Convolutional Encoder v9.0

LogiCORE IP Product Guide

Vivado Design Suite

PG026 November 18, 2015

XILINX
ALL PROGRAMMABLE™

Table of Contents

Appendix B: Debugging

Appendix C: Additional Resources and Legal Notices

Introduction

The Xilinx® LogiCORE™ IP Convolutional Encoder core can be used in a wide variety of error correcting applications and is typically used in conjunction with the Viterbi Decoder [Ref 3].

Features

- High-speed compact convolution encoder with puncturing option
- Parameterizable constraint length from three to nine
- Parameterizable convolution codes
- Parameterizable puncture codes
- Puncturing rates from 2/3 to 12/23 available

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale+™ Families UltraScale™ Architecture Zynq®-7000 All Programmable SoC 7 Series
Supported User Interfaces	AXI4-Stream
Resources	Performance and Resource Utilization web page
Provided with Core	
Design Files	Encrypted RTL
Example Design	Not Provided
Test Bench	See Test Bench.
Constraints File	N/A
Simulation Model	VHDL Behavioral VHDL and Verilog Structural
Supported S/W Driver(2)	N/A
Tested Design Flows(2)	
Design Entry	Vivado® Design Suite System Generator for DSP
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete listing of supported devices, see the Vivado IP catalog.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

The Convolutional Encoder core is used to encode data prior to transmission over a channel. The received data is decoded by the classic Viterbi decoder. In a basic convolution encoder, two or three bits (depending on the encoder output rate) are transmitted over the channel for every input bit.

Applications

The core is used in a wide variety of error correcting applications and is typically used in conjunction with the Viterbi Decoder [Ref 3].

Unsupported Features

The core only supports a fixed rate puncturing option. The puncturing rate cannot be varied dynamically.

Licensing and Ordering Information

This Xilinx® LogiCORE™ IP module is provided at no additional cost with the Xilinx Vivado® Design Suite under the terms of the Xilinx End User License. Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

The Convolutional Encoder core is used to encode data prior to transmission over a channel. The received data is decoded by the classic Viterbi decoder. In a basic convolution encoder, two or three bits (depending on the encoder output rate) are transmitted over the channel for every input bit.

The basic architecture of the Convolutional Encoder core is shown in Figure 2-1. The incoming data is brought into the constraint register a bit at a time, and the output bits are generated by modulo-2 addition of the required bits from the constraint register. The bits to be XOR'd are selected by the convolution codes as shown in Figure 2-1.

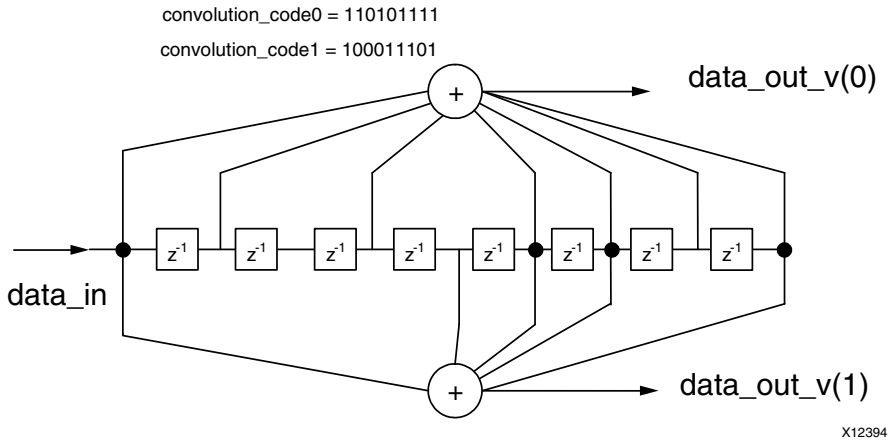

Figure 2-1: **Convolutional Encoder Constraint Length 9**

Standards

The Convolutional Encoder IP core uses the AXI4-Stream interface as per the *AMBA® AXI4-Stream Protocol Specification* (ARM IHI 0051A) [Ref 2].

Performance

This section details the performance information for various core configurations.

Latency

For this core, latency is defined as the number of rising clock edges from sampling S_DATA to the sampled value appearing on M_DATA. The latency varies with the puncture type and puncture rate. See Table 2-1 for some example latency figures.

