxial cable with BNC connectors
- Bitstream files included in the original distribution with these files:
 - v5sdi_rxtx_demo.bit
 - v5sdi_rx.cpj
- SDI video source
- ChipScope™ Pro analyzer for monitoring the EDH checker and SDI data

Receiver Demonstration Platform Setup

The SDI receiver demonstration system is set up in this manner:

1. Connect the ML571 power supply.
2. Connect the platform USB cable to the ML571 board, as shown in [Figure 11-1](#).



X1014_C9_01_033009

Figure 11-1: Platform USB Cable Connections

3. Apply power to the ML571 board using the switch, as shown in [Figure 11-2](#).

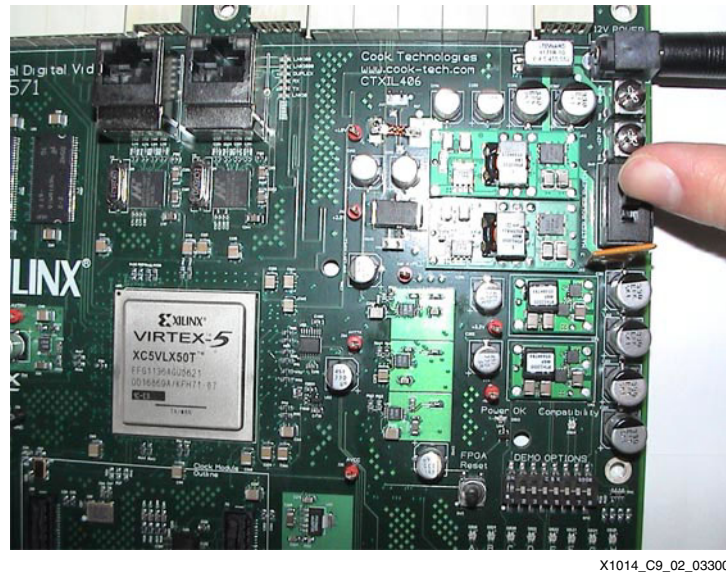
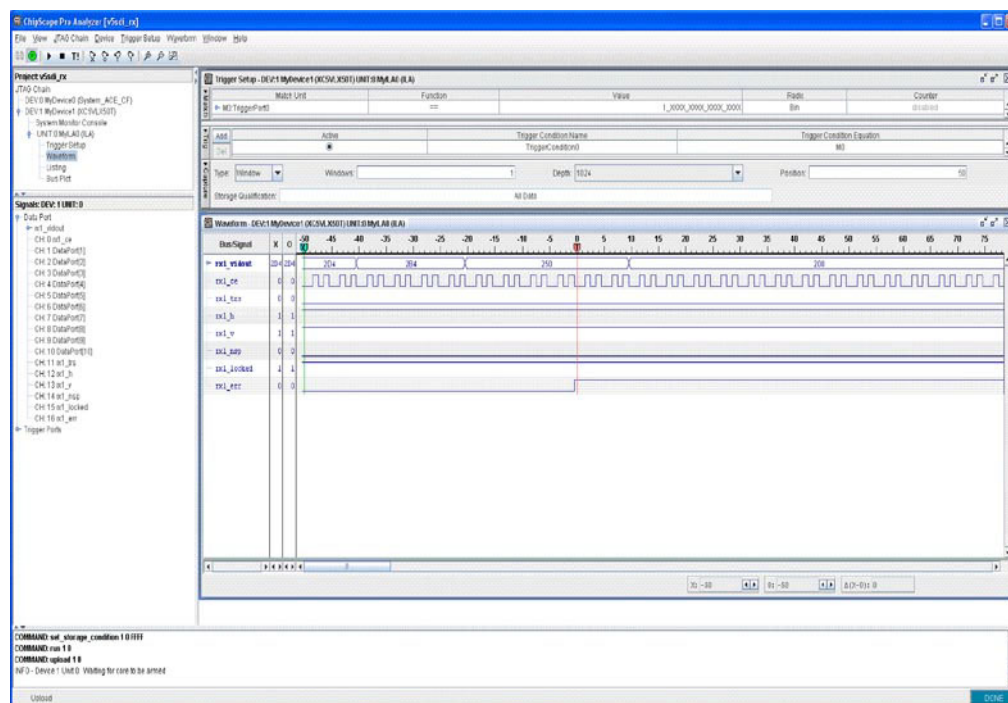


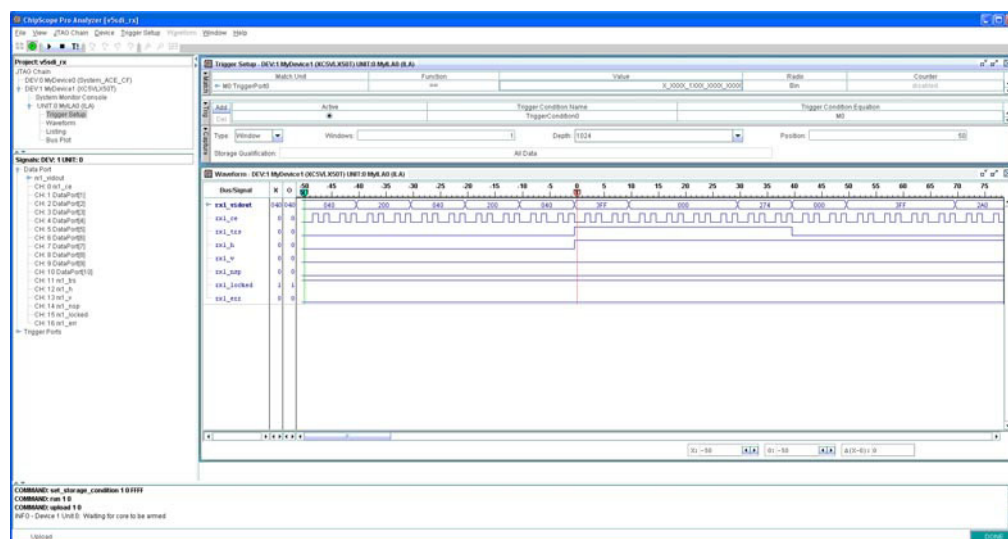
Figure 11-2: Apply Power to ML571 Board

4. Run the ChipScope Pro analyzer, open the JTAG cable, and configure the device using the `v5sdi_rxtx_demo.bit` file.
5. Load the included ChipScope Pro analyzer project file `v5sdi_rx.cpj`.
6. Use the 75Ω coaxial cable to connect a valid SDI video source to the VID RX3 input. The VID TX3 output can be used as the SDI source if no other sources are available.
7. If the design is functioning correctly, LED A and the LED next to the VID RX3 input light up green. If LED A is red, the DCMs are not locked. If the LED next to the VID RX3 input is not lit, a valid SDI input source has not been detected. LED H indicates the presence of an EDH error. If it is red, an error has been detected. Otherwise, no errors have been detected. This error LED is latched. Press PB3 to clear the LED.
8. To demonstrate the receiver, the ChipScope Pro analyzer can be used to show the data or error detection, or both. By default, the ChipScope Pro analyzer project is set up to trigger on the error output of the EDH checker. Under normal circumstances, if everything is working correctly, the integrated logic analyzer (ILA) core should never trigger. The design can be demonstrated further by changing the trigger to detect the timing reference symbol (TRS), or other conditions besides an error, and then looking at the data. [Figure 11-3](#) shows the detection of an error. [Figure 11-4](#) shows the detection of the TRS symbol.



X1014_C9_033009

Figure 11-3: EDH Error Detection



X1014_C9_04_033009

Figure 11-4: TRS Detection and Data

SD-SDI Transmitter Demonstration

This section describes the SDI transmitter demonstration design for the Virtex-5 FPGA.

Required Equipment

These items are needed for the SDI transmitter demonstration system:

- ML571 SDV demonstration board with power supply
- Platform USB cable for bitstream loading
- 75Ω coaxial cable with BNC connectors
- Bitstream file included in the original distribution with `V5sdi_rxtx_demo.bit`
- Video test equipment (or a monitor and Miranda picoLink broadcast converter, or other method of converting SDI to an input format for a standard monitor)

Transmitter Demonstration Platform Setup

The SDI transmitter demonstration system is set up in this manner:

1. Connect the power supply and platform USB cable to the ML571 board, as shown in [Figure 11-5](#).
2. Connect the 75Ω BNC coaxial cable to the VID TX 3 connector, as shown in [Figure 11-5](#). Connect the other end of the coaxial cable to the appropriate test equipment, such as the Miranda picoLink broadcast converter.

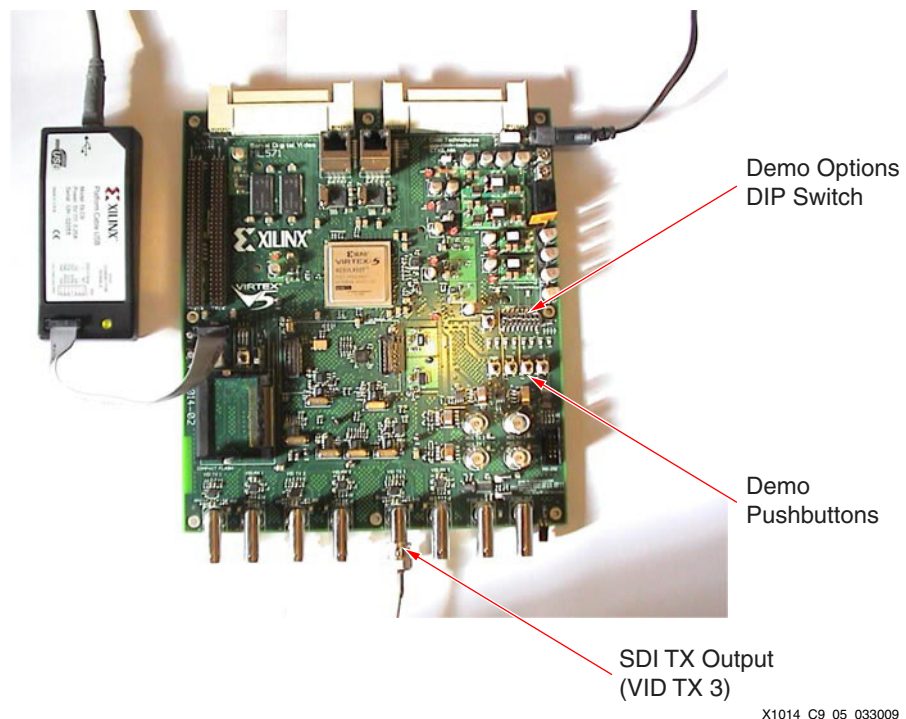


Figure 11-5: ML571 Board Connections

3. Locate the DEMO OPTIONS DIP switch near the power switch (see [Figure 11-5](#)). As shown in [Figure 11-6](#), ensure that:
 - a. SW1 is set to ON
 - b. SW2 is set to OFF
 - c. SW3 and SW4 are set to ON
 - d. SW5 to SW8 are set to OFF

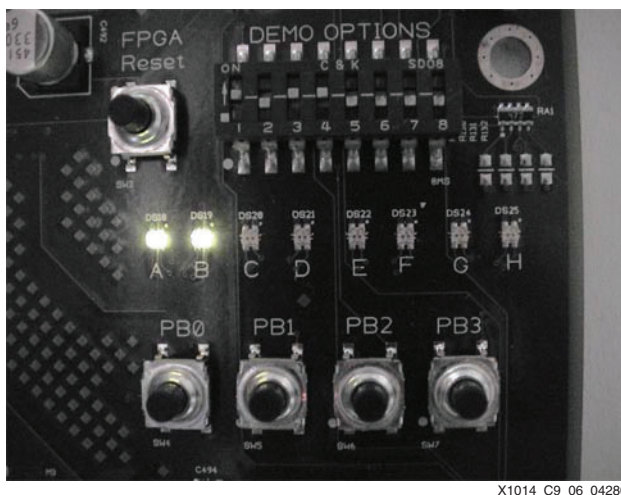


Figure 11-6: DIP Switch Settings

4. This default setup generates SMPTE color bars, ANC data multiplexing, and EDH packet generation with ANC or active picture (AP)/full field (FF) internal error detected here (IDH), or unknown error status (UES) errors. Apply power to the board.
5. Start the iMPACT software tool and load the bitstream file `v5sdi_rxtx_demo.bit`. Upon successful configuration, LED A and LED B should be green, as should the LED next to the BNC connector for VID TX 3 (see [Figure 11-7](#)). If the setup is being viewed on a monitor or test equipment, the video pattern should look like [Figure 11-8](#). LED A indicates that the DCM is locked. LED B and the VID TX 1 LED indicate that the video processor is locked to the digital video being produced by the internal pattern generator. These LEDs turn red if an error occurs.

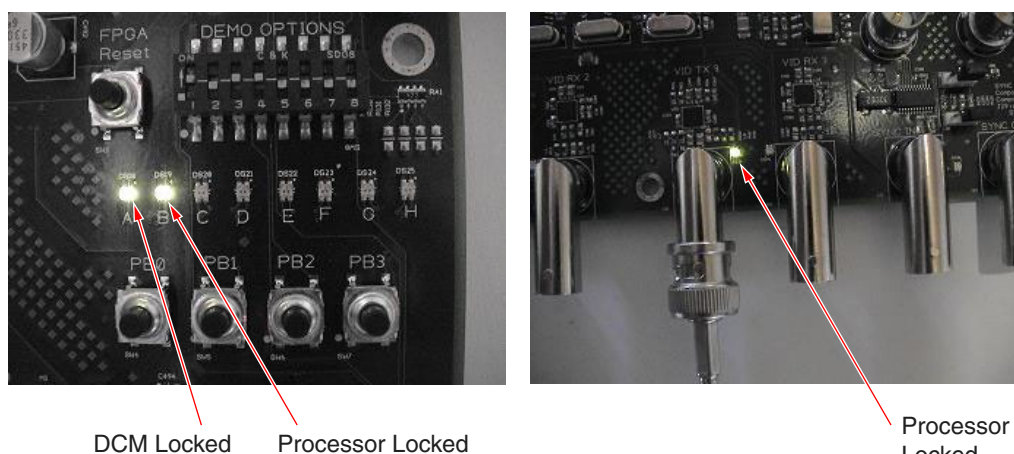


Figure 11-7: SDI TX LEDs

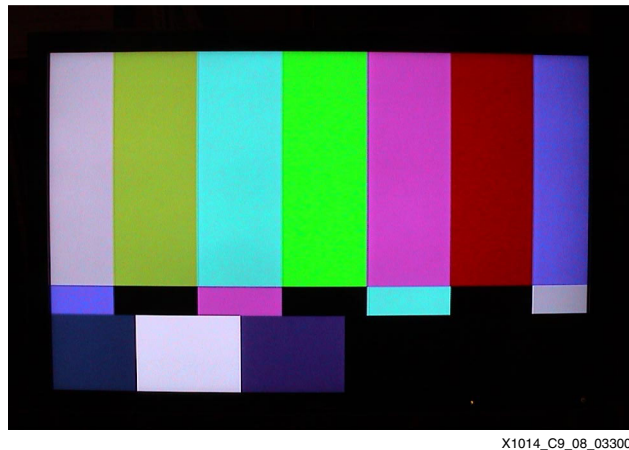


Figure 11-8: SMPTE EG-1 Color Bars

Design Summary

The SDI TX demonstration system provides for a number of options to demonstrate SDI TX functionality. By using the default setup described in [SD-SDI Transmitter Demonstration, page 301](#), the SDI transmitter produces SDI video at an output rate of 270 Mb/s. The video consists of SMPTE EG-1 color bars that can be viewed on either a monitor (additional equipment required) or on standard video test equipment. The output video format is NTSC 525i, 59.94 Hz, 4:2:2 component video.

In addition to the video output, the design automatically generates and inserts the RP 165 EDH packet to demonstrate EDH generation. Additional video test equipment is required to detect and view the packet's contents. The DIP switches described in [Running the Demonstration](#) also allow the user to set local IDH and UES bits in the EDH packet.

The design also generates and inserts ancillary data packets into the output video to demonstrate ANC multiplexing. This data can be viewed using standard video test equipment. The inserted data is fixed and cannot be changed by the user without modifying the design.

Running the Demonstration

This section discusses the DIP switch, pushbutton, and other settings that can affect the demonstration functionality. It also describes the ancillary data format inserted into the output video and how to set up the video test equipment to view this output.

As mentioned in [Design Summary](#), the transmitter produces SMPTE EG-1 color data by default. When using standard video test equipment to view this color bar pattern, it is common to see Luma gamut and red, green, and blue (RGB) errors because the SMPTE EG-1 pattern contains gamma values that are outside the expected range of the default settings for some equipment. This is typical on the Tektronix WFM700, for example, and these errors can be safely ignored. However, the video test equipment should not report errors with respect to the AP and FF CRC or other checks. The $YC_B C_R$ gamut should also be correct.

Numerous DIP switch settings affect the functionality of the design, as shown in [Table 11-1](#). All are located on the DEMO OPTIONS DIP switch on the ML571 board. For the SDI TX design, SW1 should always be in the ON position. This disables EDH checking in

the ANC/EDH processor. If this switch is set to OFF, errors are reported in the generated EDH packet due to the packets missing in the incoming video from the pattern generator.

Table 11-1: SDI Transmitter Switch Settings

Switch Number (SWn)	Function When Set to ON
SW1	Disables receiver functions (EDH checking). Set to ON for normal TX demonstration.
SW2	Disables the internal pattern generator. Set to OFF for normal TX demonstration.
SW3	Internal pattern generator pattern select. Selects between SMPTE EG-1 color bars (ON) or RP178 checkfield test patterns (OFF).
SW4	Disables TRS clipping.
SW5	Deasserts the ANC IDH error into the RP165 EDH packet.
SW6	Deasserts the ANC UES error into the RP165 EDH packet.
SW7	Deasserts the AP IDH error into the RP165 packet.
SW8	Deasserts the FF IDH error into the RP165 EDH packet.

DIP switches SW5 to SW8 require more discussion. To test the EDH functionality, these DIP switches toggle specific bits in the generated RP165 EDH packet that is always present in the TX output stream. When these DIP switches are set to ON, the bits associated with ANC IDH, UES, and AP/FF IDH errors are reset to 0. Setting these switches to OFF causes the corresponding bit in the EDH packet to be set to 1. This functionality allows demonstration of EDH generation when video test equipment such as the Tektronix WFM700 is available to view the data. When operating correctly, the test equipment should report that there are no CRC or other errors with the EDH packet.

The reference design also automatically inserts ANC data to demonstrate the ANC multiplexing function of the design. The inserted data pattern consists of 15 fixed UDWs with a fixed DID. To view this data on standard video test equipment, the equipment should be set to trigger on a DID of 0xA0 (unknown type 1 packet). When captured, the ANC data should appear as a data count (DC) of 15, with the UDW set to an incrementing value from 0x0 to 0xE. When operating correctly, the test equipment should report that there are no CRC or other errors with the ANC data.

Three pushbuttons affect the demonstration. These pushbuttons, numbered PB0 to PB2, are located directly beneath LEDs A to H (see [Platform USB Cable Connections, page 298](#)). [Table 11-2](#) defines the pushbutton functions.

Table 11-2: SDI Transmitter Pushbutton Functions

Pushbutton Number (PBn)	Function
PB0	Resets the DCM
PB1	Forces the flywheel decoder to reacquire a TRS on an incoming video signal
PB2	Resets the demonstration

Section IV: DVB-ASI using Virtex-5 FPGA SelectIO LVDS

***Audio/Video Connectivity Solutions
for Virtex-5 FPGAs***

DVB-ASI Introduction and Layer 0 Implementation

Background

Digital Video Broadcast (DVB) is the result of a large-scale cooperative effort by numerous commercial manufacturers, television network operators, broadcasters, and others to provide a uniform set of standards for digital television and data services. DVB has proven highly successful, with over 150 million DVB receivers deployed.

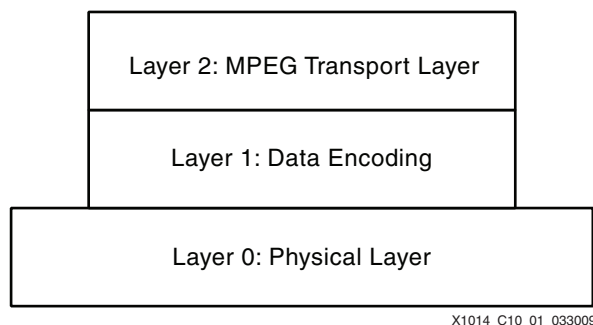
Late in 1991, at the beginning of digital television deployment, industry-leading manufacturers, broadcasters, and regulatory bodies formed the European Launching Group to guide and manage the technology requirements for digital media services. In 1993, a memorandum of understanding was signed by all participants, and the DVB project was born.

The DVB project immediately addressed the critical need for standards required to deliver digital media content via the traditional broadcast networks. The three key standards introduced during this period were DVB-S (satellite), DVB-C (cable), and DVB-T (terrestrial). These standards have been widely adopted in Europe, Australia, and Asia. DVB-S forms the basis for satellite televisions almost everywhere. DVB-C is the most common cable system format, and DVB-T is increasing in popularity.

At this time, other supporting standards are needed for service information, subtitling, and interfacing. DVB-ASI is one such standard. It is one of three primary interfaces prescribed for the transmission and reception of MPEG-2 compressed data signals. This interface is specified in detail in European Standard EN 50083-9 [Ref 8], originally approved by Cenelec in 1996.

DVB-ASI

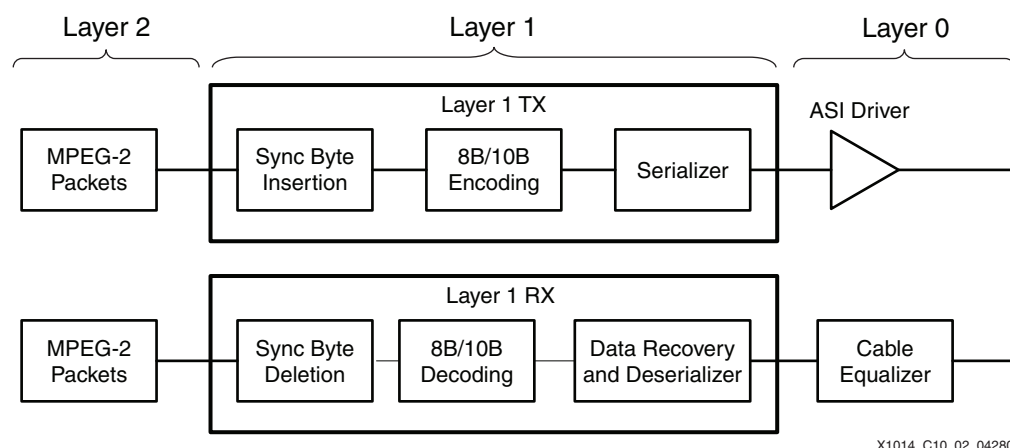
DVB-ASI describes a system interface for serial, encoded transmission of different data rates using a constant transmission rate. Based on a three-layer structure, DVB-ASI uses MPEG transport packets as described in EN/ISO/IEC 13818-1 as the top layer (layer 2) and a pair of bottom layers (layer 1 and layer 0) that are based on the Fibre Channel standard ISE/IEC/CD 141165-1, part 1. While Fibre Channel specifications form the basis for this interface, DVB-ASI is not restricted to fiber as a transmission medium. The interface described in this chapter uses standard 75Ω copper coaxial cable as defined by the standard. [Figure 12-1](#) shows the DVB-ASI layered structure.



X1014_C10_01_033009

Figure 12-1: DVB-ASI Layered Protocol

Each of the layers comprises different physical and logical aspects of the interface. Figure 12-2 shows how each layer of the DVB-ASI standard maps to a generic design implementation.



X1014_C10_02_042809

Figure 12-2: DVB-ASI Generic Design Implementation

Layer 2

Layer 2 represents the MPEG-2 transport stream. This transport stream is not part of the layer 1/layer 0 implementations that are discussed in other chapters. The MPEG-2 transport stream is composed of MPEG-2 packets that can be either 188 bytes in a standard transmission, or 204 bytes when the packets are Reed-Solomon encoded for forward error correction. These packets are passed one byte at a time to layer 1. Figure 12-3 shows the format of a basic MPEG transport packet for ASI prior to being processed by layer 1 for encoding and serialization. The DVB-ASI specification requires every MPEG transport packet to be separated by special comma characters to enable byte synchronization within one MPEG packet. The hex value BC is used as the comma character. Each MPEG packet must be separated by at least two comma characters. This is discussed in more detail in Layer 1. Commas are also referred to as sync bytes.

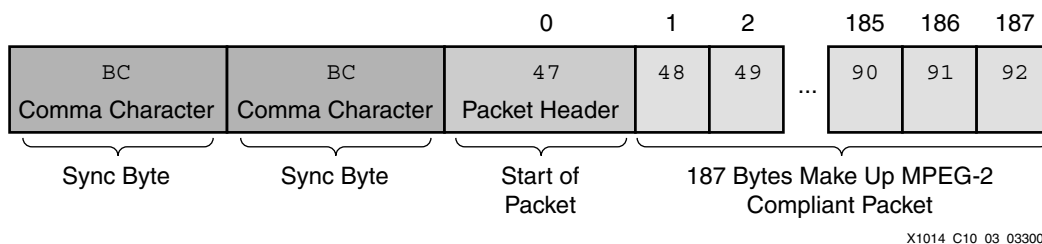


Figure 12-3: MPEG Transport Packet Format

Aside from the two-comma separation between packets, there are no rules that govern how packets are sent. Packets can be sent in any fashion as long as each packet is separated by two consecutive commas. It is common for an MPEG-2 transport stream to contain many comma characters in addition to the minimum of two commas because the full transport stream bandwidth might not be needed. For example, if the user sends a 10 Mb/s stream on a single channel, 206 Mb/s is still available. The specification calls for a transmitter to send commas if no packet data is available. Thus, a given MPEG transport stream can be padded with a lot of comma characters.

Layer 1

Layer 1 comprises data encoding and serialization, comma insertion, and in the case of the receiver, comma correction (comma insertion/deletion). The encoding block uses a DC-balanced 8B/10B transmission code. 8B/10B codes are transition rich (run length limited), have minimal DC offset, and provide for some basic error detection by detecting invalid 10-bit code words (code error) and by tracking and enforcing the notion of running disparity (disparity error). Disparity is the difference in the number of ones and zeros present in the current and subsequent 10-bit code words. Running disparity tracks this difference and enforces it to be either positive or negative, thus providing a transition-rich, run length-limited, DC-balanced transmission stream. A complete discussion of 8B/10B codes is beyond the scope of this chapter, but a complete description of the code is available in technical publications [Ref 9].

After it is encoded, the 10-bit data is converted to serial data by using a 10:1 serializer. The serializer shifts one bit per clock at a line rate of 270 Mb/s. The 8B/10B block code has a 20% overhead penalty because each byte of 8 bits requires 10 bits to be transmitted. Thus, the actual payload bandwidth of the link is 216 Mb/s.

Prior to encoding and serialization, the MPEG transport stream is byte synchronized by inserting commas into the stream for byte synchronization. Byte synchronization is necessary because the receiver samples the input asynchronously so that the data produced by the receiver can span word boundaries. To align on word boundaries, special patterns guaranteed not to occur in the input data are provided by the encoding scheme. These words, called comma or K characters, provide a mechanism to determine the offset of the incoming data relative to the correct word boundaries. The K character used for DVB-ASI is the K28.5 word as defined in the 8B/10B codes. This word decodes to an 8-bit value of BC hex, which is also a valid 8-bit data value. To distinguish between the two during encoding, the encoder has a special input to flag the current input byte as a special K character, which causes the 8-bit word to encode differently. In general, the byte synchronization pattern consists of two commas within a 5-byte window. A common pattern is two successive commas.

The ASI receiver searches the incoming data for the comma pattern. After the comma pattern is detected, an offset is calculated, indicating the amount of rotation needed in the

data words to properly align them on word boundaries. In general, the exact location of these byte-synchronizing characters is unimportant because, other than for synchronization purposes, they are to be ignored by the receiver.

The receiver first deserializes and then decodes the incoming serial data stream. First, the data is recovered by a clock and data recovery circuit using a PLL or a simple oversampling data recovery unit. Next, the data is decoded from 10 bits to 8 bits using the 8B/10B codes. Code and disparity errors are detected at this point. Lastly, the receiver performs comma correction on the data.

Comma correction is a method of matching the layer 1 receiver's clock rate to the MPEG-2 transport stream layer (layer 2). Layer 2 designers might wish to extract data from the receiver faster or slower than the incoming bit rate dictates. Because this is a streaming interface, no handshaking can take place. Instead, the receiver uses a special elastic memory buffer and inserts or deletes comma characters as needed to match the transmitter rate to the receiver rate. This allows the layer 2 implementation to extract data from layer 1 at a convenient clock rate. For example, the incoming data rate can be, on average, 27 MHz. The layer 2 implementation can use a 33 MHz clock, if necessary. By using comma correction, the layer 2 implementation can clock data from the receiver at this faster rate without underflowing the receiver and getting corrupted data. The layer 2 implementation reads and discards additional sync bytes until it sees actual data as indicated by the receiver. Conversely, if the layer 2 implementation runs at a rate slower than the receiver, layer 1 deletes comma characters as needed to prevent overflow and lost data. Only commas can be inserted or deleted because they are ignored by the layer 2 implementation.

Layer 0

Layer 0, the physical layer of the protocol, addresses jitter performance, line rates, bit timing, receiver timing acquisition, and electrical medium characteristics. While both optical and copper transmission mediums are supported by the standard, this chapter deals only with the copper coaxial cable interface.

Jitter in coaxial applications is specified by random jitter (RJ), deterministic jitter (DJ), also known as data-dependent jitter, and duty cycle distortion. Specified as peak-to-peak values, the DJ component should be no more than 10% of the line rate unit interval (UI). RJ should be no more than 8% of the UI.

Line rates are 270 Mb/s. Receivers might not recover the clock from the encoded bitstream in addition to the data. In the examples described in this chapter, only data recovery is implemented. The line rate is specified as 270 Mb/s \pm 100 ppm.

Receiver timing acquisition is the amount of time required for the ASI receiver to synchronize or bit align to the incoming data stream. This is different from the byte synchronization process discussed in [Layer 1, page 309](#). Bit alignment is the notion of sampling a data bit in the valid region, avoiding edges and metastability. In PLL-based systems, receiver timing acquisition includes the time for the PLL to lock onto the clock embedded in the data and sample the incoming serial data. In asynchronous sampled systems such as those discussed here, receiver acquisition time is a function of the design of the receiver data sampling and recovery circuit. The specification requires the receiver acquisition time to be less than or equal to 1 ms.

The electrical medium characteristics for a copper coaxial interface specify a nominal cable impedance of 75 Ω . BNC connectors are recommended for their mechanical robustness. Detailed electrical specifications can be found in EN 50083-9.

Implementing DVB-ASI Layer 0 Using SelectIO Technology

Xilinx® FPGAs feature I/O cells with a rich set of programmable standards known as SelectIO™ technology that can be used to implement DVB-ASI. Some external components are required to meet the DVB-ASI electrical specifications. In general, ASI specifies receiver and transmitter electrical characteristics without specifying the implementation details. These specifications can generally be met by utilizing techniques common in the digital television SDI standard. Receivers are required to use transformer coupling. Cable length requirements are not specified, but most customers desire cable lengths similar to those specified for SDI. DVB-ASI uses a differential interface similar to standard low-voltage differential signaling (LVDS).

DVB-ASI Receiver Implementation Using SelectIO Interface

The receiver specification is described in EN 50083-9, which calls for transformer coupling between the receiver and connector. Figure 12-4 illustrates the implementation of DVB-ASI using the Xilinx LVDS_25 I/O standard with internal termination. No external terminating resistor is required. The internal LVDS input buffer with SelectIO technology provides a robust, high-speed LVDS interface for ASI.

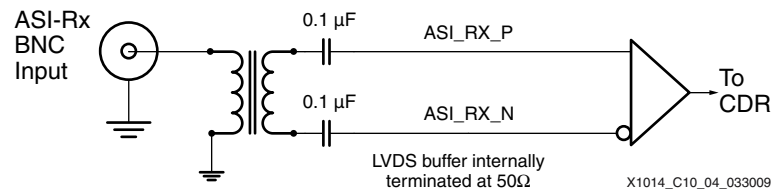
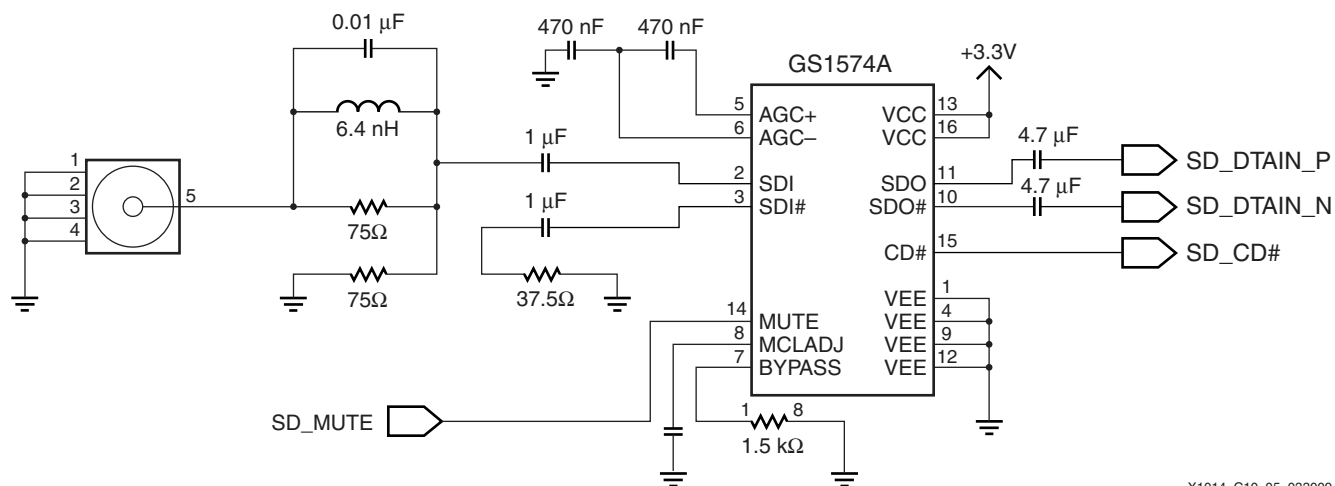


Figure 12-4: DVB-ASI Implementation Using Xilinx LVDS_25 I/O Standard

With its slower bit rates, SD-SDI allows maximum cable lengths of up to 300 meters. The coaxial cable causes frequency-dependent attenuation of the signal, where the higher frequency components of the signal are attenuated more than the lower frequency components. The coaxial cable also causes frequency-dependent phase distortion, where the higher frequency components are phase shifted more than the lower frequency components. After passing through long coaxial cables, the signal is severely distorted and attenuated. The receiver must compensate for this attenuation and distortion before attempting to recover the signal. Cable length equalization is also used for this purpose.

Typically, an adaptive cable length equalizer is used in SDI receivers. Such an equalizer actively monitors the amount of attenuation and distortion present on the incoming signal and applies the correct amount of equalization to the signal. The cable length can be changed without requiring a change to the equalizer, as would be the case if fixed-length equalization were used. The SDI specifications call for capacitive coupling between the connector and the receiver. Figure 12-5 illustrates the cable equalizer used on the SD-SDI interface, as implemented on the ML571 SDV demonstration board for the Virtex®-5 FPGA (designed by Cook Technologies).

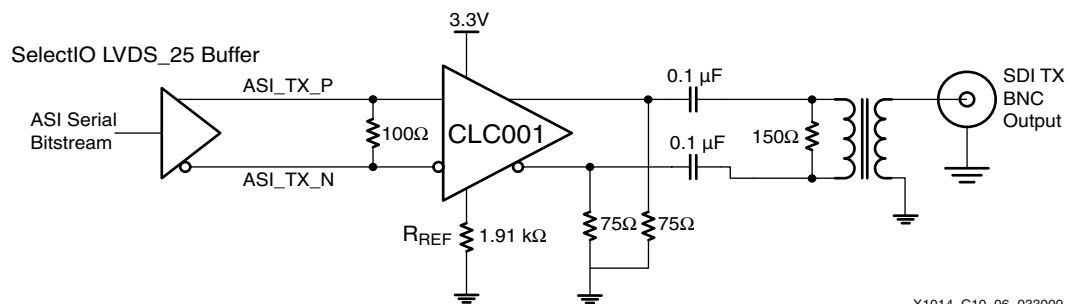


X1014_C10_05_033009

Figure 12-5: Cable Equalizer for SD-SDI Interface on ML571 Board

DVB-ASI Transmitter Implementation Using SelectIO Interface

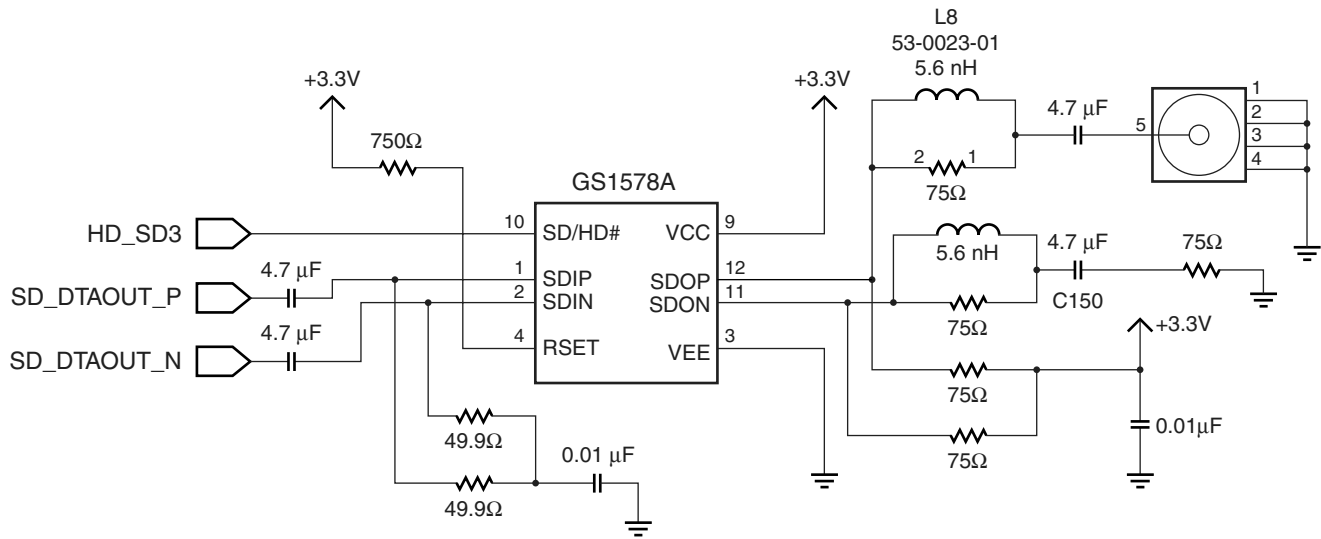
The ASI transmitter layer 0 specification calls for transformer coupling between the line driver and connector. While the signaling levels are similar to standard LVDS, the ability to drive long cables is required. Because of this, the SelectIO interface LVDS buffer cannot be used directly but must be connected to an appropriate cable driver as shown in Figure 12-6.



X1014_C10_06_033009

Figure 12-6: Cable Driver for DVB-ASI Transmitter Using SelectIO Interface LVDS Buffer

As with the receiver, it is common practice to implement an ASI transmitter layer 0 interface using the SDI transmitter specifications. This allows the user to implement a multi-standard ASI/SDI output on the same physical connection. The SDI specifications call for capacitive coupling between the line driver and the connector. Figure 12-7 illustrates the cable driver used on the SD-SDI transmitter, as implemented on the ML571 board.



X1014_C10_07_033009

Figure 12-7: Cable Driver for SD-SDI Interface on ML571 Board

Conclusion

This chapter introduces DVB-ASI and provides an implementation guide for layer 0. For complete reference designs for layer 1 and layer 2 applications of Virtex-5 FPGAs, see [Chapter 13, DVB-ASI Layer 1 and 2 Receiver](#), [Chapter 14, DVB-ASI Layer 1 and 2 Transmitter](#), and [Chapter 15, DVB-ASI Layer 1 and 2 Pass-through Demonstration Design](#).

DVB-ASI Layer 1 and 2 Receiver

Summary

DVB-ASI provides an industry-standard method for transmitting MPEG-2 compressed video over high-speed asynchronous serial interfaces. This chapter presents a DVB-ASI receiver design for the Virtex®-5 FPGA that uses the Xilinx® SelectIO™ technology. A robust 270 Mb/s ASI receiver is implemented in a cost-effective manner using advanced techniques, silicon features, and tool capabilities.

It is recommended that the reader first read [Chapter 12, DVB-ASI Introduction and Layer 0 Implementation](#) to become acquainted with the material presented in this chapter.

ASI Receiver

Figure 13-1 depicts a block diagram of the ASI receiver design.

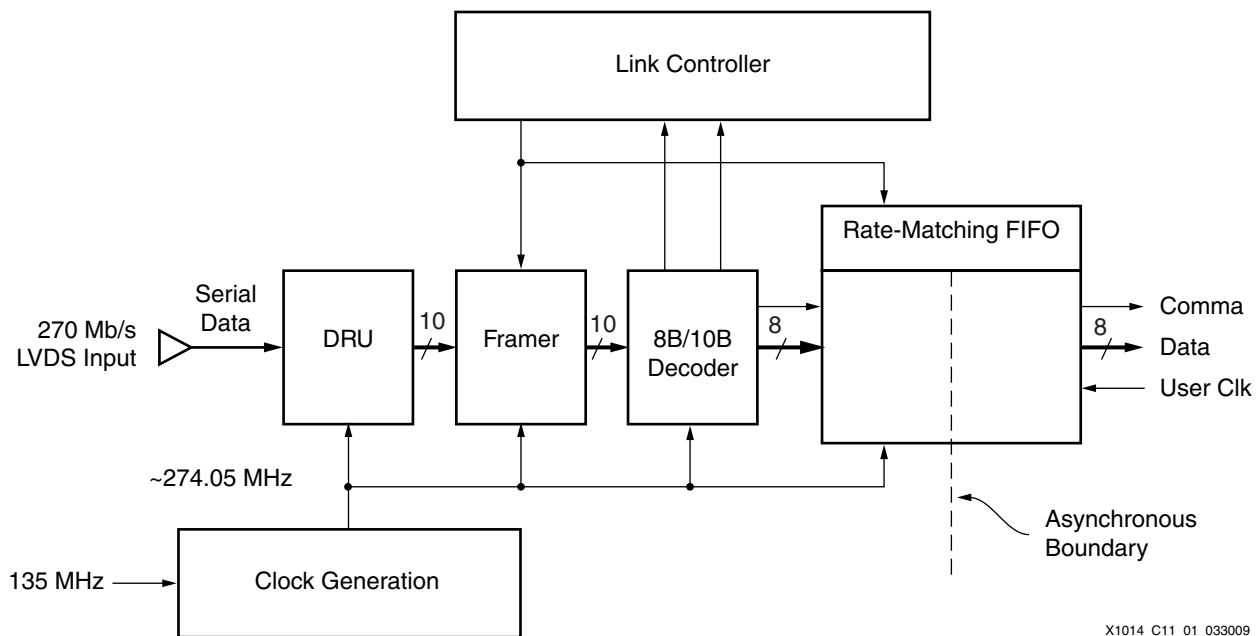


Figure 13-1: ASI Receiver Architecture

Data Recovery Unit

The DRU samples the incoming serial data stream and deserializes it to 10-bit input words. Although many methods are available for data recovery, traditional methods usually employ some sort of oversampling using eight or more sample points per bit interval. In this design, a novel method is employed that uses only one sample clock. With this clock at a sampling frequency approximately 1.5% faster than the incoming data rate, the incoming data stream is sampled at both the rising and falling edges of the clock. The sampling flip-flops are implemented using the SelectIO technology double data rate (DDR) registers in the input/output block (IOB) itself using the pipeline setting, which is present in all newer Xilinx FPGAs. The design needs just one clock buffer for the sampling clock and one global input buffer for an incoming reference, instead of many buffers as in traditional 8X oversampling techniques.

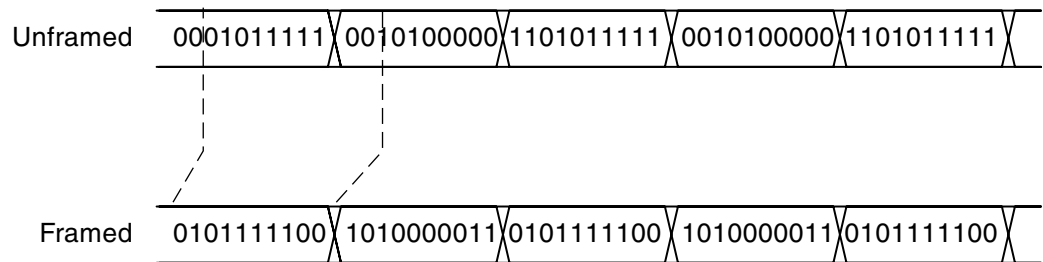
The sampling clock is known to be approximately 1.5% faster than the incoming data rate. Thus, the received data precesses through the sampling clock at a predictable rate. It requires 64 clock cycles ($1/1.5\% \sim 1/64$) to precess through one period.

Observation of data received from two flip-flops clocked on opposite edges allows the phase of the source data clock relative to the asynchronous receiver sampling clock to be determined. Data received from the two flip-flops is either aligned or off by one clock cycle. This method requires the data to be recovered to have a reasonable transition density. Standards such as DVB-ASI meet this requirement because they use 8B/10B coded data, which is run length-limited and therefore has good transition density.

ASI Parallel Framer

For DVB-ASI, the incoming data is encoded into 10-bit words using 8B/10B encoding rules. The input data is sampled asynchronously. Thus, the data produced by the DRU can span word boundaries. To align on word boundaries, special patterns guaranteed not to occur in the input data are provided by the encoding scheme. The words, called comma or K characters (8B/10B K28.5 symbols), provide a mechanism to determine the offset of the incoming data relative to the correct word boundaries. For more information, refer to [Chapter 12, DVB-ASI Introduction and Layer 0 Implementation](#).

The ASI parallel framer (framer) searches the incoming data for the comma pattern. After the comma pattern is detected, an offset is calculated, indicating the amount of rotation needed in the data words to properly align them on word boundaries. This offset is fed to a simple barrel shifter that rotates the incoming data to the correct offset, thereby aligning the data on the proper 10-bit boundaries. Because the input data is unframed, the framer uses an aggregate 30-bit word formed from multiple input words for searching. This ensures that the comma character is found regardless of the offset in the incoming data. To simplify the logic, the framer need only search for one of the K28.5 symbols because both are guaranteed to be in the data as defined by the 8B/10B encoding rules. [Figure 13-2](#) illustrates this concept.



X1014_C11_02_033009

Figure 13-2: ASI Framer Operation

Most 8B/10B codes are balanced, i.e., they have the same number of ones and zeros in each 10-bit symbol. However, for those codes that are not balanced, two code word outputs exist for the same input. These outputs are complements of one another and maintain the correct running disparity. The alternating K28.5 comma characters, 0101111100 and 1010000011, are the data shown here. This would be the transmitter output if no MPEG transport stream data were available to send. To maintain the correct running disparity, the encoder transmits one form of the K28.5 followed by its complement. In the unframed data, the 10-bit word boundaries are shifted to the right by two bits due to the asynchronous sampling of the data. After framing, the data is rotated to the left to correct the offset, thus resulting in properly framed comma characters.

The framer generally frames the input once on startup, but the ASI framer also supports reframing. A reframe forces the framer to reacquire the comma pattern in the data and recalculate the offset. The link controller is used to issue this reframe due to an error condition or if the user asserts the request externally. This reframing is described in [Link Controller](#), page 318.

8B/10B Decoder

DVB-ASI streams carry MPEG-2 compressed video data using 8B/10B encoding. A complete discussion of 8B/10B encoding is beyond the scope of this application note, but [Chapter 12, DVB-ASI Introduction and Layer 0 Implementation](#) provides some background.

The 8B/10B decoder takes 10-bit input words and decodes them to 8-bit outputs and a special code indicator output. The decoding is handled by breaking the 10-bit word into two separate 4-bit and 6-bit words called subblocks. These are referred to as the 6b and 4b subblocks, respectively, and are used to decode and check for disparity errors. The 6b subblock is decoded to a 5-bit output, and the 4b subblock to a 3-bit output. The two results are then combined to form the complete 8-bit output word.

The special code indicator is asserted whenever the 10-bit input word is one of the special K characters provided by the 8B/10B rules. The code indicator is necessary because the 8-bit data output for data and K characters can be identical. Therefore, the code indicator output indicates whether the output is data or a control character. This module is synchronous to the 54 MHz clock and has two clock cycles of latency. Outputs are also provided to indicate code and disparity errors.

In addition to decoding, the decoder also does error checking. While not exhaustive, the checks catch many simple errors due to poor transmission media or other causes. Two classes of errors are checked by the decoder:

- Code errors occur whenever the 10-bit input word is not one of the valid 268 codes that are supported. This occurrence can be caused by several conditions including data corruption errors and an unencoded or non-8B/10B-encoded input stream, e.g., attaching a serial digital interface (SDI) stream to the ASI input. An asserted code error output indicates that the code word present on the output of the decoder is invalid.
- Disparity errors are asserted whenever the 8B/10B disparity rules are violated. The disparity rules ensure that the transmission is balanced and has run lengths no longer than five. Disparity refers to the number of 1s and 0s in a 10-bit symbol or subblock. The disparity rules state that the 10-bit symbol or subblock must be balanced (zero disparity, five 1s and five 0s), have negative disparity (four 1s and six 0s), or have positive disparity (six 1s and four 0s). These values are internally represented as 0, +2, and -2, respectively. Signed arithmetic is not used; only unique constants are used for each case.

The decoder must also track the running disparity of each incoming word as a way to track the DC bias of the data stream. Initially, the decoder sets the running disparity to negative. If the first word from the encoder has negative disparity, a single error occurs. After initialization, the running disparity is updated by “adding” the current input word disparity to the current running disparity. For example, if the current input word has a disparity of +2 and the running disparity is -1, the resulting running disparity for the next input compare is +1. If the running disparity is +1 and the current input is +2, the running disparity stays at +1 and the decoder asserts an error. If the input disparity is 0, the running disparity remains the same. Running disparity is always updated, regardless of any errors.

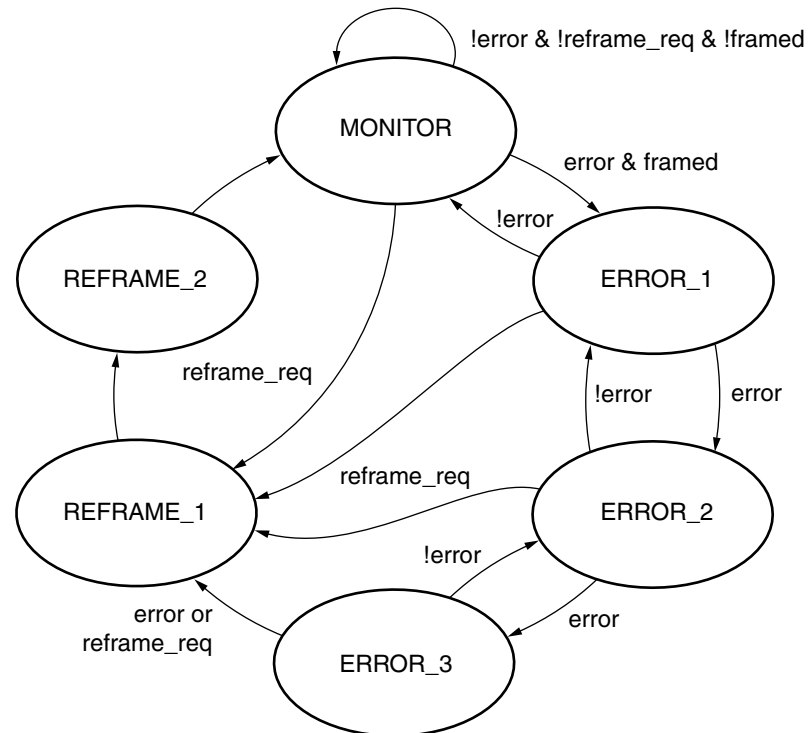
Disparity errors are caused by a violation of these basic rules:

- The 6b and 4b subblocks cannot have the same disparity.
- The 10-bit symbol cannot have the same disparity as the current running disparity.

If either of these conditions is violated, the decoder asserts the disparity error output.

Link Controller

The link controller is a simple state machine that monitors the status of the data link by tracking errors from the 8B/10B decoder. If four consecutive code or disparity errors occur, the link is assumed to be unlocked, and the link controller automatically reframes the link in an attempt to relock to the input data source. The link controller also supports reframing of the link by asserting the `reframe_req` input. This input can be external, such as a pushbutton. [Figure 13-3](#) shows the state diagram for the link controller. The framed signal input to the link controller is asserted when the framer has successfully detected a valid ASI framing sequence. After the framed signal is asserted, the link controller begins active monitoring.



error = Disparity Error or Code Error

reframe_req = Manual Request to Reframe the Link

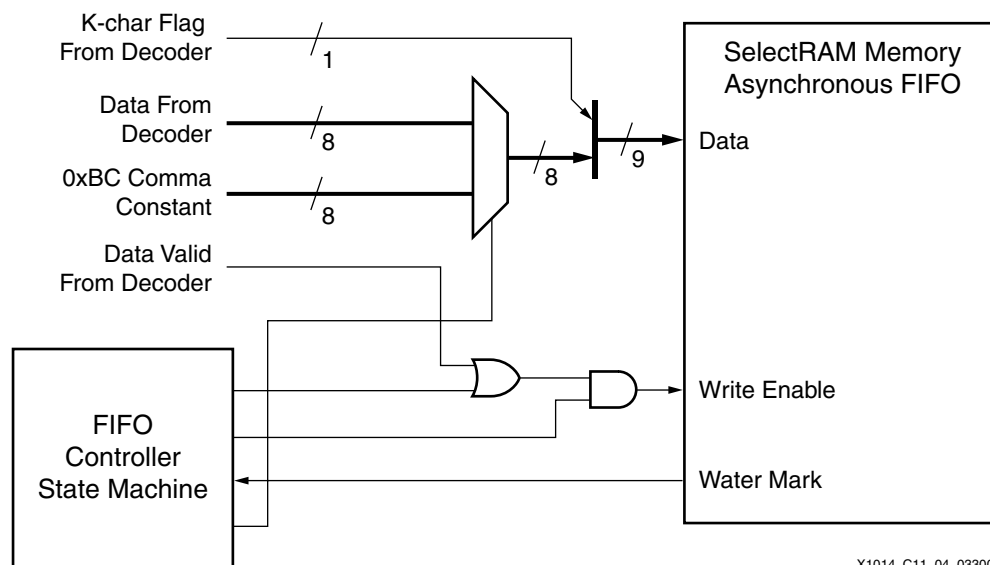
X1014_C11_03_033009

Figure 13-3: Link Controller State Diagram

Rate-Matching FIFO (Comma Correction)

A typical DVB-ASI system has the transmitter and receivers operating from different clock sources. While these clock sources are close in frequency, they can never be perfectly matched. In addition, the layer 2 implementation is commonly expected to operate faster than the layer 1 implementation. To support this capability, a method of rate matching is required to prevent the loss of data. One technique to accomplish this is called comma correction.

Comma correction relies on the fact that comma characters are present in the data stream and are ignored by the layer 2 implementation. In addition to using K28.5 comma characters for framing (also known as byte synchronization), K28.5 characters can be deleted or inserted as necessary to match the rate of the incoming data to the layer 2 clock rate. An elastic FIFO provides a rate-matching function between the incoming data rate and the desired system frequency at the layer 1/layer 2 boundary. Rate matching is handled by managing the FIFO level so that it never overflows or underflows. For example, if the layer 2 clock frequency were faster than the incoming data rate, the FIFO would eventually become empty. Conversely, if the layer 2 clock rate were lower than the incoming data rate, the FIFO would eventually overflow. In either case, data would be lost and the streaming interface would exhibit errors. Figure 13-4 shows the block diagram for the comma correction elastic FIFO.



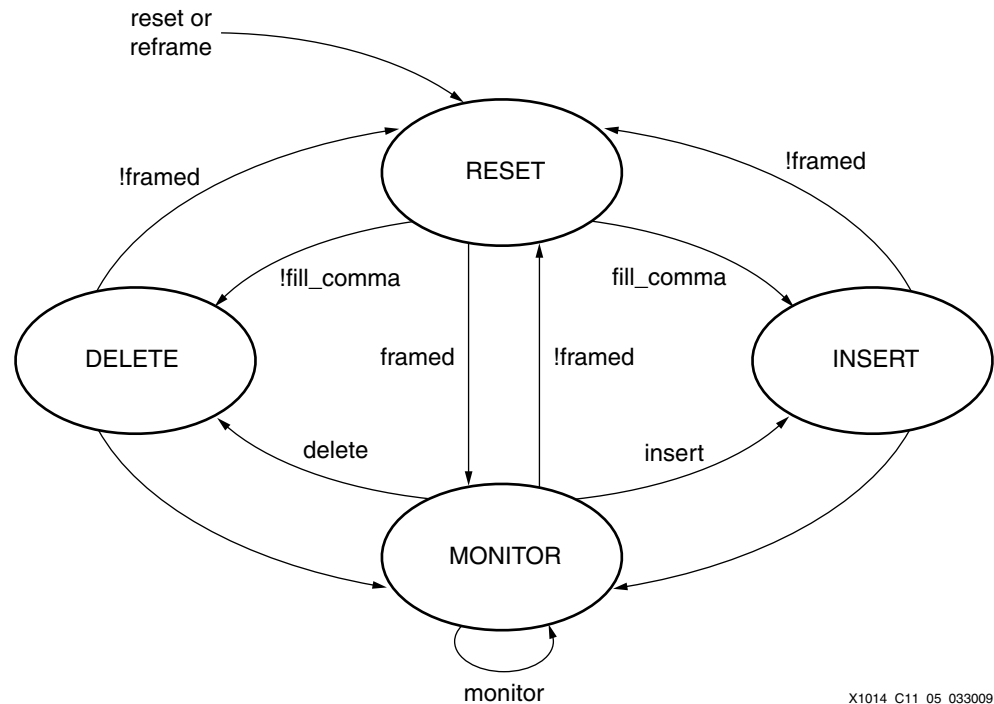
X1014_C11_04_033009

Figure 13-4: Rate-Matching FIFO Block Diagram

Xilinx FPGAs provide several different methods to implement such a FIFO. The memory element in this reference design is the Xilinx SelectRAM™ memory (block RAM). The data written to the FIFO is nine bits wide, rather than eight. The ninth bit is the K character flag output from the decoder. The 8-bit decoded data and the K character flag are combined to provide 9-bit tagged data. This method allows the passing of comma character flags through the FIFO such that the layer 2 implementation can determine when it is reading actual stream data as opposed to synchronization characters.

The FIFO also provides for a word count output that indicates the relative state of the FIFO as approaching full or approaching empty. This indicator is called a watermark. A state machine monitors the watermark of the FIFO to determine if it is close to overflowing or underflowing. Two watermark thresholds, high and low, indicate these conditions, respectively, and indicate when to insert or delete comma characters to or from the incoming data as needed to prevent the FIFO from overflowing or underflowing.

[Figure 13-5](#) shows the state diagram for this state machine.



X1014_C11_05_033009

Figure 13-5: Rate-Matching FIFO State Machine

The RESET state is used for two purposes. First, it is the safe start and return point for the state machine. The state machine is put into the RESET state by power-on reset, by the link controller issuing a reframe, or by the ASI framer indicating that the link is not framed. Second, when in the RESET state and the link is not framed, the FIFO level is monitored and comma characters are inserted and deleted as necessary by branching to the INSERT and DELETE states. This ensures that the FIFO stays filled and the layer 2 implementation always has something to read from the FIFO.

The MONITOR state monitors the high and low watermarks of the FIFO as well as the data valid output and K28.5 output flag of the decoder. If the watermarks of the FIFO indicate that the FIFO is neither too high nor too low, the state machine loops in this state, allowing writes to occur to the FIFO naturally as the decoder data valid output is asserted. If the watermark indicates that the FIFO level is too high and the current character to be written to the FIFO is a comma character, the branch to the DELETE state is taken. If the watermark indicates that the FIFO level is too low and the data valid output of the decoder is Low (indicating that there is no new data to be written), the branch to the INSERT state is taken. If the link becomes unframed in the MONITOR state, the branch to the RESET state is taken.

The INSERT state is used to insert commas into the FIFO. The data multiplexer to the FIFO input is switched to the comma constant input and the write enable is asserted, immediately writing a comma character to the FIFO on the next clock edge. The timing of the write is such that it only occurs when the data valid output of the decoder is negated, indicating an “empty” slot in the input data stream.

The DELETE state is used to delete commas from the input stream. This state negates the write enable to the FIFO at the correct time to prevent the comma character in the data stream from being written. Only comma characters in the stream are treated in this way. This is why the MONITOR state first checks to make sure that the current character is a comma and not actual MPEG data.

Rate matching using comma correction has upper and lower bounds defined by both system design restrictions and incoming data stream content. In the current design, the upper bound on rate matching is just below the maximum speed of the state machine and FIFO. The exact value cannot be easily calculated as it is dependent on environmental conditions, power supply tolerances, etc., as well as the clock rate of the incoming stream. A related rate problem is created because the incoming stream is asynchronous to the receiver and is also influenced by the same conditions. Thus, the upper limit is somewhere below the maximum operating frequency of the state machine, which is approximately 270 MHz. The lower bound is directly proportional to the comma content of the incoming stream. Commas can only be deleted when they occur. For example, in the hardware test system used to validate this design, a synthetic transport stream is used that has only 2 commas per 190 bytes, or about 1% comma content. If the incoming data rate for this stream is 27 MHz, the lower bound on rate matching is $27 - (27 \times 0.01) = 26.73$ MHz. Hardware testing validates this result as well. A stream with a higher concentration of commas allows for a proportionally lower limit on rate matching.

Clocking

Only a single clock is needed by the receiver implementation. The internal PLL in the Virtex-5 FPGA can be used to create the clock from a single clock input. In this case, the clock input is 135 MHz.

The DRU uses a ~274.05 MHz 0-degree phase clock. The DRU creates a 10-bit data word and a data-valid signal that is used to clock the downstream datapaths at an effective rate of 1/10th of the 274.05 MHz rate. The comma correction FIFO controller logic runs at the full 274.05 MHz rate.

Reference Design

A complete reference design using this receiver verified in hardware on the ML571 SDV board is available at <https://secure.xilinx.com/webreg/clickthrough.do?cid=119758>. Open the ZIP archive and extract the file `xapp1014_sec4_ASI_LVDS.zip`.

Design Hierarchy

Figure 13-6 shows the design hierarchy, including the DVB-ASI receiver reference design. Each block in Figure 13-6 is associated with a corresponding HDL module whose name is shown in parentheses.

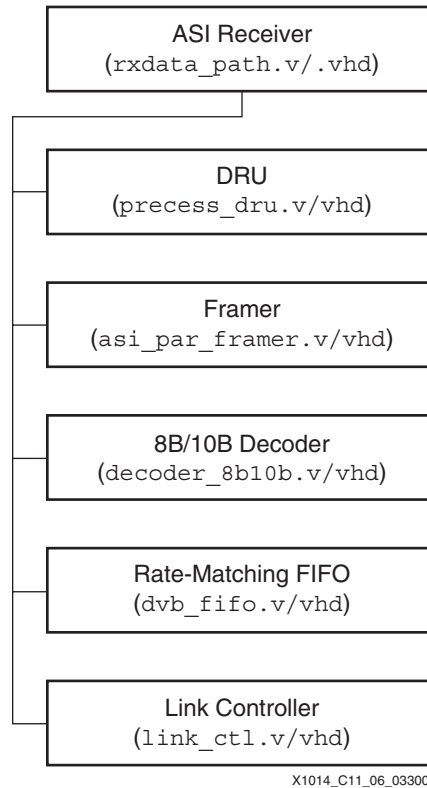


Figure 13-6: Reference Design Hierarchy

DVB-ASI Receiver Ports

Table 13-1 describes the I/O ports of the receiver module `rxdata_path.v/vhd`, which is the top level of the ASI receiver. All signals in the table use positive logic in which assertion denotes a logic 1 and negation a logic 0.

Table 13-1: ASI Receiver Ports

Port	I/O	Description
sclk	In	This is a serial clock input 1.5% faster than the 270 Mb/s data rate. This is the main receiver clock.
user_clk	In	This is a user-defined clock for the read side of the rate-matching FIFO.
reset	In	This is an active-High, synchronous reset that resets the entire receiver. It is synchronous to sclk.
asi_serin	In	This is the ASI serial data input.
rx_fifo_rden	In	This is a user read enable for the rate-matching FIFO. It must be synchronous to user_clk.

Table 13-1: ASI Receiver Ports (Cont'd)

Port	I/O	Description
reframe_req	In	This input port carries requests for immediate reframe of the link and is synchronous to sclk. The reframe_req input forces the framer to reacquire the comma sequence and restarts the link controller sequence. This might cause initial disparity or code errors, but these are normal and should not continue after the link is reframed.
sync_mode	In	This input determines if the framer frames on a single comma character or two consecutive commas within a 5-byte window. It should be set High for a two-comma synchronization sequence and Low for a single-comma sequence.
dout_8b	Out	This 8-bit output data from the rate-matching FIFO is the decoded output of the receiver. It is synchronous to user_clk.
kchar_out	Out	This is the ninth bit output from the rate-matching FIFO to indicate whether the current 8-bit word on dout_8b is data or a comma character. It is synchronous to user_clk, and assertion indicates a comma character.
fifo_empty	Out	This is the rate-matching FIFO handshake signal synchronous to user_clk. It is included primarily for debugging. Handshaking with the FIFO is not needed because it is an elastic buffer. Assertion indicates that the rate-matching FIFO is empty.
code_err	Out	This is the error output from the 8B/10B decoder. Assertion indicates that an invalid 10-bit input was detected. The code_err output is synchronous to sclk.
disp_err	Out	This is the error output from the 8B/10B decoder. Assertion indicates that the input data has violated the 8B/10B disparity rules. The disp_err output is synchronous to sclk.
framed_asi	Out	This output port indicates the link status. Assertion of framed_asi indicates that the desired comma sequence has been detected and the link is framed at the layer 1 level (synchronization byte detection).

FPGA Resource Usage

Table 13-2 shows the overall resource usage for the receiver in the Virtex-5 FPGA. Utilization numbers are obtained using XST 9.2i SP1 with the default settings. In this example, a PLL is used to create all the clocks used by the receiver.

Table 13-2: FPGA Resource Utilization

Module	Flip-Flops	LUTs	Block RAMs	BUFG/DCM/PLL (1)
Complete Receiver (rxdata_path.v/vhd)	353	377	1	1/0/1

Notes:

1. Clocks can be used for multiple RX channels.

Implementation Instructions

For the design to process correctly through the tools, the Keep Hierarchy option in XST should be set to **Yes**. The design was implemented using the ISE® software, version 9.2i SP1. The FIFOs used in this design were created using the CORE Generator™ tool.

Generating the Rate-Matching SelectRAM Memory FIFO

This section describes the parameter settings needed to generate each of the FIFO cores manually, if required. The SelectRAM memory FIFO used in the implementation of the rate-matching FIFO is a 2K x 9 asynchronous block RAM FIFO. It can be created in the CORE Generator tool by applying the parameter settings shown in [Table 13-3](#) to the FIFO Generator.

Table 13-3: Rate-Matching FIFO CORE Generator Software Parameters

Parameter	Value
Component Name	dcfifo_2kx9
FIFO Implementation	Independent clocks (RD_CLK, WR_CLK) block RAM
Read Mode	Standard FIFO
Write Width	9
Write Depth	2048
Read Width	9
Almost Full Flag	Disabled
Almost Empty Flag	Disabled
Write Acknowledge Flag	Disabled
Overflow Flag	Disabled
Valid Flag	Disabled
Underflow Flag	Disabled
Reset	Enabled
Programmable Full Type	No programmable full threshold
Programmable Empty Type	No programmable empty threshold
Write Data Count	Enabled
Write Data Count Width	11
Read Data Count	Enabled
Read Data Count Width	11

Conclusion

This chapter describes a DVB-ASI receiver for Virtex-5 devices. By utilizing advanced silicon features such as DCMs, block RAM, SelectIO technology, and DDR clocking, a robust, 270 Mb/s ASI receiver is implemented. A complete reference design, verified in hardware on the ML571 SDV board, is also available. See [Chapter 15, DVB-ASI Layer 1 and 2 Pass-through Demonstration Design](#) for details.

DVB-ASI Layer 1 and 2 Transmitter

Summary

DVB-ASI provides an industry-standard method for transmitting MPEG-2 compressed video over high-speed asynchronous serial interfaces. This chapter presents a DVB-ASI transmitter design for the Virtex®-5 FPGA that uses Xilinx® SelectIO™ technology. A robust 270 Mb/s ASI transmitter is implemented in a cost-effective manner using advanced techniques, silicon features, and tool capabilities.

It is recommended that the reader first read [Chapter 12, DVB-ASI Introduction and Layer 0 Implementation](#) to become acquainted with the material presented in this chapter.

ASI Transmitter

The ASI layer 1 transmitter consists of an 8B/10B encoder and a serializer. [Figure 14-1](#) shows the ASI transmitter block diagram.

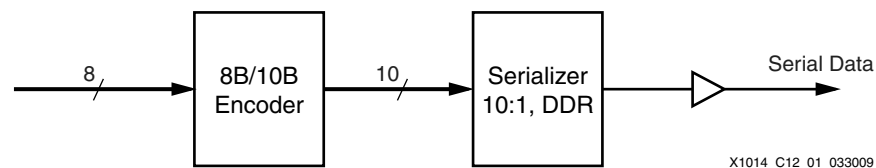


Figure 14-1: ASI Transmitter Architecture

8B/10B Encoder

The 8B/10B encoder encodes the 8-bit input video data into 10-bit words according to the 8B/10B encoding rules. The encoder is responsible not only for creating valid code outputs from the inputs, but also for ensuring that the running disparity of the outputs is correct. 8B/10B codes that are not balanced (i.e., have more than five 1s or 0s) must be transmitted with alternating polarity to ensure a DC-balanced transmission. In addition, some special cases require complements of the balanced subblocks to be used to enforce run length rules. A complete description of this is beyond the scope of this application note. More information is provided in technical publications [\[Ref 9\]](#).

For each of the 255 possible 8-bit inputs, there are 255 possible data outputs plus the special K characters for a total of 268 possible outputs. A special control pin differentiates a K code input from a regular data word input. Assertion of this input along with the data causes the encoder to create a K code output. For example, the ASI comma character is a K28.5 character. The K28.5 8-bit input value is 0xBC, which is also a valid data value. To encode the K28.5 comma character, the 0xBC value is presented to the encoder inputs and

the `kchar_in` input is asserted. All inputs to the encoder are synchronous to the clock input and are controlled by the clock enable. The encoder has two stages of pipeline delay.

Encoding is a two-step process. Each 8-bit input word is separated into a 5-bit and a 3-bit subblock. Each subblock is then encoded into the negative- or zero-disparity 6-bit subblock output, and the positive- or zero-disparity 4-bit subblock output. Because the final output of the decoder is dependent upon the running disparity, the encoder need only encode one case or the other and then complement the output, as needed, based on the current running disparity.

Each subblock output is fed into a 2:1 multiplexer that selects either the current encoding or the complement, depending on the running disparity. The running disparity is handled the same as in the decoder. In the case of the 4-bit subblock, additional logic is needed to handle some special cases related to run lengths.

Serializer

The final logic block of the transmitter is the serializer. In this reference design example, the serializer provides a 10:1 serialization ratio. To make the serializer more portable to other Xilinx FPGAs, it uses a double data rate (DDR) or half bit-rate clock instead of a full bit-rate clock. The logic shifts two bits per clock cycle to the output DDR register, which then shifts the final output data. Thus, the serialization takes place in two stages. The first stage slices the 10-bit input word into five sequential 2-bit slices. In the second stage, the DDR register provides the final 2:1 serialization.

Clocking

Only one clock is needed by the transmitter implementation. The internal PLL or DCM of the Virtex-5 FPGA can be used to create the clock from a single clock input. The transmitter is clocked by a 135 MHz clock with a clock enable for all sections of the transmitter datapath that run at the word rate of 27 MHz.

Reference Design

A complete reference design using the ASI transmitter verified in hardware on the ML571 SDV board is available at <https://secure.xilinx.com/webreg/clickthrough.do?cid=119758>. Open the ZIP archive and extract the file `xapp1014_sec4_ASI_LVDS.zip`.

Design Hierarchy

Figure 14-2 shows the design hierarchy, including the top level of the reference design. Each block in Figure 14-2 is associated with a corresponding HDL module whose name is shown in parentheses.

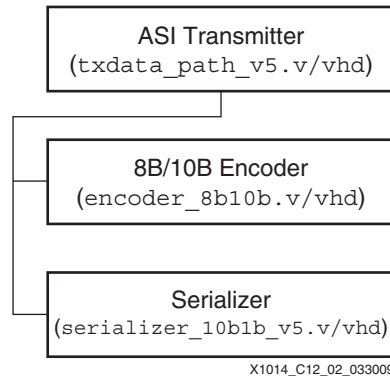


Figure 14-2: ASI Transmitter Reference Design Hierarchy

ASI Transmitter Ports

Table 14-1 shows the I/O ports of the ASI transmitter. All signals in the table use positive logic in which assertion indicates a logic 1, and negation a logic 0.

Table 14-1: ASI Transmitter Ports

Port	I/O	Description
clk	In	This is the half bit-rate clock (135 MHz), which is the main clock for the transmitter. All inputs and outputs are synchronous to this clock.
ce	In	This is the word rate (27 MHz) clock enable for the parallel section of the transmitter datapath.
rst	In	This is the synchronous reset input.
din_8b	In	This is the 8-bit unencoded ASI data input.
kchar_in	In	This is the control bit to control encoding of the data as a standard code word or as a K code. Assertion of this bit forces the encoder to code the current 8-bit input value as a K code.
force_disp	In	This input is reserved for future use and should be statically negated in design instantiation.
asi_out	Out	This is the encoded ASI data serialized to the 270 Mb/s serial stream.

FPGA Resource Usage

Table 14-2 shows the overall resource usage for the transmitter in the Virtex-5 FPGA. Utilization numbers were obtained using XST 9.2i SP1 with the default settings. In this example, a DCM generates the clocks for the transmitter that consists of a single 135 MHz clock with a clock enable at a 27 MHz rate. Thus, only one 135 MHz clock is needed.

Table 14-2: FPGA Resource Utilization

Module	Flip-Flops	LUTs	Block RAMs	BUFG/DCM/PLL ⁽¹⁾
Complete TX (txdata_path.v/vhd)	46	35	0	1/1/0

Notes:

1. Clocks can be used for multiple TX channels.

Implementation Instructions

For the design to process correctly through the tools, the Keep Hierarchy option in XST should be set to Yes. The design was implemented using the ISE® software, version 9.2i SP1. The FIFOs used in this design were created using the CORE Generator™ tool.

Generating the Rate-Matching SelectRAM Memory FIFO

This section describes the parameter settings needed to generate each of the FIFO cores manually, if needed. The SelectRAM™ memory FIFO used in the implementation of the rate-matching FIFO is a 2K x 9 asynchronous block RAM FIFO. This FIFO can be created in the CORE Generator tool by applying the parameter settings shown in Table 14-3 to the FIFO Generator.

Table 14-3: Rate-Matching FIFO CORE Generator Software Parameters

Parameter	Value
Component Name	dcfifo_2kx9
FIFO Implementation	Independent clocks (RD_CLK, WR_CLK) block RAM
Read Mode	Standard FIFO
Write Width	9
Write Depth	2048
Read Width	9
Almost Full Flag	Disabled
Almost Empty Flag	Disabled
Write Acknowledge Flag	Disabled
Overflow Flag	Disabled
Valid Flag	Disabled
Underflow Flag	Disabled
Reset	Enabled
Programmable Full Type	No programmable full threshold
Programmable Empty Type	No programmable empty threshold
Write Data Count	Enabled
Write Data Count Width	11

Table 14-3: Rate-Matching FIFO CORE Generator Software Parameters (Cont'd)

Parameter	Value
Read Data Count	Enabled
Read Data Count Width	11

Conclusion

This chapter describes a DVB-ASI transmitter for Virtex-5 devices. By utilizing advanced silicon features such as DCMs, block RAM, SelectIO technology, and DDR clocking, a robust, 270 Mb/s ASI transmitter can be implemented. A complete reference design verified in hardware on the ML571 SDV board is also available. See [Chapter 15, DVB-ASI Layer 1 and 2 Pass-through Demonstration Design](#) for details.

DVB-ASI Layer 1 and 2 Pass-through Demonstration Design

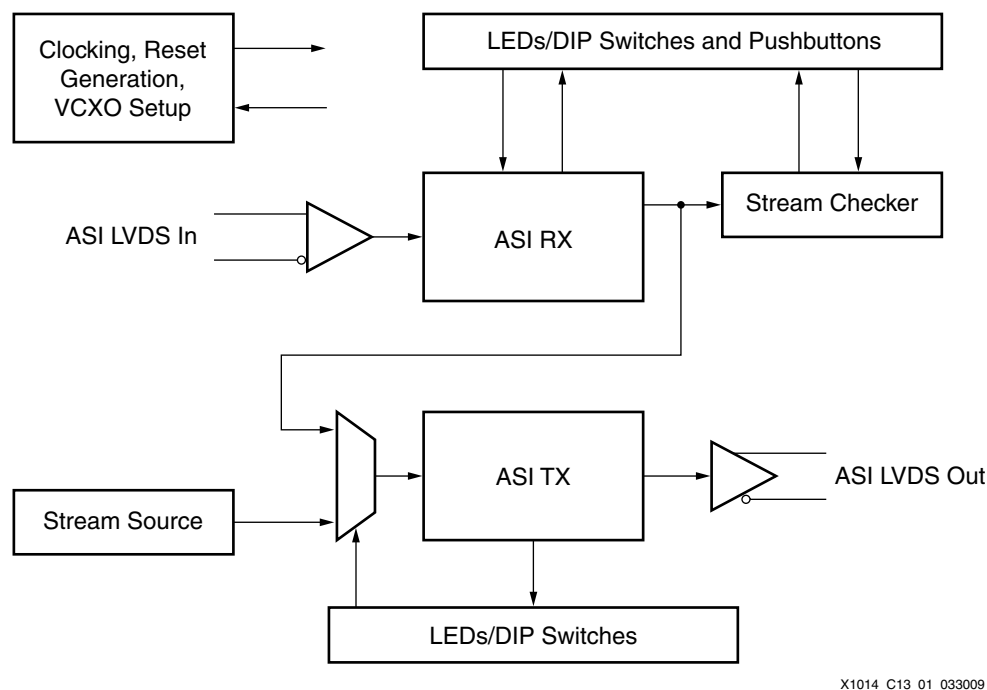
Summary

DVB-ASI provides an industry-standard method for transmitting MPEG-2 compressed video over high-speed asynchronous serial interfaces. This chapter presents a DVB-ASI pass-through design for the Virtex®-5 FPGA that uses Xilinx® SelectIO™ technology.

It is recommended that the reader first read [Chapter 12, DVB-ASI Introduction and Layer 0 Implementation](#), to become acquainted with the material presented in this chapter. In addition, this chapter only discusses the integration of the DVB-ASI receiver and transmitter into a complete, top-level reference design implementing pass-through functionality. For details on the individual receiver and transmitter, refer to [Chapter 13, DVB-ASI Layer 1 and 2 Receiver](#) and [Chapter 14, DVB-ASI Layer 1 and 2 Transmitter](#), respectively.

Reference Design

A complete DVB-ASI pass-through demonstration design for the ML571 SDV demonstration board is shown in [Figure 15-1](#). The ASI RX and TX blocks are already discussed in earlier chapters and are not covered here. Additional circuitry for generating clocks, resets, and the top-level design are discussed in this section.

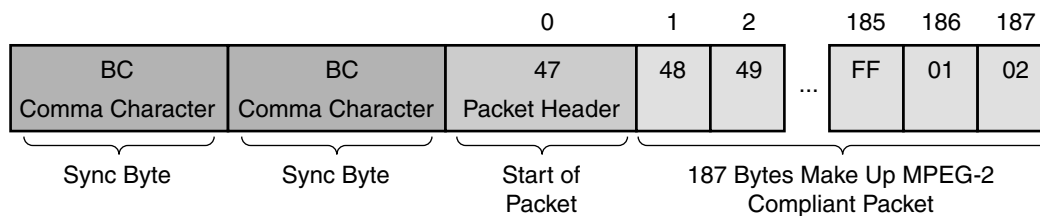


X1014_C13_01_033009

Figure 15-1: ML571 DVB-ASI Pass-through Demonstration Design

The reference design supports a pass-through mode in which the recovered data from the receiver is passed to the transmitter for re-encoding and serialization. This allows the reference design hardware to be inserted between a customer's own ASI source and destination for quick and easy testing. This method is the most popular for testing and demonstrating an ASI solution. The hardware reference design was tested using this method. An input to the top-level module can select this mode of operation.

The transport stream source illustrates a simple layer 2 transmitter implementation and provides a source for loopback testing on the same design if no other ASI sources are available. The top-level module generates 188-byte packets separated by two K28.5 commas. Each packet byte consists of an incrementing count that begins at 47 hex. The pattern repeats continually, as shown in Figure 15-2. This same pattern is used by the checker to verify error-free data transmission.



X1014_C13_02_033009

Figure 15-2: Stream Source MPEG-2 Packet

The stream checker is used as an example of a simple layer 2 receiver application illustrating how to read data from the rate-matching FIFO, discard commas, and interpret the data. The stream checker reads a 9-bit character from the FIFO. The first eight bits of this 9-bit character contain the 8-bit output from the decoder and FIFO controller, and the ninth bit is the K code flag from the decoder. The stream checker first checks the ninth bit

to determine if the word read from the FIFO contains real data or is a comma. If the ninth bit is a comma, it is ignored and is not used for the comparison check. If it is actual data, the comparison continues.

The stream checker then waits for the MPEG packet header byte value of 47 hex. At this point, a watchdog timer is also started. If the MPEG header byte is not detected within two MPEG packets, it is assumed that the link has a problem, and an error is asserted. After the MPEG packet header is found, each byte of data from the FIFO (commas excluded) is compared. A single byte mismatch causes the checker to assert its error output. The checking process is continuous.

The reference design uses the voltage-controlled crystal oscillator (VCXO) of the ML571 board to create the necessary fixed-frequency input clock. The VCXO setup module initializes the digital-to-analog converter (DAC) to the midpoint on start-up and sets the output frequency to 135 MHz. After the DAC is initialized, the DCMs and PLL are reset to ensure a proper lock to the clock source. The DAC is never changed after the initialization to the midpoint at start-up.

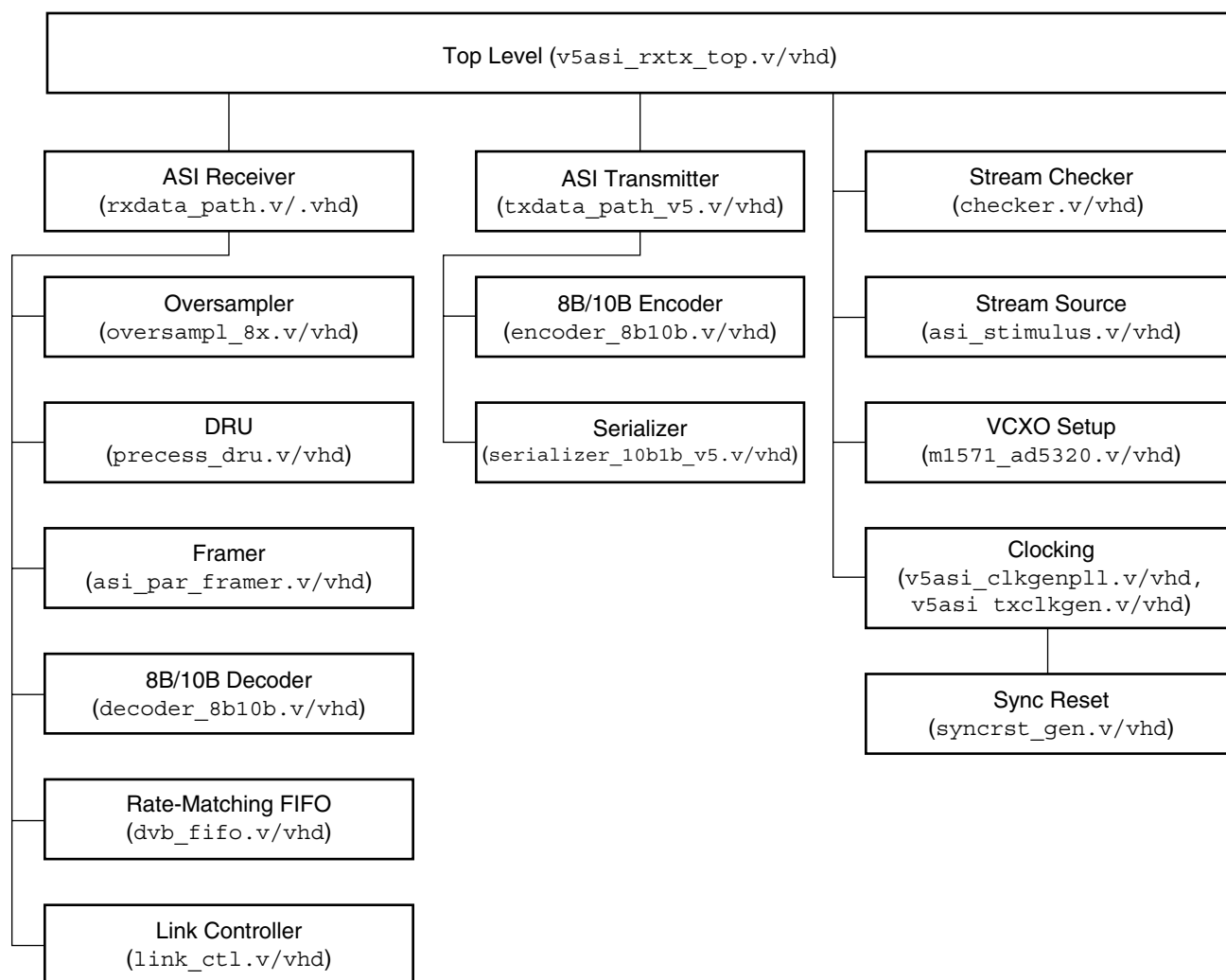
Virtex-5 FPGA DCMs and PLLs are used to generate the required clocks. In this reference design, the input clock is 135 MHz. However, any suitable low-jitter clock source can be used with the Virtex-5 FPGA DCMs or PLLs to provide the necessary clock phases and frequencies.

The Sync Reset modules ensure that logic that requires a reset input is held in reset until after the DCMs lock. All modules use synchronous resets. The Sync Reset module also takes an external reset input. Because of this module, any reset of the DCMs and PLLs also forces the logic to be reset.

A complete example of using the reference design on the ML571 board is detailed in [Running the Demonstration Design, page 338](#).

Design Hierarchy

[Figure 15-3](#) shows the design hierarchy, including the top level of the reference design. Each block in [Figure 15-3](#) is associated with a corresponding HDL module whose name is shown in parentheses.



X1014_C13_03_033009

Figure 15-3: Reference Design Hierarchy

FPGA Resource Usage

Table 15-1 shows the overall resource usage for the transmitter and receiver in the Virtex-5 FPGA. Utilization numbers were obtained using XST 9.2i SP1 with the default settings. In this example, a PLL creates all the clocks used by the receiver, whereas a DCM generates the clocks for the transmitter. In addition to the 274.05 MHz RX clock, a 33 MHz clock is used to interface to the onboard DAC. In this example, a single 135 MHz clock with a clock enable at a 27 MHz rate is used for the transmitter. Thus, the minimum number of clocks needed is two (274.05 MHz and 135 MHz).

Table 15-1: FPGA Resource Utilization

Module	Flip-Flops	LUTs	Block RAMs	BUFG/DCM/PLL (1)
Complete RX (rxdata_path.v/vhd)	353	377	1	1/0/1
Complete TX (txdata_path.v/vhd)	46	35	N/A	1/1/0

Notes:

1. Clocks can be used for multiple RX and TX channels.

Implementation Instructions

For the design to process correctly through the tools, the Keep Hierarchy option in XST should be **Yes**. The design was implemented using the ISE® software, version 9.2i SP1. The FIFOs used in this design were created using the Xilinx CORE Generator™ software. The following section discusses the parameter settings needed to generate each of the FIFO cores manually, if needed.

Generating the Rate-Matching SelectRAM Memory FIFO

The SelectRAM™ memory used in the implementation of the rate-matching FIFO is a 2K x 9 asynchronous block RAM CORE Generator software FIFO. It can be created in CORE Generator software by using the parameter settings shown in Table 15-2 in the FIFO generator.

Table 15-2: Rate-Matching FIFO CORE Generator Software Parameters

Parameter	Value
Component Name	dcfifo_2kx9
FIFO Implementation	Independent clocks (RD_CLK, WR_CLK) block RAM
Read Mode	Standard FIFO
Write Width	9
Write Depth	2048
Read Width	9
Almost Full Flag	Disabled
Almost Empty Flag	Disabled
Write Acknowledge Flag	Disabled
Overflow Flag	Disabled
Valid Flag	Disabled
Underflow Flag	Disabled
Reset	Enabled
Programmable Full Type	No programmable full threshold
Programmable Empty Type	No programmable empty threshold
Write Data Count	Enabled

Table 15-2: Rate-Matching FIFO CORE Generator Software Parameters (Cont'd)

Parameter	Value
Write Data Count Width	11
Read Data Count	Enabled
Read Data Count Width	11

Running the Demonstration Design

This section describes the equipment needed to set up the demonstration design, such as cables and boards. It also contains details on running the demonstration, including interpretation of LEDs and DIP switch settings.

Required Equipment

These items are needed for the demonstration system:

- ML571 SDV board with power supply
- Platform USB cable for bitstream loading
- One or two 75 Ω coaxial cables with BNC connectors, depending on the setup for loopback or pass-through modes
- Bitstream file included in the original distribution with `v5_as_demo.bit`

Setup Instructions

The demonstration system is set up in this manner:

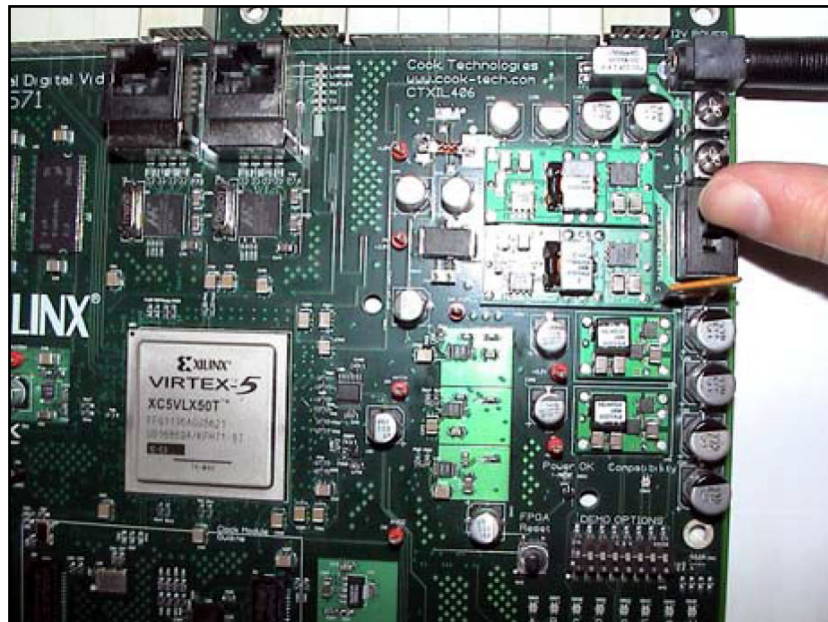
1. Connect the ML571 power supply.
2. Connect the platform USB cable to the ML571 board, as shown in [Figure 15-4](#).



X1014_C13_04_033009

Figure 15-4: ML571 USB Cable Connection

3. Turn on the power to the ML571 board using the switch shown in Figure 15-5.

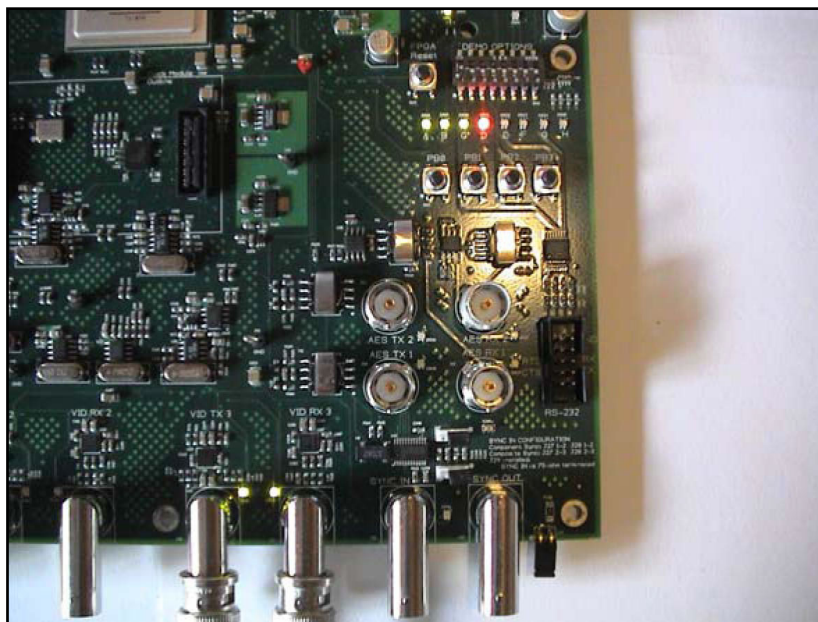


X1014_C13_05_033009

Figure 15-5: ML571 Power

4. Run the iMPACT tool and load the `v5asi_demo.bit` bitstream into the device. Ensure that the device configures successfully before going to step 5.
5. Connect the coaxial cables depending on the desired mode of operation. For pass-through mode, two cables are required. Connect one cable from the VID TX3 BNC output to the downstream ASI device. Connect the other cable from the VID RX3

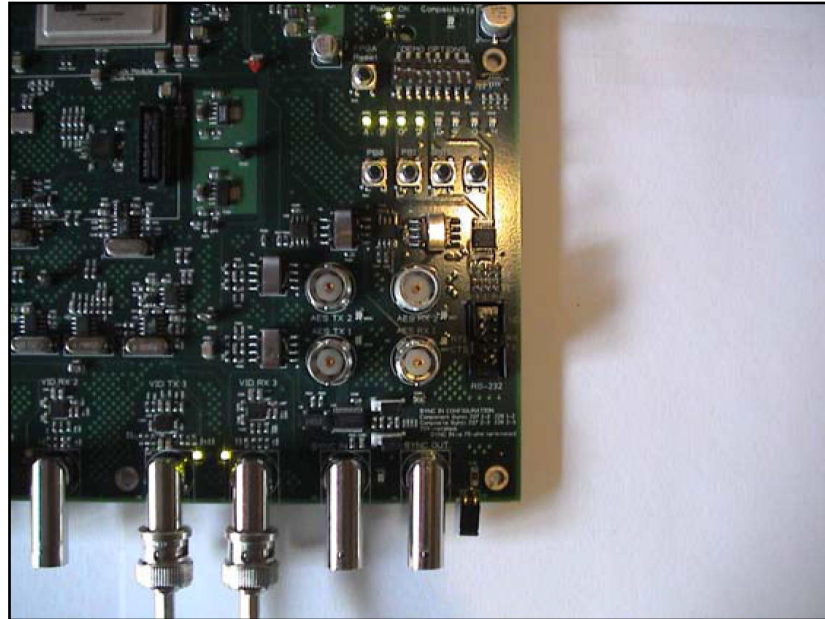
BNC connector input to the desired upstream ASI device. Set DIP switch 2 to OFF to select pass-through mode. Ensure that the LED next to the VID TX3 output is steady and not flashing. Press PB2 to clear all errors. When in pass-through mode and not using an external stimulus stream that is compatible with the demonstration platform's pattern checker, LED D is red, indicating pattern errors. In this case, the error is expected and can be ignored. The test of the design determines whether or not the MPEG-2 stream(s) on the VID RX3 input are successfully being transferred to the VID TX3 output and to the downstream ASI device. However, no other error conditions should be seen. For this mode, the LEDs on the ML571 board should look like [Figure 15-6](#).



X1014_C13_06_033009

Figure 15-6: ML571 ASI Operating in Pass-through Mode

6. If loopback mode is desired, connect a single cable from the VID TX3 output to the VID RX3 input and set DIP switch 2 to ON to select stimulus/loopback mode. Ensure that the LED next to the VID TX3 output is flashing. Press and release PB2 to clear all errors. If everything is running correctly, the LEDs on the ML571 board should look like [Figure 15-7](#). See [Design Summary](#) for an explanation of what each LED means.



X1014_C13_07_033009

Figure 15-7: ML571 ASI Operating in Loopback Mode

Design Summary

The reference design included with this demonstration provides two primary modes of operation. The first, called pass-through mode, allows for demonstrations by inserting the ML571 board between two other ASI devices. In this mode, the design recovers and decodes the data using the ASI receiver, and the LEDs operate as described later in this section. The transmitter output transmits the same data recovered by the receiver, forming a simple pass-through operation. Pass-through mode is the most popular method of testing the design. If data can pass through the receiver and transmitter without error, it can be assumed that both the receiver and transmitter work correctly.

The second mode, called test mode, disconnects the transmitter from the recovered data and sources the transmitter input from the internal stimulus generator. The stimulus generator produces synthetic MPEG-2 packets that are compatible with the internal pattern checker for this design. The ChipScope™ analyzer can be used to display this synthetic data, or a cable can be used to loop back the ML571 transmitter to the ML571 receiver. The LEDs operate as described later in this section, and the receiver recovers, decodes, and checks the data.

To summarize, the receiver first looks for the ASI comma pattern to frame to. After this pattern is detected, the receiver monitors the disparity and code error outputs of the 8B/10B decoder to ensure that the link is valid. A successful synchronization to the incoming data stream is indicated by the LED next to the VID RX3 BNC connector being green. If the LED is red, the comma sequence has not been detected, there are errors from the decoder, or both. The link monitoring is continuous.

LEDs B and C indicate disparity and code errors, respectively, from the decoder. When the LEDs are green, no errors have been detected. The LEDs are persistent, so that if even one error is detected, the LED changes to red and stays red. PB2 should be pressed to clear the LEDs. LED D indicates pattern check errors from the pattern checker. The pattern checker removes incoming commas from the data stream, looks for the MPEG-2 header byte, and

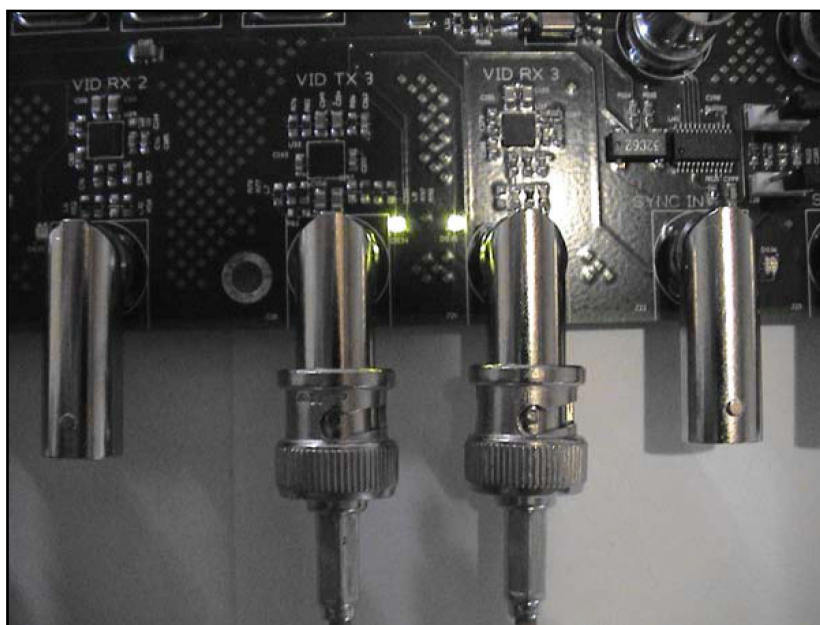
checks the incoming data against the expected pattern. A watchdog timer is reset each time an MPEG-2 header byte is detected. If a header byte is not detected, the error LED turns on. After the start of packet indicator has been detected, every byte for the next 188 bytes is checked. When a new header byte is seen, the checker resets and the process begins again. Like LEDs B and C, the error LED is persistent and PB2 should be pressed to clear this LED. Using these persistent LEDs allows for long-term unattended testing, if desired.

LED A indicates a DCM/PLL lock condition. If this LED is green, the DCMs and PLLs are locked and functioning properly. If it is red, the DCMs are not locked. The DCMs can be reset by pressing PB1. When this is done, errors are also seen on the other LEDs. PB2 should be pressed to clear these LEDs. LED A is not persistent.

PB4 is used to issue a manual reframe request to the receiver. When this button is pressed, the circuit goes through the link synchronization process again. The 8B/10B decoder is also reinitialized, making it possible to get disparity and/or code errors. As before, PB2 should be pressed and held briefly to clear the errors.

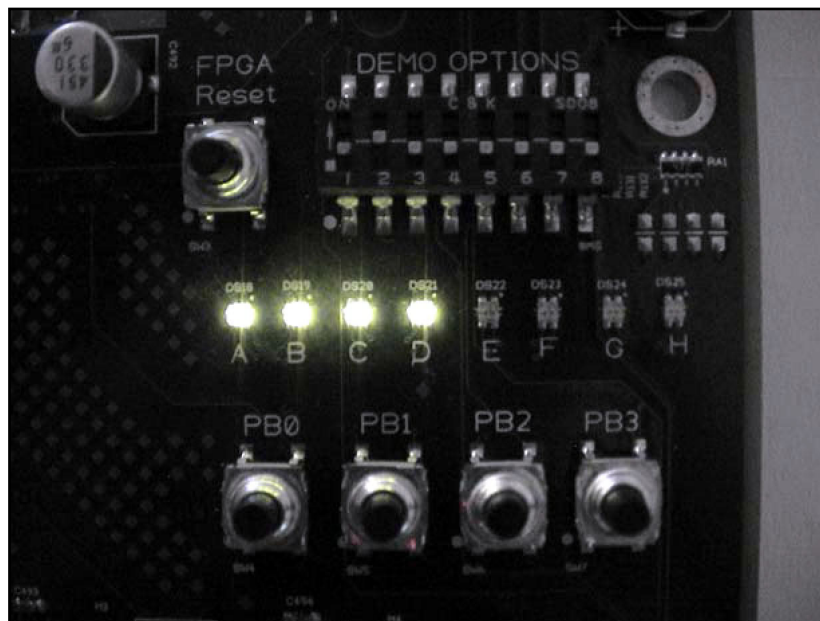
When the design is running in pass-through mode, the VID TX3 LED is steady. When the design is operating in internal stimulus mode, the VID TX3 LED blinks at about 1 Hz.

Figure 15-8 and Figure 15-9 show the LED locations.



X1014_C13_08_033009

Figure 15-8: ML571 VID TX3 and VID RX3 LEDs



X1014_C13_09_033009

Figure 15-9: ML571 LEDs A–D, DIP Switches, and Pushbuttons

Conclusion

This application note describes a DVB-ASI pass-through demonstration design for Virtex-5 devices.

Section V:

AES Digital Audio

***Audio/Video Connectivity Solutions
for Virtex-5 FPGAs***

Introduction to Digital Audio for Video Broadcasting

Audio and video are equally important elements of the programming content in the broadcast industry. At different places in the broadcast studio or production center, audio and video are combined into one digital signal. At other places, audio is transported and processed separately from video. Thus, there are standards for transporting digital audio separately from video and other standards for transporting digital audio embedded in the digital video signal.

The AES3 standard from the Audio Engineering Society is the professional digital audio transport standard. Each AES3 digital signal carries a pair of digital audio signals over twisted pair or coaxial cable.

Various SMPTE standards describe how to “embed” digital audio into the unused horizontal blanking intervals of digital video signals. These standards are based on the digital audio format specified by AES3.

Introduction to the AES3 Digital Audio Standard

AES3 is a professional standard for transporting digital audio serially over twisted pair or coaxial cable. Each AES3 audio link carries a stereo pair of digital audio channels and supports various audio sampling rates. AES3 is also called Audio Engineering Society/European Broadcasting Union (AES/EBU).

The consumer version of AES3 is called Sony/Philips Digital Interconnect Format (S/PDIF). S/PDIF is commonly used to move digital audio between pieces of consumer electronic equipment such as between a DVD player and a surround-sound receiver. The data format and data rates of S/PDIF are the same as AES3. S/PDIF differs from AES3 in the electrical and physical (cable and connector) specifications.

AES3 is defined by the AES3-2003 document from the Audio Engineering Society [\[Ref 10\]](#). The nearly identical specification from the EBU (Tech. 3250-E) describes AES/EBU [\[Ref 11\]](#).

Data Format

An AES3 interface carries two linear pulse code modulation (PCM) audio channels interleaved together in one serial bitstream. Additional SMPTE standards describe how to transport non-PCM multi-channel encoded audio (surround sound) on AES3 interfaces. S/PDIF links commonly carry surround-sound audio. However, none of these multi-channel formats are officially part of the AES3 specification. This chapter and the reference designs in this section of the application note focus on the standard two-channel PCM audio described in the AES3 specification.

Subframes, Frames, and Blocks

The basic data structure of AES3 is called a subframe. Each 32-bit subframe carries a single audio sample word for one audio channel along with a few other bits of information (Figure 16-1). A subframe begins with a 4-bit preamble. The audio word can be either 24 bits or 20 bits. Following the audio word are the valid (V), user data (U), channel status (C), and parity (P) bits. Two consecutive subframes, one for each of the two audio channels, form a complete frame. The subframe for channel 1 is always sent before the subframe for channel 2.

Figure 16-1 shows the bit order of the subframe from left to right as LSB to MSB. This corresponds to the order in which the bits are sent on the AES3 interfaces (LSB first).

Frames are grouped together in blocks of 192 frames. This grouping of frames into blocks serves to define the beginning and ending points for the sequence of channel status and user data bits. The channel status information for each channel is 192 bits long. With one channel status bit for each channel included in each frame, it takes 192 frames to transmit the 192 bits of channel status for each channel. Likewise, the user data for each channel is 192 bits long with one user data bit for each channel present in each frame. The first frame of a block contains the LSBs of the channel status and user data for each channel. The MSBs are located in the last frame of the block.

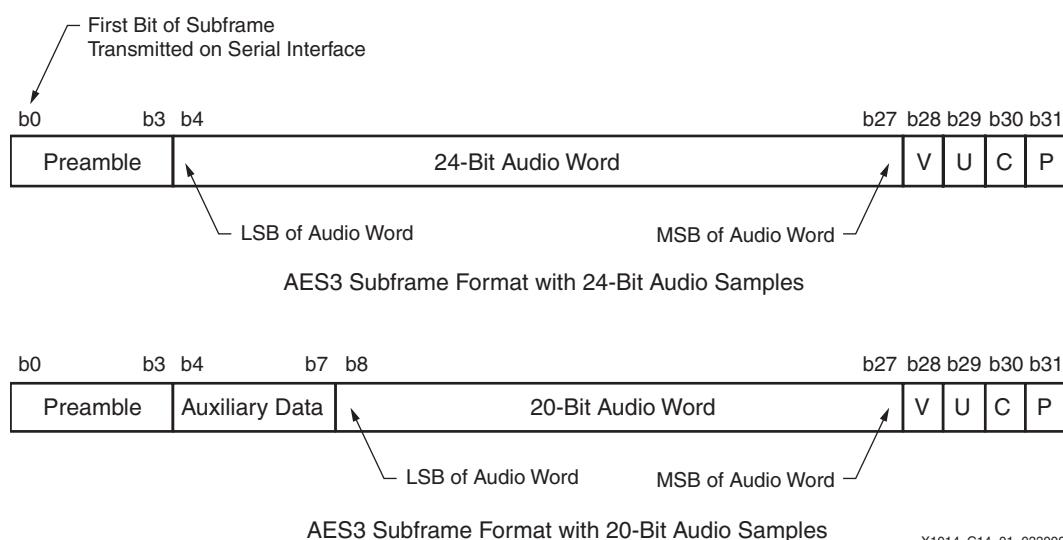


Figure 16-1: AES3 Subframe Format

Preambles

Preambles are unique sequences that do not occur anywhere except in the preamble portion of each subframe. Because preambles are unique, an AES3 receiver can find them and synchronize to the AES3 bitstream. Because there is a preamble at the beginning of each subframe, an AES3 receiver can quickly synchronize to the bitstream.

The first frame of a block is identified by a different preamble in the channel 1 subframe. In all but the first frame of a block, subframe 1 begins with an X preamble. In the first frame of a block, subframe 1 begins with a Z preamble instead of an X preamble. By detecting the Z preamble, an AES3 receiver can locate the LSB of the channel status and user data and thereby organize this information in the correct order. Every subframe 2 begins with a Y preamble.

Thus, the preambles serve three purposes:

- Unique preamble sequences allow the receiver to identify the beginning of each subframe.
- Different preambles distinguish subframe 1 from subframe 2 in each frame.
- The Z preamble, occurring only in subframe 1 of the first frame of a block, allows the receiver to locate the LSB of the channel status and user data.

Biphase-Mark Encoding

AES3 uses biphase-mark encoding. In this encoding scheme, each bit is encoded into a symbol consisting of two states. The first state of a symbol is the inverse of the second state of the previously transmitted symbol. For example, if the second state of the previous symbol was a 1, the first state of the next symbol is 0. This ensures that there is always a state transition between symbols. The second state of a symbol is the same as the first state if the bit being encoded is a 0. If the bit being encoded is a 1, the second state of the symbol is the inverse of the first state.

The preambles at the beginning of each subframe are not biphase-mark encoded. They violate the rules of biphase-mark encoding, making them easily identifiable in the bitstream. Each preamble consists of eight consecutive states, which is the equivalent of four bit times.

Valid Bit

The valid bit in each subframe indicates whether the audio sample of the subframe is valid. If the valid bit is 0, the sample is valid. If the valid bit is 1, the sample is invalid. So, for example, if only one channel of audio is being carried by an AES3 bitstream, the valid bits in all subframes of the second audio channel would be set to 1 to indicate that this audio channel is not valid.

Channel Status

The channel status information carries sideband data about each audio channel. The type of information carried by the channel status includes the audio sample rate and the number of bits in the audio sample word. The AES3 standard breaks down the 192 bits of channel status information into 24 bytes of 8 bits each and defines the purpose of each bit. Not every AES3 bitstream uses all 192 channel status bits and the AES3 standard identifies several subsets that are commonly used.

The last byte of channel status information is commonly used to hold a CRC word calculated from the other channel status bits. However, the CRC is not present in all AES3 bitstreams and should be filled with zeros if it is not used.

User Data

User data is similar to channel status data. The user data consists of 192 bits for each audio channel. User data can carry any type of information. The AES18 document defines a standard way to create and insert data packets into the user data bits of an AES3 bitstream [Ref 12].

Parity Bit

Each subframe ends with a parity bit. This even parity bit is generated so that the 28 bits of the subframe, not including the preamble, have an even number of ones and zeros.

Data Rate

The data rate of an AES3 interface is dependent upon the audio sampling rate being carried. In theory, the audio can be sampled at any rate, so the resulting AES3 bit rate can be anything. However, the AES3 document [Ref 13] defines the standard audio sampling rates for AES3 audio as 32 KHz, 48 KHz, and 96 KHz. The audio sampling rate used on audio CDs is 44.1 KHz, so this sampling rate is also often carried on AES3 interfaces as are some multiples of 44.1 KHz, such as 88.2 KHz and 176.4 KHz. Also, some high-end applications are starting to use 192 KHz as the sampling rate. Table 16-1 shows the AES3 bit rates corresponding to various common audio sampling frequencies. The symbol rate of the AES3 signal is actually twice as fast as the bit rate because each bit is encoded as two states.

The AES3 specification also defines a single-channel double-sampling frequency mode. In this mode, the two subframes in a frame carry consecutive samples for one channel instead of samples from two different channels. This allows the sampling rate to double while maintaining the same AES3 bit rates.

Table 16-1: AES3 Bit Rates

Audio Sampling Rate (KHz)	AES3 Bit Rate ⁽¹⁾ (Mb/s)
32	2.048
44.1	2.8224
48	3.072
96	6.144
192	12.288
96 ⁽²⁾	3.072

Notes:

1. The AES3 serial symbol rate is 2X the bit rate.
2. This entry shows the use of the single-channel double-sampling frequency mode for 96 KHz.

SMPTE 272M: Embedded Digital Audio for SD-SDI

Digital video streams often carry non-video ancillary data embedded in the horizontal and vertical blanking intervals. One of the most common types of ancillary data is digital audio. Multiple channels of digital audio can be embedded in the horizontal blanking intervals of digital video signals carried by SD-SDI.

The SMPTE 291M specification describes the generic format of ancillary data packets for SD-SDI. SMPTE 272M specifies how AES3 digital audio is mapped to these ancillary data packets and embedded into SD-SDI video streams.

The SMPTE 291M standard allows 16 audio channels per video stream. These channels are divided into four audio groups, numbered 1 through 4. Audio channels 1 through 4 are assigned to audio group 1, channels 5 through 8 to audio group 2, channels 9 through 12 to audio group 3, and channels 13 through 16 to audio group 4. The four channels in each audio group are also grouped into two channel pairs. Each channel pair consists of two audio channels, usually a stereo pair derived from the same audio source. An AES3 signal carries two audio channels, and these two channels are typically mapped into one channel pair of an audio group when embedded in a video stream.

In the video broadcast industry, the standard audio sample rate is 48 KHz. SMPTE 272M also supports embedded audio sample rates of 44.1 KHz and 32 KHz.

SMPTE 272M supports both 20-bit and 24-bit audio samples. Some video equipment might support only 20-bit audio. SMPTE 272M embedded audio packets are designed so that 24-bit embedded audio is usable by equipment supporting only 20-bit audio.

SMPTE 272M also permits embedded audio that is not compliant with the AES3 specification, called non-pulse-coded modulation (non-PCM) data. SMPTE 337M describes how non-PCM data is mapped into the same data structures as defined by AES3, allowing it to be embedded in SD-SDI video streams. Non-PCM data is usually compressed multi-channel (surround sound) audio.

SD Embedded Audio Packets

Three types of ancillary data packets are used for SMPTE 272M embedded audio: audio data packets, extended data packets, and audio control packets. For each embedded audio packet type, four different DID values are defined, with a unique DID assigned to each audio group. Thus, the DID word identifies both the packet type and the audio group of the packet. Table 16-2 shows the DID values assigned to the various embedded audio ancillary data packet types.

Table 16-2: DID Values for SD Embedded Audio Packets

Group	Audio Data Packets	Extended Data Packets	Audio Control Packets
Group 1	0x2FF	0x1FE	0x1EF
Group 2	0x1FD	0x2FC	0x2EE
Group 3	0x1FB	0x2FA	0x2ED
Group 4	0x2F9	0x1F8	0x1EC

SD Audio Data Packets

An audio data packet contains audio samples for a single audio group. Audio samples from different audio groups cannot be mixed together in the same audio data packet.

Figure 16-2 shows the format of the audio data packet. The audio data packet is variable length. It can contain as many audio samples as fit, given the length limit of ancillary data packets (255 data words) and the space available in the horizontal ancillary data space (HANC). Other ancillary data packets, including audio data packets for other audio groups, might also be present in the same HANC, further limiting the size of an audio data packet. However, audio data packets are usually much smaller than the size of the HANC. With 48 KHz audio, an audio data packet commonly has three or four audio samples for each active channel in the group.

Each audio sample occupies three consecutive words in the audio data packet. Samples for both channels of a channel pair must be present in the packet, even if only one channel of the pair is active. The 2004 version of the SMPTE 272M standard recommends that both channel pairs of a group always be included in the audio data packet, but this is not strictly required. It is legal for an audio data packet to contain samples for only one channel pair.