Table 2-1: **Latency**

Convolutional Encoder Type	Latency
Non Puncture	3
Punctured dual rate 3/4	15
Punctured non-dual rate 3/4	10

Throughput

The maximum raw data input rate in Mb/s can be calculated as F_{max}(MHz) for non-punctured and dual encoder and F_{max}(MHz) * n/m for the punctured non-dual encoder, where *n* is the input rate and *m* is the output rate of the punctured core

Resource Utilization

For full details about performance and resource utilization, visit the Performance and Resource Utilization web page.

Resources required for the Convolutional Encoder core are of the order of a 20 to 100 LUTs depending on the selected puncture rate. The area of the core increases with the constraint length and the punctured input and output rates if the core is punctured.

Port Descriptions

Symbols for the core with the AXI channels are shown in Figure 2-2 and Figure 2-3.

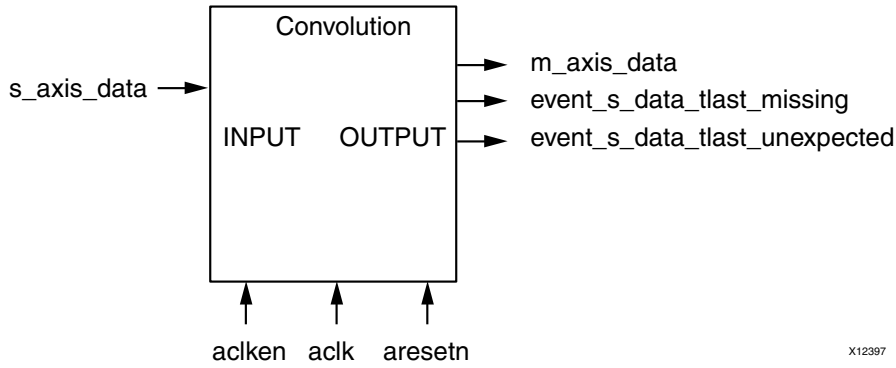

Figure 2-2: **Core AXI Channels**

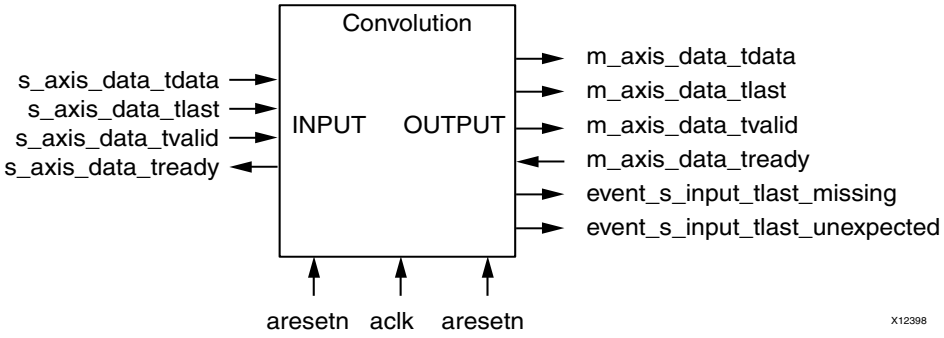

Figure 2-3: **Core Schematic Symbol**

Table 2-2 summarizes the signal functions. They are described in more detail in the remainder of this section. Timing diagrams for the signals are shown throughout this section.

Table 2-2: **Core Signal Pinout**

Signal	Direction	Description
aclk	Input	Rising edge clock
aclken	Input	Active-High clock enable (optional)
aresetn	Input	Active-Low synchronous clear (overrides aclken)
s_axis_data_tdata	Input	Input data
s_axis_data_tvalid	Input	tvalid for S_AXIS_DATA channel. See aclken for protocol.
s_axis_data_tlast	Input	Marks last symbol of input block. Only used to generate event outputs. Can be tied low or high if event outputs not used.
s_axis_data_tready	Output	tready for S_AXIS_DATA. Indicates that the core is ready to accept data. Always high, except after a reset, in the non punctured and Dual Output core if there is not a tready on the output.
m_axis_data_tdata	Output	tdata for the output data channel, convolutionally encoded data output in parallel for the non punctured and Dual Output case. Serial output for the punctured non-dual case.
m_axis_data_tvalid	Output	tvalid for M_AXIS_DATA channel
m_axis_data_tlast	Output	tlast for the output data channel, indicates the last element in the punctured group.
m_axis_data_tready	Input	tready for M_AXIS_DATA channel. Tie high if downstream slave is always able to accept data from M_AXIS_DATA
event_s_data_tlast_missing	Output	Flags that s_axis_data_tlast was not asserted when expected. Leave unconnected if not required.
event_s_data_tlast_unexpected	Output	Flags that s_axis_data_tlast was asserted when not expected. Leave unconnected if not required.