The SMPTE 272M standard specifies that the two audio samples from the two channels of a channel pair must always be paired together and appear consecutively in the audio data packet. The sample from the lower numbered channel of the channel pair must be first, followed immediately by a sample from the higher numbered channel. For example,

channels 1 and 2 are in channel pair 1 of audio group 1. In the audio data packet, a sample for channel 1 must be followed immediately by a sample for channel 2.

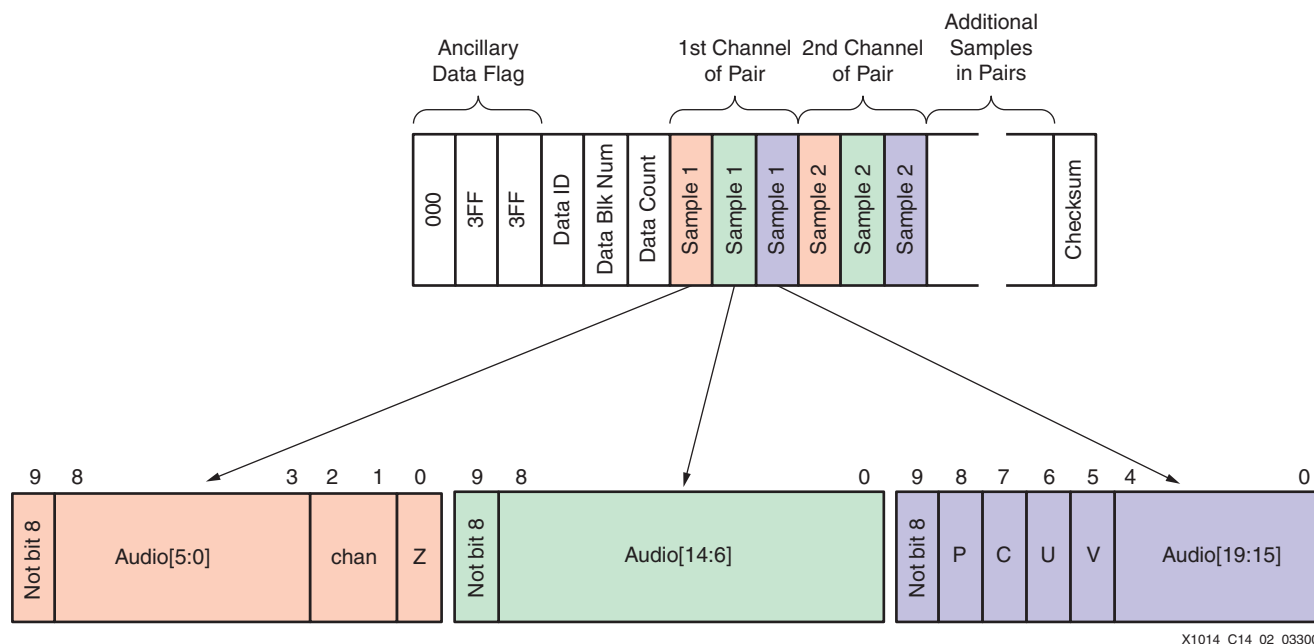


Figure 16-2: SD Audio Data Packet Format

The SMPTE 272M revision of the standard recommends that within an audio data packet, the channel pair containing the lower numbered channels precede the channel pair containing the higher numbered channels. For example, in an audio data packet for audio group 1, channels 1 and 2 should precede channels 3 and 4, but this is not a strict requirement.

The three words of an audio sample contain 20 bits of audio data plus additional information including a channel code, block sync (Z), V, U, and C bits, and a parity bit.

The 2-bit channel code indicates the audio sample's channel number within the audio group. For example, if the audio data packet is from audio group 1, channel number values of 00, 01, 10, and 10 indicate audio channels 1, 2, 3, and 4, respectively. If the audio data packet is from audio group 2, channel numbers 00, 01, 10, and 10 indicate audio channels 5, 6, 7, and 8.

The C, U, and V bits in the audio sample are identical to the AES3 C, U, and V bits. The V bit indicates whether the channel is valid (0 = valid, 1 = invalid). The channel status and user data are each organized into 192-bit data structures sent one bit per audio sample. To correctly interpret the C and U data, the receiver must know where the 192-bit block begins and ends. The Z bit is only High in the first audio sample of each 192-bit block, indicating that the sample contains the LSB of the 192-bit C and U data. Typically, the Z bits of two audio samples in a channel pair are the same because AES3 requires this of the two channels carried on one AES3 interface. However, SMPTE 272M does allow the Z bits for the two channels of a channel pair to be set independently, meaning that the 192-sample blocks in the two channels in a channel pair are not necessarily aligned in the two channels of the pair.

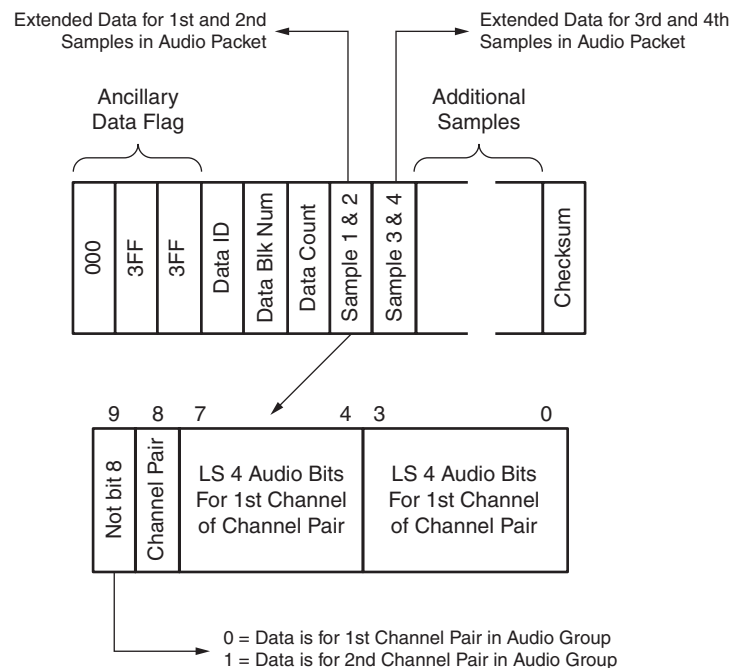
SD Extended Data Packets

Audio data packets only provide 20 bits of audio data per sample. For 24-bit audio, the additional four bits of each audio sample (called the auxiliary or extended data) are embedded in a separate ancillary data packet type called the extended data packet. There are different extended data packet DID values for each audio group, so an extended data packet only carries the extended data for one audio group.

The four bits of extended data are the least significant four bits of the 24-bit audio sample. The most significant 20 bits are in the audio data packet. If the receiving equipment doesn't support 24-bit audio, it simply uses the 20 bits in the audio data packets and ignores the extended data packets, throwing away the least significant four bits of each audio sample.

When extended data packets are present, they are always paired with their associated audio data packet. An extended data packet immediately follows its associated audio data packet. If an extended data packet does not immediately follow an audio data packet, 20-bit audio is assumed.

In an extended data packet, each data word contains the extended data for both audio samples of a channel pair (Figure 16-3). The first data word of an extended data packet contains the extended data for the first two audio samples of the preceding audio data packet. The second data word of the extended data packet carries the extended data for the next two audio samples of the preceding audio data packet, and so on. A bit in each extended data packet data word indicates whether the data word is for the first channel pair of the audio group or the second. Because the order of the data words in the extended data packet exactly matches the order of the audio samples in the audio data packet, this bit primarily serves to check whether or not the ordering of extended data is correct.



X1014_C14_03_033009

Figure 16-3: SD Extended Data Packet Format

SD Audio Control Packets

Audio control packets carry additional information about the audio stream. Audio control packets are sent once per video field, always in the second HANC after the synchronous switching interval. Audio data packets and extended data packets are typically not sent in the HANC of this line.

As with the other embedded audio packet types, SMPTE 272M reserves four different DID values for audio control packets, one for each of the four audio groups. Thus, an audio control packet is specific to an audio group. If all four audio groups are present in the video stream, four audio control packets, one for each audio group, are present in the HANC of the specified line.

Audio control packets are optional when the embedded audio conforms to the SMPTE 272M default 48 KHz synchronous audio. Otherwise, audio control packets are required.

The audio control packet is fixed in length. All data words in the packet have a defined meaning (Figure 16-4). Four types of information are contained in the audio control packet: audio frame number, audio sample rate, active channel indication, and audio delay.

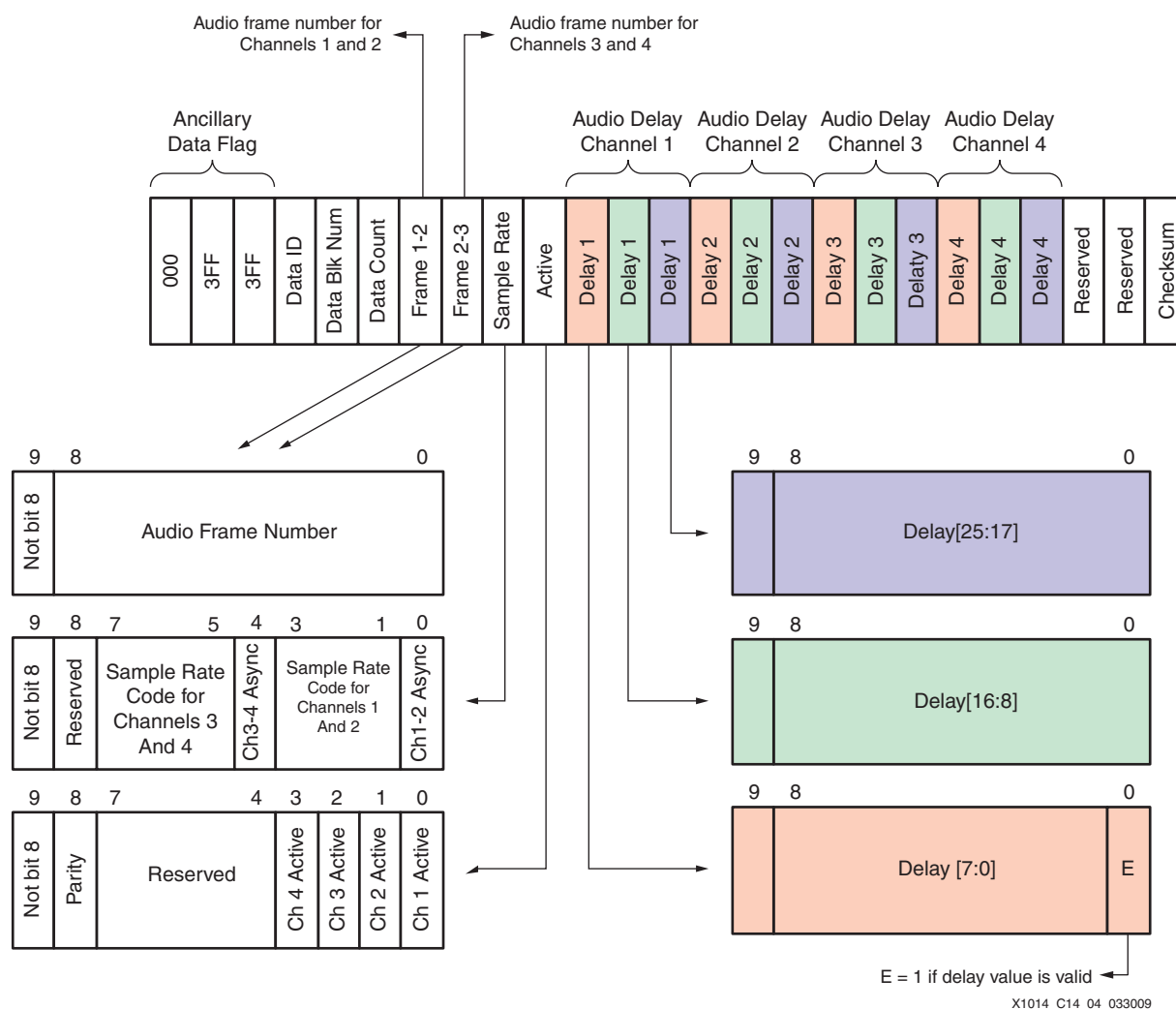


Figure 16-4: SD Audio Control Packet Format

Audio Frame Number

The first word in the audio control packet payload contains the audio frame number for the first channel pair of the audio group. The second word contains the audio frame number for the second channel pair of the audio group. The audio frame numbers provide information about how many audio samples are present in the video frame. For example, with NTSC video and 48 KHz audio, 8,008 audio samples must be sent every five video frames. Because 8,008 is not evenly divisible by 5, some video frames must carry more audio samples than other frames. SMPTE 272M strictly defines how many audio samples are to be carried in each video frame in the sequence of five frames. Frames 1, 3, and 5 of the five frame sequence must have exactly 1,602 audio samples. Frames 2 and 4 must have 1,601 audio samples. The audio frame number indicates the position of the current frame in the five-frame sequence. A frame number of 0 indicates that the audio frame number is not used.

For different video frame rates and audio sample rates, different frame sequences are required. For example, with 44.1 KHz audio and NTSC video, 147,147 audio samples must be transmitted in 100 video frames. The audio frame sequence count, in this case, would range from 1 to 100.

Audio Sample Rate

The third data word of an audio control packet indicates the audio sample rates of both channel pairs. Three bits are allocated in this word for each channel pair indicating the audio sample rate. The encoding of these bits is shown in Table 16-3. An additional bit is provided for each channel pair. If this bit is 1, the associated channel pair is operating asynchronously.

Table 16-3: Audio Sample Rate Codes

Code	Sample Rate (KHz)
000	48
001	44.1
010	32
011 to 110	Reserved
111	Undefined

Active Channels

The fourth data word of an audio control packet indicates which audio channels of the audio group are active. A bit is allocated to each of the four possible channels in the audio group to indicate whether the channel is active. The active bit for a channel is set to 1 if the channel is active.

Audio Delay

Following the active channel data word in the audio control packet are four audio delay values. Each audio delay value occupies three data words in the packet. The first three audio delay words indicate the audio delay for the first channel of the audio group. The next three words indicate the delay for the second channel of the audio group, and so on. Audio delay values are 26-bit two's-complement values indicating the accumulated audio processing delay relative to the video, expressed in audio sample intervals. An additional bit in each audio delay value indicates whether the audio delay is valid. If an audio delay value is valid, the LSB of the first word of the delay value is set to 1.

SD Audio Sample Distribution

The audio samples should be evenly distributed throughout the video frame, otherwise there is a risk that the audio input buffer in the receiver could overflow. Some lines in the vertical blanking interval are not used to carry audio samples due to possible corruption of data on these lines during video switching events.

In addition, as described in [Audio Frame Number](#), specific requirements govern how many audio samples must be included in each video frame of an audio frame sequence. The length of the frame sequence and the audio sample count in each frame of the sequence are dependent upon the video frame rate and the audio sample rate.

SMPTE 299M Specification

Embedded Digital Audio for HD-SDI, 3G-SDI, and Dual Link HD-SDI

The SMPTE 299M specification specifies how to embed digital audio in HD-SDI video streams. It is similar to SMPTE 272M, but has some significant differences. 3G-SDI standards, both level A and level B, and dual link HD-SDI also use SMPTE 299M for embedded audio.

All of these video formats have separate HANC data spaces for the Y and C channels. SMPTE 299M audio data packets are only located in the HANC space of the C channel. Audio control packets are only located in the HANC space of the Y channel. There are no extended data packets for embedded audio in HD-SDI because the audio data packets support 24 bits of audio per sample. In SMPTE 299M, the audio data packets are fixed length and carry exactly one sample for each of the four channels in the audio group. The audio data packets also carry additional data and error correction that are not present in the SD audio data packets.

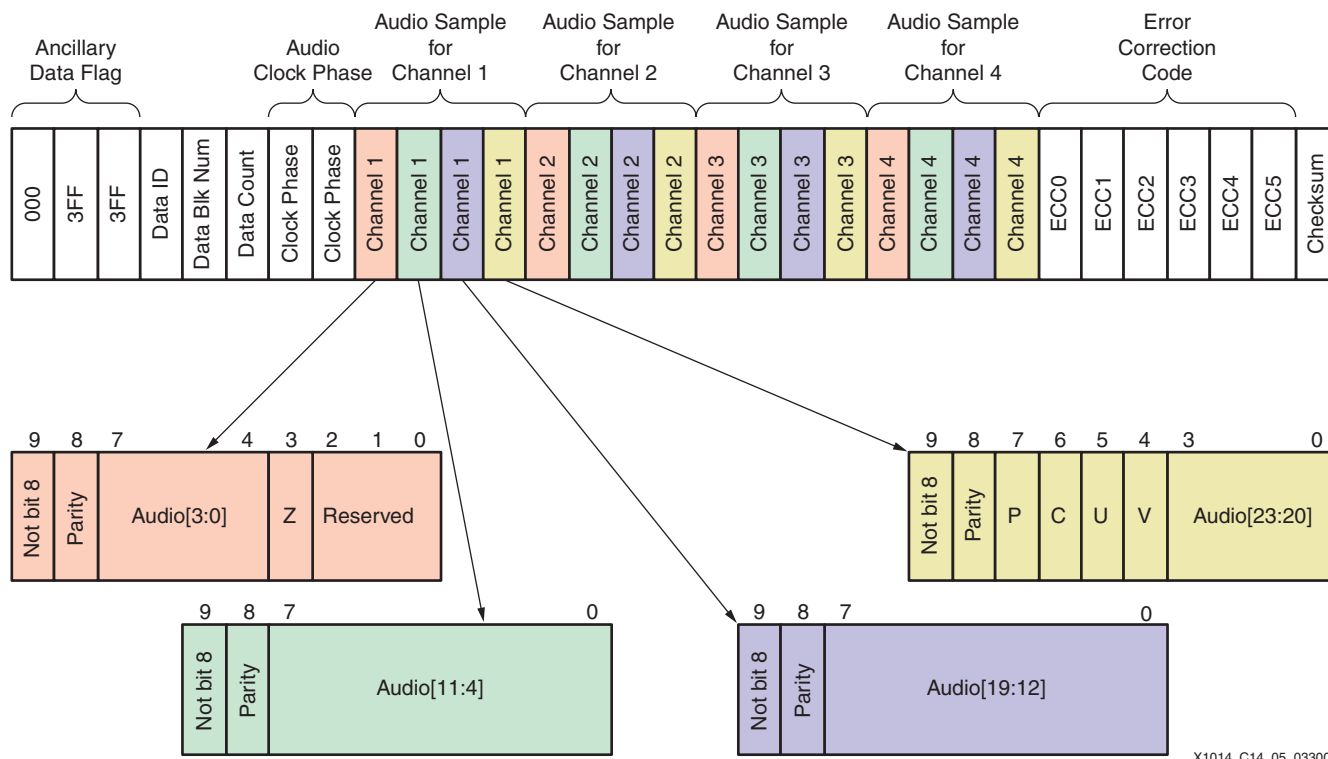
Table 16-4: DID Values for HD Embedded Audio Packets

Group	Audio Data Packets	Audio Control Packets
Group 1	0x2E7	0x1E3
Group 2	0x1E6	0x2E2
Group 3	0x1E5	0x2E1
Group 4	0x2E4	0x1E0

HD Audio Data Packets

The format of an HD audio data packet is shown in [Figure 16-5](#). The first two words of the audio data packet payload carry audio clock phase information that applies to all four audio samples in the packet. This information includes a 12-bit value indicating the number of video clocks that elapsed between the EAV and the occurrence of the audio sample. Normally, the audio data packet for a particular audio sample is located in the HANC space of the very next video line. However, due to restrictions on inserting audio data packets into the HANC space that follows the synchronous switching interval, audio packets that would have been placed in that HANC space are delayed by one line. To correctly interpret the audio clock phase information for these delayed audio packets, the multiplex position flag is set to 1 when audio packets are delayed by one video line.

Following the two audio clock phase data words, the payload has four UDWs for each of the four audio channels in the audio group. The first group of four words contains the audio sample data for the first channel of the audio group, the second group of four words contains the sample data for the second channel of the group, and so on.



X1014_C14_05_033009

Figure 16-5: HD Audio Data Packet Format

The audio sample data includes 24 bits of the actual audio information and the P, C, U, and V bits that are identical to these same bits in the AES3 audio stream. The Z bit in the first channel sample data is the Z frame indicator for both channels 1 and 2. The Z bit in the third channel sample data is the Z frame indicator for channels 3 and 4. Unlike SD-embedded audio samples, the Z frame of the two audio channels in a channel pair must always coincide. Following the user data words for the fourth audio channel, the payload contains six words of error correction code for the audio data packet.

HD Audio Control Packets

HD audio control packets are identical in format to SD audio control packets but have different DID values. As with SD audio control packets, each HD audio control packet is specific to one audio group. The audio control packets are sent once per field in the HANC space of the second line after the synchronous switching interval. Audio control packets are sent in the Y component HANC space.

HD Audio Sample Distribution

HD audio sample distribution is more restrictive than for SD. Because of the inclusion of the clock phase data, audio samples must always be inserted in the HANC interface of the following video line. This is applicable for all audio samples, except those that occur

during the video line containing the synchronous switching interval that are delayed by one line.

The maximum number of audio data packets for one audio group that can be located in one HANC space is two. When two audio data packets from the same audio group are present in the same HANC space, they must be ordered so that the audio data packet containing the earliest audio sample information occurs first.

Audio Sample Rate Conversion

Introduction

Sample rate conversion is required when audio applications need a different sample frequency from that of the audio source. Common sample rates include 32 KHz, 44.1 KHz, 48 KHz, 96 KHz. Often, source material recorded with one sample rate must be converted to another sample rate for processing. This is required in such systems as studio digital mixers, audio effects processors, digital audio broadcast equipment, and video systems with embedded audio. In some instances, the inputs and outputs can be synchronized. In others, synchronization is not possible or desirable, or the sample frequency is time-varying. Three basic categories for the state of synchronization are:

- **Synchronous:** The input sample timing and output sample timing are based on a common timing source, i.e., the output sample rate is a fixed, rational multiple of the input sample rate, and there is no drift between them.
- **Plesiochronous:** The input sample rates are nominally the same but are based on two different oscillators. Due to tolerance variations and temperature or voltage drift in the two independent oscillators, the two frequencies are not locked but differ by a small, varying amount.
- **Asynchronous:** The input rate and output rate are sourced from different oscillators and also differ in nominal frequency. The sampling rate of the input or output can be variable.

The synchronous case can be handled by a synchronous or fixed-ratio sample rate converter. The plesiochronous and asynchronous cases require an asynchronous sample rate converter.

In either case, the function of the sample rate converter is to create a stream of samples timed to match the output timing source and to accurately represent the continuous-time signal embodied by the input samples. This is illustrated in [Figure 16-6](#).

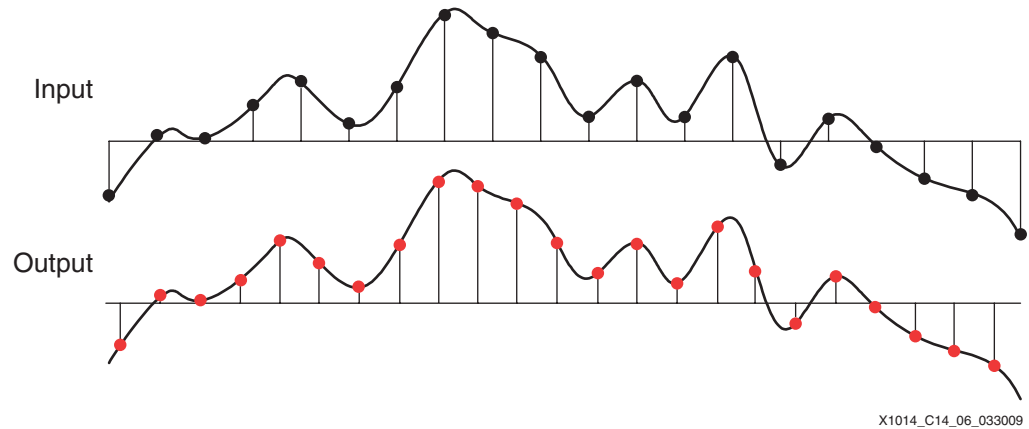


Figure 16-6: Sample Rate Conversion

Clarification of Terms

In multi-rate digital signal processing, the term “interpolate” means to increase the sample rate of a signal. In general mathematics, interpolate has a similar but more general meaning: to construct new data points from a discrete set of known data points. For clarity, this application note uses the term “up-convert” to mean increasing the sample rate of an audio signal, and “interpolate” to mean constructing intermediate data points from a set of known data points.

For symmetry, “down-convert” is used to denote decreasing the sample rate of a signal, and implies a filtering process. “Down-sample” and “decimate” mean simply selecting every n th sample and discarding the rest.

Methods of Sample Rate Conversion

Synchronous

There are a number of approaches to sample rate conversion. In the classic configuration, up-conversion (Figure 16-7) has an up-sampler followed by a low-pass filter to eliminate spectral duplication or images.

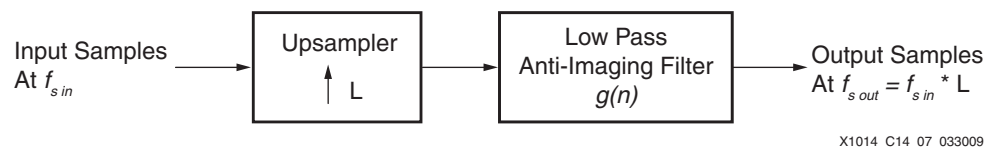


Figure 16-7: Classic Up-Conversion

Down-conversion (Figure 16-8) uses a low-pass filter on the input samples to keep high-frequency components from aliasing into the base band of the output. This is followed by down-sampling by a factor, M .

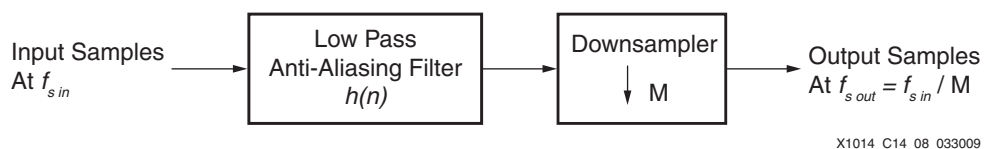


Figure 16-8: **Classic Down-Conversion**

The more general case of conversion by a rational number consists of up-conversion followed by down-conversion, as shown in Figure 16-9. The anti-aliasing filter and anti-imaging filters are combined into a single low-pass filter.

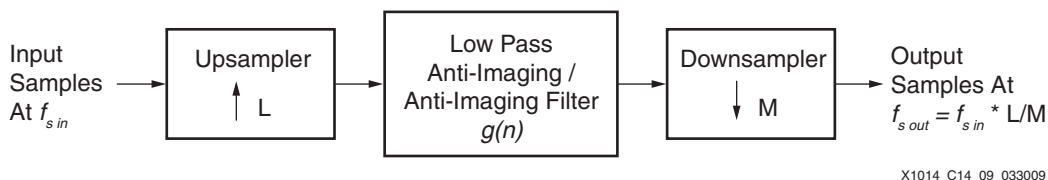


Figure 16-9: **Classic Sample Rate Conversion By a Rational Number**

In practice, there are many variations of implementation, most notably performing the up-sampling and down-sampling in multiple stages and moving the location of the low-pass filter, for efficiency, either to the first step or the last step (depending on which has the lower sampling rate). For example, up-conversion from 44.1 KHz to 48 KHz ($L/M = 160/147$) could be done in stages of 2:1, 2:1, 5:1, and 8:147, with the low-pass filter performed in the last stage. Similarly, down-conversion of 48 KHz to 44.1 KHz ($L/M = 147/160$) could be done in stages of 147:8, 1:5, 1:2, and 1:2, with the low-pass filter combined with the first stage. In both these cases, the output sample rate is a function of the input rate multiplied by a fixed rational number. In other words, they are synchronous.

Asynchronous

In many practical cases, the input-to-output ratio is not fixed and tends to wander slightly over time because the input is based on a different timing source than the output. In other words, the input and output are asynchronous. A sample rate converter with a fixed conversion ratio cannot handle such situations without one of these two scenarios occurring:

- The input-to-output latency changes due to accumulating delay.
- Artifacts are produced in the audio, such as skipping samples or repeating samples.

Both of these cases represent undesirable distortions. To handle such cases without introducing artifacts, an asynchronous sample rate converter is required. Asynchronous sample rate converters operate with a variable conversion rate. Conceptually, they up-convert the input signal into a highly over-sampled signal, then down-convert by choosing the sample at the time nearest each output sample, as shown in Figure 16-10.

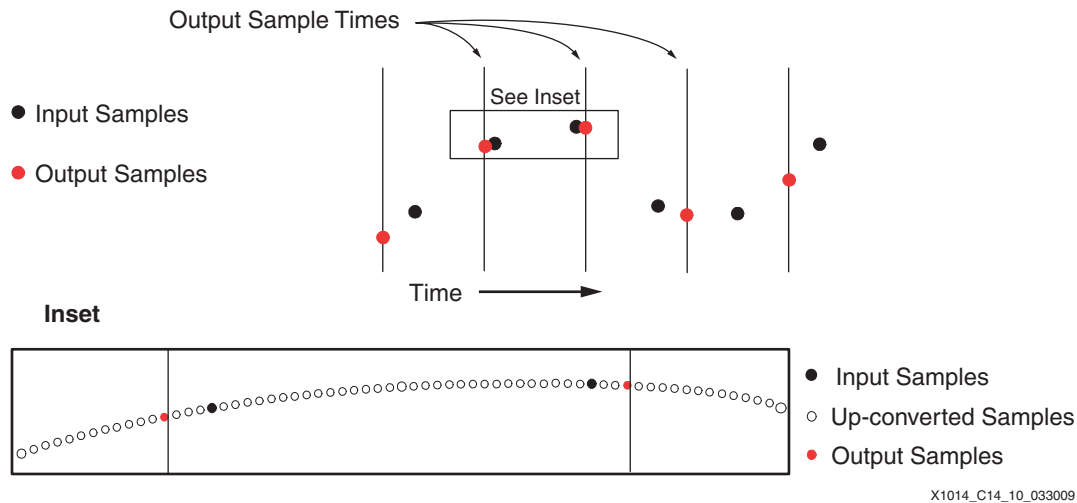


Figure 16-10: Classic Method of Asynchronous Sample Rate Conversion

Thus, the phase of the generated output samples varies smoothly and virtually continuously with respect to the input samples.

The classic method for the up-conversion step is to up-sample by adding zero-valued samples at close intervals between the input samples, then convolving the resulting signal with a low-pass filter, as shown in Figure 16-8. There are, however, other methods that are more computationally efficient.

Interpolated-Coefficient Finite Impulse Response Filter

The method described in this application note uses a polyphase filter approach in which several million possible phases of the prototype filter are interpolated at run time from a relatively small number of phases stored in RAM. These intermediate phases are produced by polynomial interpolation. As a result, the resample and filter operations are performed concurrently by a relatively small convolution operation of input samples with the subfilter for each output sample. This process, shown in Figure 16-11, is referred to in this application note as interpolated coefficient finite impulse response (ICFIR) filtering. The conversion factor L is referred to as the ratio. In general, this value can be greater than, less than, or equal to 1, and can vary over time.

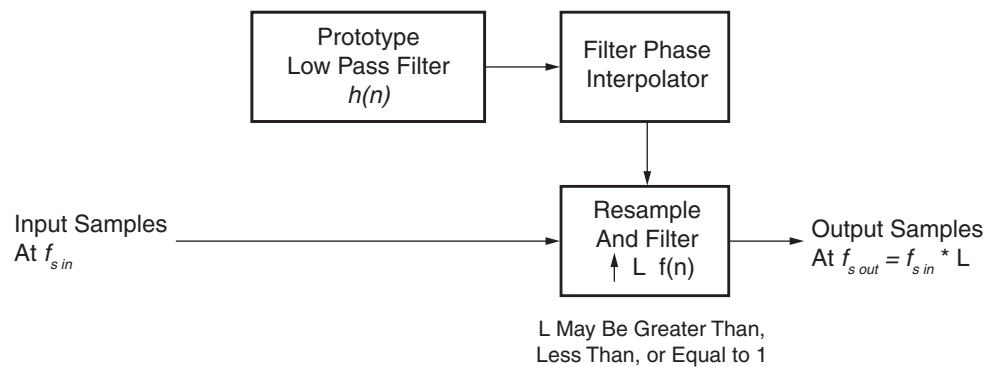


Figure 16-11: Interpolated Coefficient FIR Filtering

A unique set of coefficients for the subfilter is required for every phase of the output sample with respect to the input samples. The required subfilter is identified by centering the prototype filter at the output sample that is to be calculated. This is illustrated in Figure 16-12.

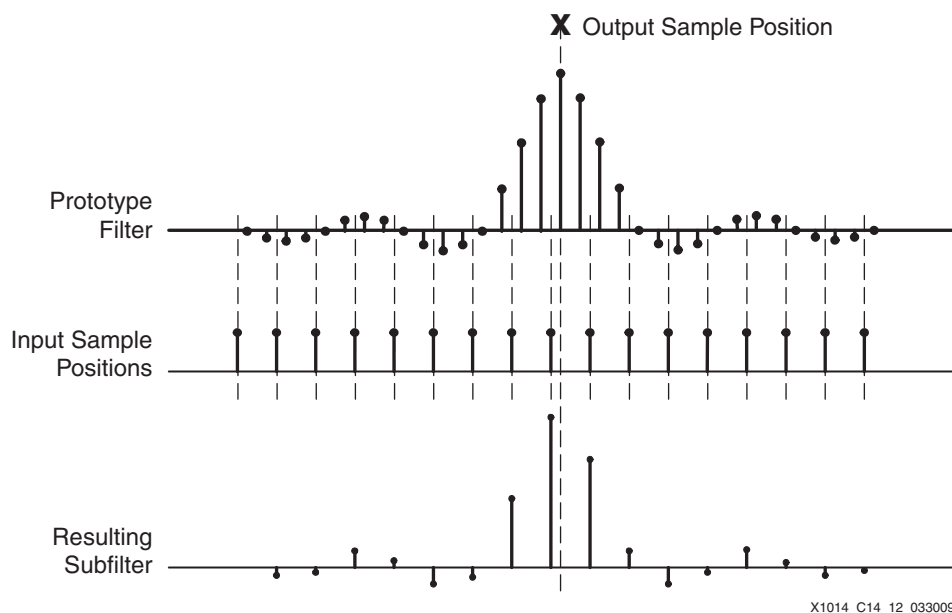


Figure 16-12: Prototype Filter Centered at Output Sample Position

The prototype filter shown is shifted so that the center of the filter lines up with the output sample that is to be calculated. The input sample positions do not necessarily align with the coefficients of the prototype filter. A set of coefficients that *do* align with the input sample positions are interpolated from the stored coefficients, resulting in the subfilter shown at the bottom of Figure 16-12. When this subfilter is convolved with the corresponding input samples, the output sample of interest is produced. This process is repeated, with new subfilter coefficients being interpolated for each output sample.

There are several advantageous characteristics of the ICFIR method:

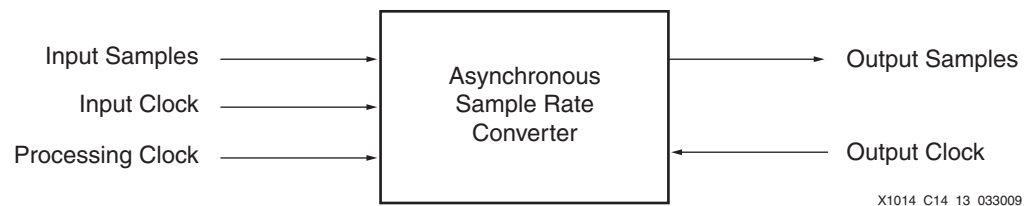
- **Manageable coefficient storage:** For asynchronous conversion, the number of possible conversion ratios and phase relationships are infinite. Therefore, storing and using fixed coefficient sets is not practical if reasonable distortion performance is to be attained. The ICFIR can handle millions of unique phases with only a few dozen stored phases.
- **Deterministic latency:** The number of taps used in the FIR convolution is fixed and relatively small. This results in low deterministic latency relative to the performance achieved. Frequency drift in the input and output is tracked out by minute adjustments to the phase of the interpolated filter.
- **Simple interpolation of coefficients:** Because the prototype filter is smooth and highly sampled, interpolation to a high degree of precision can be accomplished with polynomial interpolation at a much lower computational complexity than an equivalent FIR interpolation.
- **Common subfilter usage:** The computational cost of interpolating the filter coefficients can be amortized over multiple data streams. For an AES3 channel pair,

the same interpolated subfilter is used for both channels. Additional input channels in the same time domain can also use this same subfilter.

The IC FIR method also works for synchronous sample rate conversion. In such instances, a major simplification can be made by using a fixed ratio rather than a variable one.

ASRC Operation

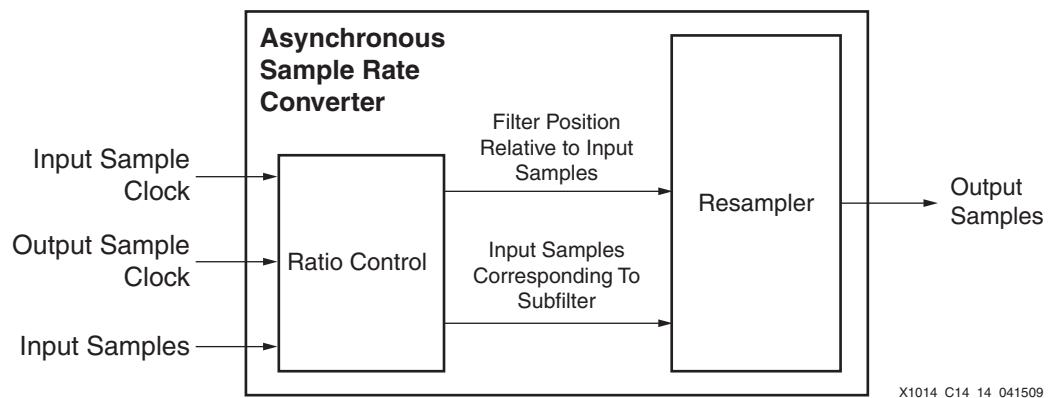
The asynchronous nature of the ASRC means that the input sample rate and output sample rate have no prescribed relationship to one another. Indeed, they are generally controlled from independent timing sources and are both inputs to the ASRC. The ASRC does not have control over either rate, only the ability to monitor the ratio between the two and resample the input data at the output rate. This is illustrated in Figure 16-13.



X1014_C14_13_033009

Figure 16-13: ASRC Inputs and Outputs

As shown in Figure 16-14, there are two main components of the sample rate converter: ratio control and resampling. The ratio control section measures the ratio of output samples to input samples and controls how fast the filter moves through the input samples to keep the input sample buffer filled to a certain level. The resampler performs the interpolation of filter coefficients from a prototype low-pass filter and applies them to the input samples in an FIR convolution.



X1014_C14_14_041509

Figure 16-14: ASRC Major Functions

Ratio Control

The ratio control function measures the period of input and output clocks to obtain a ratio value. This value is used to determine where each output sample is located, in time, relative to the input samples. There must also be a buffer for storing input samples that are used in the FIR convolution of the resampler. The calculated ratio is adjusted in minute increments to keep the input-to-output latency constant. The ratio control function outputs a precise filter position (which in turn reflects the output sample position) relative to the

input samples. This information is required to interpolate the appropriate subfilter. The set of input samples to be convolved with the subfilter coefficients is sent from the ratio control section.

Resampler

The resampler function interpolates the subfilter based on the filter position received from the ratio control function. It convolves the interpolated coefficients with the corresponding set of input samples to create the particular output sample of interest.

A set of coefficients representing the prototype low-pass filter is stored in memory. To form each output sample, the resampler receives the filter position relative to input samples. From this, a set of coefficients for the precise phase of output samples are calculated by interpolation of the stored coefficients. This is equivalent to selecting one of many subfilters.

Figure 16-15 shows a simplified graphic example of this process for up-sampling by a ratio of 2.4. The input samples times are denoted by green dots at the bottom of the figure. The output sample times are marked with x's. The stored coefficients of the prototype filter are shown with the filter centered on the output sample of interest. In this example, the stored prototype filter consists of four phases with four taps in each phase. The particular set of coefficients required for the convolution represents a filter phase whose coefficients lie between the stored coefficients. The position of the coefficients is defined by delta, a fractional value that indicates the relative position, in time, of input samples to output samples. Based on this delta, the four interpolated coefficients are calculated at the locations shown. The convolution of these coefficients with the corresponding input samples produces the output sample of interest.

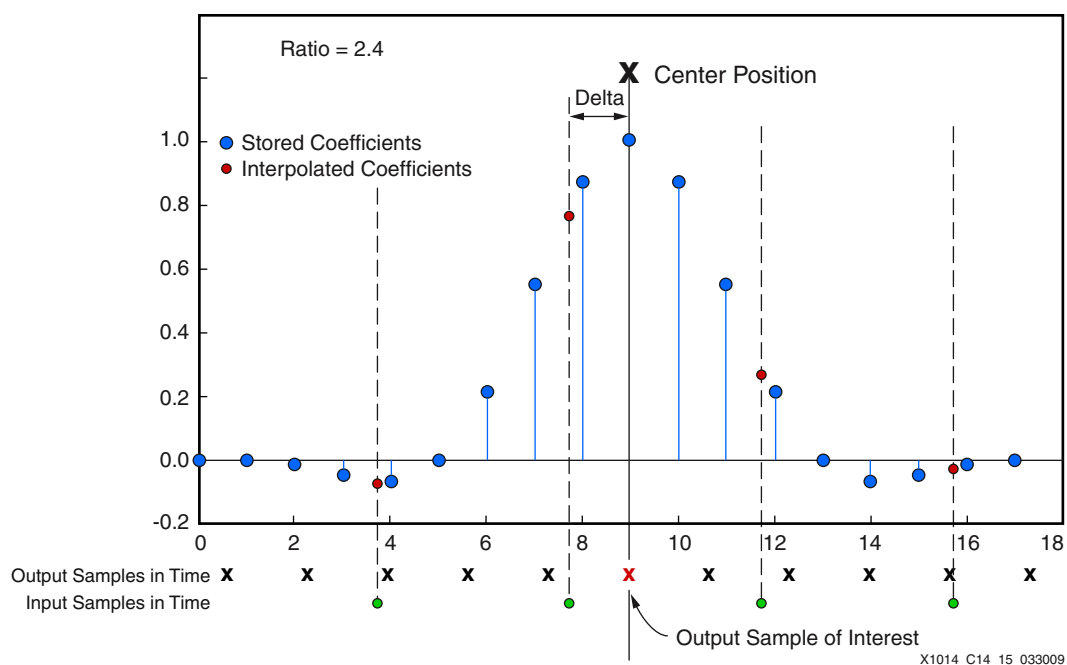


Figure 16-15: Up-Conversion Example

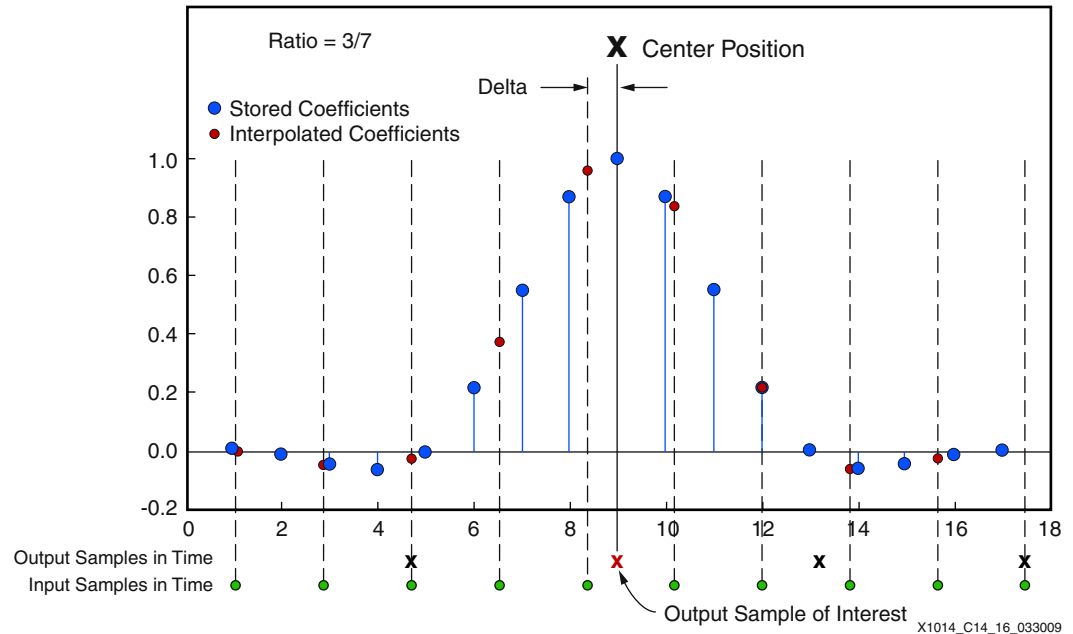
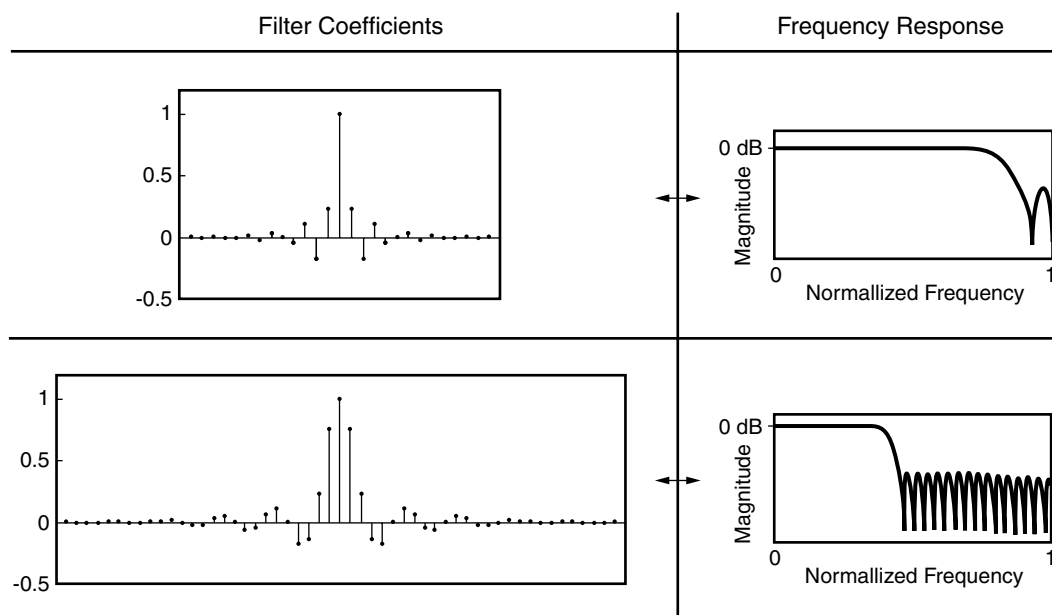


Figure 16-16: Down-Conversion Example

Filter Expansion for Down-Conversion

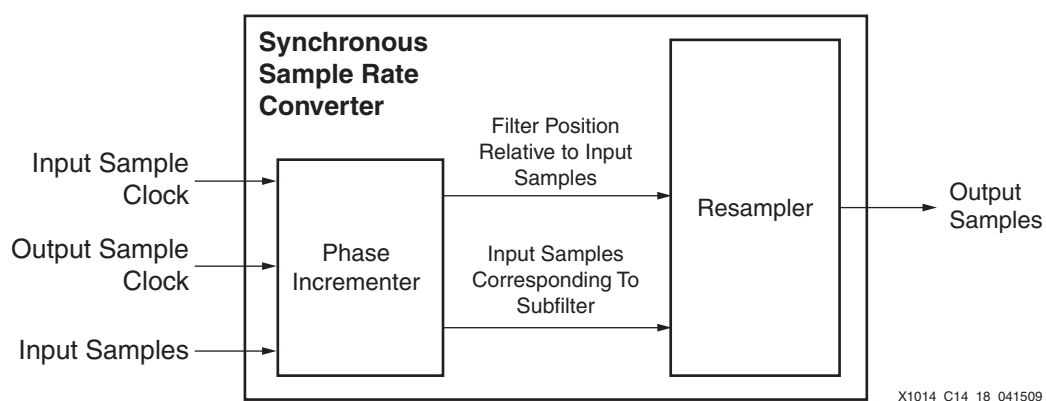
For the case of down-converting from a higher sample rate to a lower one, the cutoff frequency of the prototype filter must be lowered to avoid artifacts on the lower-bandwidth output. This is done by expanding the filter to cover more input samples, as shown in Figure 16-16. In this example, the down-conversion ratio is 3/7. The subfilter has more than the four nominal coefficients. In fact, the prototype filter is expanded to cover $1/\text{ratio}$, or $7/3$, times as many input samples as in the up-conversion case. The result in the frequency domain, by the scaling property of Fourier transforms, is compression of the spectrum proportional to the expansion in time, thus reducing the cutoff frequency. This is illustrated in Figure 16-17. This effect is equivalent to using a low-pass anti-aliasing filter on the input samples, as shown in Figure 16-8. A side effect of the spreading of the filter is that the additional terms in the convolution result in amplification of the result. Hence, the result of the convolution must be attenuated in amplitude to compensate for the increased length of the convolution.



X1014_C14_17_033009

Figure 16-17: Time Domain Spreading Lowers Cutoff Frequency

Synchronous SRC Operation



X1014_C14_18_041509

Figure 16-18: Synchronous Sample Rate Converter

For a synchronous SRC (Figure 16-18), the resampler function remains basically the same. However, the ratio detector is replaced by a phase incrementer. Rather than actively measuring the input and output period, this function increments the filter position and pointers to the input samples based on a pre-determined ratio. The fixed ratio is arbitrary, because the prototype filter of the resampler can be scaled to handle any ratio. The resampler function interpolates the required filter coefficients and applies them to the input sample set to produce the required output samples.

Lagrange Interpolation of Filter Coefficients

Lagrange polynomial interpolation (Figure 16-19) can be used efficiently to interpolate filter coefficients from the stored prototype filter. For degree n , the Lagrange interpolation

calculates a value at a given point based on a function of degree n that passes through the neighboring $n+1$ points. Zero order equates to a hold interpolator, first order equates to linear interpolation of 2 points, second order equates to interpolating quadratic equation over 3 points, and so forth. Accuracy increases with increasing order for smooth sets of points.

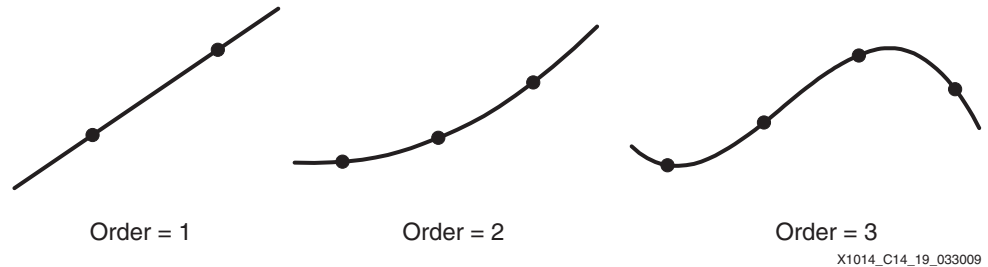


Figure 16-19: Lagrange Interpolation

This method of interpolation is well suited to interpolating intermediate values from the relatively smooth, over-sampled shape of filter coefficients. Lagrange interpolation can be used successfully to generate intermediate coefficients, as shown in Figure 16-20. The accuracy of the results is determined by the order of the interpolation.

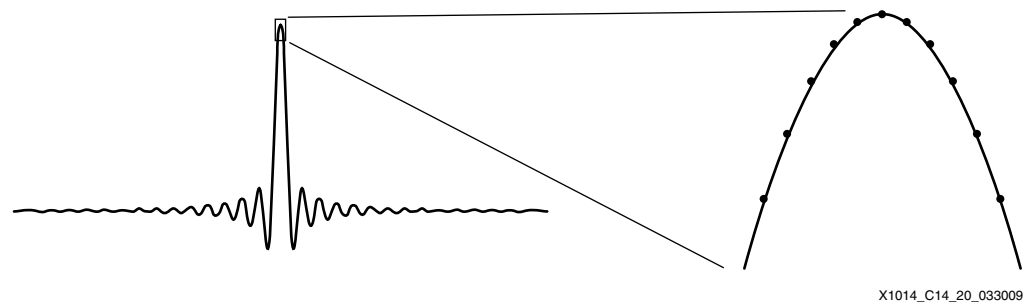


Figure 16-20: Lagrange Interpolation on a Prototype Filter

Lagrange interpolation does not, however, yield high-quality results when applied directly to the generally uneven audio samples. For example, in the case of a high-frequency tone as shown in the dashed line of Figure 16-21, the solid line shows the third order Lagrange interpolation of four sample points. The error from the theoretical signal is significant and readily apparent.

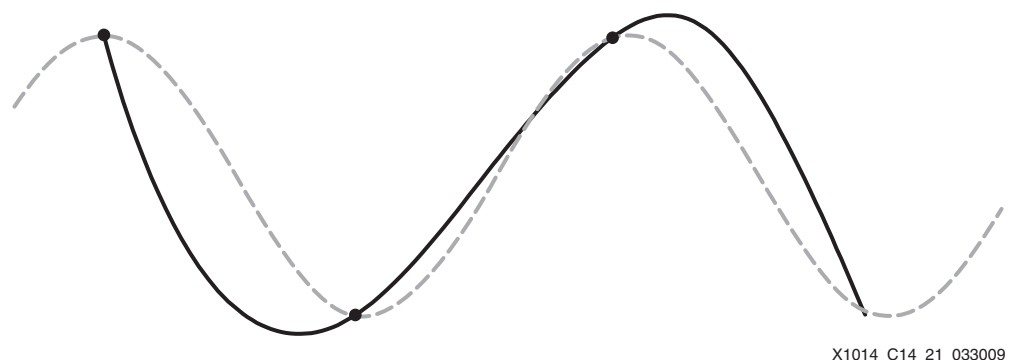


Figure 16-21: Lagrange Interpolation Unsuitable to Direct Sample Application

Performance Factors

The principal figures of merit for sample rate converters are:

- Total harmonic distortion plus noise (THD+N)
- Maximum conversion ratio, for up-conversion and down-conversion
- Maximum sample rate

THD+N, expressed in dB, defines the audio quality of the SRC. It is measured by passing a sine wave at a certain frequency through the SRC and measuring the spectrum of the result. The tone shows up as a high-amplitude spike at the given frequency. The amplitudes at all other frequencies represent distortion and noise and should be minimal. Because the inputs and outputs are all digital, the only noise is quantization noise due to the finite length of the digital words. For 24-bit audio data, the noise floor is 1 LSB or $2^{-24} = -144$ dB.

Many factors contribute to the harmonic distortion. For the sample rate converters discussed in this application note, the major factors that affect THD+N are the number of stored coefficients in the prototype filter, the bit width of the coefficients, the accuracy and stability of input-to-output phase measurement, and the prototype filter function itself (e.g., width of passband, stopband attenuation).

The maximum conversion ratios and the maximum sample rate determine what rate combinations are possible. For each output sample, a complete coefficient set interpolation and FIR convolution must be performed. The time to perform these operations is a function of the number of taps in the FIR filter, and the amount of computational horsepower dedicated to the task (number of computational elements and the clock rate). The time to compute each output sample sets an upper limit on the output sample frequency.

For up-conversion, the input sampling frequency is, by definition, lower than the output frequency, so the input frequency has the same upper limit as the output frequency. For down-conversion, the input frequency is higher than the output frequency. However, the FIR convolution is extended to cover more input samples, therefore taking longer to perform. This time increase is a function of the down-conversion ratio and has the effect of limiting the output rate to $1/\text{ratio} \times \text{up-convert maximum}$. This is equivalent to limiting the input ratio to the output sample frequency maximum. Though there are several limiting mechanisms, the net result is that the maximum sample frequency is approximately the same for the input as for the output.

For up-conversion, the maximum ratio is determined by the width of datapaths that use the ratio. Down-conversion has the added limit of storage for the additional input samples required in the expanded FIR convolution.

Prototype Filter Design

The prototype filter can be designed using the Filter Design and Analysis tool in MATLAB® software. It is a low-pass equiripple filter. It consists of N phases of M taps each. Higher M and N result in better performance, at the cost of more memory for storage. M is the number of taps in the FIR filter applied for up-sampling. N represents the number of phases to be stored as the prototype filter. Additional phases are produced at run time by interpolating between the stored phases. The prototype filter can be realized by specifying a filter order of $M * N$ and frequency specs of $1 / (M * N)$ times the desired response of each phase. The resulting coefficients must be scaled by a factor of N to fully utilize the coefficient bit width and to maintain the signal amplitude. For sample rate converters, the transition band is usually symmetric about the Nyquist frequency.

Passband Ripple vs. Stopband Attenuation

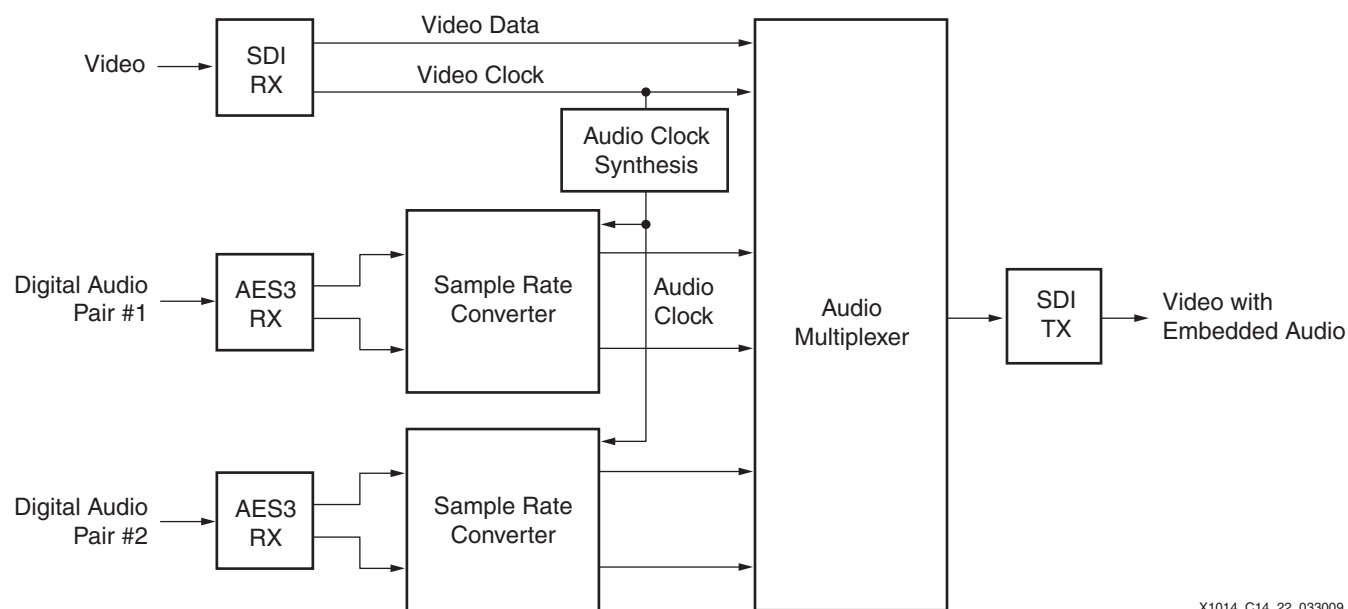
For a given filter order and transition band, increasing the stopband attenuation (how effectively high frequencies in the stop band are filtered out) results in increased passband ripple (undesirable variations in gain at different frequencies).

Stopband attenuation is important because it is a primary factor in THD+N performance. For up-conversion, up-sampling results in images of the fundamental spectrum at higher frequencies. For down-conversion, the input data stream can contain frequencies that are beyond the Nyquist rate of the output. In either case, the extraneous high-frequency energy is aliased back into the fundamental spectrum and manifested as harmonic distortion. Therefore, it must be filtered out. The stopband attenuation of the prototype filter directly affects the magnitude of this distortion.

Passband ripple is undesirable because it is a form of distortion; frequencies do not have the same gain. Passband ripple of a filter is normally specified as deviation above and below 0 dB. Digital audio, however, has fixed bit widths. Therefore, any calculations that would result in values above full scale must be limited or clamped to the full-scale value. This clamping of waves results in distortion artifacts that are unacceptable for sample rate conversion. Thus, either the samples or the filter coefficients must be attenuated by a small amount (equal to the positive amount of passband ripple) at some point so that clamping, and the resulting clipping of waves does not occur. This means that a passband ripple of $\pm R$ is manifest as 0 to $-2R$ in the implementation. Ideally, the stopband attenuation would be infinite, and the passband ripple would be 0. However, in finite length filters, these must be balanced.

Typical Applications

Figure 16-22 shows a typical audio/video multiplexer. Several channels of digital audio enter the multiplexer through AES3 audio receivers. The video signal comes from an SDI receiver and can be either HD or SD. The audio from the AES3 receivers is typically asynchronous to the video clock recovered by the SDI receiver so asynchronous audio sample rate converters are used to synchronize the audio to the video clock. If the audio sample rate is other than 48 KHz, the audio sample rate converters also usually convert the sample rate of the audio to 48 KHz. The audio multiplexer block then creates embedded audio packets for the audio data and inserts them into the horizontal blanking intervals of the video signal. The video signal, with the embedded digital audio, is then transmitted by an SDI transmitter.

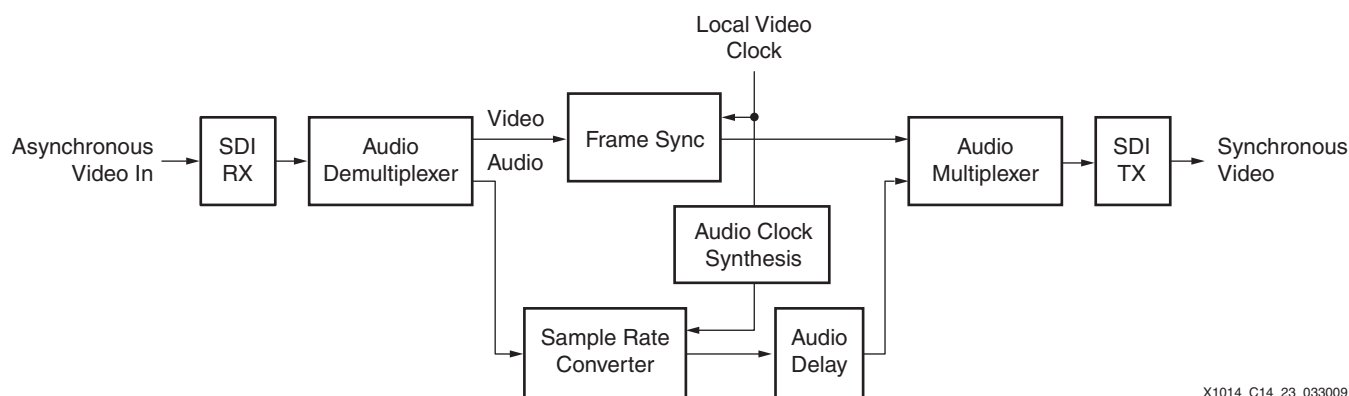


X1014_C14_22_033009

Figure 16-22: Example Audio Multiplexer Application

Figure 16-23 shows another typical application for audio multiplexing, demultiplexing, and sample rate conversion. This is a video frame synchronizer application. The video signal, with embedded audio, enters the video frame synchronizer through an SDI receiver. The video frame synchronizer stores frames of video in a buffer and then synchronizes the video to a local reference clock by occasionally dropping or repeating a frame of video.

The audio must be processed separately. Adding or dropping a frame of video containing embedded audio definitely causes an audible anomaly. An audio demultiplexer extracts the audio from the video signal before the video is written into the video buffer. The audio passes through a sample rate converter and possibly an audio delay buffer (to match the delay of the video signal). The audio is reinserted into the video by an audio multiplexer as the video is read from the frame buffer. The sample rate converter synchronizes the audio signal to the local video clock.



X1014_C14_23_033009

Figure 16-23: Example Frame Synchronizer with Audio Path

Digital Audio Reference Designs

The following chapters provide reference designs for digital audio:

- [Chapter 17, AES3 Serial Digital Audio Interfaces](#) provides a reference design for AES3 audio receivers and transmitters.
- [Chapter 18, Asynchronous Sample Rate Converter](#) provides a reference design for the Virtex®-5 FPGA asynchronous sample rate converter.
- [Chapter 19, Multi-Channel Asynchronous Sample Rate Converter](#) provides a reference design for the Virtex-5 FPGA multi-channel asynchronous sample rate converter.
- [Chapter 20, Audio Multiplexer for HD-SDI Video](#) provides a reference design for the HD-SDI audio demultiplexer.
- [Chapter 21, Audio Demultiplexer for HD-SDI Video](#) provides a reference design for the HD audio demultiplexer.
- [Chapter 22, Error Correction for HD-SDI Embedded Audio](#) provides a reference design for embedded audio error correction.
- [Chapter 23, Audio Demultiplexer for Standard-Definition Digital Video](#) provides a reference design for an audio demultiplexer for SD video.
- [Chapter 24, Audio Multiplexer for 3G-SDI, Dual-Link HD-SDI, and HD-SDI Video](#) provides a reference design for the audio demultiplexer.
- [Chapter 25, Audio Demultiplexer for 3G-SDI, Dual-Link HD-SDI, and HD-SDI Video](#) provides a reference design for the HD audio demultiplexer.

AES3 Serial Digital Audio Interfaces

Summary

AES3 is a professional standard for transporting digital audio serially over twisted pair or coaxial cable. Each AES3 audio link carries a stereo pair of digital audio channels and supports various audio sampling rates. AES3 is also called AES/EBU.

The consumer version of AES3 is called S/PDIF. S/PDIF is commonly used to move digital audio between pieces of consumer electronic equipment such as between a DVD player and a surround-sound receiver. The data format and data rates of S/PDIF are the same as AES3. S/PDIF has different electrical and physical (cable and connector) specifications from AES3.

This chapter describes how AES3 and S/PDIF receivers and transmitters can be implemented in Xilinx® FPGAs. The transmitter and receiver modules are very small and can easily be implemented in most current Virtex®-5 FPGAs. The AES audio receiver design presented in this chapter requires no external PLL components because it uses digital oversampling techniques to recover the data.

Reference Design

The reference designs for the AES3 receiver and transmitter are available at <https://secure.xilinx.com/webreg/clickthrough.do?cid=119758>. Open the ZIP archive and extract the file `xapp1014_c17_aes3_rx_tx.zip`.

These designs have been tested in hardware using Virtex-4 and Virtex-5 devices. The reference designs can be implemented in almost any Xilinx FPGA.

Receiver

The AES3 receiver consists of a DRU, a framer, and a data formatter. The `aes_rx` module contains all three of these submodules plus some output registers and channel demultiplexing logic, as shown in [Figure 17-1](#).

The DRU (`aes_multirate_dru`) digitally oversamples the AES3 bitstream. Refer to *Data Recovery* [Ref 14] for the description of the data recovery method. The DRU must first determine the approximate length of a state (half a bit) in the bitstream. Logic in the DRU determines how many clock cycles are present in the minimum state width found in the AES3 bitstream. After this measurement is taken, the DRU uses that information to pick a sample point approximately in the center of the state to sample each state's value. The DRU constantly measures the minimum state length so that it can automatically adjust when the input audio sample rate changes.

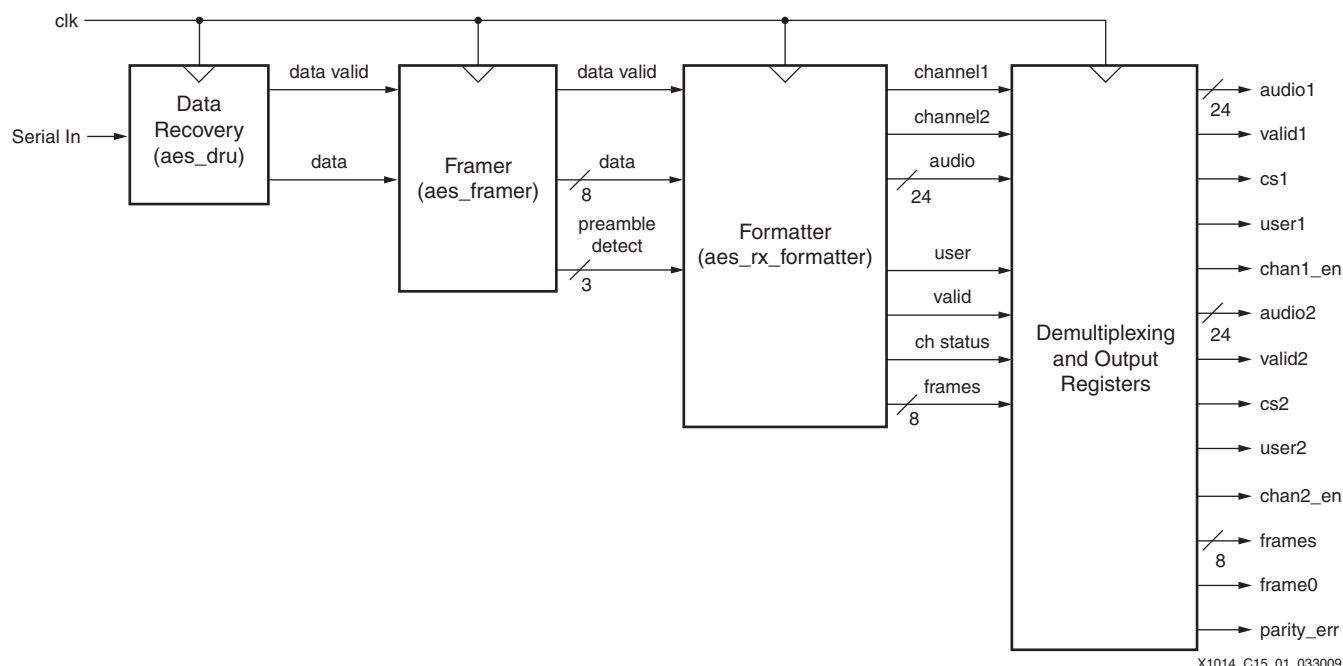


Figure 17-1: AES3 Receiver Block Diagram

The DRU produces one recovered bit every time it asserts its `data_valid` output. (In this case, a recovered bit is one state of a biphasemark encoded symbol.) The rate at which the `data_valid` output is asserted is dependent upon the audio sample rate. For example, with 192 KHz audio, the rate averages 24.576 MHz.

To provide good receiver jitter tolerance, it is recommended that the clock used for the `aes_dru` module be fast enough to sample the AES3 bitstream at least four times during each state of a biphasemark encoded bit for the highest supported audio sample rate. For example, if 192 KHz is the fastest supported audio sample rate in an application, a 100 MHz clock is sufficient. The bit rate of 192 KHz audio is 12.288 Mb/s, and the biphasemark state rate is 24.576 MHz. A 100 MHz clock is just over four times 24.576 MHz. This 100 MHz clock is used by the DRU to receive all AES3 sample rates 192 KHz and lower. A DCM can be used to multiply a lower-frequency clock to frequencies sufficient to oversample the AES3 bitstream.

A previous version of the AES3 receiver reference design used a DRU that had a fixed 4X oversampling rate. This required a different clock frequency for each AES3 sample rate. This DRU was, however, somewhat smaller than the new multi-rate DRU and could be useful for certain applications where the audio sample rate is always fixed, typically at 48 KHz. The older DRU is still provided in the reference design files and is called `aes_dru_fixed_4X`. The two DRUs are interchangeable because they have exactly the same ports. When using `aes_dru_fixed_4X`, the clock provided to the AES receiver must be very close to four times the biphasemark bit rate of the desired bitstream. For 48 KHz sample rate audio, the clock should be 24.576 MHz when using the fixed-rate DRU.

The recovered bits from the `aes_dru` module are sent to the framer module (`aes_framer`). This module detects the preamble sequences so that the data can be aligned properly to the subframe boundaries. The framer decodes the biphasemark symbols and then deserializes the data into 8-bit bytes properly aligned so that each

subframe is output from the framer in four consecutive bytes. The framer data_valid output is asserted whenever a byte of data is output on the framer data port.

The formatter module (aes_rx_formatter) takes the data from the framer and formats it into audio sample words and valid, channel status, and user data bits for each of the two audio channels. It also detects parity errors.

Table 17-1 provides details of the aes_rx module input and output ports. The two channels can be output from the module in a multiplexed or demultiplexed manner. An input port called mux_mode tells the aes_rx module which output mode to use.

Table 17-1: Input and Output Ports of aes_rx Module

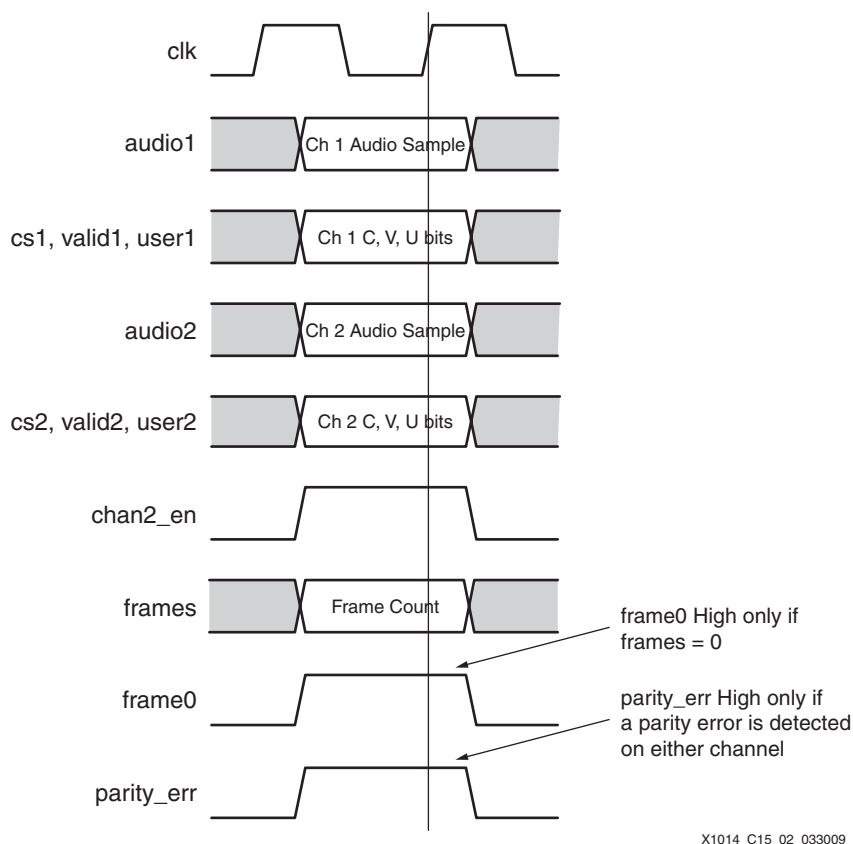
Signal	Direction	Description
clk	In	This is the oversampling clock. See the DRU description for clock frequency requirements.
rst	In	This is an asynchronous reset.
din	In	This is the AES3 serial input.
mux_mode	In	When this input is Low, the two output channels are demultiplexed. When this input is High, the two output channels are multiplexed onto the channel 2 output ports.
locked	Out	This port is High when the receiver is locked to the AES3 bitstream.
chan1_en	Out	In multiplexer mode, this output is High when the channel 1 data is present on the channel 2 output ports. In demultiplexer mode, this output is not used.
audio1[23:0]	Out	In demultiplexer mode, the audio word for channel 1 is present on this output port when chan2_en is High. In multiplexer mode, this output port is not used.
valid1	Out	In demultiplexer mode, the valid bit for channel 1 is present on this output port when chan2_en is High. In multiplexer mode, this output port is not used.
user1	Out	In demultiplexer mode, the user data bit for channel 1 is present on this output port when chan2_en is High. In multiplexer mode, this output port is not used.
cs1	Out	In demultiplexer mode, the channel status bit for channel 1 is present on this output port when chan2_en is High. In multiplexer mode, this output port is not used.
chan2_en	Out	In multiplexer mode, this output is High when the channel 2 data is present on the channel 2 output ports. In demultiplexer mode, this output is High when the channel 1 data is present on the channel 1 output ports and the channel 2 data is present on the channel 2 output ports.
audio2[23:0]	Out	This output port carries the audio for channel 2 in demultiplexer mode and for both channels in multiplexer mode.
valid2	Out	This output port carries the valid bit for channel 2 in demultiplexer mode and for both channels in multiplexer mode.
user2	Out	This output port carries the user data bit for channel 2 in demultiplexer mode and for both channels in multiplexer mode.

Table 17-1: Input and Output Ports of `aes_rx` Module (Cont'd)

Signal	Direction	Description
cs2	Out	This output port carries the channel status bit for channel 2 in demultiplexer mode and for both channels in multiplexer mode.
parity_err	Out	This port is asserted High when a parity error is detected on either channel.
frames[7:0]	Out	This port indicates the frame number of the data present on the output ports.
frame0	Out	This port is asserted High when the frames port is deasserted.

In multiplexed mode, the audio data for both channels is output on the audio2 port. Likewise, the valid, user data, and channel status bits for both channels are output on the valid2, user2, and cs2 ports, respectively. When the data for channel 1 is present on these output ports, the chan1_en signal is High. When the data for channel 2 is present on the output ports, the chan2_en signal is High. These two enable signals can be used as clock enables to the downstream logic.

In demultiplexed mode, the data for each channel is output from the receiver on separate ports. The chan2_en output is High when the data for both channels is present on the output ports. The chan1_en output is not used in this mode. Refer to Figure 17-2 for timing details of the demultiplexed mode and to Figure 17-3 for timing details of the multiplexed mode.

Figure 17-2: Timing Diagram of `aes_rx` Demultiplexed Mode

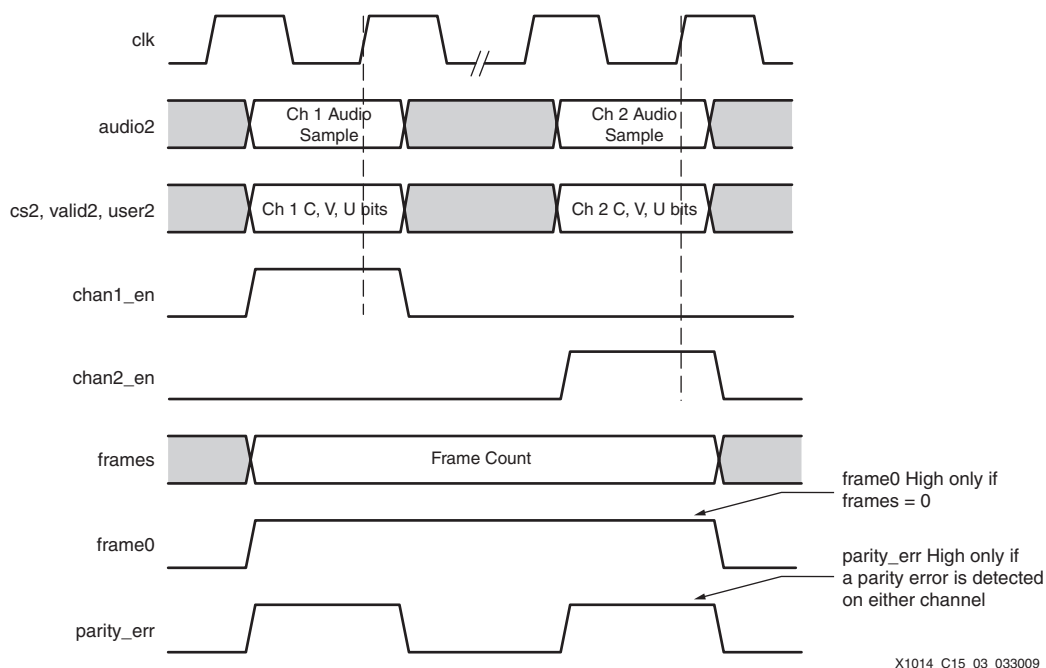
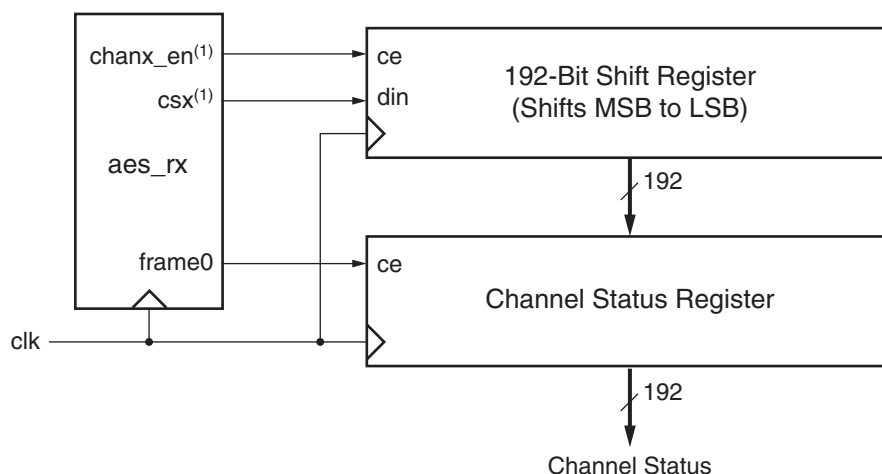


Figure 17-3: Timing Diagram of `aes_rx` Multiplexed Mode

The channel status and user data bits are grouped by the AES3 standard into 192-bit blocks. There are 192 bits of channel status and user data for each channel. The `aes_rx` module does not format the channel status and user data bits into 192-bit wide parallel data. These bits are only used occasionally and when they are used, only some of the bits are useful. Thus, it is wasteful for the reference design to format these into 192-bit parallel data when many applications do not use all of the bits. Instead, for each channel, the `aes_rx` module provides the channel status and valid bits as single output ports that are valid when the associated audio data word is present on the receiver module output. The `aes_rx` module also provides two ports, `frames` and `frame0`, that can be used by custom logic to store and format the channel status and user data as required by a particular application.

The 8-bit `frames` port indicates the current frame number from 0 to 191. The `frame0` output is asserted High when `frames` equals zero. This information can be used in many ways to format the channel status and user data. For example, Figure 17-4 shows how to deserialize the channel status and user data information using a 192-bit shift register. The shift register shifts in one bit into its MSB when the proper `chan_en` output from the receiver is asserted. When the `frame0` output is High, the data from the shift register is transferred into a 192-bit wide output register. This brute force scheme makes all the data bits available to the application at any time. However, it does require 384 flip-flops for the channel status and user data for each channel, which is a total of 1,536 flip-flops for all channel status and user data bits for both channels.

**Notes:**

1. In multiplexed mode, use chan1_en and cs2 to capture C bits for channel 1 and use chan2_en and cs2 for channel 2. In demultiplexed mode, use chan2_en and cs1 to capture C bits for channel 1 and use chan2_en and cs2 for channel 2.

X1014_C15_04_033009

Figure 17-4: Brute Force Deserialization of C and U Data

Other methods of capturing and storing the channel status and user data are more efficient and are more suited to particular applications. In [Figure 17-5](#), a dual-port memory made from the distributed RAM of Xilinx FPGAs holds the channel status or user data information in a very efficient manner. In this example, an 8-bit wide dual-port memory, 32 locations deep, is made from eight RAM32X1D primitives. The three LSBs of the frames port are decoded and used as write enables so that data bits from the channel status or user data ports are written one bit at a time. Bits [7:3] of the frames port are connected to the address lines of the RAM32X1D primitives. After it is written to the RAM, the data can be read one byte at a time through the second port of the memory. This scheme is particularly useful if the data is to be read by a PicoBlaze™, MicroBlaze™, or PowerPC® processor in the FPGA. The processor can read the data bytes in random order as needed. The frame0 output of aes_rx could be used as an interrupt to the processor to tell it when all 192 bits have been written into the dual-port RAM.

The example in [Figure 17-5](#) shows how efficient the distributed memory of the Xilinx FPGA architecture can be. It uses just 40 LUTs to capture and store the channel status or user data for a channel, 32 LUTs for the dual-port RAM, and another 8 LUTs for the bit enable decoding logic. If both the channel status and user data are being captured, or this data is being captured for both channels, the 8 LUTs used to decode the RAM write enables can be reused for each dual-port memory array. Thus, it takes 136 LUTs to capture and store both the channel status and user data for both channels.

The aes_rx module has an output signal called locked. This signal goes High when the receiver locks to a valid AES3 bitstream. The locked signal goes High when the first Y preamble is detected and stays High as long as Y preambles are detected periodically. The locked signal is not intended as a bitstream validity signal, but as a general indicator that the receiver is detecting an AES3 bitstream.



Figure 17-5: Using Dual-Port Distributed Memory for C and U Data

Transmitter

The AES3 transmitter module (`aes_tx`) takes in the audio data words for both channels along with the user data, channel status, and valid bits. The `aes_tx` module formats, encodes, and serializes the data. It also calculates parity bits for each subframe. However, it does not generate CRC values for the channel status data. These CRC values need to be calculated externally and fed to the transmitter serially on the channel status input ports at the appropriate time. The transmitter has separate input ports for the two channels and does not have a multiplexed mode like the receiver section.

Table 17-2 shows the input and output ports of the `aes_tx` module. The `clk` input port must be a multiple of the AES3 bit rate. The transmitter must also be provided with three clock enables: `ce_bp`, `ce_bit`, and `ce_word`. The `ce_bp` clock enable must be asserted at the AES3 bitstream symbol rate (twice the bit rate). The `ce_bit` clock enable must be asserted at the bit rate, and the `ce_word` clock enable must be asserted at the audio sample rate.

Table 17-2: Input and Output Ports of `aes_tx` Module

Signal	Direction	Description
<code>clk</code>	In	Transmitter reference clock. Must run at a multiple of the bit rate. The minimum clock frequency for 48 KHz sample rate audio is 6.144 MHz.
<code>rst</code>	In	This is an asynchronous reset.
<code>ce_word</code>	In	This is a word (sample) rate clock enable that must be asserted High for one cycle of <code>clk</code> at the sample rate. For example, if <code>clk</code> is 6.144 MHz and the sample rate is 48 KHz, <code>ce_word</code> must be asserted once every 128 cycles of <code>clk</code> . The audio, valid, user, and channel status data are all loaded into the transmitter when <code>ce_word</code> is High.
<code>ce_bit</code>	In	This is a bit rate clock enable that must be asserted High for one cycle of <code>clk</code> at the bit rate. For example, if <code>clk</code> is 6.144 MHz, <code>ce_bit</code> must be High every other cycle of <code>clk</code> .
<code>ce_bp</code>	In	This is a symbol rate clock enable that must be asserted at twice the bit rate. For example, if <code>clk</code> is running at 6.144 MHz, <code>ce_bp</code> must always be High.
<code>audio1[23:0]</code>	In	This is the channel 1 audio sample word.
<code>valid1</code>	In	This is the channel 1 valid bit. If the channel is valid, this input must be Low.
<code>user1</code>	In	This is the channel 1 user data bit.
<code>cs1</code>	In	This is the channel 1 channel status bit.
<code>audio2[23:0]</code>	In	This is the channel 2 audio sample word.
<code>valid2</code>	In	This is the channel 2 valid bit. If the channel is valid, this input must be Low.
<code>user2</code>	In	This is the channel 2 user data bit.
<code>cs2</code>	In	This is the channel 2 channel status bit.

Table 17-2: Input and Output Ports of aes_tx Module (Cont'd)

Signal	Direction	Description
frame0	In	This input must be High once every 192 sample words (frames). The frame0 input tells the transmitter that the current frame is the first frame of a block. When frame0 is High, the cs and user inputs must be driven with the LSB of the 192-bit channel status and UDWs.
serout	Out	This is the encoded AES3 output bitstream.

Transmitter clocking example

A 24.576 MHz clock source is supplied to the clk port of the aes_tx module. The audio sample rate is 48 KHz. (Table 16-1, page 350) shows that the AES bitstream bit rate carrying 48 KHz audio is 3.072 Mb/s, and the bitstream symbol rate is 6.144 MHz. The 24.576 MHz clock is exactly four times faster than the symbol rate. Therefore, ce_bp must be asserted once every four clock cycles and ce_bit must be asserted once every eight clock cycles (Figure 17-6). The ce_word clock enable is asserted once every 512 clock cycles ($24.576 \text{ MHz} / 512 = 48 \text{ KHz}$).

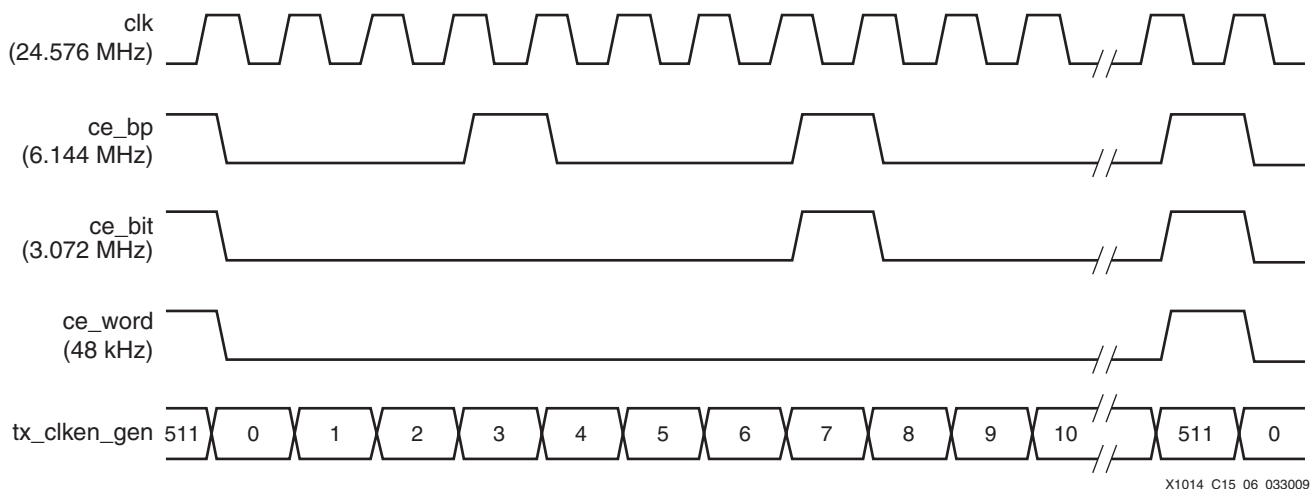


Figure 17-6: Transmitter Timing Diagram (24.576 MHz Clock and 48 KHz Sample Rate)

The Verilog code to produce the three clock enables for 48 KHz sample rate audio with a 24.576 MHz clock is given here:

```

reg [8:0] tx_clken_gen = 9'd0; // 9-bit counter
wire ce_bp; // 6.144 MHz clock enable
wire ce_bit; // 3.072 MHz clock enable
wire ce_word; // 48 KHz clock enable

always @ (posedge gclk_audio_tx)
    tx_clken_gen <= tx_clken_gen + 9'd1;

assign ce_bp = &tx_clken_gen[1:0]; // asserted at 2X bit rate
assign ce_bit = &tx_clken_gen[2:0]; // asserted at bit rate
assign ce_word = &tx_clken_gen; // asserted at sample rate

```

The minimum frequency for the aes_tx clock is equal to the AES3 bitstream symbol rate. For 48 KHz sample rate audio, the minimum clock frequency is 6.144 MHz.

The `aes_tx_clkdiv` module, included with the reference design, also provides code to generate the three clock enables required by the AES transmitter. When provided with a 24.576 MHz clock, this module can produce the correct clock enables for 192 KHz, 96 KHz, 48 KHz, 32 KHz, and 16 KHz sample rates. With a 22.5792 MHz clock, the module can produce clock enables for 176.4 KHz, 88.2 KHz, 44.1 KHz, and 22.05 KHz sample rates.

The channel status inputs (`cs1` and `cs2`) and the user data inputs (`user1` and `user2`) of the transmitter are serial in nature instead of each being 192 bits wide. The transmitter must be supplied with the 192-bit channel status and user data serially through these ports, with one channel status bit and one user bit provided with each audio sample. As with the receiver section, this can be done by different methods. One of these methods is by brute force using 192-bit shift registers. If the transmitter is retransmitting the data from an `aes_rx` module, the channel status and user data inputs can be connected directly to the corresponding serial outputs of the `aes_rx` module.

The `frame0` input port is used to tell the transmitter when the first bit (LSB) of the 192-bit channel status and UDWs are present on the inputs. The `frame0` input, when High, causes the transmitter module to insert a Z preamble for the first subframe of the current frame, marking this frame as the first frame of a 192-frame block. The `frame0` input must be asserted once every 192 frames.

Channel Status CRC

The AES3 document defines the last byte (byte 23) of the channel status data as an 8-bit CRC value calculated from the other 23 bytes of channel status data. The AES3 document also states that the channel status CRC value is optional and, when not used, the value of byte 23 must be set to zero.

The reference design includes a module for generating and checking channel status CRC values. This module, called `aes_crc`, can be used in an AES3 receiver for checking the channel status CRC value. It can also be used to generate the channel status CRC value to be inserted into channel status byte 23 in an AES3 transmitter.

The `aes_crc` module works serially, accepting a channel status bit on its input once per frame and generating the AES3 channel status CRC value using the polynomial in [Equation 17-1](#).

$$G(x) = x^8 + x^4 + x^3 + x^2 + 1 \quad \text{Equation 17-1}$$

In a receiver, the `aes_crc` module compares the 8 bits of the CRC value in byte 23 of the input data against the CRC value calculated by the module. This comparison is done one bit at a time during frames 184 through 191. The module asserts the `crc_err` output High if the CRC values are not the same. [Figure 17-7](#) shows how to use two `aes_crc` modules to check the two channel status outputs of the `aes_rx` module.

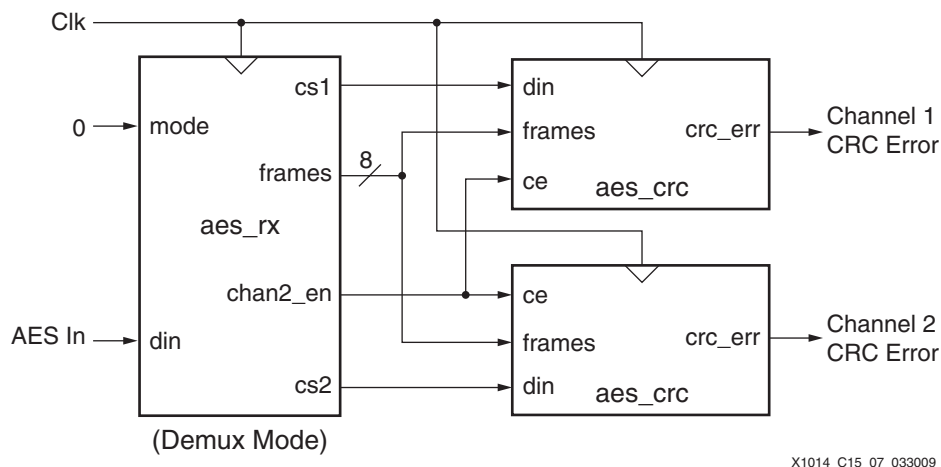


Figure 17-7: Checking C CRC in an AES Receiver

In a transmitter, the dout port provides the complete channel status serial bitstream with the CRC bits inserted. During bytes 0 through 22, the `aes_crc` module passes the `din` signal through to `dout`, but during byte 23, the calculated CRC bits are output on `dout`. Thus, the `dout` port of the module can be directly connected to the `cs1` or `cs2` input of the `aes_tx` module. The `aes_crc` module is used to calculate and insert the channel status CRC bits for an AES transmitter (Figure 17-8). The `frame191` output port of the `aes_crc` module is used to control when an 8-bit frame counter rolls over to zero after reaching a count of 191. The frame counter is only required if a frame count is not supplied to the transmitter from some other source. The `frame191` output can be delayed by one frame to become the `frame0` input required by the `aes_tx` module.

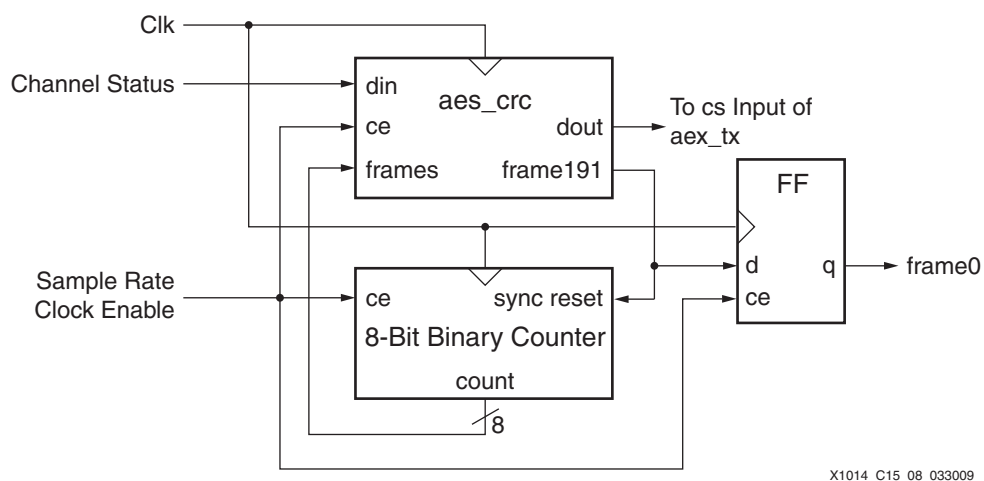


Figure 17-8: Generating C CRC for an AES Transmitter

CRC generation and checking can also be done by other methods. If a processor is being used to create or read the CRC data, the CRC generation or checking can be done in software. For example, if the channel status data is captured into dual-port RAM, the processor reading the data from the dual-port RAM after it has been captured can calculate the CRC and compare it to the captured CRC value (Figure 17-6, page 381). The ports of the `aes_crc` module are listed in Table 17-3.

Table 17-3: Input and Output Ports of `aes_crc` Module

Signal	Direction	Description
clk	In	This is the clock input. The module must be clocked at the audio sample rate.
ce	In	This is the word (sample) rate clock enable.
frames[7:0]	In	This is the frame count input. This 8-bit port must indicate the current frame number so that the module knows when to calculate the CRC and when to insert it.
din	In	The serial channel status bitstream enters the module here. One channel status bit is clocked into the module on every frame every time the ce signal is asserted.
dout	Out	The serial channel status bitstream with the CRC bits inserted is output from the module on this port.
crc_out_en	Out	This signal is High during frames 184 through 191, indicating that the CRC bits are being output from the dout port.
frame191	Out	This output is High when the frames input port equals 191.
crc_err	Out	This output is High if a CRC error is detected. During frames 184 through 191, the din port is compared to the CRC output bit. If they do not match, the crc_err output is asserted High.

Electrical Interface

The AES3 document describes a 110 Ω balanced electrical signal carried on shielded twisted pair (STP) cable with XLR connectors. The 110 Ω balanced interface is typically implemented with RS-422 drivers and receivers with transformers to provide impedance matching. Rise-time limiting components are often added to the transmitter interface to meet the rise-time requirements of the AES3 specification. The RS-422 drivers and receivers can easily be interfaced to the input and output buffers of Xilinx FPGAs.

The AES-3id-2001 document describes how to use unbalanced 75 Ω coaxial cable with BNC connectors to transport AES3 audio [Ref 15]. This unbalanced AES-3id variant is heavily used in the video broadcast industry. AES-3id is identical to AES3 and AES/EBU in data format and bit rate, differing only in the electrical interface details. Refer to the AES-3id document for details of building AES-3id compliant electrical interfaces.

If S/PDIF compliant electrical interfaces are used, the AES modules in this reference design can be used to implement S/PDIF interfaces.

Jitter

The `aes_rx` module was tested with a dScope Series III audio analyzer from Prism Sound and found to exceed the AES3 receiver input jitter tolerance requirements. The jitter tolerance of the multi-rate AES DRU is dependent upon the frequency of the oversampling clock and the jitter on this clock. When using oversampling clocks of 66 MHz and 100 MHz produced by the DCM from a 33 MHz clock source, the jitter tolerance of the receiver exceeded 20 UI to at least 4 KHz for all audio sample rates up to 96 KHz (the maximum sample rate provided by the dScope III) and exceeded 10 UI to at least 9 KHz. Above 10 KHz, the dScope III can add only up to 0.5 UI of jitter (except at 96 KHz, where the

maximum is 0.375 UI). The AES receiver tolerated these maximum jitter levels without error.

Transmitter output jitter is primarily a function of the jitter on the transmitter clock. Because the bit rates of AES3 are relatively low, the absolute jitter budget is quite large compared to most serial standards implemented with FPGAs these days. Thus, it is very easy to meet the AES3 transmitter output jitter requirements with Xilinx FPGAs.

Module FPGA Resource Usage

Table 17-4 shows the amount of FPGA resources required to implement the AES3 receiver and transmitter modules. The results for `aes_rx` include all the submodules that make up the receiver including `aes_dru`, `aes_framer`, and `aes_rx_formatter`. These results do not include any logic external to these modules such as transmitter clock enable generation or logic to collect and process channel status or user data bits. The `aes_rx` and `aes_tx` modules also do not include any channel status CRC generation or checking. However, Table 17-4 does list the resource usage for one `aes_crc` module. One `aes_crc` module is required for each channel status value to be checked or generated. These results were obtained using ISE® software, version 9.2 SP4 and targeting a Virtex-5 device.

Table 17-4: FPGA Resource Usage for AES3 Modules

Module	Flip-Flops	LUTs
<code>aes_rx</code> (fixed-rate DRU)	259	102
<code>aes_rx</code> (multi-rate DRU)	270	152
<code>aes_tx</code>	102	58
<code>aes_crc</code>	8	12

Conclusion

AES3-compliant digital audio receivers and transmitters are easily implemented in Xilinx FPGAs. The reference designs provided with this chapter use very few FPGA resources and can be used with any Xilinx FPGA family. This allows multiple AES3 transmitters and receivers to be implemented in one FPGA with plenty of logic resources left over for other functions.

Asynchronous Sample Rate Converter

Summary

The asynchronous sample rate converter (ASRC) reference design converts stereo audio from one sample frequency to another. The input and output sample frequencies can be arbitrary fractions of one another or they can be at the same frequency but based on different clocks. The output is a band-limited version of the input that is resampled to match the output sample timing. The reference design has these features:

- Fully asynchronous
- THD+N typically –130 dB and ranges from –125 dB to –139 dB
- 24-bit audio word width in and out
- Automatic ratio detection
- Up-conversion, down-conversion, and 1:1 asynchronous conversion
- Rate change tracking
- Sample clock jitter rejection. Retains full performance over AES3 jitter tolerance curve [\[Ref 16\]](#)
- Input rates from 8 KHz to 192 KHz, continuous
- Output rates from 8 KHz to 192 KHz, continuous
- Conversion ratio 1:7.5 (down) to 8:1 (up), continuous
- Low deterministic latency
- Lock status outputs provided for external muting

The design is implemented in the Virtex®-5 FPGA architecture. It uses a DSP48e slice as the main math element, and block RAM for input sample buffers and storage of the prototype filter.

Structure

The ASRC reference design consists of two main functions: the ratio control and the resampler. These main functions are further divided into smaller functional units, as shown in [Figure 18-1](#). The ratio control function has two main subunits: ratio detection and input sample storage. The resampler has two main parts: interpolation of the correct phase of the filter, and the FIR filter operation that applies the calculated filter coefficients to the set of input samples to form an output sample. The HDL breaks the operations down into modules with functional boundaries. This chapter explains what the modules are and how they fit together, and also provides details of the functional blocks.

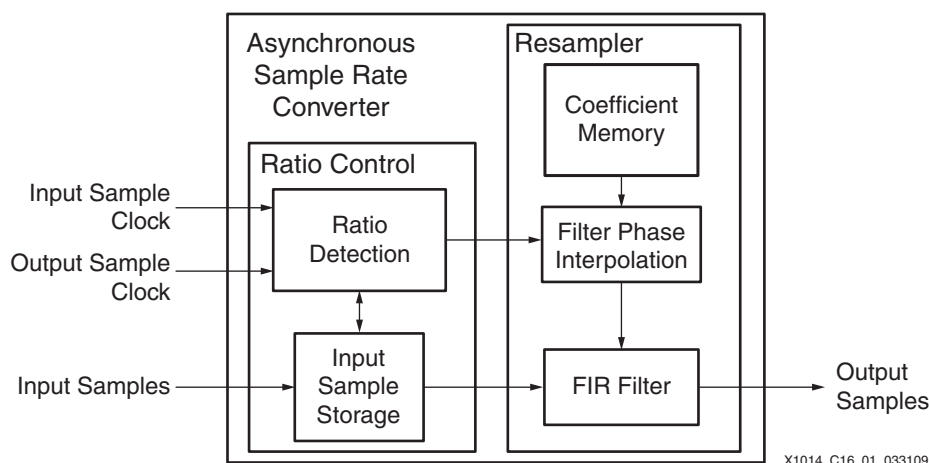


Figure 18-1: ASRC Top-Level Block Diagram

Modules

Table 18-1 lists the modules in the ASRC reference design.

Table 18-1: Reference Design Modules

Module	Description
asrc_gold	This is the top-level wrapper that instantiates and connects the lower level modules and provides the I/O interface.
timing_control	This module contains the master state machine that controls the creation of each output sample. It instantiates the divider that is used for ratio calculation and normalization of the output samples.
shared_divider_v1_0	This is a 27 x 27 bit signed serial divider. The quotient has 27 integer and 26 fractional bits. This divider is used to: <ul style="list-style-type: none"> Calculate the ratio of the output sample rate to the input sample rate Normalize output samples based on the sum of the input coefficients
ring_buffer_gold	This module provides pointers and control for the ring buffer memory. The ring buffer stores incoming samples and provides the sample stream to fir_gold.
buffer_mem_gold	This is a 512 x 48 dual-port block RAM for the ring buffer.
ratio_calc	This module contains the counters for determining input and output sample rates. These rates are sent to shared_divider_v1_0 and the calculated ratio is returned. The ratio_calc module also determines the feedback error term based on the FIFO level and regulates the ratio accordingly.
ratio_filt	This module instantiates the moving_ave_26 module. It also determines when a new ratio has been calculated and when the filter should be bypassed.

Table 18-1: Reference Design Modules (Cont'd)

Module	Description
moving_ave_26	This module performs a 16-tap moving-average filter on the calculated ratio.
filt_interp_gold	This module performs the Lagrange interpolation on the prototype filter. It interpolates a filter coefficient for every input sample in the FIR filter operation.
filt_mem_gold	This is a 2048 x 24 single-port ROM containing the prototype filter. The prototype filter is symmetrical with 4,097 coefficients, of which the middle coefficient is stored separately.
MULT35X35_SEQUENTIAL_PIPE	This is a 35 x 35 multiplier using four sequential states in a DSP48e slice.
mult_one_third	This fixed multiplier module implements a divide-by-3 on a 24-bit number.
fir_gold	This module performs a 64-tap FIR filter for each output sample. Data comes from the ring_buffer_gold module and coefficients come from filt_interp_gold.

Figure 18-2 shows the hierarchy of the modules and their relation to the functional blocks.

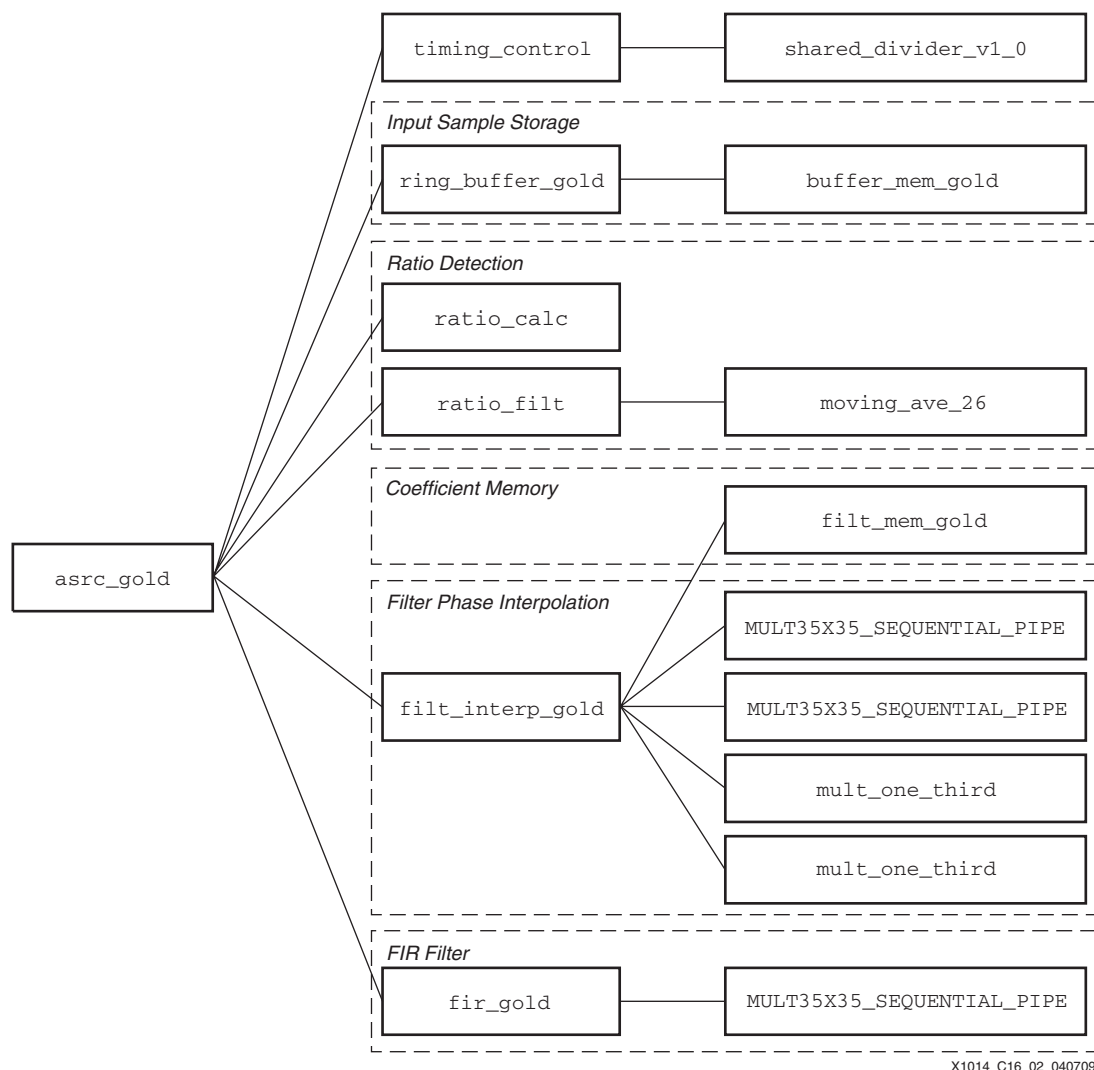


Figure 18-2: Reference Design Module Hierarchy and Relation to Functional Blocks

Functional Description

This section contains a functional description of the ASRC reference design.

Ratio Control Functional Block

The ratio control functional block uses one of two algorithms depending on whether the input rate is changing. At start-up and whenever the input or output rate changes, rate-change tracking mode is used to quickly adjust to the correct ratio and to adjust the input FIFO to the proper level. In this mode, the ratio correction term grows exponentially with error to quickly track large rate changes and reduce the error to low levels. This is illustrated in Figure 18-3.

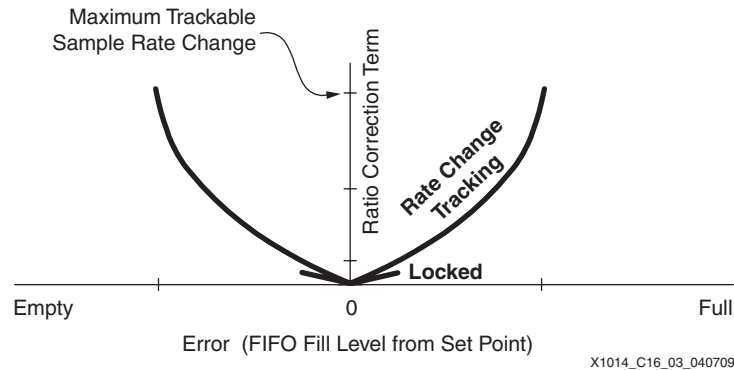


Figure 18-3: Error Correction Curves

After a small error has been achieved and the ratio is stable, an automatic switch is made to locked mode. This mode limits the amount and rate of change of the ratio to achieve maximum audio quality. The locked mode can track small drifts in the clock frequencies. However, if a large rate change occurs, the error term exceeds the locked range and the mode automatically shifts to rate change tracking. When small error is achieved and the ratio is stable, the switch to locked mode occurs again. In this manner, changes in the sample frequencies, large and small, are continuously and smoothly tracked.

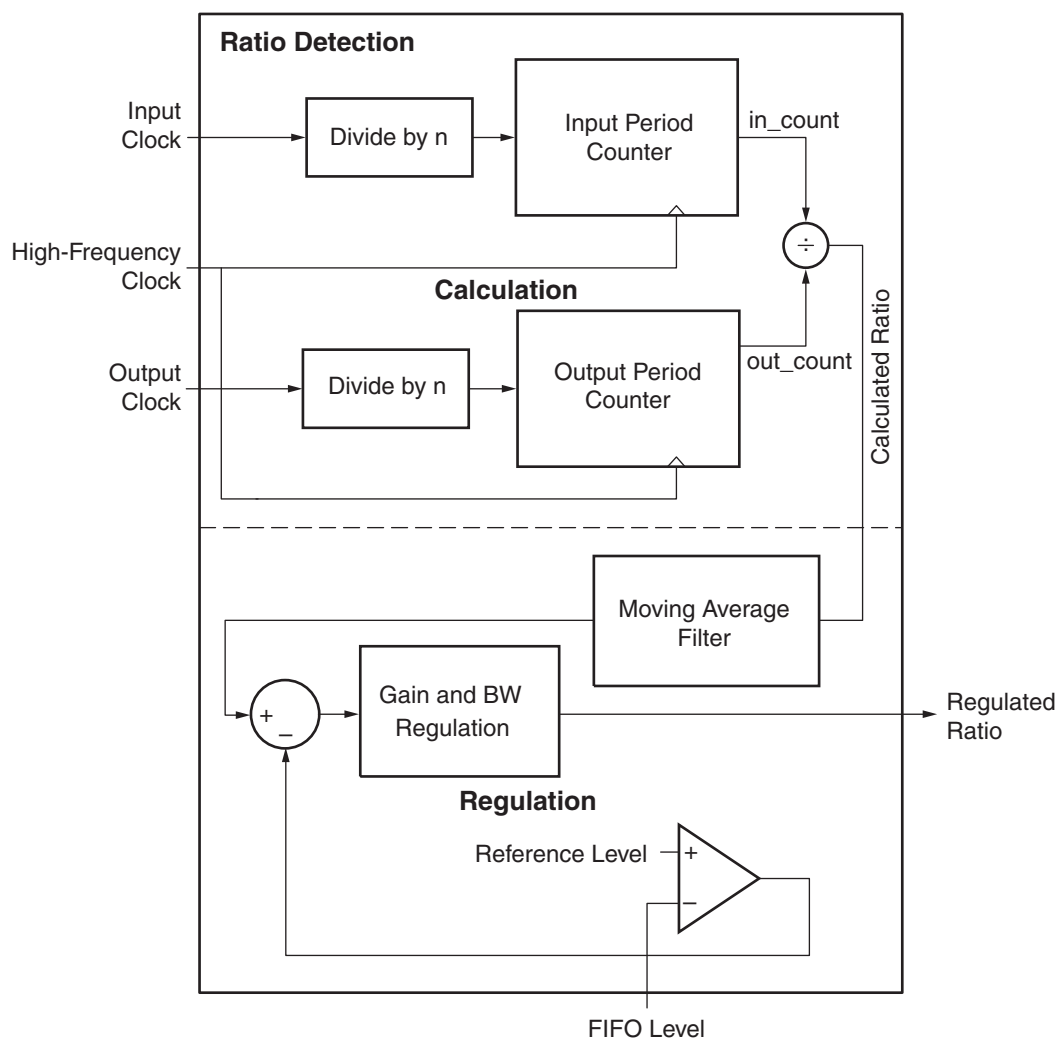
The input samples are buffered by the `ring_buffer_gold` module in the ratio control section. When a new output sample is required, the set of input samples required for the FIR filter convolution are sent to the resampler.

Ratio Detection

The ratio detection block is implemented in the `ratio_calc` and `ratio_filt` modules of the reference design. A ratio is computed by measuring the period of the input and output clock with a high-frequency clock that, in general, is not related to either the input or output clocks. This is shown in the top section of Figure 18-4. To improve the accuracy of this calculation and mitigate the effects of jitter, the input and output clocks are measured over 1,024 cycles. The input period is divided by the output period to obtain a calculated ratio.

To further attenuate sample clock jitter, the calculated ratio passes through a moving-average filter contained in the `ratio_filt` module. The moving-average filter is only applied during locked mode when the input frequency is stable. In frequency tracking mode, the moving-average filter is bypassed. The most current calculated ratio is used for ratio regulation.

To regulate the level of the input FIFO and thus, the latency, the FIFO fill level is compared to a reference level in the regulation section. The difference in levels is used as an error signal to adjust the ratio. Because the ratio determines the position of each new output sample relative to the input samples, it effectively controls the speed at which input samples are processed.

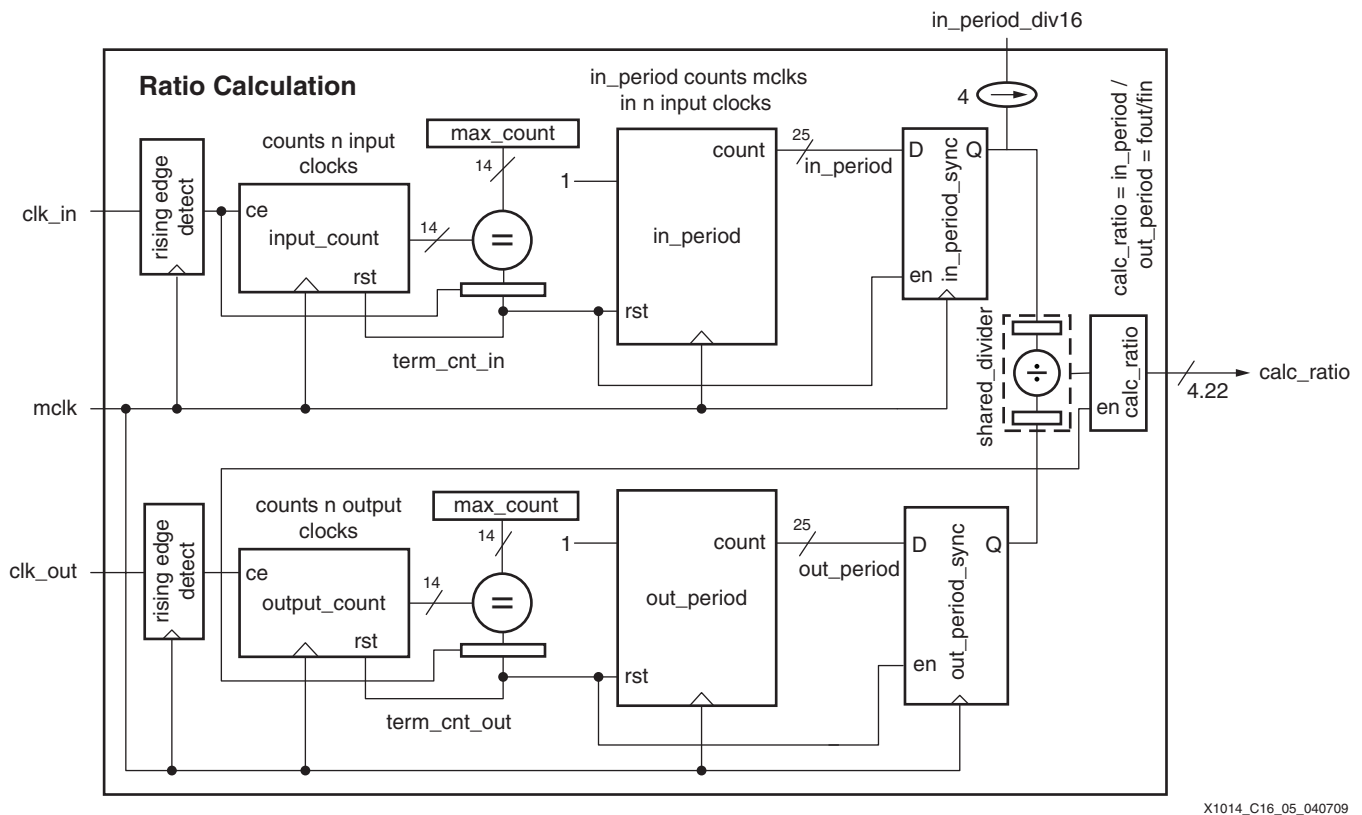


X1014_C16_04_040709

Figure 18-4: Ratio Detection Block Diagram

Ratio Calculation

Figure 18-5 is a detailed block diagram of the ratio calculation portion of the reference design.

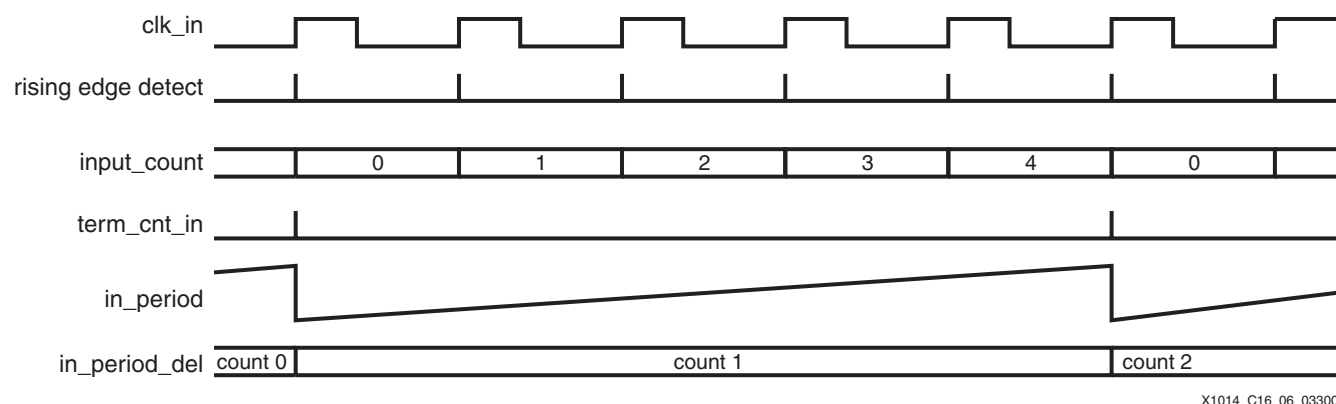


X1014_C16_05_040709

Figure 18-5: Ratio Calculation Detailed Block Diagram

Figure 18-6 illustrates how the input period measurement is made. The input_count block counts input clocks on each rising edge. The max_count value specifies how many input clocks to count before resetting the counter. It is a parameter in the reference design and is nominally set to 1024. The signal term_cnt_in pulses once every max_count + 1 input clocks. This signal resets the in_period counter as well as input_count. The in_period counter counts the number of master processing clock (mclk) cycles that occur during max_count + 1 input clocks. At every pulse of term_cnt_in, the in_period_sync register stores the latest in_period count and the counting begins again. The in_period count resets to 1 so that the resulting count is the actual number of mclk cycles over the specified period, not number of cycles - 1. The in_period_sync value is shifted right by four and sent to the sample storage section as in_period_div16.

The output clock period is measured in the same fashion. The ratio is calculated based on the timing of the output clock. To obtain the calculated ratio, calc_ratio, in_period_sync is divided by out_period_sync. This is done by the shared_divider, a multi-cycle pipelined divider in the timing_control module. The calc_ratio signal is sent to the ratio regulation section and allows for a range of 0 to 15 with 22 fractional bits.



X1014_C16_06_033009

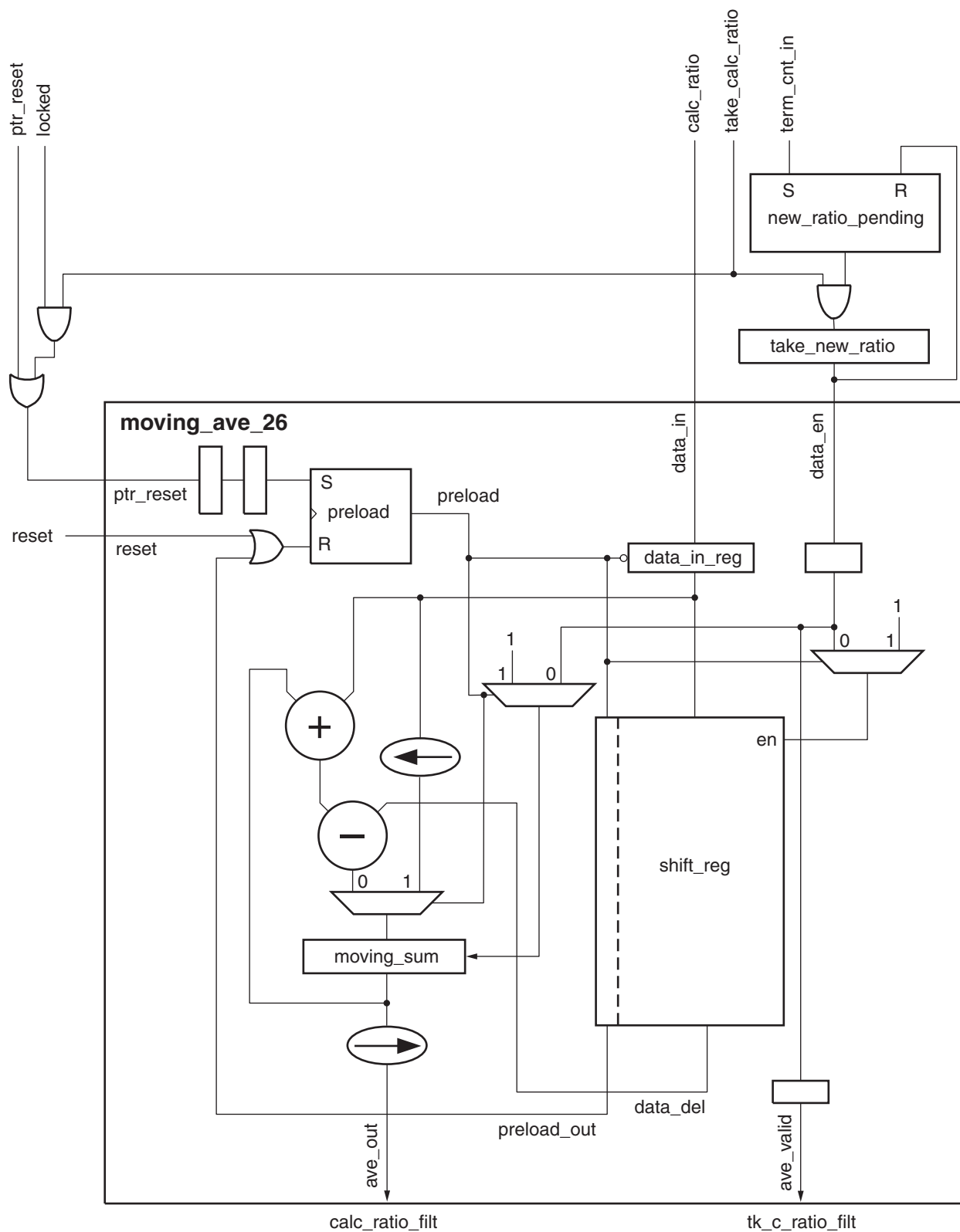
Figure 18-6: Timing Diagram of Input Period Measurement with max_count = 4

Ratio Filtering for Jitter Tolerance

The AES3 standard [Ref 16] specifies jitter tolerance for AES receivers. The AES receiver must recover the data correctly in the presence of jitter. This jitter in the timing of the audio data is propagated to the sample rate converter. Therefore, the ASRC should also have jitter tolerance equivalent to that of the AES receiver. In the reference design, the ratio is filtered for added jitter tolerance. During locked mode operation, the calculated ratio is filtered using a moving-average FIR filter to prevent short-term variations in sampling frequency from causing harmonic distortion in the output sample stream. In other words, the ratio filter attenuates the effects of input sample clock jitter. The result is no increase in distortion in the presence of jitter. Full performance is retained over the entire range of the AES3 jitter tolerance curve.

Figure 18-7 is a block diagram of the ratio filter. It is a recursive implementation requiring only one add and one subtract. As each new data point enters, it is added into the average, and the oldest data is subtracted. A shift register is used for the storage element. A 16-location shift register is implemented very efficiently in SRL16 elements requiring only one LUT per input data bit. The calculated ratio has 4 integer and 22 fractional bits. An additional bit of storage is used as a data valid to track data through the shift register. This bit is required for the preload function, which simultaneously bypasses the filter operation and preloads every location in the shift register with the current value of the input data.

When the input signal ptr_reset goes High, the preload_out flag is set High, indicating that the block is in preload mode. The preload mode prevents the data_in_register from changing, and the current data in this register is propagated directly to the moving_sum register and on to ave_out. At the same time, the shift register enable is forced active, and the shift register begins shifting in the value from data_in_reg. The preload bit also shifts through the shift register each time in parallel with the data. When the shift register has shifted the input value through every location, the preload bit is at the output of the shift register in the form of preload_out. This indicates that the preload cycle is complete. The contents of the shift register and the output register all equal the current input value in data_in_reg. This forces a reset of the preload register, which returns the module to normal filtering mode.



X1014_C16_07_040709

Figure 18-7: Ratio Filter

This preload functionality is used to bypass the moving-average filter when the ratio section is not locked (frequency tracking mode). This allows for better tracking and faster lock. When locked mode is entered, each new ratio that is calculated is averaged with the values preloaded in the shift register. The `term_cnt_in` input signals that a new input clock count has completed. The `take_calc_ratio` signal means a new output clock count has completed, as well as a divide operation to obtain `calc_ratio`. The combination of `term_cnt_in` and `take_calc_ratio` is used to form `take_new_ratio`, meaning a new data value can be taken into the moving-average filter.

Ratio Regulation

The ratio regulation section adjusts the calculated ratio to regulate the input FIFO level. [Figure 18-8](#) shows how this is done. The current FIFO level contained in `fifo_level` is compared with the target level, `fifo_setpoint`. The difference is used as the `error_term` signal that adds an offset to `calc_ratio`. The `error_term` signal is conditioned separately for locked mode and rate change mode. Parameters in the HDL specify the gain in the error term, the error dead zone, and restrictions in the ratio slew rate, if any. These parameters establish the trade-off between tight sample rate tracking and harmonic distortion performance. The trade-off for extremely tight rate-change tracking is the presence of harmonic distortion components caused by the frequent, though minute, rate adjustments.

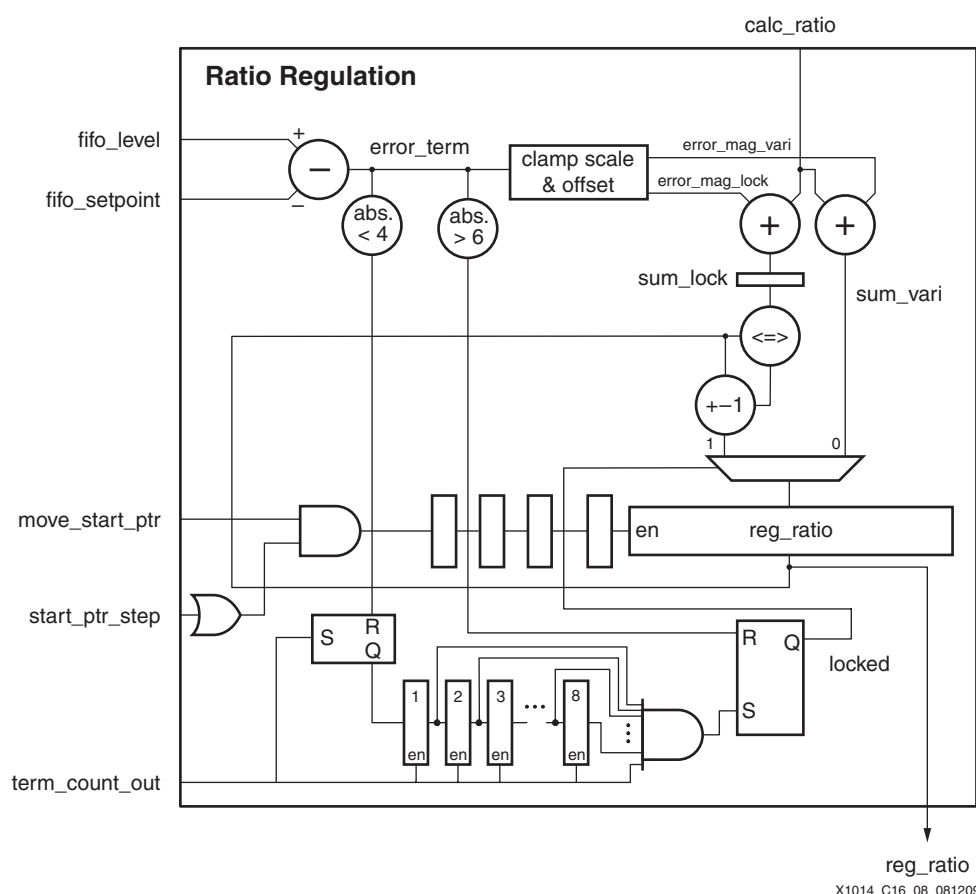


Figure 18-8: Detailed Block Diagram of Ratio Regulation Section

A small dead zone in the error term, i.e., an error threshold below which no adjustment is made to the ratio, makes rate adjustments happen less often, thereby reducing the

distortion component of the rate change. The cost is that rate-change tracking is slower and less accurate. For locked mode, the default settings in the reference design allow a small dead zone ($\frac{1}{4}$ input sample time), add no additional gain, and allow one LSB step of rate change per output sample. The frequency rate-tracking mode has no dead zone and no additional gain. This balance allows good rate-change slew and quick, reliable recovery from loss of input. At the same time, it provides good jitter rejection.

The amount and rate of the offset from the calculated ratio depends on whether or not locked mode is engaged. To enter locked mode, `error_term` must be less than four input samples for five consecutive `term_count_out` times. Unlocked mode, also called rate-change tracking mode, is entered any time the error is more than six samples.

In the rate-change tracking mode, `error_term` is added directly to `calc_ratio`. The `error_term` also has an exponential gain such that the correction factor is multiplied at higher error ratios. This facilitates faster locking at start-up and after frequency changes without dropping or repeating samples.

In the locked mode, the error term is used to increment or decrement the ratio at a maximum rate of 1 LSB per output sample. The current `reg_ratio` is fed back and compared with the target ratio, `sum_lock`. If the target is different from `reg_ratio`, 1 is either added to or subtracted from the current ratio. The `reg_ratio` value is updated when the set of input samples used for the convolution changes, indicated by a pulse on `move_start_ptr` with a non-zero value of `start_ptr_step`. This limits the slew rate of the ratio to 0.24 ppm per output sample for optimal audio performance. This mode can track slow frequency variations because `calc_ratio` is periodically updated, and the FIFO level is updated every output sample with sub-sample accuracy. This is discussed in [Ratio Control Functional Block, page 390](#). This high degree of accuracy of `fifo_level` also enables the ratio detection circuit to maintain a deterministic latency when the clocks are stable. That is, for given input and output sample rates, the latency varies by only a fraction of a sample time and the latency for any two instances of the ASRC is the same to within a fraction of a sample time.

Lock Status Indicators

Two top-level output signals indicate the status of the ratio control section: `locked` and `fifo_overflow`. These two signals can be used to mute the audio when the sample rate converter is outside its bounds of normal operation, when the input sample rate is changing, or both. When `locked` is High, ratio control is in locked mode with minimum FIFO-level error and maximum audio quality. When `locked` is Low, rate-change mode is active, meaning a more aggressive rate change tracking and correspondingly lowered THD+N performance. The `fifo_overflow` signal indicates that the input sample FIFO has overflowed or underflowed and, therefore, the output audio is corrupted. This could occur at the application or removal of the input audio stream or during extremely sharp sample rate changes.

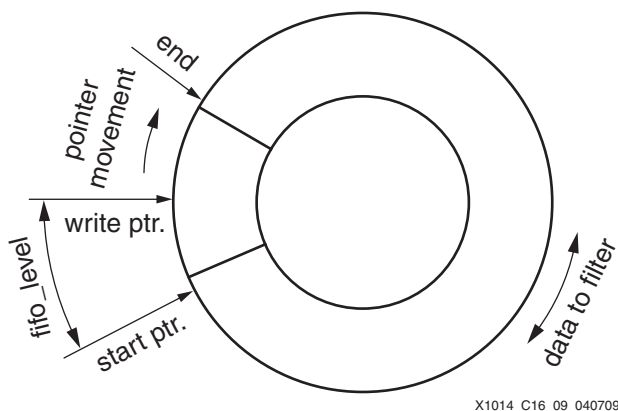
Audio quality is severely compromised when `fifo_overflow` is asserted. Depending on the application, rate-change tracking mode audio might or might not be acceptable. These two status bits are provided so that muting can be performed externally when instability in the sample rates could cause unacceptable distortion.

Input Sample Storage

The input samples are stored in a ring buffer as shown in [Figure 18-9](#), implemented in block RAM. Two data words of 24 bits each (one for each channel of the channel pair) can be stored per memory location. The ring buffer is 512 wide x 48 deep, enough to accommodate spreading the prototype filter by a factor of 7.5 with room to act as an input

FIFO. Two pointers, the write pointer and the start pointer, move through the addresses in the buffer in a circular fashion, designating the locations into which input samples are to be written and out of which input samples are to be read for the filtering operation.

Input samples are stored as they are received at the location indicated by `write_ptr`. The pointer is incremented each time a new sample is received. The first sample to be used in the FIR filter is indicated by `start_ptr`. For each output sample, a set of input samples is sent to the FIR filter, starting with the newest (at `start_ptr`) to the oldest (at `end`). The `start_ptr` is updated each time a new output sample is created.



X1014_C16_09_040709

Figure 18-9: Ring Buffer

The locations between `write_ptr` and `start_ptr` serve as an input FIFO. The difference between the two pointers is used as the `fifo_level` value. This is used as a feedback mechanism for the ratio. The net effect is to change slightly the rate at which input samples are used to keep the FIFO at a predetermined level. The level is set by parameter, and is nominally 16 in the reference design.

[Figure 18-10](#) is a block diagram of the input storage section. The ring buffer consists of the buffer memory (using dual-port block memory) and control logic for reading and writing. For the write port, a write enable pulse `write_en` is produced on the rising edge of `clkin`, the input sample clock. This pulse is used to write sample data into memory and increment the address counter to the next address.

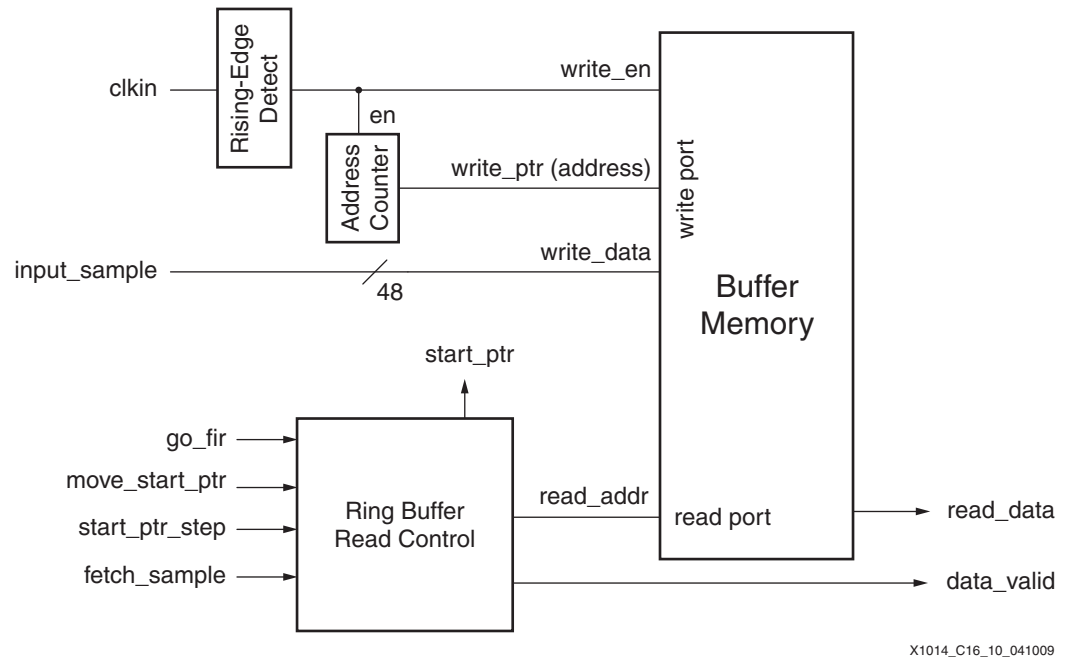
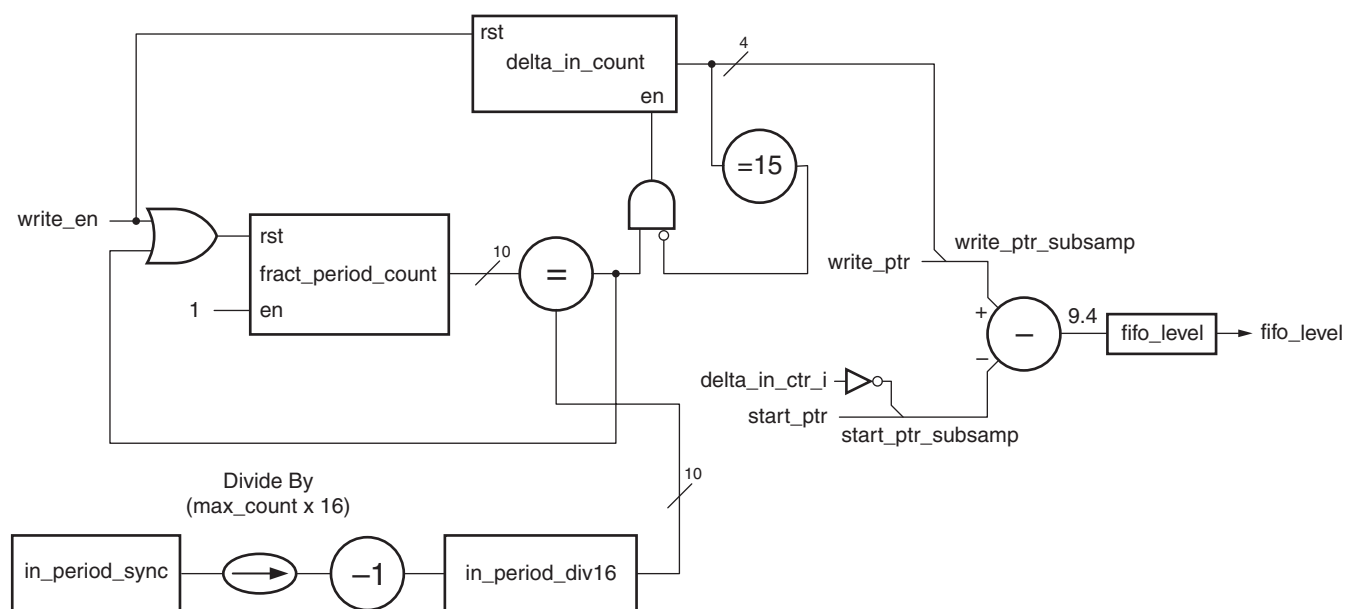


Figure 18-10: Input Buffer Storage Block Diagram

For the read port, `go_fir` signals the start of a new FIR operation to produce an output sample. This resets `read_addr` to the `start_ptr` value. The `fetch_sample` signal pulses once for each input sample used in the convolution. Each time it pulses, the `read_addr` is decremented. This sends sample data to the FIR filter in the newest-to-oldest order shown in Figure 18-9. The read control also generates an integer, `start_ptr`. The movement of this pointer is controlled by `move_start_ptr` and `start_ptr_step`. When `move_start_ptr` pulses, `start_ptr` is increased by `start_ptr_step`. The outputs of this section are input samples, `read_data`, and a `data_valid` flag.

Because `write_ptr` and `start_ptr` are updating at different times and possibly in different increments, the difference between them can vary widely even if the ratio is correct and the input and output rates are perfectly stable. For example, if the ASRC is performing a down conversion by a factor of 4, the write pointer increments four times during the time the `start_ptr` moves just once. Thus, `fifo_level` varies by 3 over the period of a single output cycle. To reduce such fluctuations, `fifo_level` is updated only after `start_ptr` is updated.

The `fifo_level` value is calculated to an accuracy of $1/16$ of a sample by creating sub-sample accurate write and start pointers. As shown in Figure 18-11, the fractional bits for `start_ptr` come from the inverse of `delta_in_ctr_i`. The `delta_in_ctr_i` signal indicates the position of the current output sample with respect to input samples. Thus, it represents a fractional start position. These bits are appended to `start_ptr` to form `start_ptr_subsamp`.



X1014_C16_11_040709

Figure 18-11: FIFO Level Calculation with Fractional Bits

The circuit in Figure 18-11 shows how the fractional bits for `write_ptr` are created. The `in_period_sync` register of the ratio calculation section contains an accurate count of the number of `mclk` periods in 1,024 input clocks (the nominal `max_count`). This number is right-shifted to obtain the number of `mclk` periods in $1/16$ of an input period. This number, `in_period_div16`, is subject to truncation errors, but it is accurate enough to use to create fractional `write_ptr` bits. The value `in_period_div16` is used as the terminal count for the `fract_period_cnt` counter. The `fract_period_cnt` thus pulses every $1/16$ input sample and is resynchronized to the updating of the write pointer by `write_en`, the write enable to the ring buffer. The `delta_in_count` counter counts each $1/16$ of an input sample time. This count saturates at 15, waiting for `write_en` to provide a reset. Thus, subsample bits are created for `write_ptr` and appended to form `write_ptr_subsamp`.

The difference between `write_ptr_subsamp` and `start_ptr_subsamp` determines `fifo_level` with 4 fractional bits. This `fifo_level` changes only when `start_ptr` is updated and is fed back to the ratio regulation section.

Clock Domain Considerations

The rising edges of the input and output clocks must be detected in several places. Because the processing clock `mclk` is asynchronous to both the input and output clocks, the control signals and data crossing these clock boundaries must be handled with care.

The pulse width of the input and output clocks must be wide enough that they are reliably sampled by `mclk`. Also, even though Virtex-5 family silicon is hardened against metastability, this does occur on occasion at clock boundaries and should be properly handled. For example, the rising edge of the input clock is detected in the `mclk` clock domain. The first register in the `mclk` domain rarely experiences metastability. Except for extremely rare cases, measured in decades per event, the output of the first register settles and meets the setup requirements of the second register. Thus, the output of the second register can be assumed to be reliable, and likewise for the third register.

Figure 18-12 shows a simple circuit that synchronizes the inputs into a new clock domain through reg_1 and reg_2, and then detects the rising edge when the input to reg_3 is High but the output is still Low. This circuit is used in the reference design as the interface to the input and output sample clocks. The timing requirements for these signals are shown in Figure 18-25 and Figure 18-26.

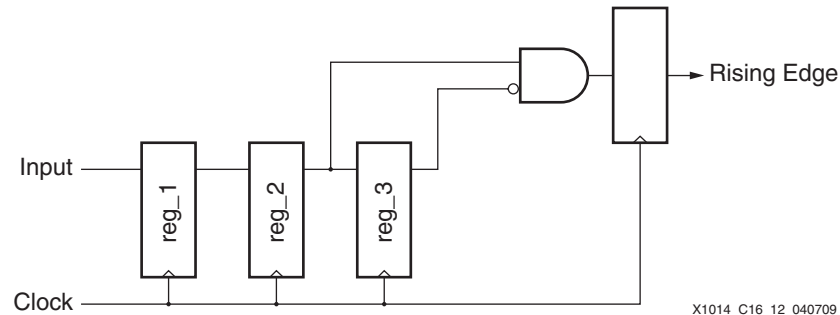


Figure 18-12: Rising Edge Detect

Resampler Functional Block

The resampler creates a set of samples from the input samples based on the output-to-input ratio produced by the ratio detection section. Two major computational tasks are required to produce each output sample:

1. Interpolate filter coefficients for the convolution based on the prototype filter
2. Perform the convolution of the interpolated coefficients with the corresponding set of input samples

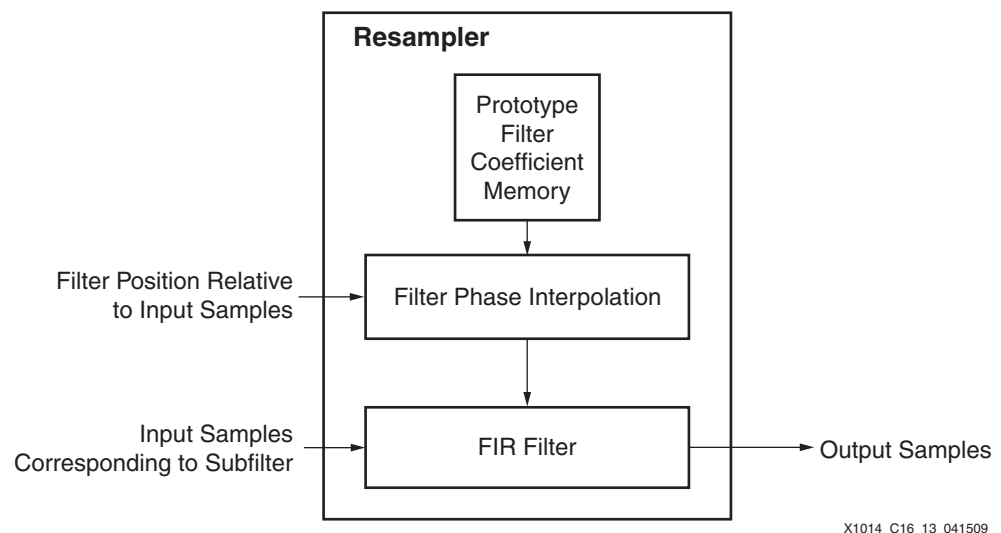


Figure 18-13: Resampler

Prototype Filter

The prototype filter was designed using the Filter Design and Analysis tool in MATLAB® software. It is a low-pass equiripple filter consisting of 64 phases of 64 taps

each. This is done with a filter order of 4096 and frequency specifications of $1/64$ of the desired response of each phase. The resulting coefficients are scaled by a factor of 64 to fully utilize the coefficient bit width and to maintain the signal amplitude. The prototype filter is symmetric, so only half of the coefficients are stored. Because the filter is of order 4096, there are actually 4097 coefficients. The center coefficient is stored in a 24-bit register and the rest are stored in block RAM-based memory of size 2048×24 .

The transition band of the filter is symmetric about the Nyquist frequency: $w_{\text{pass}} = \text{Nyquist} - 9.3\%$, $w_{\text{stop}} = \text{Nyquist} + 9.3\%$. The resulting filter has a passband of 0.4535 times the sampling rate and a stop band of 0.5465 times the sampling rate. This yields a passband of 20 KHz for a sampling rate of 44.1 KHz, for example. The parameters used for the prototype filter in the reference design are shown in Table 18-2.

Table 18-2: Prototype Filter Parameters

Parameter	Value	Comment
Response Type	Low-Pass	
Design Method	Equiripple	
Filter Order	4096	
Frequency Spec w_{pass}	$1/64 \times 2 \times 0.4535$	Normalized
Frequency Spec w_{stop}	$1/64 \times 2 \times 0.5465$	Normalized
Magnitude Spec w_{pass}	1	
Magnitude Spec w_{stop}	50000	
Density Factor	16	
Passband Ripple	± 0.016 dB	
Stopband Attenuation	149 dB	

Figure 18-14 shows the frequency response of the resulting filter.

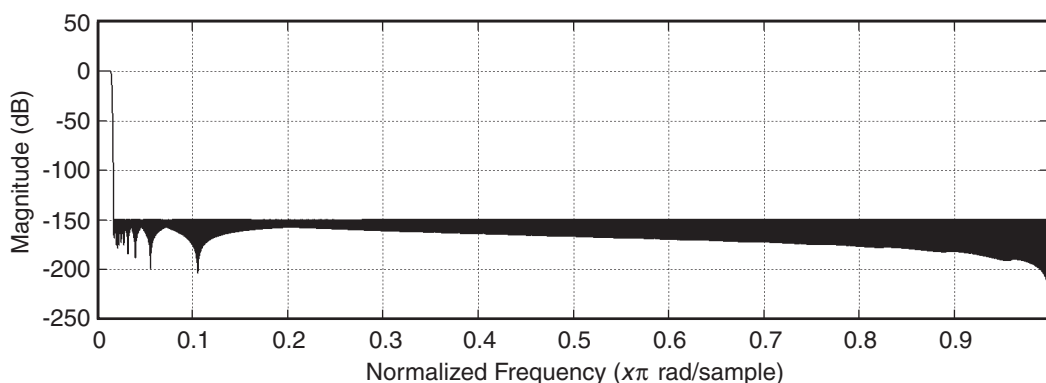
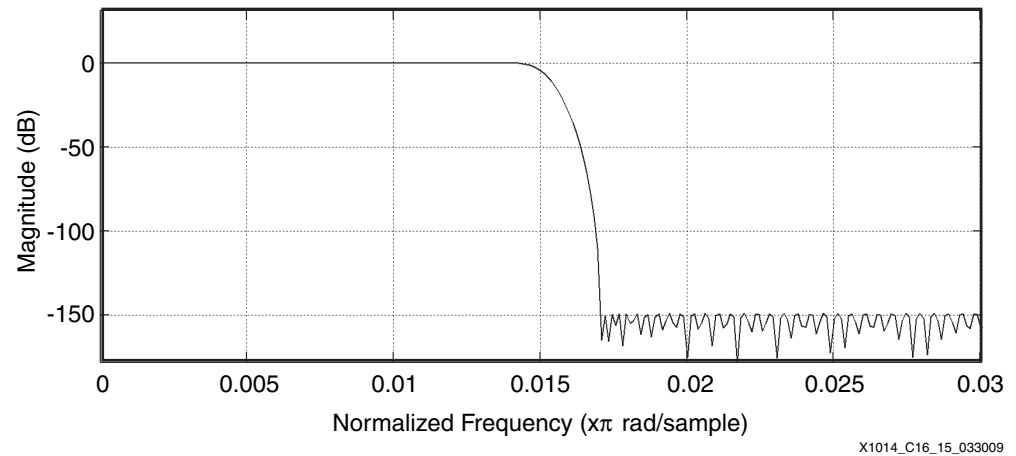


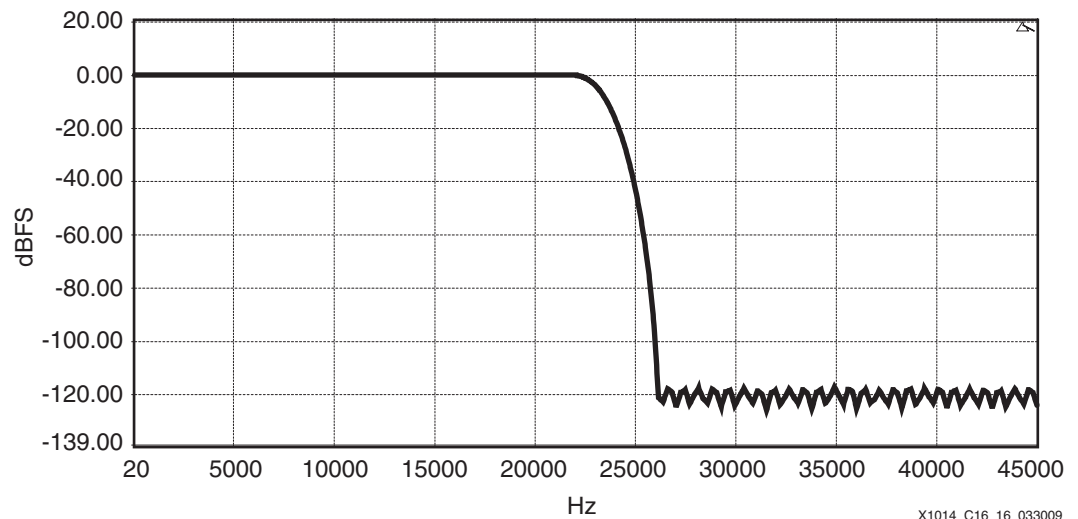
Figure 18-14: Prototype Filter Frequency Response

Figure 18-15 and Figure 18-16 show details of the transition band, and Figure 18-17 and Figure 18-18 show details of the passband. Figure 18-15 and Figure 18-17 show the calculated filter response, and Figure 18-16 and Figure 18-18 show the response based on measurements through the sample rate converter performing a 48 KHz-to-48 KHz asynchronous conversion.



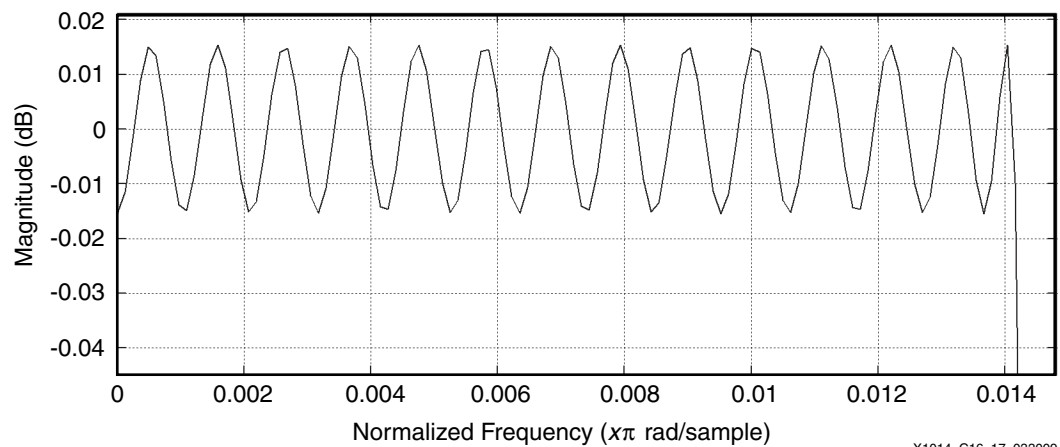
X1014_C16_15_033009

Figure 18-15: Prototype Filter Transition Band (Calculated)



X1014_C16_16_033009

Figure 18-16: Prototype Filter Transition Band (Measured)



X1014_C16_17_033009

Figure 18-17: Prototype Filter Passband (Calculated)

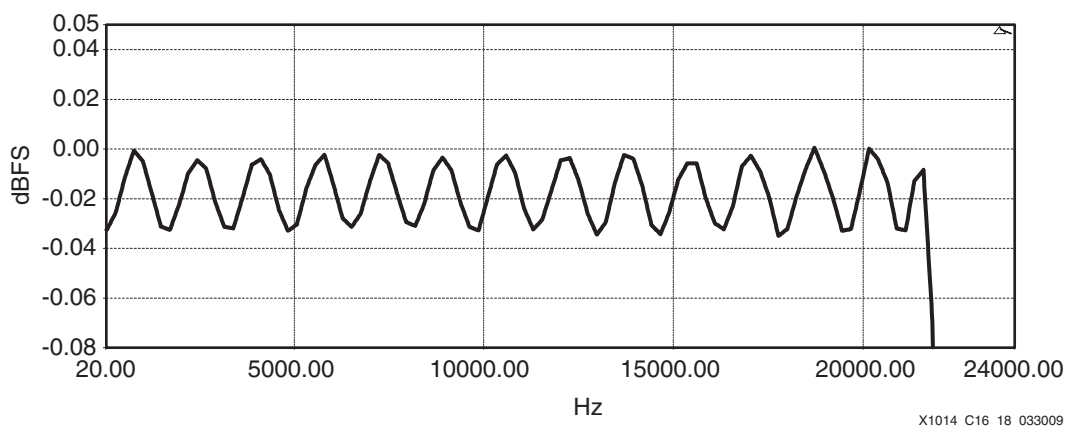


Figure 18-18: Prototype Filter Passband (Measured)

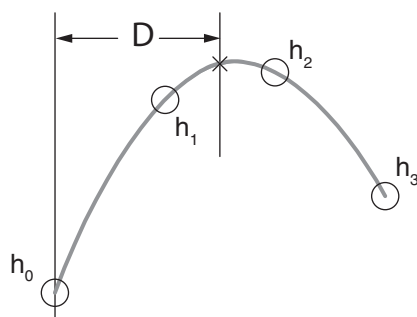
Coefficient Interpolation

Equation 1. Third-Order Lagrange Interpolation

The filter coefficients are interpolated with a third-order Lagrange interpolation according to Equation 18-1.

$$\begin{aligned}
 y = & [-(\Delta - 1)(\Delta - 2)(\Delta - 3)/6]h_0 \\
 & + [\Delta(\Delta - 2)(\Delta - 3)/2]h_1 \\
 & + [-\Delta(\Delta - 1)(\Delta - 3)/2]h_2 \\
 & + [\Delta(\Delta - 1)(\Delta - 2)/6]h_3
 \end{aligned}
 \tag{Equation 18-1}$$

where h_0 , h_1 , h_2 , and h_3 are four adjacent stored coefficients. Δ represents the difference between the location of the coefficients to be calculated (marked with an X) and h_0 , as shown in Figure 18-19.



Four Samples Required.
Follows a Cubic Polynomial
Passing through Four Points.
Continuous.

X1014_C16_19_041009

Figure 18-19: Third-Order Lagrange Interpolation

Equation 2. Third-Order Lagrange Interpolation, Factored

To minimize the number of multiplications, the equation is factored to Equation 18-2.

$$\begin{aligned}
 y = & (\Delta - 2)(\Delta - 3)/2 [-(\Delta - 1)h_0/3 + \Delta h_1] \\
 & + \Delta(\Delta - 1)/2 [-(\Delta - 3)h_2 + (\Delta - 2)(h_3)/3]
 \end{aligned}
 \tag{Equation 18-2}$$

The multiplications are performed using DSP48e blocks configured as 4-stage, 35 x 35 multipliers. Each 35 x 35 multiply makes use of 4 passes through the 18 x 18 DSP48e slice.

Input multiplexers and output registers for storage of intermediate results are used in conjunction with the multipliers to form multiply/adder units. Two multiply/adder units are used in parallel for the coefficient interpolation. Each unit operates in a 16-state sequence consisting of four 4-state multiplies.

Figure 18-20 is a block diagram of the coefficient interpolator. The input `cf_if` is the conversion factor from input samples to filter coefficients. This tells how many coefficients correspond to the distance between input samples. For up-conversion, `cf_if` is 16. For down-conversion, `cf_if` is 16 times the ratio of output rate to input rate. The input, `first_sample_f`, is the location of the first input sample to be used in the convolution relative to the stored filter coefficients. The `curr_pos_accum` register keeps track of the location of each coefficient to be accumulated relative to the stored coefficients.

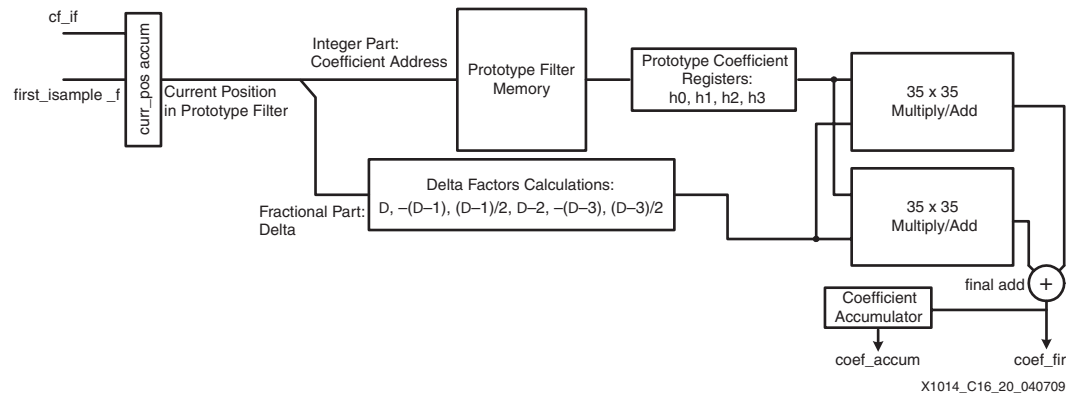


Figure 18-20: Coefficient Interpolator

When a new output sample calculation is started, `curr_pos_accum` is initialized with `first_sample_f`. As a new filter coefficient is interpolated, `curr_pos_accum` is incremented by `cf_if`. This continues until the end of the stored prototype filter is reached, indicating that all the required coefficients have been interpolated and, consequently, the convolution is complete.

The output of `curr_pos_accum` is the current position of the filter coefficient in filter space. The integer portion of this quantity is the address of the leftmost stored filter coefficient to be used in the Lagrange interpolation. The fractional portion is the delta value to be used in the interpolation.

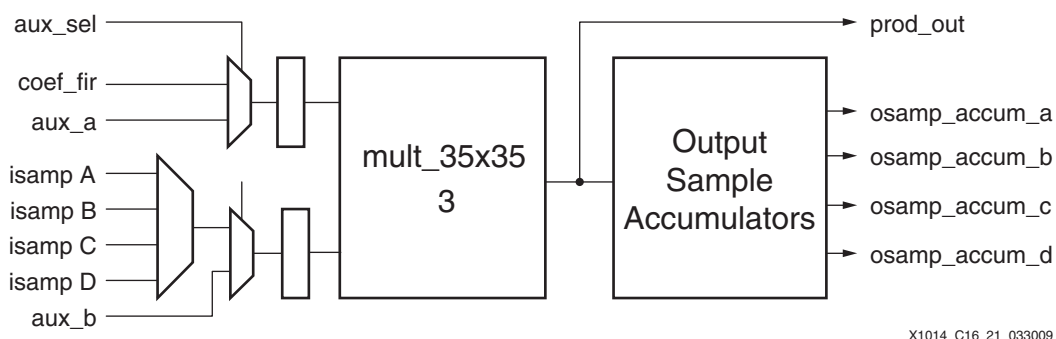
The four coefficients used in the interpolation, h_0 , h_1 , h_2 , and h_3 , are retrieved serially and stored in registers during the 16-state interpolation. Likewise, several factors are calculated from Δ and stored in registers during interpolation. The Δ -related values, along with the associated stored coefficients, are sent to the two 35 x 35 multiply/add units.

The multiply/add units operate in parallel, multiplying and summing the various terms of the Lagrange interpolation. One final addition produces the interpolated coefficient that is used in the FIR operation, `coef_fir`, as given by Equation 18-2.

As each FIR coefficient is calculated, it is accumulated in the coefficient accumulator to form `coef_accum`, which is subsequently used to normalize the result of the convolution. In the case of down-sampling, the final `coef_accum` of the convolution is virtually equal to the inverse of the scaling factor required to compensate for the increased length of the convolution. In the case of up-sampling, the final `coef_accum` is virtually equal to 1 and serves to compensate for small amplitude distortions that might otherwise occur.

FIR Filter

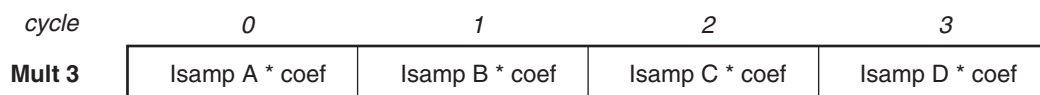
The FIR filter section (Figure 18-21) operates in parallel with coefficient interpolation. After each coefficient is interpolated, it is multiplied by the corresponding input sample and the result is accumulated. A third 35 x 35 multiplier unit performs the multiply operations.



X1014_C16_21_033009

Figure 18-21: FIR Filter Block Diagram

The FIR filter operates on the same 16-cycle sequence as the multipliers in the coefficient interpolator. Figure 18-22 shows the sequence of operations in this multiplier. Each of the labeled “cycles” consists of four clocks to perform a single 35 x 35 multiply.



X1014_C16_22_033009

Figure 18-22: Multiplier Cycles for FIR Filter

Up to four audio channels (A, B, C, and D) are accommodated. However, the reference design utilizes only the A and B channels. There is also an auxiliary set of inputs and outputs so this unit can perform other multiplications besides those for the FIR filter. These auxiliary multiplications are used by the control section for such functions as converting locations from input sample space to filter coefficient space.

There are separate accumulators for each channel. At the end of the convolution, each accumulator holds the result of the convolution. These results are normalized by dividing the accumulated output sample values, osamp_accum by the coef_sum.

Shared Divider

A module called shared_divider_v1_0 is a signed 27 x 27 multi-cycle pipelined divider with 25 fractional output bits. It has a latency of 56 states, and a new quotient is produced every 8 states. This divider is used in two ways:

- Produce the ratio and 1/ratio values
- Normalize the output sample accumulation values by dividing osamp_accum by coef_accum

The accuracy of these calculations directly affects the quality of the sample rate conversion. Thus, a high degree of precision is required. Because the divide operations are done very infrequently compared to other operations, the divider is optimized for minimum area and, thus, low throughput.

Control

The high-level control is contained in the `timing_control` module. The output sample clock starts the sample calculation sequence. Each time an output sample is taken, as indicated by `output_clk`, a new sample is calculated. Whereas ratio detection operates in a more or less continuous fashion, the resampler operates in a burst fashion. It interpolates a filter phase and performs the convolution each time an output sample is taken, and then idles until the next sample is taken and a new one can be calculated. The idle time can be very short (for example, when the output rate is very high), or it can be the majority of the time (for example, when the output rate is low and the ratio is near 1:1). In any case, the computation starts at the rising edge of the output sample clock and terminates when the end of the prototype filter is reached.

The timing control state machine controls the creation of an output sample. Figure 18-23 is a basic diagram of the state machine. The state register holds the current state. There is a count associated with each state that determines the minimum time each state lasts. State changes occur only when this delay is met, as determined by when the state delay count times out, indicated by `state_dly_tc`. The status inputs and the current state determine if a state change occurs.

The main purposes of the state machine are to control access to shared multiply and divide resources, sequence the data, and load the results into registers at the proper time. The current state, control bits, and state counter all contribute to the control and timing of these calculations.

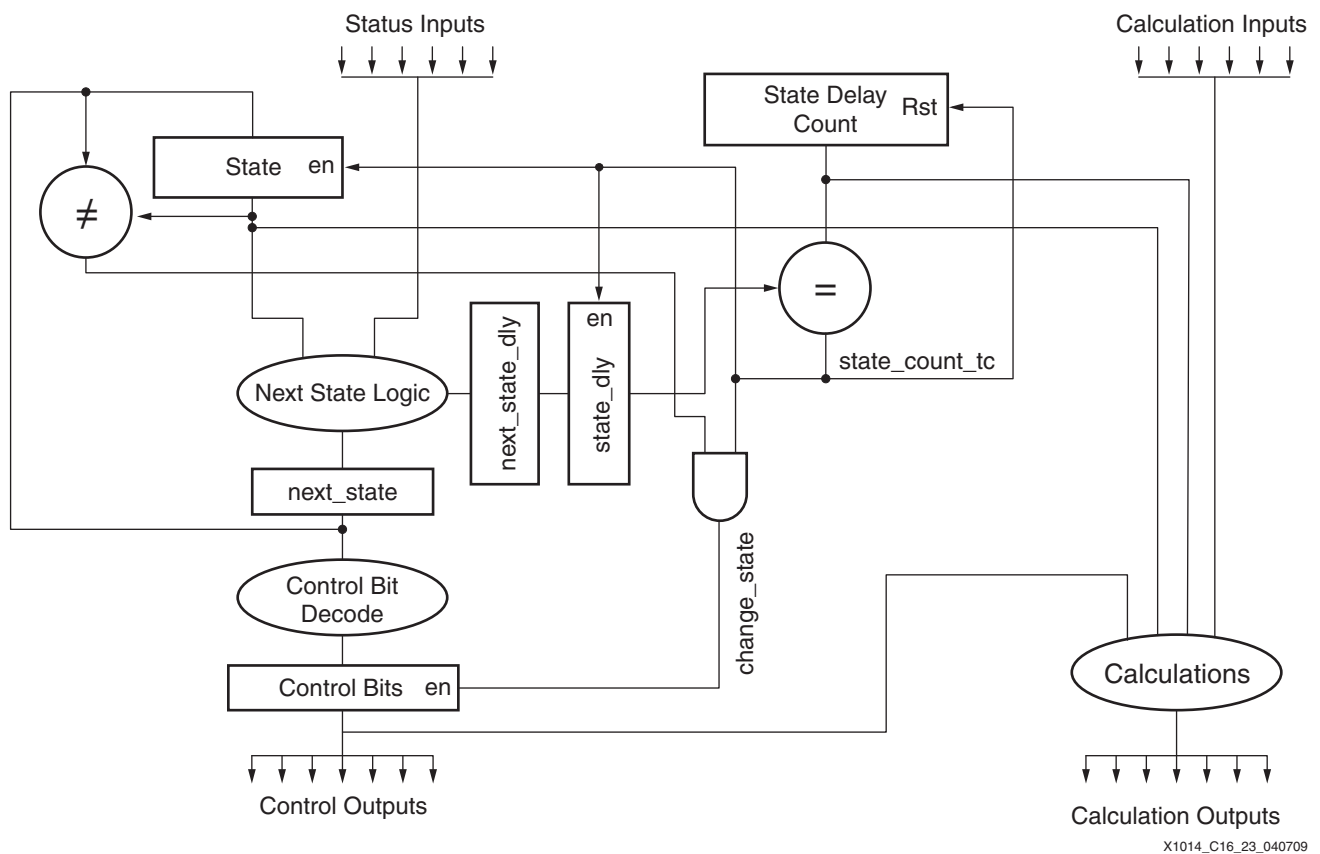


Figure 18-23: Timing Control

Figure 18-24 shows the state diagram of the top-level control state machine in the `timing_control` module. The initial state is **RATIO**. In this state, the shared divider is used to calculate the ratio from the most recent `in_period_sync` and `out_period_sync` values. The state machine remains in this state until the `get_new_sample` signal asserts in response to the rising edge of `clkout`.

In the **POSITIONS** state, a new position for the output sample relative to the input samples is calculated based on the regulated ratio, `ratio_reg`. The start position in the prototype filter is also calculated. The auxiliary functionality of multiplier unit 3 is used in some of these calculations. The **INIT_FIR** state allows the initial positions to propagate.

The bulk of the calculations happen during the **FIR** state. In this state, the `go_fir` signal pulses to indicate the start of a new FIR filter sequence. The resampler is reset and enabled to interpolate and apply filter coefficients to the input samples. The result is a single output sample. The state machine remains in this state until the `osamp_done` signal is asserted by the resampler indicating that prototype filter has been traversed and the FIR filter operation is completed.

The **SCALE** state uses `shared_divider_v1_0` to normalize the accumulated results of the FIR filter by dividing the accumulated results of the FIR by the accumulated coefficients, `coef_sum`. Each audio channel is normalized and clamped independently to produce the final output sample value.

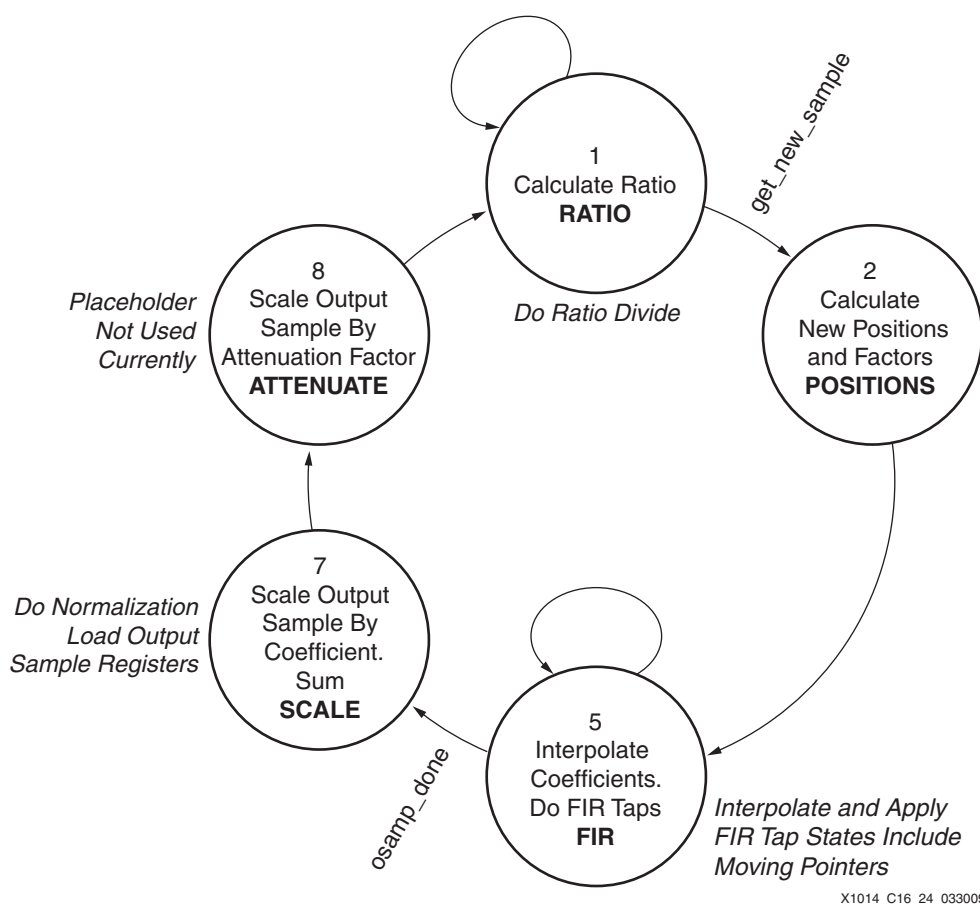


Figure 18-24: Top-Level Control State Machine

The ATTENUATE state is not used in the reference design, but is included as a state in which attenuation of the output can be performed for fade out or fade in, or overall magnitude control. The auxiliary multiplier functionality can be used to accomplish this.

Interface Timing

All data processing in the asynchronous sample rate converter is done in the mclk (high-speed processing clock) domain that, in general, has no relationship to either the input or output clock. The input and output sample clocks are treated as enables that specify when input data is valid, and, on the output, when output data is taken. The period of the sample clocks is used to determine the conversion ratio.

Figure 18-25 shows the timing requirements for input samples. The clk_{in} signal is resampled in the mclk domain with the circuit of Figure 18-12, and the rising edge is used to determine when data is valid. Therefore, setup and hold times are specified in terms of mclk periods. Relative to the rising edge of clk_{in}, there is a setup requirement of 0 and a hold requirement of 5 mclk periods. The clk_{in} signal must be High for a minimum of 5 mclk periods and Low for a minimum of 5 mclk periods for accurate edge detection.

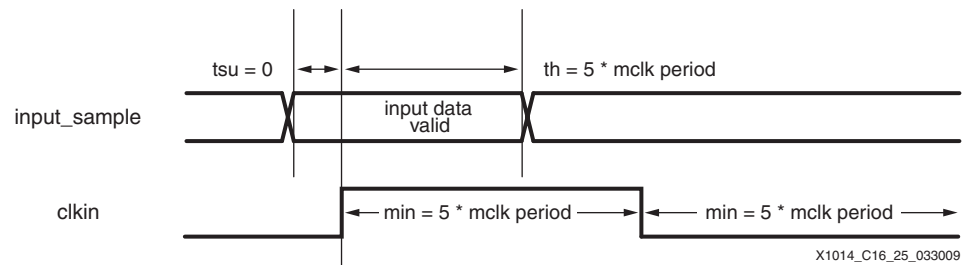


Figure 18-25: Input Timing

Figure 18-26 shows the timing characteristics of output samples. Because they are created in the mclk domain, the timing specifications are also given in terms of mclk periods. Relative to the rising edge of clk_{out}, output sample data is valid a minimum of 3 mclk periods before and remains valid until the next sample is presented. A conservative minimum time of 100 mclk periods is given for the output data to remain valid after the rising edge of clk_{out}. Like clk_{in}, clk_{out} (an input to the sample rate converter) is resampled in the mclk domain to determine the output sampling rate. Therefore, clk_{in} has a minimum High time requirement of 5 mclk periods and a minimum Low time of 5 mclk periods.

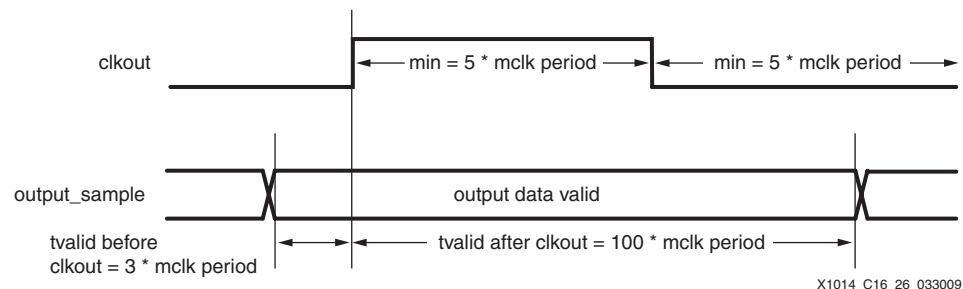


Figure 18-26: Output Timing

Performance

THD+N

The typical overall performance of the ASRC reference design is –130 dB THD+N. The performance varies somewhat according to the input and output sample rates and the frequency content of the signal. The range is from –125 dB to –139 dB.

Table 18-3 lists performance measurements taken over a variety of common ratios. Measurements were taken with a Prism Sound dScopeIII audio analyzer in the default THD+N measurement mode scanned from 20 Hz to 20 KHz. These are examples of possible conversions only. The ASRC reference design allows for virtually infinite combinations of input and output frequencies within the maximum frequency and ratio constraints.

Table 18-3: THD+N Performance vs. Conversion Frequency

Input Sample Frequency (KHz)		Output Sample Frequency (KHz)				
		32	44.1	48	88.2	96
32	Minimum	–125	–125	–125	–125	–128
	Maximum	–131	–130	–137	–133	–134
	Typical	–128	–127	–132	–130	–131
	1 KHz tone	–128	–128	–130	–131	–131
44.1	Minimum	–129	–133	–125	–126	–127
	Maximum	–131	–137	–130	–137	–134
	Typical	–129	–136	–128	–136	–130
	1 KHz tone	–130	–135	–126	–135	–128
48	Minimum	–127	–126	–133	–128	–127
	Maximum	–132	–131	–139	–132	–134
	Typical	–129	–128	–136	–131	–131
	1 KHz tone	–129	–127	–136	–128	–128
88.2	Minimum	–127	–134	–128	–131	–128
	Maximum	–132	–137	–134	–137	–134
	Typical	–129	–136	–126	–137	–131
	1 KHz tone	–130	–136	–125	–135	–130
96	Minimum	–130	–129	–125	–128	–129
	Maximum	–135	–133	–138	–134	–134
	Typical	–133	–131	–134	–132	–131
	1 KHz tone	–132	–130	–129	–129	–130

Maximum Conversion Ratios

Up-conversion: 8:1
Down-conversion: 1:7.5

The range of the up-conversion is limited by the number of integer bits in the ratio calculation. The down-conversion ratio is limited by the amount of input sample storage memory and how much of this memory is allocated to the input FIFO.

Sample Frequency Range

Input: 8 KHz to 192 KHz
Output: 8 KHz to 192 KHz

The stated sample frequencies are for a 250 MHz mclk. A slower or faster mclk reduces or increases both minimum and maximum sample frequency in approximate proportion to the change in mclk. The upper limit is a factor of processing clock frequency, and the lower limit is a function of the width of the period counters and the processing clock frequency.

Latency

For any given conversion ratio, the latency is fixed. It is determined by the phase delay of the FIR filter and the fill level of the input FIFO. The FIFO level is fixed at 16, but the size of the FIR filter and consequently its phase delay vary in the case of down-conversion. Therefore, the formula for latency is different depending on whether the sample rate converted is performing up-conversion or down-conversion.

Equation 3: Up-Conversion Latency

For up-conversion, the filter length is 64. Therefore, the phase delay is 32. [Equation 18-3](#) then gives the up-conversion latency.

$$\begin{aligned}\text{Latency} &= \text{phase delay} + \text{FIFO delay} \\ &= 32 + 16 \\ &= 48 \text{ input sample periods}\end{aligned}\quad \text{Equation 18-3}$$

The time in milliseconds depends on the input sample frequency.

Equation 4: Down-Conversion Latency

For down-conversion, the filter is spread across more samples, so the phase delay and subsequently the latency are longer in terms of the number of input samples. [Equation 18-4](#) then gives the down-conversion latency.

$$\begin{aligned}\text{Latency} &= \text{phase delay} + \text{FIFO delay} \\ &= 32 \cdot (f_{s_{out}}/f_{s_{in}}) + 16\end{aligned}\quad \text{Equation 18-4}$$

Examples:

48 KHz : 48 KHz conversion:
Latency = 32 + 16
= 48 input sample periods
= 1 ms

48 KHz : 96 KHz up-conversion:
Latency = 32 + 16
= 48 input sample periods
= 1 ms

32 KHz : 48 KHz up-conversion:
Latency = 32 + 16

$$= 48 \text{ input sample periods}$$

$$= 1.5 \text{ ms}$$

96 KHz : 48 KHz down-conversion:

$$\text{Latency} = 32 \bullet 2 + 16$$

$$= 80 \text{ input sample periods}$$

$$= 0.83 \text{ ms}$$

48 KHz : 44.1 KHz down-conversion:

$$\text{Latency} = 32 \bullet 48/44.1 + 16$$

$$= 50.83 \text{ input samples}$$

$$= 1.06 \text{ ms}$$

In cases of changing frequency, the latency changes smoothly, as specified in [Equation 18-3](#) and [Equation 18-4](#). For up-conversion, changes in input sample frequency result in changes in latency. Changes in output sample frequency do not. For down-conversion, changes in input or output sample frequency result in changes in latency.

FPGA Resource Utilization and Performance

The reference design has been implemented and hardware verified in Virtex-5 devices with the results shown in [Table 18-4](#) and [Table 18-5](#).

Table 18-4: Reference Design Resource Utilization

LUTs	Registers	DSP48e Slices	Block RAMs
2,528	3,235	3	5

Table 18-5: Reference Design Frequency Performance

Speed Grade	Processing Clock Frequency (mclk) (MHz)	General Maximum Sample Frequency (KHz)
-3	300	224
-2	255	192
-1	225	170

To calculate the minimum required processing clock frequencies for specific conversions, [Equation 18-5](#) and [Equation 18-6](#) should be used.

$$\text{Up-conversion: } F_{\text{mclk}} = F_{\text{sout}} \bullet 1325 \quad \text{Equation 18-5}$$

$$\text{Down-conversion: } F_{\text{mclk}} = F_{\text{sin}} \bullet 1030 + F_{\text{sout}} \bullet 270 \quad \text{Equation 18-6}$$

For example, a down-conversion from 192 KHz to 48 KHz requires a clock frequency of $192 \text{ KHz} \bullet 1030 + 48 \text{ KHz} \bullet 270 = 210.72 \text{ MHz}$, so a -2 device works for this particular conversion.

Additional Channels

The bulk of the FPGA resources in the reference design are required for the functions that distinguish an asynchronous sample rate converter from a regular sample rate converter: tracking the ratio and interpolating the coefficient set. This functionality is common to all channels, and it need not be replicated for additional channels that share the input and output clocks. The architecture is such that ratio detection and filter interpolation can be easily shared with additional channels. Therefore, the incremental resource usage for adding channels is small. An additional benefit from sharing these resources is that the

outputs are inherently phase matched. [Chapter 19, Multi-Channel Asynchronous Sample Rate Converter](#) is specifically designed for doing multi-channel sample rate conversion with minimal additional resources.

Data Flow Spreadsheet

The reference design includes a data flow spreadsheet, `asrc_dataflow.xls`, that graphically illustrates the timing of data as it flows and gets processed from register to register through the pipeline. The registers are listed across the top of the spreadsheet and each row represents one clock cycle, progressing from top to bottom. There are three sheets corresponding to three stages of processing: filter phase interpolation, FIR filter, and output sample normalization.

The data flow spreadsheet is an aid to understanding the intent of the RTL, as well as a powerful tool for debugging, and making and documenting modifications. Multi-cycle paths are apparent from inspection of the data flow spreadsheet. Comments are used on some of the cells to give cycle-specific explanations. There are enough passes to show the entire start-up sequence and the process end sequence for an output sample calculation. Many more passes are required to perform the filter interpolation and FIR for an output sample.

Filter Phase Interpolation and FIR Filter

The filter interpolation and FIR filter blocks operate on a 16-state cycle matching the 16-state cycle of the multipliers (4 multiplications of 4 states each). The left columns contain the cycle and sub-cycle counts and a column labeled pass. The pass field is for correlation of the start-up sequence between the filter phase interpolation page and the FIR filter page.

The cells in the interior of the spreadsheet indicate the data present in a particular register at a particular time. The registers are arranged so that the flow of data is generally upper left to lower right. In most cases, there are only entries for new, valid data. Blank cells indicate, depending on the context, either invalid data or data that remains the same as previous entries or that is not relevant at that time. To the right of the register list is a list of the state machine output bits. These illustrate the timing relationship of state machine bits to datapath registers. One cycle of the state machine is boxed. The boxed area contains a complete state machine cycle and corresponds directly to the RTL state machine definition. The state machine also contains unused control bits. These are to facilitate the addition of control outputs to the state machine and are removed by synthesis.

Output Sample Normalization

The output sample normalization page shows data flow in the `timing_control` module, particularly the last stage of output sample processing done during the SCALE state of the timing control state machine (see [Figure 18-24](#)). This sheet details the data sequencing through the divider for normalization, and the timing for the steps of rounding, clamping, and loading the output registers. Register names are shown in the top of each column. Data for each audio channel (a, b, c, and d) passes through these stages in sequence. Although time slots for four audio channels are shown, only two (a and b) are used in the reference design. This page of the spreadsheet reflects the latency (56) and throughput (one result per 8 states) of the divider. Because this is a shared divider, the timing also applies to the other operations it performs.

Reference Design

The reference design for the asynchronous sample rate converter for Virtex-5 FPGAs is available at <https://secure.xilinx.com/webreg/clickthrough.do?cid=119758>. Open the ZIP archive and extract the file `xapp1014_c18_asrc.zip`.

The maximum processing clock speed varies from family to family. This has the effect of changing the maximum sample rates that the device families can accommodate. For easy reference, [Table 18-6](#) shows approximate clock frequencies required for some common conversions. These values are derived from [Equation 18-3](#) and [Equation 18-4](#).

Table 18-6: Approximate Processing Clock Frequencies (MHz) Required for Various Conversions

Input Sample Frequency (KHz)	Output Sample Frequency (KHz)						
	32	44.1	48	88.2	96	144	192
32	45	60	65	120	130	195	255
44.1	55	60	65	120	130	195	255
48	60	60	65	120	130	195	255
88.2	100	105	105	120	130	195	255
96	110	115	115	125	130	195	255
144	160	165	165	125	175	195	255
192	210	215	215	225	225	240	255

Conclusion

The asynchronous sample rate converter reference design is useful for converting between source and destination audio streams with independent clocking sources. It has excellent THD+N performance and converts over a wide range of ratios and frequencies, automatically detecting the conversion settings and tracking changes in frequency. The reference design is implemented in Virtex-5 FPGAs using dual-port block RAM and DSP48e slices, in addition to general logic resources. The implementation uses interpolated-coefficient FIR filtering. It precisely interpolates millions of filter phases from a small set of stored phases and applies the interpolated filter to the corresponding input samples to obtain an output sample at a particular instant. The latency is low and deterministic due to the use of a single FIR filter stage. The source code and data flow spreadsheet, along with the functional description and filter parameters, are provided to facilitate integration of the design “as-is” or adaptation of the design for specific application requirements.

Multi-Channel Asynchronous Sample Rate Converter

Summary

The asynchronous sample rate converter (ASRC) reference design converts stereo audio from one sample frequency to another. The input and output sample frequencies can be an arbitrary fraction of one another or the same frequency, but based on different clocks. The reference design handles one or more stereo pairs that are synchronous with each other at the input. The output is a band-limited version of the input resampled to match the output sample timing. The reference design has these features:

- Fully asynchronous.
- Typical THD+N: -130 dB (Range: -125 dB to -139 dB).
- Input and output audio word width of 24 bits.
- Choice of automatic ratio detection or manual ratio control. Automatic ratio detection includes rate change tracking (varispeed).
- Up-conversion, down-conversion, and 1:1 asynchronous conversion support.
- Efficient multi-channel expansion expandable to eight stereo pairs for only 60% additional resources compared to one stereo pair.
- Sample clock jitter rejection. Retains full performance over AES3 jitter tolerance curve [\[Ref 16\]](#).
- Input rates ranging from 8 KHz to 192 KHz, continuous.
- Output rates ranging from 8 KHz to 192 KHz, continuous.
- Conversion ratio ranging from 1:7.5 (Down) to 8:1 (Up), continuous.
- Low deterministic latency.
- Lock status outputs provided for external muting.

The reference design is implemented in the Virtex®-5 FPGA architecture. It uses a DSP48E slice as its main math element, and block RAM for input sample buffers and storage of the prototype filter.

Structure

As shown in [Figure 19-1](#), the asynchronous sample rate converter reference design consists of two main functional units: the ratio control and the resampler. These main functions are further divided into smaller functional units. The ratio control function has two main subfunctions: ratio detection and input sample storage. The resampler also has two main subfunctions: interpolation of the correct phase of the filter, and the FIR filter operation

that applies the calculated filter coefficients to the set of input samples to form an output sample. The HDL code implementing the reference design breaks these functions down into modules with functional boundaries. This chapter explains what the modules are, how they fit together, and provides details about the functional blocks.

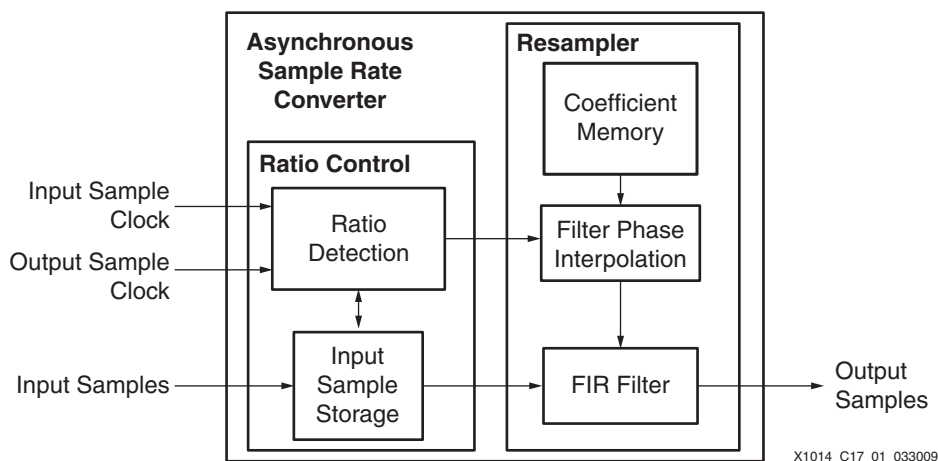


Figure 19-1: ASRC Top-Level Block Diagram (Two-Channel Configuration)

Multi-Channel Configurations

The two-channel configuration shown in Figure 19-1 is the basic configuration of the ASRC. However, the architecture of the reference design allows additional channels to be added in a straightforward way.

The bulk of the FPGA resources in the reference design are used by the functions that make the design an asynchronous converter as opposed to a regular sample rate converter, such as tracking the sample rate ratio and interpolating the coefficient set. This functionality is common to all channels, and does not need to be replicated for additional channels that share the same input and output clocks. Ratio detection and filter interpolation can be easily be shared with additional channels. Therefore, the incremental resource usage for adding channels is small. An additional benefit from sharing these resources is that the outputs are inherently phase matched.

Four-Channel Configuration

The four-channel reference design is shown in Figure 19-2. Because the FIR filter in the design already has time slots for four channels, only the input buffer needs to be replicated. This reduces the per-channel fabric resource cost by over 40%. The reference design files include this configuration. The four-channel resource utilization is given in [FPGA Resource Utilization and Performance](#), page 447.

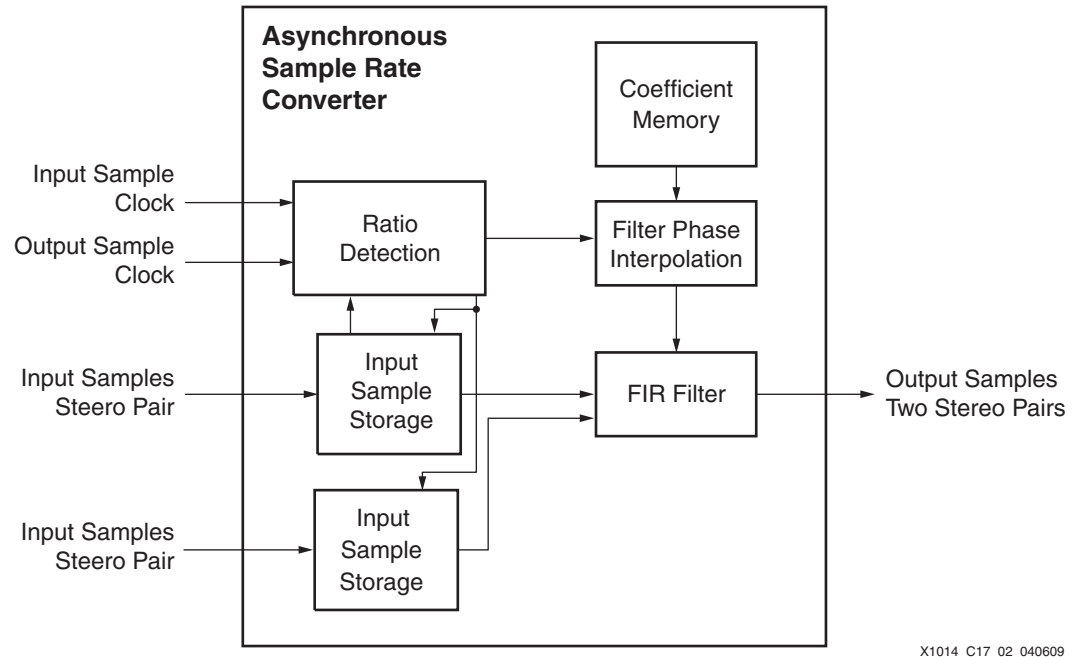
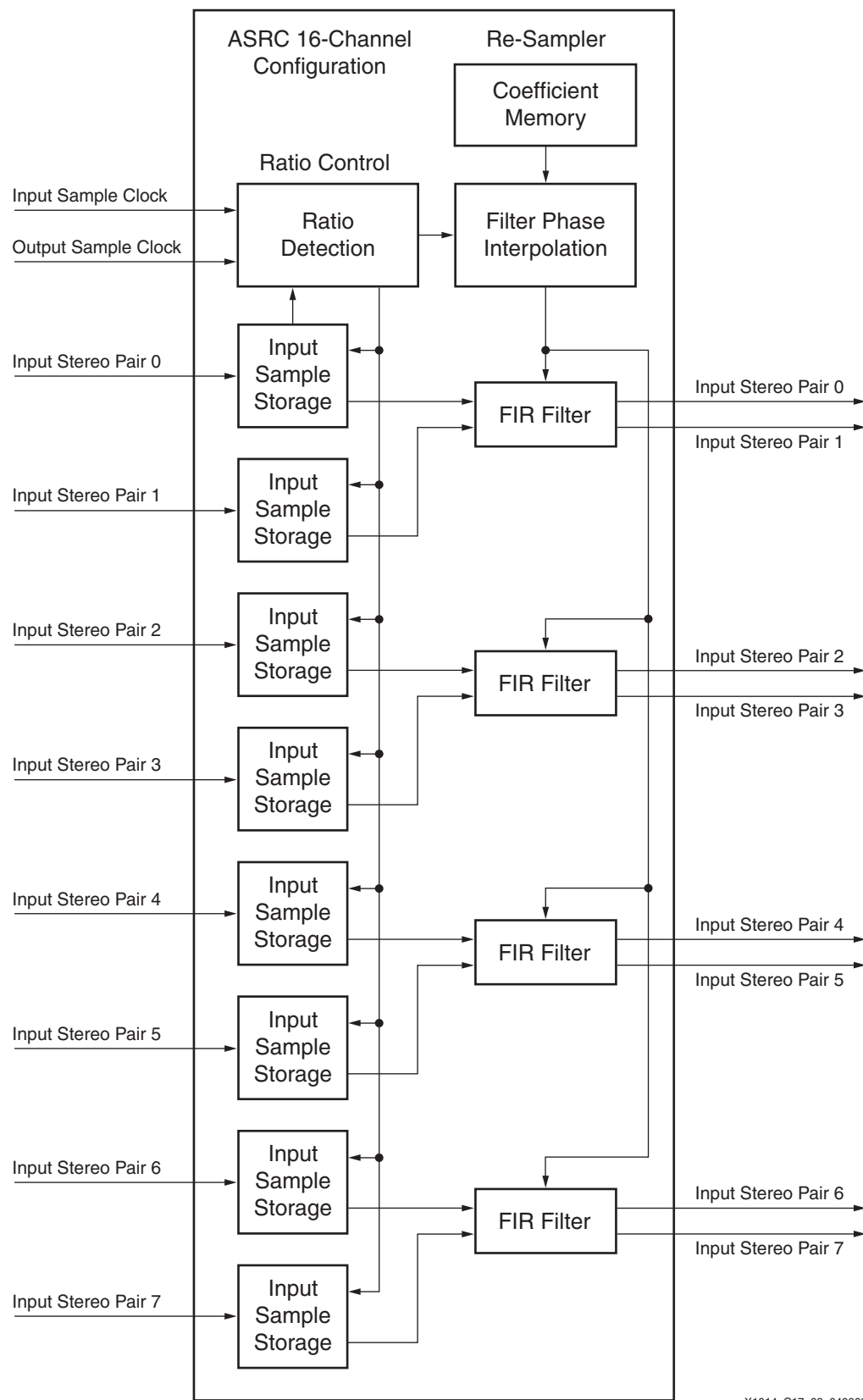


Figure 19-2: **Four-Channel Configuration**

16-Channel Configuration

For a 16-channel (8-pair) configuration, the per-pair cost goes down even more dramatically. Six additional instances of the input sample storage element are required, but only three additional instance of the FIR filter are needed. The reference design files include this configuration. The resource utilization is given in [FPGA Resource Utilization and Performance, page 447](#).



X1014_C17_03_040609

Figure 19-3: 16-Channel Configuration

Inputs and Outputs

The asynchronous sample rate converter has the inputs and outputs shown in [Table 19-1](#).

Table 19-1: ASRC Inputs and Outputs

Name	I/O	Width (Bits)	Description
mclk	In	1	This is a high-frequency processing clock.
clkin	In	1	This is a sample rate pulse for input samples.
clkout	In	1	This is a sample rate pulse for output samples.
reset	In	1	This is an asynchronous reset.
manual_ratio_en	In	1	This is a manual ratio control enable: 1 = Manual mode (use manual_ratio) 0 = Use automatic ratio tracking based on the frequency of clkin and clkout.
manual_ratio	In	26	This is the manual ratio for manual ratio mode (F_{in}/F_{out}). Its format is unsigned 4.22.
input_sample_a input_sample_1a	In	24	These are the input samples for stereo pair 1, channel a. These samples are valid at clkin.
input_sample_b input_sample_1b	In	24	These are the input samples for stereo pair 1, channel b. These samples are valid at clkin.
input_sample_na,b	In	24	These are additional input channels. They are valid at clkin.
output_sample_1a	Out	24	These are output samples for stereo pair 1, channel a. These samples are valid at clkout.
output_sample_1b	Out	24	These are output samples for stereo pair 1, channel b. These samples are valid at clkout.
output_sample_na,b	Out	24	These are additional output channels. They are valid at clkout.
fifo_level_out	Out	9	This is the input FIFO fill level. It is useful for manual ratio management.
calc_ratio_out	Out	26	This is the calculated ratio F_{clkout}/F_{clkin} . Its format is unsigned 4.22.
locked	Out	1	This active-High output indicates that ratio tracking is in a locked state, i.e., the input FIFO level is within the prescribed thresholds and stable.
fifo_overflow	Out	1	This active-High output indicates that the level of the input FIFO is beyond the safe operating range. Therefore, loss of input samples is likely.

Modules

Figure 19-4 shows the hierarchy of the modules in the ASRC reference design and the relation of modules to functional blocks. Table 19-2 lists the modules and gives descriptions of them.

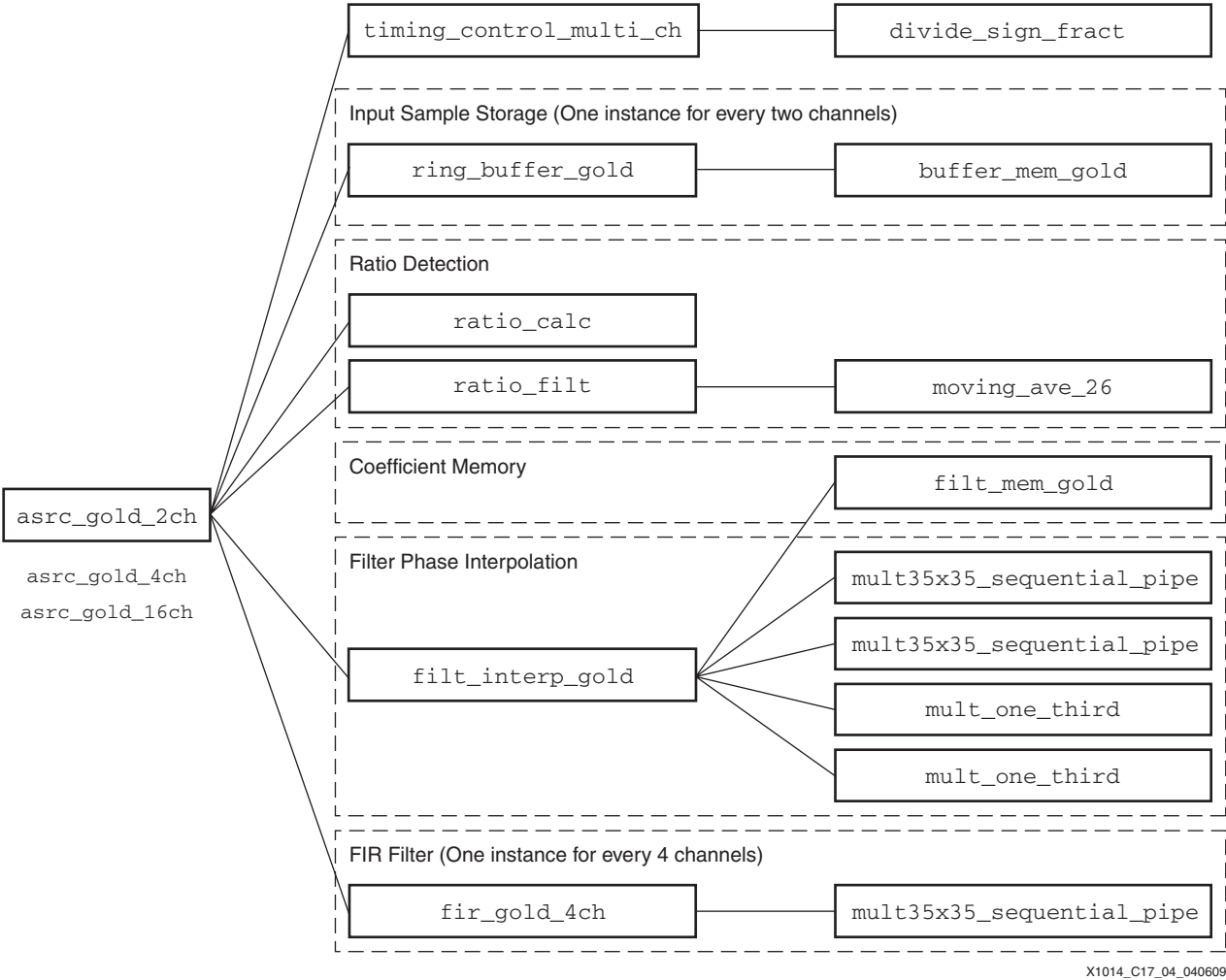


Figure 19-4: Reference Design Module Hierarchy and Relation to Functional Blocks

Table 19-2: Reference Design Module Descriptions

Module Name	Description
asrc_gold_2ch asrc_gold_4ch asrc_gold_16ch	This is the top-level wrapper that instantiates and connects the lower level modules and provides the I/O interface. There are three versions of the wrapper. Each targets a different number of audio channels (2, 4, and 16, respectively.)
timing_control_multi_ch	This module contains the master state machine that controls the creation of each output. It instantiates the divider that is used for ratio calculation and normalizing the output samples.

Table 19-2: Reference Design Module Descriptions (Cont'd)

Module Name	Description
divide_sign_fract	This is a 27 x 27 bit signed serial divider. The quotient has 27 integer and 26 fractional bits. This divider is used for: <ul style="list-style-type: none"> Calculating the ratio of output sample rate to input sample rate Normalizing output samples based on the sum of input coefficients
ring_buffer_gold	This module provides pointers and control for the ring buffer memory. The ring buffer stores incoming samples and provides the sample stream to the fir_gold_4ch module. One instance is required for each pair of channels.
buffer_mem_gold	This is a 512 x 48 dual-port RAM for the ring buffer.
ratio_calc	This module contains the counters for determining input and output sample rates. These rates are sent to the signed fractional divider and the calculated ratio is returned. The ratio_calc module also determines the feedback error term based on the FIFO level and regulates the ratio accordingly.
ratio_filt	This module instantiates the moving_ave_26 module, determines when a new ratio has been calculated, and when the filter should be bypassed.
moving_ave_26	This module performs a 16-tap moving-average filter on the calculated ratio.
filt_interp_gold	This module performs the Lagrange interpolation on the prototype filter and interpolates a filter coefficient for every input sample in the FIR filter operation.
filt_mem_gold	This is a 2048 x 24 single-port ROM containing the prototype filter. The prototype filter is 4,097 coefficients that are symmetrical. The middle coefficient is stored separately.
mult35X35_sequential_pipe	This is a 35 x 35 multiplier using four sequential states in a DSP48e slice.
mult_one_third	This is a fixed multiplier implementing a divide by three on a 24-bit number.
fir_gold_4ch	This module performs a 64-tap FIR filter for each output sample up to four channels. One instance is required for each set of four channels. Data and coefficients come from the filt_interp_gold module.

Functional Description

This section discusses the functional blocks found in the reference design, including ratio control, input sample storage, resampling, and control.

Ratio Control Functional Block

Ratio control can be manual or automatic. In some applications, it is desirable to adjust the ratio according to an algorithm using factors other than just the audio input and output rates. For example, this might be done when controlling the audio to video latency within a frame synchronizer. This also allows the reference design to be used as a fixed-rate converter. For manual ratio control, the ratio is controlled externally and input directly into the ASRC. The values used for automatic ratio adjustment (the calculated ratio and the FIFO fill level from the input FIFO) are provided as outputs for status monitoring and to facilitate external control.

For automatic ratio control, a sophisticated feedback control system is used to track rate changes under varispeed conditions. This control system must also provide stable and high-quality conversion under steady-state conditions.

Automatic ratio control uses one of two algorithms depending on whether the input rate is changing. At startup, and whenever the input or output rate changes, rate-change tracking (varispeed) mode is used to quickly adjust to the correct ratio and to adjust the level of the input FIFO to the proper level. In this mode, the ratio correction term grows exponentially with error to quickly track large rate changes and reduce the error to low levels, as shown in Figure 19-5.

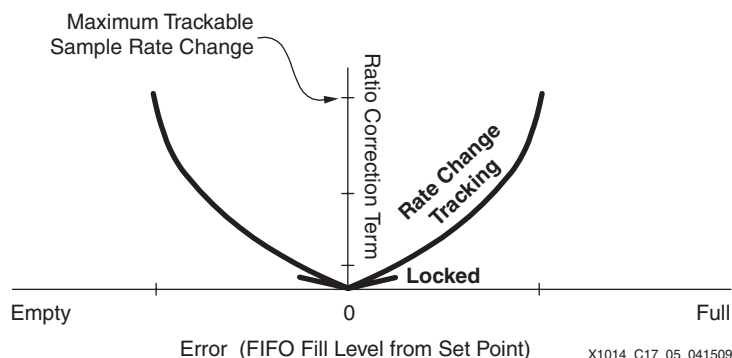


Figure 19-5: Error Correction Curves

After a small error has been achieved and the ratio is stable, an automatic switch is made to locked mode. This mode limits the amount and rate of change of the ratio to achieve maximum audio quality. The locked mode can track small drifts in the clock frequencies. However, if a large rate change occurs, the error term exceeds the locked mode range, and the mode automatically shifts to rate-change tracking. When a small error range is achieved and the ratio is stable, the switch to locked mode again occurs. In this manner, changes in the sample frequencies, large and small, are continuously and smoothly tracked.

The input samples are buffered by the `ring_buffer_gold` module in the Ratio Control function block. When a new output sample is required, the set of input samples required for the FIR filter convolution are sent to the resampler.

Ratio Detection

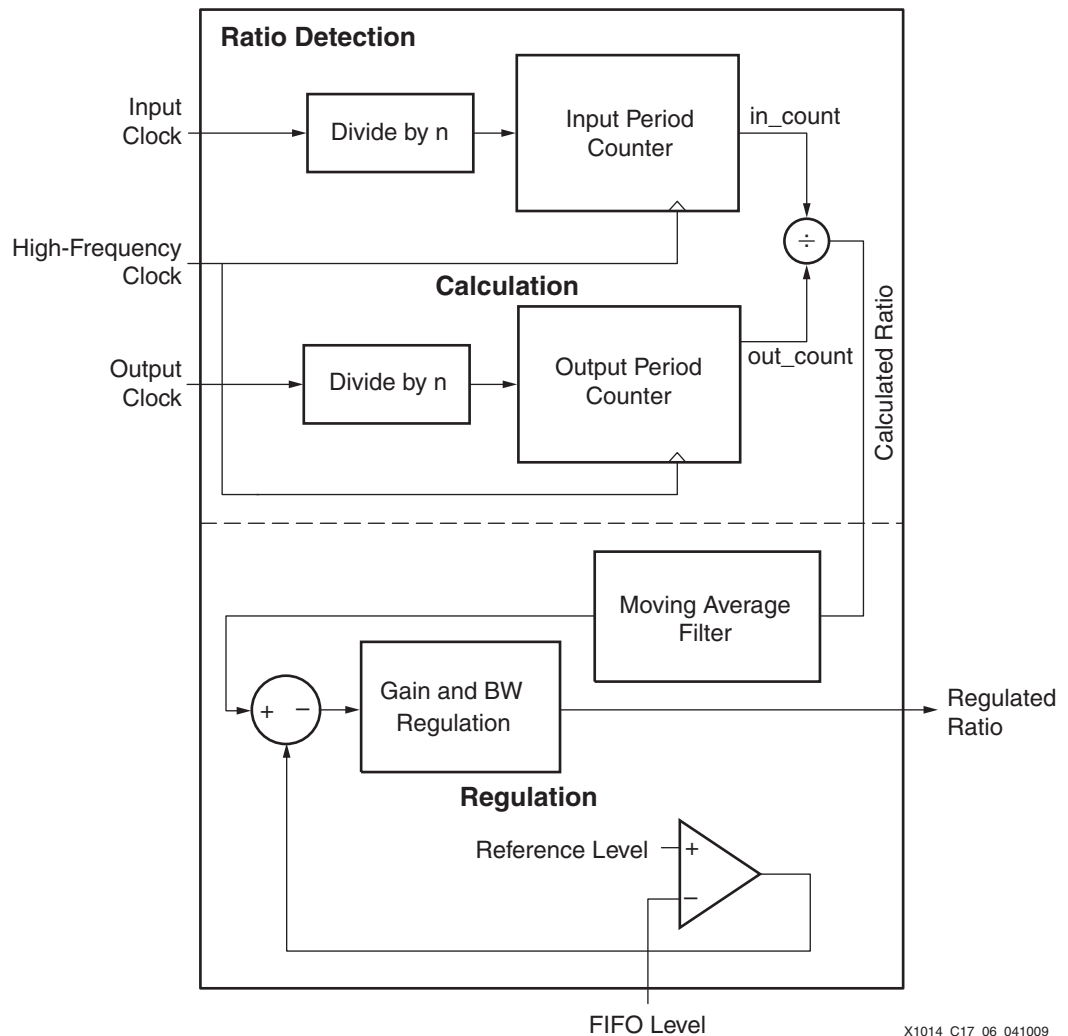


Figure 19-6: Ratio Detection Block Diagram

The ratio detection block is implemented in the `ratio_calc` and `ratio_filt` modules of the reference design. A ratio is computed by measuring the period of the input and output clock with a high frequency clock that, in general, is not related to either the input or output clocks. This is illustrated in the top section of Figure 19-6. To improve the accuracy of this calculation and mitigate the effects of jitter, the input and output clocks are measured over 1,024 cycles. The input period is divided by the output period to obtain a calculated ratio.

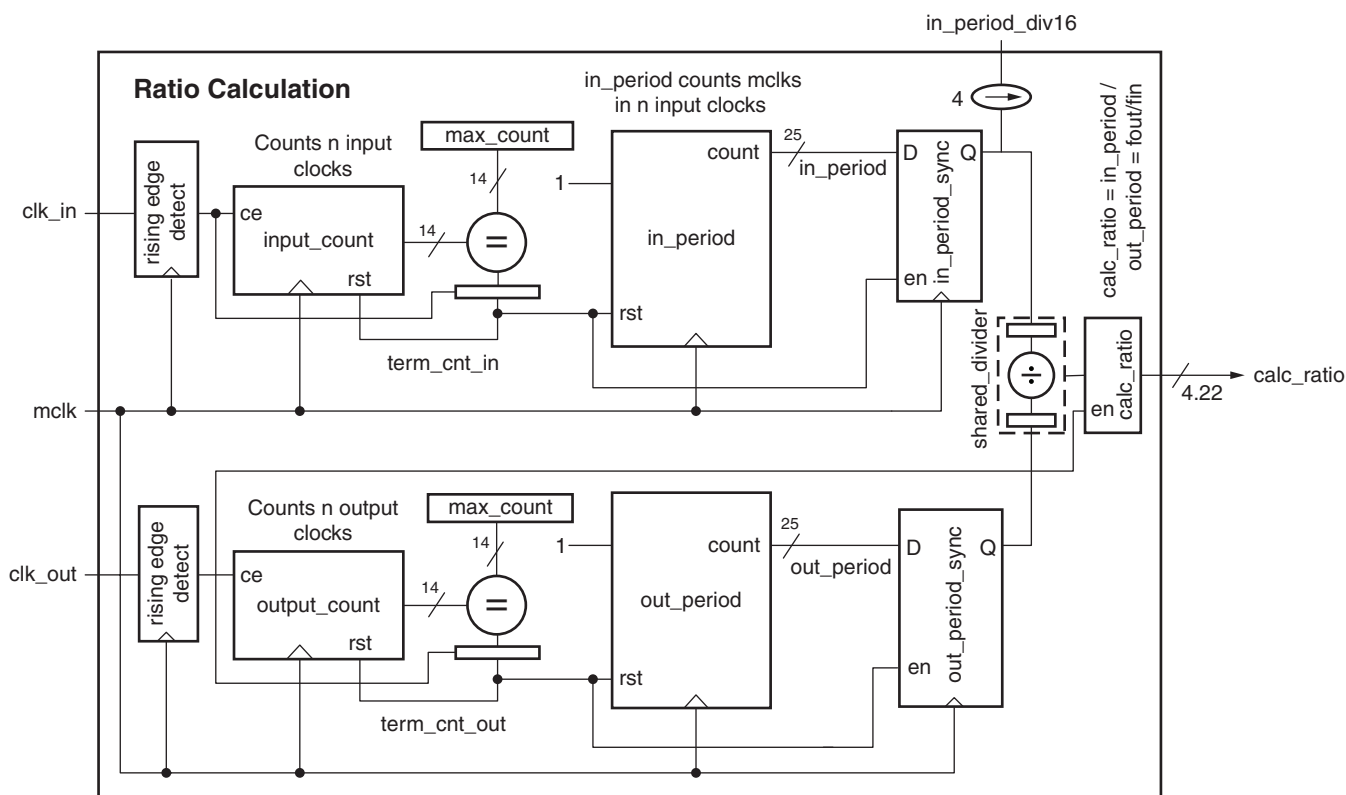
To further attenuate sample clock jitter, the calculated ratio passes through a moving-average filter contained in the `ratio_filt` module. The moving-average filter is only applied during locked mode, when the input frequency is stable. In frequency tracking mode, the moving-average filter is bypassed, and the most current calculated ratio is used for ratio regulation.

To regulate the level of the input FIFO, and thus latency, the FIFO fill level is compared to a reference level in the ratio regulation section. The difference is used as an error signal to adjust the ratio. Because the ratio determines the position of each new output sample

relative to the input samples, it effectively controls the speed at which input samples are processed.

Ratio Calculation

Figure 19-7 illustrates how the input period measurement is made. The input_count block counts each rising edge as an input clock cycle. Max_count specifies how many input clocks to count before resetting the counter. Max_count is a parameter in the reference design and is nominally set to 1024. The term_cnt_in signal pulses once every max_count + 1 input clocks. This signal resets the in_period counter as well as input_count. The in_period counter counts the number of master processing clock (mclk) cycles that occur during max_count + 1 input clocks. At every pulse of term_cnt_in, the in_period_sync register stores the latest in_period count, and the counting begins again. The in_period count resets to one so that the resulting count is the actual number of mclk cycles over the specified period, not the number of cycles minus one. The in_period_sync value is shifted right by four and sent to the Sample Storage section as in_period_div16.



X1014_C17_07_040609

Figure 19-7: Ratio Calculation Detailed Block Diagram

The timing diagram in Figure 19-8 illustrates the operation of the ratio calculation section for the clk_in period measurement. At each rising edge of clk_in, the rising edge detect circuit outputs a pulse, which is one mclk period wide. The input_count retains a count of these pulses. In the example shown, the max_count is four. Therefore, the input_count counts from zero to four. During this time, the in_period counter retains a count of the number of mclks. At the end of five periods of clk_in, term_cnt_in pulses, and the count from in_period_cnt is loaded to the in_period_sync register. This value represents the number of mclks per five periods of clk_in. In the reference design, max_count is 1023.

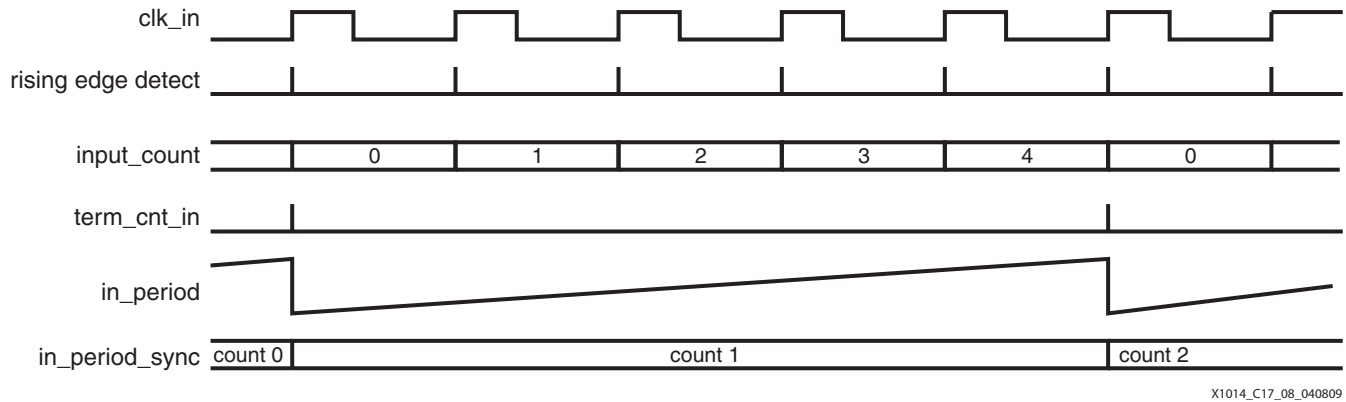


Figure 19-8: Timing Diagram of Input Period Measurement with max_count = 4

The output clock period is measured in the same way as the input period. The ratio is calculated based on the timing of the output clock. To obtain the calculated ratio (calc_ratio), in_period_sync is divided by out_period_sync. This division is done by divide_sign_fract, a multi-cycle pipelined divider in the timing_control_multi_ch module. The calc_ratio signal is then sent to the ratio regulation section. The calc_ratio signal allows for a range of 0 to 15 with 22 fractional bits. The calculated ratio is used internally for automatic ratio tracking, and is also output from the ASRC to facilitate external control if manual ratio mode is selected.

Ratio Filtering for Jitter Tolerance

The AES receiver must recover audio data correctly in the presence of jitter. Jitter in the audio data timing is propagated to the sample rate converter (SRC). Therefore, the SRC should have jitter tolerance equivalent to that of the AES receiver. The AES3 standard [Ref 10] specifies the jitter tolerance for AES receivers. The jitter tolerance curve is shown in Figure 19-9.

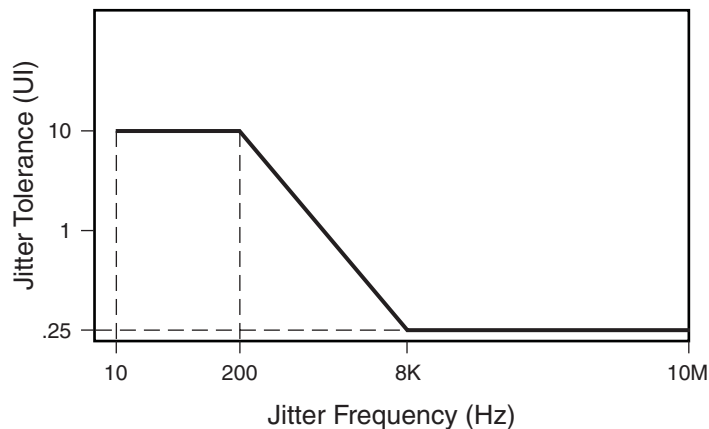
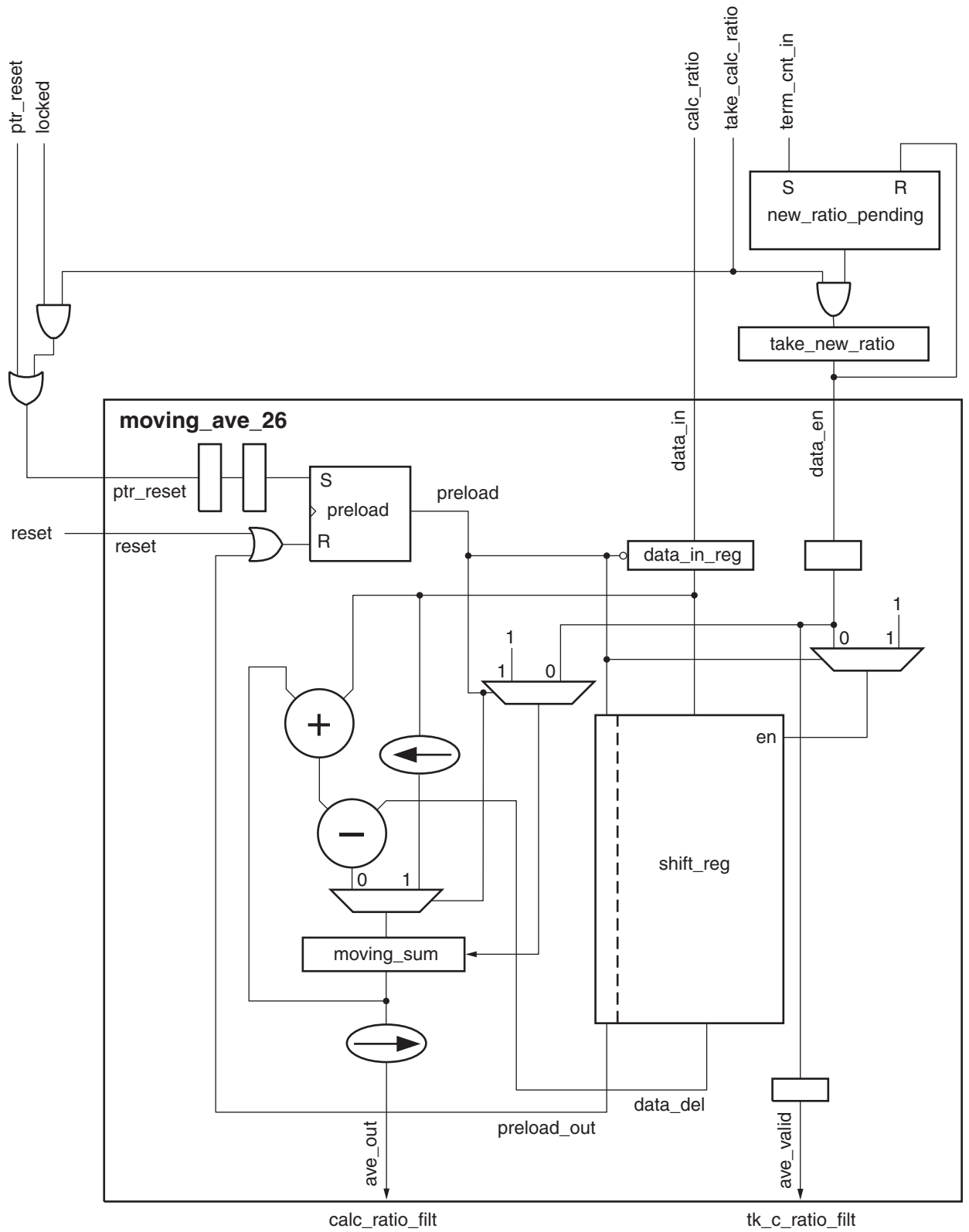


Figure 19-9: Jitter Tolerance Curve

In the reference design, the calculated ratio between sample rates is filtered for added jitter tolerance. During locked mode operation, the calculated ratio is filtered using a moving-average FIR filter. This prevents short term variations in sampling frequency from causing harmonic distortion in the output sample stream. In other words, it attenuates the

effects of input sample clock jitter. The result is that jitter causes no increase in distortion. Full performance is retained over the entire range of the AES3 jitter tolerance curve.

Figure 19-10 is a block diagram of the ratio filter. It is a recursive implementation requiring only one add and one subtract operation. As each new data point enters the filter, it is added into the average and the oldest data point is subtracted. A shift register is used for the storage element. A shift register with 16 locations is implemented very efficiently in SRL16 elements, requiring only one LUT per input data bit. The calculated ratio has four integer and 22 fractional bits. An additional bit of storage is used as a data valid to track data through the shift register. This bit is required for the preload function, which simultaneously bypasses the filter operation and preloads every location in the shift register with the current value of the input data.



X1014_C17_10_040709

Figure 19-10: Ratio Filter

When the input signal `ptr_reset` goes High, the preload flag is set High, indicating that the block is in preload mode. In preload mode, the `data_in_reg` register is held, and the current data in this register is propagated directly to the `moving_sum` register and output on the `ave_out` signal. At the same time, the shift register enable is forced active, and the shift register begins shifting in the data in the `data_in_reg` register. The preload bit also shifts through the shift register in parallel with the data. When the shift register has shifted the input value through every location, the preload bit is the output from the shift register as the `preload_out` flag. This indicates that the preload cycle is complete. The contents of the shift register and the output register all equal the current input value stored in the `data_in_reg` register. This forces a reset of the preload register, which returns the module to normal filtering mode.

This preload functionality is used to bypass the moving-average filter when the ratio section is not locked (i.e., in frequency tracking mode). This allows for better tracking and faster locking. When locked mode is entered, each new ratio calculated is averaged with the values preloaded in the shift register. The `term_cnt_in` input signal indicates that a new input clock count has completed. The `take_calc_ratio` signal indicates that a new output clock count has completed. It also indicates that a divide operation has been applied to obtain `calc_ratio`. The combination of the `term_cnt_in` and `take_calc_ratio` signals is used to form the `take_new_ratio` signal, which indicates that a new data value can be input into the moving-average filter.