Notes:

1. All AXI4-Stream port names are lower case, but for ease of visualization, upper case is used in this document when referring to port name suffixes, such as tdata or tlast.

aclken

The clock enable input (`aclken`) is an optional pin. When `aclken` is deasserted (low), all the other synchronous inputs are ignored, except aresetn, and the core remains in its current state. This pin should be used only if it is genuinely required because it has a high fan out within the core and can result in lower performance. `aclken` is a true clock enable and causes the entire core to freeze state when it is low.

An example of `aclken` operation is shown in Figure 2-4. In this case, the core ignores symbol D_4 as input to the block, and the current `m_axis_data_tdata` value remains unchanged.

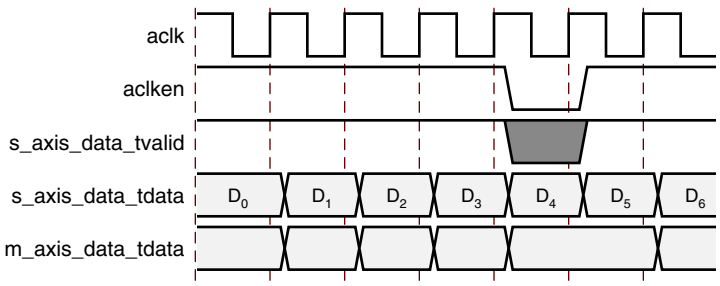

Figure 2-4: **Clock Enable Timing**

aresetn

The synchronous reset (`aresetn`) input can be used to re-initialize the core at any time, regardless of the state of `aclken`. `aresetn` needs to be asserted low for at least two clock cycles to initialize the circuit. The core becomes ready for normal operation two cycles after `aresetn` goes high if aclken is high. The timing for the `aresetn` input is shown in Figure 2-5.

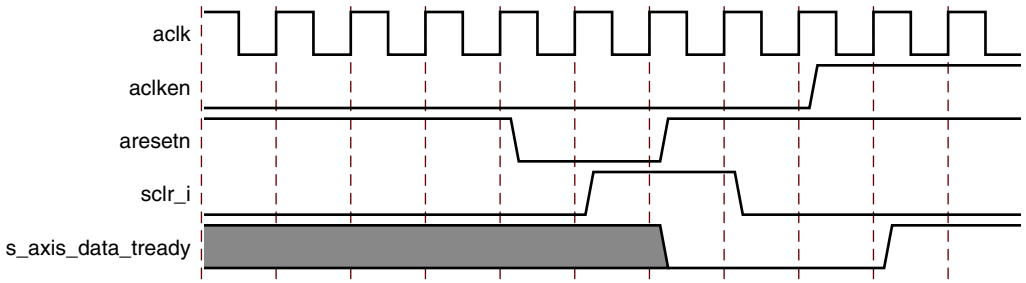

Figure 2-5: **Synchronous Reset Timing**

S_AXIS_DATA Channel

s_axis_data_tdata

Data to be processed is passed into the core on this port. To ease interoperability with byte-oriented buses, `tdata` is padded with zeros, for the convolutional encoder the bus is always 8 bits. The padding bits are ignored by the core and do not result in additional resource use. The structure is shown in Figure 2-6.

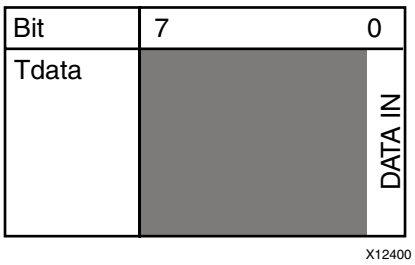

Figure 2-6: **Input Channel TDATA Structure**

DATA_IN Field

This is the input bit for the incoming data. The width of the DATA_IN field is always 1.

s_axis_data_tlast

This input can be tied low or high if the event outputs (`event_s_data_tlast_missing` and `event_s_data_tlast_unexpected`) are not used. It is present purely to provide a check that the system and core are in sync with the applied puncturing. If the event outputs are used then `s_axis_data_tlast` must be asserted high when the last symbol of a punctured group is sampled on `s_axis_data_tdata`. The core maintains its own internal count of the symbols, so it knows when the last symbol is being sampled. If `s_axis_data_tlast` is not sampled high when the last input symbol is sampled then `event_s_data_tlast_missing` is asserted until the next input sample is taken. Similarly, if `s_axis_data_tlast` is sampled high when the core is not expecting it, `event_s_data_tlast_unexpected` is asserted until the next input sample is taken. If either of these events occurs then the system and the core are out of sync and the core, and possibly the system, should be reset.

Single-Channel Output

For the single-channel output, after n input bits (puncture input rate) have been received, the `tready` signal goes low for m-n (where m is the puncture output rate) clock periods. The `m_axis_data_tvalid` signal is used to indicate that valid output data is available which is output in a block of m bits (puncture output rate), as shown in Figure 2-9.

Dual-Channel Output

For the dual-channel punctured case, `s_axis_data_tready` is always high (unless `aresetn` has been asserted or back pressure has been applied). The dual channel puncturing works on two blocks of n bits of data. After 2n input bits have been received, where n is the punctured input rate, m symbols are output on `m_axis_data_tdata` (width 2) after a certain latency. The `m_axis_data_tvalid` signal indicates the valid outputs. The data is output in blocks of m symbols (width 2) and the `m_axis_data_tvalid` signal is high for m clock cycles and low for the next 2n-m. See Figure 2-8 for the 3/4 rate dual-channel punctured encoder.

M_AXIS_DATA Channel

m_axis_data_tdata

Uncoded data sampled on `s_axis_data_tdata` is encoded and output from the core on this port. All output data is padded with zeroes to fit a bit field which is always 8 bits. The structure is shown in Figure 2-7.

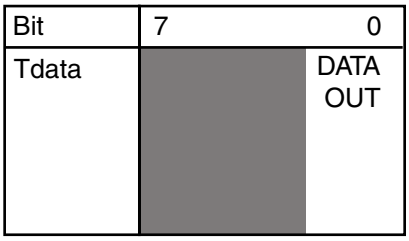

Figure 2-7: **Output Data Channel TDATA Structure**

DATA_OUT Field

This is the output field for the encoded bits. The width depends on the type of encoder. For the case of unpunctured data the width is equal to the output rate. For punctured data with serial output the width is 1 bit and for the dual channel encoder the output width is 2 bits. Corrected symbols start to appear many clock cycles after the first symbol is sampled on DATA_IN.

m_axis_data_tlast

This output is high when the last DATA_OUT of a punctured block is on `m_axis_data_tdata`. The port is not present on the non-punctured core. In the dual output case, `m_axis_data_tlast` is only asserted high when m output pairs have been output, this corresponds to 2n data inputs. This is shown in Figure 2-8. For the serial case the signal is asserted when m serial outputs have been output.This is shown in Figure 2-9.