Ratio Regulation

The ratio regulation section adjusts the calculated ratio to regulate the input FIFO level. [Figure 19-11](#) shows how this is done. The input specifying the target FIFO level, `fifo_setpoint`, is subtracted from the input specifying the current FIFO level, `fifo_level_input`. The difference, `error_term`, is used to add an offset to `calc_ratio`. The `error_term` signal is conditioned separately for locked mode and rate change mode. This conditioning is determined by parameters set in the HDL, which control the gain in the error term, the threshold applied to the error term, and restrictions in the ratio slew rate, if any. These parameters establish the trade-off between tight sample-rate tracking and harmonic distortion performance. Tighter rate-change tracking causes more frequent rate adjustments, which creates more harmonic distortion in the audio output.

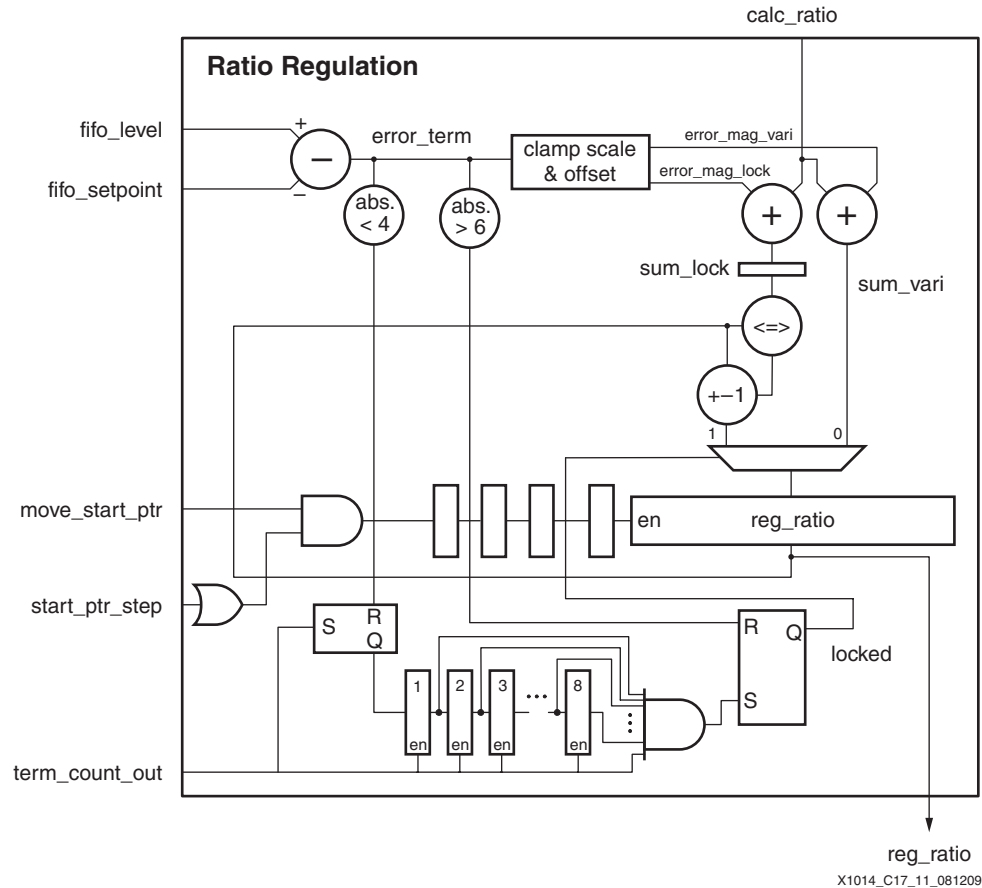


Figure 19-11: Detailed Block Diagram of Ratio Regulation Section

To reduce the distortion component of the rate change, a threshold can be placed on the error term. If the error is below this threshold, no adjustment to the rate is made. The cost is that rate-change tracking is slower and less accurate. For locked mode, the default settings in the reference design specify a low threshold ($\frac{1}{4}$ of the input sample time), no additional gain, and allow one LSB step of rate change per output sample. In frequency-rate tracking mode, there is no threshold and no additional gain. This balance allows good rate-change slew and quick, reliable recovery from loss of input. At the same time, it provides good jitter rejection.

The amount and rate of the offset from the calculated ratio depends on whether or not locked mode is engaged. To enter locked mode, the error_term must be less than four input samples for five consecutive term_count_out times. Unlocked mode, also called rate-change tracking mode, is entered any time the error is more than six samples.

In rate-change tracking mode, the error term is added directly to calc_ratio. It also has an exponential gain such that the correction factor is multiplied at higher error ratios. This facilitates faster locking at startup and after frequency changes, without dropping or repeating samples.

In locked mode, the error term is used to increment or decrement the ratio at a maximum rate of one LSB per output sample. The current reg_ratio is fed back and compared with the target ratio, sum_lock. If the target is different from reg_ratio, one is either added to or subtracted from the current ratio. The reg_ratio value is updated when the set of input samples used for the convolution changes, indicated by a pulse on move_start_ptr with a

non-zero value of `start_ptr_step`. This limits the slew rate of the ratio to 0.24 ppm per output sample for optimal audio performance. This mode can track slow frequency variations because `calc_ratio` is updated periodically, and the FIFO level is updated every output sample with subsample accuracy. This is discussed in the description of the ratio control functional block. This high degree of accuracy of `fifo_level` also enables the ratio detection circuit to maintain a deterministic latency when the clocks are stable. In other words, for given input and output sample rates, the latency varies by only a fraction of a sample time, and the latency for any two instances of the SRC is the same to within a fraction of a sample time.

Lock Status Indicators

There are two top-level output signals that indicate the status of the ratio control section: `locked` and `fifo_overflow`. These two signals can be used to mute the audio when the sample rate converter is outside its bounds of normal operation, when the input sample rate is changing, or both. When the `locked` signal is High, it means that ratio control is in locked mode, with minimal FIFO level error and maximum audio quality. When `locked` is Low, rate-change mode is active, meaning more aggressive rate change tracking and correspondingly lowered THD+N performance. The `fifo_overflow` signal indicates that the input sample FIFO has overflowed or underflowed, and therefore the output audio is corrupted. This could occur at the application of the input audio stream, because the input audio stream has been removed, or during extremely sharp sample rate changes.

Audio quality is severely compromised when `fifo_overflow` is asserted. Depending on the application, rate-change tracking mode audio might or might not be acceptable. These two status bits are provided so that muting can be performed externally when instability in the sample rates could cause unacceptable distortions.

Input Sample Storage

The input samples are stored in a ring buffer as shown in Figure 19-12, implemented in block RAM. Two 24-bit data words, one for each channel of the channel pair, can be stored per memory location. The ring buffer is 512 bits wide x 48 locations deep, enough to accommodate spreading the prototype filter by a factor of 7.5 with enough locations remaining to function as an input FIFO. Two pointers, the write pointer and the start pointer, move through the addresses in the buffer in a circular fashion, designating the locations into which input samples are to be written and out of which input samples are to be read for the filtering operation.

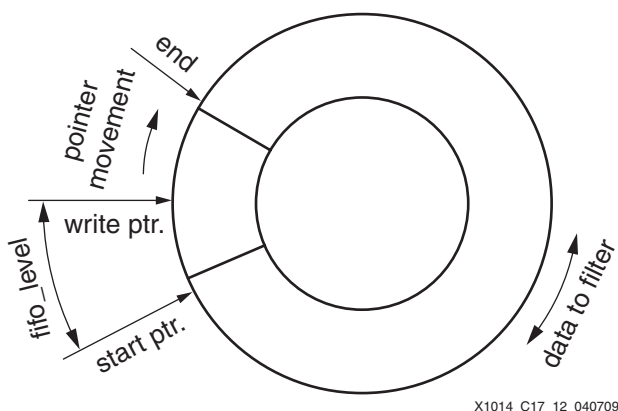
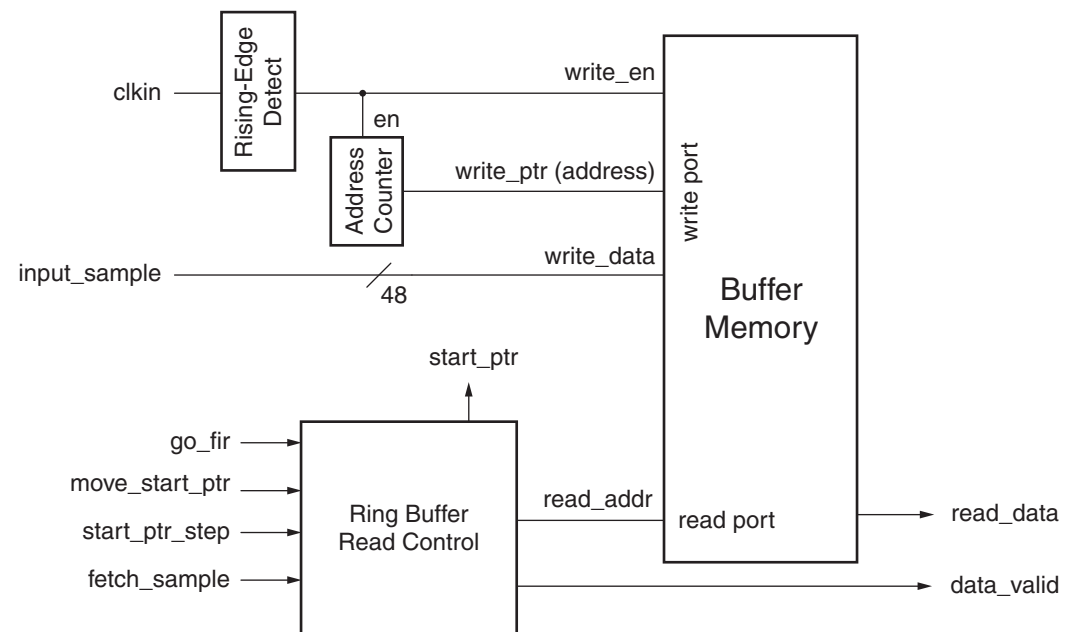


Figure 19-12: Ring Buffer

The first sample to be used in the FIR filter is indicated by `start_ptr`. Input samples are stored as they are received at the location indicated by `write_ptr`. This pointer is incremented each time a new sample is received. For each output sample, a set of input samples is sent to the FIR filter, starting with the newest (at `start_ptr`) to the oldest (at end). The `start_ptr` is updated each time a new output sample is created.

The locations between `write_ptr` and `start_ptr` serve as an input FIFO. The difference between the two pointers is used as the `fifo_level` value. This is used as a feedback mechanism for the ratio. The net effect is to change slightly the rate at which input samples are used to keep the FIFO at a predetermined level. The level is set by a parameter, and is nominally 16 in the reference design. The `fifo_level` value is also output from the ASRC to facilitate external control if manual ratio mode is selected.

Figure 19-13 is a block diagram of the input storage section. The ring buffer consists of the buffer memory (using dual-port block memory) and control logic for reading and writing. For the write port, a write-enable pulse `write_en` is produced on the rising edge of `clkin`, the input sample clock. This pulse is used to write sample data into memory and increment the address counter to the next address.



X1014_C17_13_041009

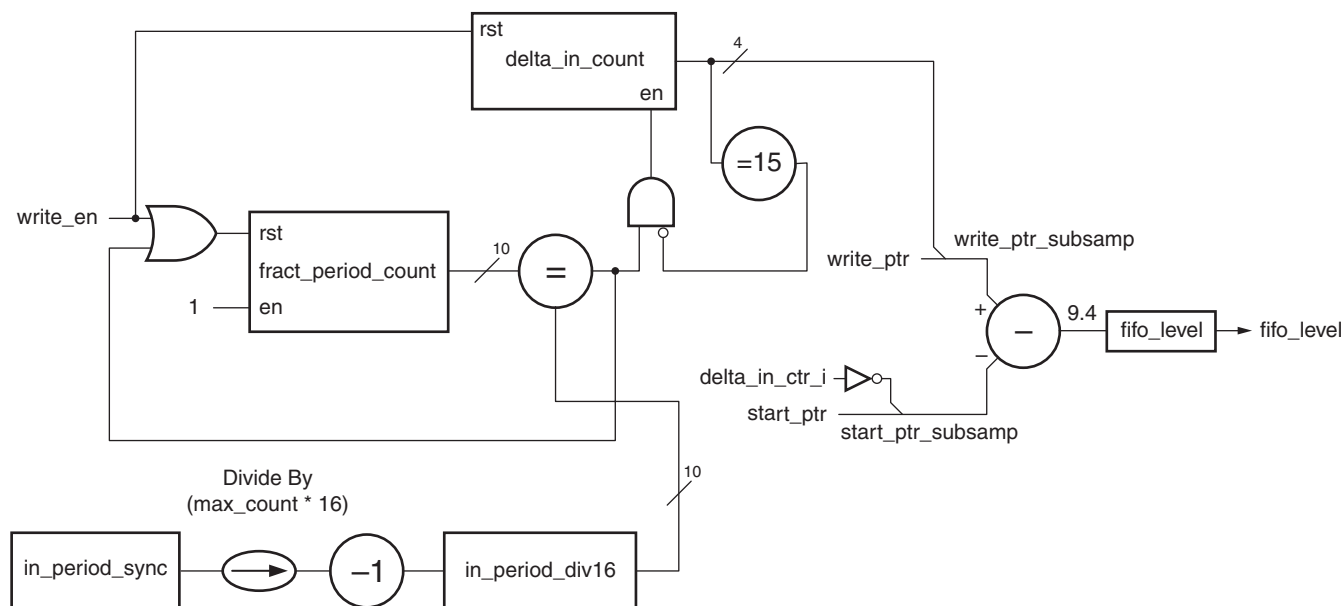
Figure 19-13: Input Buffer Storage Block Diagram

For the read port, `go_fir` signals the start of a new FIR operation, which produces an output sample. This resets `read_addr` to the `start_ptr` value. The `fetch_sample` signal pulses once for each input sample used in the convolution. Each time it pulses, the `read_addr` is decremented. This sends sample data to the FIR filter in the newest-to-oldest order shown in Figure 19-12. The read control also generates an integer, `start_ptr`. The movement of this pointer is controlled by `move_start_ptr` and `start_ptr_step`. When `move_start_ptr` pulses, `start_ptr` is increased by `start_ptr_step`. The outputs of this section are input samples on the `read_data` bus and a `data_valid` flag.

Because `write_ptr` and `start_ptr` are updating at different times and possibly in different increments, the difference between them can vary widely, even if the ratio is correct and the input and output rates are perfectly stable. For example, if the ASRC is performing a down-conversion by a factor of four, the write pointer increments four times during the

time the `start_ptr` moves just once. Thus, `fifo_level` varies by three over the period of a single output cycle. To reduce such fluctuations, `fifo_level` is updated only after `start_ptr` is updated.

The `fifo_level` is calculated to an accuracy of $1/16$ of a sample by creating subsample accurate write and start pointers. As shown in Figure 19-14, the fractional bits for the `start_ptr` signal are the inverse of `delta_in_ctr_i`. The `delta_in_ctr_i` signal indicates the position of the current output sample with respect to input samples. Therefore, it represents a fractional start position. These bits are appended to `start_ptr` to form the `start_ptr_subsamp` signal.



X1014_C17_14_040709

Figure 19-14: FIFO Level Calculation with Fractional Bits

The circuit in Figure 19-14 shows how the fractional bits for the `write_ptr` signal are created. The `in_period_sync` register of the ratio calculation section contains an accurate count of the number of `mclk` periods in 1,024 input clocks (the nominal `max_count`). This number is right-shifted to obtain the number of `mclk` periods in $1/16$ of an input period. This number, `in_period_div16`, is subject to truncation errors, but it is accurate enough to create usable fractional `write_ptr` bits. The value `in_period_div16` is used as the terminal count for the `fract_period_cnt` counter. The `fract_period_cnt` thus pulses every $1/16$ of an input sample. It is re-synchronized to the updating of the write pointer by `write_en`, the write enable to the ring buffer. The `delta_in_count` counter counts each $1/16$ of an input sample time. This count saturates at 15 and waits for `write_en` to provide a reset. Therefore, subsample bits are created for the `write_ptr` and appended to form the `write_ptr_subsamp` signal.

The difference between the `write_ptr_subsamp` and `start_ptr_subsamp` signals determines `fifo_level` with four fractional bits. This `fifo_level` changes only when the `start_ptr` is updated, and is fed back to the ratio regulation section.

Clock Domain Considerations

The rising edges of the input and output clocks must be detected in several places. Because the processing clock mclk is asynchronous to both the input and output clocks, the control signals and data crossing these clock boundaries must be handled with care.

The pulse width of the input and output clocks must be wide enough that they are reliably sampled by mclk. Additionally, even though Virtex-5 FPGA silicon is hardened against metastability, metastability does occur occasionally at clock boundaries and should be properly handled. For example, the rising edge of the input clock is detected in the mclk clock domain. The first register in the mclk domain rarely experiences metastability. Except for extremely rare cases, measured in decades per event, the output of the first register settles and meets the setup requirements of the second register. Therefore, the output of the second register can be assumed to be reliable, as can the third register.

Therefore, these registers can be used to detect the rising edge of the asynchronous input. Figure 19-15 shows a simple circuit that synchronizes the inputs into a new clock domain through reg_1 and reg_2, and then detects the rising edge when the input to reg_3 is high but the output is still low. This circuit is used in the reference design as the interface to the input and output sample clocks. The timing requirements for these signals are shown in Figure 19-28 and Figure 19-29.

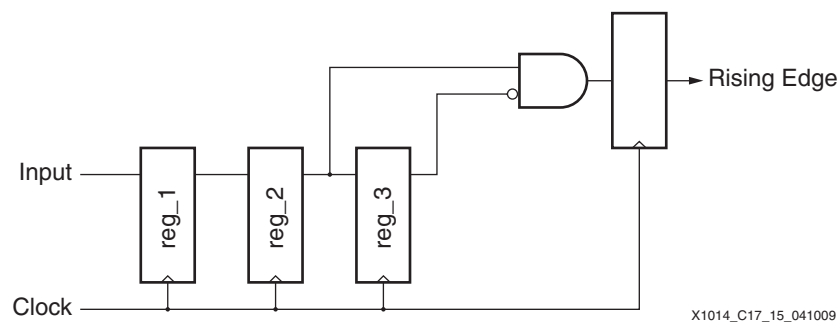
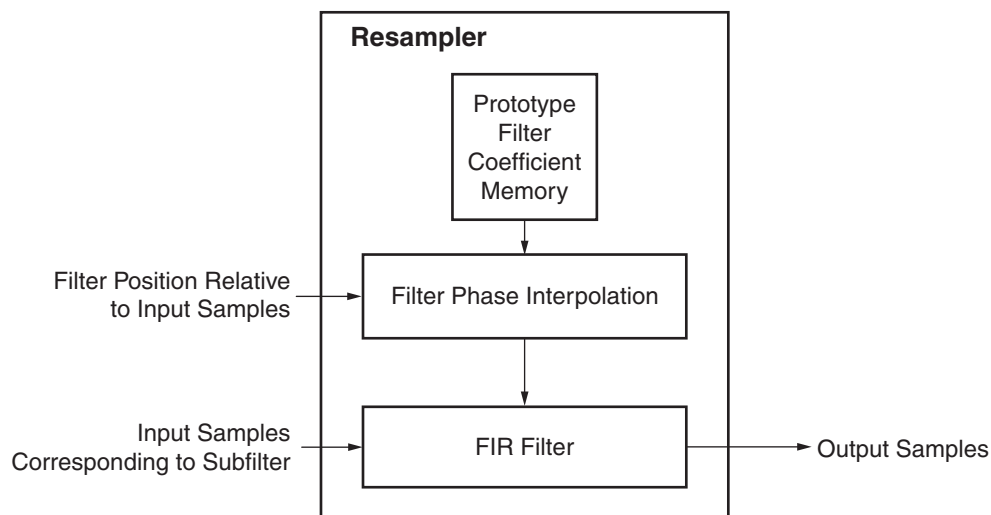


Figure 19-15: Rising Edge Detect

Resampler Functional Block

The resampler creates a set of samples from the input samples based on the output-to-input ratio produced by the ratio detection section. As shown in Figure 19-16, two major computational tasks are required to produce each output sample:

- Interpolate filter coefficients for the convolution based on the prototype filter.
- Perform the convolution of the interpolated coefficients with the corresponding set of input samples.



X1014_C17_16_041509

Figure 19-16: Re-Sampler

Prototype Filter

The prototype filter was designed using the Filter Design and Analysis tool in MATLAB® software. It is a low-pass equiripple filter consisting of 64 phases of 64 taps each. This is done with a filter order of 4096 and frequency specifications of $1/64$ of the desired response of each phase. The resulting coefficients are scaled by a factor of 64 to fully utilize the coefficient bit width and to maintain the signal amplitude. The prototype filter is symmetric, so only half of the coefficients are stored. Because the filter is of order 4096, there are actually 4097 coefficients. The center coefficient is stored in a 24-bit register and the rest are stored in block RAM-based memory of size 2048 x 24 bits.

The parameters used for the prototype filter in the reference design are shown in Table 19-3. The transition band of the filter is symmetric about the Nyquist frequency: $w_{pass} = \text{Nyquist} - 9.3\%$, $w_{stop} = \text{Nyquist} + 9.3\%$. The resulting filter has a passband of 0.4535 times the sampling rate, and a stop band of 0.5465 times the sampling rate. This yields a passband of 20 KHz for a sampling rate of 44.1 KHz, for example.

Table 19-3: Prototype Filter Parameters

Parameter	Value	Comment
Response Type	Low-pass	
Design Method	Equiripple	
Filter Order	4096	
Frequency Spec Wpass	$1/64 \times 2 \times 0.4535$	Normalized
Frequency Spec Wstop	$1/64 \times 2 \times 0.5465$	Normalized
Magnitude Spec Wpass	1	
Magnitude Spec Wstop	50000	
Density Factor	16	

Table 19-3: Prototype Filter Parameters (Cont'd)

Parameter	Value	Comment
Passband Ripple	± 0.016 dB	
Stopband Attenuation	149 dB	

Figure 19-17 shows the frequency response of the resulting filter.

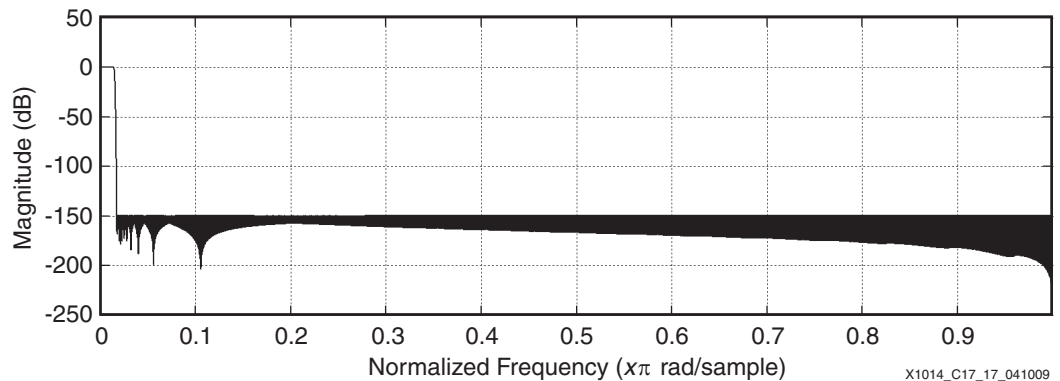


Figure 19-17: Prototype Filter Frequency Response

Figure 19-18 shows the calculated frequency response for the prototype filter transition band. Figure 19-19 shows the measured frequency response for the prototype filter transition based on measurements of the sample rate converter performing a 48 KHz-to-48 KHz asynchronous conversion. Figure 19-20 and Figure 19-21 show the calculated and measured frequency response for the prototype filter passband.

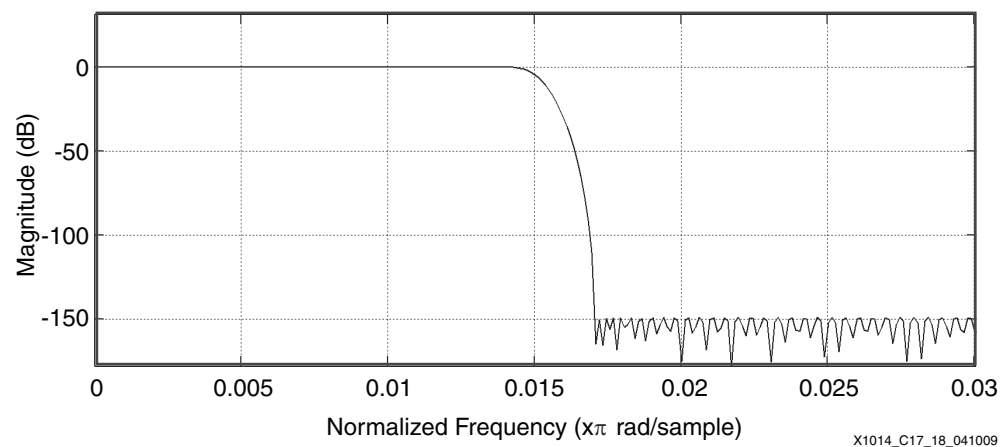


Figure 19-18: Prototype Filter Transition Band (Calculated)

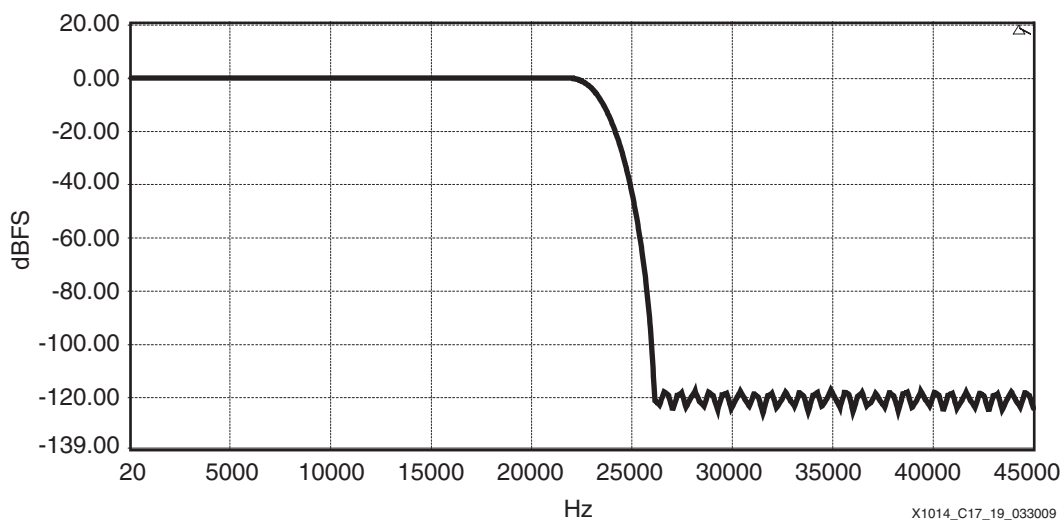


Figure 19-19: Prototype Filter Transition Band (Measured)

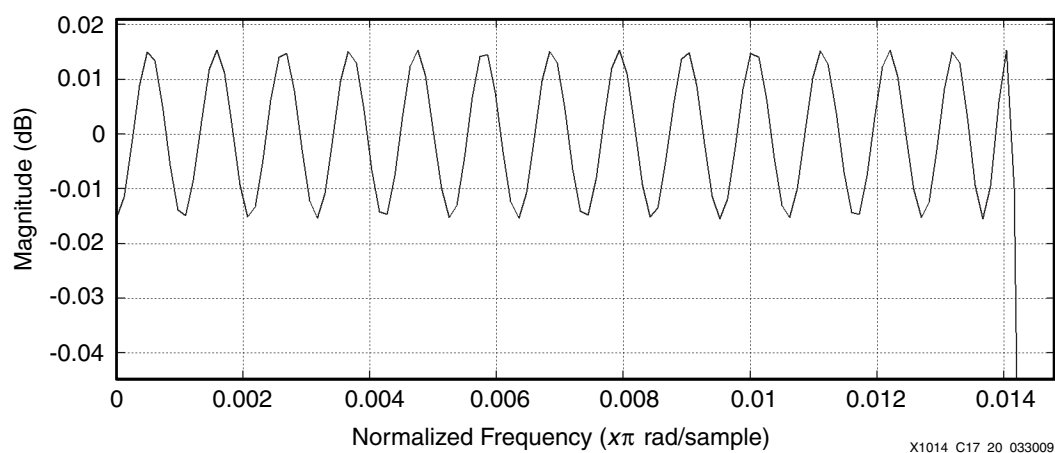


Figure 19-20: Prototype Filter Passband (Calculated)

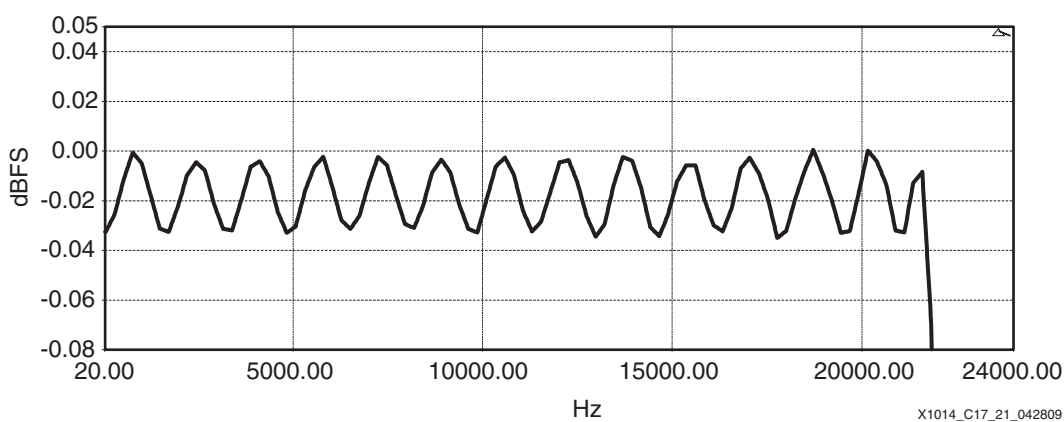


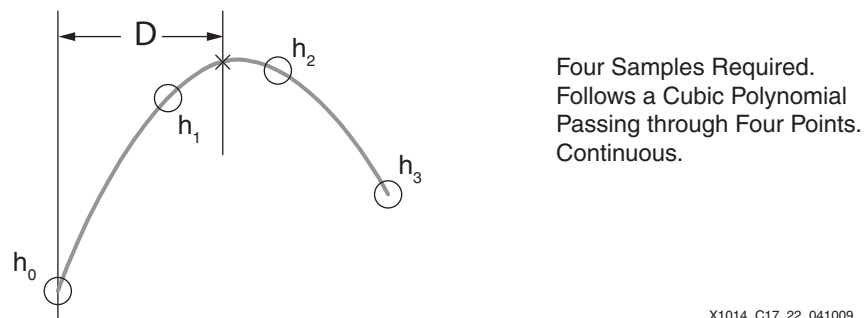
Figure 19-21: Prototype Filter Passband (Measured)

Coefficient Interpolation

The filter coefficients are interpolated with a third-order Lagrange interpolation according to Equation 19-1.

$$y = \left[\frac{-(\Delta-1)(\Delta-2)(\Delta-3)}{6} \right] h_0 + \left[\frac{\Delta(\Delta-2)(\Delta-3)}{2} \right] h_1 + \left[\frac{-\Delta(\Delta-1)(\Delta-3)}{2} \right] h_2 + \left[\frac{\Delta(\Delta-1)(\Delta-2)}{6} \right] h_3 \quad \text{Equation 19-1}$$

where h_0, h_1, h_2 , and h_3 are four adjacent stored coefficients, and Δ represents the difference between the location of the coefficients to be calculated and h_0 . Figure 19-22 illustrates the interpolation, with X marking the point on the cubic polynomial curve used for interpolation.



X1014_C17_22_041009

Figure 19-22: Third-Order Lagrange Interpolation

To minimize the number of multiplications, Equation 19-1 is factored to Equation 19-2.

$$y = \left[\frac{(\Delta-2)(\Delta-3)}{2} \right] \times \left[\frac{-(\Delta-1)h_0}{3} + \Delta h_1 \right] + \frac{\Delta(\Delta-1)}{2} \left[-(\Delta-3)h_2 + \frac{(\Delta-2)h_3}{3} \right] \quad \text{Equation 19-2}$$

The multiplications are performed using DSP48e blocks configured as 4-stage, time-sliced, 35 x 35 multipliers.

Each 35 x 35 multiply makes use of 4 passes through the 18 x 18 DSP48e slice. Input multiplexers and output registers for storage of intermediate results are used in conjunction with the multipliers to form multiply/adder units. Two multiply/adder units are used in parallel for the coefficient interpolation. Each unit operates in a 16-state sequence consisting of four 4-state multiplies.

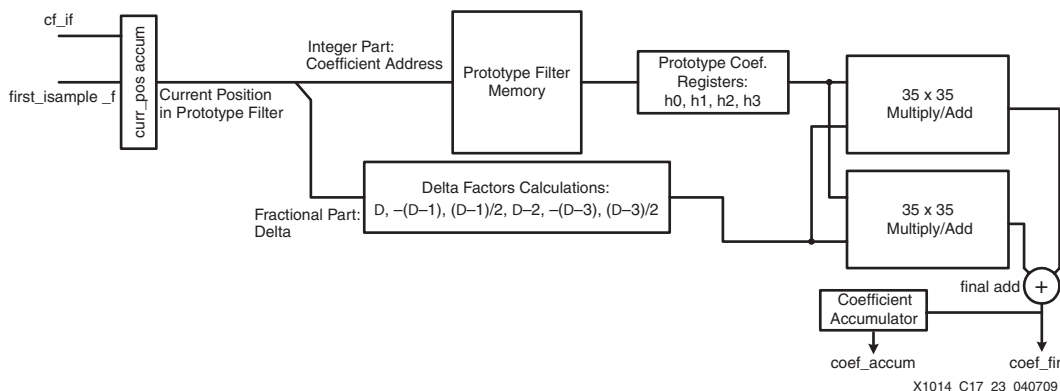


Figure 19-23: Coefficient Interpolator

Figure 19-23 is a block diagram of the coefficient interpolator. The `cf_if` input is the conversion factor from input samples to filter coefficients. This specifies how many coefficients correspond to the distance between input samples. For up-conversion, `cf_if` is 16. For down-conversion, `cf_if` is 16 times the ratio of output rate to input rate. The `first_sample_f` input is the location of the first input sample to be used in the convolution relative to the stored filter coefficients. The `curr_pos_accum` register keeps track of the location of each coefficient to be accumulated relative to the stored coefficients.

When a new output sample calculation is started, the `curr_pos_accum` register is initialized with the `first_sample_f` input. As a new filter coefficient is interpolated, the `curr_pos_accum` register is incremented by `cf_if`. This continues until the end of the stored prototype filter is reached, indicating that all the required coefficients have been interpolated and, consequently, the convolution is complete.

The output of `curr_pos_accum` is the current position of the filter coefficient in filter space. The integer portion of this quantity is the address of the leftmost stored filter coefficient to be used in the Lagrange interpolation. The fractional portion is the delta value to be used in the interpolation.

The four coefficients used in the interpolation (h_0 , h_1 , h_2 , and h_3) are retrieved serially and stored in registers during the 16-state interpolation. Likewise, several factors are calculated from Δ and stored in registers during interpolation. The Δ -related values, along with the associated stored coefficients, are sent to the two 35 x 35 multiply/add units.

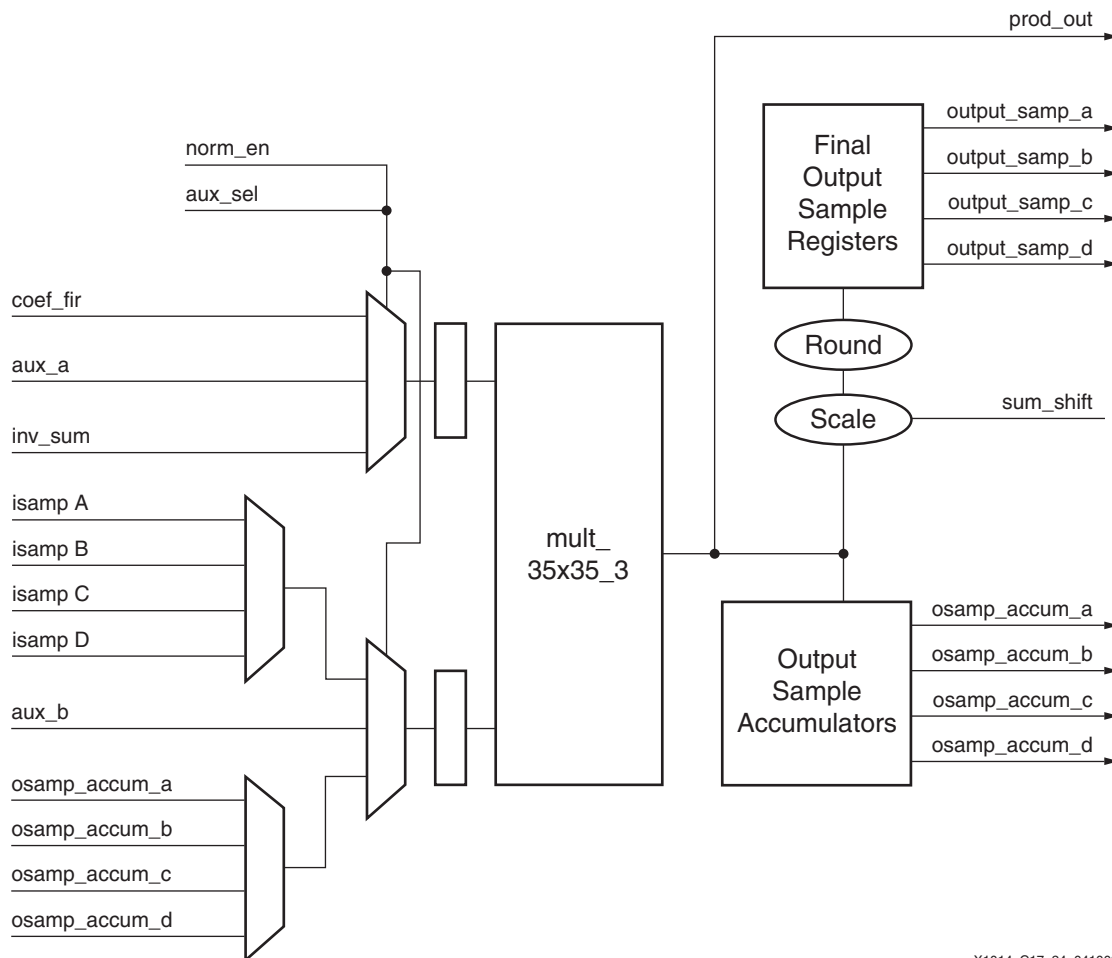
The multiply/add units operate in parallel, multiplying and summing the various terms of the Lagrange interpolation. One final addition produces the interpolated coefficient used in the FIR operation, `coef_fir`, as given by Equation 19-2.

As each FIR coefficient is calculated, it is accumulated in the coefficient accumulator, which accumulates to `coef_accum`. This value is then used to normalize the result of the convolution. In the case of down-sampling, the final `coef_accum` of the convolution is virtually equal to the inverse of the scaling factor required to compensate for the increased length of the convolution. In the case of up-sampling, the final `coef_accum` is virtually equal to one and serves to compensate for small amplitude distortions that might otherwise occur.

FIR Filter

A block diagram of the FIR filter section is shown in Figure 19-24. The FIR Filter section has three distinct functions:

- Implement the resampling FIR filter for each of four channels.
- Perform the sample normalization for each output sample after the FIR operation is complete.
- Serve as a general-purpose multiplier for the control section.



X1014_C17_24_041009

Figure 19-24: FIR Filter Block Diagram

The FIR operates in parallel with coefficient interpolation. As each coefficient (denoted `coef_fir` in Figure 19-24) is interpolated, it is multiplied by the corresponding input sample (`isamp A`, `B`, `C`, or `D`) and the result is accumulated. A 35×35 multiplier contained in the FIR filter unit performs the multiply operations. It operates on the same 16-cycle sequence as the multipliers in the coefficient interpolator. Figure 19-25 shows the sequence of operations in this multiplier. Each of the labeled “cycles” consists of four clocks to perform a single 35×35 multiply. There are separate accumulators for each channel. At the end of the convolution, each accumulator holds the result of the convolution.

<i>cycle</i>	0	1	2	3
Mult 3	lsamp A * coef	lsamp B * coef	lsamp C * coef	lsamp D * coef

X1014_C17_25_033009

Figure 19-25: Multiplier Cycles for FIR Filter

For sample normalization, the accumulated sample values (osamp_accum_a/b/c/d) are passed back through the multiplier. They are each multiplied by the inverse of the sum of the coefficients (inv_sum). This normalizes the output samples, negating gain that might otherwise occur, particularly when performing down-conversion.

Because the coefficient sum can have a value from near one up to about eight (in the case of down conversion by the maximum factor of 7.75), a floating-point scheme is used to preserve all significant bits through the multiply operation. The inverse sum (inv_sum) is block normalized prior to being sent to the FIR filter section. The sum_shift input denotes the number of non-zero integer bits, and thus how much inv_sum was shifted to the right. Therefore, the result of the multiply must be scaled to compensate for this. Because the sum that was shifted to the right has been inverted before the multiply, the result of the multiply is also shifted to the right to compensate. This is shown as the scale operation in Figure 19-24.

A rounding operation to 24 bits is provided with a corresponding enable. However, the best linearity at low amplitude is obtained without this rounding, so it is disabled in the reference design. The final result, after scaling and rounding, is stored in the final output sample registers. These registers are the outputs of the top-level module.

Up to four audio channels, A, B, C, and D, are accommodated. More channels are accommodated by adding additional instances of the FIR filter block.

The third function of the FIR Filter section—to serve as a general purpose multiplier for the control section—is enabled by the aux_sel input. These auxiliary multiplications are used by the control section for such functions as converting locations from input sample space to filter coefficient space. The additional, unrelated multiplications use an auxiliary set of inputs and outputs. The output of the auxiliary multiplies appears on the prod_out output.

Signed Fractional Divider

A module called `divide_sign_fract` is a signed 27 x 27 multi-cycle pipelined divider with 25 fractional output bits. It has a latency of 53 states. This divider is used in 2 ways:

- To produce the ratio and its inverse (1/ratio).
- To produce the inverse of the sum of coefficients (1/coef_accum) that is used to normalize the result of the sample accumulation. This normalization is performed in the FIR filter section.

The accuracy of these calculations directly affects the quality of the sample rate conversion. Thus, a high degree of precision is required. Because the divide operations are done very infrequently compared to other operations, the divider is optimized for minimum area, and thus, low throughput.

A round_en input to this module allows the signed fractional result to be rounded to 24 bits. If enabled, the rounding function is rounded to the nearest integer. If the number to be rounded is equidistant from two integers, it is rounded away from zero. In the reference design, this rounding is enabled.

Control

The high-level control for the reference design is contained in the `timing_control_multi_ch` module. The output sample clock starts the sample calculation sequence. Each time an output sample is taken, as indicated by `output_clk`, a new one is calculated. Whereas ratio detection operates in a more or less continuous fashion, the resampler operates in a burst fashion, interpolating a filter phase and performing the convolution each time an output sample is taken, then idling until the next sample is taken and a new one can be calculated. The idle time can be very short (for example, when the output rate is very high), or very long (for example, when the output rate is low, and the ratio is near 1:1). In any case, the computation starts at the rising edge of the output sample clock and terminates when the end of the prototype filter is reached.

The creation of an output sample is controlled by the timing control state machine. Figure 19-26 shows a basic diagram of the state machine. The state register holds the current state. A count is associated with each state that determines the minimum time each state lasts. State changes occur only when the state delay count times out, which is indicated by `state_dly_tc`. The status inputs and the current state determine if a state change occurs.

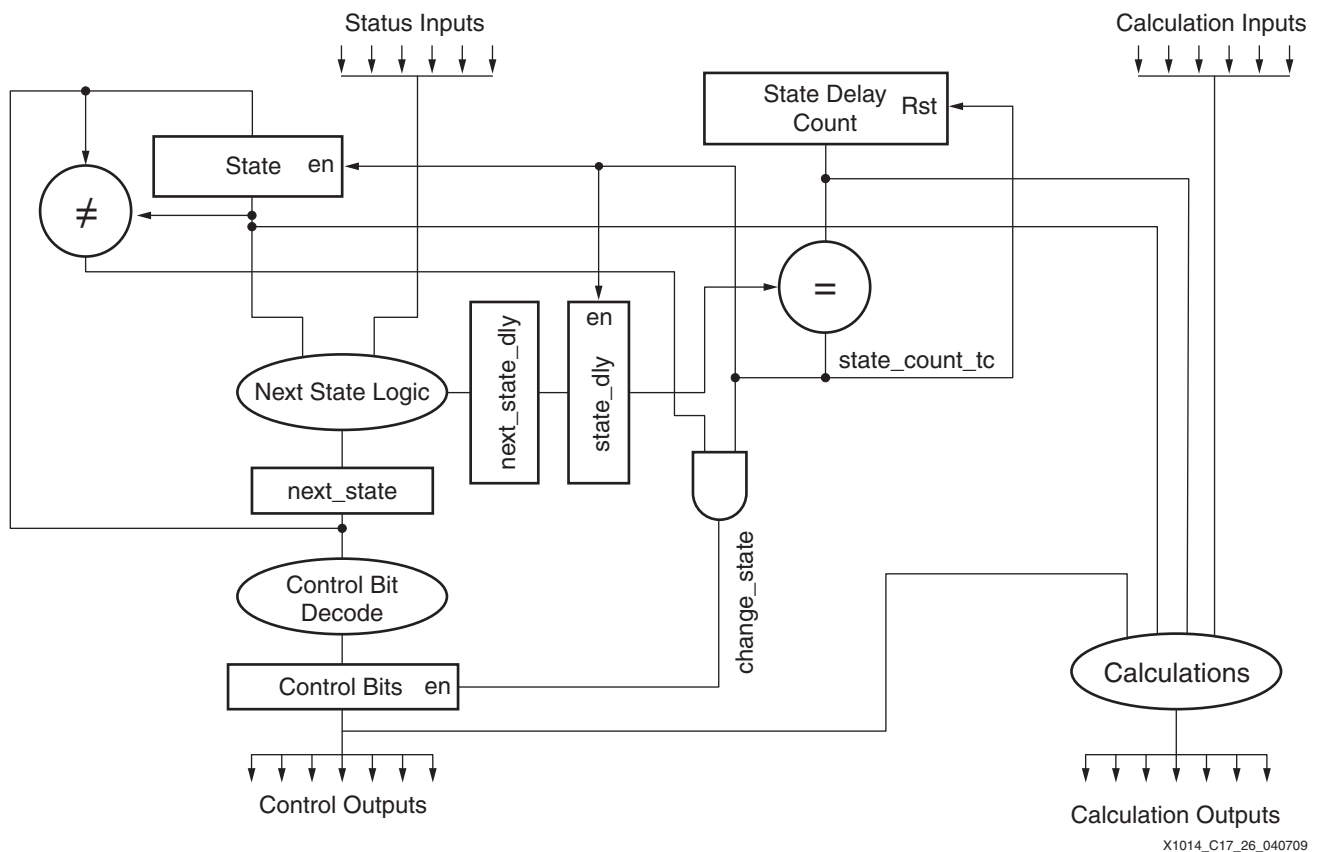


Figure 19-26: Timing Control

The main purposes of the state machine are to control access to shared multiply-and-divide resources, sequence the data, and load the results into registers at the proper time. The current state, control bits, and state counter all contribute to the control and timing of these calculations.

Figure 19-27 shows the state diagram of the top-level control state machine in the `timing_control_multi_ch` module. The initial state is **RATIO**. In this state, the signed fractional divider is used to calculate the ratio from the most recent `in_period_sync` and `out_period_sync` values. The state machine remains in this state until the `get_new_sample` signal asserts in response to the rising edge of `clkout`.

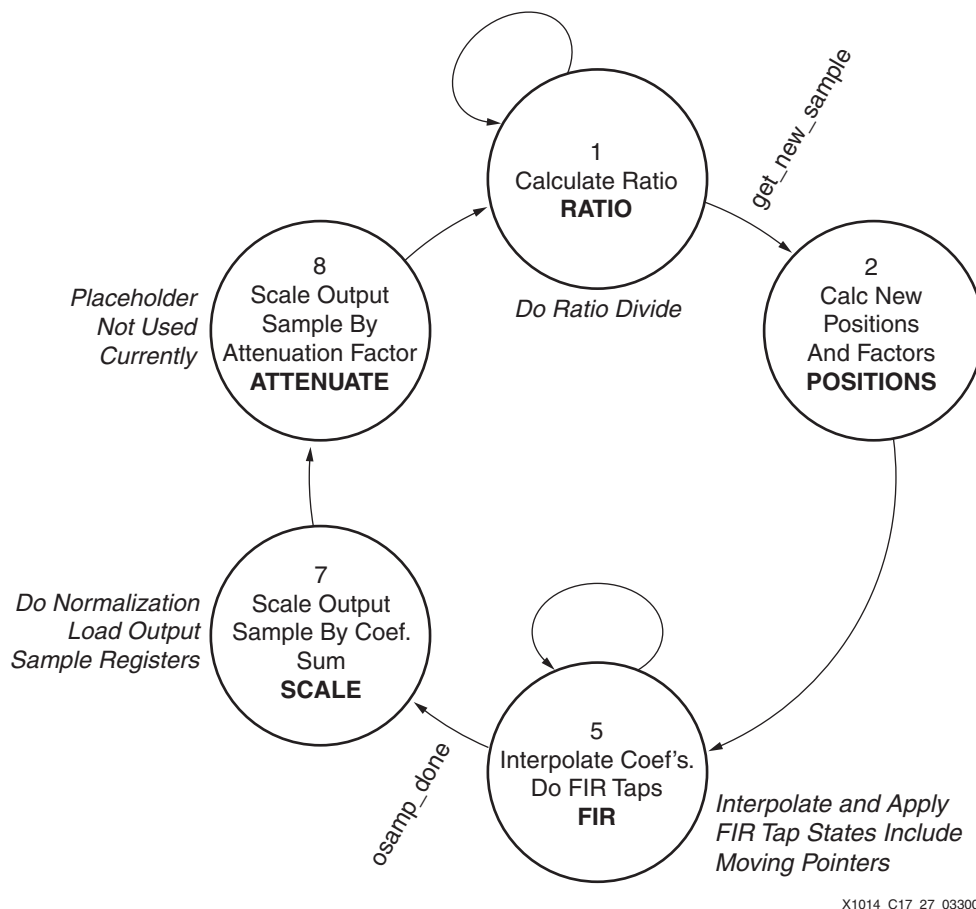


Figure 19-27: Top-Level Control State Machine

In the **POSITIONS** state, a new position for the output sample relative to the input samples is calculated based on the ratio. There are two sources for this. It can come from the regulated ratio, or from the manual ratio. The start position in the prototype filter is also calculated. The auxiliary functionality of the multiplier in the FIR filter section is used in some of these calculations. The **INIT_FIR** state simply allows the initial positions to propagate.

The bulk of these calculations happen during the **FIR** state. In this state, the `go_fir` signal pulses to indicate the start of a new FIR filter sequence. The resampler is then reset and enabled to interpolate and apply filter coefficients to the input samples. The result is a single output sample. The state machine remains in this state until the `osamp_done` signal is asserted by the resampler, indicating that the prototype filter has been traversed and the FIR filter operation is completed.

The **SCALE** state uses the `divide_sign_fract` module to normalize the accumulated results of the FIR filter by dividing the accumulated results of the FIR by the accumulated

coefficients, `coef_sum`. Each audio channel is normalized and clamped independently to produce the final output sample value.

The ATTENUATE state is not currently used in the reference design. It is a state in which attenuation of the output can be performed for fade out, fade in, or overall magnitude control. The auxiliary multiplier functionality can be used to accomplish this.

Interface Timing

All data processing in the ASRC is done in the `mclk` (high-speed processing clock) domain. In general, the `mclk` domain has no relationship to either the input or output clock. The input and output sample clocks are treated as enables that specify when input data is valid and when output data is taken. The period of the sample clocks is used to determine the conversion ratio.

Figure 19-28 shows the timing requirements for input samples. The `clk_in` signal is resampled in the `mclk` domain with the circuit shown in Figure 19-15, page 433, and the rising edge is used to determine when data is valid. Therefore, setup and hold times are specified in terms of `mclk` periods. Relative to the rising edge of `clk_in`, the setup requirement is zero `mclk` periods and the hold requirement is five `mclk` periods. The `clk_in` signal must be High for a minimum of five `mclk` periods and Low for a minimum of five `mclk` periods for accurate edge detection.

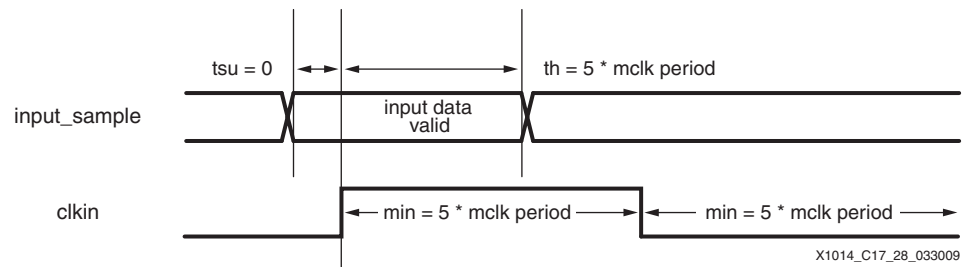


Figure 19-28: Input Timing

Figure 19-29 shows the timing characteristics of output samples. Because they are created in the `mclk` domain, the timing specifications are also given in terms of `mclk` periods. Relative to the rising edge of `clkout`, output sample data is valid a minimum of three `mclk` periods before `clkout` and remains valid until the next sample is presented. The figure of 100 `mclk` periods is given as a conservative minimum time that the output data is valid after the rising edge of `clkout`. Like `clk_in`, `clkout` (an input to the SRC) is resampled in the `mclk` domain to determine the output sampling rate. Therefore, it has a minimum High-time requirement of 5 `mclk` periods, and a minimum Low-time of 5 `mclk` periods.

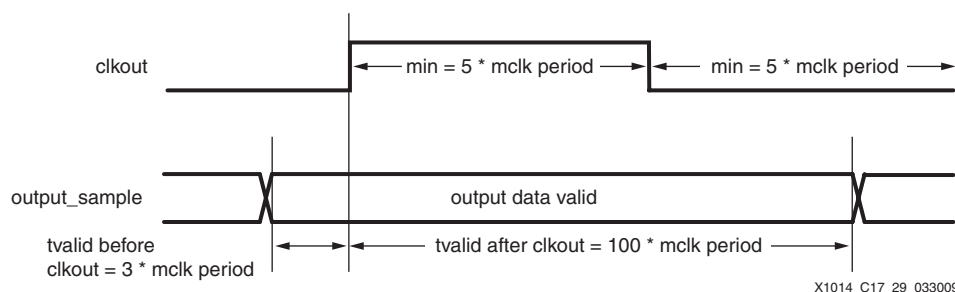


Figure 19-29: Output Timing

Performance

This section discusses the performance of the reference design in terms of THD+N, maximum conversion ratios, sample frequency range, and latency.

THD+N

Typical overall THD+N performance is -133 dB. The performance varies somewhat according to the input and output sample rates and the frequency content of the signal, ranging from -125 dB to -139 dB.

To illustrate the performance of the ASRC, a 1 KHz sine tone was input into the ASRC for two particular conversion ratios, 48 KHz to 48 KHz and 44.1 KHz to 48 KHz, in the presence of worst-case jitter. A 64K point Fast Fourier Transform (FFT) was then taken of the output. [Figure 19-30](#) shows the FFT output for the 48 KHz-to-48 KHz conversion, and [Figure 19-31](#) shows the output for the 44.1 KHz-to-48 KHz conversion.

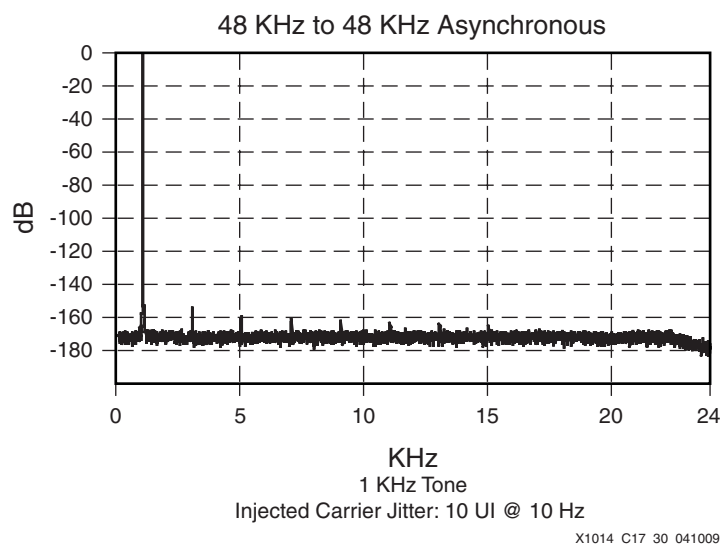


Figure 19-30: FFT for 48 KHz to 48 KHz Asynchronous Conversion

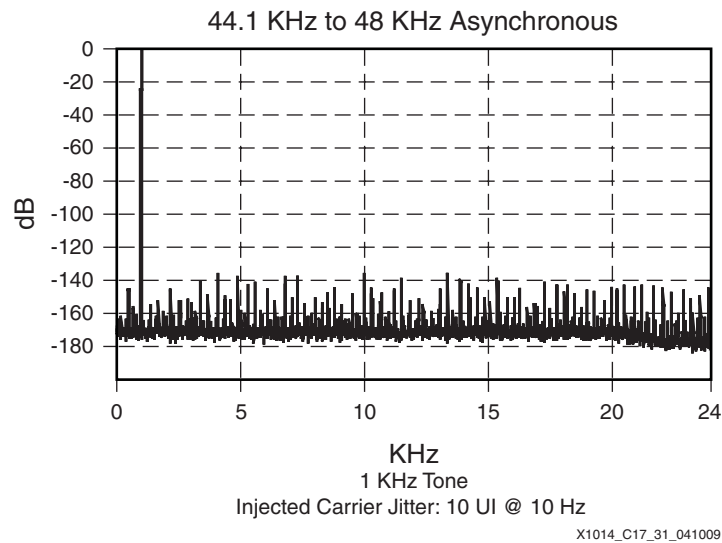


Figure 19-31: FFT for 44.1 KHz to 48 KHz Asynchronous Conversion

Table 19-4 lists performance measurements taken over a variety of common ratios. Measurements were taken with a Prism Sound dScopeIII audio analyzer in the default THD+N measurement mode. Frequency was scanned from 20 Hz to 20 KHz. The readings hold over the jitter tolerance curve of Figure 19-9, page 425. These are only examples of possible conversions. The ASRC reference design allows for virtually infinite combinations of input and output frequencies within the maximum frequency and maximum ratio constraints.

Table 19-4: THD+N Performance vs. Conversion Frequency

Input Sample Frequency		Output Sample Frequency (KHz)				
		32	44.1	48	88.2	96
32	Minimum	-135	-125	-125	-125	-125
	Maximum	-137	-130	-137	-133	-137
	Typical	-136	-127	-130	-130	-135
	1 KHz tone	-135	-128	-127	-131	-135
44.1	Minimum	-128	-133	-125	-126	-125
	Maximum	-135	-137	-135	-137	-133
	Typical	-130	-136	-129	-136	-131
	1 KHz tone	-131	-135	-126	-135	-129
48	Minimum	-125	-126	-135	-128	-127
	Maximum	-137	-131	-138	-132	-137
	Typical	-132	-128	-136	-131	-136
	1 KHz tone	-133	-127	-136	-128	-135

Table 19-4: THD+N Performance vs. Conversion Frequency (Cont'd)

Input Sample Frequency		Output Sample Frequency (KHz)				
		32	44.1	48	88.2	96
88.2	Minimum	-130	-134	-129	-131	-128
	Maximum	-136	-137	-136	-137	-134
	Typical	-132	-136	-131	-137	-131
	1 KHz tone	-133	-136	-129	-135	-130
96	Minimum	-136	-129	-136	-128	-134
	Maximum	-137	-133	-139	-134	-138
	Typical	-137	-131	-137	-132	-137
	1 KHz tone	-136	-130	-137	-129	-136

Maximum Conversion Ratios

The maximum up-conversion ratio is 8:1. The maximum down-conversion ratio is 1:7.5. The range of the up-conversion is limited by the number of integer bits in the ratio calculation. The down-conversion ratio is limited by the amount of input sample storage memory, and how much of this memory is allocated to the input FIFO.

The input sample frequency range is 8 KHz to 196 KHz. The output frequency range is also 8 KHz to 196 KHz. These sample frequency ranges are valid for a design with a 250 MHz mclk. A slower or faster mclk reduces or increases both minimum and maximum sample frequency in approximate proportion to the change in mclk. The upper limit is a factor of processing clock frequency. The lower limit is a function of the width of the period counters and the processing clock frequency.

Latency

For any given conversion ratio, the latency of the design is fixed. It is determined by the phase delay of the FIR filter and the fill level of the input FIFO. The FIFO level is fixed at 16, but the size of the FIR filter, and consequently its phase delay, vary in the case of down-conversion. Therefore, the formula for latency is different depending on whether the sample rate converted is performing up-conversion or down-conversion. The formula for determining up-conversion latency is given in Equation 19-3. The filter length is 64. Therefore, the phase delay is 32 sample periods. The latency in milliseconds depends on the input sample frequency.

$$\begin{aligned}
 \text{Latency} &= \text{phase delay} + \text{FIFO delay} \\
 &= 32 + 16 \\
 &= 48 \text{ input sample periods}
 \end{aligned}
 \tag{Equation 19-3}$$

The equation for down-conversion latency is given in Equation 19-4. For down-conversion, the filter is spread across more samples, so the phase delay and subsequently the latency are longer in terms of the number of input samples.

$$\begin{aligned}
 \text{Latency} &= \text{phase delay} + \text{FIFO delay} \\
 &= 32 \cdot (f_{s_{out}}/f_{s_{in}}) + 16
 \end{aligned}
 \tag{Equation 19-4}$$

Examples:

$$\begin{aligned}
 &48 \text{ KHz} : 48 \text{ KHz conversion:} \\
 \text{Latency} &= 32 + 16
 \end{aligned}$$

$$= 48 \text{ input sample periods}$$

$$= 1 \text{ ms}$$

48 KHz : 96 KHz up-conversion:

$$\text{Latency} = 32 + 16$$

$$= 48 \text{ input sample periods}$$

$$= 1 \text{ ms}$$

32 KHz : 48 KHz up-conversion:

$$\text{Latency} = 32 + 16$$

$$= 48 \text{ input sample periods}$$

$$= 1.5 \text{ ms}$$

96 KHz : 48 KHz down-conversion:

$$\text{Latency} = 32 \bullet 2 + 16$$

$$= 80 \text{ input sample periods}$$

$$= 0.83 \text{ ms}$$

48 KHz : 44.1 KHz down-conversion:

$$\text{Latency} = 32 \bullet 48/44.1 + 16$$

$$= 50.83 \text{ input samples}$$

$$= 1.06 \text{ ms}$$

In cases of changing frequency, the latency changes smoothly as specified in [Equation 19-3](#) and [Equation 19-4](#). For up-conversion, changes in input sample frequency result in changes in latency, while changes in output sample frequency do not. For down-conversion, changes in input or output sample frequency result in changes in latency.

FPGA Resource Utilization and Performance

The reference design has been implemented and hardware verified in Virtex-5 devices. [Table 19-5](#) and [Table 19-6](#) show the design resource utilization and clock frequencies achieved, respectively.

Table 19-5: Reference Design Resource Utilization

Number of Channels	LUTs	Registers	DSP48e Slices	Block RAMs
2	2,393	2,552	3	3
4	2,460	2,709	3	5
16	3,839	4,035	6	14
Each group of 4 additional channels	484	442	1	3

Table 19-6: Reference Design Frequency Performance

Speed Grade	Processing Clock Frequency (mclk) (MHz)	General Maximum Sample Frequency (KHz)
-12	300	224
-11	275	200
-10	250	180

The minimum required processing clock frequencies for up-conversion is given in [Equation 19-5](#).

$$F_{mclk} = F_{s_{out}} \times 1350 \quad \text{Equation 19-5}$$

The minimum required frequency for down-conversion is given in [Equation 19-6](#).

$$F_{mclk} = F_{s_{in}} \times 1030 + F_{s_{out}} \times 295 \quad \text{Equation 19-6}$$

For example, a down-conversion from 192 KHz to 48 KHz requires a clock frequency of $192 \text{ KHz} \times 1030 + 48 \text{ KHz} \times 295 = 221.72$. Therefore, a -10 part works for this particular conversion.

For easy reference, [Table 19-7](#) shows approximate clock frequencies required for some common conversions. These values reflect the formulas for up-conversion and down-conversion.

Table 19-7: Approximate Processing Clock Frequencies (MHz) Required for Various Conversions (MHz)

Input Sample Frequency (KHz)	Output Sample Frequency (KHz)						
	32	44.1	48	88.2	96	144	192
32	45	60	65	120	130	195	260
44.1	55	60	65	120	130	195	260
48	60	65	65	120	130	195	260
88.2	105	105	110	120	130	195	260
96	110	115	115	125	130	195	260
144	160	165	165	175	180	195	260
192	210	215	215	225	230	245	260

Data Flow Spreadsheet

The reference design includes a data flow spreadsheet, `asrc_dataflow.xls`, that graphically illustrates the timing of data as it flows and gets processed from register to register through the pipeline. The registers are listed across the top of the spreadsheet and each row represents one clock cycle, progressing from top to bottom. There are three sheets corresponding to three stages of processing: filter phase interpolation, FIR filtering, and output sample normalization.

The data flow spreadsheet is an aid to understanding the intent of the RTL, as well as a powerful tool for debugging, and making and documenting modifications. Multi-cycle paths are apparent from inspection of the dataflow spreadsheet. Comments are used on some of the cells to give cycle-specific explanations. There are enough cycles shown to illustrate the entire start-up sequence and the process end sequence for an output sample calculation. Obviously, many more cycles are required to perform filter interpolation and FIR filtering to obtain an output sample.

Filter Phase Interpolation and FIR Filter

The filter interpolation and FIR filter blocks operate on a 16-state cycle matching the 16-state cycle of the multipliers (four multiplications of four states each). The left columns of the spreadsheet contain the cycle and sub-cycle counts, and column labeled pass. The pass field is for correlation of the start-up sequence between the filter phase interpolation page and the FIR filter page.

The cells in the interior of the spreadsheet indicate the data present in a particular register at a particular time. The registers are arranged so that the flow of data is generally upper left to lower right. In most cases, there are only entries for new, valid data. Blank cells indicate, depending on the context, either invalid data or data that remains the same as previous entries or not relevant to the data at that time. To the right of the register list is a list of the state machine output bits. These illustrate the timing relationship of state machine bits to datapath registers. One cycle of the state machine is boxed. The boxed area contains a complete state machine cycle and corresponds directly to the RTL state machine definition. The state machine also contains unused control bits. These bits facilitate the addition of control outputs to the state machine. If unused, these bits are removed during synthesis.

Output Sample Normalization

The output sample normalization page in the spreadsheet shows data flow in the `timing_control_multi_ch` module, particularly the last stage of output sample processing done during the SCALE state of the timing control state machine (see [Figure 19-27, page 442](#)). This sheet details the data sequencing through the divider for normalization, and the timing for the steps of rounding, clamping, and loading the output registers. Register names are shown in the top of each column. Data for each audio channel (A, B, C, and D) passes through these stages in sequence. Although time slots for four audio channels are shown, only two (A and B) are used in the two-channel reference design. This page of the spreadsheet reflects the latency (56 mclk periods) and throughput (one result per eight states) of the divider. Because this is a shared divider, the timing also applies to the other operations it performs.

Design Files

The reference design for the asynchronous sample rate converter is available at <https://secure.xilinx.com/webreg/clickthrough.do?cid=119758>. Open the ZIP archive and extract the file `xapp1014_c19_asrc_multi_ch.zip`.

Conclusion

The asynchronous sample rate converter reference design is useful for converting between source and destination audio streams that have independent clocking sources. It has excellent THD+N performance, and converts a wide range of frequencies and frequency ratios. Additionally, it does so automatically, detecting the conversion settings and tracking changes in frequency. The reference design is implemented in Virtex-5 FPGAs using dual-port block RAM and DSP48e slices, in addition to general logic resources. The implementation uses an interpolated-coefficient FIR filter. It precisely interpolates millions of filter phases from a small set of stored phases and applies the interpolated filter to the corresponding input samples to obtain an output sample at a particular instant. The latency is low and deterministic due to the use of a single FIR filter stage. The source code and data flow spreadsheet, along with the functional description and filter parameters, are provided to facilitate integration of the design as is, or for adaptation of the design for specific application requirements.

Audio Multiplexer for HD-SDI Video

Introduction

In many applications, audio is embedded in the video stream to facilitate the transport of the combined audio and video stream. For HD-SDI video, the audio information can be multiplexed into the HANC portion of the video. Refer to SMPTE 299M for particulars of the data formats for embedded audio.

This chapter describes how audio embedding is done and presents a hardware-verified reference design that implements an audio multiplexer for HD-SDI video.

Features

Some of the features of the audio multiplexer are:

- Single audio group granularity: One instance of the multiplexer embeds the four channels (two channel pairs) of a single audio group. The audio group number is an input to the design. Audio from multiple audio groups can be embedded by using multiple instances of the basic design in a daisy-chain fashion.
- Audio control packet support: Control packets are generated and inserted on the appropriate lines. Input ports are provided for control packet information. Control packet insertion can be disabled.
- 24-bit audio at multiple sample rates: Audio is received as 24 bits plus Z, V, U, and C flags with an audio valid strobe to indicate timing.
- Multiple HD-SDI standards: An input field designates what line standard is being used.
- Synchronous and asynchronous audio support.
- Automatic clock phase and error correction code (ECC) generation.
- Overwrites of incoming embedded audio: Existing embedded audio packets (data and control) in the designated group are overwritten with new packets to maximize utilization of ancillary space. Other ancillary data is passed. All embedding can be disabled, in which case no packets are added or deleted, and the video stream is passed unaltered.
- Support of many Xilinx® device families: The reference design has been hardware verified on the Virtex®-5 FPGA but uses features common to all Xilinx FPGAs.

Embedded Audio

Audio information is embedded in SDI data streams in the form of packets that are placed in the horizontal blanking period of the video stream. The general specification for HD-SDI

video formats and timing is given in SMPTE 292M. Among other things, this standard specifies how timing references and line numbers are transmitted.

Embedded audio packets are just a few of many different types of ancillary packets that might be inserted in the video stream during blanking periods. SMPTE 291M specifies the general format of ancillary packets (see [Figure 20-1](#)). Common elements include a three-word ancillary data flag (ADF), a data identity (DID) to specify the packet type, a data block number (DBN), and data count (DC) fields. A checksum (CS) is the final field of every ancillary packet.

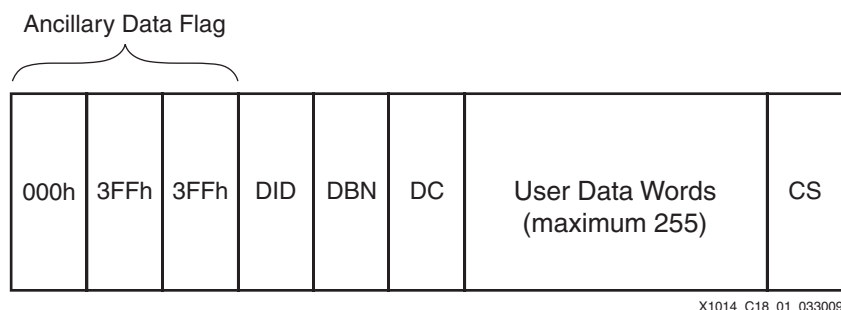


Figure 20-1: Ancillary Data Packet

There are two types of embedded audio packets: audio data packets and audio control packets. Audio data packets contain audio samples for the various channels. Audio control packets contain various metadata fields describing such things as sample rates and audio processing delay. The DID values for embedded audio in HD-SDI are shown in [Table 20-1](#). Each audio data packet contains four samples, one from each channel in a group. Likewise, each audio control packet contains metadata that applies to one audio group.

Table 20-1: DID Values for HD Embedded Audio Packets

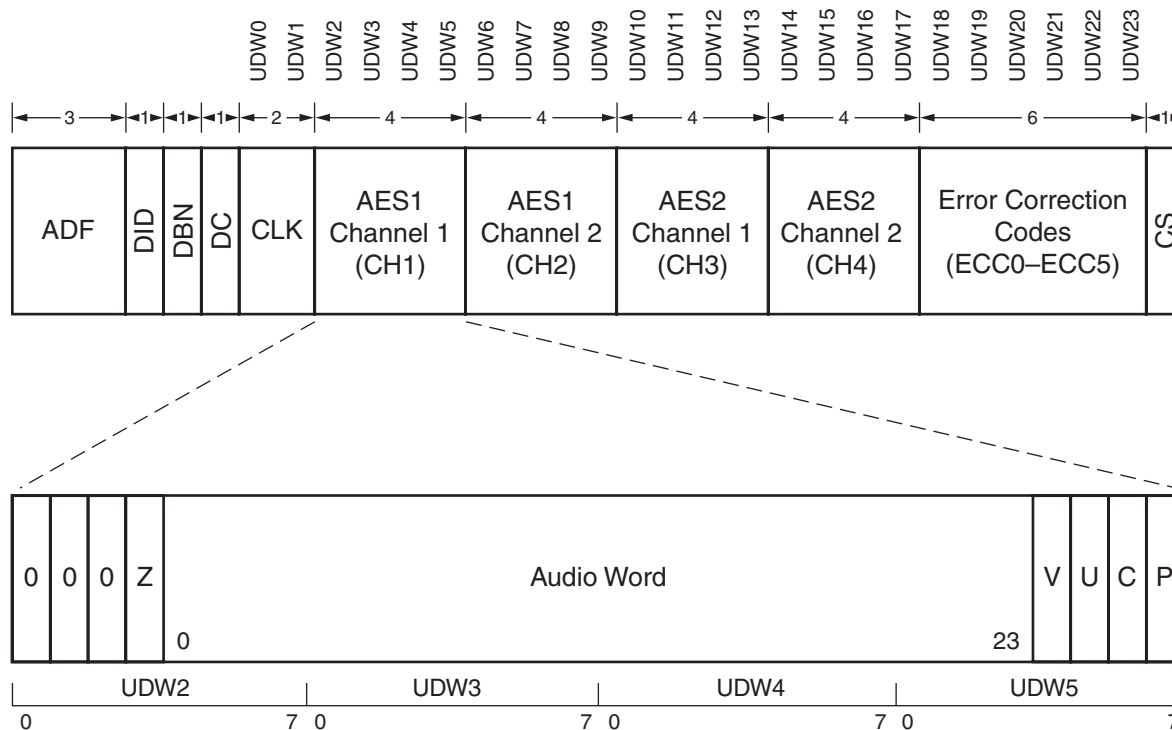
Group	Audio Data Packets	Audio Control Packets
1	0x2E7	0x1E3
2	0x1E6	0x2E2
3	0x1E5	0x2E1
4	0x2E4	0x1E0

While ancillary data packets are required to be contiguous and begin immediately after the EAV, the exact location of packets of a given type is not specified. Therefore, to multiplex audio into the video stream, all ancillary data types must be examined to identify obsolete packets that must be deleted. The ancillary data types must also be examined to determine where to insert new audio packets. This means that the DID field must be checked, and the DC must be read to determine the length of the packet and thus, where to look for the next packet.

Audio Data Packets

The format of audio data packets is shown in [Figure 20-2](#). These have the standard ancillary packet format. The value of DID depends on the audio group, as specified in [Table 20-1](#). DC is always 0x218. The DBN increments for each packet of the same DID. In addition, the audio data packet contains 24 UDWs, a two-word clock phase field (CLK), six

ECC words, and four audio words (two stereo pairs). The fields encompassed by the UDWs are described in this section.



X1014_C18_02_033009

Figure 20-2: Audio Data Packet Format

Audio data packets can be multiplexed onto any video line, with the exception of the lines following the synchronous switching points. The samples must be more or less evenly distributed. The number of audio data packets on any line containing packets can vary by at most one.

Audio Sample Words

The audio sample words are in AES format. Each AES word (AES_n) includes Z, V, U, C, and P bits. The Z bit is present for CH1 and CH3 only. The audio word itself spans the four UDWs as shown in Figure 20-2. All of the words in the audio data packet are 10 bits wide. The eight LSBs contain the actual data. Bit 8 contains even parity for the eight LSBs. Bit 9 contains the inverse of bit 8.

Audio Clock Phase Data

The CLK field contains information about the timing of the audio clock with respect to the video lines. Specifically, the audio clock phase is the number of video clock times (rounded to the nearest integer) between the samples contained in the packet and the EAV immediately preceding the arrival of the audio clock for those samples. Due to the prohibition of packets in the line following the synchronous switching point, the audio sample packet sometimes occurs more than a line time after the EAV to which it is referenced. For these cases, a multiplex position flag (mpf) is asserted. The mpf flag indicates that the given clock phase is from the next-to-last EAV. This implies that there must be at least two video lines of latency in the demultiplexing process.

Error Correction Codes

UDWs 18 to 23 are error correction codes. The reference design of this chapter produces error correction codes in the ECC encoder section.

Reference Design

The reference design for the audio multiplexer is for one audio group. Multiple instances can be daisy-chained to multiplex audio for multiple audio groups. Ancillary data other than for the targeted audio group is passed unaltered. The block diagram for the audio multiplexer is shown in [Figure 20-3](#).

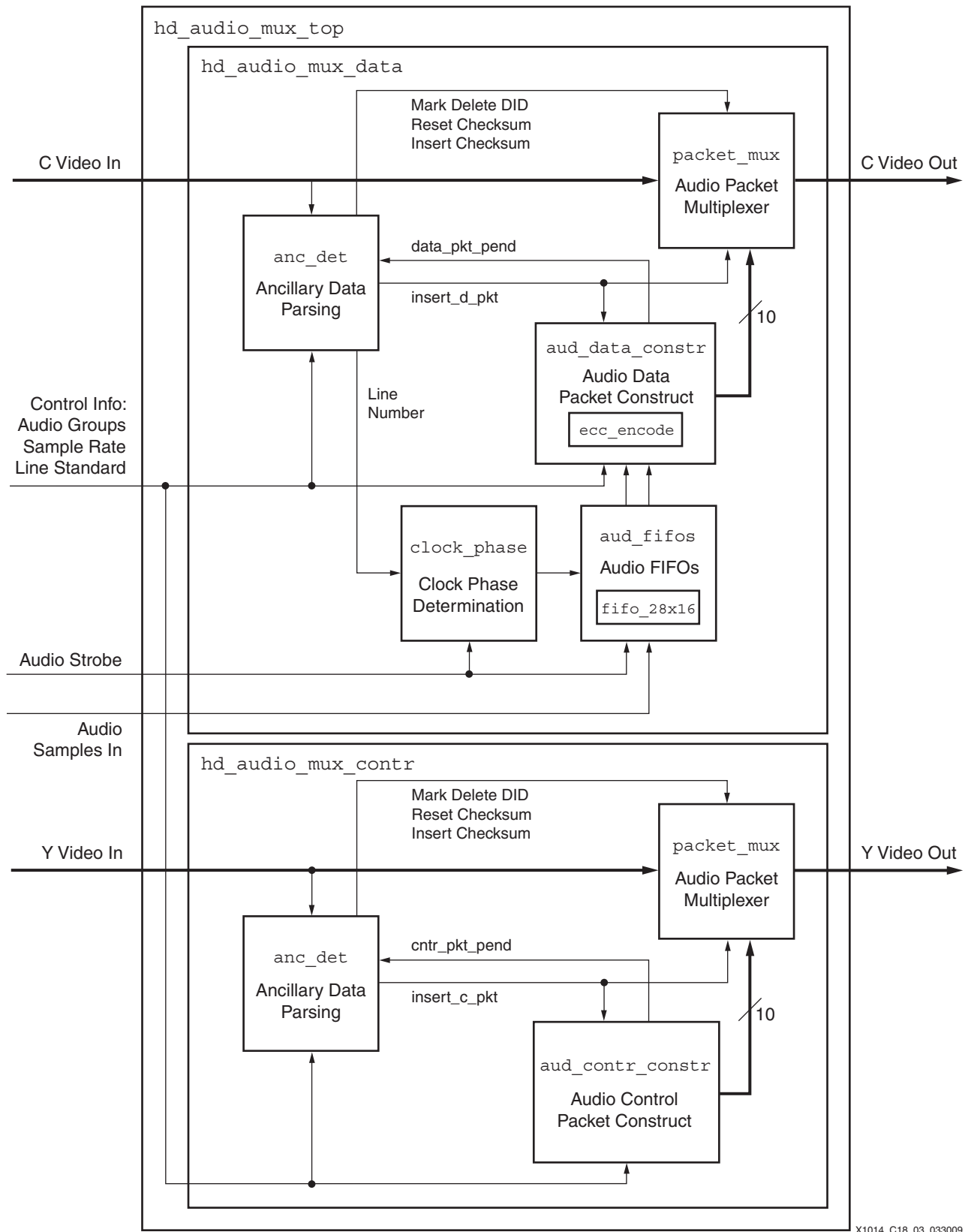


Figure 20-3: HD-SDI Audio Multiplexer Block Diagram

All video data passes through the audio multiplexer, but only ancillary data is modified. The C_{B/C_R} video stream passes through the `hd_audio_mux_data` module for embedding of audio data packets. The Y video stream passes through the `hd_audio_mux_contr` module for embedding of audio control packets. In each case, the stream is monitored by the Ancillary Data Parsing module. This module looks for the presence of ancillary data packets and keeps track of line numbers. Two types of modifications can be done to ancillary packets: existing audio packets can be marked for deletion, and audio packets can be inserted. The actual modifications are done in the Audio Packet Multiplexer module. This module is controlled by the Ancillary Data Parsing module and receives packet data from either the Audio Data Packet Construct module for data packets or the Audio Control Packet Construct module for control packets.

Audio data packets are assembled by the Audio Data Construct module. This module receives timing and control information from the Ancillary Data Parsing module and data from the `aud_fifos` module. At the appropriate time, the Audio Data Construct module feeds packet data to the Audio Packet Multiplexer module for inclusion in the video stream. As the data is fed to the Audio Packet Multiplexer module, ECCs are calculated and also sent to be included in the packet. The Audio Control Packet Construct module performs a similar function, providing control packet words at the appropriate time. It receives control fields directly from inputs to the module.

The Clock Phase Determination module receives line numbers and inferred line timing from the Ancillary Data Parsing module. It also receives the audio strobe, indicating the timing of the audio samples. From this, the Clock Phase Determination module creates clock phase data that is included in the audio data packets. This clock phase information is stored, along with the samples themselves, in `aud_fifos`. The `aud_fifos` module store samples in sets of four, one for each channel, along with the clock phase information.

Inputs and Outputs

Table 20-2 lists the inputs and outputs of the top-level module, `hd_audio_mux_top`.

Table 20-2: Inputs and Outputs of `hd_audio_mux_top` Module

Signal	Direction	Description
<code>vid_clk</code>	In	This is the video clock.
<code>aud_clk</code>	In	This is the input audio clock for audio data FIFOs.
<code>rst</code>	In	This is an asynchronous reset.
<code>vid_ce</code>	In	This is the video clock enable for audio data FIFOs.
<code>vid_samp_ce</code>	In	This is the video sample-rate clock enable. It is the same as <code>vid_ce</code> , except in the case of 12-bit video samples.
<code>aud_ce</code>	In	This is the audio clock enable.
<code>aud_mux_en</code>	In	This input enables insertion of audio data packets: 0 = Disable 1 = Enable
<code>contr_pkt_en</code>	In	This signal enables insertion of audio control packets: 0 = Disable 1 = Enable

Table 20-2: Inputs and Outputs of `hd_audio_mux_top` Module (Cont'd)

Signal	Direction	Description
<code>aud_group[1:0]</code>	In	This is the audio group number: 0 = Group 1 1 = Group 2, etc.
<code>chan_valid[3:0]</code>	In	These are the channel valid bits in unary format. These bits denote the valid audio channels of the group.
<code>sample_rate[2:0]</code>	In	This is the audio sample rate: 000 = 48 KHz 001 = 44.1 KHz 010 = 32 KHz
<code>asx</code>	In	This control packet value is a synchronous data flag: 0 = Synchronous 1 = Asynchronous
<code>delay_1_2[25:0]</code>	In	This control packet value is the audio processing delay for channels 1 and 2.
<code>delay_1_2_val</code>	In	This control packet value is the valid flag for the <code>delay_1_2</code> audio processing delay.
<code>delay_3_4[25:0]</code>	In	This control packet value is the audio processing delay for channels 3 and 4.
<code>delay_3_4_val</code>	In	This control packet value is the valid flag for the <code>delay_3_4</code> audio processing delay.
<code>line_std[3:0]</code>	In	This output port indicates the video format. These bits are encoded as: 0000 = SMPTE 260M 1035i 30 Hz 0001 = SMPTE 295M 1080i 25 Hz 0010 = SMPTE 274M 1080i or 1080sF 30 Hz 0011 = SMPTE 274M 1080i or 1080sF 25 Hz 0100 = SMPTE 274M 1080p 30 Hz 0101 = SMPTE 274M 1080p 25 Hz 0110 = SMPTE 274M 1080p 24 Hz 0111 = SMPTE 296M 720p 60 Hz 1000 = SMPTE 274M 1080sF 24 Hz 1001 = SMPTE 296M 720p 50 Hz 1010 = SMPTE 296M 720p 30 Hz 1011 = SMPTE 296M 720p 25 Hz 1100 = SMPTE 296M 720p 24 Hz
<code>vid_y_in[9:0]</code>	In	This is the Y input video stream.
<code>vid_c_in[9:0]</code>	In	This is the $C_B C_R$ input video stream.
<code>ovw_en</code>	In	This is the overwrite enable: 0 = Marks incoming packets in the selected group for deletion 1 = Overwrites incoming data and control packets in the selected group
<code>vid_y_out[9:0]</code>	Out	This is the Y output video stream.

Table 20-2: Inputs and Outputs of `hd_audio_mux_top` Module (Cont'd)

Signal	Direction	Description
vid_c_out[9:0]	Out	This is the C _B C _R output video stream.
vid_c_out_dup[9:0]	Out	This is an identical copy of vid_c_out.
aud_valid	In	This is the audio valid strobe. Logic High indicates that valid audio data is present on the inputs. The aud_valid signal applies to all four channels.
ch1_audio[23:0]	In	Channel 1 audio sample data.
ch1_z	In	Channel 1 Z flag.
ch1_c	In	Channel 1 channel status data.
ch1_u	In	Channel 1 user data bit.
ch1_v	In	Channel 1 valid bit.
ch2_audio[23:0]	In	Channel 2 audio sample data.
ch2_c	In	Channel 2 channel status data.
ch2_u	In	Channel 2 user data bit.
ch2_v	In	Channel 2 valid bit.
ch3_audio[23:0]	In	Channel 3 audio sample data.
ch3_z	In	Channel 3 Z flag.
ch3_c	In	Channel 3 channel status data.
ch3_u	In	Channel 3 user data bit.
ch3_v	In	Channel 3 valid bit.
ch4_audio[23:0]	In	Channel 4 audio sample data.
ch4_c	In	Channel 4 channel status data.
ch4_u	In	Channel 4 user data bit.
ch4_v	In	Channel 4 valid bit.

Modules

The reference design contains 10 modules. These are listed in Table 20-3 and are described in more detail in the remainder of this section.

Table 20-3: List of Modules

Module	Description
hd_audio_mux_top	This is a top-level wrapper for the audio multiplexer.
hd_audio_mux_data	This is a multiplexer for audio data packets on the C video stream.
jd_audio_mux_contr	This is a multiplexer for audio control packets on the Y video stream.
anc_det	This module detects the presence, type, and location of ancillary data packets. It also controls the deletion of packets.

Table 20-3: List of Modules (Cont'd)

Module	Description
packet_mux	This module modifies the C _B C _R video stream to mark packets for deletion and to insert audio data and control packets.
clock_phase	This module determines clock phase data based on the timing of the audio valid strobe to the line and sample position.
sync_one_shot	This module detects the rising edge of strobes and outputs a one-clock-wide pulse for each rising edge.
aud_fifos	This is a wrapper for instances of fifo_28x16 that stores incoming audio samples prior to embedding.
fifo_28x16	This is a FIFO element, 28 bits wide by 16 locations deep, built from LUTs.
aud_data_constr	This module constructs the audio data packets for insertion into the video stream.
ecc_encode	This module calculates the ECCs for the audio data packets.
aud_contr_constr	This module constructs the audio control packets for insertion into the video stream.

Data Packet Multiplexer for C_BC_R Video Stream (hd_audio_mux_data)

The hd_audio_mux_data module is the top level for multiplexing audio data packets onto the C_BC_R video stream. A block diagram of this module is shown in the top half of Figure 20-3. This module instantiates and connects the lower-level modules shown.

Data Packet Multiplexer for Y Video Stream (hd_audio_mux_contr)

The hd_audio_mux_contr module is also the top level for multiplexing audio control packets onto the Y video stream. A block diagram of this module is shown in the bottom half of Figure 20-3. This module instantiates and connects the lower-level modules shown.

Ancillary Data Parsing (anc_det)

This module detects ancillary data, checks for audio packets, and determines where the audio packets can be inserted. If audio packets for the designated group are already present in the video stream, they are overwritten with new packets if overwrite is enabled (ovw_en = 1). If overwrite is disabled, the existing packets are marked for deletion. If there are more existing packets than new packets to overwrite them, the extra packets are also marked for deletion.

Ancillary data packets must be contiguous. If there are more ancillary data packets than new packets to overwrite them or there are more packets than in the incoming video, new audio packets can be appended immediately after the end of all existing packets. If no ancillary packets are present in the incoming stream, new packets are appended immediately after the EAV. In addition, it is possible that an end-marker packet is present at the end of all existing ancillary packets (per SMPTE 291M). If additional packets are to be inserted, the end-marker packet is overwritten with new packets or marked for deletion before new packets are appended.

Because all channels in a given group must be synchronous, there is no practical way to add or replace audio samples in an existing packet. To add a new channel into an existing

group, the audio from that group must be demultiplexed, synchronized, and input to this module with the other channels in the group. The original audio packets are then replaced with the new packets created for that group.

Figure 20-4 shows the main elements of ancillary data parsing. A state machine detects the EAV, SAV, and ancillary packets. It looks at the DID to determine which, if any, packets must be overwritten or marked for deletion. The word counter counts words for multi-word fields. The HANC counter determines the size of the horizontal ancillary space, the current position within the HANC space, and whether or not there is enough room to insert another packet. The number of data packets inserted on a line is also limited by the maximum number of packets per line. This is a function of the line standard and sample rate and enforced by the `aud_data_constr` module. This module deasserts the Data Packet Pending signal when the maximum number of samples per line has been reached.

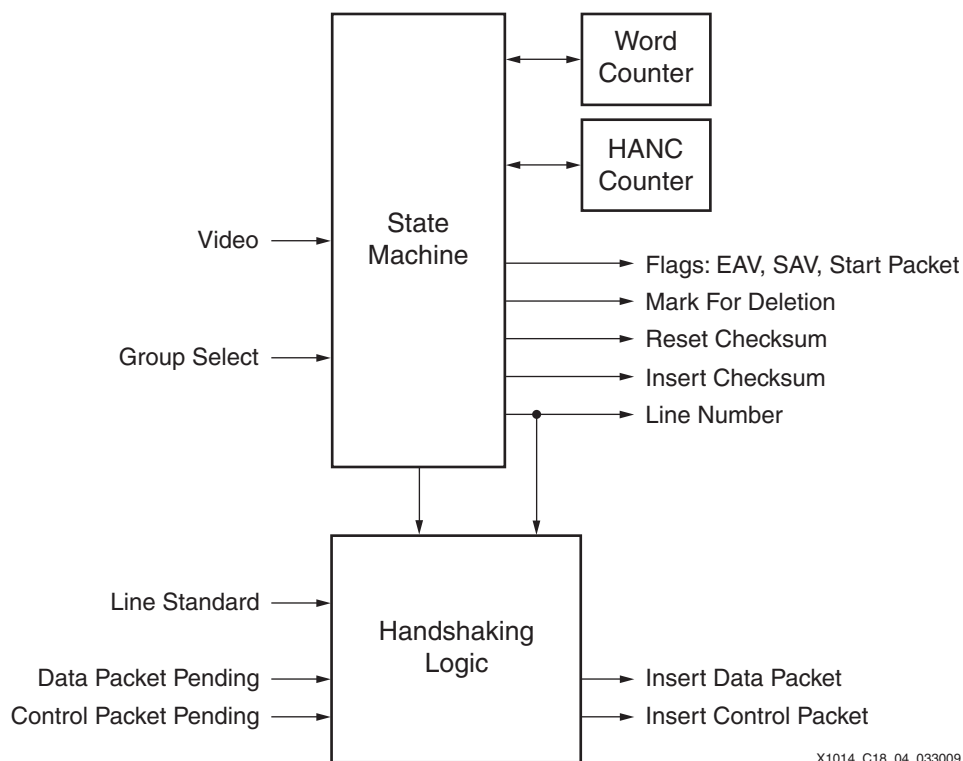
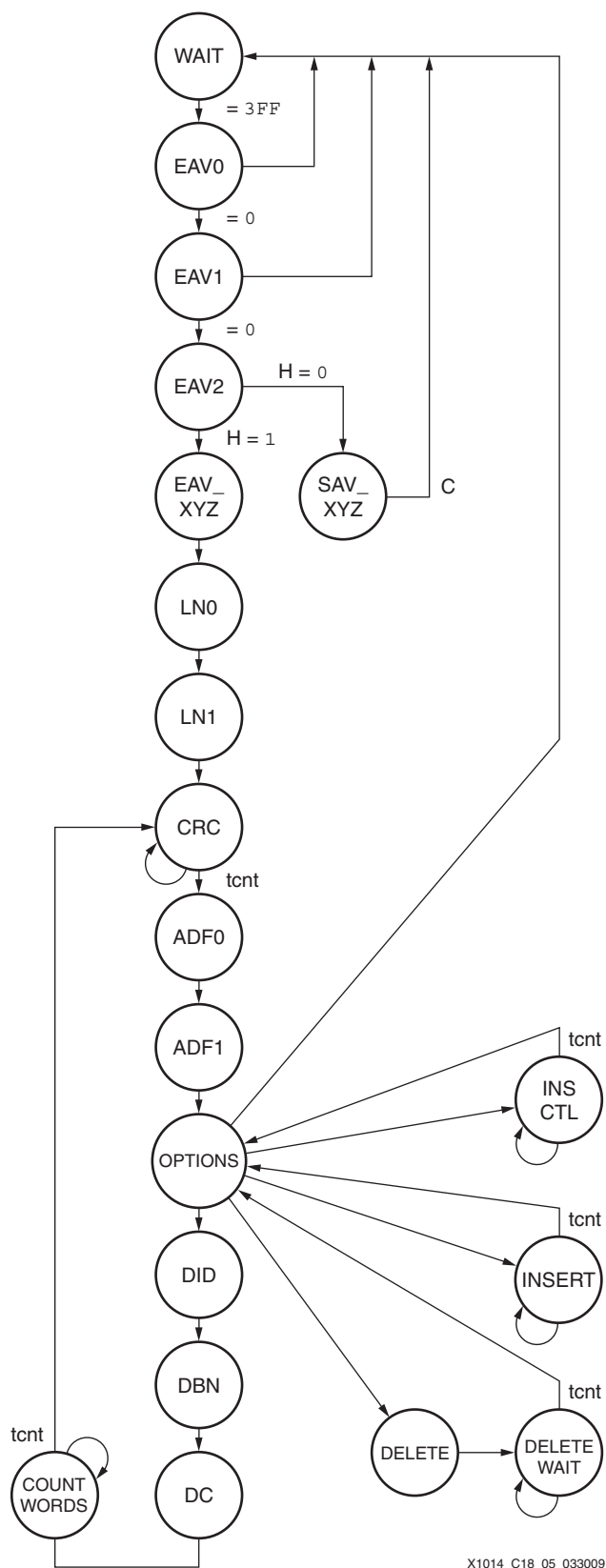


Figure 20-4: Block Diagram of `anc_det` Module

The `anc_det` module controls when the `packet_mux` module inserts packets, marks packets for deletion, or both. The state machine also indicates when to start the checksum operation and where to insert the checksum. It extracts the line number that follows each EAV.

The state diagram for this state machine is shown in Figure 20-5. The states correspond to fields in the timing reference sequences and ancillary data packets. Unless otherwise noted, the values denoted by equal signs are the values of the video data required for the transition to the associated state. The `tcnt` signal referenced is the terminal count signal from the word counter. The states roughly correspond to timing references of SMPTE 292 (EAV and SAV) and to the fields of ancillary data defined by SMPTE 291. In the Options state, the decision is made to insert, overwrite, delete, or skip a packet.



X1014_C18_05_033009

Figure 20-5: Ancillary Data Parser State Diagram

The handshaking logic receives timing and line number information from the state machine. It also uses the video line standard to determine which lines are valid for inserting data and control packets. The packet producing modules for data and control packets (`aud_data_constr` and `aud_contr_constr`) send requests in the form of Data Packet Pending and Control Packet Pending signals. At the appropriate point in the appropriate line, the handshaking logic asserts the Insert Data Packet signal or the Insert Control Packet signal to tell the packet producing modules and the data multiplexing module to insert packets into the video stream. This decision is based on a number of factors, such as whether an ancillary packet is already embedded in the video stream, what the packet is, whether new packets are available, what the current video line is, and how many packets have already been written on the current line.

Audio Packet Multiplexer (`packet_mux`)

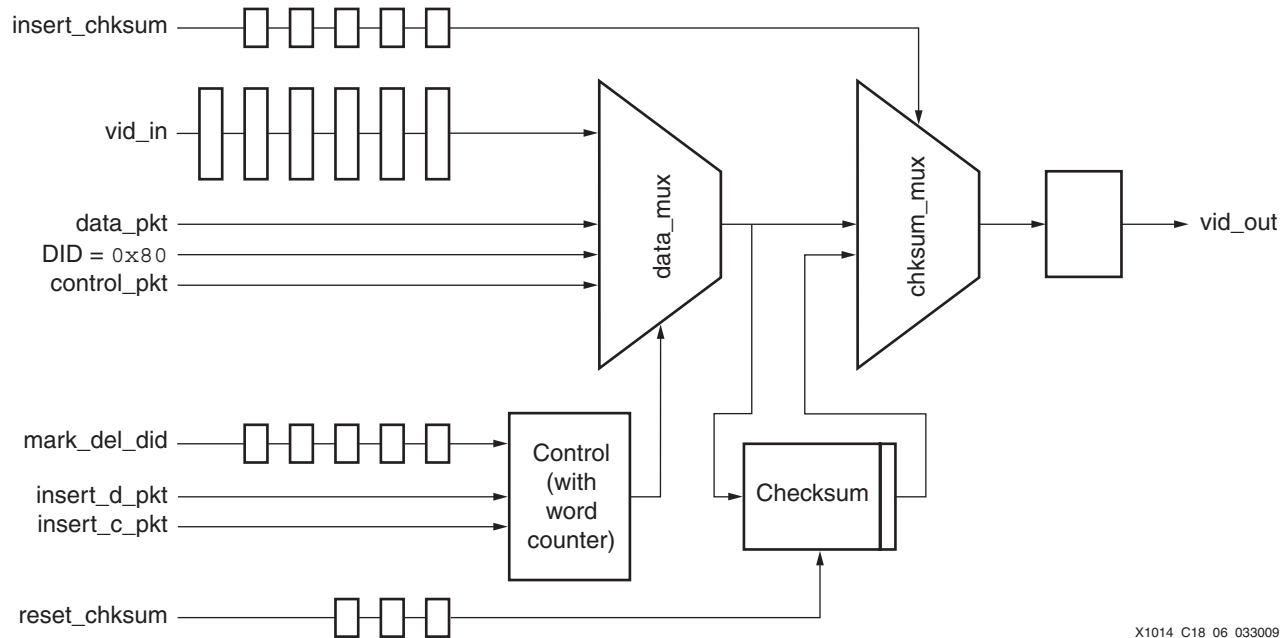
The `packet_mux` module has these functions:

- Controls the embedding of audio data and control packets into the video data stream
- Receives control information on when to append packets from the Ancillary Data Parsing module
- Receives requests from and issues acknowledges to the modules that construct data and control packets
- Monitors the line numbers and determines the appropriate times to embed the various packets

The `packet_mux` module also has a word counter and a checksum adder. Thus, the checksum of each packet is inserted as the last word for data and control packets and packets marked for deletion.

Inputs are delayed by various amounts to match the delays of packet information coming from various sources. Fundamentally, the proper place for packet insertion/deletion can only be determined several words into the packet. The data stream must be delayed until this determination can be made, and this information is received by the packet producing modules.

For packets marked for deletion, when overwrite is disabled, the `packet_mux` module replaces the DID of the packet with a value of `0x80` (meaning that it is marked for deletion) and calculates a new checksum for the packet. [Figure 20-6](#) shows a block diagram of the `packet_mux` module.



X1014_C18_06_033009

Figure 20-6: Block Diagram of `packet_mux` Module

Clock Phase Determination (`clock_phase`)

The `clock_phase` module generates clock phase information for each audio sample. Figure 20-7 shows a block diagram of this module. The `clock_phase` module uses received EAV and video clock information to determine when the audio samples arrive relative to the video lines. This information is sent to the `aud_fifos` module with each audio sample for inclusion in the audio data packets. The line number is received from the `anc_det` module. The sample number is determined by a video sample counter that is reset each time the line changes, as shown in Figure 20-8. The clock counter increments for each video sample. The audio strobe might be completely asynchronous with the video clock. Therefore, the strobe is detected and synchronized with an instance of the synchronizing one-shot module/entity. The synchronized audio strobe clocks in the current video sample count. This count, along with the line number from the clock phase information, is stored in the clock phase FIFO.

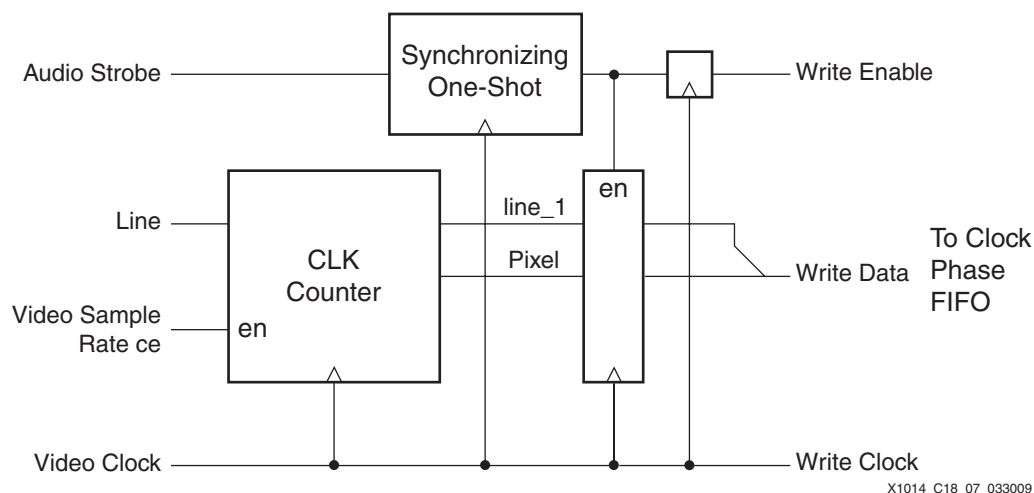
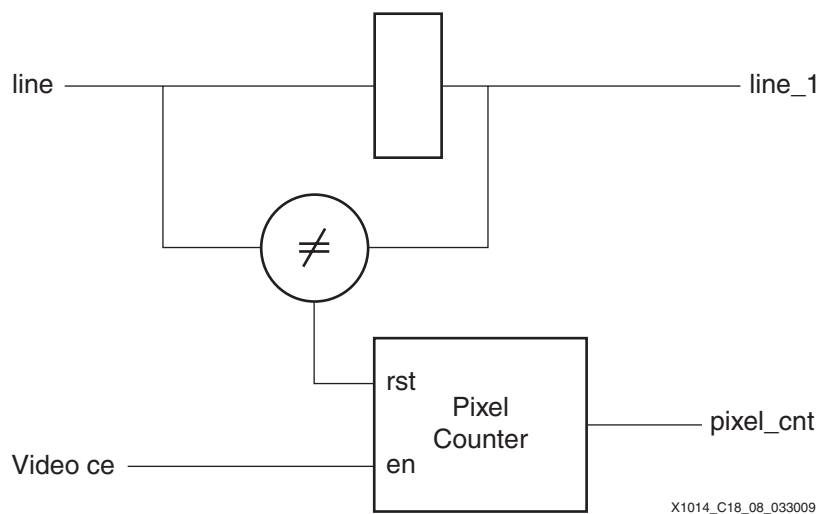
Figure 20-7: Block Diagram of `c1ock_phase` Module

Figure 20-8: Clock Counter

Synchronizing One-Shot (`sync_one_shot`)

This small circuit detects a pulse, synchronizes it to another clock, identifies the rising edge, and outputs a one-cycle-wide pulse in the new clock domain. Figure 20-9 shows the details of the synchronizing one-shot circuit. It can accommodate an extremely wide range of pulse widths on the input. The minimum width of the pulse is just long enough to be recognized as a High level by the SR latch. There is no maximum width. However, the input must remain Low for at least four clock cycles for the next rising edge to be recognized as a distinct pulse.

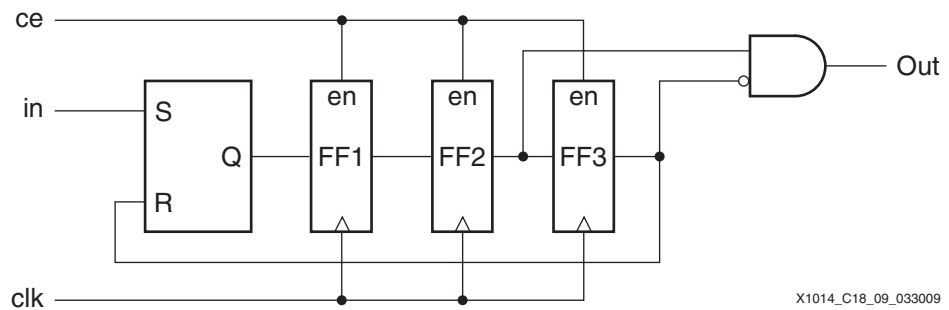


Figure 20-9: Synchronizing One-Shot

Figure 20-10 illustrates the operation of the synchronizing one-shot for both short and long input pulses. Regardless of the length of the input pulse, the output is a one-clock-wide pulse, synchronized to the clock, and occurring near the rising edge of the input pulse.

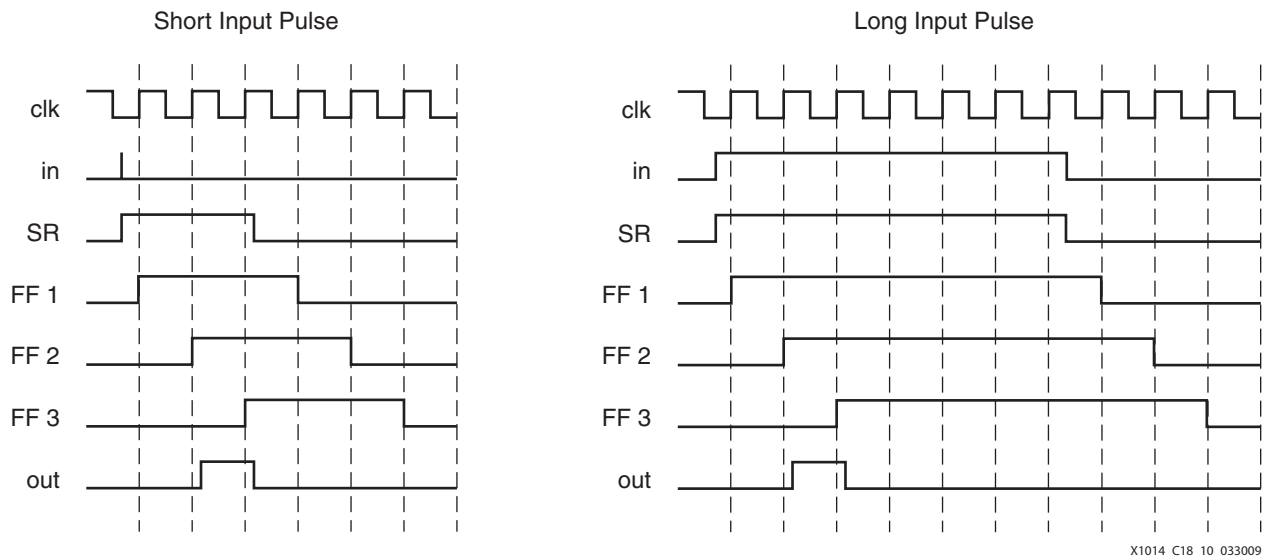
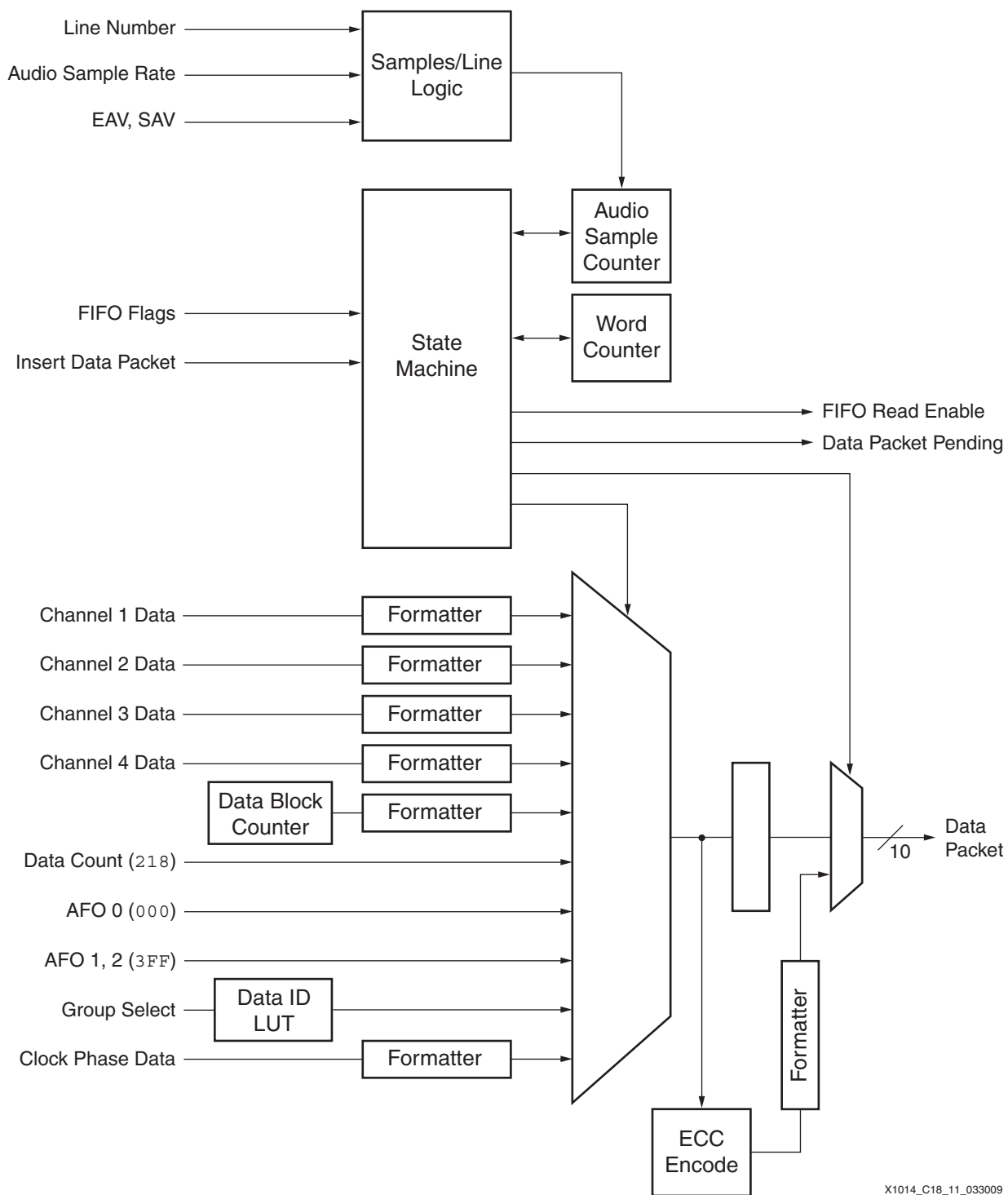
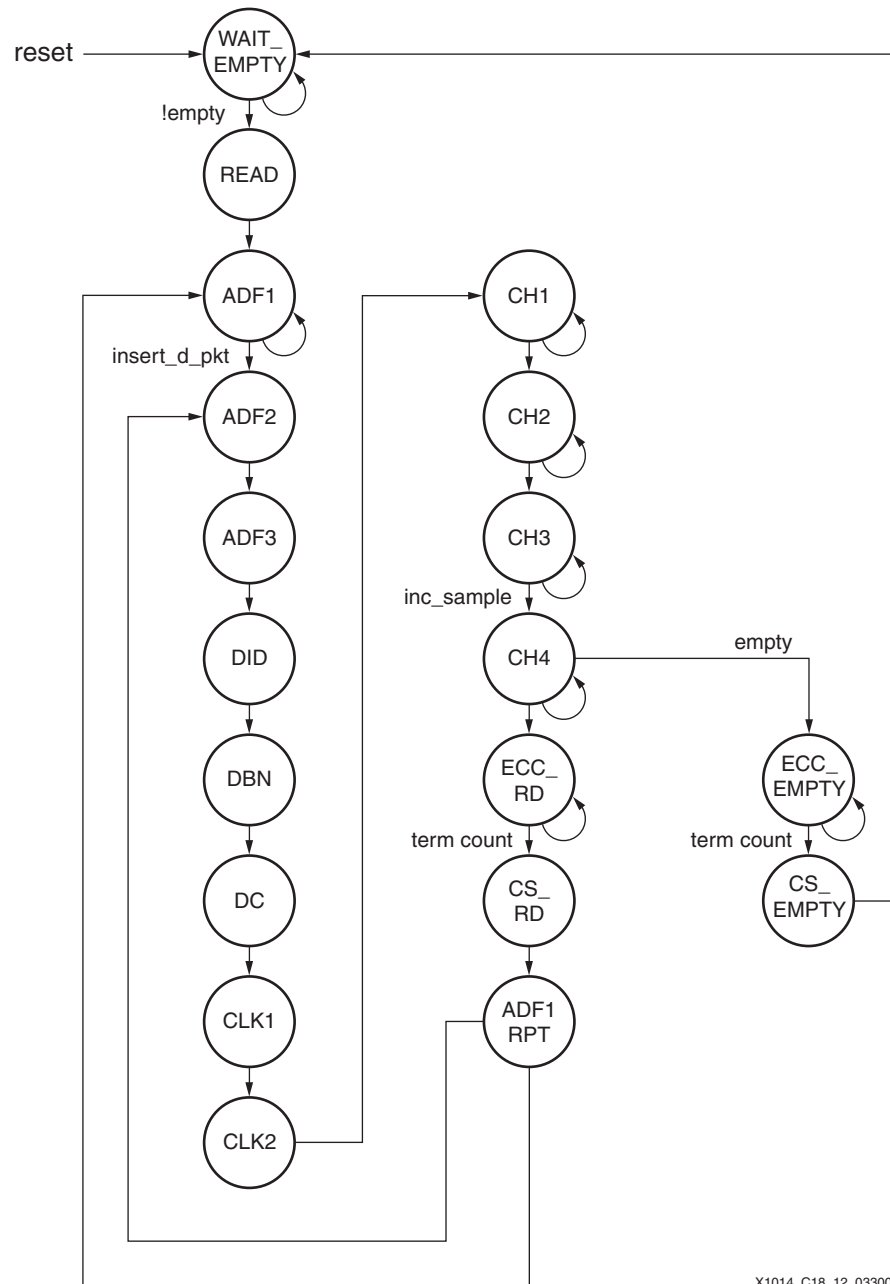


Figure 20-10: Synchronizing One-Shot Timing (ce = 1)

Audio Data Packet Construct (aud_data_constr)

This module constructs the audio packet. It consists of a state machine to control the timing and multiplexers to place the elements of the packet in sequence to be multiplexed into the video. Figure 20-11 shows a block diagram of this module. The state machine also controls handshaking with the ancillary data parser and the reading of the aud_fifos module. Figure 20-12 shows a state diagram for the state machine. The next state for conditional branches is determined by the state of the Insert Data Packet handshaking signal, and whether or not the FIFOs are empty and the word counter has reached the terminal count.