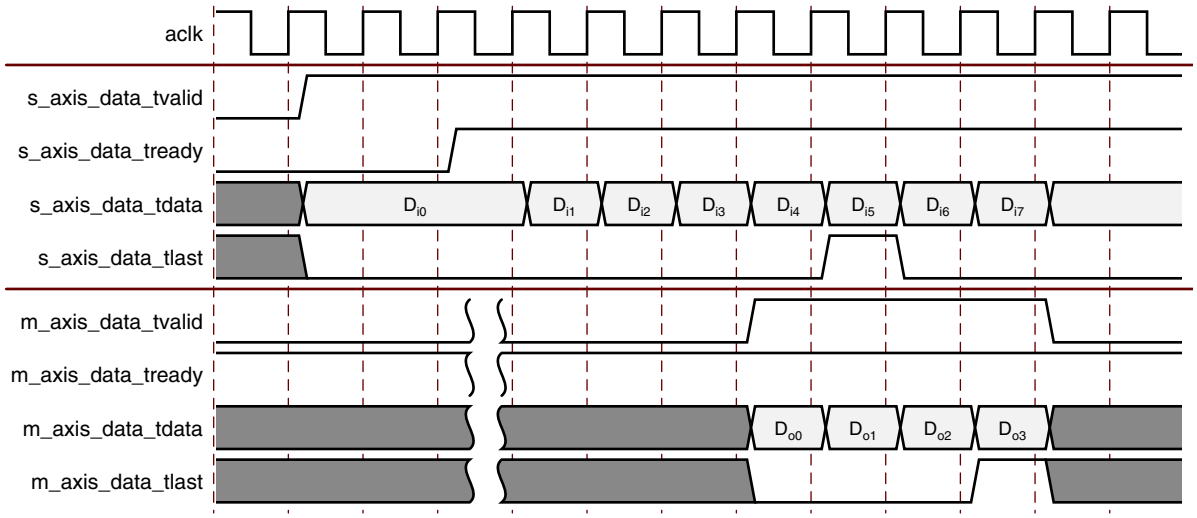

Figure 2-8: **Dual Output Timing for Rate 3/4 Puncturing**

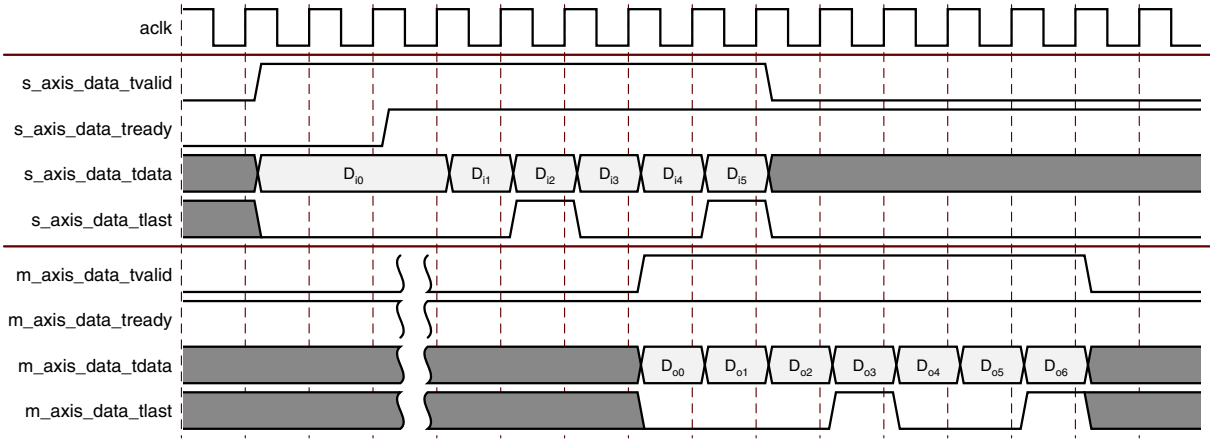

Figure 2-9: **Serial Output Timing for Rate 3/4 Puncturing**

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

General Design Guidelines

Puncturing

The encoder can also puncture the data prior to transmission. In this case, the basic convolutional encoder is always a rate 1/2 encoder, two bits output for every one bit input. After the encoding, certain bits of the rate 1/2 encoded data are punctured (or deleted) and not transmitted. Thus, for a rate 3/4 punctured encoder, for every three input bits, only four of the six encoded bits generated by the encoder are actually transmitted, as shown in Figure 3-1. The two bits output from the encoder are punctured according to a pair of puncture codes. The puncture code is a bit pattern that identifies which bits from the encoder are to be transmitted. The use of puncturing significantly reduces the number of bits to be transmitted over the channel. For rate n/m puncturing, a puncture code of length n is used and exactly m bits are transmitted where n<m<2n. For rate n/m puncturing, the length of each puncture code must be n bits and the total number of non-zero bits must be equal to m.

Note: The order of application of the puncture code is now left to right and matches the order of application of the convolution codes, see Figure 3-1, puncture code 1. The value of n can range from 2 to 12.