Figure 20-11: Block Diagram of `aud_data_constr` Module



X1014_C18_12_033009

Figure 20-12: State Diagram for Data Packet Construction

The clock phase information and sample data for each channel are inputs that get formatted for the data packet. Several constant fields are part of the packet, such as data count and ADF values. Group select is a static value used to look up the DID. A data block counter increments for each new packet, and the count is inserted in the DBN field. ECCs are generated simultaneously by the `ecc_encode` module described in [Error Correction Encode \(ecc_encode\)](#), page 472.

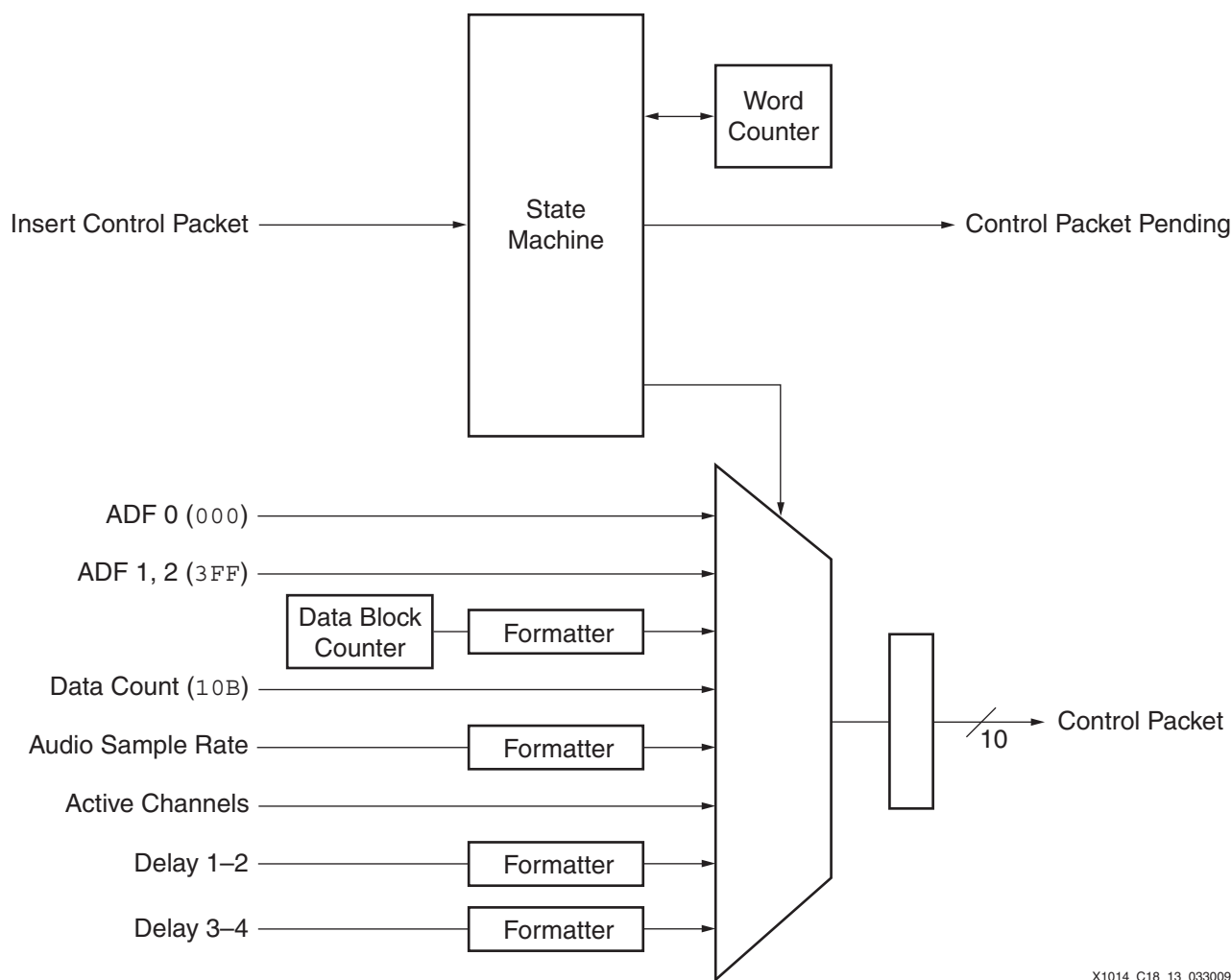
Two counters are controlled by the state machine: the audio sample counter and the word counter. The audio sample counter counts the number of audio samples that are inserted into each video line. The flags from the audio sample FIFOs are sent to the Ancillary Data Parsing module, so that the number of audio packets on each line can be adjusted to not

exceed the number of audio samples per line as determined by the line standard and audio sample rate. The word counter counts words for multi-word fields of the packet, such as the audio samples. This count is also used to control formatting of multi-word fields.

Audio Control Packet Construct (`aud_contr_constr`)

The `aud_contr_constr` module forms audio control packets to append to ancillary data. This module operates in a manner similar to the Audio Data Packet Construct module. It consists of a state machine to control the timing and multiplexers to place the elements of the packet in sequence to be multiplexed into the video. A block diagram of the `aud_contr_constr` module is shown in Figure 20-13. The state machine also controls handshaking with the ancillary data parser. Figure 20-14 is a state diagram for the state machine.

Most packet information is received from inputs such as the audio sample rate, the active channels, and the delay of the channels with respect to the video. Several constant fields are part of the packet, such as data count and ADF values. A data block counter increments for each new packet and the count is inserted in the DBN field. A word counter controlled by the state machine counts words for multi-word fields of the packet, such as delay.



X1014_C18_13_033009

Figure 20-13: Block Diagram of `aud_contr_constr` Module

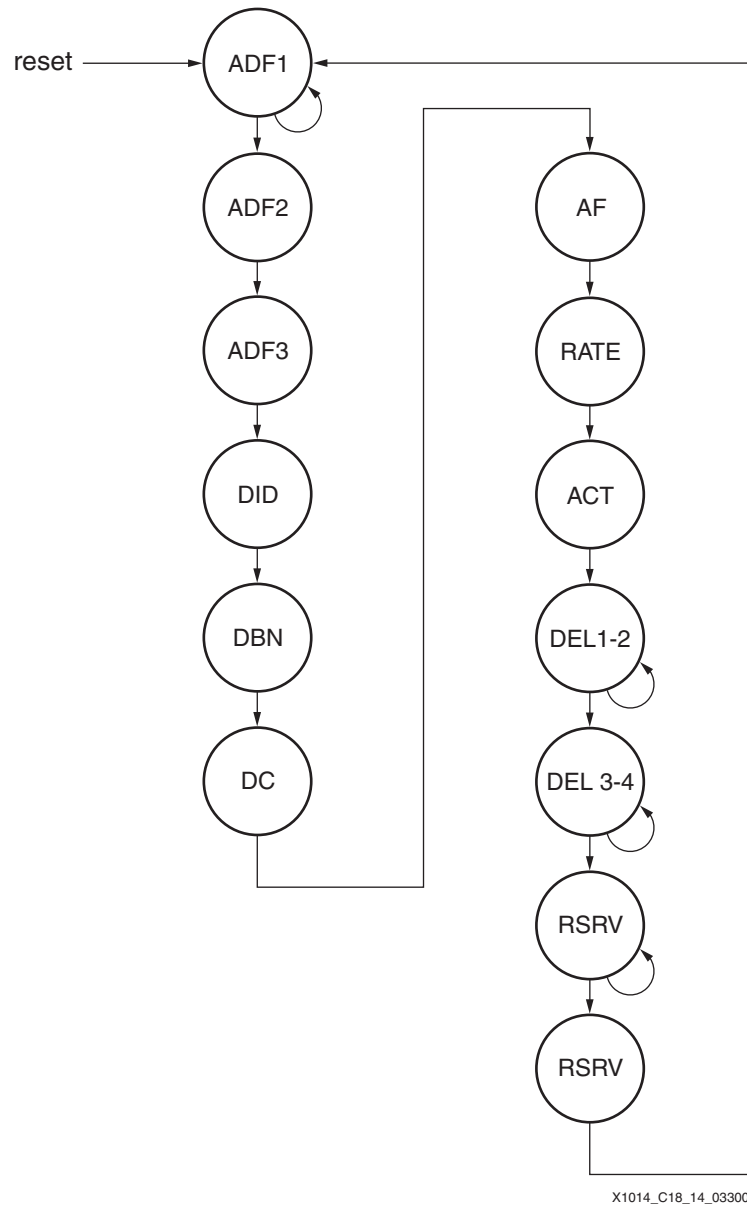
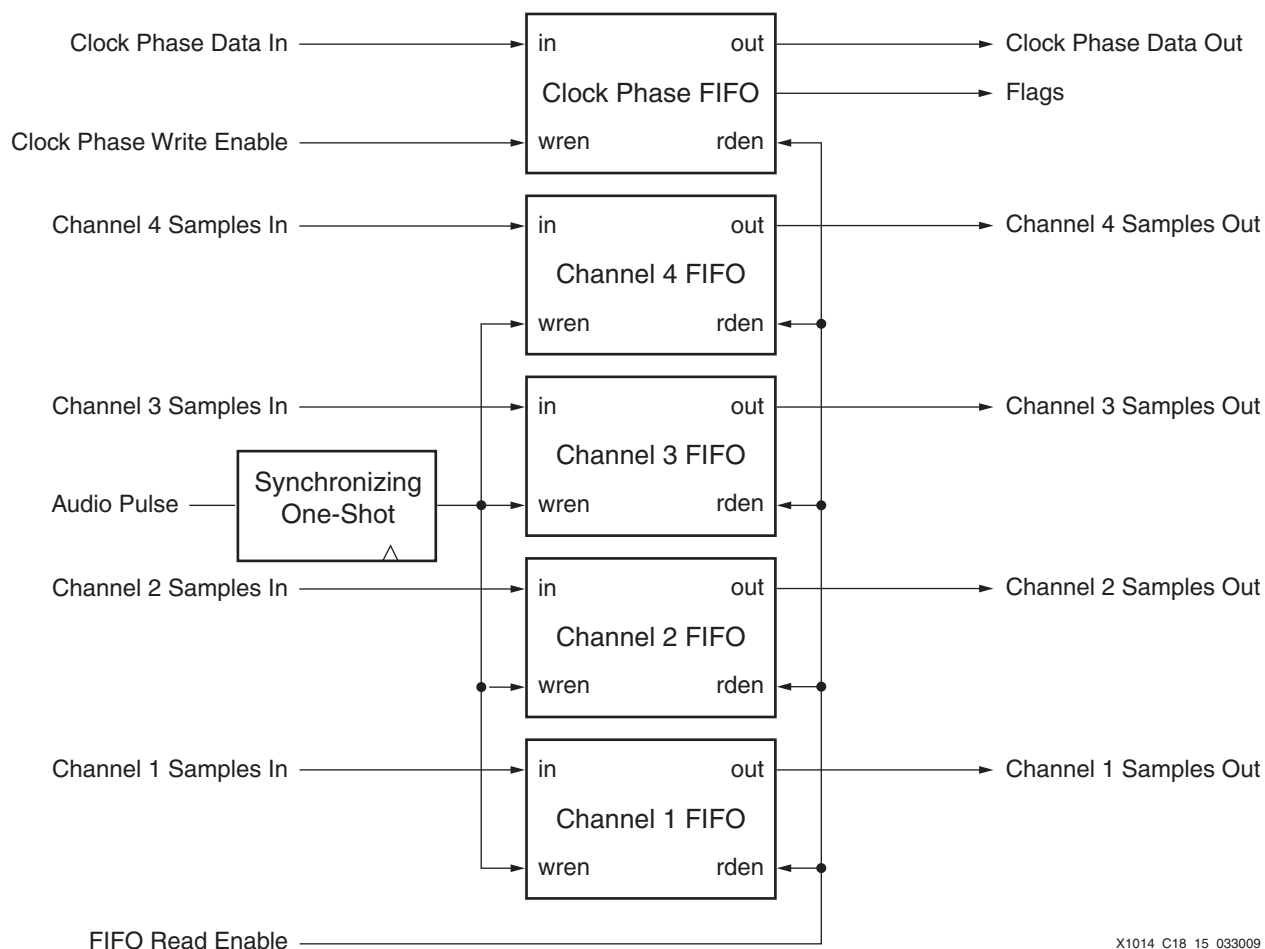


Figure 20-14: State Diagram for Control Packet Construction

Audio FIFOs (aud_fifos)

The `aud_fifos` module acts as a small buffer between the incoming sample stream, in which the samples are evenly spaced in time, and the Audio Data Packet Construct module, which retrieves samples at an irregular rate based on the samples to be embedded on a video line. The `aud_fifos` module contains five instances of the FIFO module `fifo_28x16`, one for each audio channel of the group and one for the clock phase information. A block diagram of the `aud_fifos` module is shown in Figure 20-15. Reads and writes are done in parallel to all FIFOs, with the exception of writes to the Clock Phase FIFOs.



X1014_C18_15_033009

Figure 20-15: Block Diagram of `aud_fifo` Module

FIFO (`fifo_28x16`)

This basic FIFO module consists of SelectRAM™ memory built from LUTs. It is 28 bits wide by 16 locations deep. Each FIFO stores either video samples or clock phase information. In video sample FIFOs, each location contains one 24-bit video sample and the associated Z, C, U, and V flags. For the clock phase FIFO, each location contains the clock phase information associated with the corresponding audio samples.

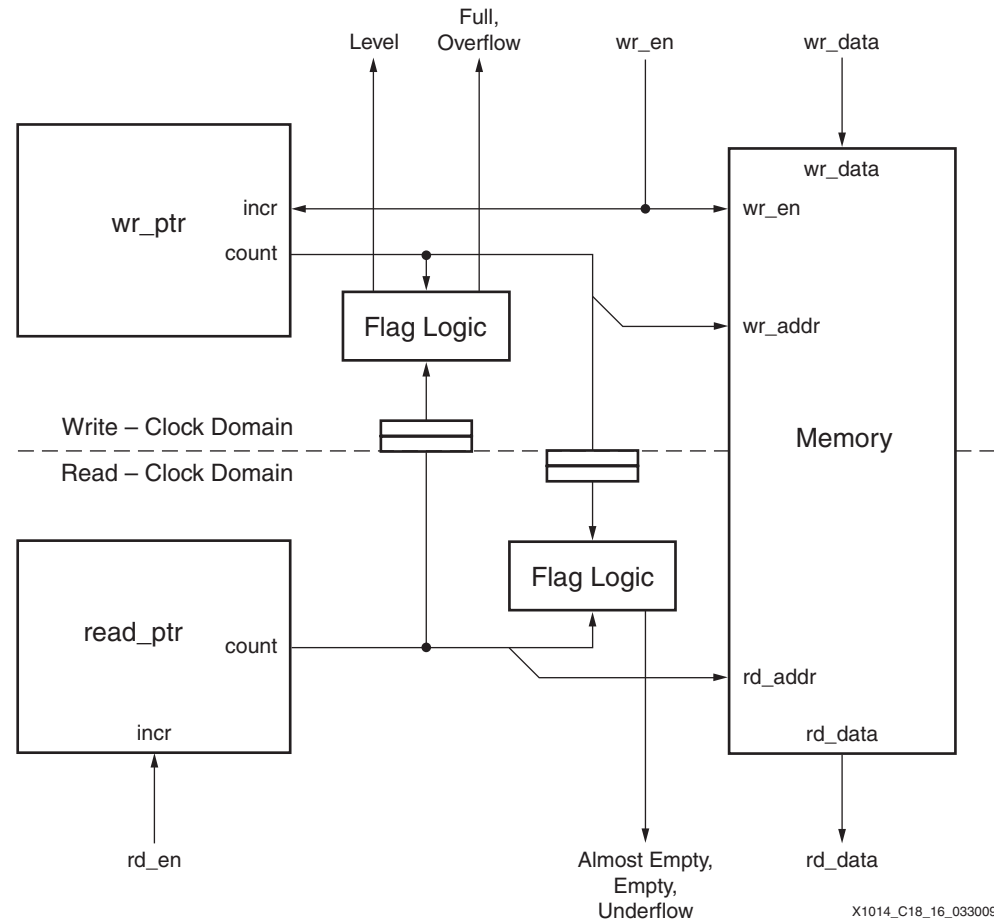
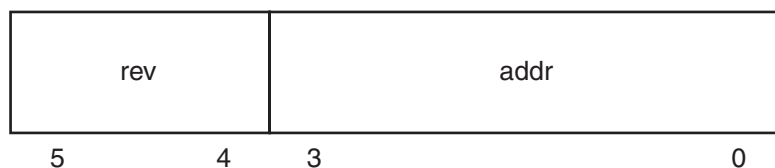


Figure 20-16: FIFO Block Diagram

The control logic is built specifically for use in the reference design. In particular, the control logic protects against underflow by not allowing the read pointer to increment when the FIFO is empty. At start-up, it is likely for read requests to occur when the FIFO is empty. Due to an inherent latency in flag logic crossing clock domains, these read requests can happen before the empty flag has propagated to the controlling logic for the read operation. In such cases, the read operation, specifically the moving of the read pointer, is inhibited. This is important because there are occasions in which the FIFO must be emptied fully, but must also be guaranteed to work properly without rolling over when the next valid write data arrives.

As shown in Figure 20-17, the read and write pointers have two extra MSBs beyond the address. These indicate the number of passes or revolutions through the memory address space. In this way, when the read and write pointers point to the same address, it can be unambiguously determined whether this represents a full condition (write revolution does not equal read revolution) or an empty condition (write revolution equals read revolution).

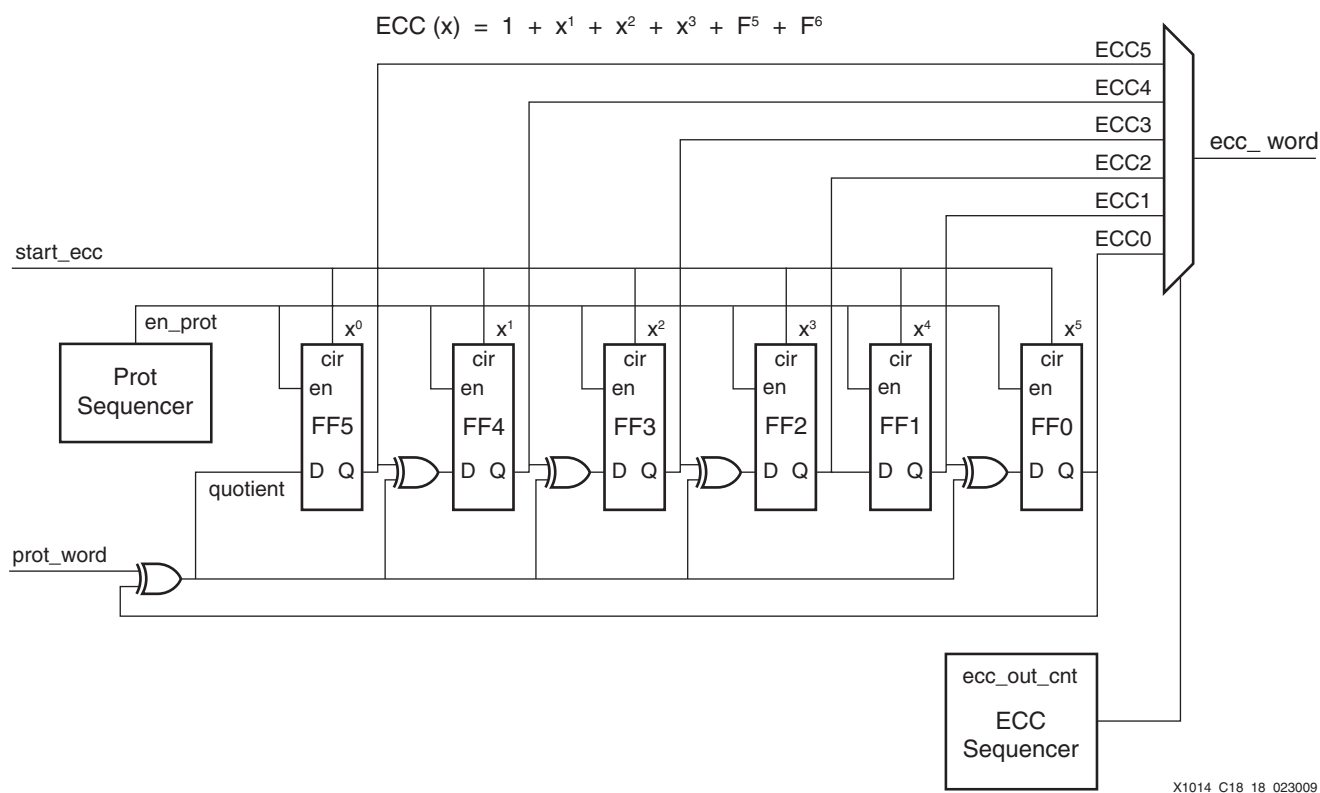


X1014_C18_17_033009

Figure 20-17: FIFO Pointer Format

Error Correction Encode (ecc_encode)

The `ecc_encode` module, referenced by the Audio Data Packet Construct module, creates the ECCs based on the Bose-Chaudhuri-Hocquenghem (BCH) 31, 25 code specified in SMPTE 299M. For error correction of embedded audio, retransmission is not practical. Therefore, forward error correction is used in which redundant data is sent, making it possible for the receiving processor to correct errors in the corrupted data without retransmission. The ECCs cover all the preceding words of the packet, as shown in Figure 20-18.



X1014_C18_18_023009

Figure 20-18: ECC Protection

The ECC bits are calculated as the packet data passes word by word through the `ecc_encode` module by using the circuit given in SMPTE 299M. Figure 20-19 is a diagram of the ECC generation. There are eight copies of this circuit, one for each bit of the UDWs, resulting in eight 6-bit ECC words. These are formatted into six 8-bit data words in the audio data packet.

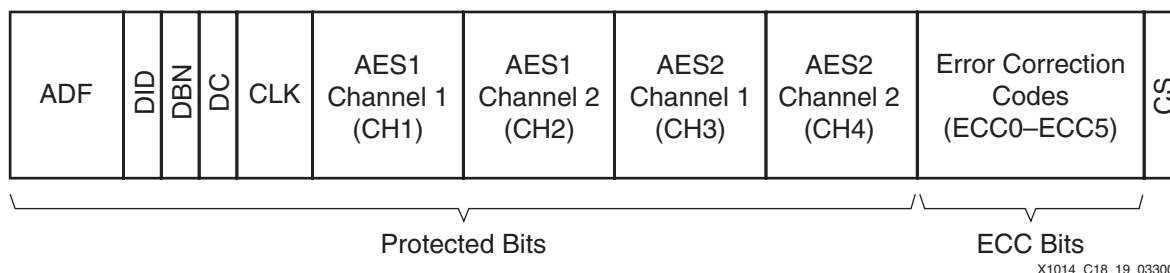


Figure 20-19: ECC Generation

Usage Models

This section provides some examples of ways in which the HD-SDI audio multiplexer can be used.

Embedding Audio from a Synchronous Audio Source

If the audio source is synchronous with the video, audio can be embedded directly in a synchronous fashion, as shown in [Figure 20-20](#).

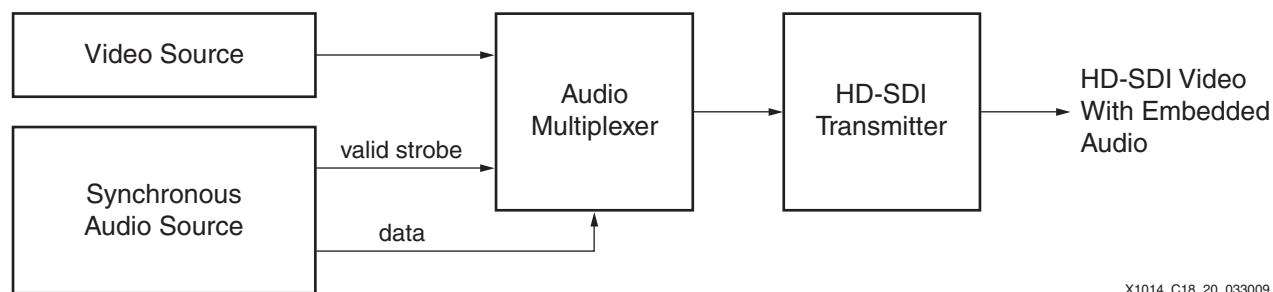
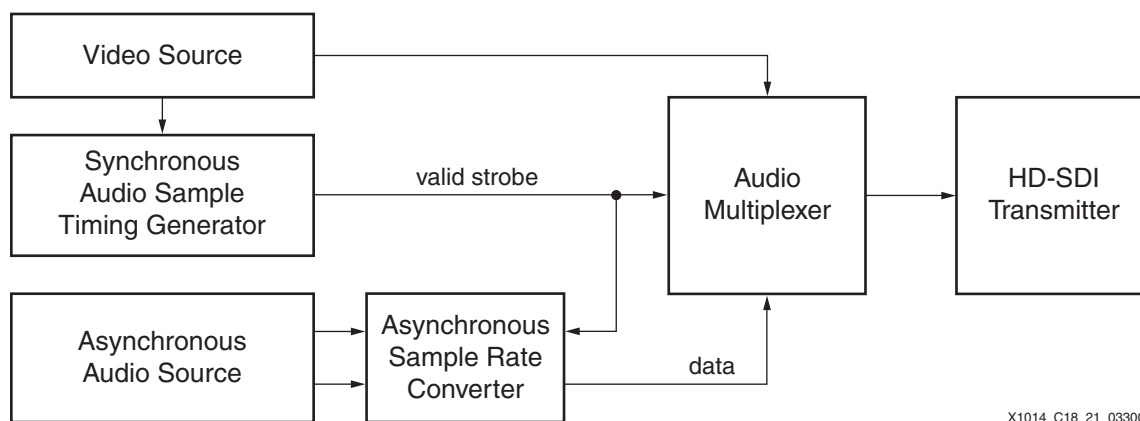


Figure 20-20: Embedding Synchronous Audio

Synchronizing Audio via Sample Rate Conversion

If synchronous audio is desired but the audio source is asynchronous with the video, it can be synchronized via sample rate conversion. An audio timing strobe synchronized to the video is required to act as a reference to the output side of the sample rate converter and to be used by the audio multiplexer. This is shown in [Figure 20-21](#).

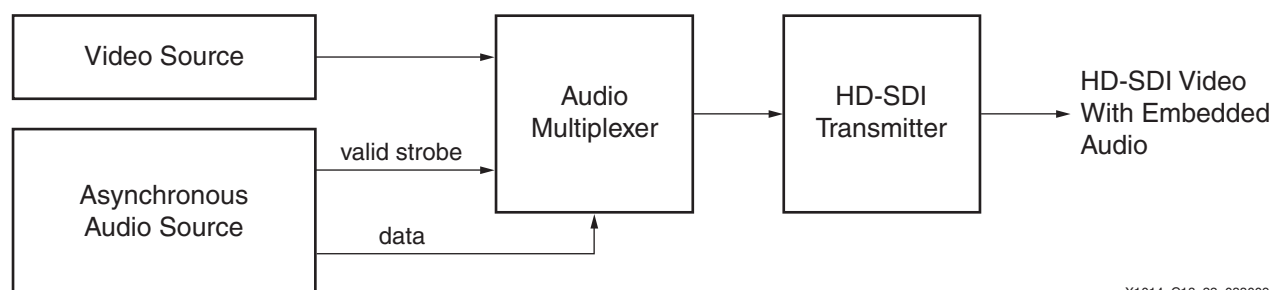


X1014_C18_21_033009

Figure 20-21: Synchronizing Audio with Sample Rate Conversion

Embedding Asynchronous Audio

The audio multiplexer supports embedding of asynchronous audio as allowed in SMPTE 299M (Figure 20-22). Audio sample timing is denoted by the clock phase information.

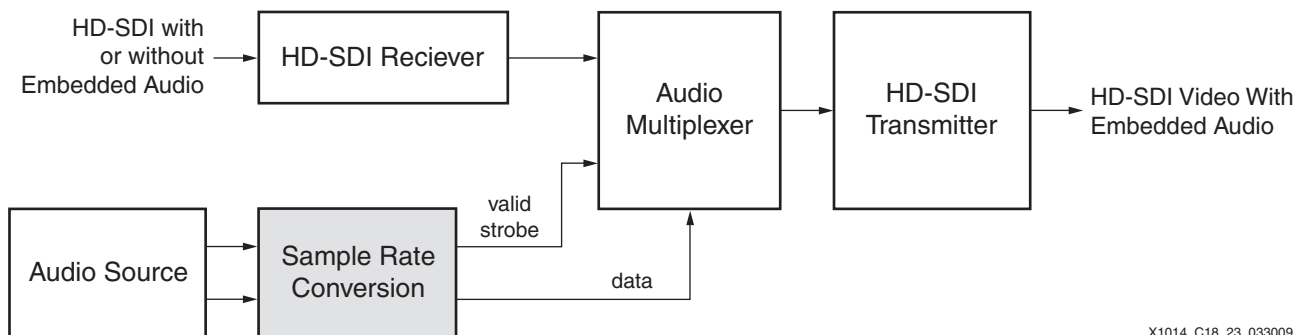


X1014_C18_22_033009

Figure 20-22: Embedding Asynchronous Audio

Replacing Audio Embedded in Received Video

Audio embedded in video received from a remote source can be replaced from an alternate audio source (Figure 20-23). The audio group specified for the audio multiplexer has all existing audio packets marked for deletion. New audio packets for the audio source are embedded in the specified audio group. All other ancillary data in the received video is passed unaltered. The audio can optionally be passed through a sample rate converter.

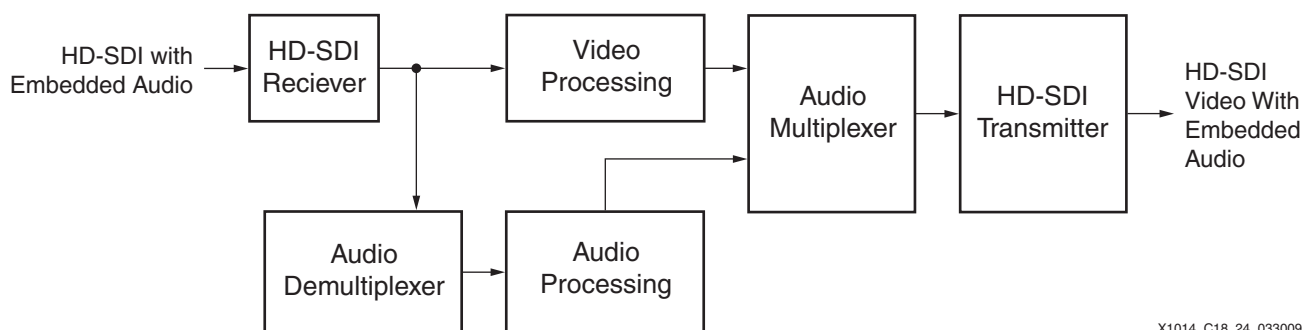


X1014_C18_23_033009

Figure 20-23: Replacing Embedded Audio

Reembedding Audio after Separate Video/Audio Processing

In some applications, audio and video should be processed separately from one another (Figure 20-24). For example, a video frame synchronizer drops complete frames of video. However, the audio cannot tolerate the equivalent dropping of samples, and so must be processed separately. In such cases, the audio is de-embedded from the video and processed in a separate path. After processing is completed, the audio and video can be combined again in the audio multiplexer.



X1014_C18_24_033009

Figure 20-24: Separate Audio and Video Processing

FPGA Resources

The reference design is implemented and hardware verified in a Virtex-5 device but it can also be used with other Xilinx FPGA families. The resources required for the reference design in a Virtex-5 device are shown in Table 20-4. These results were obtained in ISE® software, version 9.2i SP2. Timing was constrained for a processing clock of 148.5 MHz. Timing was met for -1, -2, and -3 speed grade devices.

Table 20-4: FPGA Resources

Flip-Flops	LUTs	Block RAMs	DSP Blocks
652	889	0	0

Design Files

The reference design for the HD-SDI audio demultiplexer is available at <https://secure.xilinx.com/webreg/clickthrough.do?cid=119758>. Open the ZIP archive and extract the file `xapp1014_c20_hd_audio_mux.zip`.

Conclusion

This chapter explains embedded audio in HD-SDI, the standards used to define it, and the data formats of the audio packets. The reference design of this chapter is useful for embedding audio into HD-SDI video. It can be used in a variety of applications and implemented in all Xilinx FPGA families. The reference design supports multiple HD-SDI line standards and multiple audio sample rates, both synchronous and asynchronous. The modularity of the design supports from one to four audio groups by adding multiple instances in series. Metadata is also encoded in audio control packets and multiplexed into the video stream.

Audio Demultiplexer for HD-SDI Video

Introduction

In many applications, audio is embedded in the video stream to facilitate the transport of the combined audio and video stream. For HD-SDI video, the audio information is carried in the HANC space portion of the video. The specification for this is given in SMPTE 299M.

The audio can be demultiplexed from the video for processing or for output. This chapter describes how this demultiplexing is done and presents a hardware-verified reference design that implements an audio demultiplexer for HD-SDI video.

Features

The audio demultiplexer has these features:

- Single audio group granularity: One instance of the demultiplexer de-embeds the four channels (two channel pairs) of a single audio group. The audio group number is an input to the design. Audio from multiple audio groups can be de-embedded by using multiple instances of the basic design in a daisy-chain fashion.
- Multiple HD-SDI standards are supported, such as SMPTE 274M, SMPTE 295, SMPTE 296M, and SMPTE 260.
- 24-bit audio support at multiple sample rates: Audio is output as 24 bits plus Z, V, U, and C flags with an audio valid strobe to indicate timing. There are four such outputs, one for each channel in the group.
- Synchronous and asynchronous audio support: Audio sample timing can be input directly or, alternately, an audio valid pulse is generated based on embedded clock phase data.
- De-embedding of audio control packets: Output ports contain control packet information.
- Support of many Xilinx® device families: The reference design has been hardware verified on the Virtex®-5 FPGA but uses features common to all Xilinx FPGAs.
- Compatibility with forward error correction: Error correction using embedded ECCs can be done prior to demultiplexing using the reference design of [Chapter 22, Error Correction for HD-SDI Embedded Audio](#).

Embedded Audio

Audio information is embedded in SDI data streams in the form of packets that are placed in the horizontal blanking period of the C_{bC_R} data stream. The general specification for HD-SDI video formats and timing is given in SMPTE 292M. Among other things, this standard specifies how timing references and line numbers are transmitted.

Embedded audio packets are just a few of many different types of ancillary packets that might be inserted in the video stream during blanking periods. SMPTE 291M specifies the general format of ancillary packets (see [Figure 21-1](#)). Common elements include a three-word ancillary data flag (ADF), a data identity (DID) to specify the packet type, a data block number (DBN), and data count (DC) fields. A checksum (CS) is the final field of every ancillary packet.

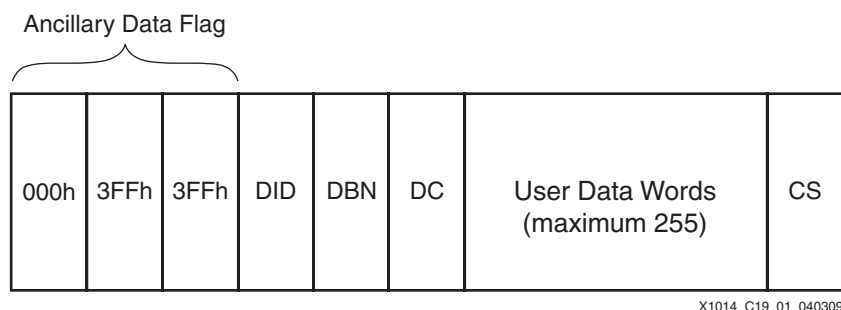


Figure 21-1: Ancillary Data Packet

There are two types of embedded audio packets: audio data packets and audio control packets. Audio data packets contain audio samples for the various channels. Audio control packets contain various metadata fields describing such things as sample rates and audio processing delay. The DID values for embedded audio in HD-SDI are shown in [Table 21-1](#). Each audio data packet contains four samples, one from each channel in a group. Likewise, each audio control packet contains metadata that applies to one audio group.

Table 21-1: DID Values for HD Embedded Audio Packets

Group Number	Audio Data Packets	Audio Control Packets
1	0x2E7	0x1E3
2	0x1E6	0x2E2
3	0x1E5	0x2E1
4	0x2E4	0x1E0

While ancillary data packets are required to be contiguous and begin immediately after the EAV, the exact location of packets of a given type is not specified. Therefore, in order to demultiplex audio from the video stream, all ancillary data types must be examined to determine if the packet is one of interest. This means that the DID field must be checked, and the DC must be read to determine the length of the packet, and thus, where to look for the next packet.

Audio Data Packets

The format of audio data packets is shown in [Figure 21-2](#). These packets have the standard ancillary packet format. The value of DID depends on the audio group, as specified in [Table 21-1](#). DC is always 0x218. The DBN increments for each packet of the same DID. In addition, the audio data packet contains 24 UDWs, a two-word CLK, six ECC words, and four audio words (two stereo pairs). The fields encompassed by the UDWs are described in this section.

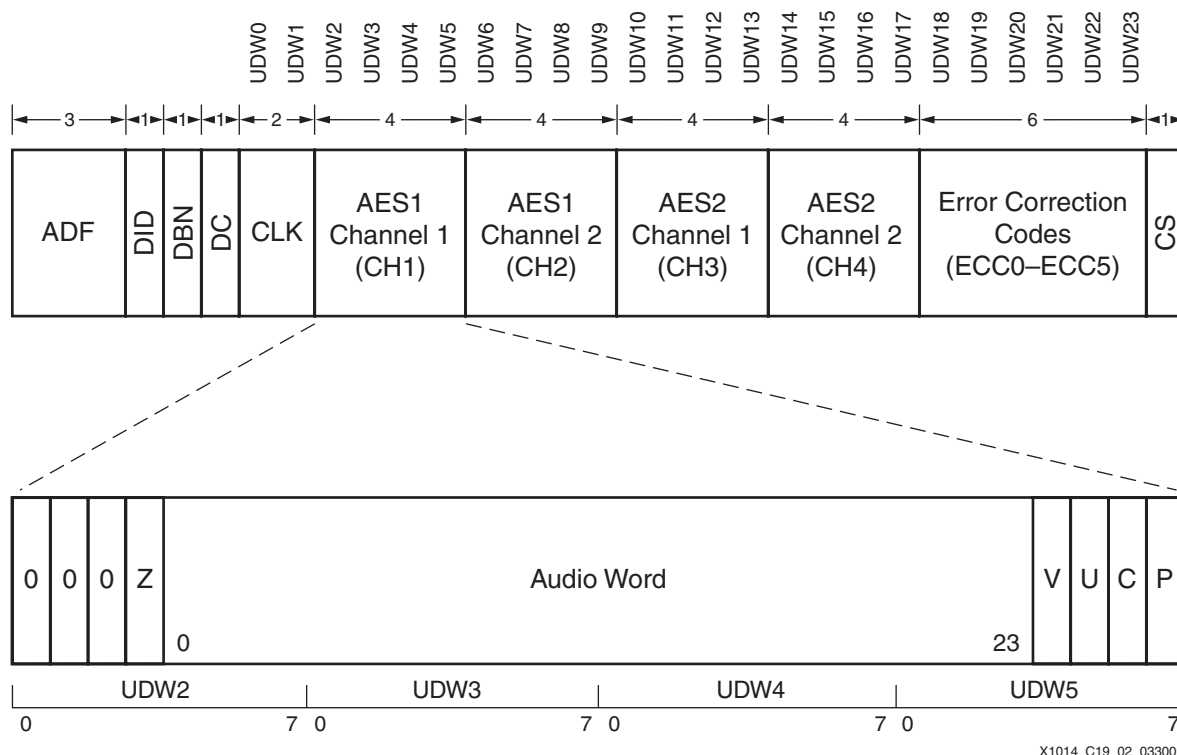


Figure 21-2: Audio Data Packet Format

Audio data packets can be multiplexed onto any video line, with the exception of the lines following the synchronous switching points. The samples must be more or less evenly distributed. The number of audio data packets on any line containing packets can vary by at most one.

Audio Sample Words

The audio words are in AES format. Each AES word (AES_n) includes Z, V, U, C, and P bits. The Z bit is present for CH1 and CH3 only. The audio word itself spans the four UDWs as shown in Figure 21-2. All of the words in the audio data packet are 10 bits wide. The eight LSBs contain the actual data. Bit 8 contains even parity for the eight LSBs. Bit 9 contains the inverse of bit 8.

Audio Clock Phase Data

The CLK field contains information about the timing of the audio clock with respect to video lines. Specifically, the audio clock phase is the number of video clock times (rounded to the nearest integer) between the samples contained in the packet and the EAV immediately preceding the arrival of the audio clock for those samples. Due to the prohibition of packets in the line following the synchronous switching point, the audio sample packet sometimes occurs more than a line time after the EAV to which it is referenced. For these cases, an mpf is asserted. The mpf flag indicates when the clock phase given is from the next-to-last EAV. This implies that there must be at least two video lines of latency in the demultiplexing process.

SMPTE 299M states that valid CLK data is required. However, it is common knowledge that some equipment does not produce the CLK field correctly, and most demultiplexing

equipment completely ignores this field. The reference design described in this chapter provides the option of whether or not to use CLK. Clock phase data is essential for the asynchronous embedded audio that is provided in the SMPTE standard but seldom used. If clock phase data is not used, a synchronous audio sample clock must be derived from the video and provided as the output audio sample clock.

Error Correction Codes

UDWs 18 to 23 are error correction codes for forward error correction of audio data packets. The reference design of this chapter does not support error correction. Refer to [Chapter 22, Error Correction for HD-SDI Embedded Audio](#) for a reference design and detailed explanation of the error correction process. Error correction must be done prior to demultiplexing audio data packets.

Audio Control Packets

Audio control packets are transmitted once per field in an interlaced system or once per frame in a progressive system. They are placed in the blanking period of the second line after the synchronous switching point. The packet contains information about the channels in one group. Thus, if multiple audio groups are used, multiple audio control packets should be sent. Audio control packets have the standard ancillary format with 11 UDWs (see [Figure 21-3](#)). The UDWs are 10 bits wide, with most UDWs having 9 bits of data. Active (ACT) is the only UDW with parity (on bit 8). In all UDWs, the MSB (bit 9) is the complement of bit 8.

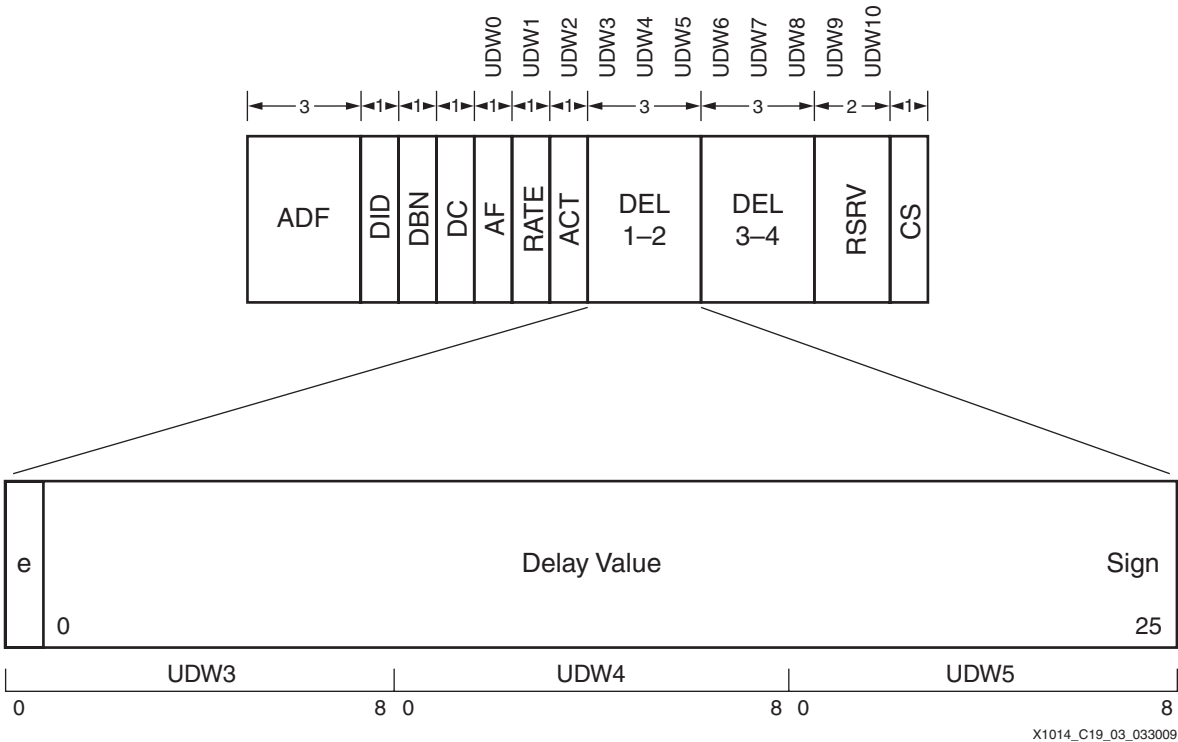


Figure 21-3: Audio Control Packet

The DID value depends on the audio group, as shown in [Table 21-1, page 478](#). DC is always 0x10B. The audio frame (AF) number, if used, indicates the position of the field

with respect to the audio frame sequence. If not used, it is set to all zeros. The sampling rate (RATE) indicates the sample rate of the embedded audio. The LSB is the asynchronous flag, asx. When asx is asserted, it means that the audio is asynchronous with the video. The rate codes are shown in Table 21-2. A synchronous sample rate of 48 KHz is almost exclusively used in the industry.

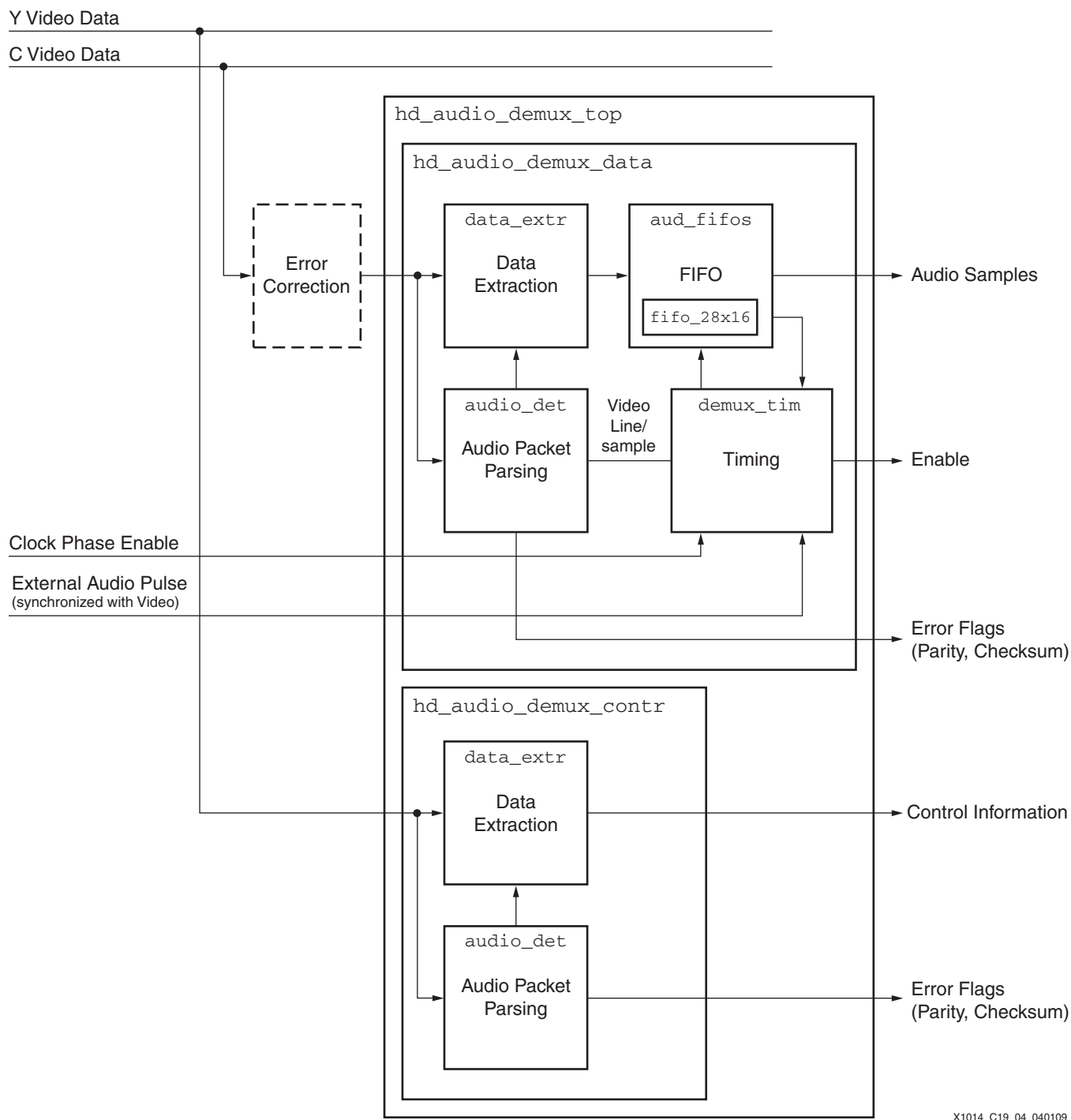
Table 21-2: Sample Rate Codes

RATE[3:1]	Sample Rate (KHz)
000	48.0
001	44.1
010	32.0
111	Free running
Other	Reserved

The ACT word indicates which of the four channels in the group is active. Bit 0 corresponds to CH1, bit 1 to CH2, and so forth. A logic 1 means that the channel is active. The delay (DEL) fields indicate the amount of accumulated audio delay relative to video in the indicated channel pair. This delay is measured in audio sample intervals and is a 26-bit two's-complement number. Positive values indicate that the video leads the audio. The LSB of the entire field is an enable bit. When this bit is set High, the delay value is valid. UDWs 9 and 10 are reserved and are set to 0.

Reference Design

The reference design for the audio demultiplexer is for one audio group. Multiple instances can be connected in parallel to demultiplex audio for multiple audio groups. The video stream, including ancillary data, is not altered. Error correction, if used, must be done prior to demultiplexing. A block diagram of the HD-SDI audio demultiplexer is shown in Figure 21-4.



X1014_C19_04_040109

Figure 21-4: HD-SDI Audio Demultiplexer Block Diagram

Two principal modules are contained within the top level of the demultiplexer: **hd_audio_demux_data**, for de-embedding audio data packets from the $C_B C_R$ video stream, and **hd_audio_mux_contr**, for de-embedding audio control packets from the Y video stream. Within these blocks are lower level blocks that perform various functions.

Each instance of the Audio Packet Parsing block monitors the video data looking for audio packets. One instance monitors the C_{BCR} data stream of data packets while another monitors the Y video stream for control packets. The Audio Packet Parsing block sends control information to the Data Extraction block that extracts the packet data from the video stream. While information from audio control packets is output directly, samples from audio data packets are loaded into a FIFO along with the corresponding clock phase information.

The timing block controls when the samples from the FIFO are output. There are two timing modes: internal and external timing. The Clock Phase Enable input controls the timing mode. For internal timing mode, output sample timing is created by comparing the clock phase data from the sample in the FIFO to the video line and sample information extracted by the Audio Packet Parsing block, and delaying the data by a few line times. In this mode, the Enable output pulse indicates the moment when each new set of samples is valid on the Audio Samples output. In external timing mode, the output of the FIFO is controlled by the External Audio Pulse input. At each pulse, a new set of samples is output from the FIFO. The External Audio Pulse must be derived from a clock synchronized with input video to avoid underflowing or overflowing the FIFO.

Inputs and Outputs

Table 21-3 lists the inputs and outputs of the top-level module, `hd_audio_demux`.

Table 21-3: Inputs and Outputs of `hd_audio_demux`

Signal	Direction	Description
vid_clk	In	This is the video clock.
rst	In	This is an asynchronous reset.
vid_ce	In	This is the video clock enable.
vid_samp_ce	In	This is the video sample-rate clock enable. This is the same as vid_ce except in the case of 12-bit video samples.
vid_y_in[9:0]	In	This is the Y input video stream.
vid_c_in[9:0]	In	This is the C_{BCR} input video stream.
group_sel[1:0]	In	This is the audio group number: 0 = Group 1 1 = Group 2, etc.
ext_audio_pulse	In	This is the timing pulse for output samples when clk_phase_en is asserted. When clk_phase_en is deasserted, ext_audio_pulse is ignored.
clk_phase_en	In	This signal enables the use of clock phase data to determine output timing. Assertion means that clock phase data is used, and deassertion means that ext_audio_pulse is used for the output sample timing.

Table 21-3: Inputs and Outputs of `hd_audio_demux` (Cont'd)

Signal	Direction	Description
audio_valid	Out	<p>This is the audio valid strobe and is used internally to read data from the FIFOs. Assertion indicates that new valid audio data is present on the outputs. This signal applies to all four channels.</p> <p>When <code>clk_phase_en</code> is asserted, <code>audio_valid</code> reflects the timing derived from the clock phase enable data.</p> <p>When <code>clk_phase_en</code> is deasserted, <code>audio_valid</code> is <code>ext_audio_pulse</code> after being synchronized to <code>clk</code> and made one <code>clk_en</code> period long.</p>
ch1_audio[23:0]	Out	Channel 1 audio sample data.
ch1_z	Out	Channel 1 and 2 AES audio block Z flag.
ch1_c	Out	Channel 1 channel status data.
ch1_u	Out	Channel 1 user data bit.
ch1_v	Out	Channel 1 valid bit.
ch2_audio[23:0]	Out	Channel 2 audio sample data.
ch2_c	Out	Channel 2 channel status data.
ch2_u	Out	Channel 2 user data bit.
ch2_v	Out	Channel 2 valid bit.
ch3_audio[23:0]	Out	Channel 3 audio sample data.
ch3_z	Out	Channel 3 and 4 AES audio block Z flag.
ch3_c	Out	Channel 3 channel status data.
ch3_u	Out	Channel 3 user data bit.
ch3_v	Out	Channel 3 valid bit.
ch4_audio[23:0]	Out	Channel 4 audio sample data.
ch4_c	Out	Channel 4 channel status data.
ch4_u	Out	Channel 4 user data bit.
ch4_v	Out	Channel 4 valid bit.
audio_frame[8:0]	Out	This is the audio frame number.
audio_rate[1:0]	Out	<p>This is the audio sample rate:</p> <p>00 = 48 KHz 01 = 44.1 KHz 10 = 32 KHz</p>
asx	Out	<p>This control packet value is a synchronous data flag:</p> <p>0 = Synchronous 1 = Asynchronous</p>
delay_1_2[25:0]	Out	This control packet value is the audio processing delay for channels 1 and 2.

Table 21-3: Inputs and Outputs of hd_audio_demux (Cont'd)

Signal	Direction	Description
delay_1_2_val	Out	This control packet value is the valid flag for delay_1_2 audio processing delay.
delay_3_4 [25:0]	Out	This control packet value is the audio processing delay for channels 3 and 4.
delay_3_4_val	Out	This control packet value is the valid flag for delay_3_4 audio processing delay.
parity_err_y	Out	This is the parity error flag for C _B C _R ancillary packets. Assertion indicates a parity error. This signal is asserted for one vid_ce period for each word that has a parity error.
chksum_err_c	Out	This is the checksum error flag for Y stream ancillary packets. Assertion of this signal indicates a checksum error. This signal is asserted for one vid_ce period for each ancillary packet that has a checksum error.
parity_err_y	Out	This is the parity error flag for Y stream ancillary packets. Assertion indicates a parity error. This signal is asserted for one vid_ce period for each word that has a parity error.
chksum_err_y	Out	This is the checksum error flag for C _B C _R ancillary packets. Assertion of this signal indicates a checksum error. This signal is asserted for one vid_ce period for each ancillary packet that has a checksum error.

Modules

The reference design contains seven modules. These are listed in [Table 21-4](#) and are described in more detail in the remainder of this section.

Table 21-4: List of Modules

Module	Description
hd_audio_demux_top	This is a top-level wrapper for the audio demultiplexer.
hd_audio_demux_data	This module demultiplexes audio data packets from the C _B C _R video stream.
hd_audio_demux_contr	This module demultiplexes audio control packets from the Y video stream.
audio_det	This module detects the presence, type, and location of ancillary data packets.
data_extr	This module extracts information from the audio data and control packets.
demux_tim	This module creates the timing for demultiplexed samples based on the clock phase information.
aud_fifos	This is the wrapper for instances of fifo_28x16 that store incoming audio samples prior to embedding.

Table 21-4: List of Modules (Cont'd)

Module	Description
<code>fifo_28x16</code>	This is a FIFO element, 28 bits wide by 16 locations deep, built from LUTs.
<code>sync_one_shot</code>	This module detects the rising edge of strobes and outputs a one-clock-wide pulse for each rising edge.

Data Packet Demultiplexer for C_BC_R Video Stream (`hd_audio_demux_data`)

This module is the top level for demultiplexing audio data packets from the C_BC_R video stream. A block diagram of the `hd_audio_demux_data` module is shown in the top half of [Figure 21-4, page 482](#). This module instantiates and connects the lower level blocks shown in the figure.

Data Packet Demultiplexer for Y Video Stream (`hd_audio_demux_contr`)

This module is the top level for demultiplexing audio control packets from the Y video stream. A block diagram of the `hd_audio_demux_contr` module is shown in the bottom half of [Figure 21-4, page 482](#). This module instantiates and connects the lower-level blocks shown in the figure.

Audio Packet Parser (`aud_det`)

This module monitors the incoming video stream, checking for EAV, SAV, and ancillary data packets. In particular, the module looks for audio data packets and audio control packets to be extracted. It outputs flags indicating the presence of EAV and SAV, the start of an ancillary packet, the type of packet present, the audio group to which it is targeted, and the number of the UDW. A FIFO enable for each group is output. At the top level, the FIFO enable corresponding to the group of interest enables writing of the FIFO. [Figure 21-5](#) is a block diagram of this module. Checking is done for parity errors in audio packets and checksum errors in all ancillary packets.

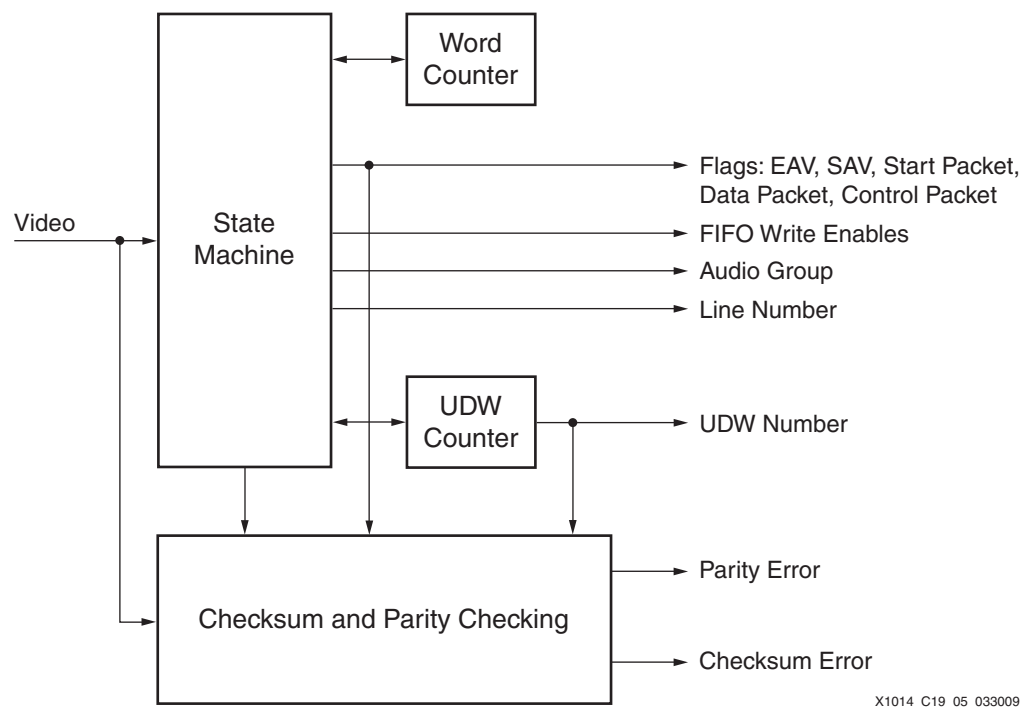


Figure 21-5: **Block Diagram of aud_det Module**

Figure 21-6 shows the state diagram for the state machine. In this figure, the equals sign on conditionals refers to the value of the video data, and *tcnt* refers to assertion of the terminal count of the word counter.

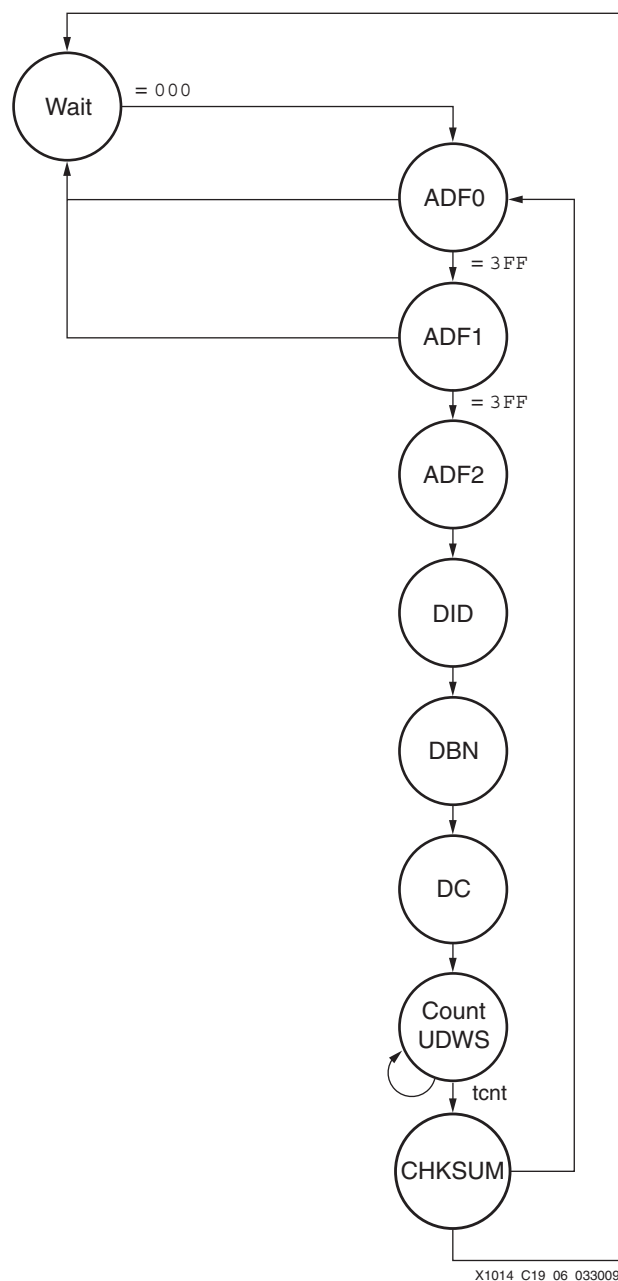
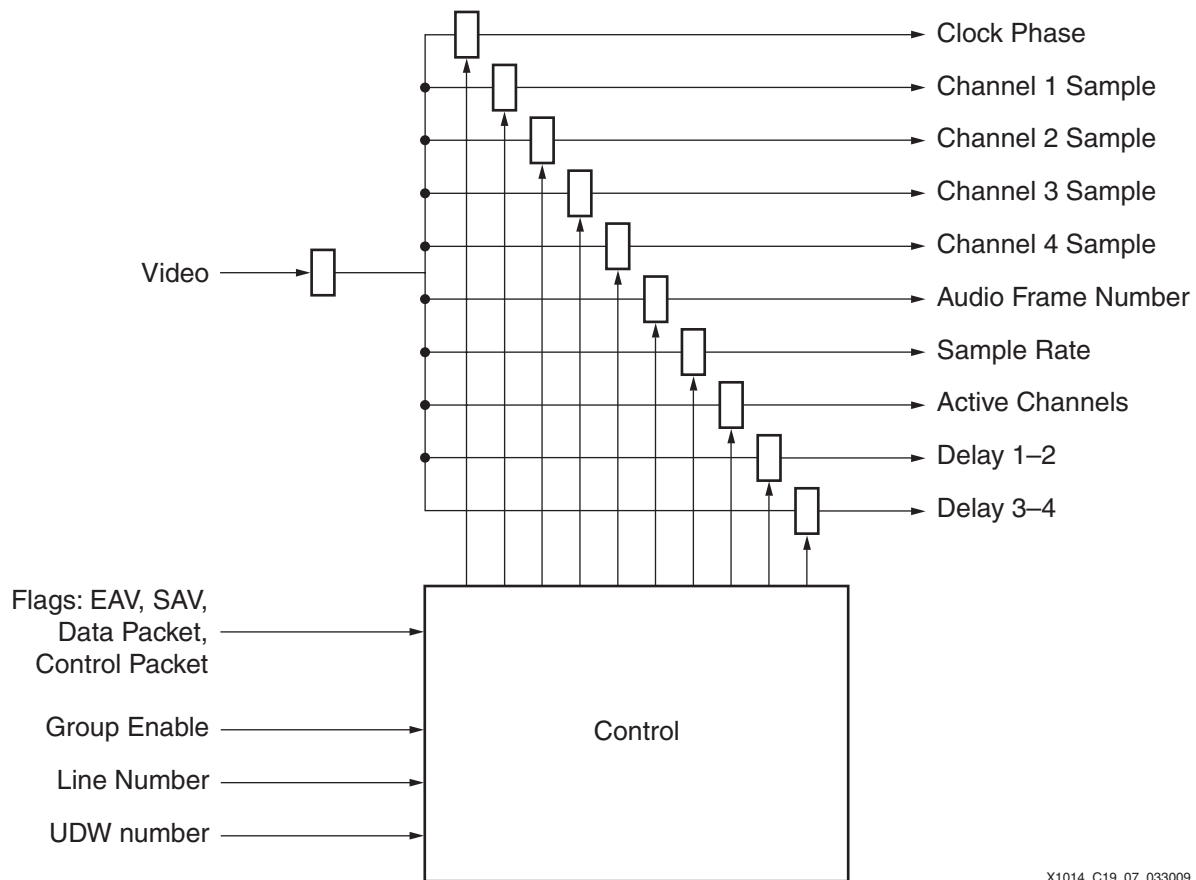


Figure 21-6: Audio Packet Parser State Diagram

Data Extraction (data_extr)

This module extracts the audio data from the video stream byte by byte, and puts it in a parallel format for FIFO storage. Information is received from the audio packet parser on the type of packet, and which UDW of that packet is present in the video stream in each clock cycle. Based on this, the audio data is extracted from the video stream, formatted, and sent to the FIFO section. Figure 21-7 shows a block diagram of the data_extr module.

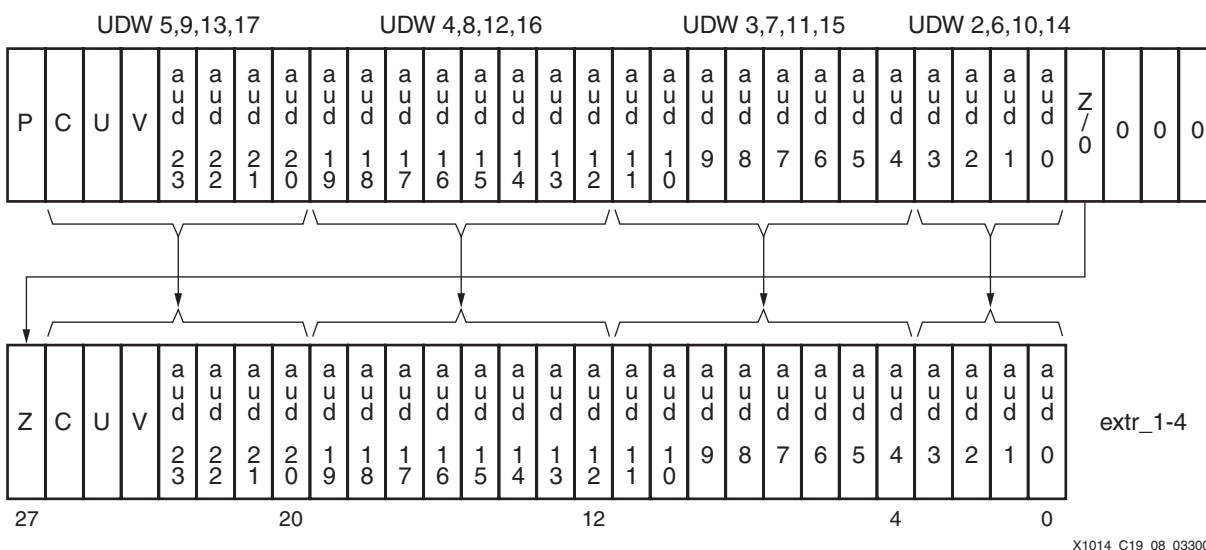


X1014_C19_07_033009

Figure 21-7: Block Diagram of `data_extr` Module

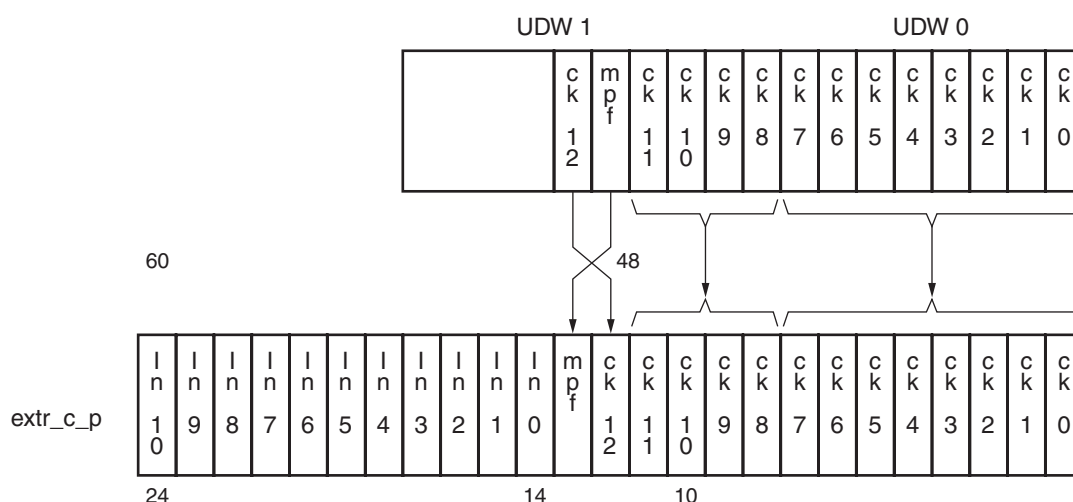
Line number information from the Audio Packet Parser module is combined with the clock phase information extracted in the Data Extraction module and stored in a FIFO along with the audio sample data. Figure 21-8 shows the formatting of the audio sample data, and Figure 21-9 shows the formatting of the clock phase data from the UDWs. Both types of data are stored to FIFOs at corresponding locations.

Audio control information is extracted from the audio control packets. Rather than being stored in a FIFO, the various fields are stored to registers and are parallel outputs of the `data_extr` module.



X1014_C19_08_033009

Figure 21-8: Formatting of Audio Sample Data

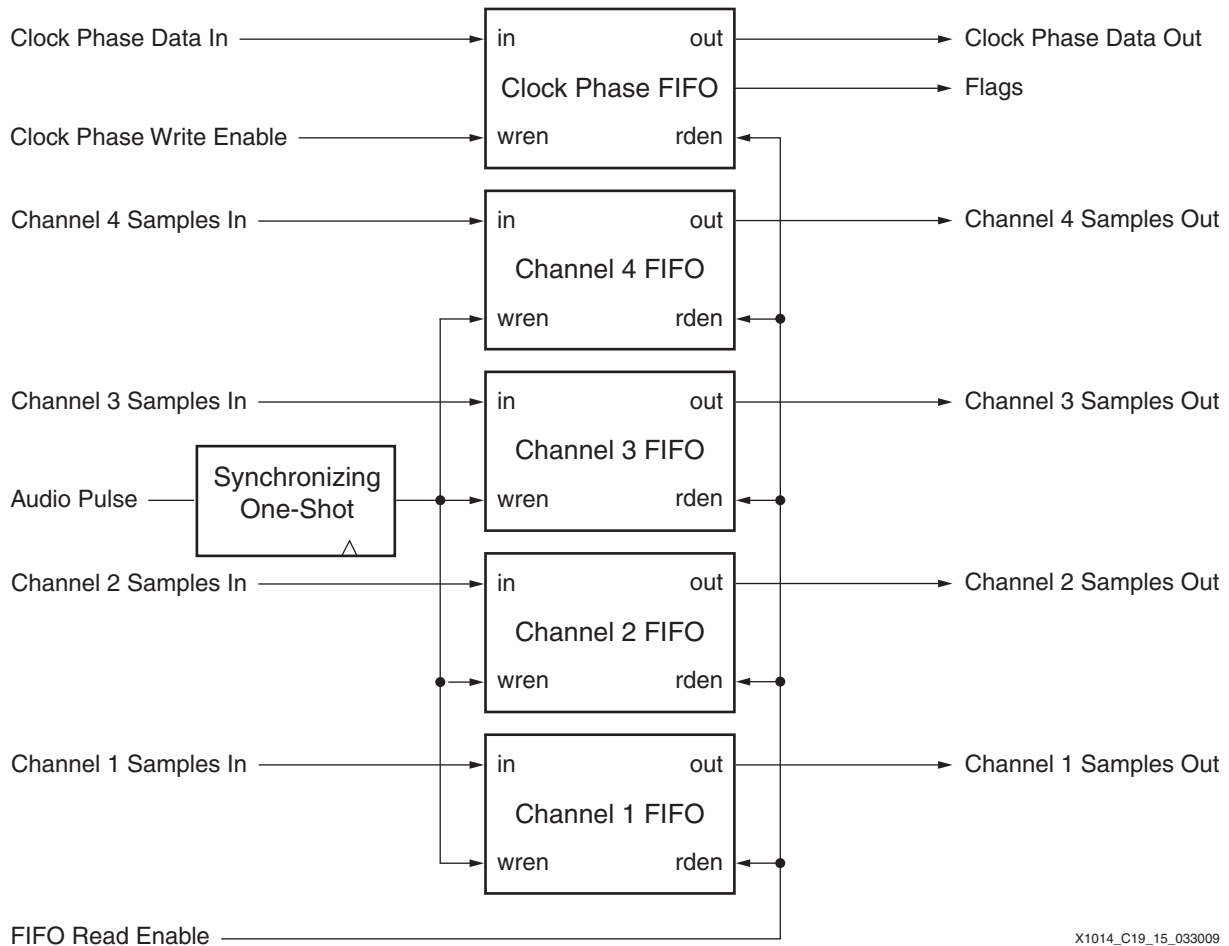


X1014_C19_09_033009

Figure 21-9: Formatting of Clock Phase Data

Audio FIFOs (aud_fifos)

The `aud_fifos` module acts as a small buffer between the demultiplexed sample stream, in which the samples are irregularly spaced in time, and the audio output, where audio samples are output on an evenly spaced basis (see Figure 21-10). The `aud_fifos` module contains five instances of the FIFO module `fifo_28x16`, one for each audio channel of the group, and one instance for the clock phase information. Reads are done in parallel to all FIFOs. Writes are done in parallel to the audio FIFOs, but the clock phase data is written separately.



X1014_C19_15_033009

Figure 21-10: Block Diagram of `aud_fifos` Module

FIFO (`fifo_28x16`)

This is the basic FIFO module consisting of SelectRAM™ memory built from LUTs (see Figure 21-11). It is 28 bits wide by 16 locations deep. Each FIFO stores either video samples or clock phase information. In video sample FIFOs, each location contains one 24-bit video sample and the associated Z, C, U, and V flags. For the clock phase FIFO, each location contains the clock phase information associated with the corresponding audio samples.

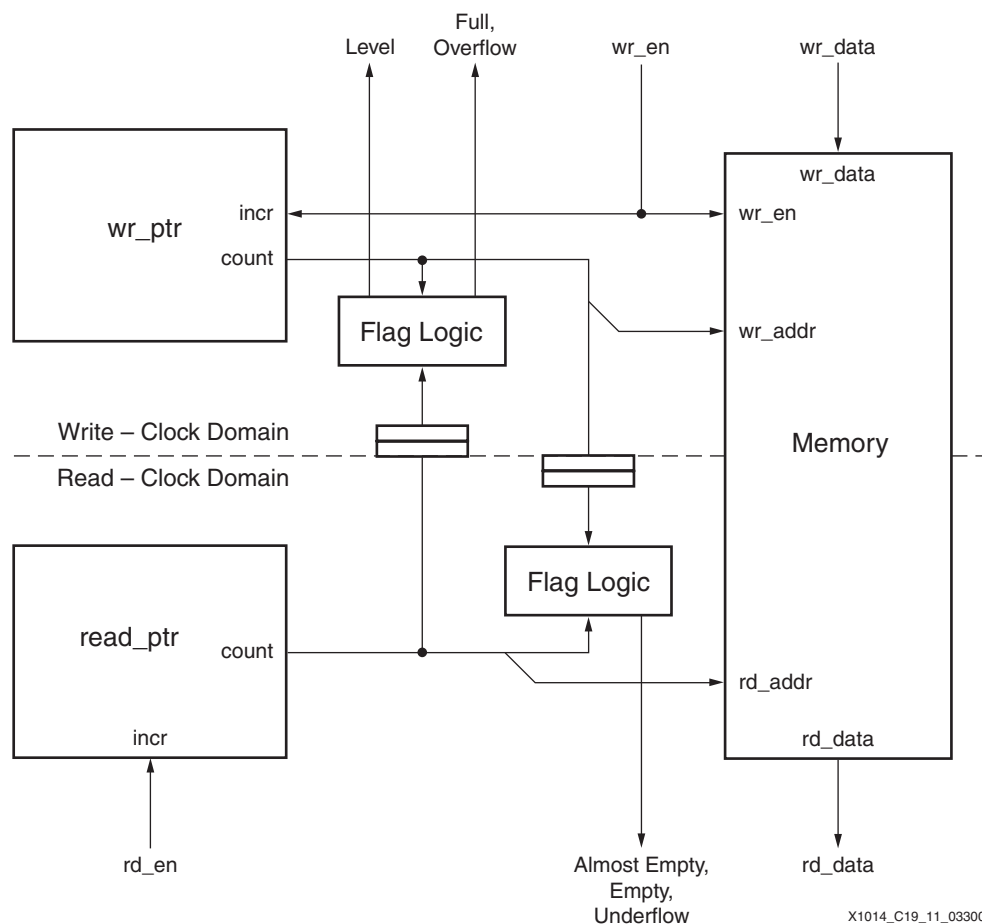


Figure 21-11: Block Diagram of FIFO

The control logic is built specifically for use in the reference design. In particular, it protects against underflow by not allowing the read pointer to increment when the FIFO is empty. At start-up, it is likely for read requests to occur when the FIFO is empty. Due to inherent latency in flag logic crossing clock domains, this can happen before the empty flag has propagated to the controlling logic for the read operation. In such cases, the read operation, specifically the moving of the read pointer, is inhibited. This is important because there are occasions in which the FIFO must be emptied fully, but must also be guaranteed to work properly, without rolling over when the next valid write data arrives.

As shown in Figure 21-12, the read and write pointers have two extra MSBs beyond the address. These indicate the number of passes or revolutions through the memory address space. When the read and write pointers are pointing to the same address, it can be unambiguously determined whether this represents a full condition (write revolution does not equal read revolution) or an empty condition (write revolution equals read revolution).

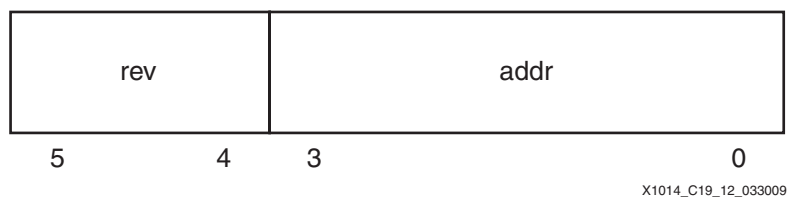
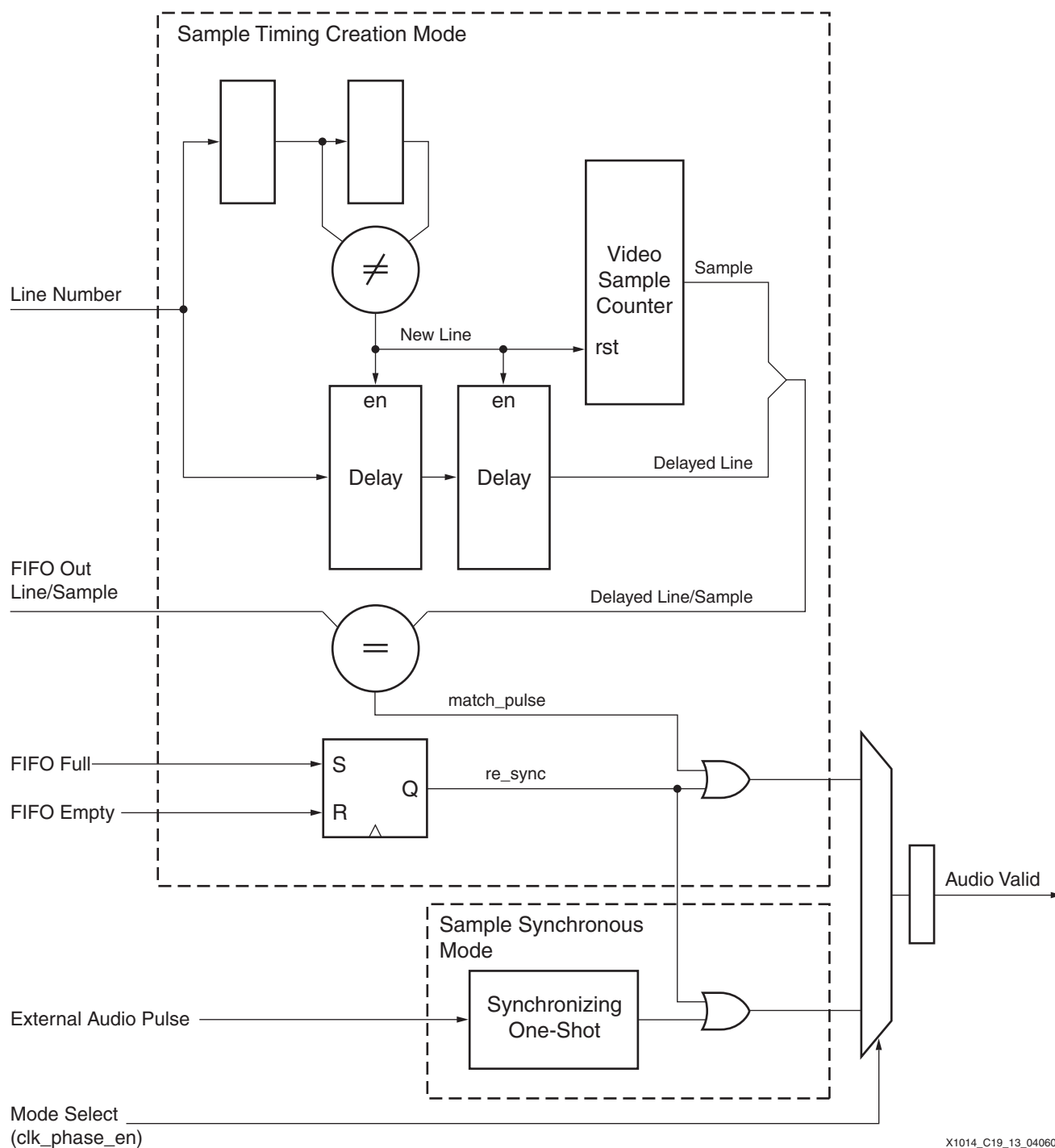


Figure 21-12: FIFO Pointer Format

Timing Control (demux_tim)

This module controls the timing of reads from the audio sample FIFO (see [Figure 21-13](#)). It has two modes of operation, referred to in this chapter as sample timing creation mode and sample synchronous mode.



X1014_C19_13_040609

Figure 21-13: Block Diagram of Timing Control Module

In sample timing creation mode, the line number and clock phase of the sample on the output of the FIFO are compared with the line and sample information from the video data to determine when the next sample should be read. The line number from the current scan is delayed by several lines to ensure that the audio samples are available in the FIFO before the time at which they are read.

The `demux_tim` module produces an audio valid pulse when the timing match occurs. The video clock and clock enable are used in this module and thus, the timing is accurate only to a video clock level. This module requires some type of sample rate smoothing, sample rate conversion, or both, if it is to be used to drive an AES output. Sample timing creation mode supports asynchronous audio.

Sample synchronous mode, on the other hand, requires that the embedded audio be synchronous, and that an audio-rate pulse be provided that is synchronized to the video. This pulse is used instead of `match_pulse` to read the FIFO.

In either mode, an automatic resynchronization operation is used if the FIFO becomes full to ensure that the FIFO maintains valid data. At start-up and anytime the FIFO becomes full, the `re_sync` signal is set and the `audio_valid` pulse is asserted until the FIFO is empty. At that point, the audio valid strobe and reading of the FIFO are disabled until the FIFO fills up to a certain level. In the sample synchronous mode, reading of the FIFO resumes with the next external audio pulse. This might result in attempted reads of an empty FIFO, but in that case, the read pointer is not advanced. In either mode, the result is a steady flow of data after a few video frames from start-up. In sample timing creation mode, reading resumes when the delayed line and sample match the line and sample value at the FIFO output. The FIFO must maintain a certain minimum fill level so that it can compensate for the burst nature of the writes from the demultiplex process.

Synchronizing One-Shot (`sync_one_shot`)

This small circuit detects a pulse, synchronizes it to another clock, identifies the rising edge, and outputs a one-cycle-wide pulse in the new clock domain. Figure 21-14 shows the details of the synchronizing one-shot circuit. This circuit can accommodate an extremely wide range of pulse widths on the input. The minimum width of the pulse is just long enough to be recognized as a High level by the SR latch. There is no maximum width. However, the input must remain Low for at least four clock cycles for the next rising edge to be recognized as a distinct pulse.

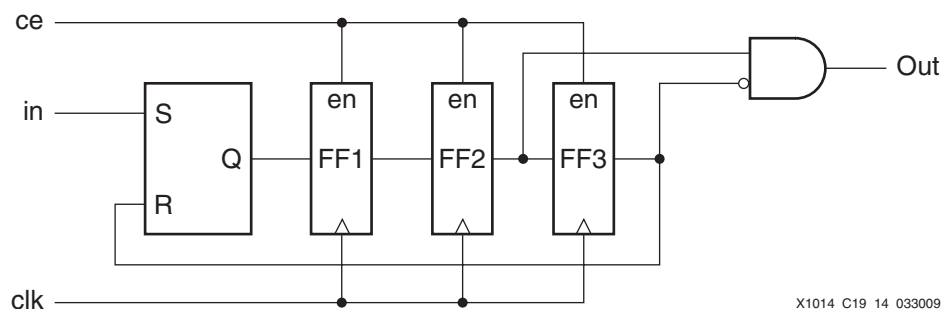


Figure 21-14: Synchronizing One-Shot Circuit

Figure 21-15 illustrates the operation of the synchronizing one-shot circuit for both short and long input pulses. Regardless of the length of the input pulse, the output is a one-clock-wide pulse, synchronized to the clock and occurring near the rising edge of the input pulse.

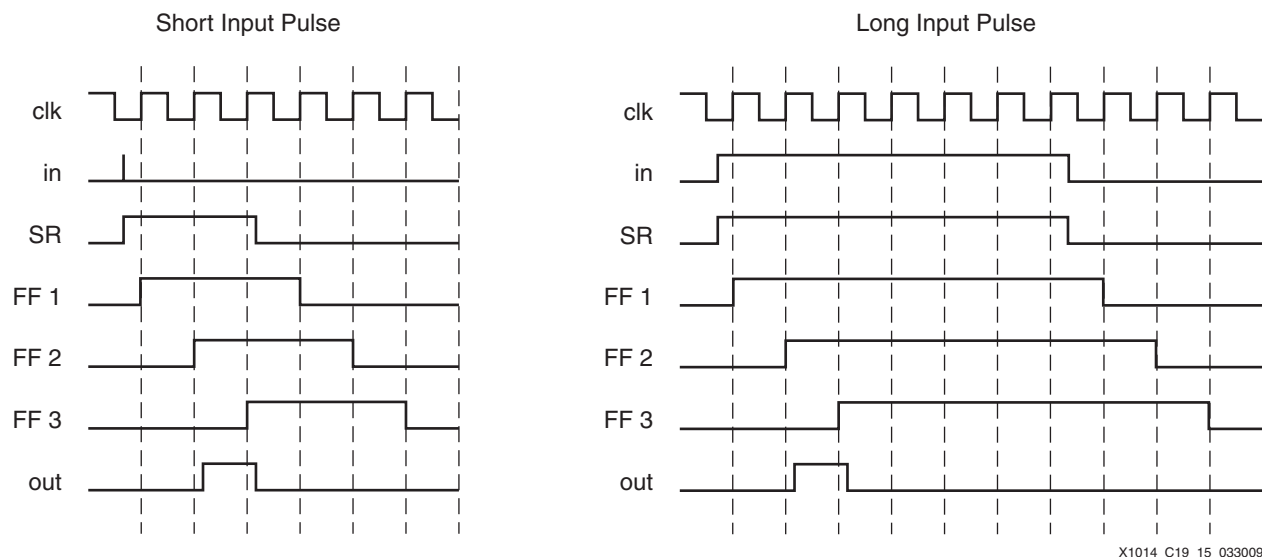


Figure 21-15: Synchronizing One-Shot Timing ($ce = 1$)

Usage Models

This section provides some examples of ways in which the HD-SDI audio demultiplexer can be used.

Synchronous De-embedding

For synchronous embedded audio, a synchronous, recovered audio clock can be used to directly time the output of de-embedded audio samples and an AES transmitter as shown in Figure 21-16.

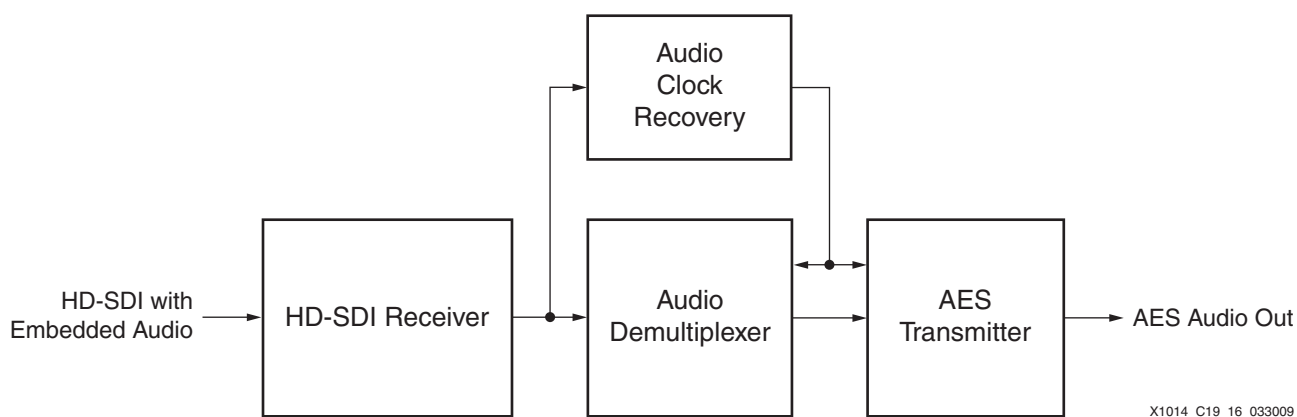


Figure 21-16: Synchronous De-embedding

Asynchronous De-embedding

An AES transmitter with arbitrary timing transmits de-embedded audio when a sample rate converter converts the de-embedded audio to a new timing base, as used by the AES transmitter. This is illustrated in Figure 21-17.

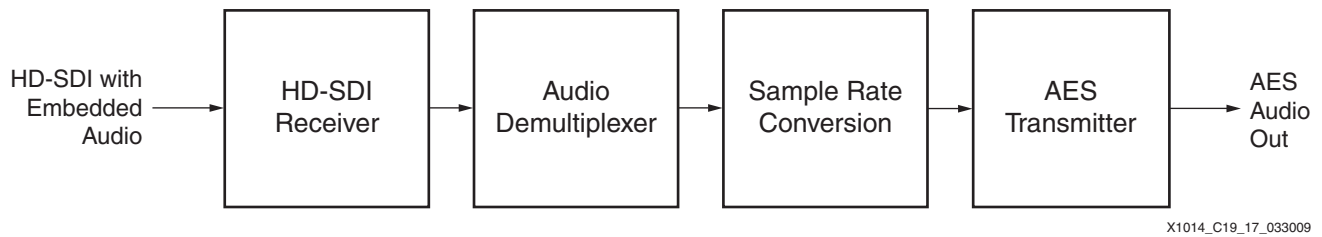


Figure 21-17: Asynchronous De-embedding

De-embedding Audio for Separate Video/Audio Processing

In some applications, audio and video should be processed separately from one another, as shown in Figure 21-18. For example, a video frame synchronizer drops complete frames of video. However, the audio cannot tolerate the equivalent dropping of samples, and so must be processed separately. In such cases, the audio is de-embedded from the video by the audio demultiplexer and processed in a separate path. After processing is completed, the audio and video are combined again in an audio multiplexer.

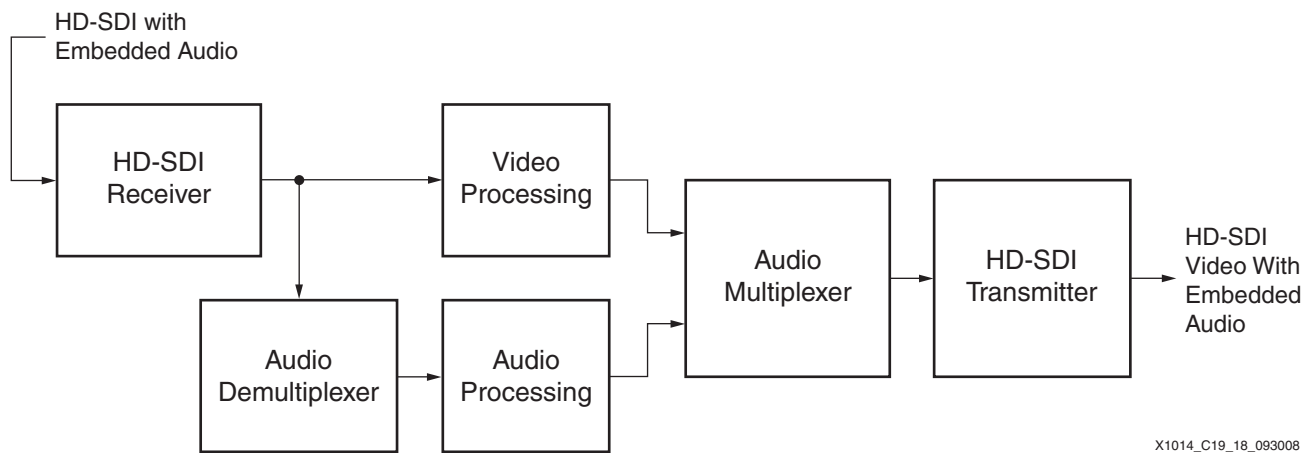


Figure 21-18: Separate Audio and Video Processing

Performance and Resources

The reference design is implemented and hardware verified in a Virtex-5 device. The reference design also works in other Xilinx FPGA families. The resources required for the reference design in a Virtex-5 device are shown in Table 21-5. These results were obtained in ISE® software, version 9.2i SP2. Timing, constrained for a processing clock of 148.5 MHz, was met for -1, -2, and -3 speed grade devices.

Table 21-5: FPGA Resources

Flip-Flops	LUTs	Block RAM	DSP Blocks
687	550	0	0

Design Files

The reference design for the HD audio demultiplexer is available at <https://secure.xilinx.com/webreg/clickthrough.do?cid=119758>. Open the ZIP archive and extract the file `xapp1014_c21_hd_audio_demux.zip`.

Conclusion

This chapter explains embedded audio in HD-SDI, the standards used to define it, and the data formats of the audio packets. The reference design of this chapter is useful for de-embedding audio from HD-SDI video. This design can be used in a variety of applications and can be implemented in all Xilinx FPGA families. It supports multiple HD-SDI line standards and multiple audio sample rates, both synchronous and asynchronous. The modularity of the design supports from one to four audio groups by adding multiple instances in parallel. Metadata contained in audio control packets is also decoded and output from the reference design.

Error Correction for HD-SDI Embedded Audio

Summary

This chapter introduces error correction concepts and methods as they pertain to SMPTE 299M, the audio format standard for multiplexed audio in HD-SDI video. This standard specifies the error correction scheme but offers little explanation. This chapter provides a background in information theory and number theory pertinent to implementing error correction for multiplexed audio without undergoing a rigorous mathematical derivation. The description given in this chapter is based on digital hardware engineering and, specifically, implementation in an FPGA. The included reference design implements error correction for HD-SDI embedded audio.

Features

The reference design has these features:

- Corrects up to eight single-bit errors in each audio data packet and up to eight sequential errors in the serial video stream.
- Performs error correction for up to 16 audio channels (four audio groups).
- Supports multiple HD-SDI standards. Being independent of line standard, the design supports all HD-SDI line standards.
- Supports many Xilinx® device families. The reference design has been hardware verified on the Virtex®-5 FPGA but uses features common to all Xilinx FPGAs.

Introduction

In practical communications systems, numerous sources of potential error for the communications channel can occur such as timing jitter, electrical noise, crosstalk, etc. Random errors in uncompressed video are manifest as random pixels of the wrong color or intensity. While this is undesirable and might be noticeable, a very small percentage of errors is generally tolerable if the errors are distributed more or less randomly. However, for multiplexed audio, a random bit error could cause, for example, a very loud pop or click at a quiet passage. This would be similar to an occasional frame of video that is totally white in an otherwise dark scene. Even though the error might be infrequent, it is extremely objectionable when it does occur.

Because the cost of bit errors in multiplexed audio is relatively high and the cost of error correction is relatively low, error correction techniques are used to minimize the chance of audio being permanently corrupted by the communications channel.

Error Correction Theory

Error correction schemes always involve error detection as a first step, and generally fall into one of two categories:

- **Retransmission:** When an error is detected, the data containing the error is discarded, and the sending entity is notified of the error. The data is then retransmitted by the sender. An example PCI Express® device has multiple levels of error detection using cyclic redundancy codes (CRCs). These errors are resolved by discarding and resending the packets containing the errors.
- **Forward Error Correction (FEC):** Retransmission is often not practical because there is no handshaking mechanism to communicate the need for retransmission or the time penalty for retransmission cannot be tolerated. In such cases, redundant data is sent that makes it possible for the receiving processor to correct errors in the corrupted data without retransmission. Sending redundant data amounts to a trade-off of lower peak data rates in exchange for a higher minimum data rate. For most video applications, the data rate must be constant, so FEC is the method of choice.

For multiplexed audio, forward error correction is mandated by SMPTE 299M. Error correction code fields are allocated in audio data packets to carry the redundant data necessary for error correction. The error detection and correction mechanism for embedded audio in HD-SDI video is specified as BCH (31, 25). BCH stands for Bose-Chaudhuri-Hocquenghem, who each independently discovered the properties of this code. BCH is a subset of linear block codes, where linear means that it comprises a group of vectors with q-ary elements (binary in this case), and block means that the vectors have a fixed size in terms of the number of bits in each code word. BCH codes are a superset of Reed-Solomon codes.

BCH codes have several parameters that are used to characterize them. These are presented in Table 22-1 along with their specific values for the error correction method for SMPTE 299M.

Table 22-1: Parameters for BCH Codes

Parameter	Relation to other Parameters	Description	Value for BCH (31, 25)
m		This is the extension field length and the number of ECC bits in the code word.	6
n	$N = 2^m - 1$	This is the length of the code word in bits.	31
k		This is the number of information bits in the code word.	25
t	$t \leq k/(n - m)$	This is the maximum number of error bits that can be corrected.	1
D_{\min}	$D_{\min} \geq 2t + 1$	D_{\min} represents minimum distance and is the minimum number of bits by which any two codes differ.	3

Table 22-1: Parameters for BCH Codes (Cont'd)

Parameter	Relation to other Parameters	Description	Value for BCH (31, 25)
Number of detectable errors	$2t$ or $D_{\min} - 1$	This is the maximum number of error bits that can be detected. If there are more than these, the code might be aliased as a different valid code word.	2
Generator polynomial		This number is used as a divisor for encoding and decoding BCH codes.	$x^6 + x^5 + x^3 + x^2 + x + 1$ (1101111)

The encoder takes 25 information bits and calculates a six-bit ECC. The ECC is appended to the audio data packets. This is illustrated in Figure 22-1. Eight ECCs are calculated independently, in parallel, one for each bit of the first 24 words of the audio data packet.

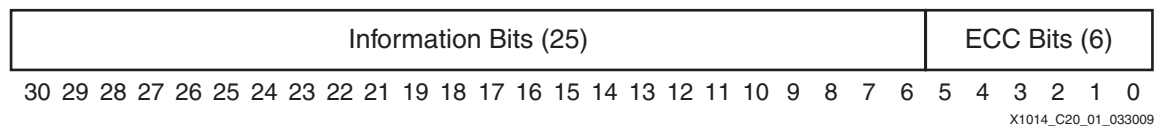


Figure 22-1: BCH (31, 25) Code Word

Single bit errors can be corrected based on the syndrome. If two bits are in error, the error is detected but cannot be corrected. If three or more bits are in error, the error might not be detected, and it cannot be corrected. This is a function of the parameters of the BCH code used. The minimum distance is 3. Distance refers to the Hamming distance, which is the minimum number of bits that are different between two adjacent code words. If a received erroneous code word differs from a valid code word by less than half the distance between two code words (in this case, less than half of 3, i.e., 1 bit), it can definitely be determined that the erroneous code word is closer to that valid code word than any other. If, however, the erroneous code word is different from the valid code word by two bits (the distance is 2), it is not closer to that valid code word than another and cannot be corrected accurately. If the distance differs by 3, the erroneous code word could alias as another correct code word (because the minimum distance between code words is 3) and the error would be detected, but it would be erroneously corrected as a single-bit error at yet another location.

Even though the BCH codes only correct single-bit errors, burst error correction is afforded. This is due to the fact that eight ECCs are calculated in parallel but operate on bits that are transmitted serially over the HD-SDI transmission channel. Thus, eight sequential bit errors in the channel are manifest as a single-bit error on each of eight parallel bits of a data word in the decoder, and each of these single-bit errors can be corrected.

Finite Fields

The numbers used for audio packet error correction use mathematical structures called finite fields. Finite fields are fully defined in pure mathematics, used extensively in the applied mathematics discipline of code theory, and realized in practical systems as error detection and correction codes. A finite field is a set of elements or numbers in which any operation of two numbers in the set results in a number that is also in the set. An example of a finite field is a set of memory addresses for a hardware memory. Each memory address has a fixed number of bits and, by definition, any address derived from manipulating

another address has the same number of bits. In contrast, an example of an infinite field is the set of real numbers.

The definition of finite fields also includes the operators of addition and multiplication, as well as multiplicative and additive identities, inverses, and commutative properties. A commonly used field is one with just two numbers {0,1}, known as a finite field of order 2. This is also known as a Galois field of order two, abbreviated as GF(2). This field is used as a building block for “extension fields,” as discussed later in this section. GF(2) is very similar to, but not the same as, a binary bit. The arithmetic for this field is simple and also has similarities to binary arithmetic. The differences result from the fact that the field is closed, meaning that all the operations result in either a 1 or a 0. There is no provision for making larger numbers. GF(2) uses modulo 2 arithmetic. Addition is an XOR operation (no carry), and multiplication is an AND operation.

For practical ECC codes, extension fields of the basic GF(2) are used to represent polynomials. These extension fields are very much like binary numbers used to represent integers but are not the same. For example, in binary notation, the number 110101 is interpreted in [Equation 22-1](#).

$$1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2 + 1 = 32 + 2^4 + 2^2 + 1 = 53 \quad \text{Equation 22-1}$$

The analogous finite field number, 110101, is in the extension field GF(2⁶). It represents the polynomial in [Equation 22-2](#).

$$1x^5 + 1x^4 + 0x^3 + 1x^2 + 0x + 1 = x^5 + x^4 + x^2 + 1 \quad \text{Equation 22-2}$$

In [Equation 22-2](#), x is only a placeholder in the notation, and the coefficients can only take on the values of the GF(2) field, namely, 0 or 1. In discussions of finite fields, the more cumbersome polynomial notation is often used for clarity, rather than the much more compact vector notation. For most FPGA designers, the vector notation is more convenient and is used in the rest of this discussion of ECC. However, the distinction between the finite fields and binary numbers should be borne in mind. Another caveat is that there is no strict convention as to how bit order (MSB to LSB) corresponds to the degree of the polynomial term. Whereas in binary notation, 11001 is always interpreted as $2^4 + 2^3 + 1$, in finite field notation, 11001 could represent $x^4 + x^3 + 1$ or $x^4 + x + 1$, depending on the context. Thus, it is important to understand which notation is being used. This chapter uses the convention that is most similar to binary notation, i.e., the highest degree term is on the left. For example, $10000 = x^4$.

The addition operation on extension field numbers (technically polynomials) amounts to a bitwise XOR (technically a component-wise modulo 2 addition). Multiplication is a bitwise AND.

$\begin{array}{r} 1\ 0\ 1\ 1\ 0\ 1 \\ +\ 1\ 0\ 1\ 0\ 1\ 1 \\ \hline 0\ 0\ 0\ 1\ 1\ 0 \end{array}$	$\begin{array}{r} 1\ 0\ 1\ 1\ 0\ 1 \\ \times\ 1\ 1\ 1\ 0\ 1\ 1 \\ \hline 1\ 0\ 1\ 0\ 0\ 1 \end{array}$
---	--

Much like binary numbers, finite field numbers can be converted to different lengths by padding with leading zeros. Thus, numbers of different lengths can be added or multiplied.

Finite field subtraction of GF(2^m) numbers is also an XOR function. Long division is performed in the usual way using the rules of finite field subtraction and multiplication. For example, division of 100000000000 by 1101111 yields a quotient of 111001 with a remainder of 111, as shown in [Figure 22-2](#).

[illegible]

Figure 22-2: Long Division of GF(2) Numbers

The remainder from a finite field divide operation is called a syndrome. Most error correction codes, including the ones for multiplexed audio on HD-SDI, are composed of a remainder from a divide operation, i.e., a syndrome. A syndrome is also used on the receiving end for error detection and correction. In the case of BCH codes, the information word (polynomial) is the dividend, and a special number called the generator polynomial is the divisor.

The generator polynomial is defined as a factor of $1 + x^n$ of order m . The derivation and choice of generator polynomials are beyond the scope of this chapter. They are chosen to provide maximum error correction capability. In the case of multiplexed audio, the generator polynomial is specified as $g(x) = x^6 + x^5 + x^3 + x^2 + x + 1$ or, in vector notation, 1101111. The prime factors of $1 + 2^{31}$ in modulo 2 arithmetic are $(x + 1)(x^5 + x^2 + 1)(x^5 + x^3 + 1)(x^5 + x^3 + x^2 + x + 1)(x^5 + x^4 + x^2 + x + 1)(x^5 + x^4 + x^3 + x + 1)(x^5 + x^4 + x^3 + x^2 + 1)$. The generator polynomial is the product of two of these: $(x + 1)(x^5 + x^2 + 1)$. This polynomial can be checked for compliance to the criteria of a generator polynomial. From Table 22-1, for BCH (31, 25) codes, $m = 6$ and $n = 31$. It can be seen by inspection that $g(x)$ is of order m . The generator polynomial can be verified to be a factor of $1 + x^n$ by performing long division of this polynomial, 100000000000000000000000000001 by the generator, 1101111, as shown in Figure 22-3.

[illegible]

$$\begin{array}{r}
 0\ 0\ 0\ 0\ 1\ 1\ 1 \\
 \underline{1\ 1\ 0\ 1\ 1\ 1\ 1} \\
 0\ 0\ 1\ 1\ 1\ 1\ 1 \\
 \underline{1\ 1\ 0\ 1\ 1\ 1\ 1} \\
 0\ 0\ 1\ 0\ 0\ 1\ 1 \\
 \underline{1\ 1\ 0\ 1\ 1\ 1\ 1} \\
 0\ 1\ 0\ 0\ 0\ 1\ 1 \\
 \underline{1\ 1\ 0\ 1\ 1\ 1\ 1} \\
 0\ 1\ 0\ 1\ 0\ 0\ 1 \\
 \underline{1\ 1\ 0\ 1\ 1\ 1\ 1} \\
 0\ 1\ 1\ 1\ 1\ 0\ 1 \\
 \underline{1\ 1\ 0\ 1\ 1\ 1\ 1} \\
 0\ 0\ 1\ 0\ 1\ 0\ 1 \\
 \underline{1\ 1\ 0\ 1\ 1\ 1\ 1} \\
 0\ 1\ 1\ 1\ 0\ 1\ 1 \\
 \underline{1\ 1\ 0\ 1\ 1\ 1\ 1} \\
 0\ 0\ 1\ 1\ 0\ 0\ 1 \\
 \underline{1\ 1\ 0\ 1\ 1\ 1\ 1} \\
 0\ 0\ 0\ 1\ 0\ 1\ 1 \\
 \underline{1\ 1\ 0\ 1\ 1\ 1\ 1} \\
 0\ 1\ 1\ 0\ 1\ 1\ 1\ 1 \\
 \underline{1\ 1\ 0\ 1\ 1\ 1\ 1} \\
 0\ 0\ 0\ 0\ 0\ 0\ 0
 \end{array}$$

Figure 22-3: $1 + x^n$ Divided by the Generator Polynomial

The syndrome is all zeros, meaning that the generator polynomial is indeed a factor of $1 + x^n$. This calculation is instructive because it illustrates how error correction is implemented. The ECC codes are constructed such that all complete code words (information words and ECC word) are multiples of the generator polynomial. At the receiving end, the complete code word is divided by the generator polynomial. Just like in the calculation of Figure 22-2, page 503, the resulting syndrome is all zeros if the word is correct.

Error Correction Hardware

Embedded audio in HD-SDI is defined for error correction purposes as the 30 LSBs of a 31-bit BCH (31, 25) code word. The MSB, bit 30, is considered to be constant at 0.

Figure 22-4 shows the format of the HD-SDI audio data packet.

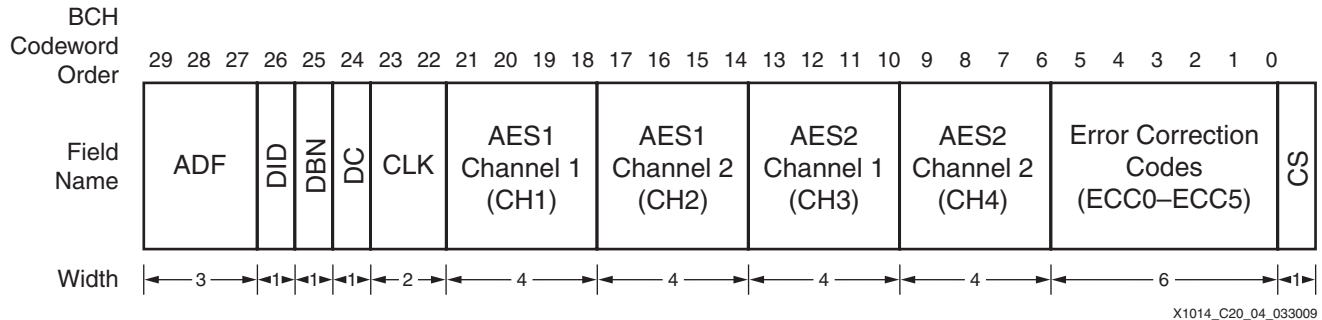


Figure 22-4: Audio Data Packet in HD-SDI

Because remainders (syndromes) are used as ECCs, the primary hardware task is to perform the divide operations. This is efficiently implemented with a linear feedback shift register (LFSR) with XOR gates and shift registers (see Figure 22-5). While there are many variations of the same basic circuit, this chapter uses the form described in SMPTE 299M. This is a fixed divisor serial divider.

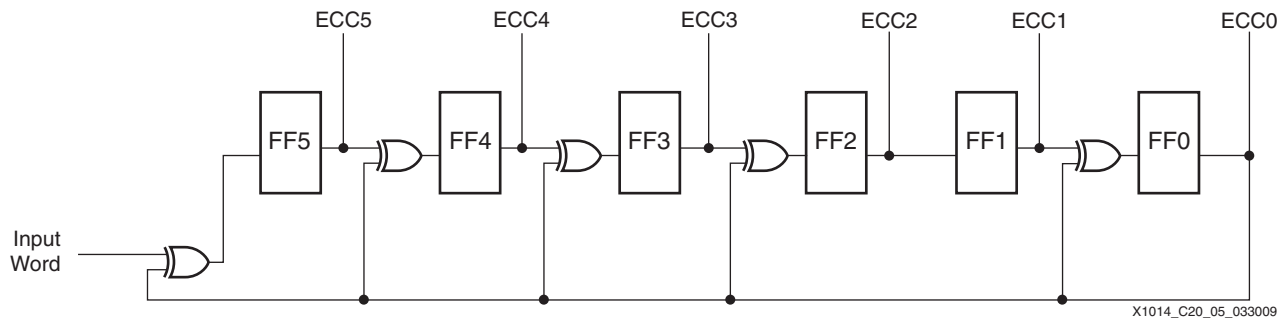


Figure 22-5: ECC Code Generation and Error Detection Circuit

Conceptually, the divider amounts to a series of subtractions, each time appending the new values to the x^0 term as done in long division. Because the divide operation is performed in a bitwise-like fashion, only the coefficient of the most significant bit is required to compare the divisor to the current portion of the dividend. After all of the information bits have been fed in, the remainder (syndrome) is contained in the six registers. FF0 corresponds to the MSB of the calculations, while FF5 corresponds to the LSB. Also, the input word is fed in from MSB to LSB, meaning that the MSB of the information is the first temporal bit. This is from the ancillary data flag in the SMPTE 299M standards.

Encoding

To create the ECC word, the information word is divided by the generator polynomial. The remainder of this operation is the 6-bit ECC word. There are eight copies of this circuit, one for each bit of the UDWs, resulting in eight 6-bit ECC words.

Decoding

Decoding is accomplished by dividing the entire code word (25 information bits and 6 ECC bits) by the generator polynomial using the same circuit used for encoding. The ECC bits are appended with ECC0 as the MSB and ECC5 as the LSB. If the remainder (syndrome) is 000000, the code word was received correctly. If the remainder is not all zeros, an error is detected. If the error is a single bit, it can be corrected based on the syndrome.

Several methods are used to determine where errors in BCH codes occur. BCH codes with $t = 1$ (i.e., only one error can be corrected) have the simplifying property that the location of the error is uniquely determined by the error syndrome of the code word. Thus, when the full 31-bit code word is divided by the generator polynomial, the remainder has a one-to-one correspondence to the bit location that is in error. Therefore, a simple LUT can be used to determine which bit is in error. Because there are six syndrome bits, the LUT must have $2^6 = 64$ locations. Three separate cases can be determined directly from the syndrome:

- No error: The code word is valid. One address, equal to 0000000.
- One error: The bit location of the error can be determined. 31 addresses.
- More than one error: The error is detected but cannot be corrected. 32 addresses.

Some errors with more than two erroneous bits are interpreted as having no error. This is the result of having a minimum distance of 3. In the worst case, the error can be mistaken for a correctable error and corrupt yet another bit of data. For embedded audio, there is no possibility of retransmission, so either there are correctable errors, in which case they are corrected, or the data is corrupt and irrecoverable. Thus, at the end of the error correction process, there are only two possibilities:

- The data is correct. Either it was received with no errors or had one error that was corrected.
- The data is corrupt. It was received with more than one error, and the errors were not successfully corrected.

Reference Design

The reference design consists of two major parts, packet parsing and error correction. As shown in [Figure 22-6](#), the error correction reference design of this chapter is designed to precede the audio demultiplexer of [Chapter 21, Audio Demultiplexer for HD-SDI Video](#). The reference design corrects errors in all audio groups, so one instance of error correction can be used with multiple instances of the demultiplexer.

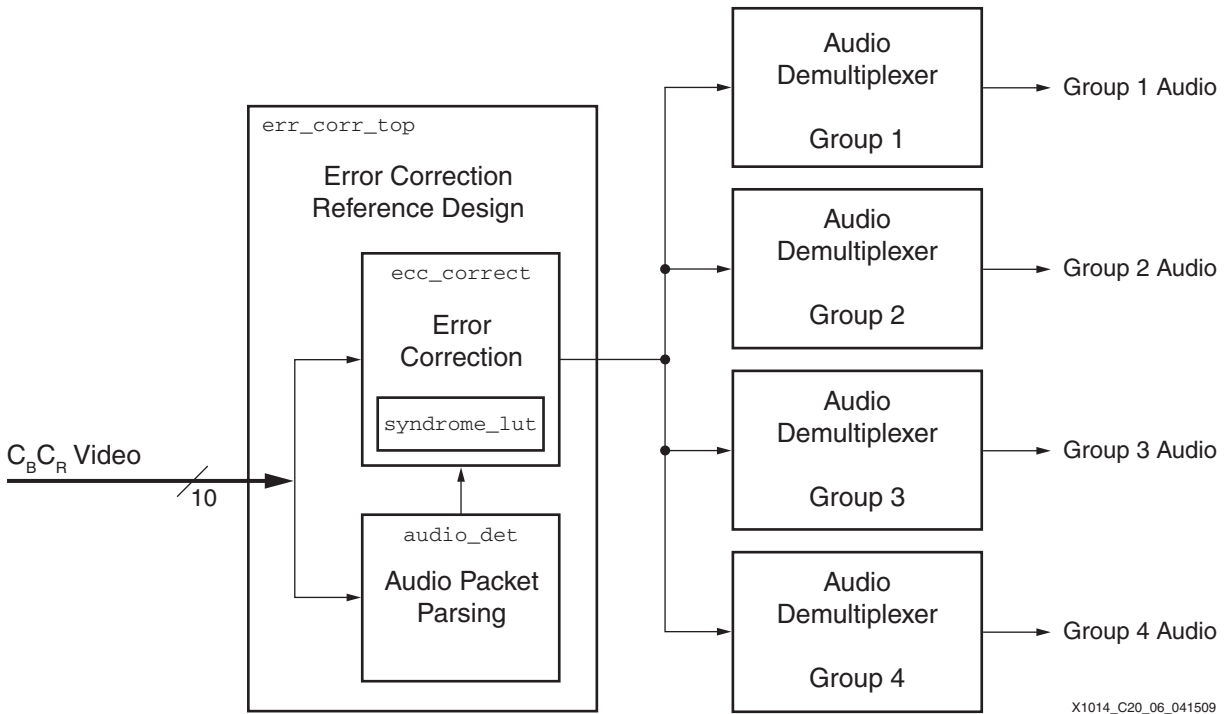


Figure 22-6: Error Correction and Audio Demultiplexing

All audio packets are in the $C_B C_R$ video stream, not in the Y stream, so this is the only data input to the error correction design. The only part of the video stream that might be changed is the audio data packets. The rest of the video stream, i.e., video data and other ancillary packets, is passed unmodified. The delay through the reference design is approximately 33 video sample times. It is possible to use the error correction reference design on its own without subsequent demultiplexing. This requires that the Y video stream be delayed to match the latency of error correction.

Inputs and Outputs

Table 22-2 lists the inputs and outputs of the top-level module, `err_corr_top`.

Table 22-2: Inputs and Outputs

Signal	I/O	Description
clk	In	This is the video rate clock input.
rst	In	This is the reset input.
clk_en	In	This is the clock enable. The data advances only when <code>clk_en</code> is active.
start_corr	In	This is the start correction flag. This flag signals the start of an ancillary packet.
valid_d_pkt	In	This is the valid data packet. This flag means that an audio data packet has been detected.
video_in[9:0]	In	This video input is the $C_B C_R$ video stream.

Table 22-2: Inputs and Outputs (Cont'd)

Signal	I/O	Description
video_out[9:0]	Out	This video output is the $C_B C_R$ video stream with error corrected in the audio data packets.
err_corr	Out	This is the corrected error flag. It pulses High for every packet with errors that are corrected.
err_uncorr	Out	This is the uncorrected error flag. It pulses High for every packet with detected errors that cannot be corrected.

Modules

The reference design contains four modules. These are listed in Table 22-3 and are described in more detail in the remainder of this section.

Table 22-3: Reference Design Modules

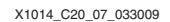
Module	Description
err_corr_top	This is the top-level wrapper for audio error correction.
ecc_correct	This module performs error correction by identifying the location of erroneous bits and inverting them.
syndrome_lut	This module contains the LUT for error position, based on the syndrome.
audio_det	This module detects the start of ancillary packets and identifies audio data packets.

Top Level Wrapper (err_corr_top)

This module instantiates the two lower level modules, `ecc_correct` and `audio_det`. A block diagram of this module is shown in Figure 22-6, page 507.

ECC-Based Error Correction (ecc_correct)

This module calculates the syndromes of the received data. It looks up the location of the erroneous bit based on the syndrome, and inverts the proper bit to correct the error. This is done in parallel on the eight bits of video data. Figure 22-7 is a block diagram of the `ecc_correct` module.



The most significant two bits of the incoming video stream are largely parity bits and are excluded from the error correction process. The low-order eight bits go to the ECC divider. The ECC divider is the same circuit that is used for generating ECC codes for audio data packets. There are eight such units operating in parallel. The syndrome generation differs from the ECC generation in that the received ECC codes are also run through the divider. Thus, the calculation has an extra six states, one for each of the six ECC words in the embedded audio packets. The result of this division is the syndrome or, in this case, eight syndromes. The output of the divider is six 8-bit values corresponding to the flip-flops of the divider circuit. This is regrouped into eight 6-bit values that constitute the syndromes

for each of the eight least significant data bits of the audio data packet. If the syndrome is zero, there are no errors. If the syndrome is other than zero, there are one or more errors.

Each syndrome goes to a LUT that maps the syndrome to the location of the error. Because the syndrome is six bits wide, the syndrome LUT has 64 entries. The length of the code word is 32 bits, so six bits are required to designate the location of the error. In addition, the LUT contains flags to indicate whether or not there is an error, and if there is an error, whether the error is correctable or not. Thus, each entry in the LUT has eight bits, six for the location plus two flags. The LUT is in a separate module, as described in [Error Look-up Table \(syndrome_lut\)](#).

The first four words of the packet are required to identify it as an audio data packet. Therefore, errors in these words are not corrected because they are a prerequisite to the error correction process. All other single-bit errors have a syndrome that corresponds to a correctable error. Any even number of errors result in a syndrome at a location designated uncorrectable. An odd number of errors of three or more alias to a correctable error and result in an erroneous attempt to correct the error. In any case, no more than one error in the sequence can be corrected. This is the inherent limitation of BCH (31, 25) codes. However, the fact that eight corrections happen in parallel means that a total of eight errors can be corrected in each audio data packet as long as they all occur on different data lines. Because HD-SDI is a serial protocol, the effective maximum error length is eight bits, i.e., an error sequence of eight sequential bits in the audio control packet transmitted via HD-SDI can be fully corrected.

While the syndrome calculation is being performed, the video data is delayed by 31 states through a shift register. The actual correction to the video stream is done by comparing the error pointer from the LUT to the correction word number. The correction word counter counts the words of the audio data packet as they exit the delay block in BCH order, i.e., from 30 down to 0. This is because division by the generator polynomial is done from MSB to LSB, and the first temporal word is the most significant, i.e., bit 29 (bit 30 is always 0). When there is a match between the error bit from the table and the word number from the correction word counter, the video word present on the output of the delay has an error at the corresponding bit position. The associated bit in the video word is toggled, thereby correcting the error. This compare-and-toggle operation is performed independently for each of the eight LSBs of the video word. The corrected word is clocked into the video_out register. The two MSBs bypass the correction operation and go straight to the video_out register.

The status logic block monitors the flags from the LUTs to determine whether the packet contains errors and whether or not the errors are correctable. The status logic block outputs two flags that apply to the entire packet. The error detected flag indicates when one or more errors were detected and corrected. The error uncorrected flag indicates when one or more uncorrectable errors were detected.

Error Look-up Table (syndrome_lut)

The `syndrome_lut` module maps the calculated syndrome to the error position in the BCH code word. There are eight instances of this module in `ecc_correct`. The contents of the LUT are given in [Table 22-4](#).

Table 22-4: Syndrome LUT

Address	Correctable Flag	Error Flag	Error Location	Table Entry (Hex)
0	0	0	0	00 ⁽¹⁾
1	1	1	26	1A

Table 22-4: Syndrome LUT (Cont'd)

Address	Correctable Flag	Error Flag	Error Location	Table Entry (Hex)
2	1	1	27	1B
3	0	1	-	40
4	1	1	28	1C
5	0	1	-	40
6	0	1	-	40
7	1	1	6	06
8	1	1	29	1D
9	0	1	-	40
10	0	1	-	40
11	1	1	22	16
12	0	1	-	40
13	1	1	3	03
14	1	1	7	07
15	0	1	-	40
16	1	1	30	1E
17	0	1	-	40
18	0	1	-	40
19	1	1	12	0C
20	0	1	-	40
21	1	1	17	11
22	1	1	23	17
23	0	1	-	40
24	0	1	-	40
25	1	1	20	14
26	1	1	4	04
27	0	1	-	40
28	1	1	8	08
29	0	1	-	40
30	0	1	-	40
31	1	1	10	0A
32	0	1	(31)	40 ⁽²⁾
33	0	1	-	40
34	0	1	-	40
35	1	1	14	0E

Table 22-4: Syndrome LUT (Cont'd)

Address	Correctable Flag	Error Flag	Error Location	Table Entry (Hex)
36	0	1	-	40
37	0	1	-	40
38	1	1	13	0D
39	0	1	-	40
40	0	1	-	40
41	1	1	15	0F
42	1	1	18	12
43	0	1	-	40
44	1	1	24	18
45	0	1	-	40
46	0	1	-	40
47	1	1	1	01
48	0	1	-	40
49	1	1	2	02
50	1	1	21	15
51	0	1	-	40
52	1	1	5	05
53	0	1	-	40
54	0	1	-	40
55	1	1	25	19
56	1	1	9	09
57	0	1	-	40
58	0	1	-	40
59	1	1	19	13
60	0	1	-	40
61	1	1	16	10
62	1	1	11	0B
63	0	1	-	40

Notes:

1. No error.
2. This syndrome represents an uncorrectable error because BCH word 31 does not correspond to any data in the packet.

Audio Packet Parsing (aud_det)

This module is the same as the one used for audio demultiplexing. The error correction function uses only a subset of its full functionality, specifically, identifying the start of all

ancillary data packets and the presence of valid audio data packets. This module monitors the incoming video stream, checking for EAV, SAV, and ancillary data packets. The Audio Packet Parsing module looks for audio data packets and audio control packets to be extracted. It outputs flags indicating the presence of EAV and SAV, the start of an ancillary packet, the type of packet present, the audio group to which it is targeted, and the number of the UDW. A FIFO enable for each group is output. Checksum and parity errors are also checked. For this application, only the start of packet and type of packet information is required. The other outputs are left unconnected, and the optimization process removes the logic for the unneeded functionality. Figure 22-8 is a block diagram of the Audio Packet Parsing module.

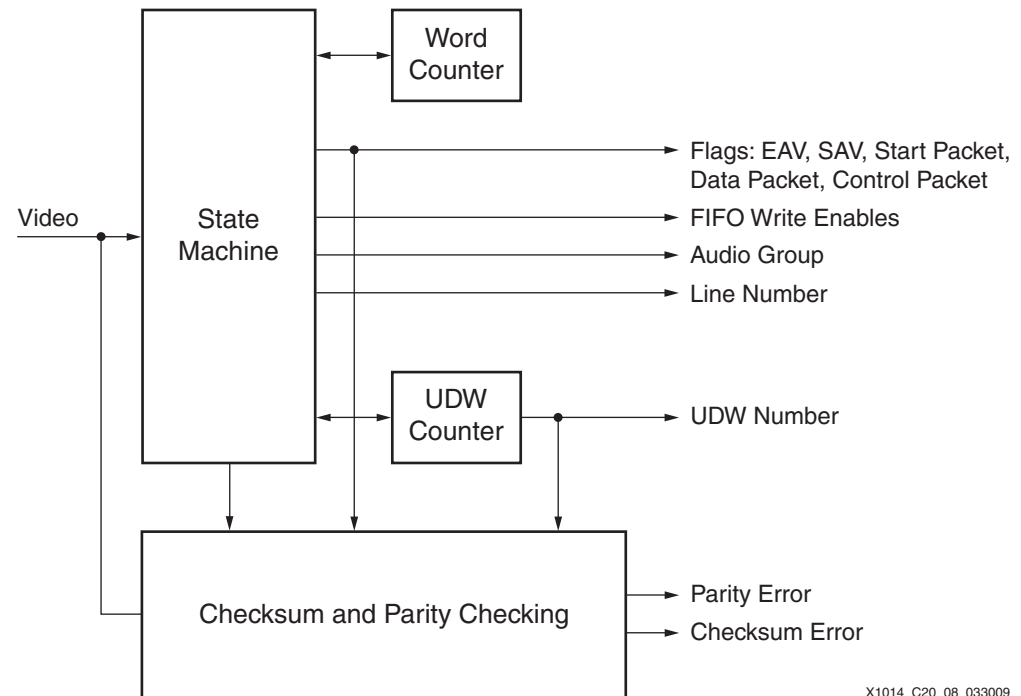


Figure 22-8: Block Diagram of aud_det Module

Figure 22-9 is the state diagram for the state machine. In this figure, the equals sign on conditionals refers to the value of the video data, and **tcnt** refers to assertion of the terminal count of the word counter.

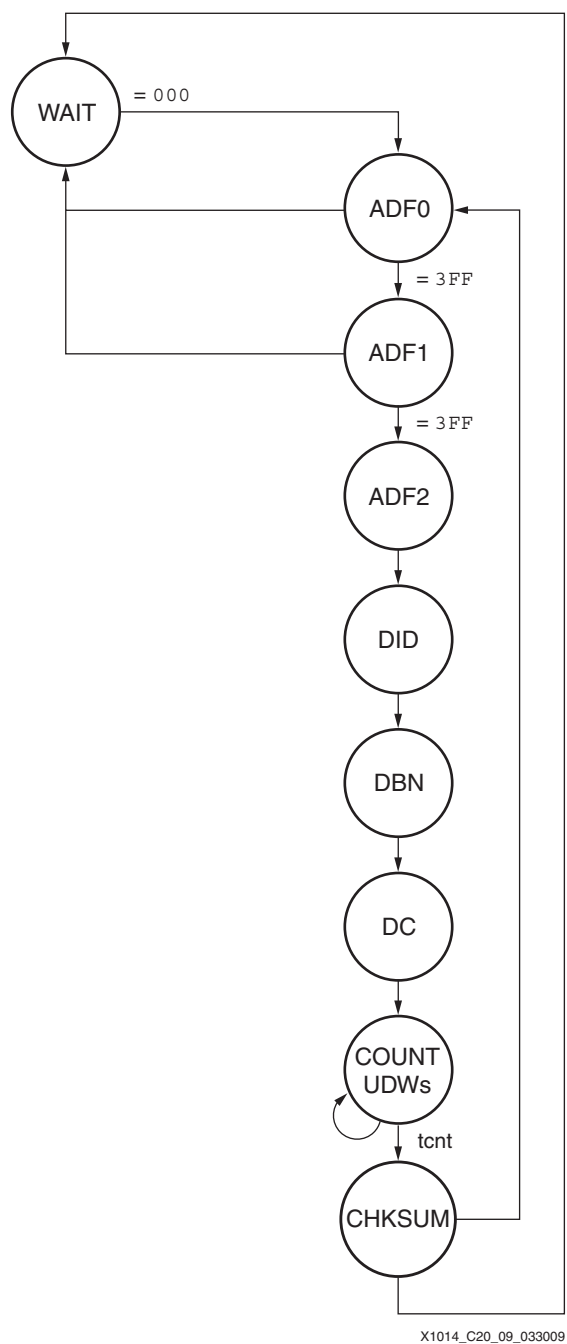


Figure 22-9: Audio Packet Parser State Diagram

FPGA Resources

The reference design is implemented and hardware verified in a Virtex-5 device. The reference design also works in other Xilinx FPGA families. The resources required for the reference design in a Virtex-5 device are shown in Table 22-5. These results were obtained in ISE® software, version 9.2i SP2. Timing, constrained for a processing clock of 148.5 MHz, was met for -1, -2, and -3 speed grade devices.

Table 22-5: **FPGA Resources**

Flip-Flops	LUTs	Block RAM	DSP Blocks
237	279	0	0

Design Files

The reference design for embedded audio error correction is available at <https://secure.xilinx.com/webreg/clickthrough.do?cid=119758>. Open the ZIP archive and extract the file xapp1014_c22_err_corr_emb_audio.zip.

Conclusion

This chapter explains error correction and code theory as it pertains to correcting errors in embedded audio packets embedded in HD-SDI video streams. The error correction scheme uses BCH (31, 25), which has the capacity to correct single bit errors and detect two-bit errors. The theory is implemented by LFSRs for both encoding and decoding. For decoding, the syndrome is used directly to look up the location of the error. The error correction reference design is placed before audio demultiplexing and corrects audio packet errors for all embedded audio data packets.

Audio Demultiplexer for Standard-Definition Digital Video

In many applications, embedded audio must be extracted or demultiplexed from the video stream so that the audio and video can be processed separately. This chapter describes an audio demultiplexer for SD digital video. This small, efficient reference design can be implemented in the Virtex®-5 FPGA family. One module can demultiplex audio from up to 16 separate video streams.

Specifications

The demultiplexer detects and extracts embedded audio conforming to SMPTE 272M from SD 4:2:2 component digital video streams conforming to SMPTE 125M and ITU-R BT.656. The demultiplexer supports non-PCM data conforming to SMPTE 337M. The demultiplexer does not distinguish between non-PCM data and regular PCM embedded audio, treating both types of data identically. It is up to logic external to the demultiplexer to determine whether the data is PCM or non-PCM and treat it appropriately.

Features

The AES3 audio demultiplexer has these features:

- **Multiple input streams:** A single audio demultiplexer module can demultiplex the audio from multiple video streams. Running at the standard SD video clock rate of 27 MHz, the demultiplexer supports up to six NTSC video streams (five for PAL). Running at 72.5 MHz, the demultiplexer supports up to 16 input video streams. The demultiplexer has a maximum capacity of 16 input video streams. The input video streams can either be synchronous or asynchronous relative to the demultiplexer's clock.
- **24-bit audio support:** The demultiplexer supports both 20-bit and 24-bit audio samples. It automatically detects and processes the extended data packets containing the extra four bits of audio data for each sample. An output from the module indicates whether each audio sample is 20 or 24 bits wide.
- **Audio group support:** The demultiplexer supports all four audio groups and both channel pairs in each audio group, for a total of 16 audio channels per video stream. Inputs to the module allow control over which audio groups and channel pairs are demultiplexed and which are not. This selection is independent for each input video stream when multiple video streams are processed by one demultiplexer module.
- **Indication of channel pairs present in the video stream:** The demultiplexer has "sticky" status outputs indicating which channel pairs in each audio group are present in each input video stream. The time period over which these sticky status bits

are accumulated is user defined and controlled by inputs to the module. There are independent clear inputs for the status bits of each input video stream.

- Full AES3 data support: The demultiplexer outputs the AES3 channel status bit, user data bit, valid bit, and Z frame bit with each audio sample.
- Audio control packet support: The demultiplexer supports audio control packets. The data words from the payload portion of audio control packets are output in their raw format from the module on a separate port along with several status signals. This allows the application to capture all of the data from the control packet or only specific words that are of interest to a particular application.
- Audio packet deletion in video stream: The demultiplexer can optionally modify the video stream to mark audio packets as deleted. This feature is only available when processing a single video stream with the demultiplexer.
- Error detection and handling: The demultiplexer easily handles and reports a variety of error conditions:
 - Each output audio sample is accompanied by a parity error signal generated by checking the parity of the audio sample.
 - The demultiplexer rejects audio packets that have an invalid data count (DC). The data count of audio packets, even non-PCM data packets, must be a multiple of 6 to be valid. The module generates an error code when it rejects an audio packet for this reason.
 - The demultiplexer rejects extended data packets that have an invalid DC. Extended data packets must be of the proper length to match the preceding audio packet. The module generates an error code when it rejects an extended data packet for this reason. All audio samples associated with the rejected extended data packet are output as 20-bit samples.
 - Audio packets that have a checksum error are processed and the audio samples output as normal, but the last sample of the packet is accompanied by a checksum error flag. The module has an output that indicates the first sample of each audio packet. By tracking the start of packet signal and the checksum error signal, it is possible to determine which audio samples are potentially corrupted, allowing the implementation of techniques like soft muting to avoid potentially objectionable audio noise. Optionally, audio packets with checksum errors can be rejected by the demultiplexer. An input pin determines which policy is observed for audio packets with checksum errors.
 - Extended data packets with checksum errors are processed as normal and the audio samples are output as 24-bit audio samples.
 - Incomplete audio packets, those that are prematurely terminated by another ancillary data packet or by the end of the HANC interval, are rejected. The module generates an error code when it rejects a packet for this reason.
 - Incomplete extended data packets, those that are prematurely terminated by another ancillary data packet or by the end of the HANC interval, are rejected. The audio samples from the associated audio packet are output as 20-bit audio samples.
 - The data block number (DBN) from each audio data packet is output with the audio sample. This makes it possible to check the DBN values on the output of the demultiplexer for discontinuities. DBN discontinuities indicate missing audio packets. These discontinuities can be caused by the video stream switching between video sources or by the demultiplexer dropping packets for various errors as indicated above.

Usage Models

The demultiplexer module is very flexible and can be used in a variety of ways. Some typical usage models are described here.

One Video Stream with Video Rate Clock

In [Figure 23-1](#), a single video stream is connected to the input of the demultiplexer. The clock used by the demultiplexer is the standard 27 MHz video clock, and the video is synchronous with this clock. The module processes each and every word of the video stream. The TRS and ADF detection module indicates the presence of these special sequences to the demultiplexer. This mode of operation supports the audio packet deletion feature. Audio packet deletion from the video stream is optional and can easily be disabled.

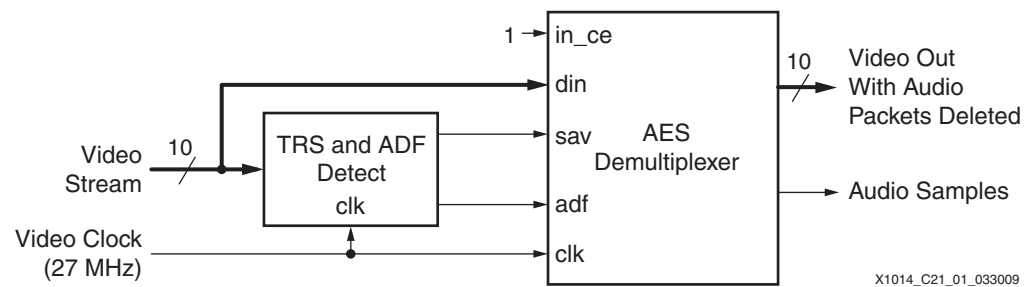


Figure 23-1: One-Input Video Stream with 27 MHz Video Clock

Some power savings can be achieved if the clock enable of the demultiplexer is controlled so that the demultiplexer is only enabled during the HANC portion of each active video line. This can be done by asserting the demultiplexer's `in_ce` port when the EAV is detected and negating it when the SAV is detected. The video packet deletion feature cannot be used if this power saving scheme is implemented.

One Video Stream with Faster Clock

When using oversampling data recovery techniques for receiving SD-SDI with GTP transceivers, it is common for the frequency of the clock from the SD-SDI receiver to be a multiple of 27 MHz, such as 108 MHz (4X), 148.5 MHz (5.5X), or 270 MHz (10X). The SD-SDI oversampling data recovery unit produces a clock enable that is asserted whenever a data word is recovered, throttling the data rate down to 27 MHz. The demultiplexer can be interfaced to the output of such a DRU by connecting the `in_ce` port to the clock enable signal produced by the DRU ([Figure 23-2](#)). This mode supports the audio packet deletion feature.

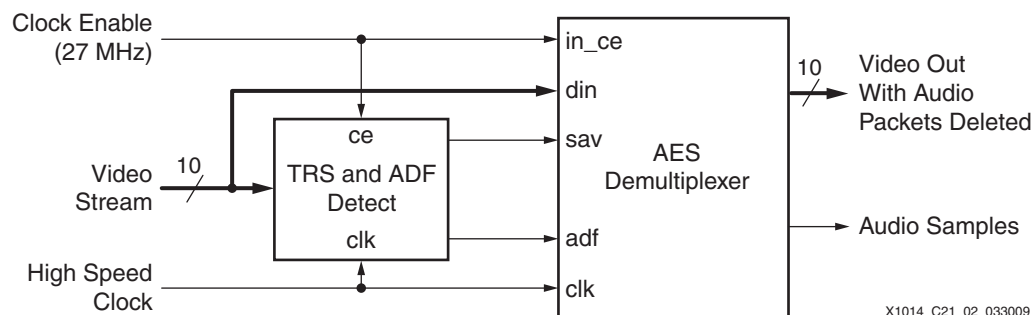
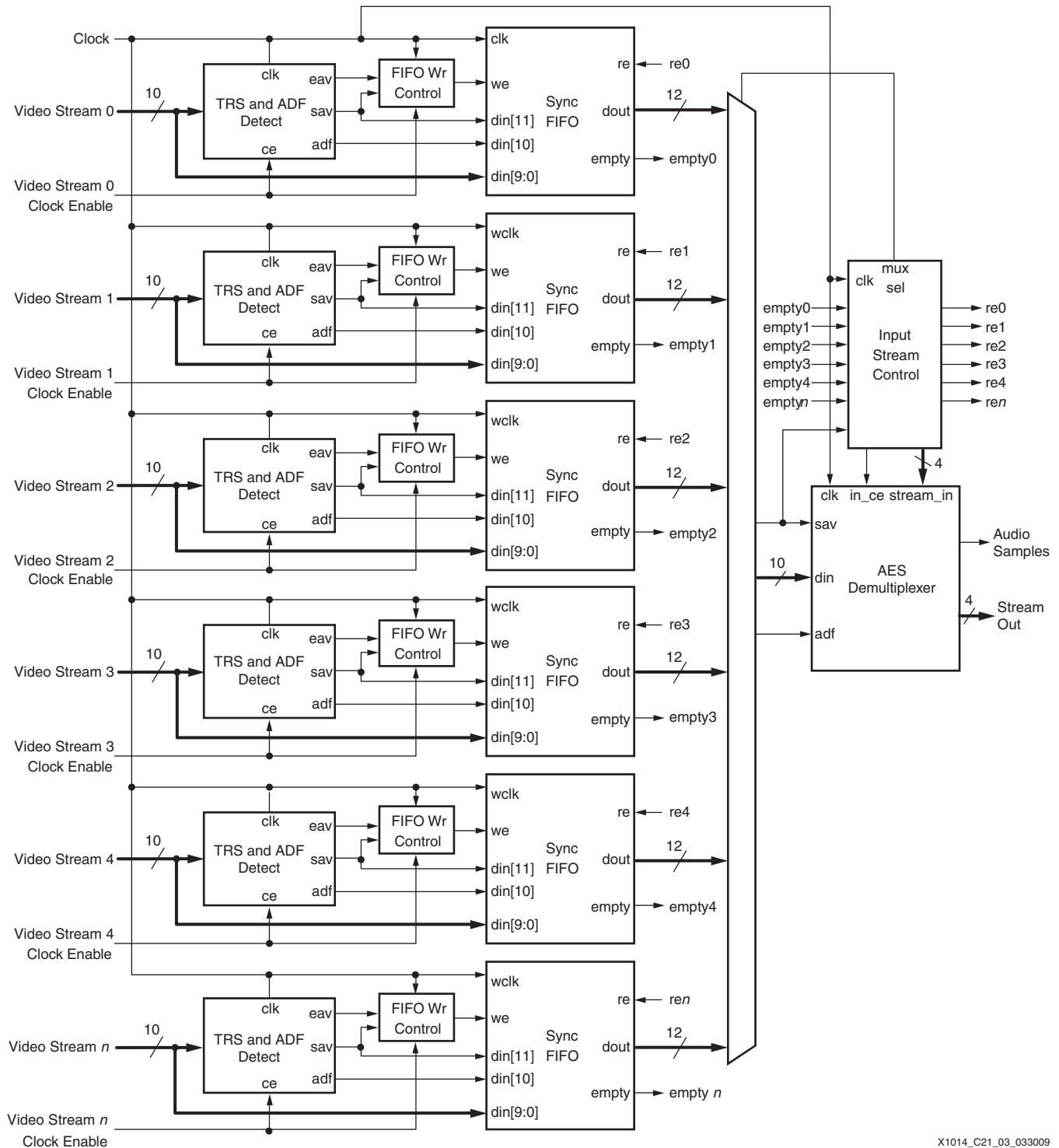


Figure 23-2: One-Input Video Stream with High-Speed Clock and Input Clock Enable

The demultiplexer has an independent output clock enable for the output data. The input and output sides of the demultiplexer use the same clock, but have separate clock enables, allowing the output data rate to be independent of the input data rate.

Multiple Synchronous Video Streams

The demultiplexer can process up to six NTSC or five PAL input video streams when running from a standard 27 MHz video clock, or up to 16 input video streams with a 72.5 MHz clock. Processing multiple input streams requires FIFOs and control logic on the input of the demultiplexer so that the demultiplexer only has to process the HANC data of each video stream. In [Figure 23-3](#), the TRS and ADF detection module and the FIFO write control logic work together to write only the HANC data from each input video stream to the associated FIFO.



X1014_C21_03_033009

Figure 23-3: Multiple Synchronous Video Streams

The input stream control module selects which FIFO is read by the demultiplexer. The demultiplexer cannot switch from one input stream to another arbitrarily. It is designed to only switch after all of the HANC data for one stream has been processed. When the last

word of an SAV is read from a FIFO, the input stream controller selects another video stream, one with a non-empty FIFO. If all of the FIFOs are empty, the demultiplexer is disabled until one of the FIFOs contains data. If the FIFO for the currently selected input stream becomes empty before the end of the HANC (i.e., before the SAV is detected), the demultiplexer stalls until that FIFO has more data available. The input stream controller has a timeout mechanism that prevents a stalled video stream from stopping the processing of data for other streams. The input stream controller can force the demultiplexer to switch to another input stream. However, if the demultiplexer is in the middle of processing an audio packet when it stalls, some data could be lost. The demultiplexer treats this as a prematurely terminated packet. In the reference design, the timeout period is selected by a Verilog parameter or VHDL generic. This period should be set appropriately, taking several factors into account:

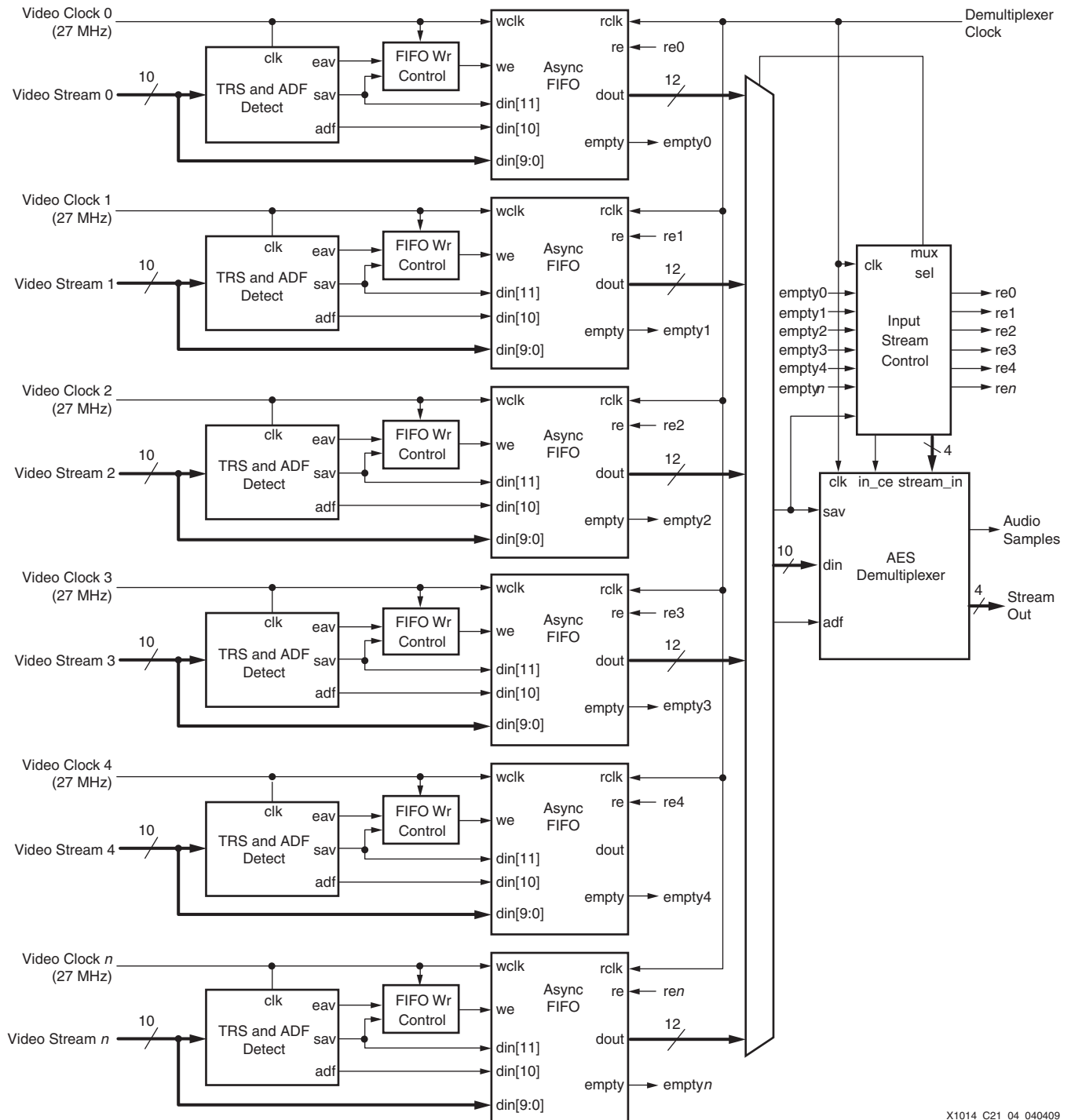
- The amount of time that the application allows the demultiplexer to stall, waiting for a video stream that has stopped, is determined primarily by how much overhead the input clock frequency allows. For example, with six-input NTSC video streams and the demultiplexer running at 27 MHz, the clock is almost 1 MHz faster than is needed to process the six video streams. Thus, there is plenty of allowance for overhead, and the timeout period can be generous. Timeout periods that are too short can cause the demultiplexer to lose audio data.
- The timeout counter is not enabled by `in_ce` because a stalled video stream could possibly cause the clock enable to remain Low. The timeout period must be adjusted to take into consideration the overclocking factor. For example, if a 108 MHz clock is used and `in_ce` is normally asserted once every four clock cycles, the timeout period should be four times longer than if the input clock is 27 MHz.

In [Figure 23-3](#), all of the input video streams are synchronous, running from the same video clock. Each input video stream can, however, have an independent clock enable signal to control writes to the input FIFO.

The input clock also clocks the output side of the FIFOs, the stream control module, and the core of the demultiplexer. Synchronous FIFOs are used in this application because the input and output side of the FIFOs use the same clock.

Multiple Asynchronous Video Streams

The demultiplexer can be used with up to sixteen asynchronous video streams, depending on the frequency of the clock provided to the demultiplexer. This application is very similar to the multiple synchronous video stream application. In this case, however, asynchronous FIFOs are used. As shown in [Figure 23-4](#), each FIFO has an independent video clock that can be a 27 MHz clock or an oversampling clock with a clock enable. The output side of each FIFO is in the same clock domain as the input stream controller and the demultiplexer. The asynchronous FIFOs handle the synchronization of the data between the input and output clock domains.



X1014_C21_04_040409

Figure 23-4: Multiple Asynchronous Video Streams

Clock Requirements for Processing Multiple Video Streams

For both the synchronous and asynchronous cases, the minimum clock frequency required for the demultiplexer when processing multiple input video streams is determined by the number of words in the horizontal blanking interval of the video formats to be supported, the number of video lines per frame, and the video frame rate.

NTSC has 276 words in the blanking interval, including the EAV and SAV. With 525 lines per frame and a frame rate of 29.97 Hz, the required demultiplexer clock frequency is calculated as:

$$276 \text{ words/line} \times 525 \text{ lines/frame} \times 29.97 \text{ frames/sec} \times \text{number of input video streams}$$

This works out to about 4.34 MHz per input video stream. Thus, a 27 MHz clock is sufficient to support six input streams of NTSC video.

PAL has 288 words in the horizontal blanking interval, 625 lines per frame, and 25 frames per second. This works out to exactly 4.5 MHz per input video stream or exactly 27 MHz for six PAL streams. However, a few clock cycles of overhead are used by the demultiplexer when switching between input video streams. When clocked at 27 MHz, the demultiplexer does not have extra clock cycles available for this overhead. Thus, a 27 MHz clock only supports five PAL video streams. Some overhead should also be included to compensate for shortened lines that might occur during synchronous switching events. Even though there are usually no audio packets in the HANC on the lines surrounding the synchronous switching interval, the demultiplexer still processes all of the HANC words on these lines.

If other video standards, such as widescreen NTSC and PAL, are supported by the application, the required clock frequency must be calculated using the size of the HANC interval of these additional video standards.

Modules

The basic audio demultiplexer module is called `sd_aes_demux`. This module has all the ports necessary to support 16 video streams but does not have the support modules such as the input stream controller, FIFOs, and TRS and ADF detector.

Because this module has so many ports to support 16 video streams, it is unwieldy when used with a single input stream. The module called `sd_aes_demux_1` is a wrapper around the `sd_aes_demux` module, exposing only those I/O ports required to support a single video stream.

Several wrapper modules are optimized to support different numbers of video streams. The module `sd_aes_demux_4` supports four video streams and includes the input stream controller. For convenience, additional wrapper modules are provided that also include the input FIFOs, FIFO input control logic, and TRS and ADF detection.

The `sd_aes_demux_4_sync` module provides everything necessary to support four synchronous input streams. In addition, the `sd_aes_demux_4_async` module provides everything necessary to support four asynchronous input streams. These two modules are wrappers around the `sd_aes_demux_4` module, which in turn is a wrapper around `sd_aes_demux`. Equivalent wrapper modules support 8- and 16-input streams.

For applications with other than 1-, 4-, 8-, or 16-input streams, one of the wrapper modules can be modified to support the required number of video streams. Alternatively, a wrapper module supporting more streams than required can be used with the unused input ports hardwired appropriately.

Table 23-1 lists all of the main modules included in the reference design.

Table 23-1: List of Modules

Module	Description
sd_aes_demux	This is the main demultiplexer module with support for up to 16 input streams.
sd_aes_demux_1	This module provides the ports necessary to support one input stream. The TRS and ADF detection module is not included.
sd_aes_demux_4	This module provides the ports necessary to support four input streams. Input FIFOs and TRS and ADF detection are not included.
sd_aes_demux_4_sync	This module supports four synchronous input streams. It includes the synchronous input FIFOs and TRS and ADF detection module.
sd_aes_demux_4_async	This module supports four asynchronous input streams. It includes the asynchronous input FIFOs and TRS and ADF detection module.
sd_aes_demux_8	This module provides the ports necessary to support eight input streams. Input FIFOs and TRS and ADF detection are not included.
sd_aes_demux_8_sync	This module supports eight synchronous input streams. It includes the synchronous input FIFOs and TRS and ADF detection module.
sd_aes_demux_8_async	This module supports eight asynchronous input streams. It includes the asynchronous input FIFOs and TRS and ADF detection module.
sd_aes_demux_16	This module provides the ports necessary to support sixteen input streams. Input FIFOs and TRS and ADF detection are not included.
sd_aes_demux_16_sync	This module supports sixteen synchronous input streams. It includes the synchronous input FIFOs and TRS and ADF detection module.
sd_aes_demux_16_async	This module supports sixteen asynchronous input streams. It includes the asynchronous input FIFOs and TRS and ADF detection module.
sd_aes_demux_buffer_512	This module is instantiated in sd_aes_demux. It provides the block RAMs required for the 512-deep sample buffer. This is the standard buffer size.
sd_aes_demux_buffer_2K	This module is an example of how to implement a larger sample buffer. It is 2,048 samples deep and can optionally replace the 512-deep sample buffer.
sd_aes_demux_infifo_ctrl	This module controls writing of video data to an input FIFO so that only HANC data is written. One instance of this module is included for each FIFO in the demultiplexer modules that include input FIFOs.

Table 23-1: List of Modules (Cont'd)

Module	Description
<code>sd_aes_demux_infsm</code>	This module implements the input side state machine in <code>sd_aes_demux</code> .
<code>sd_aes_demux_instream_ctrl</code>	This is the input stream controller module. It is included in all multi-stream demultiplexer modules.
<code>sd_aes_demux_outfsm</code>	This module implements the output side state machine in <code>sd_aes_demux</code> .
<code>sd_aes_pkt_del</code>	This module implements the audio packet deletion feature. It is included in the <code>sd_aes_demux_1</code> module.
<code>sd_aes_present_flags</code>	This module is used by <code>sd_aes_demux</code> to implement the channel pair detection flags.
<code>sd_aes_pri_encoder_16</code>	This module is part of the arbiter in the <code>sd_aes_demux_instream_ctrl</code> module.
<code>sd_aes_trs_adf_detect</code>	This module implements TRS (EAV and SAV) and ADF detection. An instance of this module is included for each video stream in the multi-stream demultiplexer modules listed above as including TRS and ADF detection.

I/O Port Description

This section describes the I/O ports of the audio demultiplexer wrapper modules. Only those ports for the `sd_aes_demux_1`, `sd_aes_demux_x_async`, and `sd_aes_demux_x_sync` modules are described.

Clock and Control Signals

Table 23-2 shows the input signals that provide clocks, clock enables, resets, etc.

Table 23-2: Clock and Control Signals

Port	Direction	Width	Description
<code>clk</code>	In	1	This is the clock for the demultiplexer. For all modules except those with asynchronous FIFOs, all inputs are synchronous with this clock. All outputs from the demultiplexer are also synchronous with this clock.
<code>rst</code>	In	1	This is an asynchronous reset input. To ensure that the state machines in the demultiplexer exit reset cleanly, it is recommended that the falling edge of the <code>rst</code> signal meet setup and hold times on all flip-flops clocked by the <code>clk</code> input. All critical control logic in the demultiplexer exits FPGA configuration in a known good state, so for most applications, the <code>rst</code> input can be hardwired Low.

Table 23-2: Clock and Control Signals (Cont'd)

Port	Direction	Width	Description
in_ce	In	1	<p>This is the input-side clock enable for the demultiplexer. For <code>sd_aes_demux_1</code>, the demultiplexer must be given one input video word on every clock when <code>in_ce</code> is asserted.</p> <p>The multi-stream modules have individual write enables for each stream, but they also have this <code>in_ce</code> port. For these multi-stream modules, <code>in_ce</code> only controls the clock rate of the input state machine in the demultiplexer. It is usually tied High for the multi-stream modules, but can be used to throttle the demultiplexer's clock rate. For example, if <code>clk</code> is 270 MHz, it could be difficult to meet timing in the demultiplexer, so <code>in_ce</code> could be asserted every other clock cycle to throttle the clock rate in the demultiplexer to 135 MHz. This would make it easier to meet timing or possibly lower power consumption.</p>
out_ce	In	1	<p>This clock enable input controls when the output state machine in the demultiplexer is clocked. All outputs, except as noted, change only when <code>out_ce</code> is asserted. This clock enable is primarily used to throttle the data rate of the output state machine to a manageable frequency to meet timing.</p>
ignore_cs_err	In	1	<p>If this input is High, audio samples are processed normally when the demultiplexer detects an audio packet checksum error. If this input is Low, all audio samples in any audio packet with a checksum error are rejected and are not output from the demultiplexer.</p>

The demultiplexer has two state machines, one on the input side and the other on the output side. These two state machines are independent of each other, although they share a common clock. The input state machine and the output state machine each have their own clock enable signals, called `in_ce` and `out_ce`. The input and output sides of the demultiplexer can have different clock rates by asserting these two clock enables at different rates. Most of the outputs from the demultiplexer are controlled by `out_ce`. However, some outputs are generated by the input state machine and are controlled by `in_ce`. These exceptions are noted in the descriptions of the ports.

Audio samples must be read from the output of the demultiplexer fast enough that the internal sample buffer does not overflow. Samples are read by asserting both `out_ce` and `output_ack` when the `output_ready` output is asserted. The output state machine takes several clock cycles to output each audio sample. In fact, it takes seven clock cycles to output one sample pair (two samples from the same channel pair), and `out_ce` must be asserted during all seven of the clock cycles. The output state machine discards audio samples from channel pairs that are not enabled for output. The output state machine requires three clock cycles with `out_ce` asserted to discard an audio sample from a channel pair that is not enabled. The `out_ce` signal should not be used to clock the output state machine at the audio sample rate because it must be asserted faster than the audio sample rate. The primary use of this signal is to throttle the output state machine clock rate down to a manageable frequency for timing purposes if the `clk` input is a high-frequency clock.

Input Video Streams

The video input signals vary considerably between the single-stream module, the multi-stream asynchronous modules, and the multi-stream synchronous modules. [Table 23-3](#), [Table 23-4](#), and [Table 23-5](#) cover these three cases, respectively.

Table 23-3: Input Video Stream for Single-Stream Module (`sd_aes_demux_1`)

Port	Direction	Width	Description
adf	In	1	This input must be asserted during the third word of every 3-word ADF sequence.
sav	In	1	This input must be asserted during the XYZ (fourth) word of every 4-word SAV sequence.
din	In	10	This is the digital video word input. The <code>sd_aes_demux_1</code> module does not contain a TRS and ADF detector. The <code>sd_aes_trs_adf_detect</code> module can be used to provide the adf and sav input signals for this module, or these inputs can be driven by the SD-SDI receiver if it provides these signals with the correct timing required by the demultiplexer. A video input word must be provided on din, along with valid adf and sav inputs, on every clock cycle when <code>in_ce</code> is asserted.

Each asynchronous input video stream has the set of four ports shown in Table 23-4. Separate write clocks and write enables are provided. A valid video data word must be present on the `strmn_din` port every clock cycle in which `strmn_we` is asserted ($n = 4, 8, \text{ or } 16$). A TRS and ADF detector is included in these modules for each video stream input.

Table 23-4: Input Video Streams for Asynchronous Multi-Stream Modules

Port	Direction	Width	Description
<code>strmn_wclk</code>	In	1	For each input stream, a write clock is provided. These clocks are considered to be asynchronous to each other and to the main demultiplexer clock (<code>clk</code>).
<code>strmn_we</code>	In	1	For each input stream, a write enable input is provided. A video data word must be provided on the <code>strmn_din</code> port every clock cycle in which the write enable is asserted.
<code>strmn_din</code>	In	10	These are the video data input ports for each stream.
<code>strmn_full</code>	Out	1	These are the FIFO full outputs for each stream.

Each synchronous input video stream has the set of three ports shown in Table 23-5. All of these ports are synchronous to the `clk` input. A valid data word must be present on the `strmn_din` port every rising edge of `clk` in which `strmn_we` is asserted. A TRS and ADF detector is included in these modules for each video stream input.

Table 23-5: Input Video Streams for Synchronous Multi-Stream Modules

Port	Direction	Width	Description
<code>strmn_we</code>	In	1	For each input stream, a write enable input is provided. A video data word must be provided on the <code>strmn_din</code> port every clock cycle in which the write enable is asserted.
<code>strmn_din</code>	In	10	These are the video data input ports for each stream.
<code>strmn_full</code>	Out	1	These are the FIFO full outputs for each stream.

Audio Sample Output Ports

The audio samples are output from the demultiplexer one at a time as they are received and processed. Each audio sample is accompanied by additional signals that provide information about the audio sample.

Whenever an audio sample is present on the output, the output_ready signal is asserted High. The audio sample remains on the output port and the output_ready signal remains High until the next rising edge of clk when both out_ce and output_ack are asserted. The output_ack input serves as an output handshaking signal. If this handshaking mechanism is not required, output_ack must be tied High.

No matter how many input video streams the demultiplexer handles, there is only one output port, and all audio samples from all input video streams are output on this port. When handling multiple video streams, logic and a buffer on the output of the demultiplexer module might be required for each video stream to send each audio sample to the appropriate audio datapath.

The chan, channel_pair, audio_group, and stream_out ports serve to identify the audio channel, audio channel pair, audio group, and input video stream of the audio sample. The audio_group, channel_pair, and chan bits combine to form a 4-bit audio channel number. Each video stream can have up to 16 channels. For the multi-channel modules, the stream_out signal indicates which input video stream is contained the audio sample. All the outputs listed in Table 23-6 are controlled by the out_ce clock enable.

Table 23-6: Audio Sample Output Ports

Port	Direction	Width	Description
output_ready	Out	1	This output is asserted High when an audio sample is present on the output of the demultiplexer.
output_ack	In	1	The demultiplexer keeps an audio sample on the output port until both output_ack and out_ce are High during the rising edge of clk.
sf	Out	1	This output indicates which audio subframe the sample came from in the embedded stream. Generally, the chan output is more useful than sf, but this output can provide some debugging information.
audio	Out	24	This is the output port for the audio sample. For 20-bit audio, the sample is left justified and the least significant 4 bits of this port are zeros.
audio_is_24b	Out	1	1 = 24-bit audio sample 0 = 20-bit audio sample.
stream_out	Out	2/3/4	This output indicates which input video stream produced the current audio output sample. This output port is not present on the sd_aes_demux_1 module. The width of this port is dependent on how many input video streams are supported. The port is 2 bits wide for the 4-stream modules, 3 bits wide for the 8-stream modules, and 4 bits wide for the 16-stream modules.
audio_group	Out	2	This output indicates the embedded audio group of the audio sample.
channel_pair	Out	1	This output indicates the channel pair of the audio sample.
chan	Out	1	This output indicates the audio channel (within the channel pair) of the audio sample.
z	Out	1	This is the AES3 Z frame indicator. This bit is asserted High on the first audio sample of each 192-sample block.
c	Out	1	This is the AES3 channel status bit.
u	Out	1	This is the AES3 user data bit.
v	Out	1	This is the AES3 valid bit. This bit is Low if the audio sample is valid, and High if it is not. This bit is the V bit from the audio data packet and is not derived from the channel valid bits in the audio control packet.

Table 23-6: Audio Sample Output Ports (Cont'd)

Port	Direction	Width	Description
dbn	Out	8	This is the data block number from the audio data packet contained in the audio sample.
parity_err	Out	1	This output is High if a parity error is detected in the audio sample.

Channel Pair Present Flags

The demultiplexer produces a set of output flags that indicate which audio channel pairs are detected in each input stream. These flags are “sticky,” i.e., a flag is set when an audio sample is detected for the corresponding channel pair. The flag remains set until cleared by the associated `clr_present` input.

For each video stream, an 8-bit output port with one bit is allocated to each of the eight channel pairs in the stream. Each video stream also has an individual `clr_present` input so the flags can be managed on a per-stream basis (see [Table 23-7](#) and [Table 23-8](#)).

The flags are independent of the demultiplexer enable status of the channel pair (see [Channel Pair Demultiplexer Control Ports](#), page 531). If a channel pair is detected in a video stream, its present flag is set, even if that channel pair is currently being rejected by the demultiplexer control mechanism. The outputs are all controlled by the `out_ce` clock enable.

Table 23-7: Channel Pair Present Flags for `sd_aes_demux_1`

Port	Direction	Width	Description
clr_present	In	1	All present flags are reset to zero on the rising edge of <code>clk</code> when <code>clr_present</code> and <code>out_ce</code> are both asserted.
present	Out	8	These are the channel-pair-present flags. A channel pair has been detected in the input video stream when the corresponding bit on this port is High.

Table 23-8: Channel Pair Present Flags (Multi-Stream Modules)

Port	Direction	Width	Description
clr_present	In	n	All present flags for a particular stream are reset to zero on the rising edge of <code>clk</code> when the corresponding bit of this input port and <code>out_ce</code> are both asserted High. There is one bit for each supported video stream. The <code>clr_present[0]</code> bit clears the flags for stream 0, <code>clr_present[1]</code> clears the flags for stream 1, and so on.
strmn_present	Out	8	These are the channel-pair-present flags. Each supported input stream has a corresponding <code>strmn_present</code> output port. A channel pair has been detected in the input video stream when the corresponding bit on this port is High.

[Table 23-9](#) shows how the eight bits of the present flag output port correspond to audio groups and channel pairs.

Table 23-9: Channel Pair Present Flag Vector Mapping

strmn_presentBit	Audio Group	Channel Pair	Audio Channels
0	1	1	1, 2
1	1	2	3, 4
2	2	1	5, 6

Table 23-9: Channel Pair Present Flag Vector Mapping (Cont'd)

strmn_presentBit	Audio Group	Channel Pair	Audio Channels
3	2	2	7, 8
4	3	1	9, 10
5	3	2	11, 12
6	4	1	13, 14
7	4	2	15, 16

Channel Pair Demultiplexer Control Ports

The demultiplexer outputs samples only for those audio channel pairs that are enabled. An 8-bit input port allows each audio channel pair to be individually enabled or disabled on a per-stream basis. Each bit in the chpair_demux_en port enables one channel pair in the video stream (see Table 23-10.) Mapping of channel pairs to the bits of the chpair_demux_en port is identical to the mapping used for the channel pair present flags shown in Table 23-9.

Table 23-10: Channel Pair Filtering Ports

Port	Direction	Width	Description
chpair_demux_en	In	8	This input port determines which channel pairs are to be demultiplexed and which are not. If the bit for a channel pair is High, the channel pair is demultiplexed and output, if present, in the video stream. If the bit for a channel pair is Low, the channel pair is not demultiplexed and no samples for this channel pair are output by the demultiplexer.
current_stream	Out	2/3/4	This output port indicates which video stream is being processed by the output state machine in the demultiplexer. It is intended to be used to select the appropriate data for the chpair_demux_en input port so that different channel pairs can be enabled for each input video stream. This port is not available on the sd_aes_demux_1 module. The port is 2 bits wide on the 4-stream modules, 3 bits wide on the 8-stream modules, and 4 bits wide on the 16-stream modules.

There is only one 8-bit channel pair demultiplexer enable port, even for the multi-stream modules. However, it is possible to have different channel pairs enabled for each different video stream. The demultiplexer indicates which video stream is currently being processed on the current_stream output port. By changing the data on the chpair_demux_en port whenever the current_stream port changes, different channel pairs can be enabled for each video stream. The simple way to do this is with a multiplexer, as shown in Figure 23-5. Alternatively, the filter bits can be stored in a small distributed dual-port RAM with the current_stream port providing the read address to the RAM (see Figure 23-6). If the same filter selection is desired for all input video streams, the current_stream port can be ignored. The chpair_demux_en input value must be updated during the same clock cycle in which the current_stream port changes.

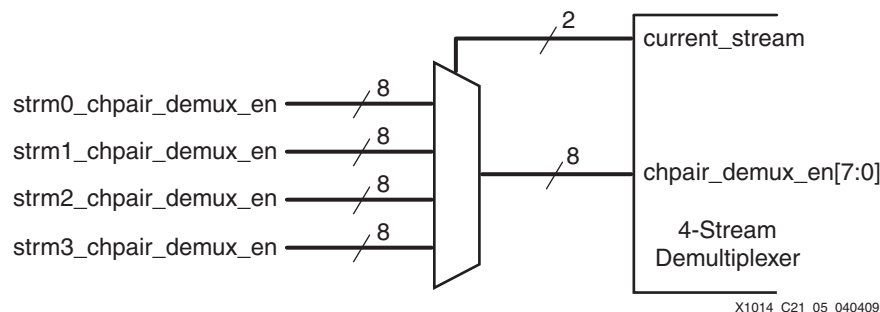


Figure 23-5: Driving the Channel Pair Demultiplexer Enables with a Multiplexer

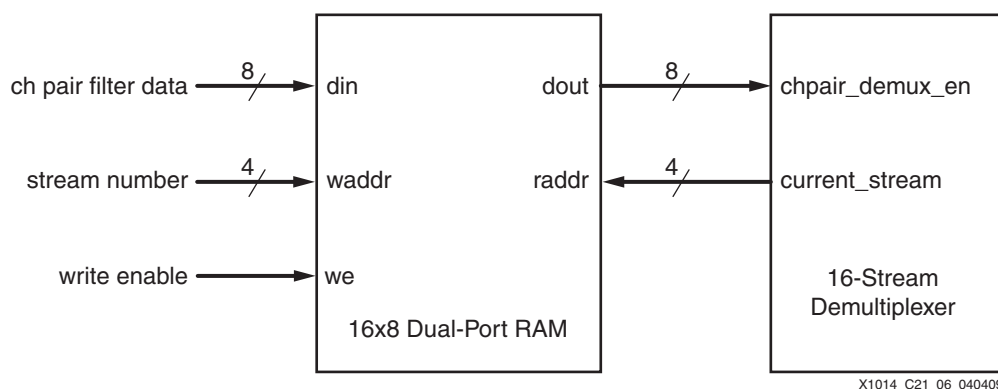


Figure 23-6: Driving the Channel Pair Demultiplexer Enables with a RAM

Input Packet Error Ports

The demultiplexer provides several outputs that indicate the error status of detected audio packets, as shown in Table 23-11.

Table 23-11: Input Packet Status Ports

Port	Direction	Width	Description
drop_pkt_err	Out	1	This signal is asserted High if an audio or extended data packet is rejected because of an error.
drop_pkt_code	Out	3	The code on this port is valid when drop_pkt_err is High and indicates the reason the packet was rejected: 000 = No error 001 = Audio packet rejected due to invalid data count 010 = Extended data packet rejected due to invalid data count 011 = Reserved 100 = Audio packet ended prematurely by SAV 101 = Audio packet ended prematurely by ADF 110 = Audio packet rejected due to checksum error 111 = Reserved
pkt_cs_err	Out	1	This output is asserted on the last audio sample of an audio packet when a checksum error is detected in the packet.
pkt_start	Out	1	This output is asserted on the first audio sample of each audio packet.

The `drop_pkt_err` and `drop_pkt_code` outputs are generated by the input state machine and are controlled by `in_ce`. After `drop_pkt_err` is asserted, it remains asserted until the next rising edge of `clk` with `in_ce` High.

The `pkt_cs_err` and `pkt_start` outputs are generated by the output state machine and are controlled by `out_ce` and `output_ack`. They are valid until the audio sample present on the output of the demultiplexer is acknowledged by `output_ack` asserted High with `out_ce` High.

Checksum errors in audio data packets are indicated in two different ways, depending on the checksum error policy selected by the `ignore_cs_errs` input.

If `ignore_cs_errs` is High, audio samples from audio data packets with checksum errors are output as normal and the checksum error is indicated by assertion of the `pkt_cs_err` signal during the last audio sample of the packet. It is possible to determine which samples are potentially affected by the checksum error by monitoring both the `pkt_start` and `pkt_cs_err` signals. The `pkt_start` output is asserted when the first audio sample of each packet is output from the demultiplexer. The demultiplexer does not interleave samples from different packets or different streams on the output. Thus, after `pkt_start` is asserted, all the audio samples output from the demultiplexer until the next assertion of `pkt_start` come from the same packet. When `pkt_cs_err` is asserted, all samples previously output back to when `pkt_start` was asserted are suspect. See Figure 23-7 for details.

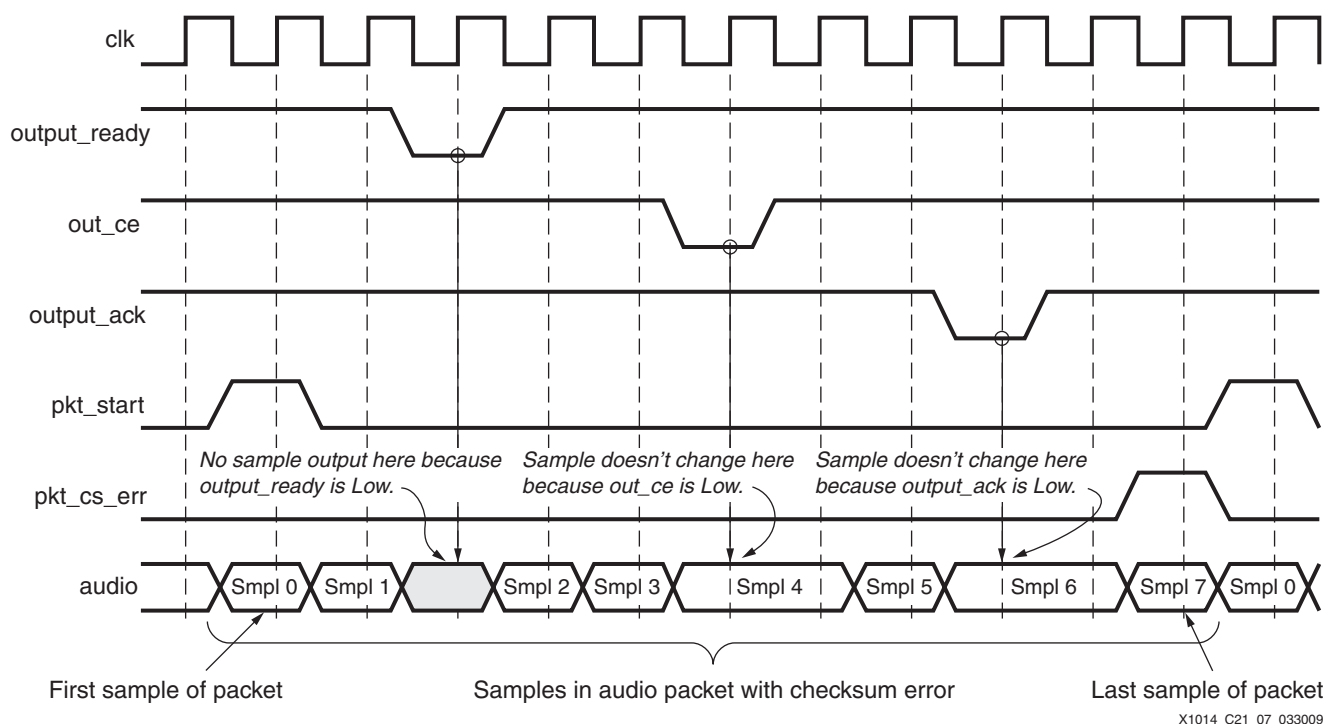


Figure 23-7: Timing of `pkt_start` and `pkt_cs_err` Signals

If `ignore_cs_errs` is Low, the demultiplexer rejects any audio packet with a checksum error and the audio samples contained in the packet are not output. These rejected packets are indicated by assertion of `drop_pkt_err` and a code of 110 on `drop_pkt_code`.

Audio Control Packet Ports

The demultiplexer detects and outputs audio control packets. It does not interpret or format the audio control packet data in any way. The words of the audio control packet are output sequentially by the demultiplexer along with some status signals, including a data word count indicating which data word of the control packet is currently on the output port. Only the UDWs in the payload portion of the audio control packet are output by the demultiplexer. These ports are all controlled by the input state machine and are thus controlled by the `in_ce` clock enable (see Table 23-12).

Table 23-12: Audio Control Packet Ports

Port	Direction	Width	Description
<code>ctrl_pkt_strobe</code>	Out	1	This output is High whenever a data word from an audio control packet is on the <code>ctrl_pkt_data</code> port.
<code>ctrl_pkt_data</code>	Out	10	The audio control packet data words are output on this port.
<code>ctrl_pkt_group</code>	Out	2	This output port indicates which audio group the audio control packet belongs to.
<code>ctrl_pkt_stream</code>	Out	2/3/4	For the multi-stream demultiplexers, this output port indicates which input stream the audio control packet was found in. This port is 2 bits wide for the 4-stream modules, 3 bits wide for the 8-stream modules, and 4 bits wide for the 16-bit demultiplexer. This port is not present on the <code>sd_aes_demux_1</code> module.
<code>ctrl_pkt_word</code>	Out	5	This port indicates the word number of the current UDW on the <code>ctrl_pkt_data</code> port. The value on this port is zero when the first UDW of an audio control packet is on the <code>ctrl_pkt_data</code> port and increments by one with each UDW from the control packet.

Control packet output timing is shown in Figure 23-8.

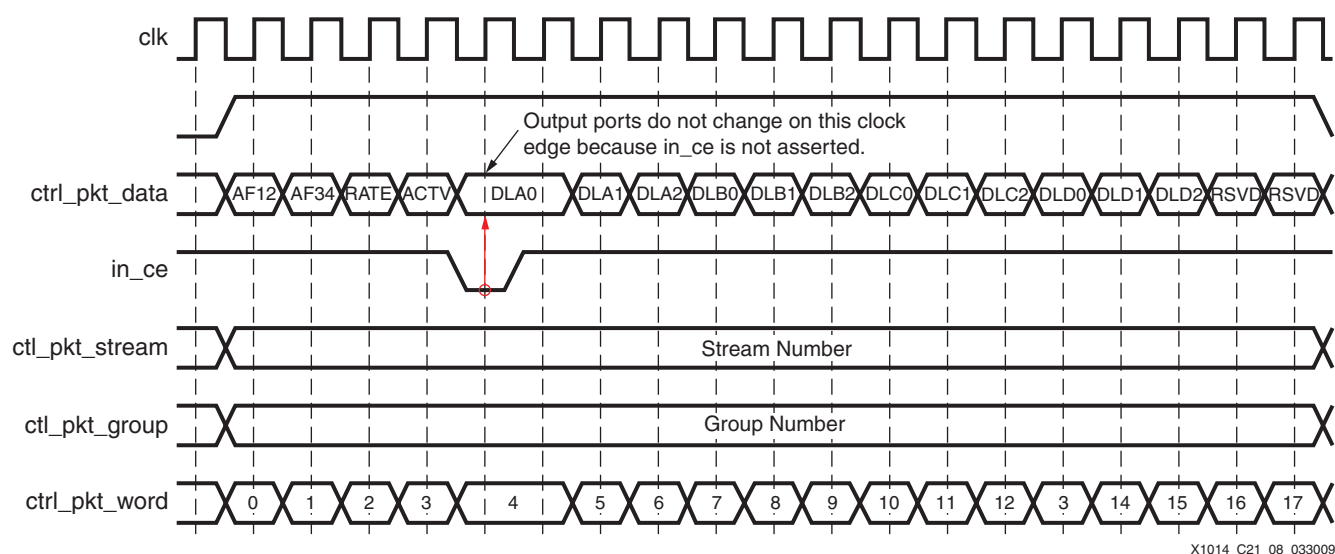


Figure 23-8: Audio Control Packet Output Timing

Audio Packet Deletion Ports

The `sd_aes_demux_1` module has two ports not present on the multi-stream modules to support the audio packet deletion feature. An input port called `del_groups` enables audio packet deletion individually by audio group. The port has one input bit for each of the four audio groups. The LSB of this port controls packet deletion for audio group 1. The MSB controls packet deletion for audio group 4. A High on a bit in the `del_groups` port enables deletion of the audio data packet, audio control packets, and extended data packets for that audio group.

The audio packet deletion function is independent of the audio sample demultiplexer feature. It is possible to enable different audio groups for demultiplexing than for deletion because the `chpair_demux_en` port controls demultiplexing and the `del_groups` port controls deletion.

After audio packet deletion, the modified video is output on the `video_output` port. This port is controlled by the `in_ce` clock enable so that a video word is output every clock cycle when `in_ce` is High. One clock cycle of latency (counting only those clock cycles with `in_ce` asserted) is added to the video between the video input port (`din`) and the `video_out` port (see [Table 23-13](#)).

Table 23-13: Audio Control Packet Ports

Port	Direction	Width	Description
<code>del_groups</code>	In	4	A High input bit enables audio packet deletion for the corresponding audio group. Bit 0 controls audio group 1. Bit 3 controls audio group 4.
<code>video_out</code>	Out	10	After audio packet deletion, the modified video is output on this port. A video word is output each clock cycle when <code>in_ce</code> is asserted.

Audio data packets, audio control packets, and extended data packets are deleted by changing their DID words to `0x180`. This is the DID value for a deleted packet. The checksum word is also modified so that checksum is correct for the deleted packet. If the video stream contains an SMPTE RP 165 error detection and handling (EDH) packet, the full-field CRC value in the EDH packet is not correct after audio packet deletion. The video stream should be reprocessed by an EDH processor to calculate and insert new EDH values after audio packet deletion.

Parameters

The reference design has several Verilog parameters or VHDL generics that affect the operation of the demultiplexer. The parameter/generic called `STRICT_EXT_PKT_MODE` determines how strictly the demultiplexer enforces the rule that an extended data packet must immediately follow its associated ancillary data packet. If this parameter is 1, the demultiplexer requires that the extended data packet immediately follow the audio data packet with no HANC words in between. If this parameter is 0, the demultiplexer implements a more liberal policy and waits until the end of the HANC interval for a potential extended data packet. In both cases, the extended data packet must be the next ancillary data packet in the HANC interval after the associated audio data packet. The only difference is that extraneous data words can exist between the audio data packet and the extended data packet when the parameter is 0.

Setting `STRICT_EXT_PKT_MODE` to 0 does slightly affect the timing of the demultiplexer. If there are no extended data packets, the samples from the last audio data packet in the HANC interval are not output until the end of the HANC interval is detected. If this parameter is 1, the audio samples from an audio data packet are output immediately as

20-bit samples if an extended data packet does not immediately follow an audio data packet.

The multi-stream demultiplexer modules have another parameter/generic called `TIMEOUT_CNTR_BITS` that sets the stream timeout period. The timeout counter is a binary counter whose width is set by this parameter. The timeout period always elapses when the timeout counter reaches its terminal count with all bits High. The input stream timeout feature is discussed in further detail in [Multiple Synchronous Video Streams](#), page 520.

Sample Buffer Size

Audio samples embedded in a video stream arrive in bursts, but are often consumed on the output of a demultiplexer at a fixed rate equal to the audio sample rate. Buffers are required in any audio demultiplexer to smooth out this difference. The SMPTE 272M standard recommends a minimum sample buffer depth of 80 samples per active channel. A total buffer size of 1,280 samples is required to support all 16 possible audio channels in one stream and 20K samples for all 16 audio channels in all 16 input video streams. However, few, if any, video streams have 16 active audio channels.

The `sd_aes_demux` module has an internal audio sample buffer located between the input and output state machines. This buffer is not designed to meet the SMPTE 272M recommended sample buffer size. Its default size is 512 samples. This is sufficient to handle 16 channels in 16 input streams if the samples are taken from the demultiplexer at a sufficiently high rate. For example, in NTSC video, it is possible to have a maximum of about 82 samples in one HANC. Thus, in the case of a demultiplexer handling 16 video channels, it must be possible to read 1,312 samples (82×16), if available, from the demultiplexer during every video line to guarantee that the buffer does not overflow. The demultiplexer does not check for overflows of the internal sample buffer, so data is lost if an overflow occurs.

The 512-sample internal buffer is sufficient to meet the SMPTE 272M buffer depth recommendations for six active audio channels. If more active channels are supported by an application, either the internal sample buffer in the demultiplexer must be increased or additional sample buffers outside of the demultiplexer must be used. In a multi-stream application, it is generally preferred to use individual audio sample buffers external to the demultiplexer for each input stream. This is usually more efficient than expanding the internal buffer that carries extra information used internally by the demultiplexer.

All audio samples present in an input video stream, even those that are not output from the demultiplexer because they are not enabled, are stored in the sample buffer. It is possible to increase the size of the internal sample buffer. The default 512-sample buffer uses two 18K block RAMs, one for the audio data and one for the extended data. The audio data RAM is configured as 512×36 . The extended data RAM is configured as 1024×18 , but only one quarter of the extended data RAM is used. Doubling the size of the buffer to 1,024 samples requires adding one additional block RAM for the audio samples, but an additional extended data RAM is not required.

A total sample buffer depth of 20K samples is required to meet the SMPTE 272M buffer depth required for 6 active channels in 16 input video streams. Such a sample buffer requires a total of 60 block RAMs, 50 for the audio data and 10 for the extended data. It might be more efficient to place the sample buffers on the output of the demultiplexer rather than expanding the internal sample buffer.

It is easy to change the width of the buffer address paths in the demultiplexer to accommodate larger buffers. A single Verilog local parameter or VHDL constant called

ADR_WIDTH in the `sd_aes_demux` module specifies the number of address bits used for the sample buffer. The two block RAMs that make up the buffer are located in a separate module called `sd_aes_demux_buffer_512`. This module can be modified to create a larger sample buffer. A 2K-sample buffer module, `sd_aes_demux_buffer_2K`, is included as an example of how to implement a larger sample buffer. If this 2K-sample buffer is used, the ADR_WIDTH constant in `sd_aes_demux` must be set to 11.

FPGA Resource Requirements

Table 23-14 shows the FPGA resources required to implement the audio demultiplexer for the single-stream, 4-stream, 8-stream, and 16-stream modules. These results were obtained using ISE® software, version 8.1 SP1, and XST for synthesis. All results are with the default sample buffer size of 512.

Table 23-14: FPGA Resources

Module Name	LUTs	Flip-Flops	Block RAMs
<code>sd_aes_demux_1</code>	261	195	2
<code>sd_aes_demux_4_sync</code>	907	491	6
<code>sd_aes_demux_4_async</code>	1115	727	6
<code>sd_aes_demux_8_sync</code>	1558	782	10
<code>sd_aes_demux_8_async</code>	1974	1254	10
<code>sd_aes_demux_16_sync</code>	2783	1351	18
<code>sd_aes_demux_16_async</code>	3615	2295	18

Theory of Operation

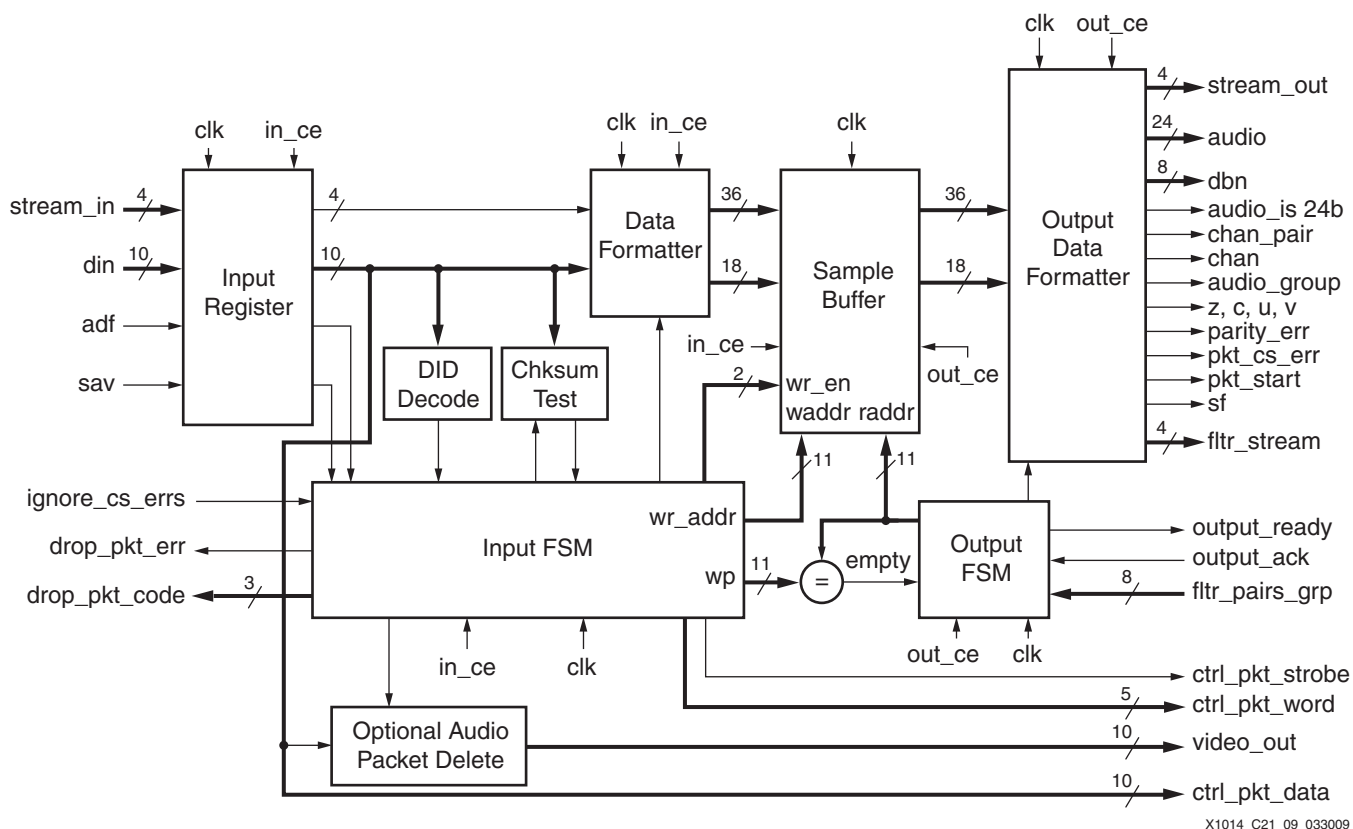
This section provides the theory of operation for the SD audio demultiplexer reference design.

Overview

The core demultiplexer module, `sd_aes_demux`, has three main pieces: an input state machine, an output state machine, and an audio sample buffer located between the input and output state machines, as shown in Figure 23-9. The sample buffer has data formatting logic on its input and output sides. The input state machine controls the input data formatting logic and writes to the sample buffer. The output state machine controls reads from the sample buffer and the output formatting logic.

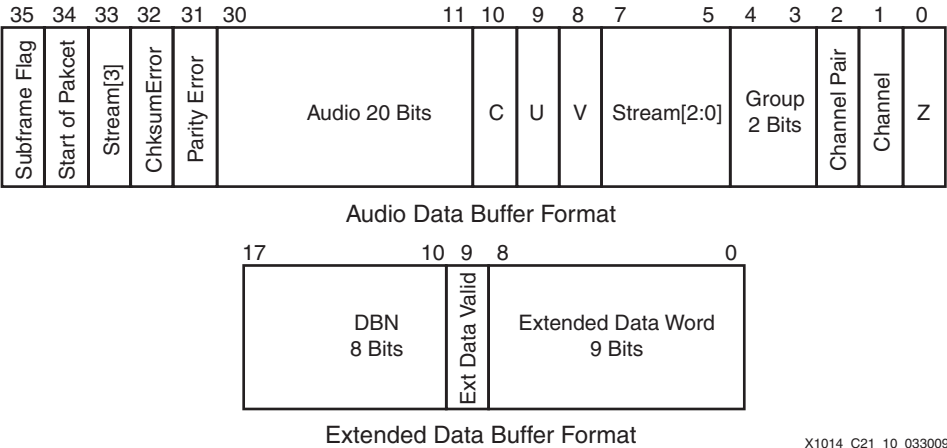
The sample buffer is a circular buffer. When the read pointer from the output state machine equals the write pointer from the input state machine, the buffer is empty.

The input state machine processes audio data packets and extended data packets and writes the audio samples into the sample buffer. When an audio data packet and its associated extended data packet have been processed and all audio samples from the packet have been written to the sample buffer, the input state machine changes the write pointer to point to the next empty buffer location past the samples that have just been written. The output state machine sees that the buffer is not empty, reads each audio sample from the sample buffer, and outputs the samples from the demultiplexer.

Figure 23-9: Block Diagram of `sd_aes_demux` Module

The input state machine writes all audio samples to the sample buffer, regardless of whether the samples are from channel pairs enabled for output or not. As the output state machine reads each sample from the sample buffer, it checks to see if the sample is from a channel pair that is enabled for output. If the channel pair is enabled, the sample is output from the demultiplexer. If the channel pair is disabled, the sample is discarded.

The sample buffer has two separate RAMs, one holding the audio data and the other holding the extended data. Figure 23-10 shows the format of the data written into the two parts of the sample buffer.

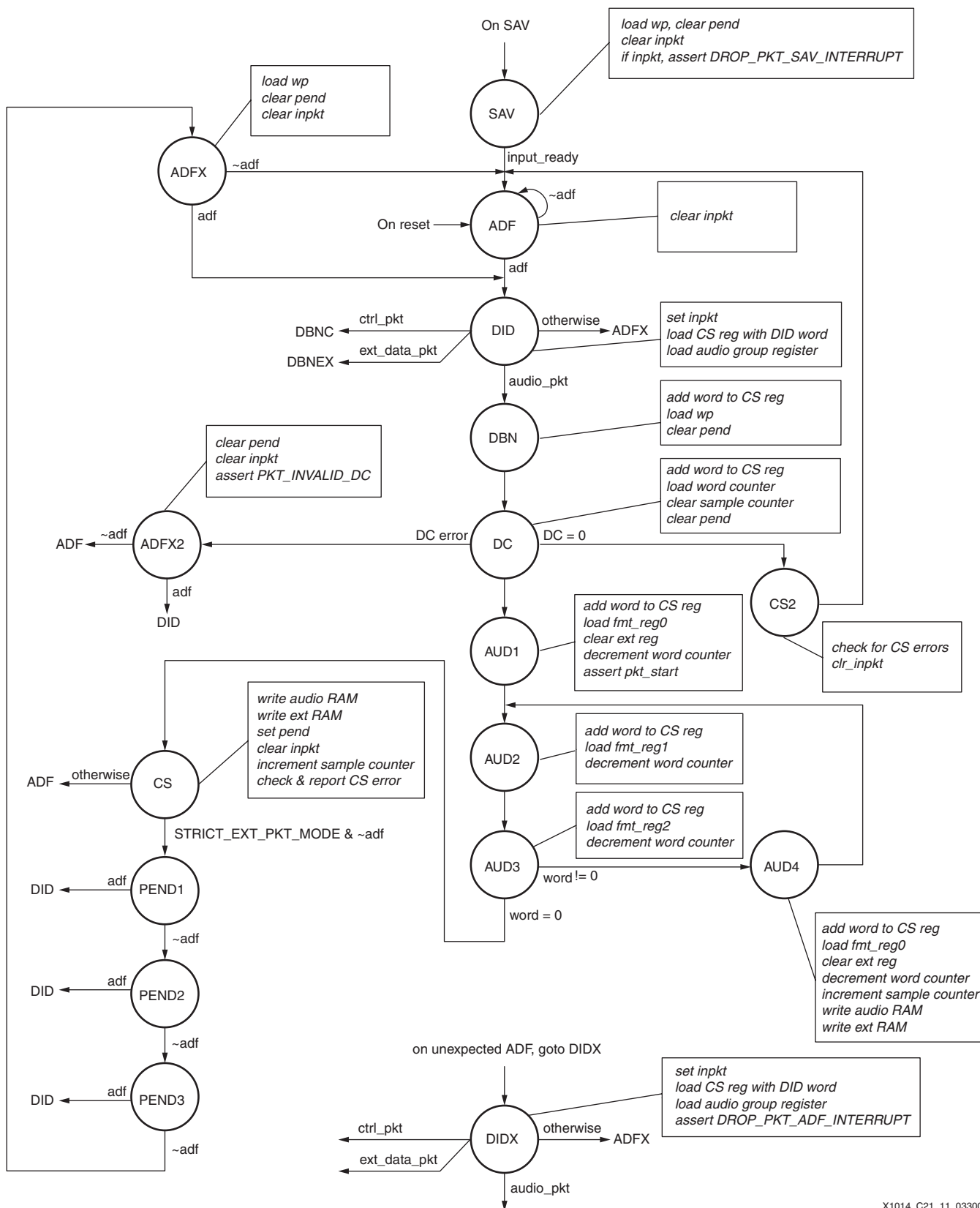


X1014_C21_10_033009

Figure 23-10: Sample Buffer Data Formats

Input State Machine

Figure 23-11, Figure 23-12, and Figure 23-13 show the state diagrams of the input's finite state machine.



X1014_C21_11_033009

Figure 23-11: Input FSM Audio Data Packet Processing

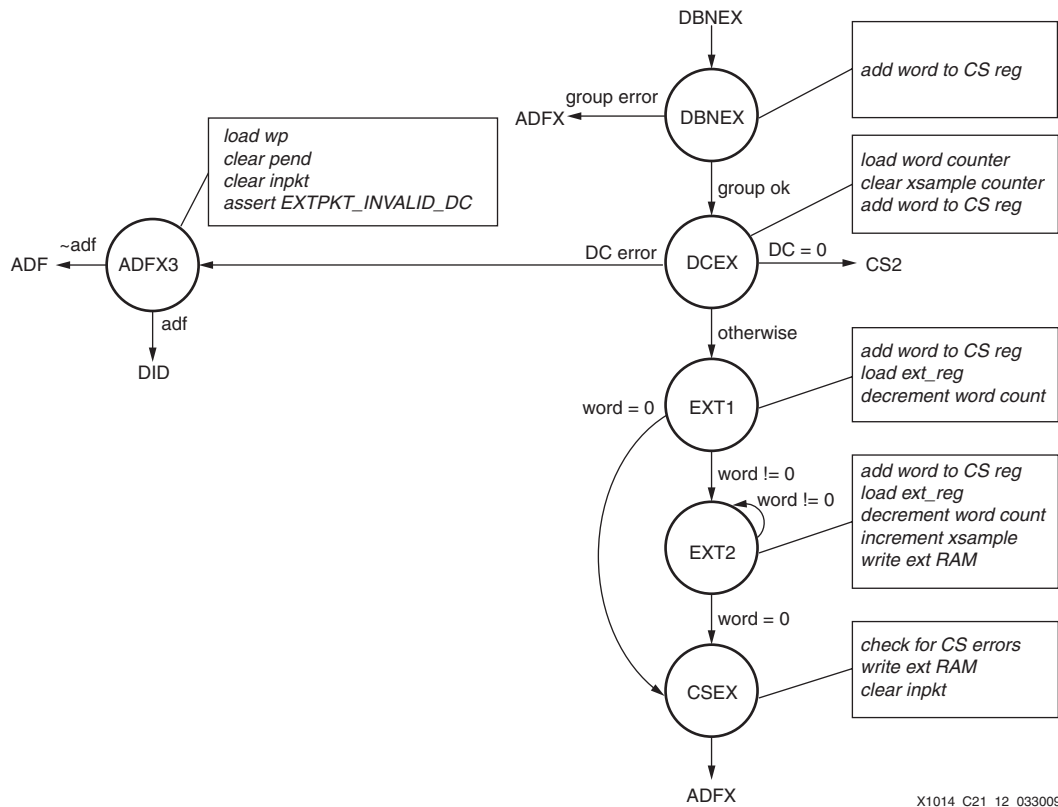


Figure 23-12: Input FSM Extended Data Packet Processing

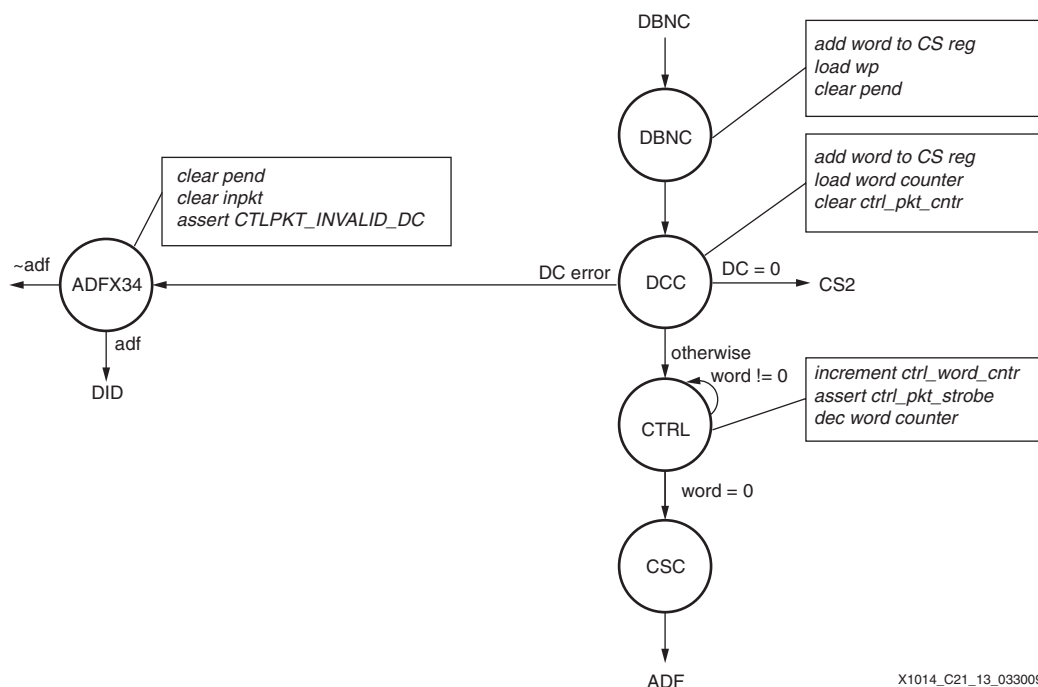


Figure 23-13: Input FSM Audio Control Packet Processing

The input state machine remains idle until the `adf` input is asserted, indicating the start of an ancillary packet. After `adf` is asserted, the input state machine decodes the DID word of the ancillary packet to determine if the packet is an embedded audio packet and, if it is, whether it is an audio data packet, an extended data packet, or an audio control packet that branches to different algorithms for each of these types of packets. If the ancillary packet is not an embedded audio packet, the state machine goes back to waiting for another ancillary packet.

As the input state machine processes an audio sample from an audio data packet, it formats the three data words that contain the audio sample, along with some additional information like the video stream number, into a 36-bit word that is written to the audio data portion of the sample buffer. At the same time, the input state machine also writes to the extended data portion of the sample buffer, clearing the Ext Data Valid bit to 0 to indicate that valid extended data has not yet been written to the sample buffer for that audio sample.

After the input state machine finishes processing an audio data packet, it waits to see if an extended data packet follows. If there is an extended data packet, the input state machine goes back and writes to only the extended data portion of the sample buffer, setting the Ext Data Valid bit to 1 as it writes the extended data word from the packet into the sample buffer. Each word in the extended data packet contains the extended data for both audio samples of a channel pair. The input state machine writes the extended data for both samples of the channel pair only to the sample buffer location of the first sample of the channel pair.

When the extended data packet has been fully processed or when no extended data packets are found after an audio data packet, the input state machine advances the write pointer, causing the output state machine to begin processing the audio samples. This write pointer is not exactly the same as the write address that the input state machine uses to write audio and extended data to the sample buffer. The write address is manipulated by the input state machine to write all the audio samples from an audio packet and all the extended data from an associated extended data packet to the sample buffer without changing the write pointer. Only when the input state machine has finished processing the audio packet and the extended data packet does it advance the write pointer, thereby handing the completed audio samples to the output state machine.

Audio control packets are not written to the audio sample buffer. The data words of the audio control packet are output directly to the `ctrl_pkt_data` port without going through the audio sample buffer.

When the input state machine is processing a packet, it can be interrupted by the assertion of the `adf` or `sav` signals. Assertion of the `sav` signal indicates the end of the horizontal blanking interval. If the state machine is processing a packet and the end of that packet does not occur before the `sav` signal is asserted, the state machine jumps to the SAV state and aborts the packet. Likewise, if the `adf` signal is asserted to indicate the start of a new packet, before the end of the current packet is reached, the state machine aborts the packet and jumps to the DIDX state to begin processing the new packet.

If an audio data packet is aborted by the `sav` or `adf` signals, the audio samples from the aborted packet are lost and are not output by the demultiplexer. If an extended data packet is aborted, the audio samples from the associated audio data packet are output as 20-bit samples. If an audio control packet is aborted, all the data words received prior to the assertion of the `sav` or `adf` signals are output on the `ctrl_pkt_data` port.

The input state machine uses two flags to keep track of the status of packets. The `inpkt` flag indicates that the state machine is currently processing a packet. This flag is checked in the interrupt states (SAV and DIDX) to see if a packet was interrupted so that the state machine

can properly abort the packet by resetting its internal sample buffer write address to the first sample of the aborted packet. The pend flag is set when the input state machine has finished processing an audio data packet. It indicates that audio samples are pending in the sample buffer, waiting for extended data. If the next ancillary packet is not an extended data packet, or if no more ancillary packets are found in the HANC, the input state machine checks the pend flag to see if audio samples are pending in the sample buffer. If the pend flag is set, the input state machine updates the write pointer, sending these samples to the output state machine as 20-bit samples without extended data.

Output State Machine

The state diagram for the output state machine is shown in Figure 23-14.

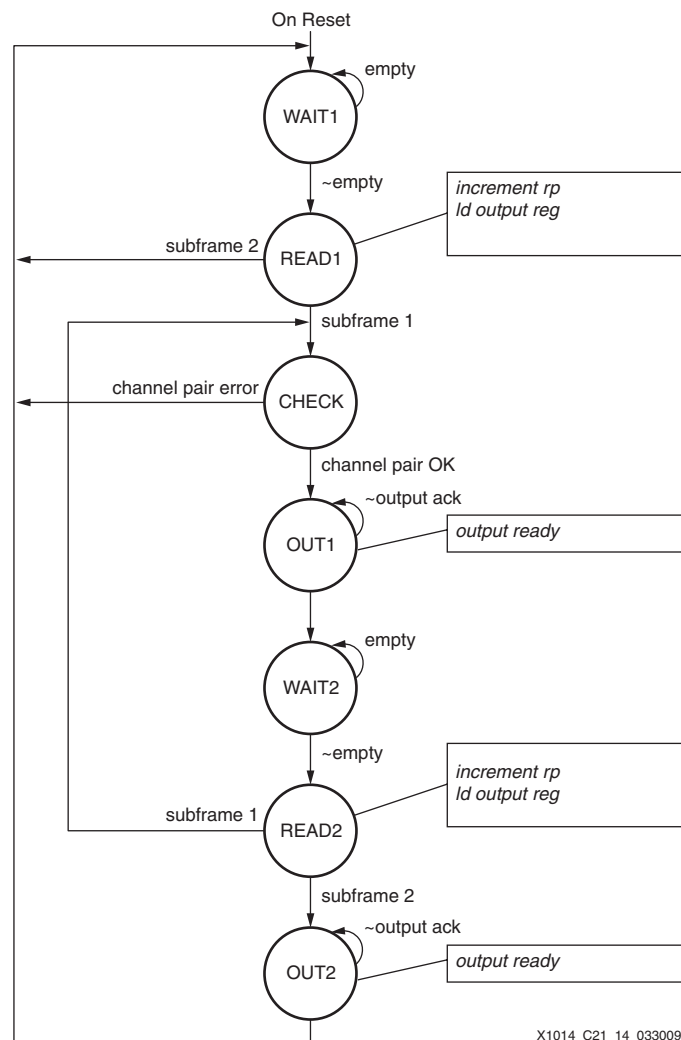


Figure 23-14: Output FSM State Diagram

The output state machine is designed to process audio samples from the sample buffer in pairs, with both samples belonging to the same channel pair. The output state machine waits in the WAIT1 state for the availability of the audio samples in the sample buffer. When the empty signal is deasserted, the state machine moves to the READ1 state and reads the first sample of the channel pair from the sample buffer. The state machine checks

to make sure that this is the first sample of the sample pair by examining the `sf` bit. If the `sf` bit indicates that this is not the first sample of the sample pair, the output state machine considers itself to be out of sync with the data and returns to the `WAIT1` state, discarding the audio sample. If the sample is the first sample of the channel pair, the state machine advances to the `CHECK` state.

In the `CHECK` state, the state machine determines whether the sample is from a channel pair that is enabled for output. If the channel pair is not enabled, the state machine returns to the `WAIT1` state, discarding the sample. If the channel pair is enabled, the sample is output in the `OUT1` state. This process is repeated for the second sample of the pair in the `WAIT2`, `READ2`, and `OUT2` states. The state machine does not check to see if the channel pair is enabled for the second sample because it has already determined that the channel pair is enabled.

Output handshaking is implemented in the `OUT1` and `OUT2` states. In these states, the `output_ready` signal is asserted and the state machine waits until the `output_ack` signal is asserted before proceeding.

Audio Packet Deletion

The audio packet deletion feature found in the single stream demultiplexer is implemented by the `sd_aes_pkt_del` module. This module is entirely controlled by timing signals from the input state machine. The input state machine tells the module when the current input video word contains an audio packet DID. The module checks to see if the audio group is to be deleted, and if so, replaces the DID with the deleted packet DID value, `0x180`.

The `sd_aes_pkt_del` module must also calculate a new checksum value for the packet that is deleted. The new checksum calculation begins with the new DID value. The module accumulates each video word into the checksum value until the input state machine indicates that the current word is the packet's checksum word. At that point, the module replaces the old checksum value with the newly calculated value.

Input Stream Control Module

The multi-stream wrapper modules include a module called `sd_aes_demux_instream_ctrl` to control the multiple input streams. This module monitors the empty signals from all of the input FIFOs and selects a FIFO as the input stream source by asserting the FIFO's read enable signal. The module informs the demultiplexer module which stream is currently selected by driving the `stream_in` port of the demultiplexer.

The input stream controller normally only switches streams when a word is read from the selected FIFO with the `SAV` flag set, indicating the end of the HANC data. When this occurs, the controller selects another stream whose FIFO is not empty to be the new source stream. If a FIFO becomes empty before the `SAV`, the controller begins a timeout period. If the selected stream does not begin supplying data before the end of the timeout period, the controller switches to another stream. However, the timeout period does not expire if all of the FIFOs are empty. The length of the timeout period is controlled by the parameter/generic `TIMEOUT_CNTR_BITS`. This value specifies the number of bits used in the timeout counter. The timeout period expires when the timeout counter reaches the terminal count. The terminal count occurs when all of the bits of the counter are 1. The timeout counter is not enabled by `in_ce` and thus counts every clock cycle.

The controller generates a clock enable signal to the demultiplexer. This clock enable is asserted when `in_ce` is asserted and a non-empty FIFO has been selected. To ensure that the

demultiplexer sees SAV and timeout events, the clock enable to the demultiplexer is asserted for one clock cycle when the SAV is read from a FIFO (which usually coincides with the FIFO becoming empty) and when a timeout event occurs.

The controller uses a priority encoder as an arbiter to select the next input stream when switching streams. This priority encoder gives priority to lower numbered input streams. However, because the FIFOs for the streams are written only during the HANC interval, they fill up only once per video line. The demultiplexer must be clocked fast enough to process all the HANC data for every input stream on every line. Thus, under these conditions, the priority encoder arbiter allows each and every input stream to be processed every video line. The priority encoder is a separate module and can be replaced with a different arbitration scheme, if desired.

Design Files

The reference design for the AES3 audio demultiplexer is available at <https://secure.xilinx.com/webreg/clickthrough.do?cid=119758>. Open the ZIP archive and extract the file `xapp1014_c23_sd_audio_demux.zip`.

Conclusion

FPGAs are now commonly used to implement SD-SDI receivers. Embedded audio is often included with the video in an SD-SDI stream. The embedded audio must generally be demultiplexed from the video so that it can be processed separately.

The reference design described in this chapter detects and demultiplexes audio from the SD-SDI video stream. The module supports up to 16-input video streams. This allows audio and video to be processed by the FPGA, increasing the level of integration and decreasing the cost of video applications.

Audio Multiplexer for 3G-SDI, Dual-Link HD-SDI, and HD-SDI Video

Introduction

In many applications, audio is embedded in the video stream to facilitate the transport of the combined audio and video stream. For HD-SDI video, the audio information can be multiplexed into the HANC portion of the video. The specification for this is given in SMPTE 299M. Refer to this document for particulars of the data formats for embedded audio. Standards for 3G-SDI and dual link HD-SDI also reference SMPTE 299M as the format for embedded audio data. The most commonly used audio sample rate for video is 48 KHz. A revision of the SMPTE standard, underway at the time of publication but not finalized, specifies a modified data packet format for carrying 96 KHz samples.

This chapter describes how audio embedding is done and presents a hardware-verified reference design that implements an audio multiplexer compatible with HD-SDI, 3G-SDI, and dual link HD-SDI video. The embedded audio standard for SD-SDI is substantially different and is not supported by the reference design.

Features

Some of the features of the audio multiplexer are:

- Single audio group granularity: One instance of the multiplexer embeds the four channels (two channel pairs) of a single audio group. The audio group number is an input to the design. Audio from multiple audio groups can be embedded by using multiple instances of the basic design in a daisy-chain fashion.
- Support for 3G-SDI level A and level B (SMPTE 425M) and dual link HD-SDI (SMPTE 372M).
- Multiple HD standards: An input field designates what line standard is being used.
- Audio control packets: Control packets are generated and inserted on the appropriate lines. There are input ports for control packet information. Control packet insertion can be disabled.
- 24-bit audio at multiple sample rates: Audio is received as 24 bits plus Z, V, U, and C flags with an audio valid strobe to indicate timing.
- Multiple sample rates: 32 KHz, 44.1 KHz, 48 KHz, and 96 KHz are supported. The base design supports 32 KHz, 44.1 KHz, and 48 KHz. The reference design includes an optional formatter module that adds support for 96 KHz.
- Synchronous and asynchronous audio.
- Automatic clock phase and ECC generation.

- Overwrites of incoming embedded audio: Existing embedded audio packets (data and control) in the designated group are overwritten with new packets to maximize utilization of ancillary space. Other ancillary data is passed. All embedding can be disabled, in which case no packets are added or deleted, and the video stream is passed unaltered.
- Support of many Xilinx® device families: The reference design has been hardware verified on the Virtex®-5 FPGA but uses features common to all Xilinx FPGAs.

Embedded Audio

Audio information is embedded in SDI data streams in the form of packets that are placed in the horizontal blanking period of the C_{B/C_R} data stream. The general specification for HD-SDI video formats and timing is given in SMPTE 292M. 3G-SDI is specified by SMPTE 425M and dual link HD-SDI by SMPTE 372M. Among other things, these standards specify how timing references and line numbers are transmitted. The embedded audio format for all of these video standards is given in SMPTE 299M. Thus, the formats for HD-SDI also apply to 3G-SDI level A and level B and dual link HD-SDI.

Embedded audio packets are just a few of many different types of ancillary packets that might be inserted in the video stream during blanking periods. SMPTE 291M specifies the general format of ancillary packets (see Figure 24-1). Common elements include a three-word ancillary data flag (ADF), a data identity (DID) to specify the packet type, a data block number (DBN), and data count (DC) fields. A checksum (CS) is the final field of every ancillary packet.

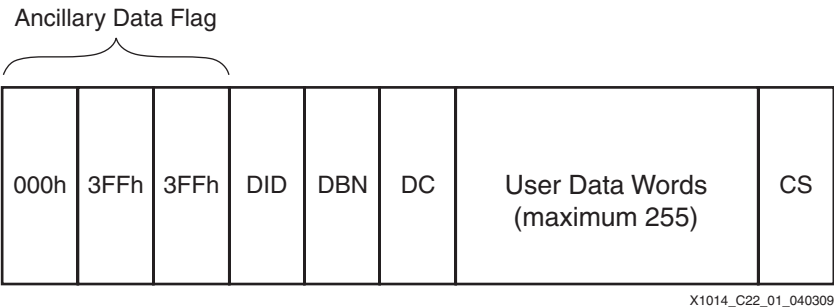


Figure 24-1: Ancillary Data Packet

There are two types of embedded audio packets: audio data packets and audio control packets. Audio data packets contain audio samples for the various channels. Audio control packets contain various metadata fields describing such things as sample rates and audio processing delay. The DID values for embedded audio in HD-SDI are shown in Table 24-1. Each audio data packet contains four samples, one from each channel in a group. Likewise, each audio control packet contains metadata that applies to one audio group.

Table 24-1: DID Values for 3G-SDI/HD-SDI Embedded Audio Packets

Group	Audio Data Packets	Audio Control Packets
1	0x2E7	0x1E3
2	0x1E6	0x2E2
3	0x1E5	0x2E1
4	0x2E4	0x1E0

While ancillary data packets are required to be contiguous and begin immediately after the EAV, the exact location of packets of a given type is not specified. Therefore, to multiplex audio into the video stream, all ancillary data types must be examined to identify obsolete packets that must be deleted or overwritten. The ancillary data types must also be examined to determine where to insert new audio packets. This means that the DID field must be checked, and the DC must be read to determine the length of the packet and thus, where to look for the next packet.

Audio Data Packets

There are two data packet formats, one for the 32 KHz to 48 KHz sampling rates, and one for the 96 KHz sampling rate. The main difference between the two is that the 96 KHz format has two sequential samples of two channels in a packet, whereas the 32 KHz to 48 KHz format has one sample of four channels in each packet.

32 KHz to 48 KHz Audio Data Packets

The format of audio data packets is shown in Figure 24-2. These have the standard ancillary packet format. The value of DID depends on the audio group, as specified in Table 24-1. DC is always 0x218. The DBN increments for each packet of the same DID. In addition, the audio data packet contains 24 UDWs, a two-word clock phase field (CLK), six ECC words, and four audio words (two stereo pairs). The fields encompassed by the UDWs are described in this section.

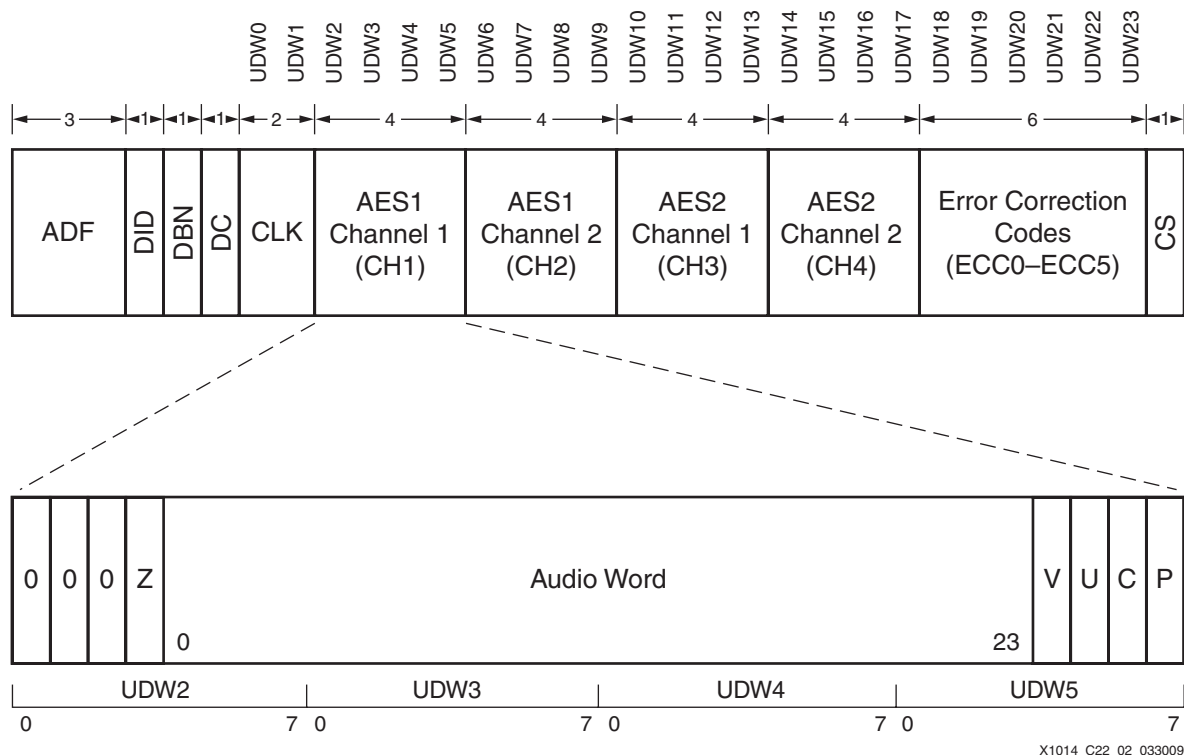


Figure 24-2: Audio Data Packet (32 KHz to 48 KHz)

Audio data packets can be multiplexed onto any video line, with the exception of the lines following the synchronous switching points. The samples must be more or less evenly

distributed. The number of audio data packets on any line containing packets can vary by at most one.

Audio Sample Words

The audio sample words are in AES format. Each AES word (AES_n) includes Z, V, U, C, and P bits. The Z bit is present for CH1 and CH3 only. The audio word itself spans the four UDWs, as shown in [Figure 24-2](#). All of the words in the audio data packet are 10 bits wide. The eight LSBs contain the actual data. Bit 8 contains even parity for the eight LSBs. Bit 9 contains the inverse of bit 8.

Audio Clock Phase Data

The CLK field contains information about the timing of the audio clock with respect to the video lines. Specifically, the audio clock phase is the number of video clock times (rounded to the nearest integer) between the samples contained in the packet and the EAV immediately preceding the arrival of the audio clock for those samples. Due to the prohibition of packets in the line following the synchronous switching point, the audio sample packet sometimes occurs more than a line time after the EAV to which it is referenced. For these cases, a multiplex position flag (mpf) is asserted. The mpf flag indicates when the given clock phase is from the next-to-last EAV. This implies that there must be at least two video lines of latency in the demultiplexing process.

Error Correction Codes

UDWs 18 to 23 are ECCs. The reference design of this chapter produces ECCs in the ECC encoder section.

96 KHz Audio Data Packets

The data packets for 96 KHz differ from those for 32 KHz to 48 KHz in that they contain two sequential samples from two channels. The format of the 96 KHz data packet is shown in [Figure 24-3](#).

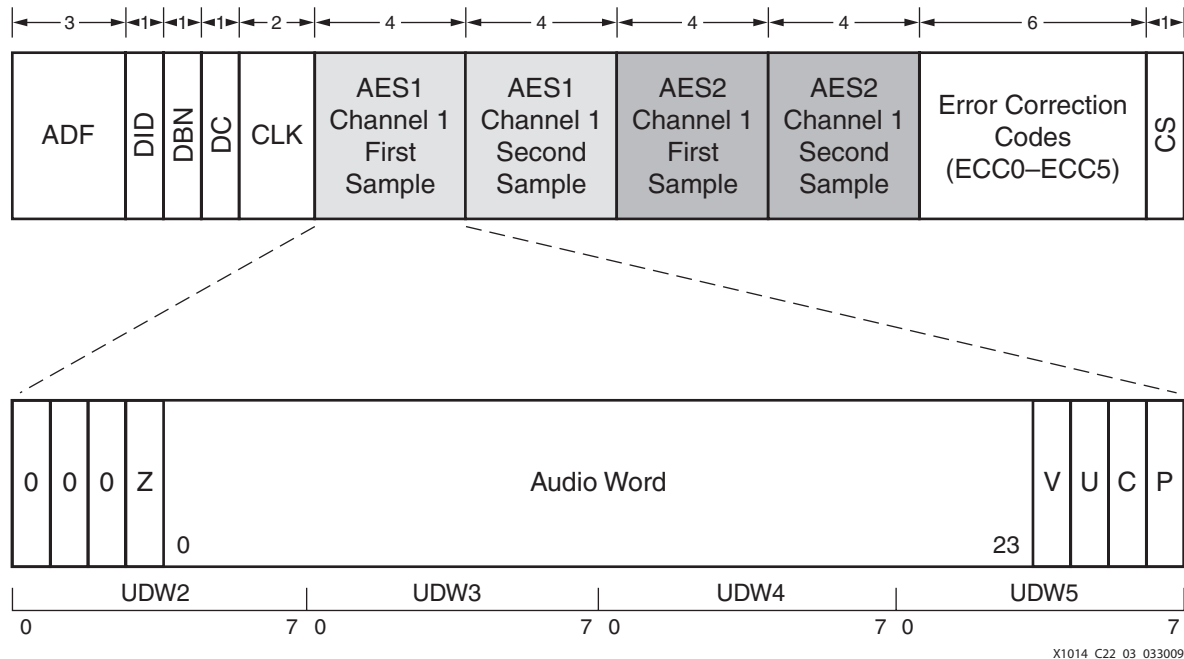


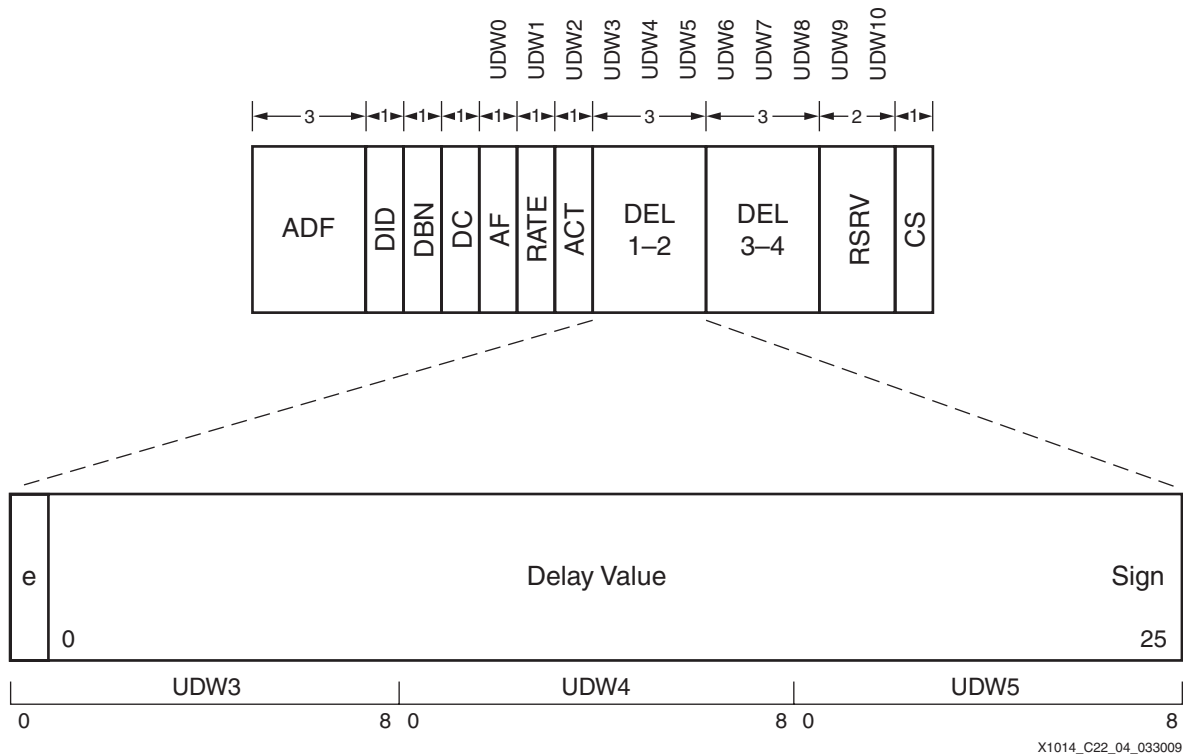
Figure 24-3: Audio Data Packet (96 KHz)

The 96 KHz data packet format has these implications:

- Because there are only two unique audio channels per packet, there are only two channels per group. Therefore, the four defined audio groups can contain only eight audio channels, compared to 16 channels for the 32 KHz to 48 KHz format. For HD-SDI and 3G-SDI level A, this means that only eight channels of 96 MHz audio are uniquely defined. For dual link HD-SDI, each link can carry eight unique audio channels. Therefore, 16 unique audio channels can be identified. This also applies to 3G-SDI level B.
- Because there is a single clock phase field, it can apply to only one of the two sequential samples. The clock phase field is defined to apply to the second sequential sample. No explicit clock phase data is transmitted for the first sample.
- On average, the same number of packets per group is transmitted on each video line as for 48 MHz sampling. In other words, the number of packets in the 96 KHz format is the same as for the 48 KHz format. However, the number of channels carried in those packets is halved while the number of samples per channel is doubled.

Audio Control Packets

Audio control packets are transmitted once per field in an interlaced system or once per frame in a progressive system. They are placed in the blanking period of the second line after the synchronous switching point. The packet contains information about the channels in one group. Thus, if multiple audio groups are used, multiple audio control packets should be sent. Audio control packets have the standard ancillary format with 11 UDWs (see Figure 24-4). The words are 10 bits wide, with most UDWs having 9 bits of data. ACT is the only UDW with parity (on bit 8). In all UDWs, the MSB (bit 9) is the complement of bit 8.



X1014_C22_04_033009

Figure 24-4: Audio Control Packet

The DID value depends on the audio group, as shown in Table 24-1. DC is always 0x10B. The AF number, if used, indicates the position of the field with respect to the audio frame sequence. If not used, the AF number is set to all zeros. The sampling rate (RATE) indicates the sample rate of the embedded audio. The LSB is the asynchronous flag, asx. When asserted, this flag indicates that the audio is asynchronous with the video. The rate codes are shown in Table 24-2. A synchronous sample rate of 48 KHz is almost exclusively used in the industry.

Table 24-2: Sample Rate Codes

RATE[3:1]	Sample Rate (KHz)
000	48.0
001	44.1
010	32.0
100	96.0
111	Free running
101, 110	Reserved

The ACT word indicates which of the four channels in the group is active. Bit 0 corresponds to CH1, bit 1 to CH2, and so forth. A logic 1 means that the channel is active. The delay (DEL) fields indicate the amount of accumulated audio delay relative to video in the indicated channel pair. The delay is measured in audio sample intervals as a 26-bit two's-complement number. Positive values indicate that the video leads the audio. The

LSB of the entire field is an enable bit. When this bit is set High, the delay value is valid. UDWs 9 and 10 are reserved and are set to 0.

Reference Design

The reference design for the audio multiplexer is for one audio group. Multiple instances can be daisy-chained to multiplex audio for multiple audio groups. Ancillary data other than for the targeted audio group is passed unaltered. The block diagram of the audio multiplexer is shown in [Figure 24-5](#).



Figure 24-5: 3G-SDI/HD-SDI Audio Multiplexer Block Diagram

All video data passes through the audio multiplexer, but only ancillary data is modified. The C_{B/C_R} video stream passes through the `hd_audio_mux_data` module for embedding of audio data packets. The Y video stream passes through the `hd_audio_mux_contr` module for embedding of audio control packets. In each case, the stream is monitored by the Ancillary Data Parsing module. This module looks for the presence of ancillary data packets and keeps track of line numbers. Two types of modification can be done to ancillary packets: existing audio packets can be marked for deletion, and audio packets can be inserted. The actual modifications are done in the Audio Packet Multiplexer module. This module is controlled by the Ancillary Data Parsing module and receives packet data from either the Audio Data Packet Construct module for data packets or the Audio Control Packet Construct module for control packets.

Audio data packets are assembled by the Audio Data Construct module. This module receives timing and control information from the Ancillary Data Parsing module and data from the `aud_fifos` module. At the appropriate time, the Audio Data Construct module feeds packet data to the Audio Packet Multiplexer module for inclusion in the video stream. As the data is fed to the Audio Packet Multiplexer module, ECCs are calculated and also sent to be included in the packet. The Audio Control Packet Construct module performs a similar function, providing control packet words at the appropriate time. This module receives control fields directly from outputs to the module.

The Clock Phase Determination module receives line numbers and inferred line timing from the Ancillary Data Parsing module. It also receives the audio strobe, indicating the timing of the audio samples. From this, the Clock Phase Determination module creates clock phase data that is included in the audio data packets. This clock phase information is stored, along with the samples themselves, in the `aud_fifos` module. The `aud_fifos` module stores samples in sets of four, one for each channel, along with the clock phase information.

Inputs and Outputs

There are two modules at the top level, `hd_audio_mux_top` and `format96_mux`. [Table 24-3](#) lists the inputs and outputs of `hd_audio_mux_top`.

Table 24-3: Inputs and Outputs of `hd_audio_mux_top` Module

Signal	Direction	Description
<code>vid_clk</code>	In	This is the video clock.
<code>aud_clk</code>	In	This is the input audio clock for audio data FIFOs.
<code>rst</code>	In	This is an asynchronous reset.
<code>vid_ce</code>	In	This is the video clock enable for audio data FIFOs.
<code>vid_samp_ce</code>	In	This is the video sample-rate clock enable. This is the same as <code>vid_ce</code> except in the case of 12-bit video samples.
<code>aud_ce</code>	In	This is the audio clock enable.
<code>aud_mux_en</code>	In	This input enables insertion of audio data packets: 0 = Disable 1 = Enable

Table 24-3: Inputs and Outputs of `hd_audio_mux_top` Module (Cont'd)

Signal	Direction	Description
<code>contr_pkt_en</code>	In	This signal enables insertion of audio control packets: 0 = Disable 1 = Enable
<code>aud_group[1:0]</code>	In	This is the audio group number: 0 = Group 1 1 = Group 2, etc.
<code>chan_valid[3:0]</code>	In	These are the channel valid bits in unary format. These bits denote the valid audio channels of the group.
<code>sample_rate[2:0]</code>	In	This is the audio sample rate: 0 = 48 KHz 1 = 44.1 KHz 2 = 32 KHz 4 = 96 KHz
<code>asx</code>	In	This control packet value is a synchronous data flag: 0 = Synchronous 1 = Asynchronous
<code>delay_1_2[25:0]</code>	In	This control packet value is the audio processing delay for channels 1 and 2.
<code>delay_1_2_val</code>	In	This control packet value is the valid flag for <code>delay_1_2</code> audio processing delay.
<code>delay_3_4[25:0]</code>	In	This control packet value is the audio processing delay for channels 3 and 4.
<code>delay_3_4_val</code>	In	This control packet value is the valid flag for <code>delay_3_4</code> audio processing delay.
<code>line_std[3:0]</code>	In	This input port indicates the video format. These bits are encoded as: 0000 = SMPTE 260M 1035i 30 Hz 0001 = SMPTE 295M 1080i 25 Hz 0010 = SMPTE 274M 1080i or 1080sF 30 Hz 0011 = SMPTE 274M 1080i or 1080sF 25 Hz 0000 = SMPTE 274M 1080p 30 Hz or 3G 1080p 60 Hz if <code>threeG</code> is asserted 0101 = SMPTE 274M 1080p 25 Hz or 3G 1080p 50 Hz if <code>threeG</code> is asserted 0110 = SMPTE 274M 1080p 24 Hz 0111 = SMPTE 296M 720p 60 Hz 1000 = SMPTE 274M 1080sF 24 Hz 1001 = SMPTE 296M 720p 50 Hz 1010 = SMPTE 296M 720p 30 Hz 1011 = SMPTE 296M 720p 25 Hz 1100 = SMPTE 296M 720p 24 Hz
<code>vid_y_in[9:0]</code>	In	This is the Y input video stream.
<code>vid_c_in[9:0]</code>	In	This is the $C_B C_R$ input video stream.

Table 24-3: Inputs and Outputs of `hd_audio_mux_top` Module (Cont'd)

Signal	Direction	Description
ovw_en	In	This is the overwrite enable: 0 = Marks incoming packets in the selected group for deletion 1 = Overwrites incoming data and control packets in the selected group
ext_ln_en	In	This is the enable for using external line numbers: 0 = Uses line numbers decoded from the video stream 1 = Uses line numbers from ext_ln input
ext_ln[10:0]	In	This is an externally provided line number. In general, externally provided line numbers must be used in cases where valid line numbers have not yet been embedded in the input video stream.
threeG	In	This input indicates that the line standard designated by line_std is a 3G-SDI line standard.
vid_y_out[9:0]	Out	This is the Y output video stream.
vid_c_out[9:0]	Out	This is the C _B C _R output video stream.
vid_c_out_dup[9:0]	Out	This is an identical copy of vid_c_out.
aud_valid	In	This is the audio valid strobe. Logic High indicates that valid audio data is present on the inputs. The aud_valid signal applies to all four channels.
ch1_audio[23:0]	In	Channel 1 audio sample data.
ch1_z	In	Channel 1 Z flag.
ch1_c	In	Channel 1 channel status data.
ch1_u	In	Channel 1 user data bit.
ch1_v	In	Channel 1 valid bit.
ch2_audio[23:0]	In	Channel 2 audio sample data.
ch2_c	In	Channel 2 channel status data.
ch2_u	In	Channel 2 user data bit.
ch2_v	In	Channel 2 valid bit.
ch3_audio[23:0]	In	Channel 3 audio sample data.
ch3_z	In	Channel 3 Z flag.
ch3_c	In	Channel 3 channel status data.
ch3_u	In	Channel 3 user data bit.
ch3_v	In	Channel 3 valid bit.
ch4_audio[23:0]	In	Channel 4 audio sample data.
ch4_c	In	Channel 4 channel status data.

Table 24-3: Inputs and Outputs of `hd_audio_mux_top` Module (Cont'd)

Signal	Direction	Description
ch4_u	In	Channel 4 user data bit.
ch4_v	In	Channel 4 valid bit.

Table 24-4 lists the inputs and outputs of the `format96_mux` module.

Table 24-4: Inputs and Outputs of `format96_mux` Module

Signal	Direction	Connection to <code>hd_audio_mux_top</code>	Description
clk	In	Same as vid_clk	This is a clock that is synchronous with in_samp_valid.
ce	In	Same as vid_ce	This is the clock enable.
format96_en	In	N/A	This is the 96 KHz format enable: 1 = Enables 96 MHz formatting 0 = Passes inputs straight through to outputs
in_samp_valid	In	N/A	This is the audio sample valid input. It is active for one ce period for each sample.
in_samp_1[23:0]	In	N/A	Audio sample stream input 1.
in_samp_2[23:0]	In	N/A	Audio sample stream input 2.
in_samp_3[23:0]	In	N/A	Audio sample stream input 3.
in_samp_4[23:0]	In	N/A	Audio sample stream input 4.
in_flags_1[3:0]	In	N/A	These are the input flags for stream 1: bit[3] = Z bit[2] = C bit[1] = U bit[0] = V
in_flags_2[3:0]	In	N/A	These are the input flags for stream 2: bit[2] = C bit[1] = U bit[0] = V
in_flags_3[3:0]	In	N/A	These are the input flags for stream 3: bit[3] = Z bit[2] = C bit[1] = U bit[0] = V

Table 24-4: Inputs and Outputs of `format96_mux` Module (Cont'd)

Signal	Direction	Connection to <code>hd_audio_mux_top</code>	Description
<code>in_flags_4[3:0]</code>	In	N/A	These are the input flags for stream 4: bit[2] = C bit[1] = U bit[0] = V
<code>out_samp_1[23:0]</code>	Out	<code>ch1_audio</code>	Audio sample stream output 1.
<code>out_samp_2[23:0]</code>	Out	<code>ch2_audio</code>	Audio sample stream output 2.
<code>out_samp_3[23:0]</code>	Out	<code>ch3_audio</code>	Audio sample stream output 3.
<code>out_samp_4[23:0]</code>	Out	<code>ch4_audio</code>	Audio sample stream output 4.
<code>out_flags_1[3:0]</code>	Out	<code>ch1_z</code> , <code>ch1_c</code> , <code>ch1_u</code> , <code>ch1_v</code>	These are the output flags for stream 1: bit[3] = Z bit[2] = C bit[1] = U bit[0] = V
<code>out_flags_2[3:0]</code>	Out	<code>ch2_c</code> , <code>ch2_u</code> , <code>ch2_v</code>	These are the output flags for stream 2: bit[2] = C bit[1] = U bit[0] = V
<code>out_flags_3[3:0]</code>	Out	<code>ch3_z</code> , <code>ch3_c</code> , <code>ch3_u</code> , <code>ch3_v</code>	These are the output flags for stream 3: bit[3] = Z bit[2] = C bit[1] = U bit[0] = V
<code>out_flags_4[3:0]</code>	Out	<code>ch4_c</code> , <code>ch4_u</code> , <code>ch4_v</code>	These are the output flags for stream 4: bit[2] = C bit[1] = U bit[0] = V
<code>out_samp_valid</code>	Out	N/A	This is the output sample valid. It is active for one ce period per sample.

Modules

The reference design contains 14 modules. These are listed in [Table 24-5](#) and are described in more detail in the remainder of this section.

Table 24-5: List of Modules

Module	Description
hd_audio_mux_top	This is a top-level wrapper for the audio multiplexer.
hd_audio_mux_data	This is a multiplexer for audio data packets on the C video stream.
jd_audio_mux_contr	This is a multiplexer for audio control packets on the Y video stream.
anc_det	This module detects the presence, type, and location of ancillary data packets. Controls deleting of packets.
packet_mux	This module modifies the $C_B C_R$ video stream to mark packets for deletion and audio data and control packets.
clock_phase	This module determines clock phase data based on the timing of the audio valid strobe to the line and sample position.
sync_one_shot	This module detects the rising edge of strobes and outputs a one-clock-wide pulse for each rising edge.
aud_fifos	This is a wrapper for instances of <code>fifo_28x16</code> that stores incoming audio samples prior to embedding.
fifo_28x16	This is a FIFO element, 28 bits wide by 16 locations deep, built from LUTs.
aud_data_constr	This module constructs the audio data packets for insertion into the video stream.
ecc_encode	This module calculates the ECCs for the audio data packets.
aud_contr_constr	This module constructs the audio control packets for insertion into the video stream.
format96_mux	This module converts a sequence of two samples into a pair of parallel audio samples in the 96 KHz format.
split96_deser	This module deserializes two sequential samples of 96 KHz audio samples into one pair of samples for the audio packet.

Top-Level Module (hd_audio_mux_top)

This module instantiates and connects the two packet multiplexer modules for the $C_B C_R$ and Y video streams, `hd_audio_mux_data` and `hd_audio_mux_control`. The `hd_audio_mux_top` module is shown in [Figure 24-5, page 554](#).

Data Packet Multiplexer (hd_audio_mux_data)

This module is the top level for multiplexing audio data packets onto the $C_B C_R$ video stream. A block diagram of the Data Packet Multiplexer module is shown in the top section of [Figure 24-5, page 554](#). This module instantiates and connects the lower-level blocks shown.

Control Packet Multiplexer (hd_audio_mux_contr)

This module is the top level for multiplexing audio control packets onto the Y video stream. A block diagram of the Control Packet Multiplexer module is shown in the bottom

section of [Figure 24-5, page 554](#). This module instantiates and connects the lower-level blocks shown.

Ancillary Data Parsing (anc_det)

This module detects ancillary data, checks for audio packets, and determines where the audio packets can be inserted. If audio packets for the designated group are already present in the video stream, they are overwritten with new packets if overwrite is enabled (`ovw_en = 1`). If overwrite is disabled, the existing packets are marked for deletion. If there are more existing packets than new packets to overwrite them, the extra packets are also marked for deletion.

Ancillary data packets must be contiguous. If there are more ancillary data packets than the ones to be overwritten, or there are more packets to be written than the ones to be overwritten in the incoming video, new audio packets are appended immediately after the end of all existing packets. If no ancillary packets are present in the incoming stream, new packets are appended immediately after the EAV. In addition, it is possible that an end-marker packet is present at the end of all existing ancillary packets (per SMPTE 291M). If additional packets are to be inserted, the end-marker packet is overwritten with new packets or marked for deletion before new packets are appended.

Because all channels in a given group must be synchronous, there is no practical way to add or replace audio samples in an existing packet. To add a new channel into an existing group, the audio from that group must be demultiplexed, synchronized, and input to this module with the other channels in the group. The original audio packets are then replaced with the new packets created for that group.

[Figure 24-6](#) shows the main elements of ancillary data parsing. A state machine detects the EAV, SAV, and ancillary packets. It looks at the DID to determine which, if any, packets must be overwritten or marked for deletion. The word counter counts words for multi-word fields. The HANC counter is used to determine the size of the horizontal ancillary space, the current position within the HANC space, and whether or not there is enough room to insert another packet. The number of data packets inserted on a line is also limited by the maximum number of packets per line. This is a function of the line standard and sample rate and enforced by the `aud_data_constr` module. This module deasserts the Data Packet Pending signal when the maximum number of samples per line has been reached.

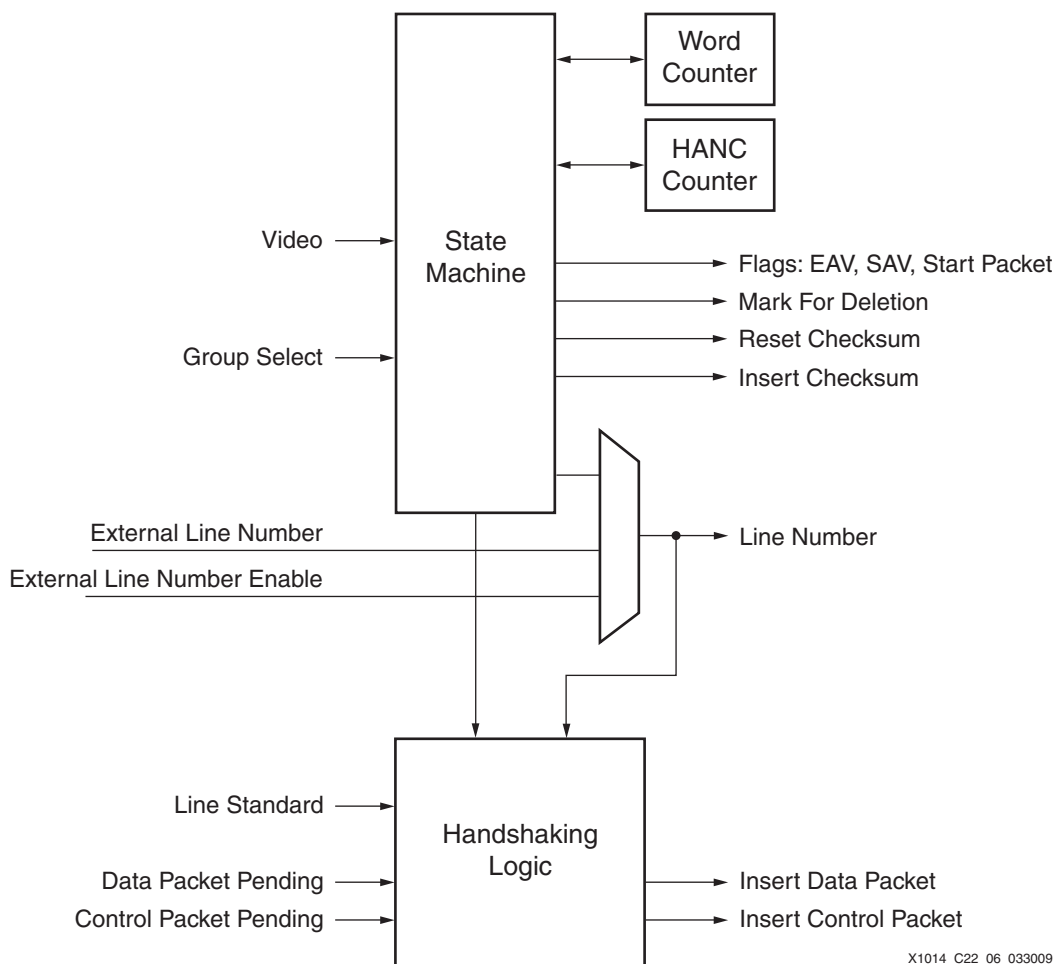


Figure 24-6: Block Diagram of `anc_det` Module

The `anc_det` module controls when the `packet_mux` module inserts packets, marks packets for deletion, or both. The state machine also indicates when to start the checksum operation and where to insert the checksum. It extracts the line number that follows each EAV. In some usage models, valid line numbers might not be available in the video stream. For example, audio embedding might take place prior to embedding of the line numbers. For such cases, an external line number input, `ext_ln`, is provided to supply line numbers directly. The external line number enable, `ext_ln_en`, controls whether the output line number is the external line number or the one extracted from the video stream.

The state diagram for this state machine is shown in Figure 24-7. The states correspond to fields in the timing reference sequences and ancillary data packets. Unless otherwise noted, the values denoted by equal signs are the values of the video data required for the transition to the associated state. The `tcnt` signal referenced is the terminal count signal from the word counter. The states roughly correspond to timing references of SMPTE 292 (EAV and SAV) and to the fields of ancillary data defined by SMPTE 291. In the Options state, the decision is made to insert, overwrite, delete, or skip a packet.

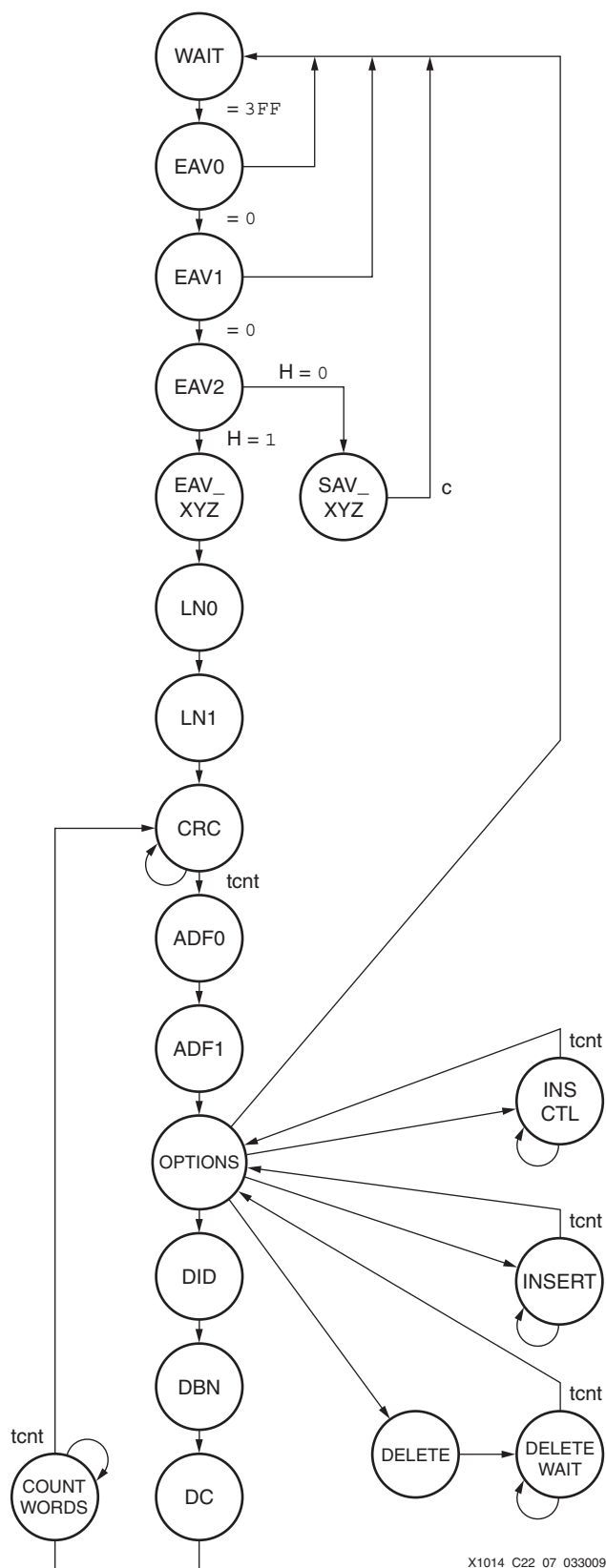


Figure 24-7: Ancillary Data Parser State Diagram

The handshaking logic receives timing and line number information from the state machine. It also uses the video line standard to determine which lines are valid for inserting data and control packets. The packet producing modules for data and control packets (`aud_data_constr` and `aud_contr_constr`) send requests in the form of Data Packet Pending and Control Packet Pending signals. At the appropriate point in the appropriate line, the handshaking logic asserts the Insert Data Packet signal or the Insert Control Packet signal to tell the packet producing modules and the data multiplexing module to insert the packets into the video stream. This decision is based on a number of factors, such as whether an ancillary packet is already embedded in the video stream, what the packet is, whether new packets are available, what the current video line is, and how many packets have already been written on the current line.

Audio Packet Multiplexer (`packet_mux`)

The `packet_mux` module has these functions:

- Controls embedding of audio data and control packets into the video data stream
- Receives control information on when to append packets from the Ancillary Data Parsing module
- Receives requests from and issues acknowledges to the modules that construct data and control packets
- Monitors the line numbers and determines the appropriate times to embed the various packets

The `packet_mux` module also has a word counter and a checksum adder. Thus, the checksum of each packet is inserted as the last word for data and control packets and packets marked for deletion.

Inputs are delayed by various amounts to match delays of packet information coming from various sources. Fundamentally, the proper place for packet insertion/deletion can only be determined several words into the packet. The data stream must be delayed until this determination can be made, and this information is received by the packet producing modules.

For packets marked for deletion, when overwrite is disabled, the `packet_mux` module replaces the DID of the packet with a value of `0x80` (meaning it is marked for deletion) and calculates a new checksum for the packet. [Figure 24-8](#) shows a block diagram of the `packet_mux` module.

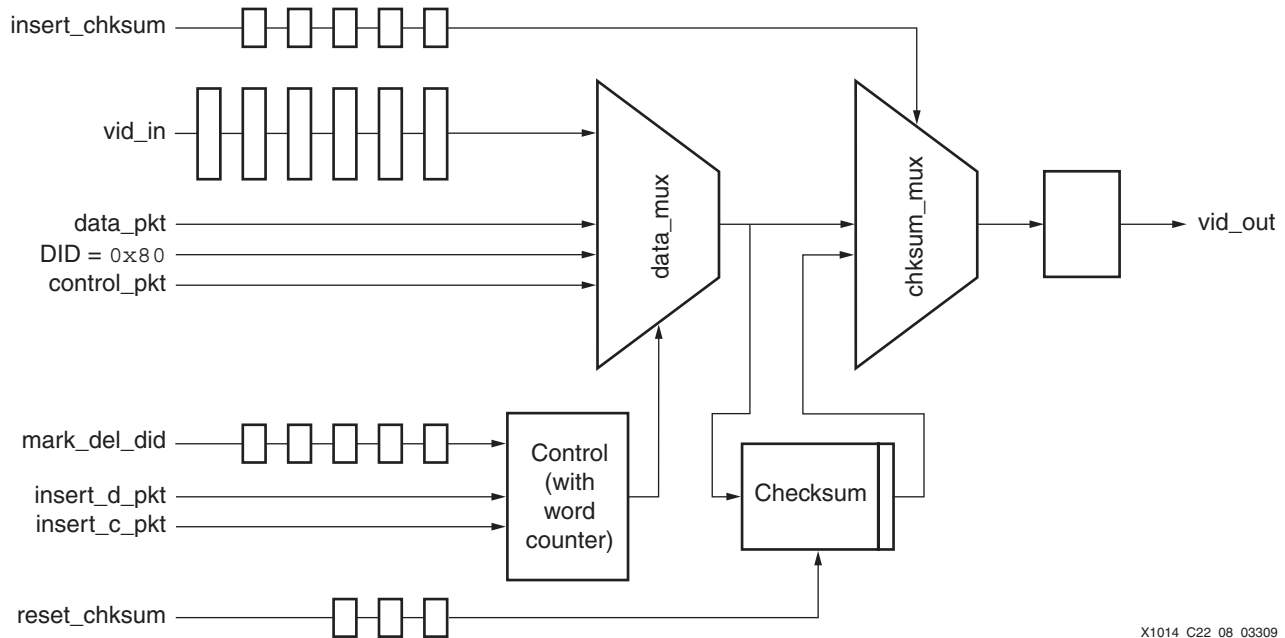


Figure 24-8: Block Diagram of `packet_mux` Module

Clock Phase Determination (`clock_phase`)

The `clock_phase` module generates clock phase information for each audio sample. Figure 24-9 shows a block diagram of this section. The `clock_phase` module uses received EAV and video clock information to determine when the audio samples arrive relative to the video lines. This information is sent to the `aud_fifos` module with each audio sample for inclusion in the audio data packets. The line number is received from the `anc_det` module. The sample number is determined by a video sample counter that is reset each time the line changes, as shown in Figure 24-10. The clock counter increments for each video sample. The audio strobe might be completely asynchronous with the video clock. Therefore, the strobe is detected and synchronized with an instance of the synchronizing one-shot module/entity. The synchronized audio strobe clocks in the current video sample count. This count, along with the line number from the clock phase information, is stored in the clock phase FIFO.

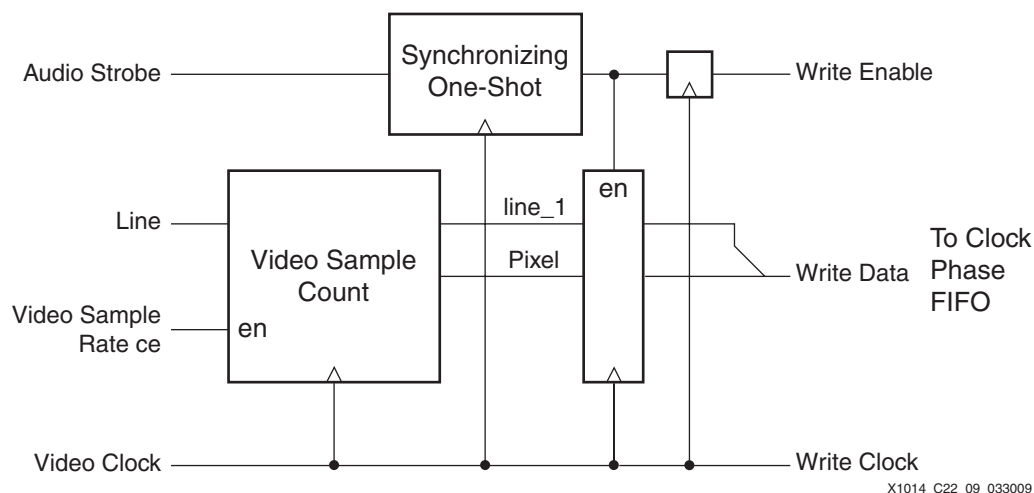
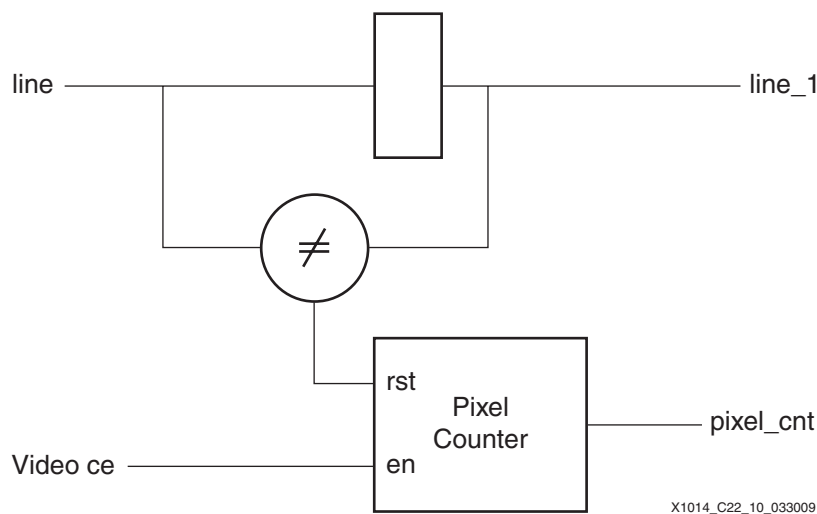
Figure 24-9: Block Diagram of `c1ock_phase` Module

Figure 24-10: Clock Counter

Synchronizing One-Shot (`sync_one_shot`)

This small circuit detects a pulse, synchronizes it to another clock, identifies the rising edge, and outputs a one-cycle-wide pulse in the new clock domain. Figure 24-11 shows the details of the synchronizing one-shot circuit. It can accommodate an extremely wide range of pulse widths on the input. The minimum width of the pulse is just long enough to be recognized as a High level by the SR latch. There is no maximum width. However, the input must remain Low for at least four clock cycles for the next rising edge to be recognized as a distinct pulse.

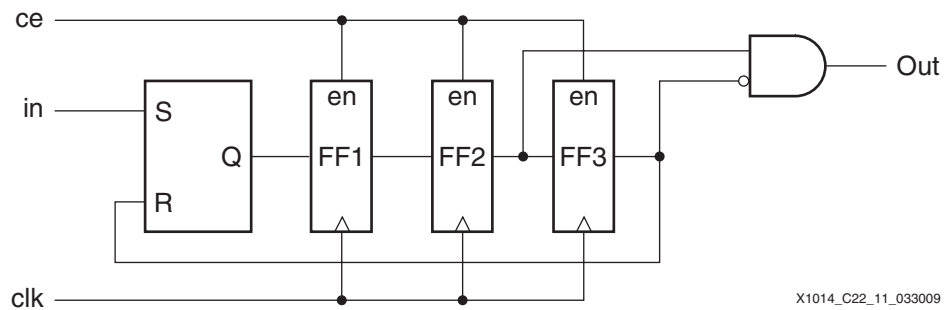


Figure 24-11: Synchronizing One-Shot

Figure 24-12 illustrates the operation of the synchronizing one-shot for both short and long input pulses. Regardless of the length of the input pulse, the output is a one-clock-wide pulse, synchronized to the clock, and occurring near the rising edge of the input pulse.

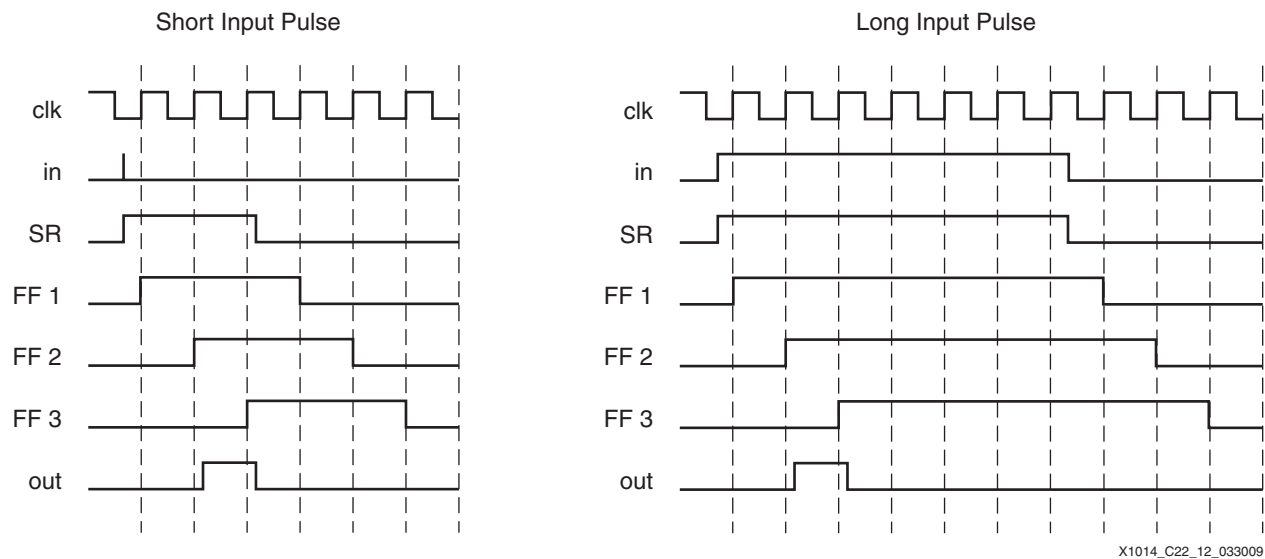
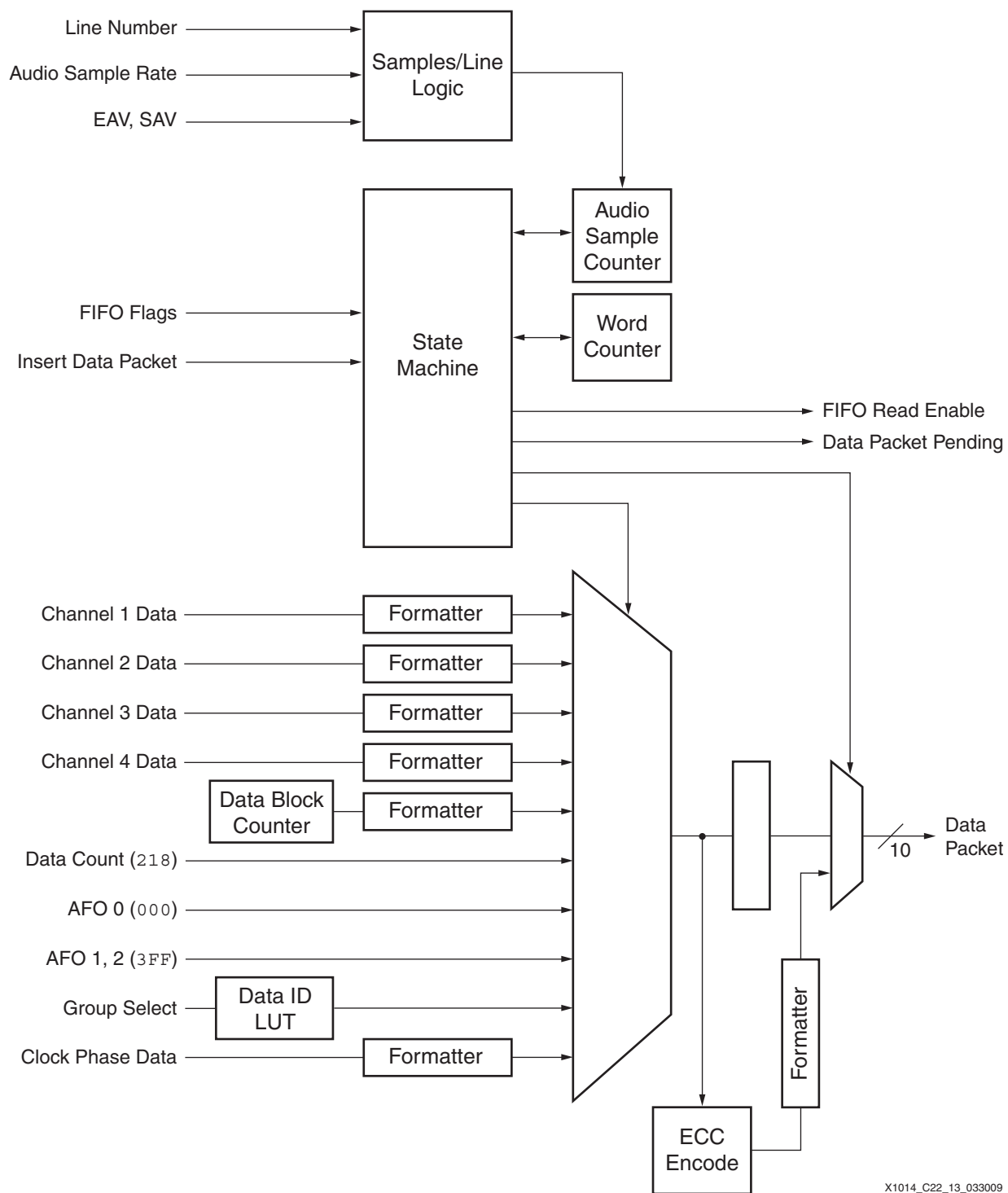


Figure 24-12: Synchronizing One-Shot Timing (ce = 1)

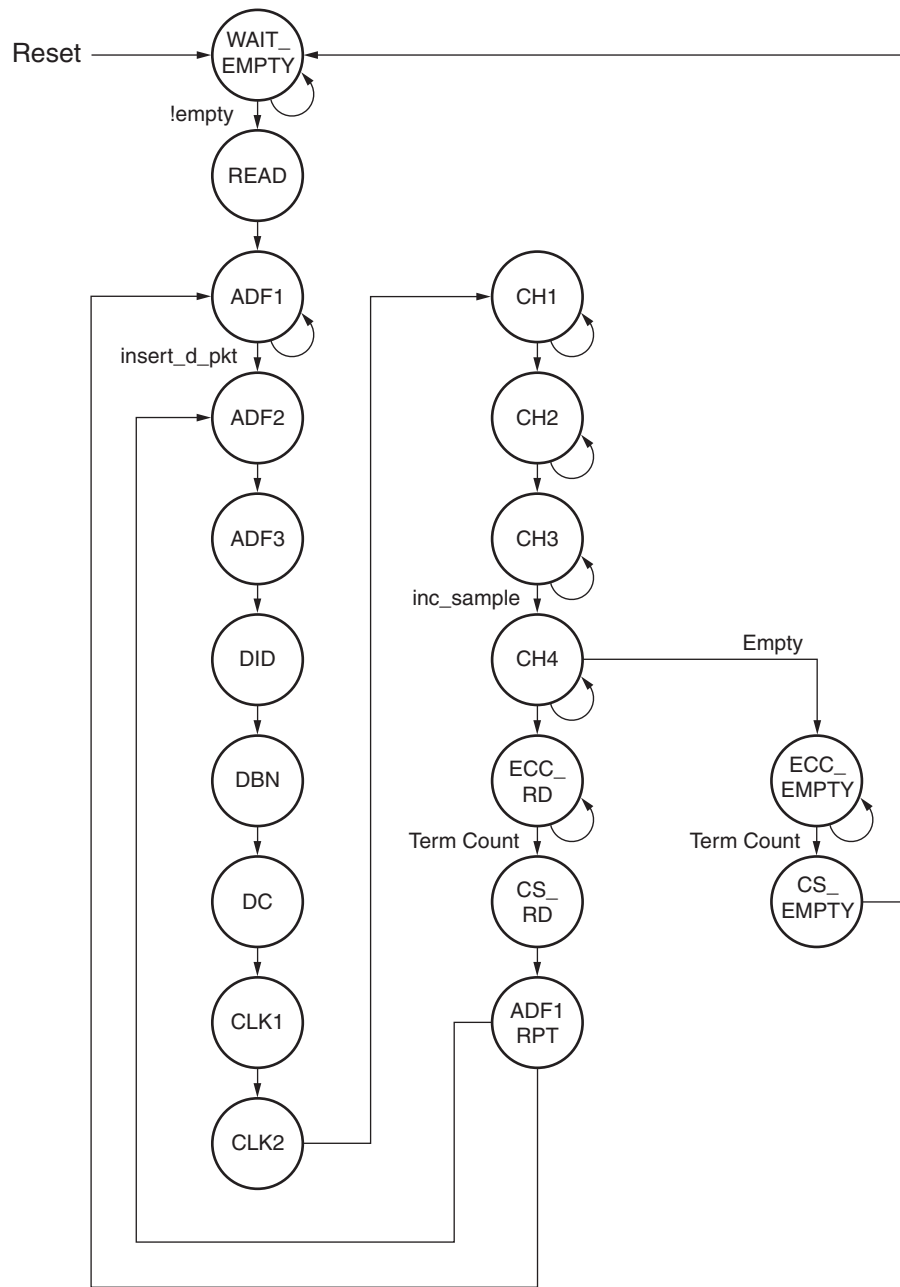
Audio Data Packet Construct (aud_data_constr)

This module constructs the audio packet. It consists of a state machine to control the timing and multiplexers to place the elements of the packet in sequence to be multiplexed into the video. Figure 24-13 shows a block diagram of this module. The state machine also controls handshaking with the ancillary data parser and the reading of the Audio FIFOs module. Figure 24-14 is a state diagram for the state machine. The next state for conditional branches is determined by the state of the Insert Data Packet handshaking signal, and whether or not the FIFOs are empty and the word counter has reached the terminal count.



X1014_C22_13_033009

Figure 24-13: Block Diagram of `aud_data_constr` Module



X1014_C22_14_033009

Figure 24-14: State Diagram for Data Packet Construction

The clock phase information and sample data for each channel are inputs that get formatted for the data packet. Several constant fields are part of the packet, such as data count and ancillary data flag (ADF) values. Group select is a static value used to look up the DID. A data block counter increments for each new packet, and the count is inserted in the DBN field. ECCs are generated simultaneously by the `ecc_encode` module described in [Error Correction Encode \(ecc_encode\)](#), page 575.

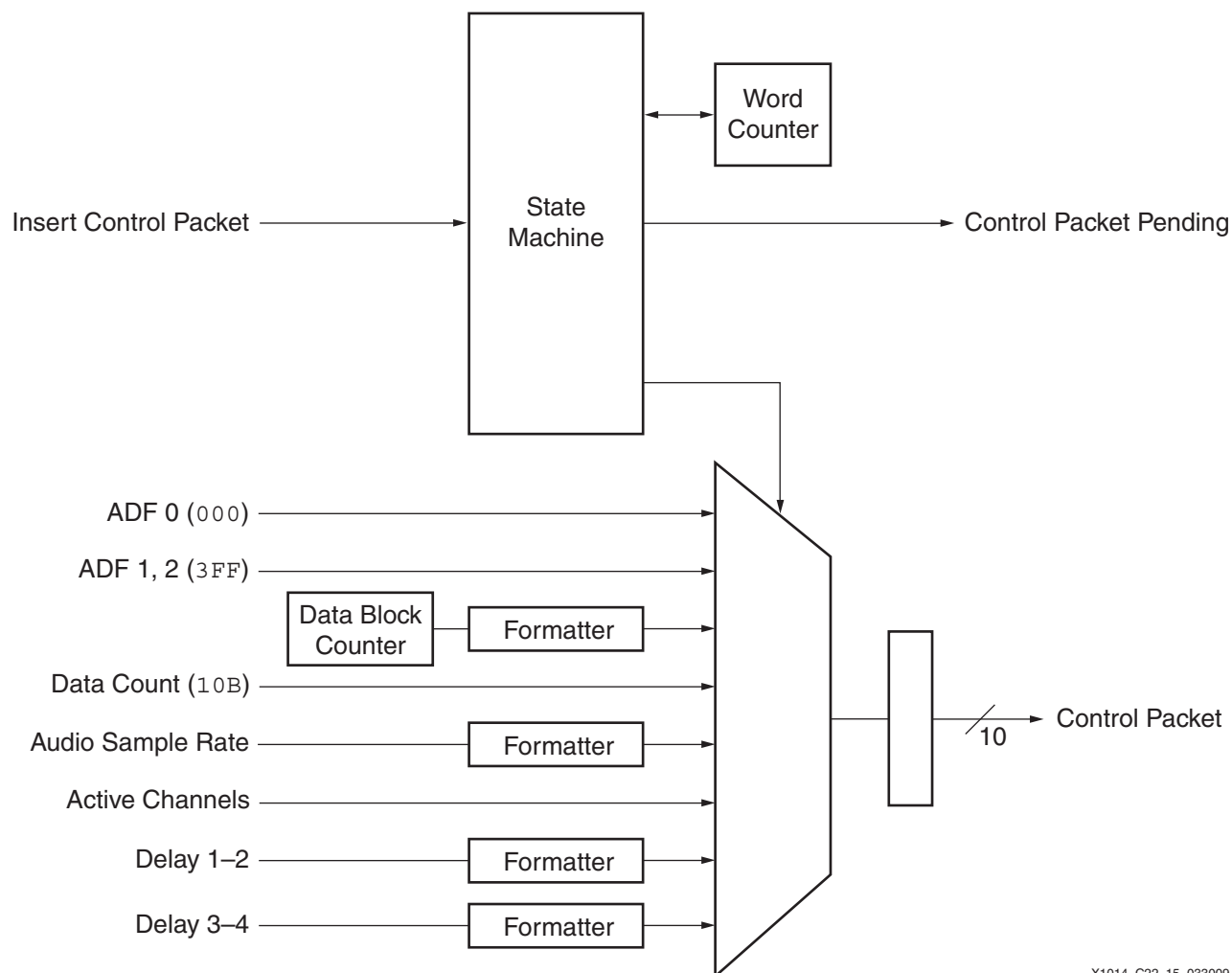
Two counters are controlled by the state machine: the audio sample counter and the word counter. The audio sample counter counts the number of audio samples that are inserted into each video line. The flags from the audio sample FIFOs are sent to the Ancillary Data

Parsing module so that the number of audio packets on each line can be adjusted to not exceed the number of audio samples per line as determined by the line standard and audio sample rate. The word counter counts words for multi-word fields of the packet, such as the audio samples. This count is also used to control formatting of multi-word fields.

Audio Control Packet Construct (aud_contr_constr)

The `aud_contr_constr` module forms audio control packets to append to ancillary data. This module operates in a manner similar to the Audio Data Packet Construct module. It consists of a state machine to control the timing and multiplexers to place the elements of the packet in sequence to be multiplexed into the video. A block diagram of the `aud_contr_constr` module is shown in [Figure 24-15](#). The state machine also controls handshaking with the ancillary data parser. [Figure 24-16](#) is a state diagram for the state machine.

Most packet information is received from inputs such as the audio sample rate, the active channels, and the delay of the channels with respect to the video. Several constant fields are part of the packet, such as data count and the ADF values. A data block counter increments for each new packet and the count is inserted in the DBN field. A word counter controlled by the state machine counts words for multi-word fields of the packet, such as delay.



X1014_C22_15_033009

Figure 24-15: Block Diagram of `aud_contr_constr` Module

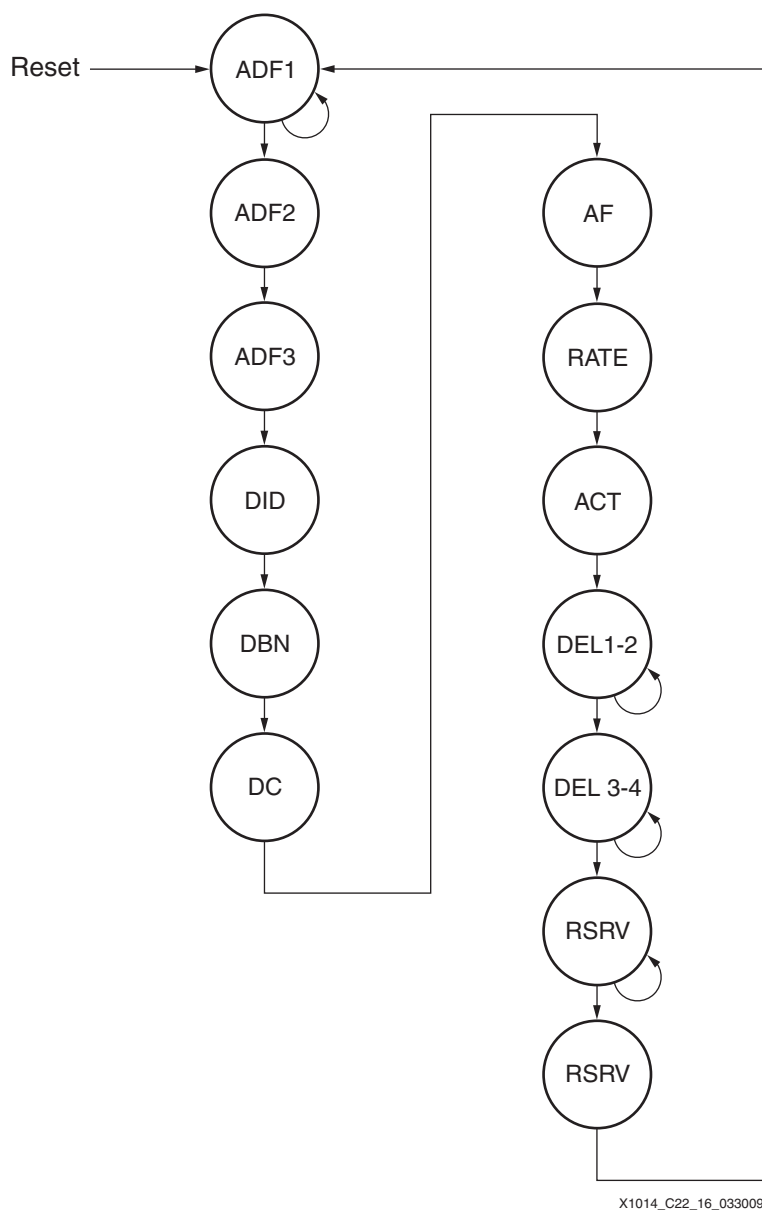


Figure 24-16: State Diagram for Control Packet Construction

Audio FIFOs (aud_fifos)

The `aud_fifos` module acts as a small buffer between the incoming sample stream, in which the samples are evenly spaced in time, and the Audio Data Packet Construct module, which retrieves samples at an irregular rate based on the samples to be embedded on a video line. The `aud_fifos` module contains five instances of the FIFO module `fifo_28x16`, one for each audio channel of the group and one for the clock phase information. A block diagram of the `aud_fifos` module is shown in Figure 24-17. Reads and writes are done in parallel to all FIFOs, with the exception of writes to the Clock Phase FIFOs.

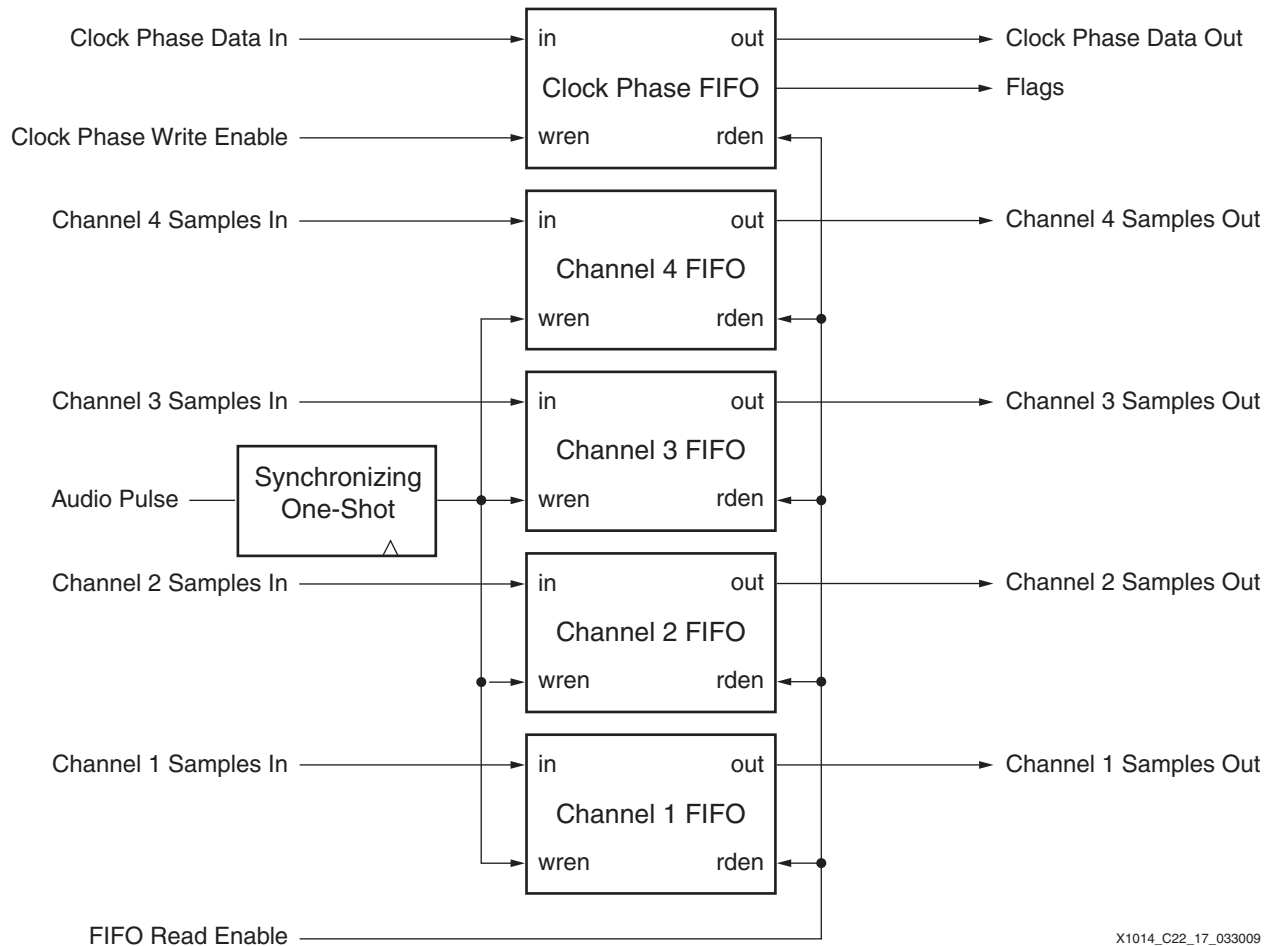
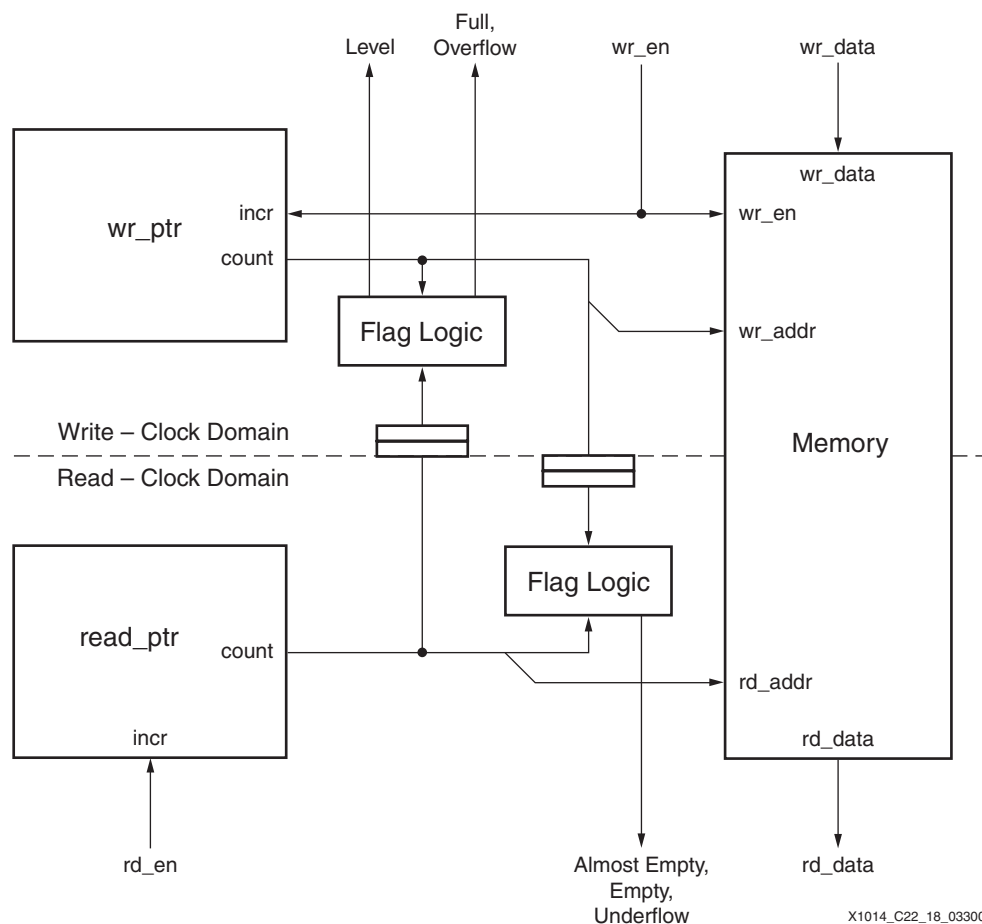


Figure 24-17: Block Diagram of `aud_fifos` Module

FIFO (`fifo_28x16`)

This basic FIFO module consists of SelectRAM™ memory built from LUTs. It is 28 bits wide by 16 locations deep. Each FIFO stores either video samples or clock phase information. In video sample FIFOs, each location contains one 24-bit video sample and the associated Z, C, U, and V flags. For the clock phase FIFO, each location contains the clock phase information associated with the corresponding audio samples. A block diagram of the `fifo_28x16` module is shown in [Figure 24-18](#).

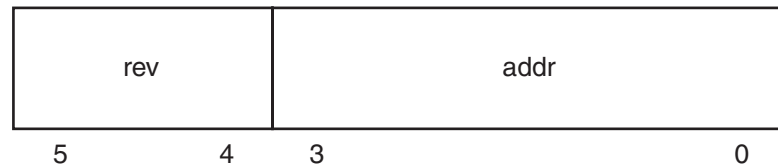


X1014_C22_18_033009

Figure 24-18: FIFO Block Diagram

The control logic is built specifically for use in the reference design. In particular, the control logic protects against underflow by not allowing the read pointer to increment when the FIFO is empty. At start-up, it is likely for read requests to occur when the FIFO is empty. Due to inherent latency in flag logic crossing clock domains, these read requests can happen before the empty flag has propagated to the controlling logic for the read operation. In such cases, the read operation, specifically the moving of the read pointer, is inhibited. This is important because there are occasions in which the FIFO must be emptied fully, but must also be guaranteed to work properly without rolling over when the next valid write data arrives.

As shown in Figure 24-19, the read and write pointers have two extra MSBs beyond the address. These indicate the number of passes or revolutions through the memory address space. In this way, when the read and write pointers point to the same address, it can be unambiguously determined whether this represents a full condition (write revolution does not equal read revolution) or an empty condition (write revolution equals read revolution).

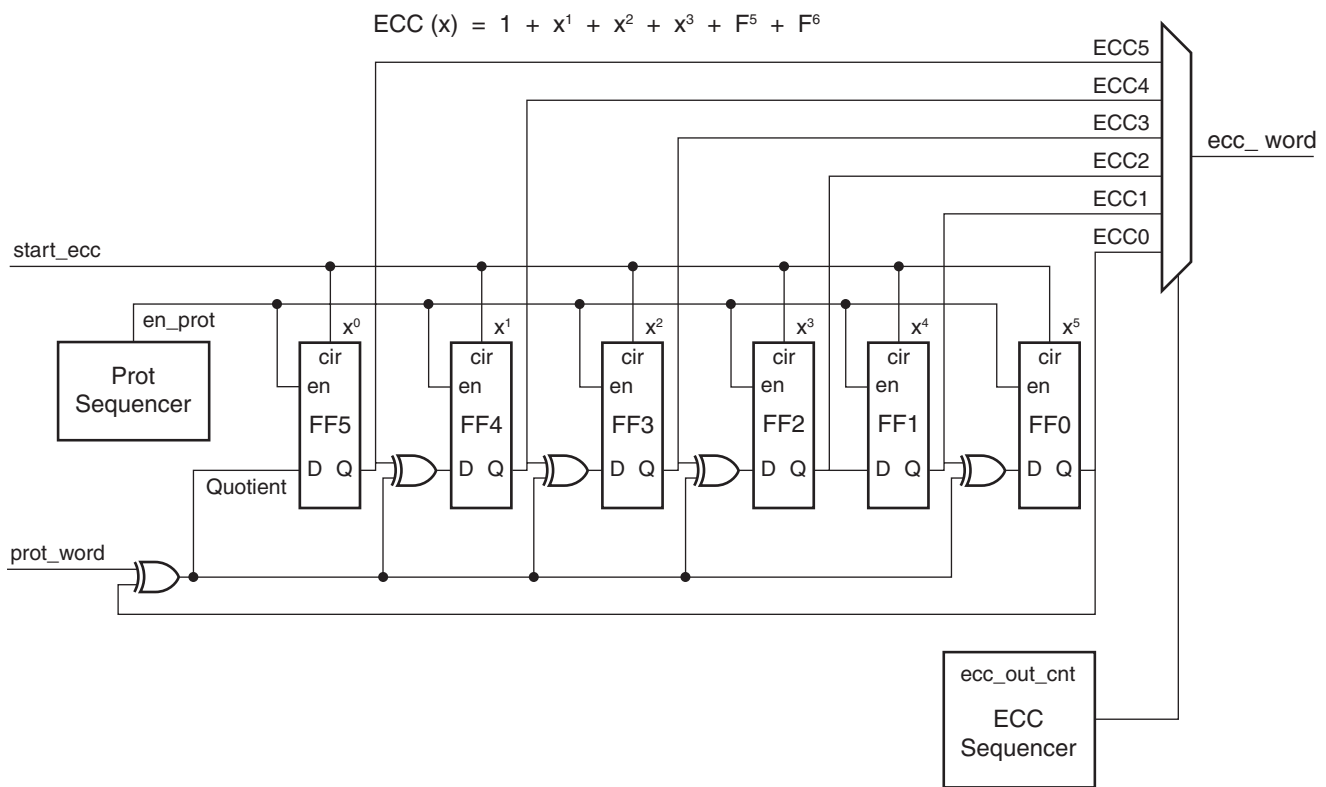


X1014_C22_19_033009

Figure 24-19: FIFO Pointer Format

Error Correction Encode (ecc_encode)

The `ecc_encode` module, referenced by the Audio Data Packet Construct module, creates the ECCs based on the BCH 31, 25 code specified in SMPTE 299M. For error correction of embedded audio, retransmission is not practical. Therefore, forward error correction is used in which redundant data is sent, making it possible for the receiving processor to correct errors in the corrupted data without retransmission. The ECCs cover all the preceding words of the packet, as shown in Figure 24-20.



X1014_C22_20_033009

Figure 24-20: ECC Protection

The ECC bits are calculated as the packet data passes word by word through the `ecc_encode` module by using the circuit given in SMPTE 299M. Figure 24-21 is a diagram of the ECC generation. There are eight copies of this circuit, one for each bit of the UDWs, resulting in eight 6-bit ECC words. These are formatted into six 8-bit data words in the audio data packet.

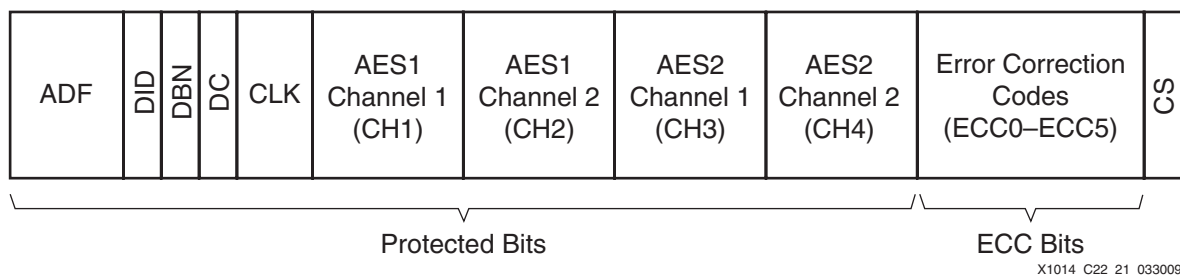


Figure 24-21: ECC Generation

Format 96 Multiplexer (format96_mux)

The format96_mux module consists of a multiplexer and two instances of the split96_deser module. A block diagram of this module is shown in Figure 24-22. When 96 KHz mode is enabled via split96_en on channel 1, two sequential samples are output on channels 1 and 2. Likewise, from the channel 2 input, two sequential samples are output on channels 3 and 4. When 96 KHz mode is disabled, the four input channels are passed straight through to the outputs.

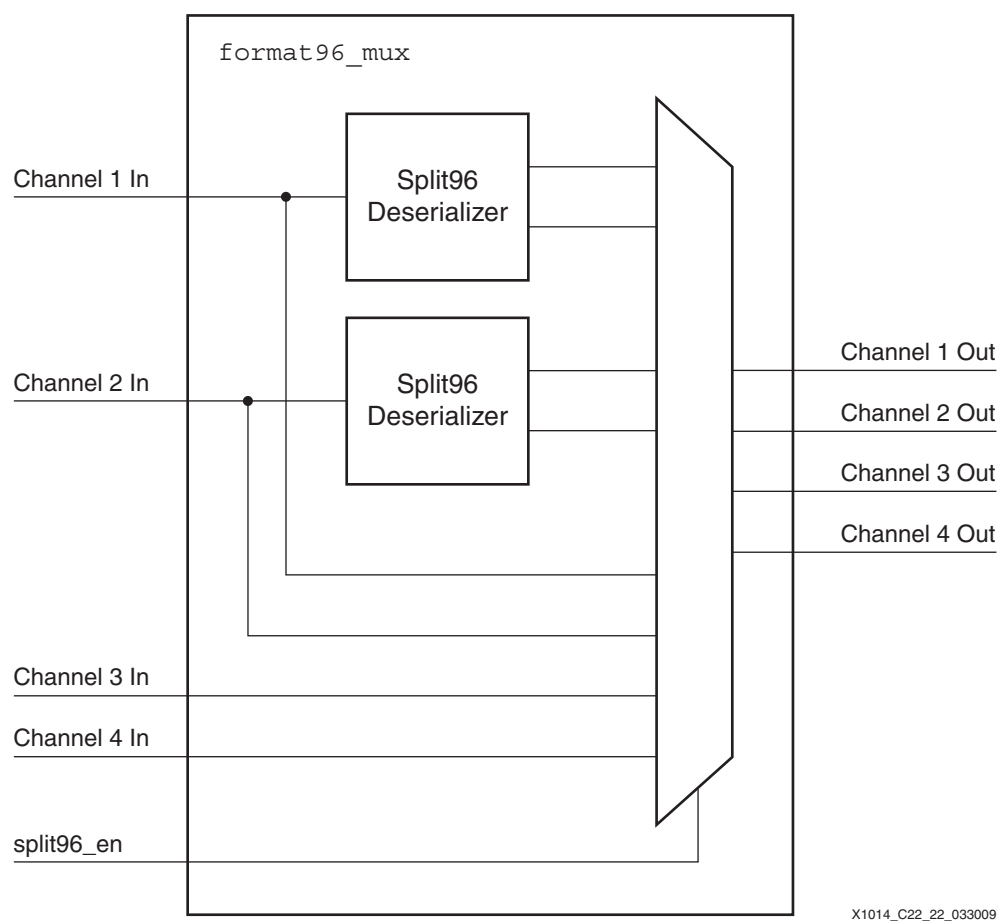


Figure 24-22: Block Diagram of format96_mux Module

Split 96 Deserializer (split96_deser)

The `split96_deser` module parallelizes two serial samples into the 96 KHz packet format with a corresponding doubling of the period of the valid signal. A block diagram of this module is shown in Figure 24-23. The timing for the `split96_deser` module is shown in Figure 24-24. The timing control block issues an output valid pulse at every other input valid pulse. It also controls the timing of the data register such that the first sample of the pair is output on output sample 1 and the second sample is output on output sample 2.

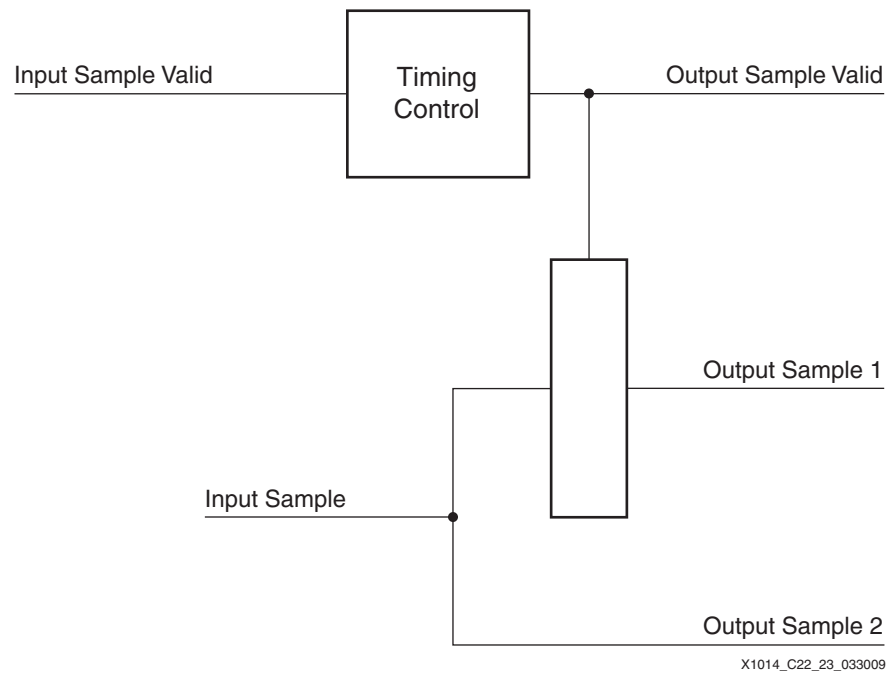
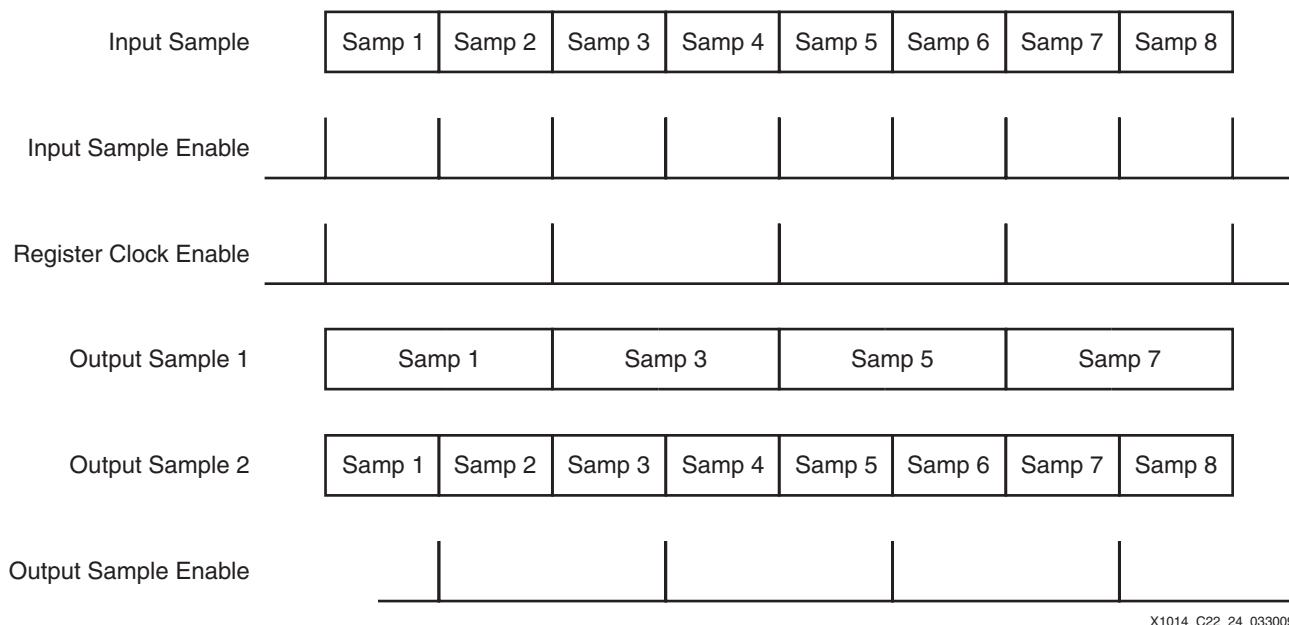


Figure 24-23: Block Diagram of `split96_deser` Module

Figure 24-24: Timing Diagram of `split96_deser` Module

Usage Models

This section provides some examples of ways in which the HD-SDI audio multiplexer can be used. In these examples, the video used can be either 3G-SDI level A or HD-SDI. For these video modes, up to 16 channels (8 channels for 96 KHz) can be accommodated by using four instances of the top-level design in series, as shown in Figure 24-25. Each instance is configurable to operate on any group via the `group_sel` input port. The instances can also be arranged in any order.

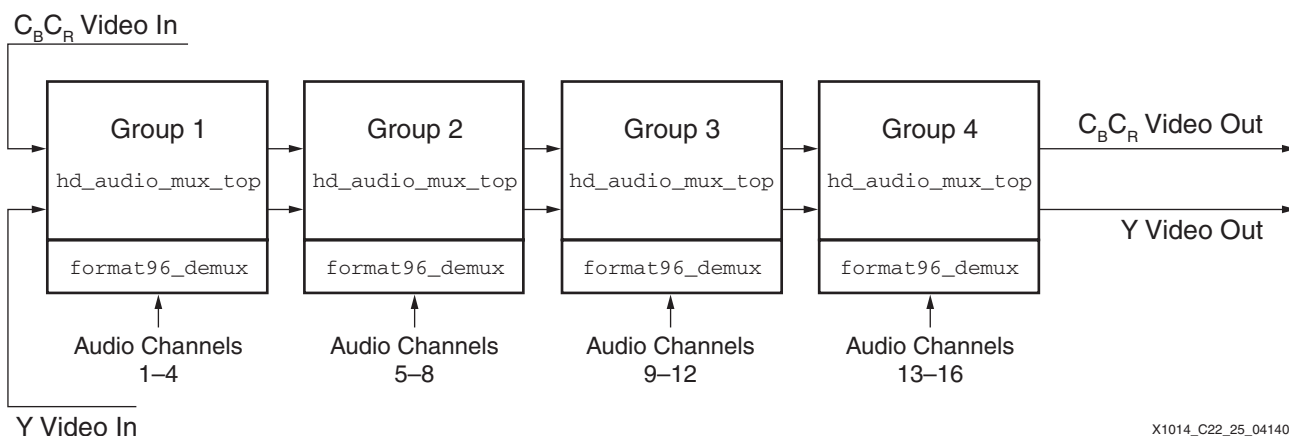
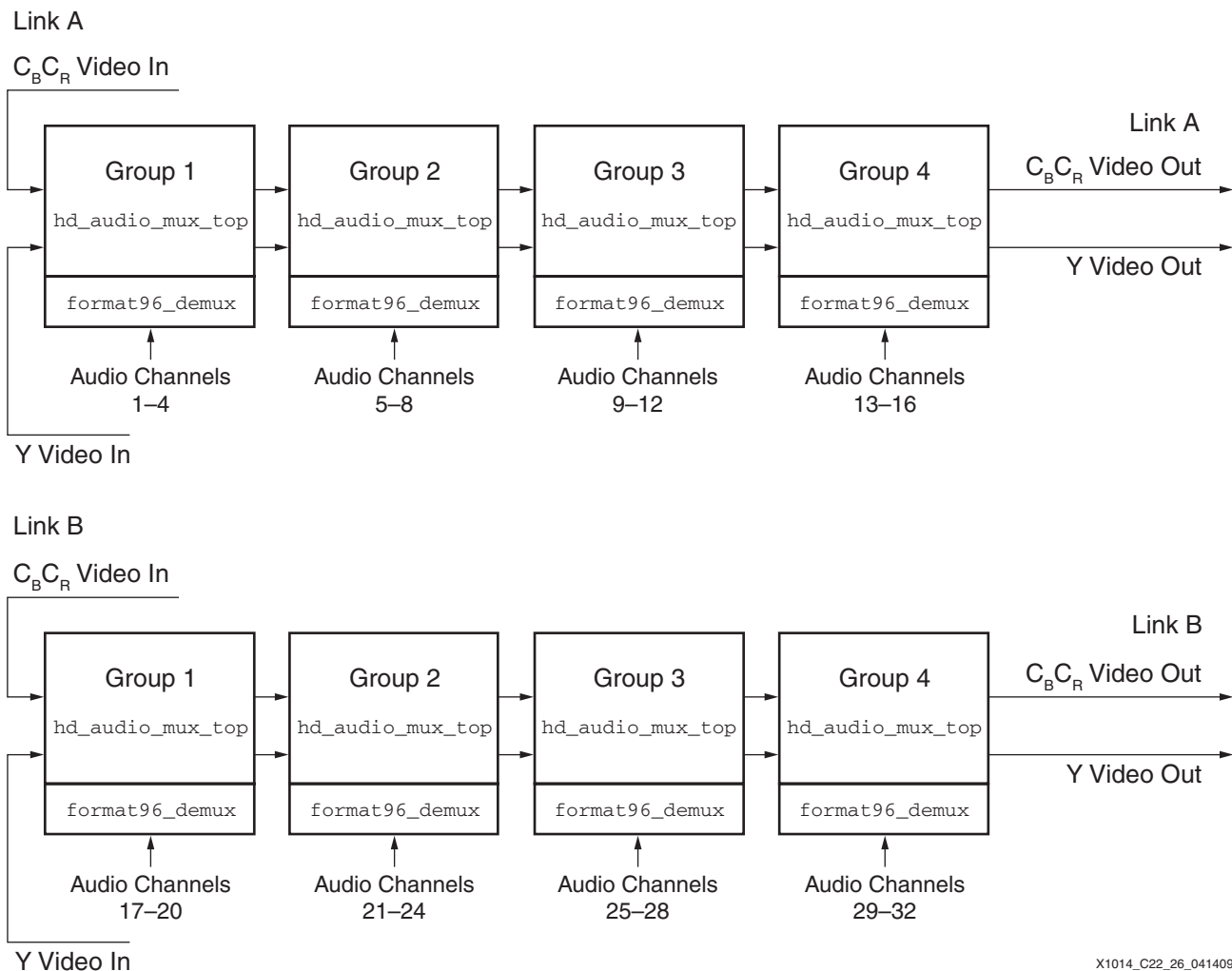


Figure 24-25: Multiplexing of Multiple Audio Groups Using Multiple Instances of the Reference Design

To support dual link HD-SDI and 3G-SDI level B, the audio demultiplexer is connected to link A of the two internal HD-SDI links and operates in HD-SDI mode. If more than 16 channels of 32 KHz to 48 KHz (8 channels of 96 KHz) are present, additional instances should be added to link B, as shown in Figure 24-26. The SMPTE standard for dual link

HD-SDI and 3G-SDI level B specify that link B is only used for audio if all available channels on link A have been utilized. For example, if there are 20 audio channels in 32 KHz to 48 KHz mode, 16 channels are mapped to link A, and 4 channels are mapped to link B.

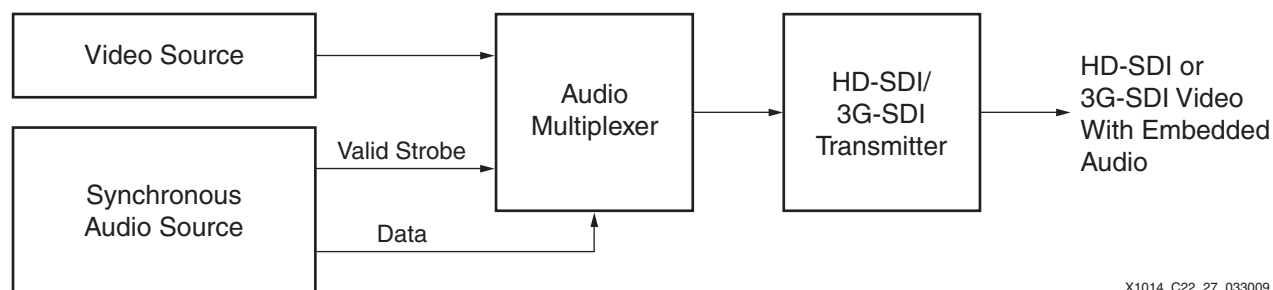


X1014_C22_26_041409

Figure 24-26: Additional Audio Channels on Dual Link HD-SDI and 3G-SDI Level B

Embedding Audio from a Synchronous Audio Source

If the audio source is synchronous with the video, audio can be embedded directly in a synchronous fashion, as shown in [Figure 24-27](#).

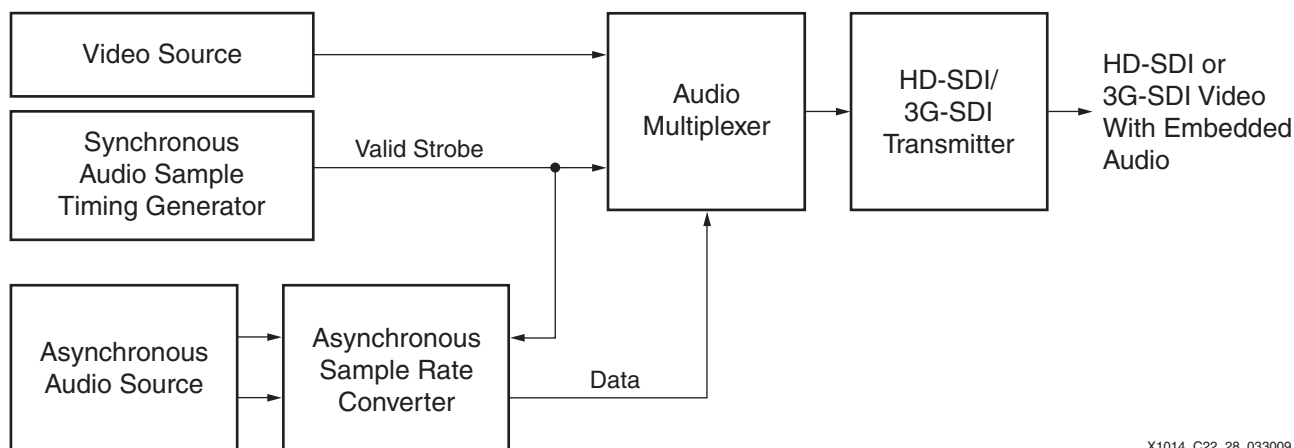


X1014_C22_27_033009

Figure 24-27: Embedding Synchronous Audio

Synchronizing Audio via Sample Rate Conversion

If synchronous audio is desired but the audio source is asynchronous with the video, it can be synchronized via sample rate conversion. An audio timing strobe synchronized to the video is required to act as a reference to the output side of the sample rate converter and to be used by the audio multiplexer. This is shown in Figure 24-28.

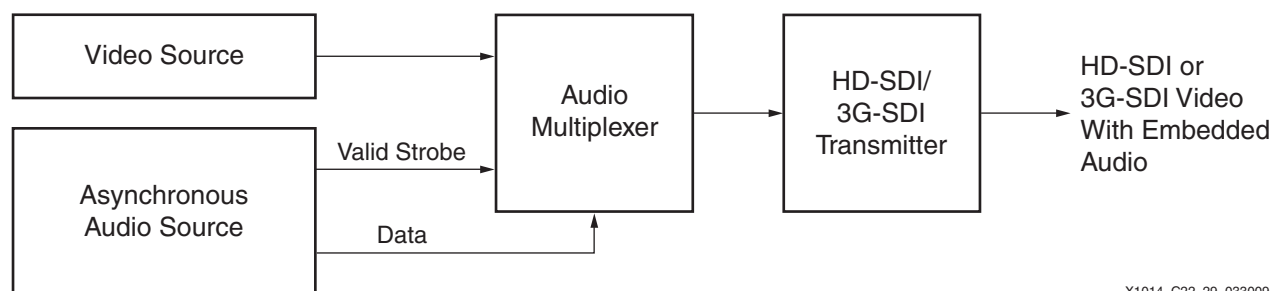


X1014_C22_28_033009

Figure 24-28: Synchronizing Audio with Sample Rate Conversion

Embedding Asynchronous Audio

The audio multiplexer supports embedding of asynchronous audio as allowed in SMPTE 299M (Figure 24-29). Audio sample timing is denoted by the clock phase information.



X1014_C22_29_033009

Figure 24-29: Embedding Asynchronous Audio

Replacing Audio Embedded in Received Video

Audio embedded in video received from a remote source can be replaced from an alternate audio source (Figure 24-30). The audio group specified for the audio multiplexer has all existing audio packets marked for deletion. New audio packets for the audio source are embedded in the specified audio group. All other ancillary data in the received video is passed unaltered. The audio can optionally be passed through a sample rate converter.

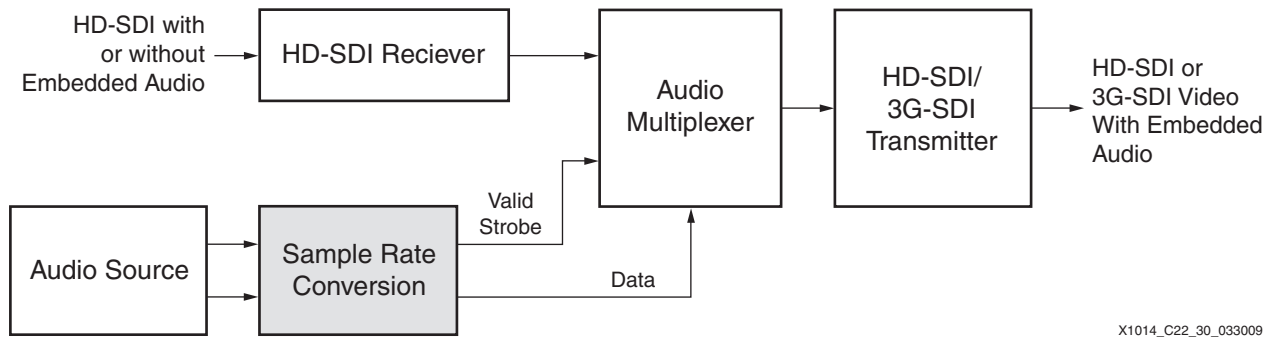


Figure 24-30: Replacing Embedded Audio

Reembedding Audio after Separate Video/Audio Processing

In some applications, audio and video should be processed separately from one another (Figure 24-31). For example, a video frame synchronizer drops complete frames of video. However, the audio cannot tolerate the equivalent dropping of samples, and so must be processed separately. In such cases, the audio is de-embedded from the video and processed in a separate path. After processing is completed, the audio and video can be combined again in the audio multiplexer.

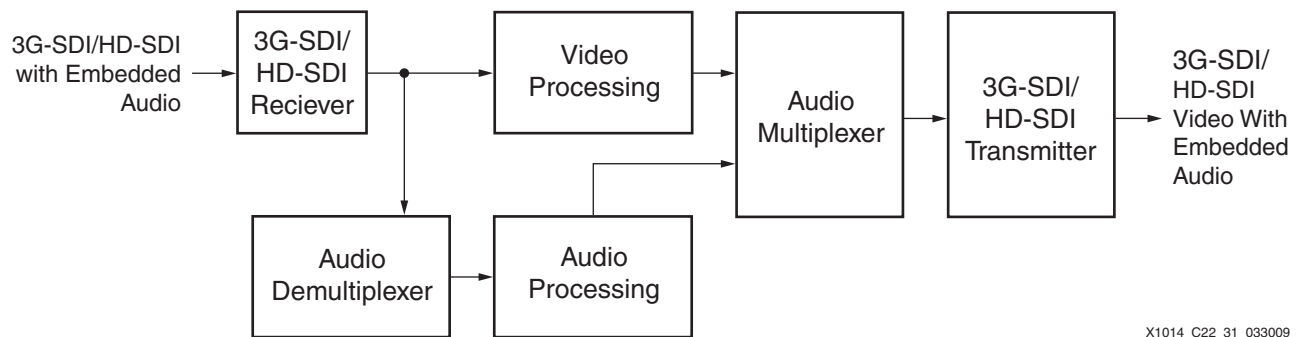


Figure 24-31: Separate Audio and Video Processing

FPGA Resources

The reference design is implemented and hardware verified in a Virtex-5 device, but it can also be used with other Xilinx FPGA families. The resources required for the reference design in a Virtex-5 device are shown in Table 24-6. The formatter for 96 KHz samples is independent of the rest of the audio multiplexer so that it can be included only where needed.

The results shown in Table 24-6 were obtained using ISE[®] software, version 9.2i SP2. Timing was constrained for a processing clock of 297 MHz. Timing was met for -1, -2, and -3 speed grade devices.

Table 24-6: **FPGA Resources**

Top Level Module	Flip-Flops	LUTs	Block RAM	DSP Blocks
hd_audio_mux_top	676	685	0	0
format96_mux	51	157	0	0

Design Files

The reference design for the audio demultiplexer is available at <https://secure.xilinx.com/webreg/clickthrough.do?cid=119758>. Open the ZIP archive and extract the file xapp1014_c24_3G_audio_mux.zip.

Conclusion

This chapter explains embedded audio in HD-SDI, 3G-SDI, and dual link HD-SDI, the standards used to define it, and the data formats of the audio packets. The reference design of this chapter is useful for embedding audio into 3G-SDI and HD-SDI video. It can be used in a variety of applications and can be implemented in all Xilinx FPGA families. The reference design supports multiple video standards and multiple audio sample rates, including 96 KHz, in both synchronous and asynchronous formats. The modularity of the design supports from one to four audio groups by adding multiple instances in series. Metadata is also encoded in audio control packets and multiplexed into the video stream.

Audio Demultiplexer for 3G-SDI, Dual-Link HD-SDI, and HD-SDI Video

Introduction

In many applications, audio is embedded in the video stream to facilitate the transport of the combined audio and video stream. For HD-SDI video, the audio information is carried in the HANC portion of the video. The specification for this is given in SMPTE 299M. Standards for 3G-SDI and dual link HD-SDI also reference SMPTE 299M as the format for embedded audio data. The most commonly used audio sample rate for video is 48 KHz. A revision of the SMPTE standard, underway at the time of publication but not finalized, specifies a modified data packet format for carrying 96 KHz samples.

The audio can be demultiplexed from the video for processing or for output. This chapter describes how this demultiplexing is done and presents a hardware-verified reference design that implements an audio demultiplexer compatible with HD-SDI, 3G-SDI, and dual link HD-SDI video. The embedded audio standard for SD-SDI is substantially different and is not supported by the reference design.

Features

Some of the features of the audio demultiplexer are:

- Single audio group granularity: One instance of the demultiplexer de-embeds the four channels (2 channel pairs) of a single audio group of 48 KHz audio or two channels (one channel pair) of 96 KHz audio samples. The audio group number is an input to the design. Audio from multiple audio groups can be de-embedded by using multiple instances of the basic design in a daisy-chain fashion.
- Support for 3G-SDI level A and level B (SMPTE 425M) and dual link HD-SDI (SMPTE 372M).
- Multiple HD-SDI standards: The demultiplexer supports SMPTE 274M, SMPTE 295, SMPTE 296M, and SMPTE 260 line standards.
- 24-bit audio support: Audio is output as 24 bits plus Z, V, U, and C flags with an audio valid strobe to indicate timing. There are four such outputs, one for each channel in the group.
- Multiple sample rates: 32 KHz, 44.1 KHz, 48 KHz, and 96 KHz are supported. The base design supports 32 KHz, 44.1 KHz, and 48 KHz. The reference design includes an optional formatter module that adds support for 96 KHz.
- Synchronous and asynchronous audio support: A recovered audio sample pulse can be input directly, or alternately, an audio valid pulse is generated based on embedded clock phase data.

- De-embedding audio control packets: Information from audio control packets is available on output ports.
- Support of many Xilinx® device families: The reference design has been hardware verified on the Virtex®-5 FPGA but uses features common to all Xilinx FPGAs.
- Compatible with forward error correction: Error correction using embedded error correction codes (ECCs) can be done prior to demultiplexing using the reference design of [Chapter 22, Error Correction for HD-SDI Embedded Audio](#).

Embedded Audio

Audio information is embedded in SDI data streams in the form of packets that are placed in the horizontal blanking period of the $C_B C_R$ data stream. The general specification for HD-SDI video formats and timing is given in SMPTE 292M. 3G-SDI is specified by SMPTE 425M and dual link HD-SDI by SMPTE 372M. Among other things, these standards specify how timing references and line numbers are transmitted. The embedded audio format for all of these video standards is given in SMPTE 299M. Thus, the formats for HD-SDI also apply to 3G-SDI level A and level B and dual link HD-SDI.

Embedded audio packets are just a few of many different types of ancillary packets that might be inserted in the video stream during blanking periods. SMPTE 291M specifies the general format of ancillary packets (see [Figure 25-1](#)). Common elements include a three-word ancillary data flag, a data identity (DID) to specify the packet type, a data block number (DBN), and data count (DC) fields. A checksum (CS) is the final field of every ancillary packet.

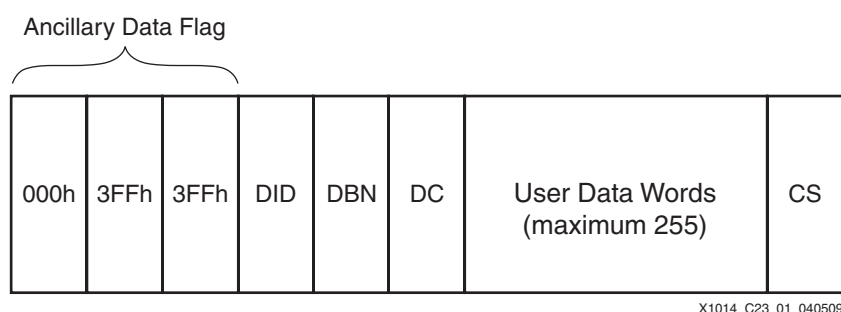


Figure 25-1: Ancillary Data Packet

There are two types of embedded audio packets: audio data packets and audio control packets. Audio data packets contain audio samples for the various channels. Audio control packets contain various metadata fields describing such things as sample rates and audio processing delay. The DID values for embedded audio in HD-SDI are shown in [Table 25-1](#). Each audio data packet contains four samples, one from each channel in a group. Likewise, each audio control packet contains metadata that applies to one audio group.

Table 25-1: DID Values for 3G-SDI/HD-SDI Embedded Audio Packets

Group	Audio Data Packets	Audio Control Packets
1	0x2E7	0x1E3
2	0x1E6	0x2E2
3	0x1E5	0x2E1
4	0x2E4	0x1E0

While ancillary data packets are required to be contiguous and begin immediately after the EAV, the exact location of packets of a given type is not specified. Therefore, to demultiplex audio from the video stream, all ancillary data types must be examined to determine if the packet is of interest. That is, the DID field must be checked and the DC must be read to determine the length of the packet, and thus, where to look for the next packet.

Audio Data Packets

There are two data packet formats, one for the 32 KHz to 48 KHz sampling rates, and one for the 96 KHz sampling rate. The main difference between the two is that the 96 KHz format has two sequential samples of two channels in a packet whereas the 32 KHz to 48 KHz format has one sample of four channels in each packet.

32 KHz to 48 KHz Audio Data Packets

The format of audio data packets is shown in Figure 25-2. These have the standard ancillary packet format. The value of DID depends on the audio group, as specified in Table 25-1. DC is always 0×218 . The DBN increments for each packet of the same DID. In addition, the audio data packet contains 24 user data words (UDWs), a two-word clock phase field (CLK), six ECC words, and four audio words (two stereo pairs). The fields encompassed by the UDWs are described in this section.

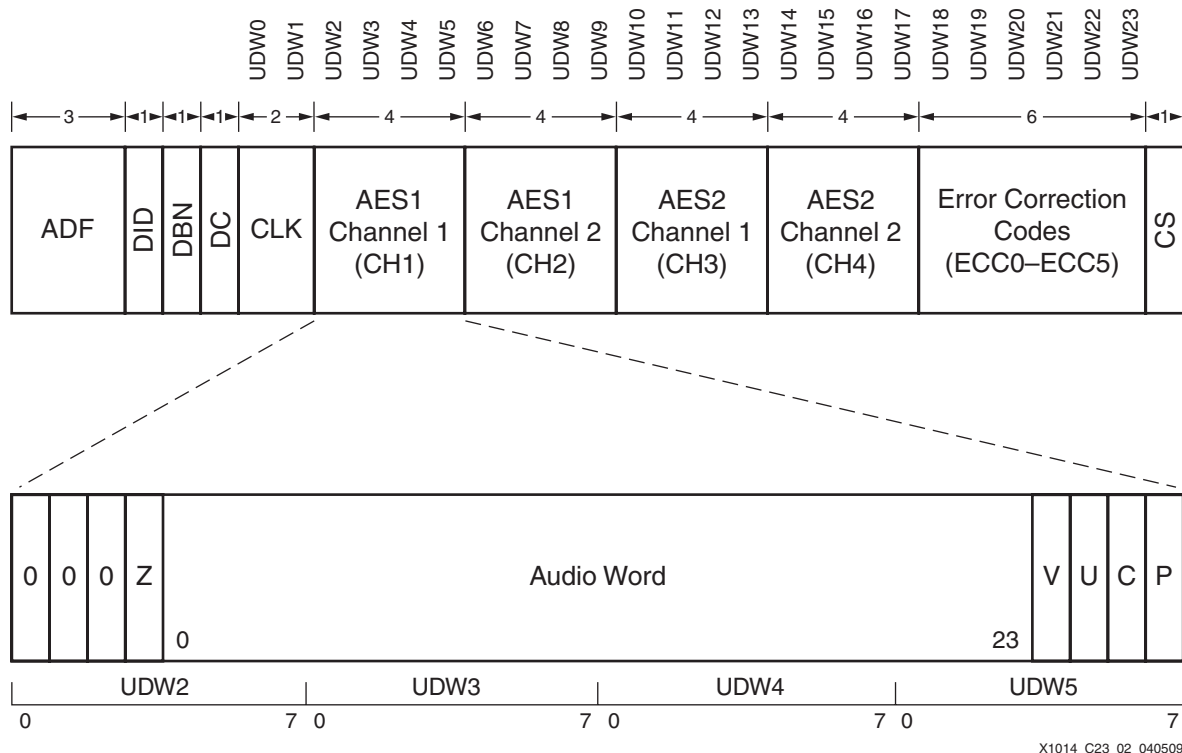


Figure 25-2: Audio Data Packet (32 KHz to 48 KHz)

Audio data packets can be multiplexed onto any video line, with the exception of the lines following the synchronous switching points. The samples must be more or less evenly distributed. The number of audio data packets on any line containing packets can vary by at most one.

Audio Sample Words

The audio sample words are in AES format. Each AES word (AES_n) includes Z, V, U, C, and P bits. The Z bit is present for CH1 and CH3 only. The audio word itself spans the four UDWs, as shown in [Figure 25-2](#). All of the words in the audio data packet are 10 bits wide. The eight LSBs contain the actual data. Bit 8 contains even parity for the eight LSBs. Bit 9 contains the inverse of bit 8.

Audio Clock Phase Data

The CLK field contains information about the timing of the audio clock with respect to the video lines. Specifically, the audio clock phase is the number of video clock times (rounded to the nearest integer) between the samples contained in the packet and the EAV immediately preceding the arrival of the audio clock for those samples. Due to the prohibition of packets in the line following the synchronous switching point, the audio sample packet sometimes occurs more than a line time after the EAV to which it is referenced. For these cases, a multiplex position flag (mpf) is asserted. The mpf flag indicates when the given clock phase is from the next-to-last EAV. This implies that there must be at least two video lines of latency in the demultiplexing process.

SMPTE 299M states that valid CLK data is required. However, it is common knowledge that some equipment does not produce the CLK field correctly, and most demultiplexing equipment completely ignores this field. The reference design described in this chapter provides the option of whether or not to use CLK. Clock phase data is essential for asynchronous embedded audio that is provided in the SMPTE standard, but seldom used. If clock phase data is not used, a synchronous audio sample clock must be derived from the video and provided as the output audio sample clock.

Error Correction Codes

UDWs 18 to 23 are ECCs for forward error correction of audio data packets. The reference design of this chapter does not support error correction. For a reference design and detailed explanation of the error correction process, refer to [Chapter 22, Error Correction for HD-SDI Embedded Audio](#). Error correction must be done prior to demultiplexing audio data packets.

96 KHz Audio Data Packets

The data packets for 96 KHz differ from those for 32 KHz to 48 KHz in that they contain two sequential samples from two channels. The format of the 96 KHz data packet is shown in [Figure 25-3](#).

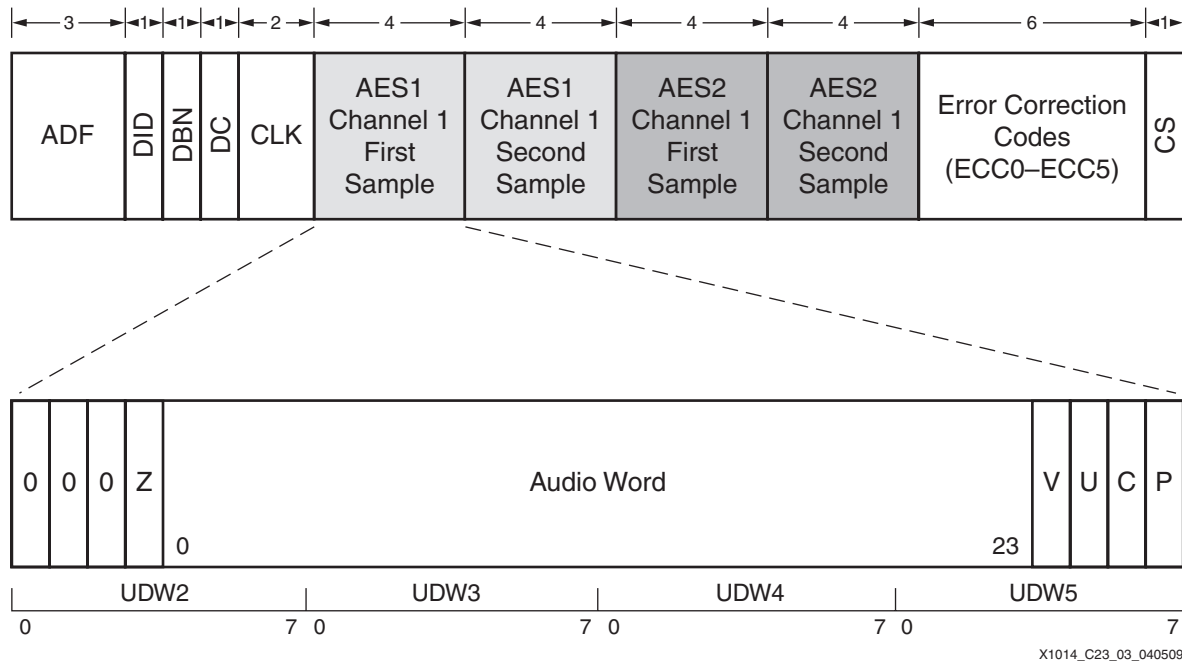


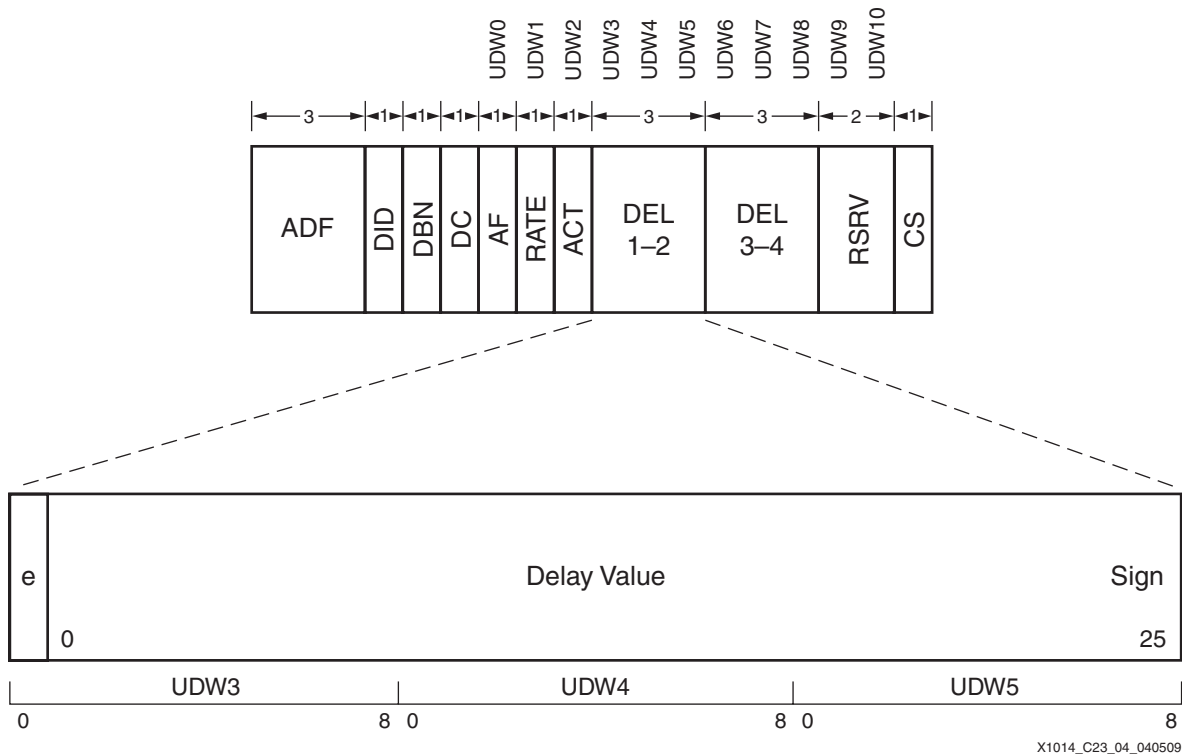
Figure 25-3: Audio Data Packet (96 KHz)

The 96 KHz data packet format has these implications:

- Because there are only two unique audio channels per packet, there are only two channels per group. Therefore, the four defined audio groups can contain only eight audio channels, compared to 16 channels for the 32 KHz to 48 KHz format. For HD-SDI and 3G-SDI level A, this means that only eight channels of 96 MHz audio are uniquely defined. For dual link HD-SDI, each link can carry eight unique audio channels. Therefore, 16 unique audio channels can be identified.
- Because there is a single clock phase field, it can apply to only one of the two sequential samples. The clock phase field is defined to apply to the second sequential sample. No explicit clock phase data is transmitted for the first sample.
- On average, the same number of packets per group is transmitted on each video line as for 48 MHz sampling. In other words, the number of packets in the 96 KHz format is the same as for the 48 KHz format. However, the number of channels carried in those packets is halved while the number of samples per channel is doubled.

Audio Control Packets

Audio control packets are transmitted once per field in an interlaced system or once per frame in a progressive system. They are placed in the blanking period of the second line after the synchronous switching point. The packet contains information about the channels in one group. Thus, if multiple audio groups are used, multiple audio control packets should be sent. Audio control packets have the standard ancillary format with 11 UDWs (see Figure 25-4). The words are 10 bits wide, with most UDWs having 9 bits of data. ACT is the only UDW with parity (on bit 8). In all UDWs, the MSB (bit 9) is the complement of bit 8.



X1014_C23_04_040509

Figure 25-4: Audio Control Packet

The DID value depends on the audio group, as shown in Table 25-1. DC is always 0x10B. The AF number, if used, indicates the position of the field with respect to the audio frame sequence. If not used, the AF number is set to all zeros. The sampling rate (RATE) indicates the sample rate of the embedded audio. The LSB is the asynchronous flag, asx. When asserted, this flag indicates that the audio is asynchronous with the video. The rate codes are shown in Table 25-2. A synchronous sample rate of 48 KHz is almost exclusively used in the industry.

Table 25-2: Sample Rate Codes

RATE[3:1]	Sample Rate (KHz)
000	48.0
001	44.1
010	32.0
100	96.0
111	Free running
101, 110	Reserved

The ACT word indicates which of the four channels in the group is active. Bit 0 corresponds to CH1, bit 1 to CH2, and so forth. A logic 1 means that the channel is active. The delay (DEL) fields indicate the amount of accumulated audio delay relative to video in the indicated channel pair. The delay is measured in audio sample intervals as a 26-bit two's-complement number. Positive values indicate that the video leads the audio. The

LSB of the entire field is an enable bit. When this bit is set High, the delay value is valid. UDWs 9 and 10 are reserved and are set to 0.

Reference Design

The reference design for the audio demultiplexer is for one audio group. Multiple instances can be connected in parallel to demultiplex audio for multiple audio groups. The video stream, including ancillary data, is not altered. Error correction, if used, must be done prior to demultiplexing. A block diagram of the audio demultiplexer is shown in [Figure 25-5](#).

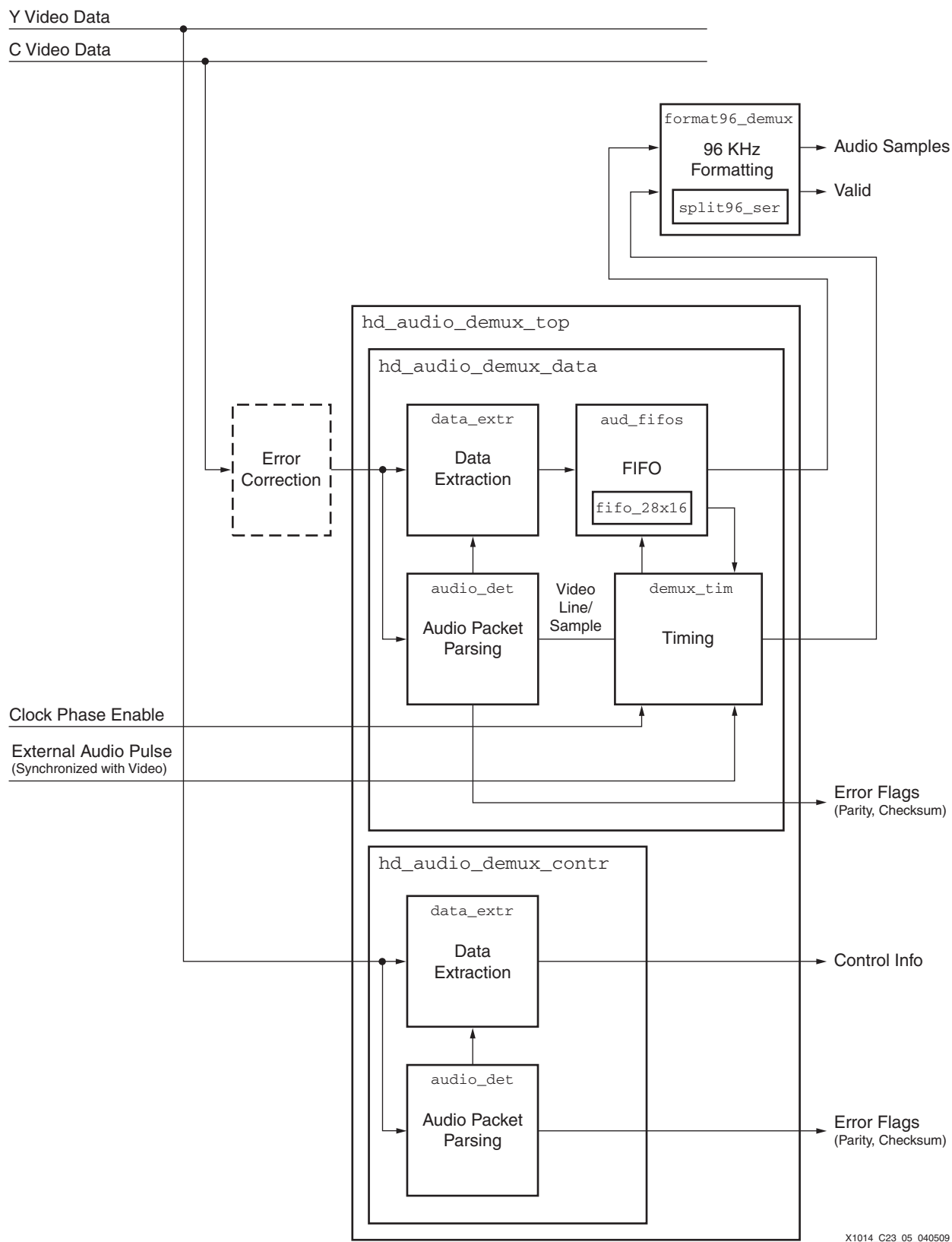


Figure 25-5: 3G-SDI/HD-SDI Audio Demultiplexer Block Diagram

There are two principal modules within the top level: `hd_audio_demux_data` for de-embedding audio data packets from the $C_B C_R$ video stream, and `hd_audio_mux_contr` for de-embedding audio control packets from the Y video stream. These blocks contain lower level blocks that perform various functions.

Each instance of the Audio Packet Parsing block monitors the video data looking for audio packets. One instance monitors the $C_B C_R$ data stream of data packets while another monitors the Y video stream for control packets. The Audio Packet Parsing block sends control information to the Data Extraction block that extracts the packet data from the video stream. While information from audio control packets is output directly, samples from audio data packets are loaded into a FIFO along with the corresponding clock phase information.

The timing block controls when the samples from the FIFO are output. There are two timing modes: internal and external. The Clock Phase Enable input controls the timing mode. For internal timing mode, output sample timing is created by comparing the clock phase data from the sample in the FIFO to video line and sample information that is extracted by the Audio Packet Parsing block, and delaying the data by a few line times. In this mode, the Enable output pulse indicates the moment when each new set of samples is valid on the Audio Samples output. In external timing mode, the output of the FIFO is controlled by the External Audio Pulse input. At each pulse, a new set of samples is output from the FIFO. The External Audio Pulse must be derived from a clock synchronized with input video to avoid underflowing or overflowing the FIFO.

Inputs and Outputs

There are two modules at the top level, `hd_audio_mux_top` and `format96_mux`. [Table 25-3](#) lists the inputs and outputs of the `hd_audio_demux_top` module.

Table 25-3: Inputs and Outputs of `hd_audio_mux_top` Module

Signal	Direction	Description
<code>vid_clk</code>	In	This is the video clock.
<code>rst</code>	In	This is the asynchronous reset.
<code>vid_ce</code>	In	This is the video clock enable.
<code>vid_samp_ce</code>	In	This is the video sample-rate clock enable. This is the same as <code>vid_ce</code> , except in the case of 12-bit video samples.
<code>vid_y_in[9:0]</code>	In	This is the Y input video stream.
<code>vid_c_in[9:0]</code>	In	This is the $C_B C_R$ input video stream.
<code>group_sel[1:0]</code>	In	This is the audio group number: 0 = Group 1 1 = Group 2, etc.
<code>ext_audio_pulse</code>	In	This is the timing pulse for output samples. This signal is enabled when <code>clk_phase_en</code> = 0 and disabled when <code>clk_phase_en</code> = 1.
<code>clk_phase_en</code>	In	This input enables the use of clock phase data to determine output timing: 0 = Uses <code>ext_audio_pulse</code> for output sample timing 1 = Uses clock phase data

Table 25-3: Inputs and Outputs of `hd_audio_mux_top` Module (Cont'd)

Signal	Direction	Description
audio_valid	Out	This is the audio valid strobe. Logic High indicates that new valid audio data is present on the outputs. This signal applies to all four channels and is used internally to read data from the FIFOs. When <code>clk_phase_en</code> = 1, the <code>audio_valid</code> signal reflects the timing derived from the clock phase enable data. When <code>clk_phase_en</code> = 0, the <code>audio_valid</code> signal is <code>ext_audio_pulse</code> after being synchronized to <code>clk</code> and made one <code>clk_en</code> period long.
ch1_audio[23:0]	Out	Channel 1 audio sample data.
ch1_z	Out	Channel 1, 2 AES audio Z flag.
ch1_c	Out	Channel 1 channel status data.
ch1_u	Out	Channel 1 user data bit.
ch1_v	Out	Channel 1 valid bit.
ch2_audio[23:0]	Out	Channel 2 audio sample data.
ch2_c	Out	Channel 2 channel status data.
ch2_u	Out	Channel 2 user data bit.
ch2_v	Out	Channel 2 valid bit.
ch3_audio[23:0]	Out	Channel 3 audio sample data.
ch3_z	Out	Channel 3, 4 AES audio Z flag.
ch3_c	Out	Channel 3 channel status data.
ch3_u	Out	Channel 3 user data bit.
ch3_v	Out	Channel 3 valid bit.
ch4_audio[23:0]	Out	Channel 4 audio sample data.
ch4_c	Out	Channel 4 channel status data.
ch4_u	Out	Channel 4 user data bit.
ch4_v	Out	Channel 4 valid bit.
audio_frame[8:0]	Out	This is the audio frame number.
audio_rate[2:0]	Out	This is the audio sample rate: 000 = 48 KHz 001 = 44.1 KHz 010 = 32 KHz 100 = 96 KHz
asx	Out	This control packet value is a synchronous data flag: 0 = Synchronous 1 = Asynchronous

Table 25-3: Inputs and Outputs of **hd_audio_mux_top** Module (Cont'd)

Signal	Direction	Description
active_chan[3:0]	Out	These are channel valid bits that denote the active channels when High: bit[0] = CH1 bit[1] = CH2 bit[2] = CH3 bit[3] = CH4
delay_1_2[25:0]	Out	This control packet value is the audio processing delay for channels 1 and 2.
delay_1_2_val	Out	This control packet value is the valid flag for the delay_1_2 audio processing delay.
delay_3_4[25:0]	Out	This control packet value is the audio processing delay for channels 3 and 4.
delay_3_4_val	Out	This control packet value is the valid flag for the delay_3_4 audio processing delay.
parity_err_c	Out	This is the parity error flag for C _B C _R -stream ancillary packets. Assertion indicates a parity error. This signal is asserted for one vid_ce period for each word that has a parity error.
chksum_err_c	Out	This is the checksum error flag for C _B C _R -stream ancillary packets. A logic High indicates a checksum error. This signal is asserted for one vid_ce period for each ancillary packet that has a checksum error.
parity_err_y	Out	This is the parity error flag for Y ancillary packets. Assertion indicates a parity error. This signal is asserted for one vid_ce period for each word that has a parity error.
chksum_err_y	Out	This is the checksum error flag for Y ancillary packets. A logic High indicates a checksum error. This signal is asserted for one vid_ce period for each ancillary packet that has a checksum error.

The inputs and outputs of the **format96_demux** module are shown in Table 25-4.

Table 25-4: Inputs and Outputs of **format96_demux** Module

Signal	Direction	Connection to hd_audio_demux_top	Description
clk	In	Same as vid_clk	This is a clock that is synchronous with in_samp_valid.
ce	In	Same as vid_ce	This is the clock enable.
format96_en	In	N/A	96 KHz format enable: 1 = Enables 96 KHz formatting 0 = Passes the inputs straight through to the outputs

Table 25-4: Inputs and Outputs of `format96_demux` Module (Cont'd)

Signal	Direction	Connection to <code>hd_audio_demux_top</code>	Description
<code>in_samp_valid</code>	In	<code>audio_valid</code>	This is the audio sample valid input. It is active for one ce period for each sample.
<code>in_samp_1[23:0]</code>	In	<code>ch1_audio</code>	Audio sample stream input 1.
<code>in_samp_2[23:0]</code>	In	<code>ch2_audio</code>	Audio sample stream input 2.
<code>in_samp_3[23:0]</code>	In	<code>ch3_audio</code>	Audio sample stream input 3.
<code>in_samp_4[23:0]</code>	In	<code>ch4_audio</code>	Audio sample stream input 4.
<code>in_flags_1[3:0]</code>	In	<code>ch1_z</code> , <code>ch1_c</code> , <code>ch1_u</code> , <code>ch1_v</code>	These are the input flags for stream 1: bit[3] = Z bit[2] = C bit[1] = U bit[0] = V
<code>in_flags_2[3:0]</code>	In	<code>ch2_c</code> , <code>ch2_u</code> , <code>ch2_v</code>	These are the input flags for stream 2: bit[2] = C bit[1] = U bit[0] = V
<code>in_flags_3[3:0]</code>	In	<code>ch3_z</code> , <code>ch3_c</code> , <code>ch3_u</code> , <code>ch3_v</code>	These are the input flags for stream 3: bit[3] = Z bit[2] = C bit[1] = U bit[0] = V
<code>in_flags_4[3:0]</code>	In	<code>ch4_c</code> , <code>ch4_u</code> , <code>ch4_v</code>	These are the input flags for stream 4: bit[2] = C bit[1] = U bit[0] = V
<code>out_samp_1[23:0]</code>	Out	N/A	Audio sample stream output 1.
<code>out_samp_2[23:0]</code>	Out	N/A	Audio sample stream output 2.
<code>out_samp_3[23:0]</code>	Out	N/A	Audio sample stream output 3.
<code>out_samp_4[23:0]</code>	Out	N/A	Audio sample stream output 4.
<code>out_flags_1[3:0]</code>	Out	N/A	These are the output flags for stream 1: bit[3] = Z bit[2] = C bit[1] = U bit[0] = V

Table 25-4: Inputs and Outputs of `format96_demux` Module (Cont'd)

Signal	Direction	Connection to <code>hd_audio_demux_top</code>	Description
<code>out_flags_2[3:0]</code>	Out	N/A	These are the output flags for stream 2: bit[2] = C bit[1] = U bit[0] = V
<code>out_flags_3[3:0]</code>	Out	N/A	These are the output flags for stream 3: bit[3] = Z bit[2] = C bit[1] = U bit[0] = V
<code>out_flags_4[3:0]</code>	Out	N/A	These are the output flags for stream 4: bit[2] = C bit[1] = U bit[0] = V
<code>out_samp_valid</code>	Out	N/A	This is the output sample valid. It is active for one ce period per sample.

Modules

The reference design contains 11 modules. These are listed in Table 25-5 and are described in more detail in the remainder of this section.

Table 25-5: List of Modules

Module Name	Module Description
<code>hd_audio_demux_top</code>	This is a top-level wrapper for the audio demultiplexer.
<code>hd_audio_demux_data</code>	This module demultiplexes audio data packets from the <code>C_BC_R</code> video stream.
<code>hd_audio_demux_contr</code>	This module demultiplexes audio control packets from the Y video stream.
<code>audio_det</code>	This module detects the presence, type, and location of ancillary data packets.
<code>data_extr</code>	This module extracts information from the audio data and control packets.
<code>demux_tim</code>	This module creates the timing for demultiplexed samples based on the clock phase information.
<code>aud_fifos</code>	This is the wrapper for instances of <code>fifo_28x16</code> that stores incoming audio samples prior to embedding.
<code>fifo_28x16</code>	This is a FIFO element, 28 bits wide by 16 locations deep, built from LUTs.

Table 25-5: List of Modules (Cont'd)

Module Name	Module Description
sync_one_shot	This module detects the rising edge of strobes and outputs a one-clock-wide pulse for each rising edge.
format96_demux	This module converts a pair of parallel audio samples in the 96 KHz format into a sequence of two samples.
split96_ser	This module serializes one pair of 96 KHz audio samples into two sequential samples.

Top-Level Module (hd_audio_demux_top)

This module instantiates and connects the two packet demultiplexer modules for the $C_B C_R$ and Y video streams, `hd_audio_demux_data` and `hd_audio_demux_control`. The `hd_audio_demux_top` module is shown in [Figure 25-5, page 590](#).

Data Packet Demultiplexer (hd_audio_demux_data)

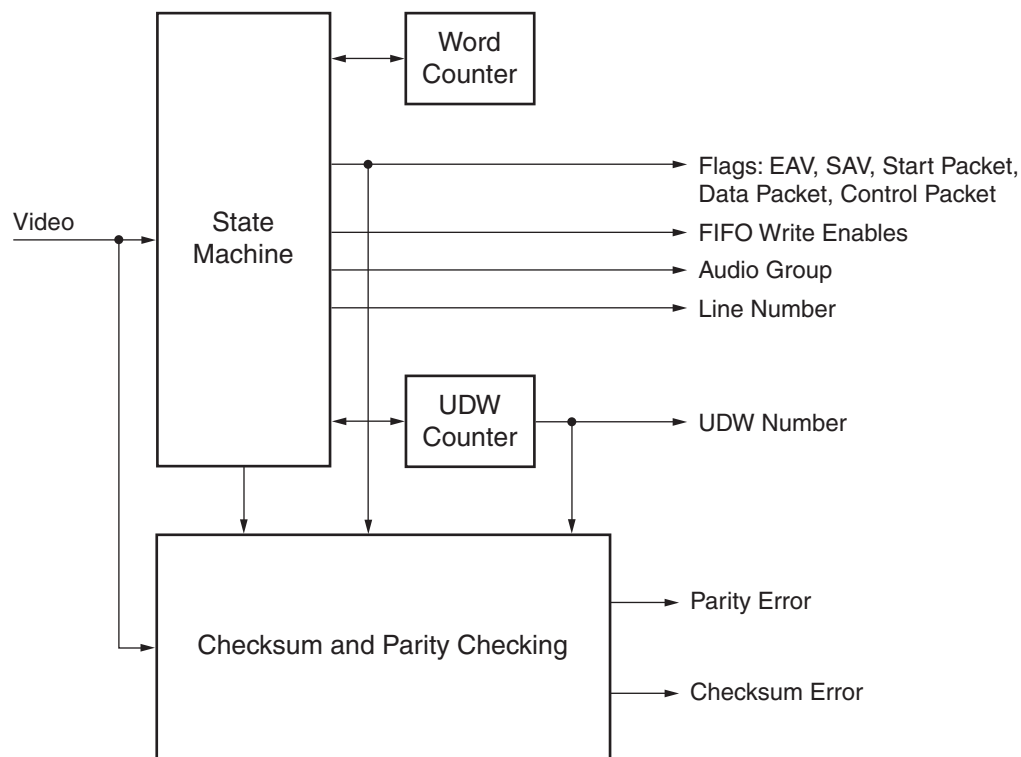
This module is the top level for demultiplexing audio data packets from the $C_B C_R$ video stream. A block diagram of the Data Packet Demultiplexer module is shown in the top section of [Figure 25-5, page 590](#). This module instantiates and connects the lower-level blocks shown.

Control Packet Demultiplexer (hd_audio_demux_contr)

This module is the top level for demultiplexing audio control packets from the Y video stream. A block diagram of the Control Packet Demultiplexer module is shown in the bottom section of [Figure 25-5, page 590](#). This module instantiates and connects the lower-level blocks shown.

Audio Packet Parser (aud_det)

This module monitors the incoming video stream, checking for EAV, SAV, and ancillary data packets. In particular, the module looks for audio data packets and audio control packets to be extracted. It outputs flags indicating the presence of EAV and SAV, the start of an ancillary packet, the type of packet present, the audio group to which it is targeted, and the number of the UDW. A FIFO enable is output for each group. At the top level, the FIFO enable corresponding to the group of interest enables writing of the FIFO. [Figure 25-6](#) is a block diagram of this module. Checking is done for parity errors in audio packets and checksum errors in all ancillary packets.



X1014_C23_06_040209

Figure 25-6: **Block Diagram of `aud_det` Module**

Figure 25-7 shows the state diagram for the state machine. In this figure, the equals sign on conditionals refers to the value of the video data, and `tcnt` refers to assertion of the terminal count of the word counter.

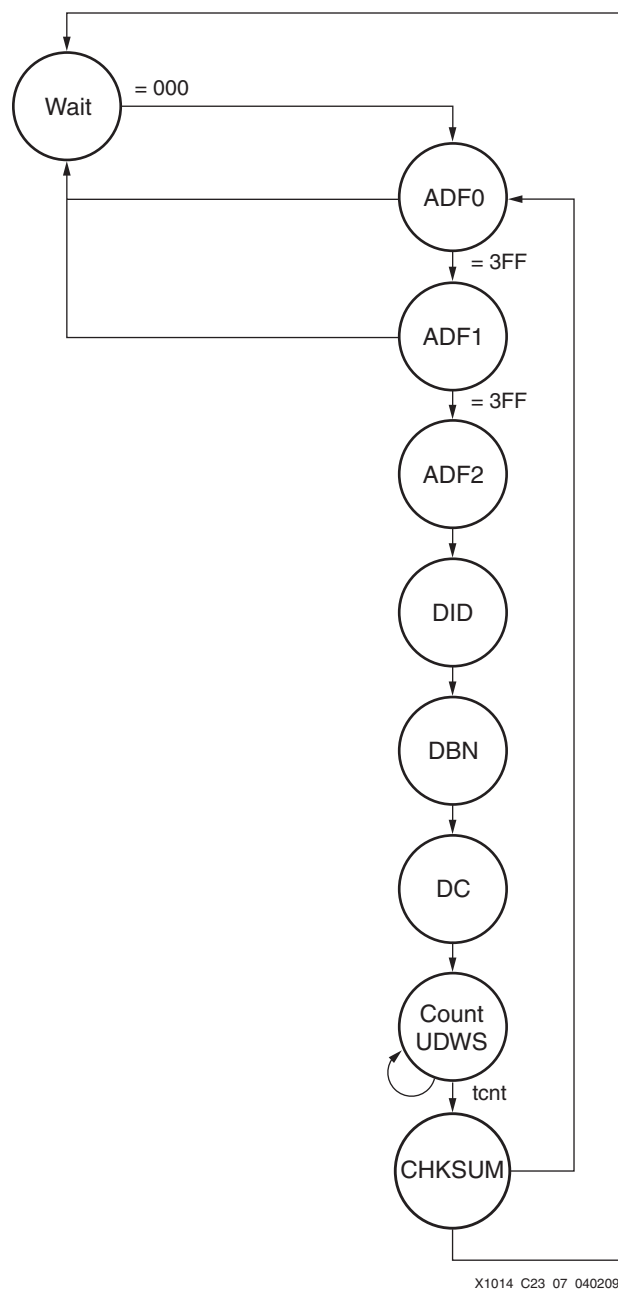
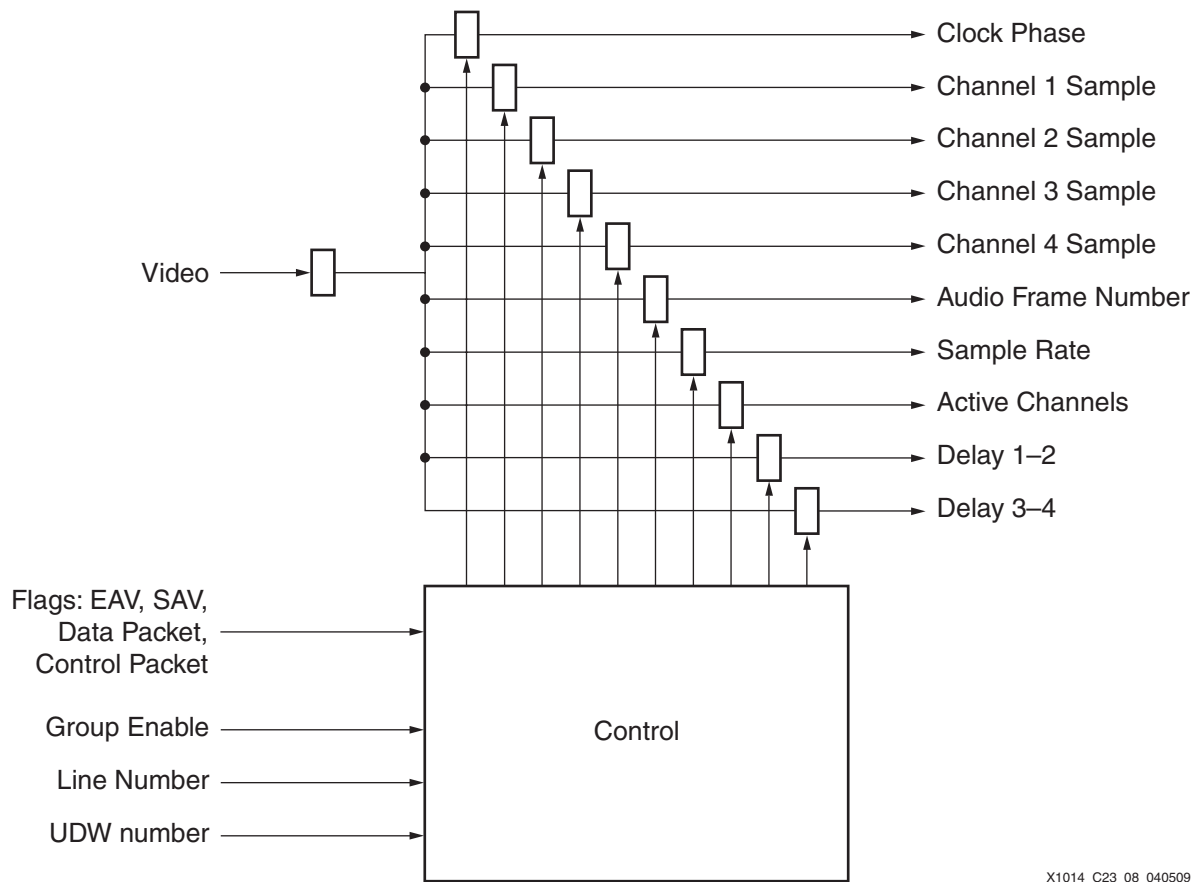


Figure 25-7: Audio Packet Parser State Diagram

Data Extraction (data_extr)

This module extracts the audio data from the video stream byte by byte and puts it in a parallel format for FIFO storage. Information is received from the audio packet parser on the type of packet and which UDW of that packet is present in the video stream in each clock cycle. Based on this, the audio data is extracted from the video stream, formatted, and sent to the FIFO section. Figure 25-8 shows a block diagram of the `data_extr` module.



X1014_C23_08_040509

Figure 25-8: Block Diagram of `data_extr` Module

Line number information from the Audio Packet Parser module is combined with the clock phase information extracted in the Data Extraction module and stored in a FIFO along with the audio sample data. Figure 25-9 shows the formatting of the audio sample data, and Figure 25-10 shows the formatting of the clock phase data from the UDWs. Both types of data are stored to FIFOs at corresponding locations.

Audio control information is extracted from the audio control packets. Rather than being stored in a FIFO, the various fields are stored to registers and are parallel outputs of the `data_extr` module.

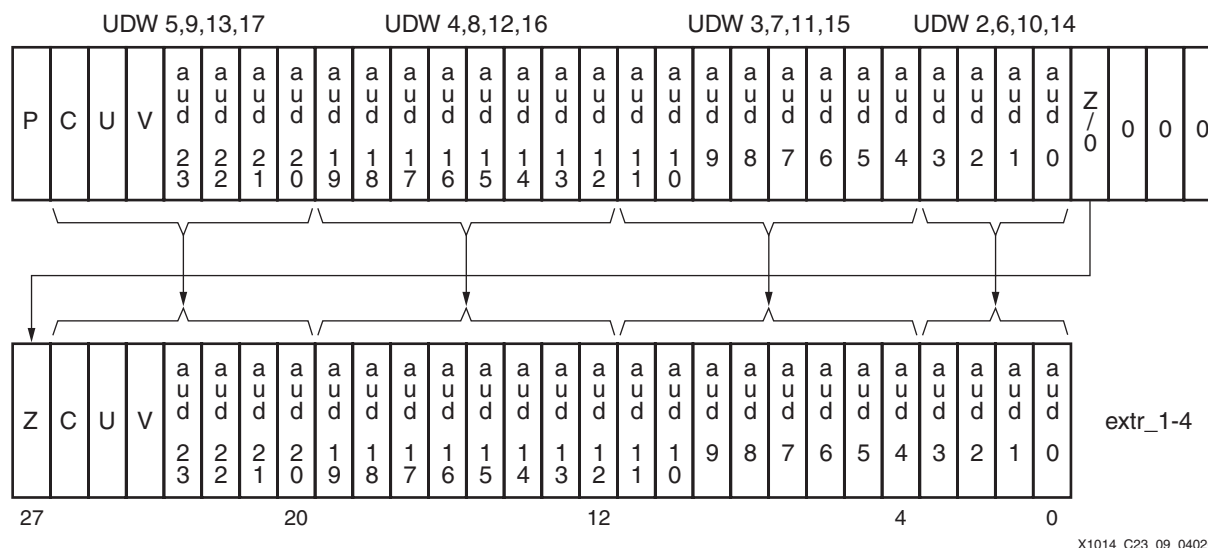


Figure 25-9: Formatting of Audio Sample Data

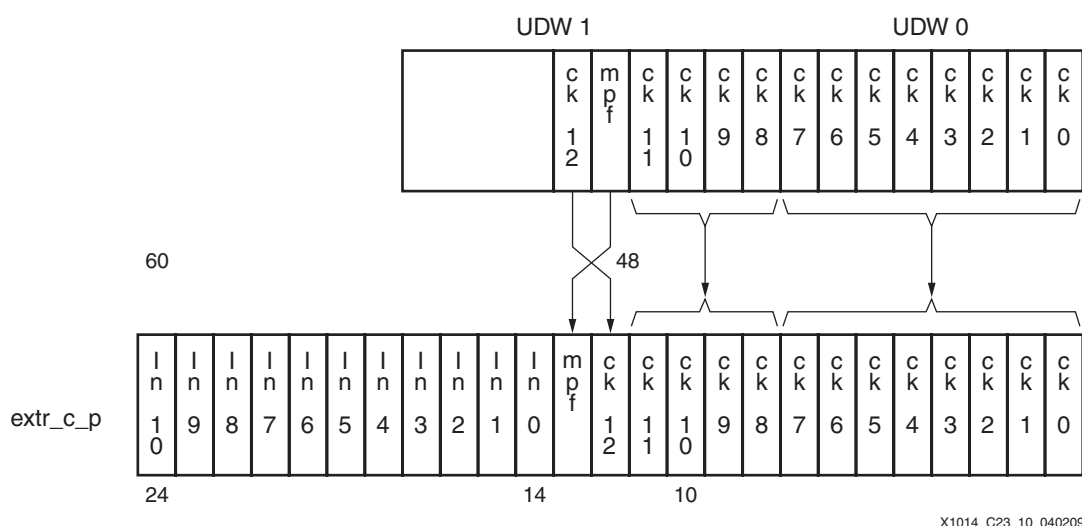


Figure 25-10: Formatting of Clock Phase Data

Audio FIFOs (aud_fifos)

The `aud_fifos` module acts as a small buffer between the demultiplexed sample stream, in which the samples are irregularly spaced in time, and the audio output, where audio samples are output on an evenly spaced basis (see Figure 25-11). The `aud_fifos` module contains five instances of the FIFO module `fifo_28x16`, one for each audio channel of the group, and one instance for the clock phase information. Reads are done in parallel to all FIFOs. Writes are done in parallel to `aud_fifos`, but the clock phase data is written separately.

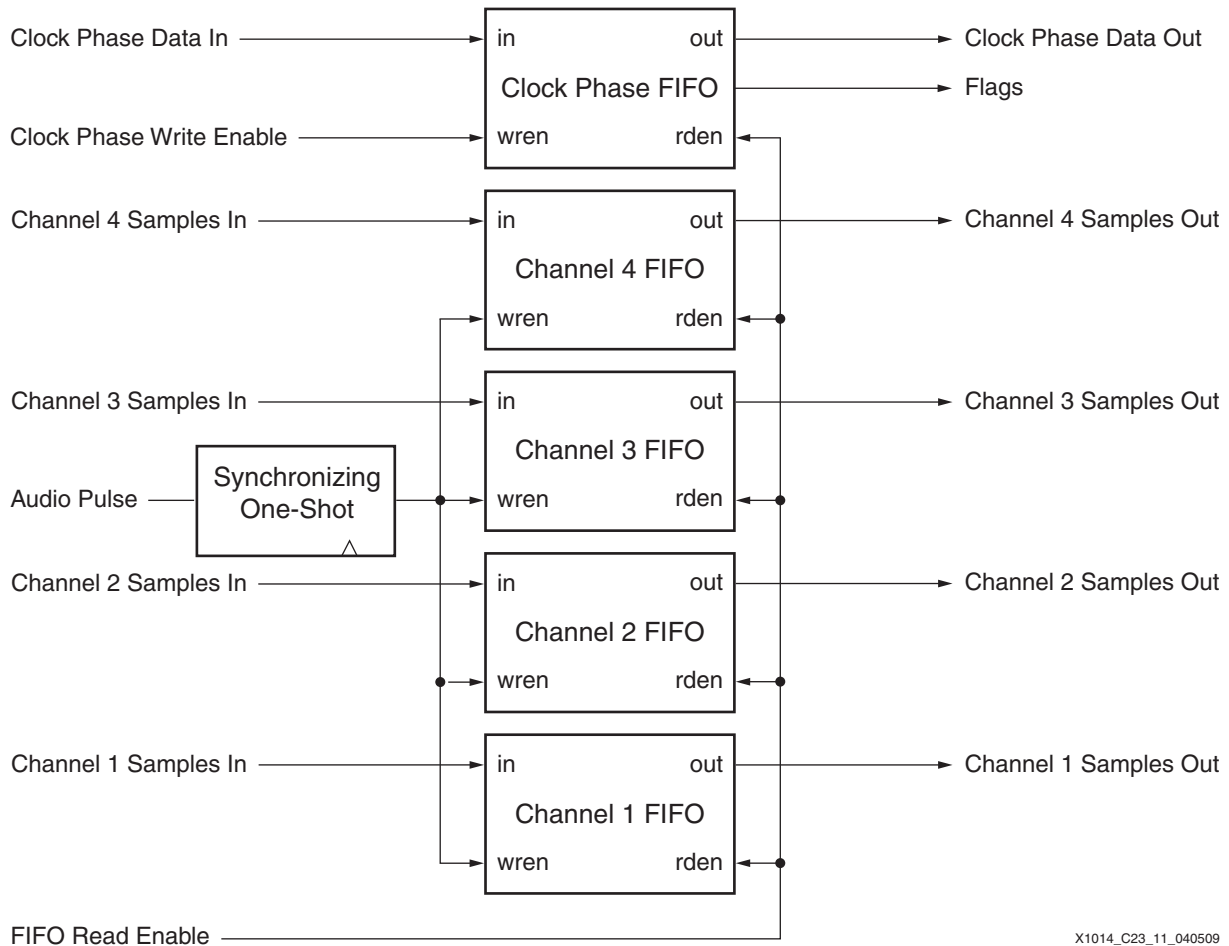
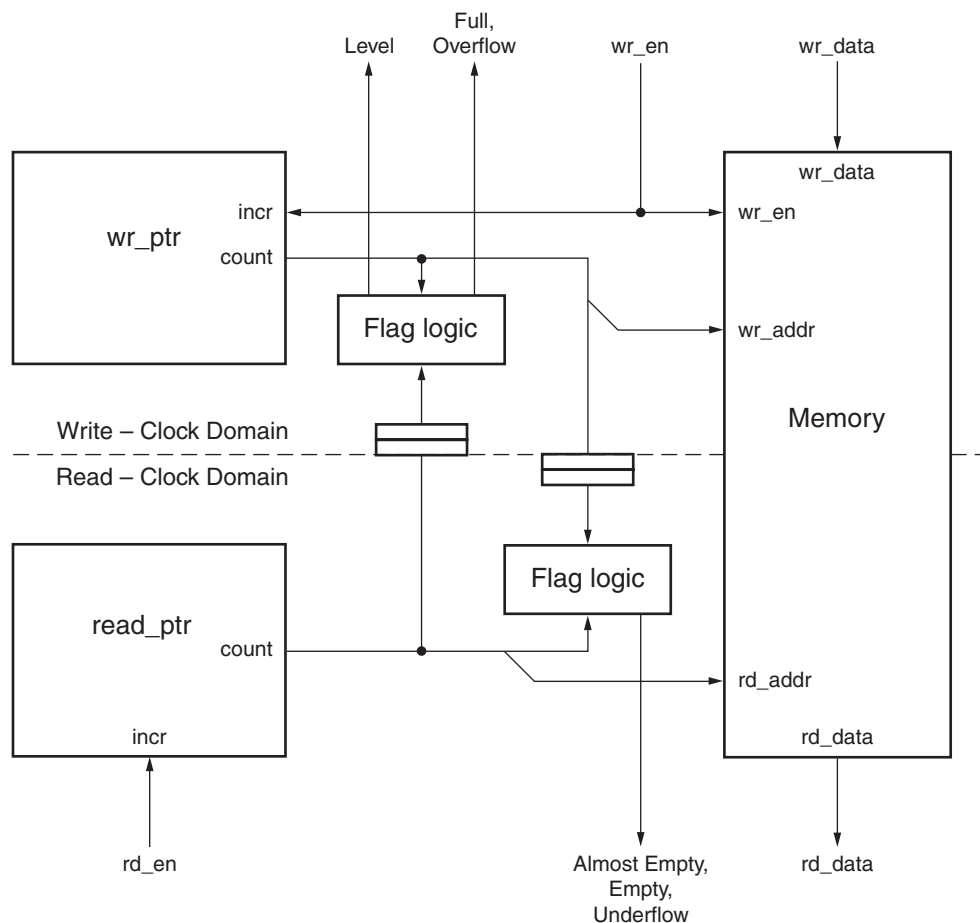


Figure 25-11: Block Diagram of `aud_fifos` Module

FIFO (`fifo_28x16`)

This is the basic FIFO module consisting of SelectRAM™ memory built from LUTs (see [Figure 25-12](#)). It is 28 bits wide by 16 locations deep. Each FIFO stores either video samples or clock phase information. In video sample FIFOs, each location contains one 24-bit video sample and the associated Z, C, U, and V flags. For the clock phase FIFO, each location contains the clock phase information associated with the corresponding audio samples.

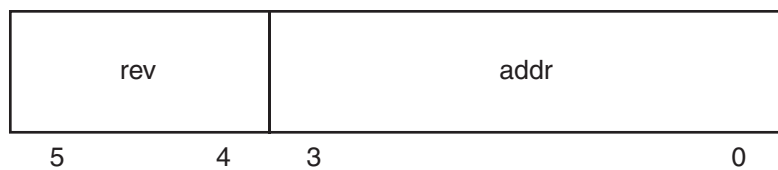


X1014_C23_12_040209

Figure 25-12: Block Diagram of FIFO

The control logic is built specifically for this reference design. In particular, it protects against underflow by not allowing the read pointer to increment when the FIFO is empty. At start-up, it is likely for read requests to occur when the FIFO is empty. Due to inherent latency in flag logic crossing clock domains, these read requests can happen before the empty flag has propagated to the controlling logic for the read operation. In such cases, the read operation, specifically the moving of the read pointer, is inhibited. This is important because there are occasions in which the FIFO must be emptied fully, but must also be guaranteed to work properly without rolling over when the next valid write data arrives.

As shown in Figure 25-13, the read and write pointers have two extra MSBs beyond the address. These indicate the number of passes or revolutions through the memory address space. When the read and write pointers are pointing to the same address, it can be unambiguously determined whether this represents a full condition (write revolution does not equal read revolution) or an empty condition (write revolution equals read revolution).

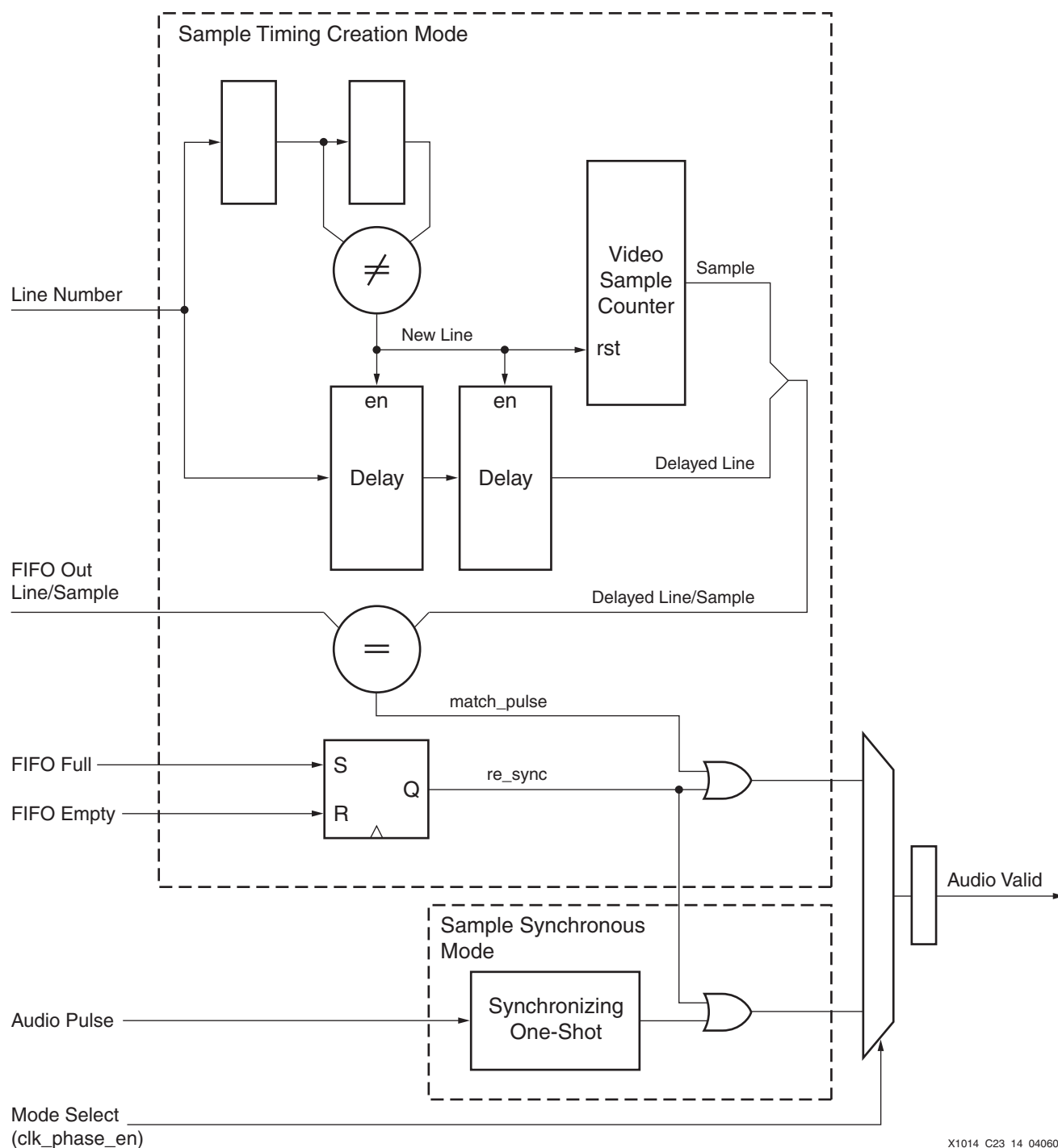


X1014_C23_13_040209

Figure 25-13: FIFO Pointer Format

Timing Control (demux_tim)

The `demux_tim` module controls the timing of reads from the audio sample FIFO (see [Figure 25-14](#)). It has two modes of operation, referred to in this chapter as sample timing creation mode and sample synchronous mode.



X1014_C23_14_040609

Figure 25-14: Block Diagram of `demux_tim` Module

In sample timing creation mode, the line number and the clock phase of the sample on the FIFO output are compared with the line and sample information from the video data to determine when the next sample should be read. The line number from the current scan is delayed by several lines to ensure that the audio samples are available in the FIFO before the time at which they are read.

The `demux_tim` module produces an audio valid pulse when a timing match occurs. The video clock and clock enable are used in this module and thus, the timing is accurate only to a video clock level. This module requires some type of sample rate smoothing, sample rate conversion, or both, if it is to be used to drive an AES output. The sample timing creation mode supports asynchronous audio.

Sample synchronous mode, on the other hand, requires that the embedded audio be synchronous and that an audio-rate pulse be provided that is synchronized to the video. This pulse is used instead of `match_pulse` to read the FIFO.

In either mode, an automatic resynchronization operation is used if the FIFO becomes full to ensure that the FIFO maintains valid data. At start-up, and whenever the FIFO becomes full, the `re_sync` signal is set and the `audio_valid` pulse is asserted until the FIFO is empty. At that point, the audio valid strobe and reading of the FIFO are disabled until the FIFO fills up to a certain level. In sample synchronous mode, reading of the FIFO resumes until the next external audio pulse. This might result in attempted reads of an empty FIFO, but in that case, the read pointer is not advanced. In either mode, the result is a steady flow of data after a few video frames from start-up. In sample timing creation mode, reading resumes when the delayed line and sample match the line or sample value at the FIFO output. The FIFO must maintain a certain minimum fill level to compensate for the burst nature of the writes from the demultiplex process.

Synchronizing One-Shot (`sync_one_shot`)

This is a small circuit that detects a pulse, synchronizes it to another clock, identifies the rising edge, and outputs a one-cycle-wide pulse in the new clock domain. [Figure 25-15](#) shows the details of the synchronizing one-shot circuit. This circuit can accommodate an extremely wide range of pulse widths on the input. The minimum width of the pulse is just long enough to be recognized as a High level by the SR latch. There is no maximum width; however, the input must remain Low for at least four clock cycles for the next rising edge to be recognized as a distinct pulse.

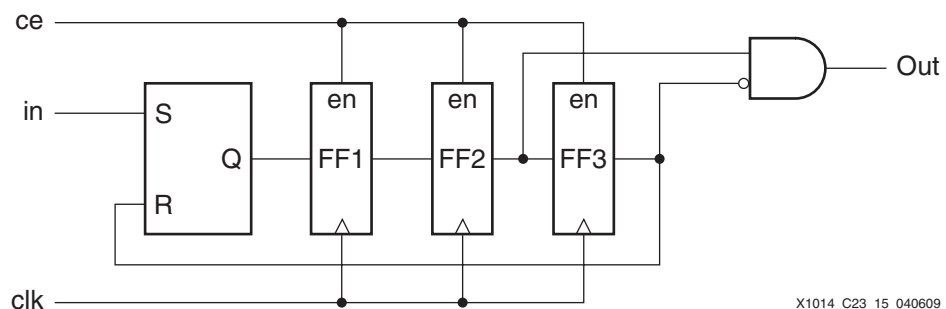
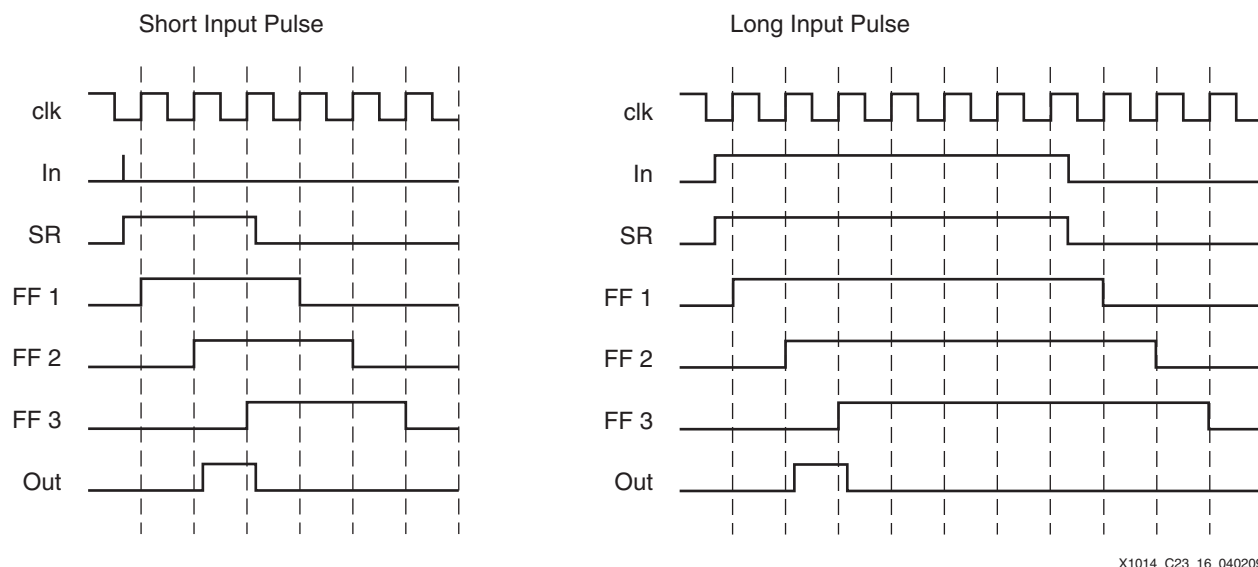


Figure 25-15: Synchronizing One-Shot Circuit

[Figure 25-16](#) illustrates the operation of the synchronizing one-shot circuit for both short and long input pulses. Regardless of the length of the input pulse, the output is a one-clock-wide pulse, synchronized to the clock and occurring near the rising edge of the input pulse.



X1014_C23_16_040209

Figure 25-16: Synchronizing One-Shot Timing (ce = 1)

Format 96 Demultiplexer (format96_demux)

The `format96_demux` module consists of a multiplexer and two instances of the `split96_ser` module. A block diagram of this module is shown in [Figure 25-17](#). When 96 KHz mode is enabled via `split96_en`, the four channel inputs are time multiplexed through the `split96_ser` modules and output on channels 1 and 2. When 96 KHz mode is disabled, the four channel inputs are passed straight through to the outputs.

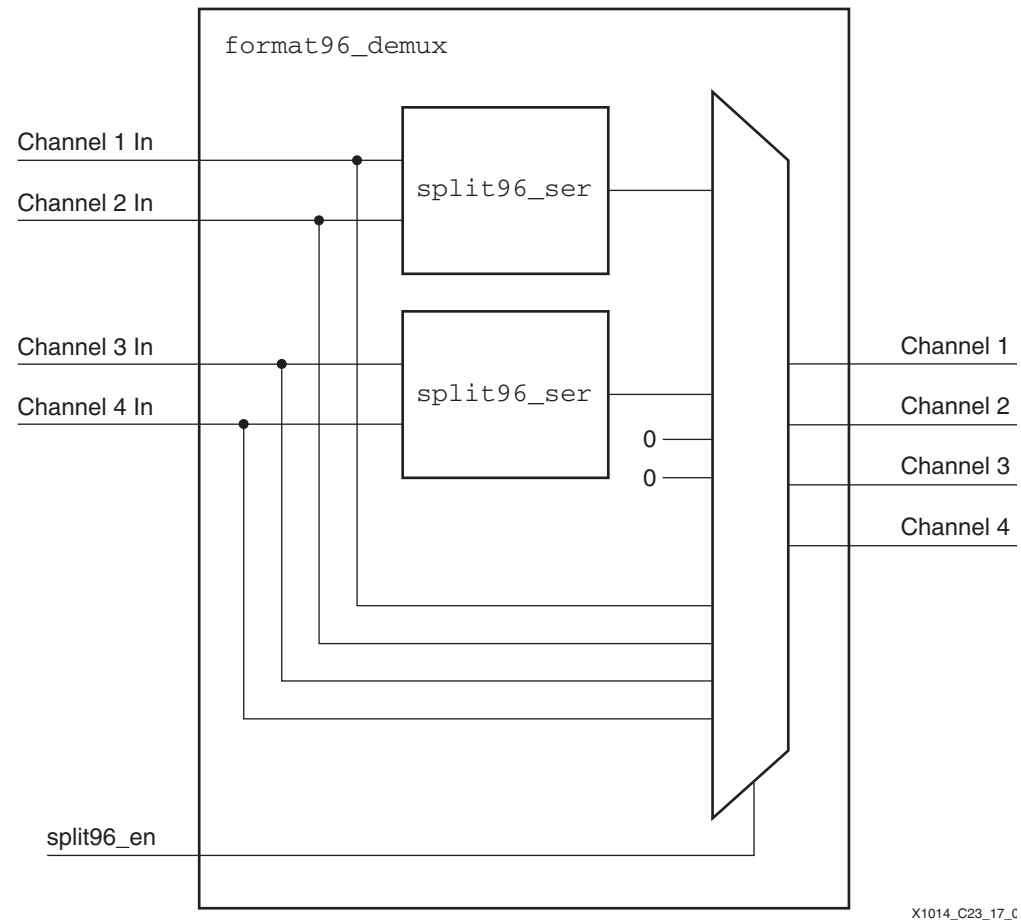
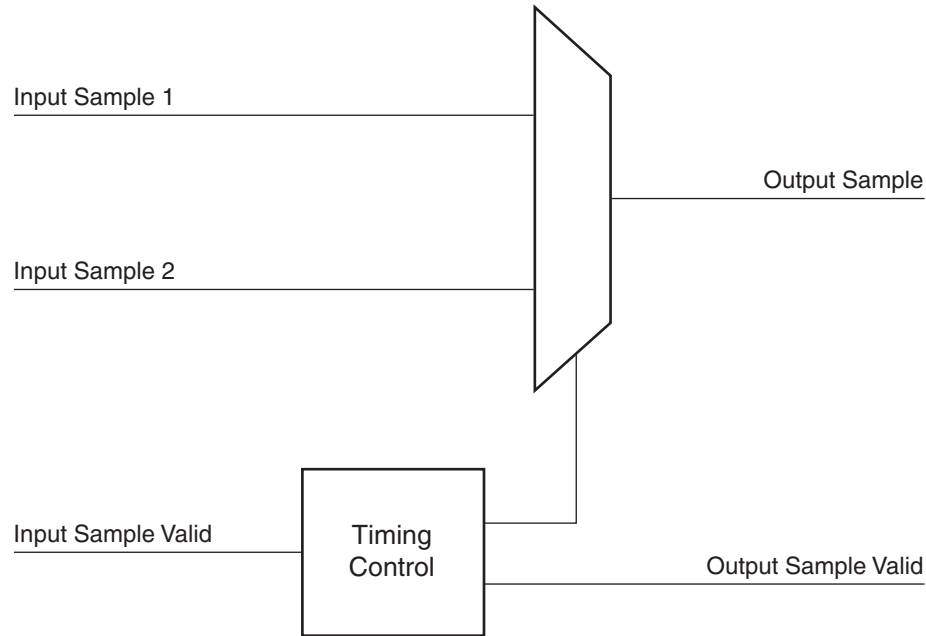


Figure 25-17: Block Diagram of `format96_demux` Module

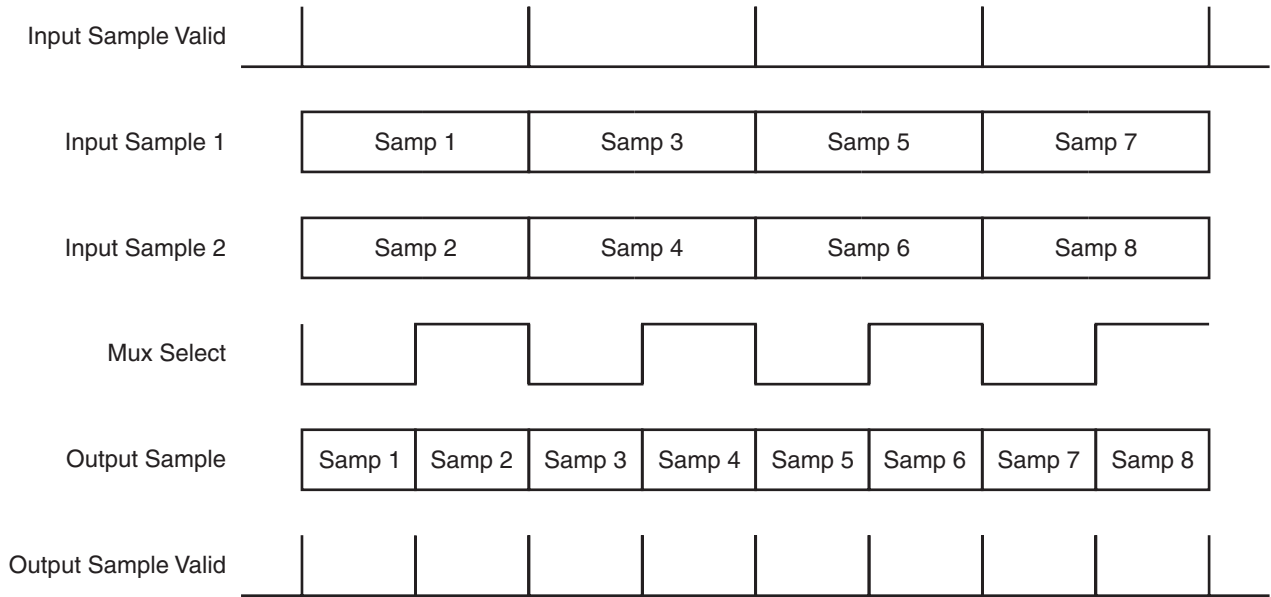
Split 96 Serializer (`split96_ser`)

The `split96_ser` module serializes two parallel samples in the 96 KHz packet format into a sequence of two sequential samples with a corresponding double-rate valid signal. A block diagram of this module is shown in Figure 25-18. The timing for the `split96_ser` module is shown in Figure 25-19. The timing control block uses a high-speed clock to count the period of the input sample valid signal, thereby doubling the frequency to control the output multiplexer and produce the output sample valid pulse.



X1014_C23_18_040209

Figure 25-18: Block Diagram of `split96_ser` Module



X1014_C23_19_040209

Figure 25-19: Timing Diagram of `split96_ser` Module

Usage Models

This section provides some examples of ways in which the audio demultiplexer reference design can be used. In these examples, the video used can be either 3G-SDI level A or HD-SDI. For these video modes, up to 16 channels (8 channels for 96 KHz) can be accommodated by using four instances of the top-level design, as shown in Figure 25-20. Each instance is configurable to operate on any group via the group_sel input port.

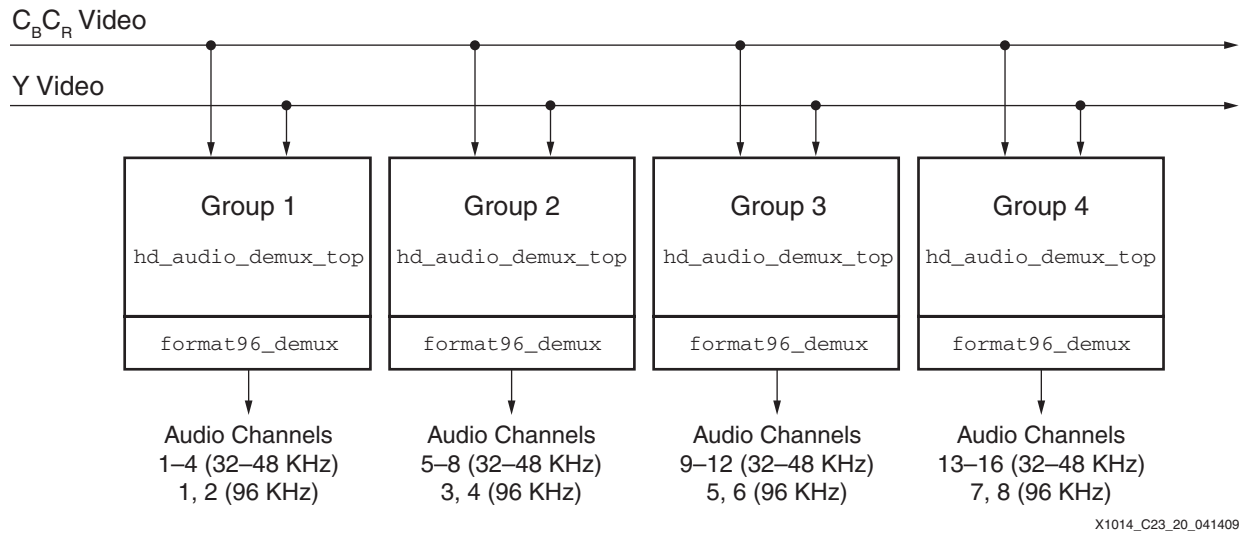


Figure 25-20: Demultiplexing of Multiple Audio Groups Using Multiple Instances of the Reference Design

To support dual link HD-SDI and 3G-SDI level B, the audio demultiplexer is connected to link A of the two internal HD-SDI links and operates in HD-SDI mode. If more than 16 channels of 32 KHz to 48 KHz (8 channels of 96 KHz) are present, additional instances should be added to link B, as shown in Figure 25-21. The SMPTE standard for dual link HD-SDI and 3G-SDI level B specify that link B only be used for audio if all available channels on link A have been utilized. For example, if there are 20 audio channels in 32 KHz to 48 KHz mode, 16 channels are mapped to link A and 4 channels are mapped to link B.

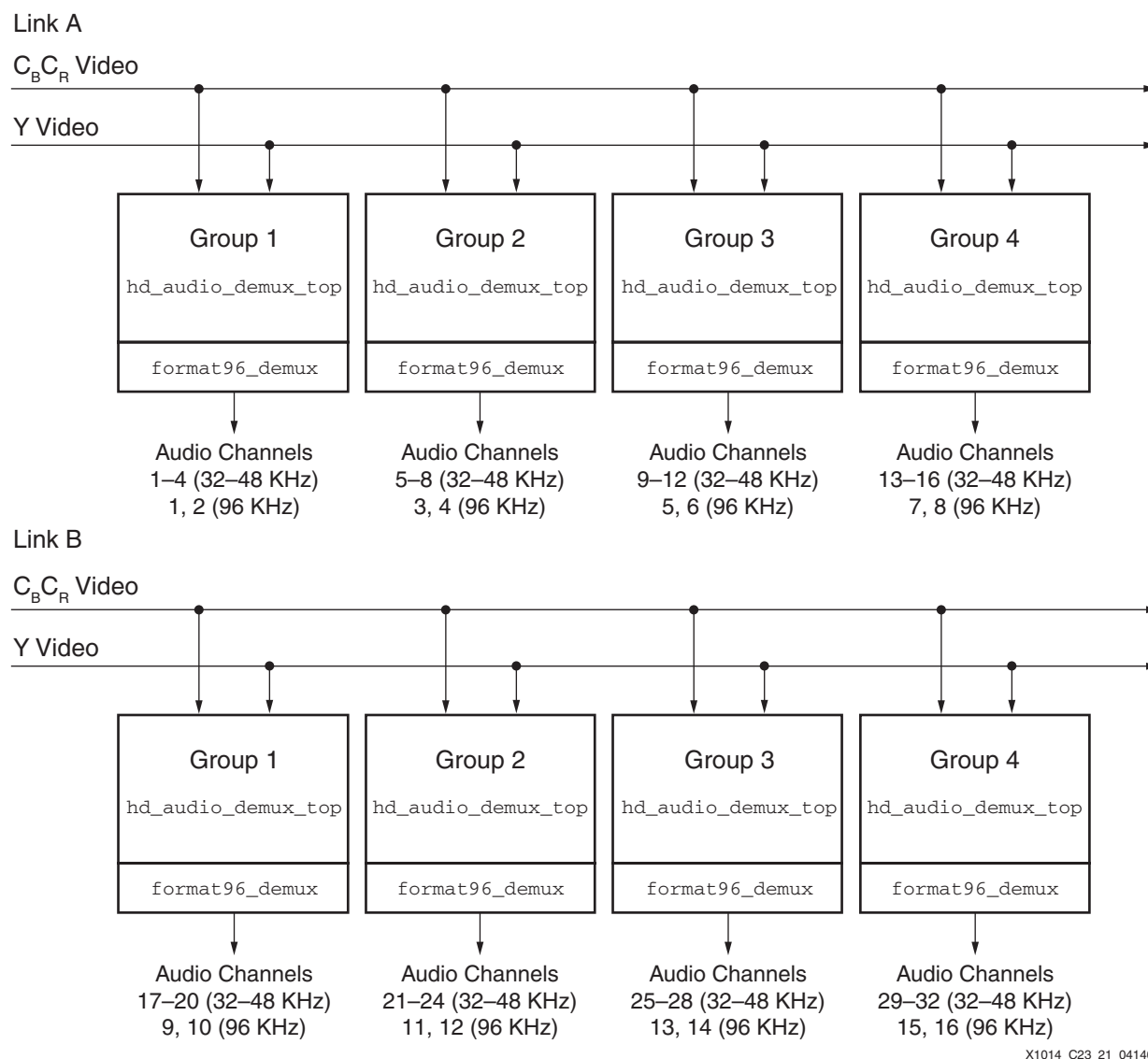
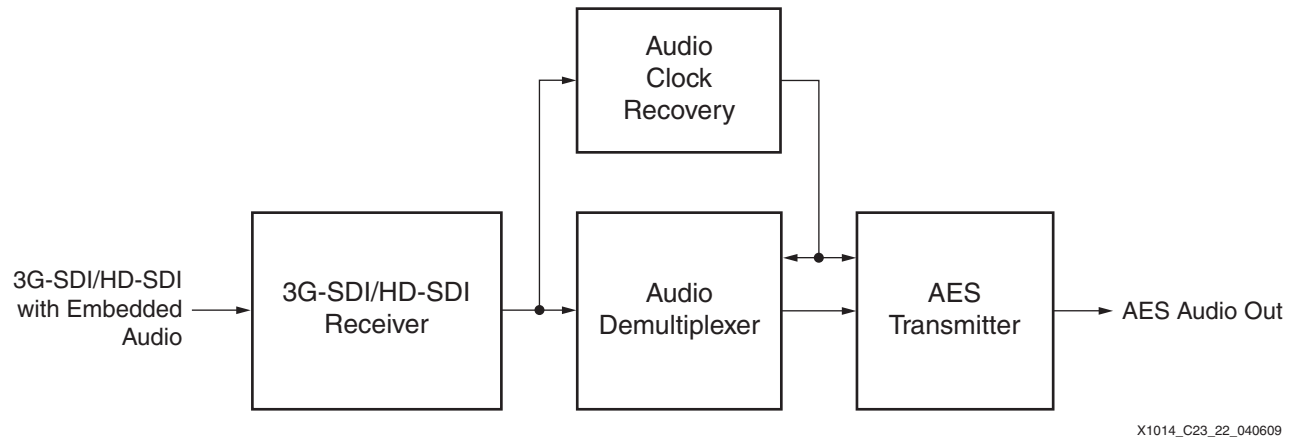


Figure 25-21: Additional Audio Channels on Dual Link HD-SDI and 3G-SDI Level B

Synchronous De-embedding

For synchronous embedded audio, a synchronous, recovered audio clock can be used to directly time the output of de-embedded audio samples and an AES transmitter, as shown in Figure 25-22.

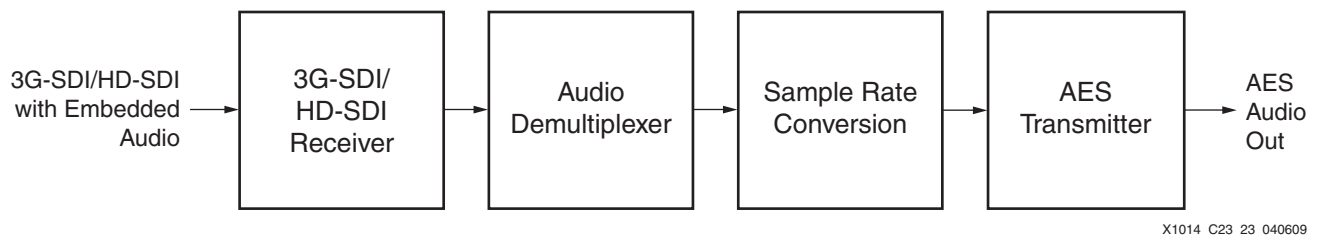


X1014_C23_22_040609

Figure 25-22: **Synchronous De-embedding**

Asynchronous De-embedding

An AES transmitter with arbitrary timing transmits de-embedded audio when a sample rate converter is used to convert the de-embedded audio to a new timing base used by the AES transmitter. This asynchronous de-embedding is illustrated in Figure 25-23.

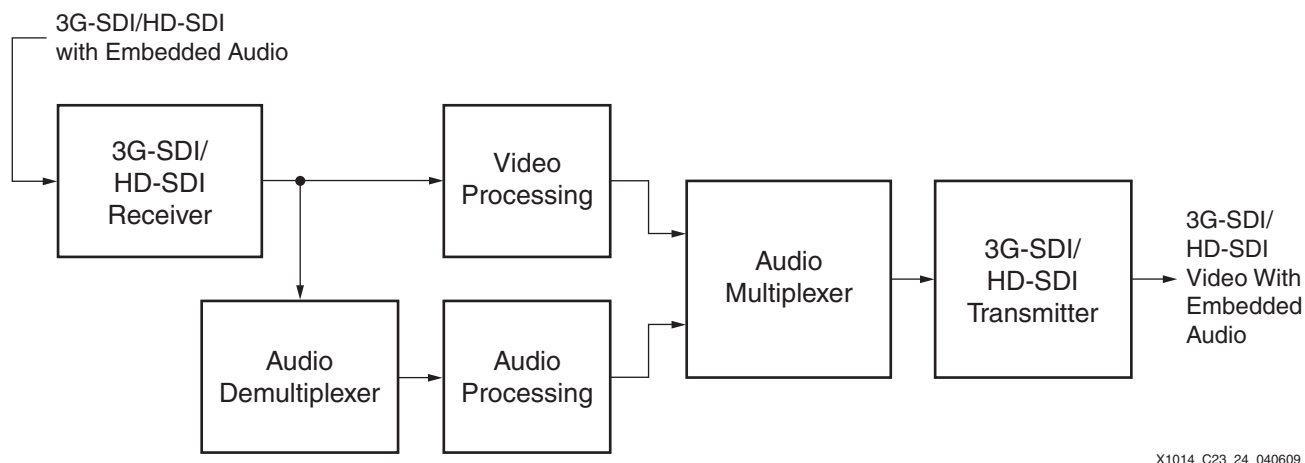


X1014_C23_23_040609

Figure 25-23: **Asynchronous De-embedding**

De-embedding Audio for Separate Audio/Video Processing

In some applications, audio and video should be processed separately from one another, as shown in Figure 25-24. For example, a video frame synchronizer drops complete frames of video. However, the audio cannot tolerate the equivalent dropping of samples, and so must be processed separately. In such cases, the audio is de-embedded from the video by the audio demultiplexer and processed in a separate path. After processing is completed, the audio and video are combined again in an audio multiplexer.



X1014_C23_24_040609

Figure 25-24: Separate Audio and Video Processing

Performance and Resources

The reference design is implemented and hardware verified in a Virtex-5 device, but it can also be used with other Xilinx FPGA families. The resources required for the reference design in a Virtex-5 device are shown in Table 25-6. The formatter for 96 KHz samples is independent of the rest of the audio demultiplexer so that it can be included only where needed.

The results shown in Table 25-6 were obtained using ISE® software, version 9.2i SP2. Timing was constrained for a processing clock of 297 MHz. Timing was met for -1, -2, and -3 speed grade devices.

Table 25-6: FPGA Resources

Top-Level Module	Flip-Flops	LUTs	Block RAM	DSP Blocks
hd_audio_demux_top	676	685	0	0
format96_demux	51	157	0	0

Design Files

The reference design for the HD audio demultiplexer is available at <https://secure.xilinx.com/webreg/clickthrough.do?cid=119758>. Open the ZIP archive and extract the file xapp1014_c25_3G_audio_demux.zip.

Conclusion

This chapter explains embedded audio in HD-SDI, 3G-SDI, and dual link HD-SDI, the standards used to define it, and the data formats of the audio packets. The reference design of this chapter is useful for de-embedding audio from 3G-SDI and HD-SDI video. It can be used in a variety of applications and can be implemented in all Xilinx FPGA families. The reference design supports multiple video standards and multiple audio sample rates, including 96 KHz, in both synchronous and asynchronous formats. The modularity of the design supports from one to four audio groups by placing multiple instances in parallel. Metadata contained in audio control packets is also decoded and output from the reference design.

Section VI: Miscellaneous Audio and Video Topics

***Audio/Video Connectivity Solutions
for Virtex-5 FPGAs***

SMPTE 352M Video Payload Identification Packet Processing

Summary

The SMPTE 352M standard defines an ancillary packet type that provides specific information about the video payload carried by a digital interface. These VPID packets are placed in the HANC space of specific video lines and carry information such as the interface type, sampling structure, component bit depth, and picture update rate. These packets are compatible with virtually all SMPTE digital video interfaces, serial or parallel, and all forms of uncompressed digital video, including both HD and SD, that are compatible with the SMPTE digital interfaces.

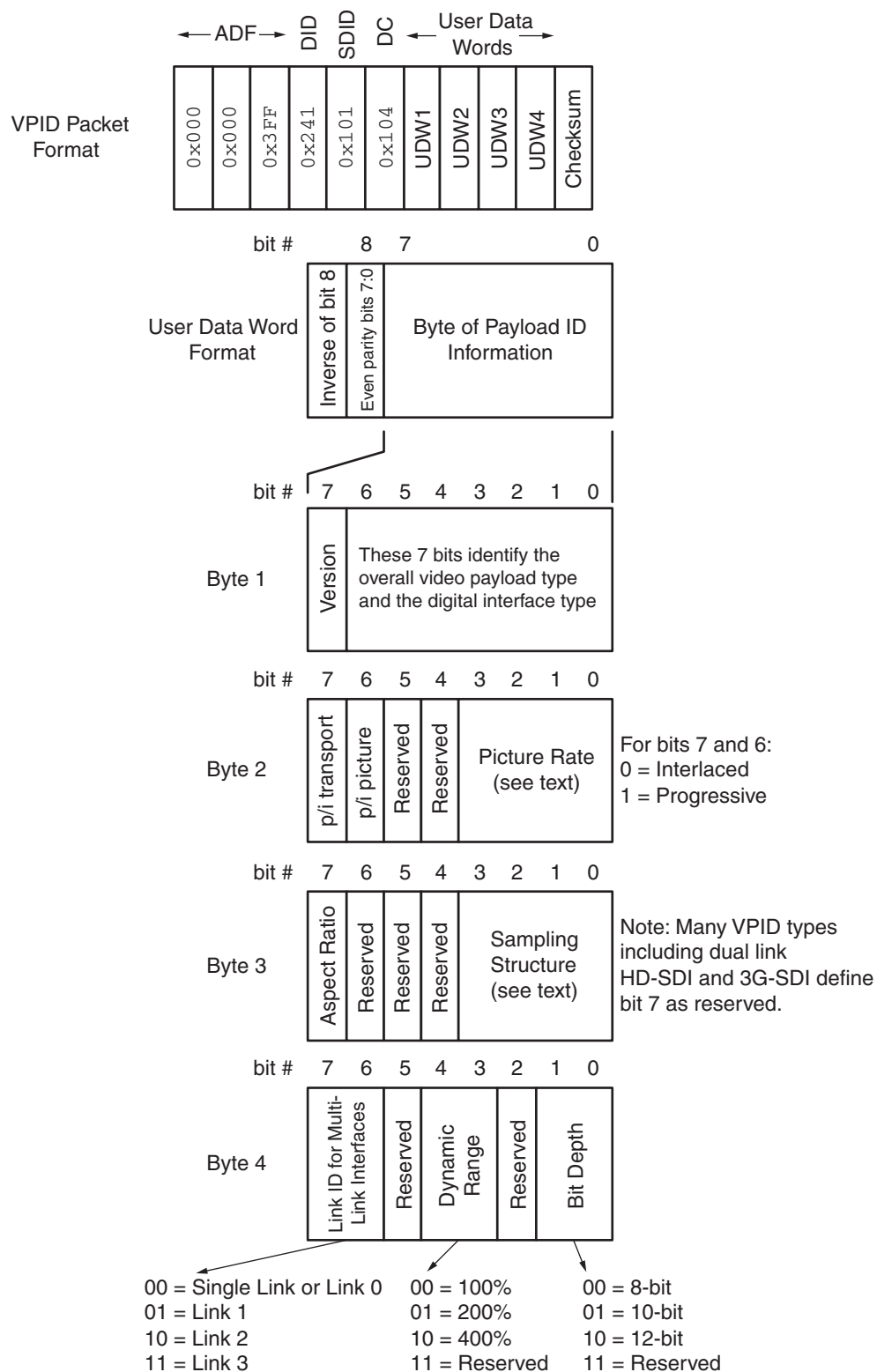
The VPID packets are highly recommended or even required for recent SMPTE interfaces such as dual link HD-SDI and 3G-SDI, because it is very difficult, if not impossible, to properly interpret the video data without the information provided by the VPID packets. Older interfaces such as HD-SDI and SD-SDI can also use the VPID packets, although the use of VPID packets with these older interfaces is not yet widespread.

The reference designs described in this chapter allow VPID packets to be captured from and inserted into video streams. They can be implemented in most Xilinx® FPGAs including all varieties of Spartan®-3, Virtex®-II Pro, and Virtex-5 FPGAs. The modules are designed to be integrated with the HD-SDI, SD-SDI, dual link HD-SDI, and 3G-SDI reference designs.

SMPTE 352M Description

Previously, it was possible to uniquely identify the video payload carried on an interface by inspecting the video timing. However, with the proliferation of digital video formats and interfaces, it is no longer always possible to uniquely determine essential characteristics of the video payload simply by its timing. For example, it is difficult or impossible to distinguish between two video payloads that differ only by the color space used. If SMPTE 352M VPID packets are present in the data stream, the important characteristics of the video payload can be easily determined.

VPID packets are compliant with the Type 2 ANC packet format defined in SMPTE 291M. The VPID packet format is shown in [Figure 26-1](#).



X1014_C24_01_040909

Figure 26-1: VPID Packet Format

VPID packets always have four user data words (UDWs). Each UDW provides 8 bits of data for a total of 32 bits of payload identification data organized into four bytes. These four bytes of data identify such characteristics of the video stream as:

- Interface type: single link HD-SDI, dual link HD-SDI, 3G-SDI, SD-SDI, etc.
- Interlaced or progressive picture and transport
- Picture rate
- Sampling structure, including whether the alpha component is active
- Component bit depth
- Link identification for multi-link interfaces

SMPTE 352M defines the basic structure of the VPID packet, including default definitions for many of the bits in the packet. However, many of the bits in the packet are application specific, meaning that their definition can be different on different interfaces. Therefore, the interpretation of the contents of the packet depends on the value of the first byte of the packet.

The default format of the VPID packet is shown in [Figure 26-1](#). The appropriate SMPTE interface standards should always be consulted when determining the format of a VPID packet for a particular digital interface and video payload.

Byte 1

The first byte of user data in the VPID packet identifies the interface type and also dictates the format of the information in the other three data bytes.

Bit 7, the most significant bit of this byte distinguishes between revisions of the VPID packet standard. The trial version of the specification was implemented by some equipment manufacturers and is identified by a 0 in bit 7. The official version of VPID, currently defined by SMPTE 352M, defines the format of the data bytes differently than the trial version and is identified by a 1 in bit 7. While the reference design is fully compatible with the trial version of the VPID specification, this chapter focuses only on the released version of the specification. The currently defined byte 1 values include those shown in [Table 26-1](#).

Table 26-1: Byte 1 Values

Byte 1	Video Payload	Interface
0x81	525i or 625i	SMPTE 259M SD-SDI (270 Mb/s or 360 Mb/s)
0x82	525p or 625p	SMPTE 259M SD-SDI (dual link 270 Mb/s or single link 360 Mb/s)
0x83	525 or 625 line	SMPTE 344M SD-SDI (540 Mb/s)
0x84	SMPTE 296M 720p	SMPTE 292M HD-SDI
0x85	SMPTE 274M 1080i and 1080p	SMPTE 292M HD-SDI
0x86	525 or 625 line	SMPTE 292M HD-SDI using SMPTE 349M mapping
0x87	SMPTE 274M 1080i and 1080p	SMPTE 372M dual link HD-SDI
0x88	SMPTE 296M 720p	SMPTE 425M-A 3G-SDI
0x89	SMPTE 274M 1080i and 1080p	SMPTE 425M-A 3G-SDI

Table 26-1: Byte 1 Values (Cont'd)

Byte 1	Video Payload	Interface
0x8A	SMPTE 372M dual link	SMPTE 425M-B 3G-SDI
0x8B	2 x SMPTE 296M	SMPTE 425M-B 3G-SDI
0x8C	2 x SMPTE 274M	SMPTE 425M-B 3G-SDI
0x8D	2 x 525 or 625 line	SMPTE 425M-B 3G-SDI

Byte 2

Bits 7 and 6 in byte 2 are used to identify whether the transport and the picture are interlaced or progressive, with separate bits for each. It is possible to carry progressive video on an interlaced transport; progressive segmented frame video is an example of this. Thus, these two bits do not always match. Table 26-2 shows the encoding for the picture rate field in byte 2. All values not shown are reserved.

Table 26-2: Picture Rate Values for Byte 2

Picture Rate (Hz)	Value	Picture Rate (Hz)	Value
24/1.001	0x2	24	0x3
25	0x5	50	0x9
30/1.001	0x6	30	0x7
60/1.001	0xA	60	0xB

SMPTE 352M does not define exactly what picture rate means for interlaced video. For example, it is not clear whether the picture rate of PAL is 25 Hz (the frame rate) or 50 Hz (the field rate). After polling members of the SMPTE technical committees, it has been determined that the picture rate field of byte 2 of SMPTE 352M packets should indicate the frame rate for interlaced video. Thus, for PAL, the frame rate should be set to 25 Hz. This is verified by section A.1 of SMPTE 352M, which indicates that 0x5 (25 Hz) is the correct value to use for PAL. Some equipment on the market does not follow this convention and incorrectly sets the picture rate equal to the field rate for interlaced video.

Byte 3

Bit 7 of byte 3 indicates the aspect ratio of the image (0 = 4:3 and 1 = 16:9). However, this bit is reserved and must be set to 0 for many interfaces including dual link HD-SDI and 3G-SDI, regardless of the aspect ratio. Table 26-3 shows the encoding for the sampling structure field in byte 3.

Table 26-3: Picture Rate Values for Byte 3

Structure	Value	Structure	Value	Structure	Value	Structure	Value
4:2:2 (Y'C _B 'C _R)	0x0	4:4:4 (Y'C _B 'C _R)	0x1	4:4:4 (R'G'B')	0x2	4:2:0	0x3
4:2:2:4 (Y'C _B 'C _R 'A)	0x4	4:4:4:4 (Y'C _B 'C _R 'A)	0x5	4:4:4 (R'G'B'A)	0x6	Reserved	0x7

Table 26-3: Picture Rate Values for Byte 3 (Cont'd)

Structure	Value	Structure	Value	Structure	Value	Structure	Value
4:2:2:4 (Y'C _B 'C _R 'D)	0x8	4:4:4:4 (Y'C _B 'C _R 'D)	0x9	4:4:4:4 (R'G'B'D')	0xA	Reserved	0xB
Reserved	0xC	Reserved	0xD	4:4:4 (X'Y'Z')	0xE	Reserved	0xF

Byte 4

Byte 4 has several multiple-bit fields as shown in Figure 26-1. The link ID bits are used by multiple-link interfaces such as SMPTE 372M to uniquely identify each link. These link ID bits are also used when dual link HD-SDI is mapped onto a single 3G-SDI interface using level B of SMPTE 425M to identify which of the links from the dual link interface is mapped to each of the two data streams of the 3G-SDI interface.

The use of the dynamic range field is not well defined in the current SMPTE standards. SMPTE technical committee members recommend setting the dynamic range to 100% for 8-bit video components, 200% for 10-bit video components, and 400% for 12-bit video.

VPID Packet Location

VPID packets must be located on specific video lines, as shown in Table 26-4, at the beginning of the HANC space, right after the EAV or CRC words that follow the EAV on those interfaces that use CRC words. For SMPTE 292M HD-SDI interfaces, the VPID packets are placed only in the Y data stream. For SMPTE 372M dual link HD-SDI interfaces, the VPID packets are placed in the Y data streams of both links. For SMPTE 425M 3G-SDI, the VPID packets are placed in both data streams.

Table 26-4: VPID Packet Locations

Video Format	Field	Line Number
525i	1	13
	2	276
625i	1	9
	2	322
1080i	1	10
	2	572
525p	-	13
625p	-	9
720p	-	10
1080p	-	10

Reference Design

The reference design files for this chapter can be downloaded at <https://secure.xilinx.com/webreg/clickthrough.do?cid=119758>. Open the ZIP archive and extract the file xapp1014_c26_SMPTE352.zip.

Two primary modules are supplied by this reference design. SMPTE352_vpid_capture is used to detect VPID packets and capture the payload ID bytes from the packet. SMPTE352_vpid_insert is used to insert VPID packets into data streams.

SMPTE352_vpid_capture

The SMPTE352_vpid_capture module watches one 10-bit data stream from an interface and captures the VPID packets present in that data stream. For dual link HD-SDI interfaces, two of these modules are required to look for VPID packets on the Y data streams of both links. For 3G-SDI interfaces, two of these modules can be used, one for each of the two data streams of the 3G-SDI interface because the 3G-SDI specification calls for VPID packets in both data streams. Figure 26-2 through Figure 26-5 provide examples of how this module is used with various interface standards.

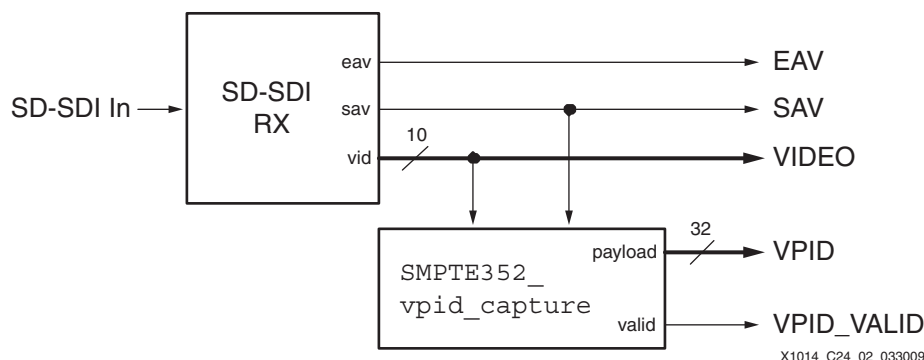


Figure 26-2: VPID Capture for SD-SDI

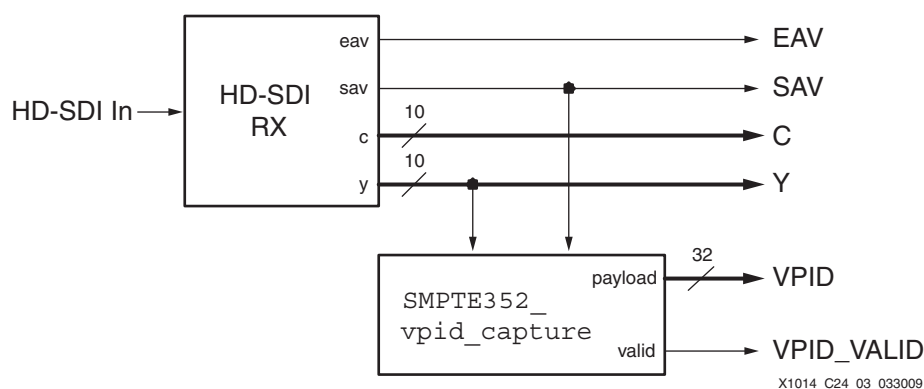


Figure 26-3: VPID Capture for HD-SDI

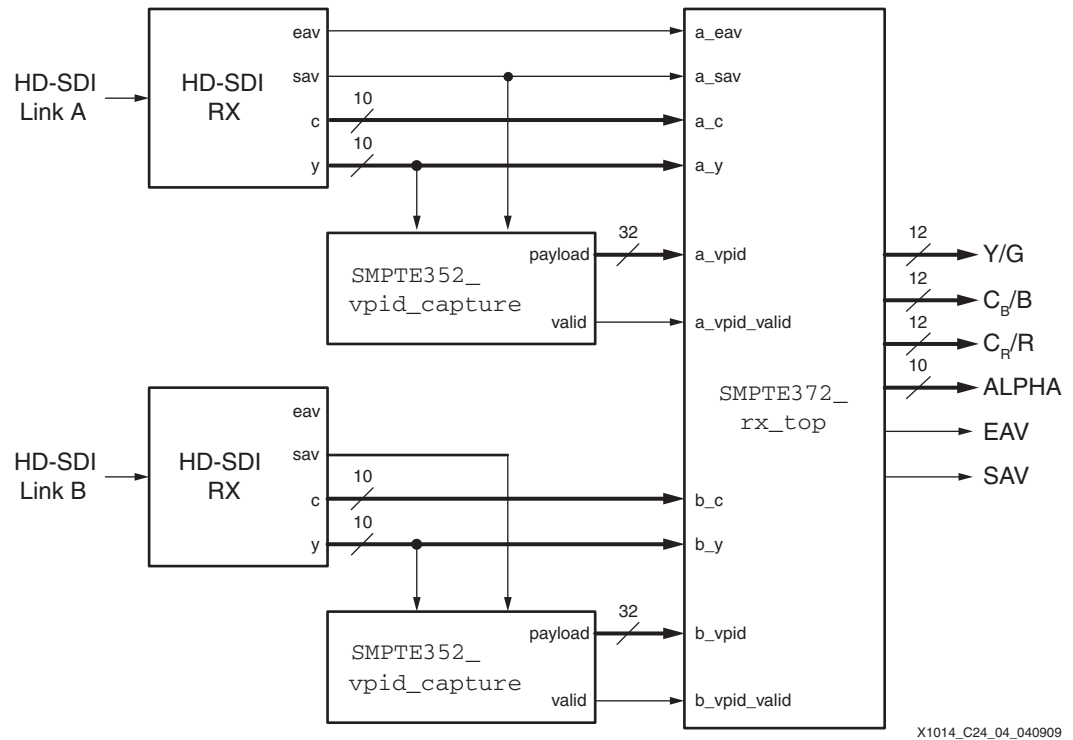


Figure 26-4: VPID Capture for Dual Link HD-SDI

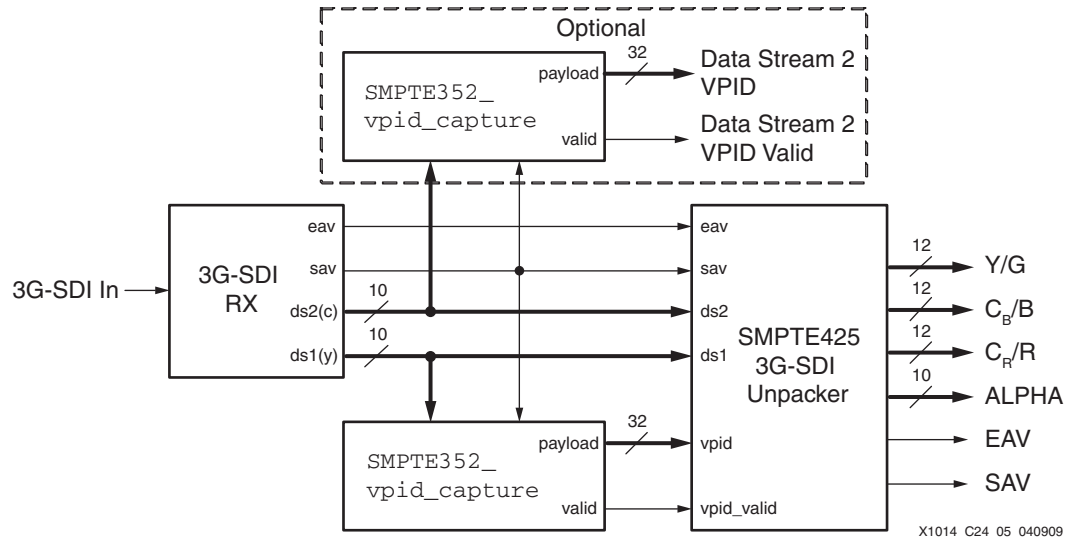


Figure 26-5: VPID Capture for 3G-SDI Level A

Figure 26-5 shows how to capture VPID packets for 3G-SDI level A interfaces. In this figure, the second SMPTE352_vpid_capture module is shown as optional. If the receiver is designed only for SMPTE 425M level A operation, the second module is optional because the VPID packets on the two data streams of the 3G-SDI interface should always be identical. SMPTE 425M level B has a different method of inserting the VPID packets. Essentially, the packets are inserted as they would be in a dual link HD-SDI (SMPTE 352M) interface. The 3G-SDI data streams must be unpacked into two HD-SDI

data streams, and then the VPID packets can be captured from the two Y data streams as shown in [Figure 26-4](#).

The `SMPTE352_vpid_capture` module sits in parallel with the data stream, monitoring the data stream as it passes by. The module does not sit directly in the video datapath and can be placed anywhere in the application. However, in 3G-SDI and dual link HD-SDI applications, ancillary data such as SMPTE 352M packets is destroyed by the unpacking process used to convert the SDI data streams to native video. In these cases, the SMPTE 352M packets must be captured from the raw SDI streams prior to unpacking them into native video.

The `SMPTE352_vpid_capture` module outputs a signal called `valid` that indicates whether valid VPID packet data is present on the module's payload output port. This payload output port is updated each time an SMPTE 352M packet is detected, as long as it is error free. Erroneous packets are discarded and the payload port retains the information from the last good packet received. The `valid` output goes High immediately after receipt of a good SMPTE 352M packet. If SMPTE 352M packets with errors are detected or the packets are no longer present in the video stream, the `valid` output remains asserted High for some number of frames or fields after the last good packet is received. This timeout period is controlled by the module's `VPID_TIMEOUT_VBLANKS` parameter/generic. The timeout period is in vertical blanking intervals and is equal to $2^{VPID_TIMEOUT_VBLANKS}$. `VPID_TIMEOUT_VBLANKS` defaults to a value of 4, giving a timeout period of sixteen (2^4) frames for progressive video and sixteen fields for interlaced video. Such a timeout period is useful because downstream modules in the application must often adjust their operation based on the VPID information captured by the `SMPTE352_vpid_capture` module. The operation of the downstream modules should not be disrupted by one erroneous VPID packet.

The actual interpretation of the video payload identification data is not done by the `SMPTE352_vpid_capture` module. This interpretation is very much application dependent and is based on which types of video interfaces are to be supported and which portions of the VPID data are interesting to the application. The module just provides all four bytes of the VPID data on its payload output port.

[Table 26-5](#) lists the ports of the `SMPTE352_vpid_capture` module.

Table 26-5: SMPTE352_vpid_capture Module Ports

Port	I/O	Width	Description
clk	In	1	This input clock must run at the video word rate or a multiple thereof.
ce	In	1	This is a clock enable. If clk is a multiple of the video word rate, ce must be asserted at the video word rate whenever an input video sample is available. If clk runs at the video word rate, ce must be High all of the time.
rst	In	1	This asynchronous reset input resets the module when High. The falling edge of this input must meet setup and hold times of the flip-flops relative to the rising edge of clk.
sav	In	1	This input must be High during the fourth (XYZ) word of each SAV sequence.
vid_in	In	10	This is a 10-bit data stream input.

Table 26-5: **SMPTE352_vpid_capture Module Ports (Cont'd)**

Port	I/O	Width	Description
payload	Out	32	This is an output port with all four payload identification bytes from the last good received packet arranged as byte4, byte3, byte2, byte1.
valid	Out	1	This output is always High when the payload output port has valid data from a recently received packet. After a certain period of time without detection of good packets, the valid output times out and goes Low. The timeout period is controlled by VPID_TIMEOUT_VBLANKS.

SMPTE352_vpid_insert

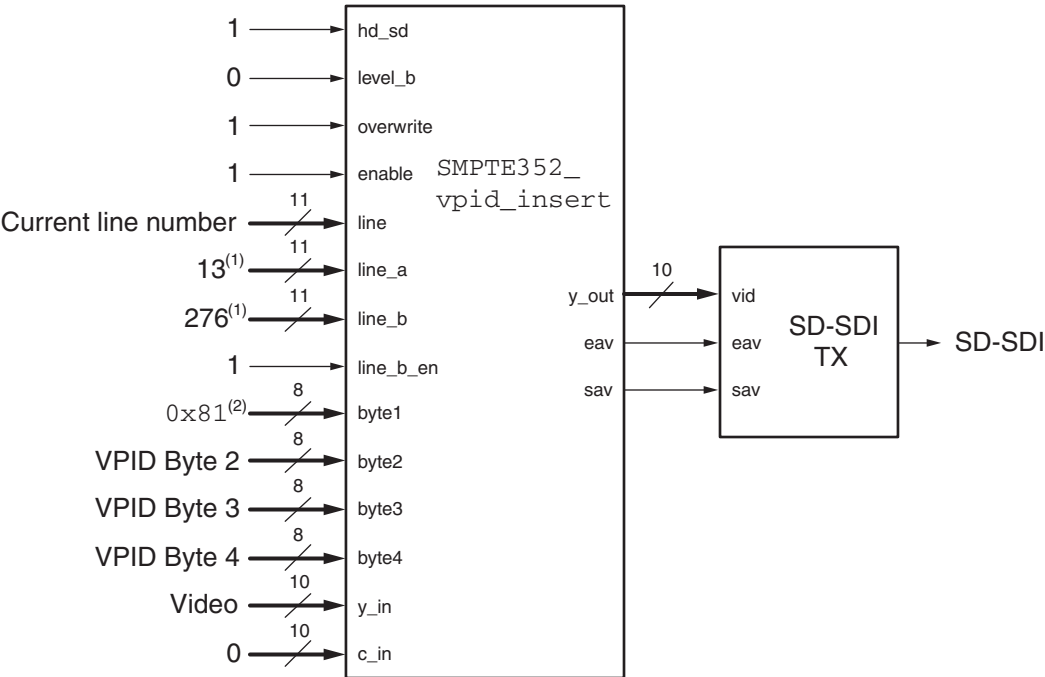
The SMPTE352_vpid_insert module inserts VPID packets on the specified lines on one 10-bit data stream. It supports insertion of the packets in both SD and HD video streams.

This module does not create the four data bytes of the payload information that go into the UDWs of the packet. This must be done by the application based on the video payload and interface types currently in use by the application.

The SMPTE352_vpid_insert module sits directly in the video datapath and modifies the Y data stream as it passes through the module. It introduces five clock cycles of latency (five clock cycles with the clock enable asserted). To keep the C data stream synchronized with the Y data stream, the C data stream passes through the module and is also delayed by five clock cycles. The module also creates EAV and SAV timing outputs that are properly synchronized to the output data streams.

For interfaces such as 3G-SDI, VPID packets must be inserted into both data streams. For those applications, two SMPTE352_vpid_insert modules are used with data stream 1 processed by the Y channel of the first module and data stream 2 processed by the Y channel of the second module. In this case, the C data streams through the two modules are unused.

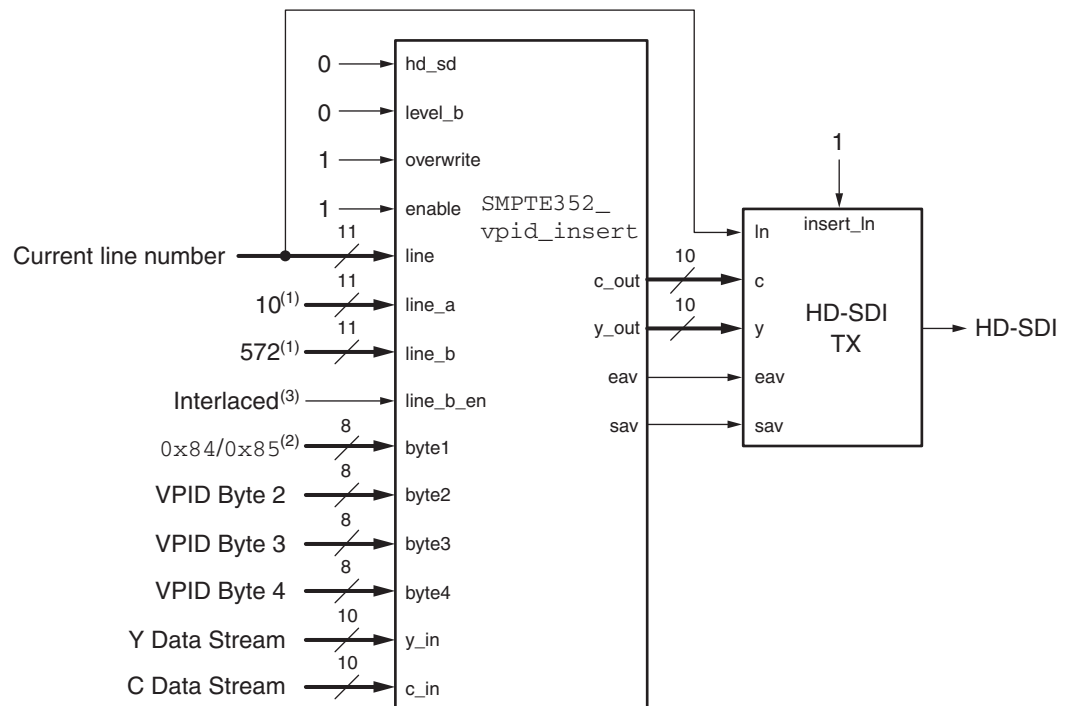
Figure 26-6 and Figure 26-7 show how the VPID insertion module is used for SD-SDI and HD-SDI interfaces, respectively.



1. Values shown are the VPID line numbers for 525-line NTSC video.
2. 0x81 is the correct value for 525 lines or 625 lines on a 270 Mb/s SD-SDI interface.

X1014_C24_06_040809

Figure 26-6: VPID Insertion for SD-SDI



1. Values shown are the VPID line numbers for HD video.
2. 0x84 is the correct value for SMPTE 296M 720p video on an HD-SDI interface, and 0x85 is the correct value for SMPTE 274M 1080i and 1080p video on an HD-SDI interface.
3. Insertion of VPID packets on line 572 (line_b) must be enabled for interlaced video and disabled for progressive video.

X1014_C24_07_040809

Figure 26-7: VPID Insertion for HD-SDI

Table 26-6 lists the ports of the SMPTE352_vpid_capture module.

Table 26-6: SMPTE352_vpid_insert Module Ports

Port	I/O	Width	Description
clk	In	1	This input clock must run at the video word rate or a multiple thereof.
ce	In	1	This is a clock enable. If clk is a multiple of the video word rate, ce must be asserted at the video word rate whenever an input video sample is available. If clk runs at the video word rate, ce must be High all of the time.
rst	In	1	This asynchronous reset input resets the module when High. The falling edge of this input must meet setup and hold times of the flip-flops relative to the rising edge of clk.
hd_sd	In	1	This input specifies whether the video stream is HD (0) or SD (1). The module needs to know whether the data stream is HD or SD so that it knows where the HANC space begins relative to the end of the EAV. This input should be Low for 3G-SDI.

Table 26-6: SMPTE352_vpid_insert Module Ports (Cont'd)

Port	I/O	Width	Description
level_b	In	1	This input must be High if the data stream carries level B SMPTE 425M 3G-SDI data, otherwise it must be Low. The designer should ensure that this input is Low for level A 3G-SDI data streams. The VPID packets are improperly positioned and can overwrite EAV, CRC, and LN data if this signal is not properly controlled.
enable	In	1	If this input is Low, VPID packet insertion is disabled. This input must be High for normal operation.
overwrite	In	1	If this input is Low, the module does not insert new VPID packets if VPID packets already exist at the correct locations in the data stream. If this input is High, existing VPID packets are overwritten with new VPID packets.
line	In	11	This is the current line number. This line number is compared to the line_a and line_b inputs to determine whether the current line should contain a VPID packet.
line_a	In	11	This is the line number where the VPID packet is to be inserted for field 1.
line_b	In	11	This is the line number where the VPID packet is to be inserted for field 2.
line_b_en	In	1	If this input is Low, packets are not inserted into the line specified by the line_b port. If this input is High, packets are inserted into the line specified by line_b. This input must be Low for progressive transport and High for interlaced transport.
byte1	In	8	This is the value of VPID packet byte 1.
byte2	In	8	This is the value of VPID packet byte 2.
byte3	In	8	This is the value of VPID packet byte 3.
byte4	In	8	This is the value of VPID packet byte 4.
y_in	In	10	This is the Y data stream input to the module. The VPID packet is inserted into this data stream.
c_in	In	10	This is the C data stream input to the module. VPID packets are not inserted into the c_in data stream, but this data stream is delayed by the same amount as the Y data stream so that the data streams are synchronized on the output of the module.
y_out	Out	10	This is the Y data stream output.
c_out	Out	10	This is the C data stream output.
eav_out	Out	1	This output is asserted High when the XYZ word of each EAV is present on the y_out and c_out ports.
sav_out	Out	1	This output is asserted High when the XYZ word of each SAV is present on the y_out and c_out ports.

The application must specify to the module which video lines the VPID packets are to be inserted into. This is done with the line_a and line_b ports. The values placed on these ports are specific to the current video format, as shown in Table 26-4. For progressive video, VPID packets are only inserted into one line per frame. This line must be specified on the line_a port, and the line_b_en input must be Low to disable packet insertions into the line specified by the line_b port. The insertion module obeys these rules for inserting and overwriting VPID packets:

- If there is an existing VPID packet at the beginning of the HANC space on a line where the VPID packet is supposed to occur, the module overwrites the packet with the new VPID information if the overwrite input is High. If the overwrite input is Low, the module does nothing.
- If there is a different type of ANC packet (or multiple ANC packets) at the beginning of the HANC space on a line where the VPID packet is supposed to occur, the module does not overwrite this existing ANC packet(s). Instead, it looks for an empty space in the HANC space to insert the VPID packet after the existing ANC packet(s). If the module finds a VPID packet later in the HANC space before it finds an empty space, it overwrites the existing VPID packet with new data if overwrite is High or does nothing if overwrite is Low.
- If there is no ANC packet at the beginning of the HANC space on a line where the VPID packet is supposed to occur, the module inserts the VPID packet at the beginning of the HANC space.
- If the module finds a VPID packet on a line where VPID packets are not supposed to occur and the overwrite input is High, the existing VPID packet is marked for deletion by changing its DID value to 0x180 and updating the checksum of the packet. If overwrite is Low, the module does nothing.

Designers using the insertion module should also be aware of these behaviors of the module:

- The SMPTE 352M standard specifies that the VPID packet should go at the very beginning of the HANC space. The module tries to do this, but it does not overwrite existing ANC packets if they occur at the beginning of the HANC space. The module tries to insert the VPID packet later in the HANC space, as described in the second rule above. To be fully compliant with the standard requires the application to always provide the module a data stream with blank HANC spaces on the lines where the VPID packets are to be inserted so that the VPID packet can be placed at the beginning of the HANC space.
- When looking for a place to insert the VPID packet, if the module finds a deleted ANC packet (DID = 0x180) with a data count of four or more, the module overwrites the deleted packet with the VPID packet. If the deleted packet is longer than the VPID packet, portions of the deleted packet are left over after the VPID packet. The module does not overwrite those additional words of the deleted packet or attempt to create a new deleted ANC packet to encompass the leftover words. *This behavior can lead to a violation of the ANC packet rules as defined in SMPTE 291M. Designers are cautioned to understand this behavior and only use this module when this behavior will not cause problems.*
- If the module inserts a VPID packet and later finds another VPID packet in the same HANC space, it does not delete or overwrite the existing VPID packet.
- When inserting a VPID packet, the module does not check to determine if there is enough space for the packet. It simply inserts the packet according to the rules listed above. However, the module never writes data beyond the end of the HANC space, so it never overwrites the SAV. As a result, it is possible for the module to insert a partial VPID packet before running out of room in the HANC space.

It is possible to use the `SMPTE352_vpid_insert` module to delete all VPID packets in the stream without inserting any packets. To do this, `overwrite` should be set High and `line_a` and `line_b` should be set to all zeros. Because a line number value of 0 is never found in the video, but the line number comparison treats a value of 0 on the `line_a` and `line_b` inputs as valid, the module always treats any detected VPID packets as not being on the correct line and changes them to deleted ANC packets.

Other Modules Used in the Reference Design

The `SMPTE352_vpid_insert` module uses a module called `wide_SRLC16E`. This is a shift register function built in a look-up table (SRL) that is up to 16 locations deep and multiple bits wide. The width of the module is controlled by a Verilog parameter or VHDL generic applied to the module when it is instantiated. This module works with any Xilinx FPGAs that support SRL primitives.

Module FPGA Resource Usage

Table 26-7 shows the FPGA resources required to implement the `SMPTE352_vpid_capture` and the `SMPTE352_vpid_insert` modules. The results in Table 26-7 were obtained with ISE® software, version 9.1 with XST set to optimize for area for the Virtex devices and speed for the Spartan-3E devices. The design meets timing when running at a 148.5 MHz sample rate at the slowest speed grade of Virtex-II Pro and Virtex-5 FPGAs. However, in the Spartan-3E FPGA, the fastest speed grade must be used to support 148.5 MHz sample rates. If only 74.25 MHz sample rates are needed, the slowest speed grade Spartan-3E devices can be used. 148.5 MHz sample rates are only required for dual link HD-SDI with 1080p 50 Hz, 59.94 Hz, and 60 Hz video, and with 3G-SDI level A.

Table 26-7: FPGA Resource Usage for SMPTE 352M Packet Capture and Insert Modules

Family	Reference Design	Flip-Flops	LUTs
Spartan-3E	SMPTE352_vpid_capture	87	52
Spartan-3E	SMPTE352_vpid_insert	101	202
Virtex-II Pro	SMPTE352_vpid_capture	84	48
Virtex-II Pro	SMPTE352_vpid_insert	105	167
Virtex-5	SMPTE352_vpid_capture	84	38
Virtex-5	SMPTE352_vpid_insert	105	146

Conclusion

VPID packets provide a means of uniquely identifying the essential characteristics of the video payload carried on SMPTE digital interfaces. VPID packets are required by the SMPTE 372M dual link HD-SDI and SMPTE 425M 3G-SDI standards and are optional with HD-SDI and SD-SDI interfaces.

Section VII: Appendices

Audio/Video Connectivity Solutions for Virtex-5 FPGAs

SMPTE Standards Related to SDI

This appendix lists some of the SMPTE standards related to SD-SDI, HD-SDI, dual link HD-SDI, and 3G-SDI. All SMPTE standards are available at www.smpte.org. The ITU has equivalents to some of these standards at www.itu.int.

EG 33: Jitter Characteristics and Measurements. This document provides guidelines for measuring jitter on SDI interfaces.

RP 165: Error Detection Checkwords and Status Flags for Use in Bit-Serial Digital Interfaces for Television. This is the EDH standard for SD-SDI.

RP 168: Definition of Vertical Interval Switching Point for Synchronous Video Switching. This standard defines the location of the synchronous switching point for various video formats. The synchronous switching point is the location where video equipment can switch between two different synchronous video signals safely without disrupting the video image.

RP 178: Serial Digital Interface Checkfield for 10-Bit 4:2:2 Component and 4fSC Composite Digital Signals. This standard defines a test pattern used to produce the SD-SDI pathological patterns, which are worst-case encoded patterns that stress the cable equalizer and the clock and data recovery unit of an SD-SDI receiver.

RP 184: Specification of Jitter in Bit-Serial Digital Systems. This standard defines how jitter is specified in SMPTE serial digital interfaces such as HD-SDI and 3G-SDI.

RP 198: Bit-Serial Digital Checkfield for Use in High-Definition Interfaces. This standard defines a test pattern used to produce the HD-SDI pathological patterns, which are worst-case encoded patterns that stress the cable equalizer and the clock and data recovery unit of an HD-SDI receiver. This document also applies to 3G-SDI.

RP 291: Assigned Ancillary Identification Codes. This companion to SMPTE 291M documents all defined ANC packed DID codes.

SMPTE 125M: Component Video Signal 4:2:2–Bit-Parallel Digital Interface. This standard defines the parallel format of standard-definition 4:2:2 digital component NTSC video that can be carried by SD-SDI.

SMPTE 259M: SDTV Digital Signal/Data–Serial Digital Interface. This is the SD-SDI standard.

SMPTE 272M: Formatting AES Audio and Auxiliary Data into Digital Video Ancillary Data Space. This is the embedded audio standard for SD-SDI.

SMPTE 274M: 1920 x 1080 Image Sample Structure, Digital Representation and Digital Timing Reference Sequences for Multiple Picture Rates. This standard defines the 1080-line digital video formats that are commonly used in the broadcast industry. These video formats can be carried by HD-SDI, dual link HD-SDI, and 3G-SDI.

SMPTE 291M: Ancillary Data Packet and Space Formatting. This standard defines the generic format of ancillary data packets and specifies where in the data stream these packets can be placed.

SMPTE 292M: 1.5 Gb/s Signal/Data Serial Interface. This is the HD-SDI standard.

SMPTE 296M: 1280 x 720 Progressive Image Sample Structure–Analog and Digital Representation and Analog Interface. This standard defines the 720-line digital video formats that are commonly used in the broadcast industry. These video formats can be carried by HD-SDI and 3G-SDI.

SMPTE 299M: 24-Bit Digital Audio Format for SMPTE 292M Bit-Serial Interface. This is the embedded audio standard for HD-SDI. It also applies to dual link HD-SDI and 3G-SDI.

SMPTE 305M: Serial Data Transport Interface. Also known as SDTI, this standard describes how packetized data can be carried on SD-SDI interfaces in place of uncompressed digital video.

SMPTE 344M: 540 Mb/s Serial Digital Interface. This standard defines the physical layer for a 540 Mb/s version of SD-SDI.

SMPTE 346M: Time Division Multiplexing Video Signals and Generic Data over High-Definition Interfaces. This standard describes how to transport multiple standard-definition video streams and other data on a single HD-SDI interface.

SMPTE 347M: 540 Mb/s Serial Digital Interface–Source Image Format Mapping. This standard defines how to map various digital video formats onto the SMPTE 344M 540 Mb/s SD-SDI interface.

SMPTE 348M: High Data-Rate Serial Data Transport Interface (HD-SDTI). This standard defines how to transport packetized data rather than uncompressed video on the HD-SDI interface.

SMPTE 349M: Transport of Alternate Source Image Formats through SMPTE 292M. This standard specifies how to transport standard definition video formats over HD-SDI.

SMPTE 352M: Video Payload Identification for Digital Interfaces. This standard defines an ANC packet type that provides information about the video payload carried by a digital interface. SMPTE 352M packets are required by the dual link HD-SDI and 3G-SDI standards. These packets are optional for SD-SDI and HD-SDI.

SMPTE 372M: Dual link 1.5 Gb/s Digital Interface for 1920 x 1080 and 2048 x 1080 Picture Format. This is the dual link HD-SDI standard. It describes how to combine two HD-SDI interfaces into a single virtual interface with about 3 Gb/s of bandwidth.

SMPTE 424M: 3 Gb/s Signal/Data Serial Interface. This is the 3G-SDI standard.

SMPTE 425M: 3 Gb/s Signal/Data Serial Interface–Source Image Format Mapping. This is the 3G-SDI format mapping document. It describes how to map various video formats on to the 3G-SDI interface.

SMPTE 428-9: D-Cinema Distribution Master – Image Pixel Structure Level 3–Serial Digital Interface Signal Formatting. This standard describes how to map the 2048 x 1080p 24 Hz digital cinema video format (defined in SMPTE 428-1) into a container file compatible with dual link HD-SDI and 3G-SDI.

Glossary

3G-SDI: Common name for SMPTE 424M, the 3 Gb/s serial digital interface.

Active line: The portion of the video line that includes visible picture information.

AES/EBU: A standard for digital audio jointly defined by the Audio Engineering Society and the European Broadcasting Union. This is essentially the same as AES3.

AES3: A commonly used standard for professional digital audio. AES3 and AES/EBU are essentially identical. The format for embedded audio in SDI streams is based on the AES3 and AES/EBU standards.

Alpha: Also called the alpha channel or key channel, this is the fourth component included in some video formats to indicate the transparency of the video signal. The transparency information is used when multiple layers of video are alpha blended or composited.

Ancillary (ANC) data: Non-video data embedded in unused portions of the video data stream. One very common type of ANC data is embedded digital audio. ANC data must be formatted into packets, as specified by SMPTE 291M, when embedded in SDI data streams.

Ancillary data flag (ADF): A sequence of three words, unique in the data stream, that marks the beginning of an ANC packet.

Aspect ratio: The ratio of the length to the height of pictures. The aspect ratio for most standard definition video is 4:3. The aspect ratio for high-definition video is 16:9.

Channel: In some cases, the digital interface has sufficient capacity to carry more than one video payload, and each video payload becomes a channel of the digital interface. The word “channel” also has a variety of other meanings in various SMPTE documents, making it an overloaded and ambiguous term. For example, the term “audio channel” identifies one of a group of usually related audio signals.

Chroma or chrominance: The color information of the video signal.

Component video: A video signal in which the luma and chroma are in separate components, usually with multiple chroma components.

Composite video: A video signal such as NTSC or PAL where the luma, chroma, and sync signals are combined into a single signal.

Data identification (DID): The value in an ancillary data packet that immediately follows the ADF and identifies the data packet type.

Data stream: The actual data into and out of the interface. The data stream must be formatted according to the transport data structure as it enters and exits the interface.

Dual link: This occurs when two SD-SDI or HD-SDI interfaces are combined to carry a single video stream. For example, SMPTE 372M dual link HD-SDI doubles the bandwidth

of a single HD-SDI signal to carry higher bandwidth HD video formats such as 1080p 60 Hz.

EDH: The error detection and handling protocol for SD-SDI. This is defined by SMPTE RP 165.

Embedded audio: Generally refers to digital audio that is carried, along with the video, in an SDI signal.

End of active video (EAV): In digital video streams, the EAV is a sequence of four words, unique in the data stream, marking the end of the active video portion of a line and the start of the horizontal blanking interval. Each video line is considered to begin with the first word of the EAV.

Format: Video pictures are generally defined by their format, which includes such characteristics as the number of lines, the number of samples per line, the frame rate, and whether the picture is interlaced or progressive.

HD-SDI: Common name for the SMPTE 292M 1.5 Gb/s serial digital interface.

Horizontal ancillary data (HANC): Data embedded in the horizontal blanking interval of a line. The horizontal blanking interval, exclusive of the SAV and the EAV (and its subsequent LN and CRC words), is also called the HANC space.

Interface: An interconnection defined by its electrical characteristics, cables, connectors, scrambling system, bit rates, and transport data structure.

Interlaced: A scanning system in which the video frame is divided into two sequential fields. Field one consists of the odd lines and field two consists of the even lines that are displayed between the odd lines of field one. The two fields represent different pictures displaced in time by one half of the frame time.

Link: If the picture's bandwidth exceeds the capacity of the digital interface, two or more digital interfaces can be ganged together to increase the capacity. Each digital interface is one link of the combined interface set.

Luma (luminance): The intensity or lightness component of the video signal. Technically, the terms "luma" and "luminance" are not precisely the same, but are often used interchangeably.

Mapping: The method used to place the various video components of video samples onto the data streams of the interface. HD-SDI uses a very simple mapping: Y' component on data stream 1, and C_{B'} and C_{R'} components interleaved on data stream 2. Dual link HD-SDI and 3G-SDI each have a number of different mappings to deal with different video formats, component bit depths, and sampling structures.

Progressive: A non-interlaced scanning system. All lines of the progressive frame belong to the same picture. The frame is not divided into temporally displaced fields.

Progressive segmented frame (PsF): A transport system in which a progressive frame is divided into two fields that are sent sequentially. The image contained in the two fields is a single progressive frame, not two fields of an interlaced frame. PsF is used to allow a progressive video signal to appear to be interlaced so as to be compatible with equipment, typically video tape recorders, that do not support progressive video formats. To be displayed properly, the two fields of a PsF image must be recombined into a single frame and displayed progressively.

R'G'B': This color space has three color components: red (R'), green (G'), and blue (B'). The apostrophe indicates gamma-corrected components.

Sampling structure: The method by which each picture sample is structured. For HD-SDI, the most common sampling structure is 4:2:2, where the luma component is horizontally sampled at twice the rate of the two color difference components (often called chroma components). A full description of the notation used to describe the sampling structure, such as 4:2:2, is beyond the scope of this document. However, for the purposes of this application note, the first number is almost always 4, indicating the rate at which the luma component is sampled. The second digit represents the horizontal subsampling of C_B and C_R relative to Y , with a value of 2 indicating that C_B and C_R are sampled at one-half the rate of the Y component. The third digit is the same as the second digit if the C_B and C_R components are sampled at the same rate as Y in the vertical direction, or 0 if they are sampled at half the rate of Y vertically. The other common sampling structure used for HD-SDI is 4:4:4, where all three components are sampled at the same rate both horizontally and vertically. Sometimes, a fourth component, usually called the alpha component, is also included, in which case the sampling structure can be indicated as 4:2:2:4 or 4:4:4:4.

Serial Digital Interface (SDI): Originally referred to SMPTE 259M, the standard-definition serial digital interface. With the advent of HD-SDI and 3G-SDI, SMPTE 259M is now often called SD-SDI to avoid confusion. This document uses the term SDI to generically refer to SD-SDI, HD-SDI, and 3G-SDI. When referring specifically to SMPTE 259M, this document uses SD-SDI.

SD-SDI: Common name for SMPTE 259M, the standard-definition serial digital interface.

SMPTE: Society of Motion Picture and Television Engineers.

Start of active video (SAV): In the digital video stream, the SAV is a sequence of four words, unique in the data stream, marking the end of the horizontal blanking interval and the start of the active video portion of a line. The first active video sample of a line, usually called sample 0, occurs immediately after the SAV.

Synchronous switching (point, interval, line): RP 168 defines the point(s) in a video frame where it is permissible to switch between synchronous video sources. This is often called the synchronous switching point, but is actually defined as an interval—a portion of a line, rather than an exact point on a line. The line that contains the synchronous switching interval is called the synchronous switching line.

Transport: The data structure of an interface data stream or streams. The transport data structure defines the EAV and SAV sequences used to carry video timing information and synchronize the transport words at the receiver.

Timing reference signal (TRS): A generic term referring to both EAV and SAV sequences.

Vertical ancillary data (VANC): Data embedded in the active portion of lines in the vertical blanking interval. The active portions of all lines in the vertical blanking interval are also collectively called the VANC space.

Video payload: The picture carried by a digital interface.

XYZ: The fourth word of each EAV or SAV is called the XYZ word. This word carries the horizontal (H), vertical (V), and field (F) bits that indicate the video timing and protection bits that allow detection of errors in the XYZ word. This should not be confused with the $X'Y'Z'$ color space used by digital cinema video formats (SMPTE 428-1) compatible with dual link HD-SDI and 3G-SDI.

$Y'C_B'C_R'$: This color space has three components: the luma component Y' and two color difference components C_B' and C_R' . The apostrophe after each component indicates that it is a gamma-corrected component. $Y'C_B'C_R'$ always describes digital video. The analog equivalent is $Y_P B_P R_P$.