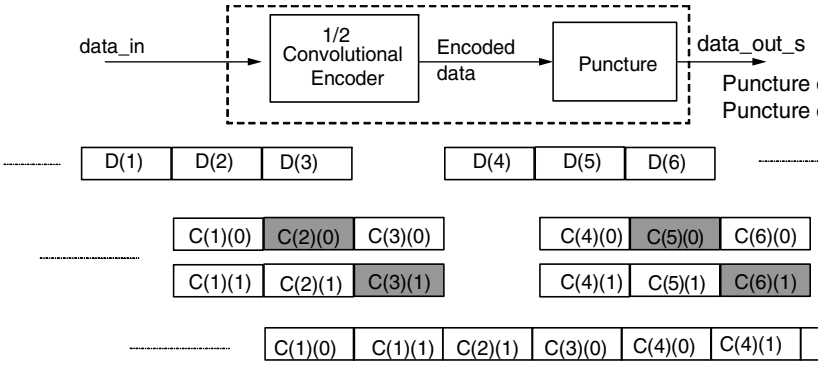

Figure 3-1: **Puncture Encoding with Single-Channel Output**

For the punctured encoder, the data can be output as a single bitstream, as shown in Figure 3-1. This single-channel output requires the use of the slave input `tvalid` and slave output `tready` as there are more bits output than bits input. Also, in the single-channel case, the `tready` signal is low for m-n clock cycles after receiving the n input bits, as shown Figure 2-8.

The data can also be output two bits at a time for the punctured encoder when the dual-channel output mode is selected, as shown in Figure 3-2. In this situation, there is less data output than input and the `tready` signal, if present, is only low if the master output data stream has been halted or the core reset. The master `tvalid` signal indicates when the output data is valid in both cases. For timing diagrams for the dual output case, see Figure 2-8**.**

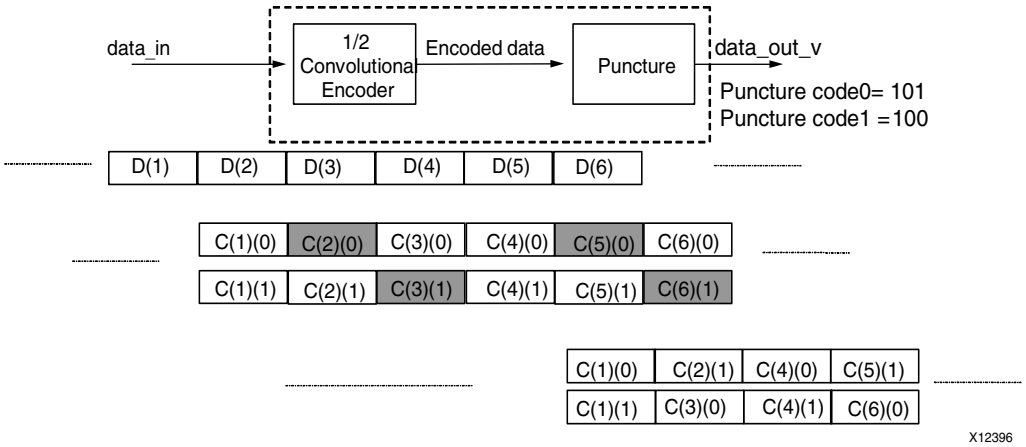

Figure 3-2: **Puncture Encoding with Dual-Channel Output**

Clocking

The Convolutional Encoder core is a fully synchronous circuit with a single clock. All interfaces are synchronous to the rising edge of the `aclk` signal.

Resets

The core can be returned to its default state by driving the `aresetn` signal Low. On the cycle directly following the deassertion of `aresetn`, the core is again idle and ready to accept new data. An initial application of `aresetn` at startup is permitted, but is not strictly necessary.

Protocol Description

AXI4-Stream Protocol

The use of AXI4-Stream interfaces standardizes and enhances the interoperability of Xilinx® LogiCORE™ IP solutions. Other than general control signals such as `aclk`, `aclken` and `aresetn`, and event outputs, all inputs and outputs to the core are conveyed through AXI4-Stream channels. A channel consists of `tvalid` and `tdata` always, plus several optional ports and fields. In the Convolutional Encoder, the additional ports used are `tready` and `tlast`. Together, `tvalid` and `tready` perform a handshake to transfer a value, where the payload is `tdata`, and `tlast`. The payload is indeterminate when `tvalid` is deasserted.

The Convolutional Encoder operates on the values contained in the S_AXIS_DATA channel `tdata` fields and outputs the results in the `tdata` fields of the M_AXIS_DATA channel. The Convolutional Encoder core does not use input `tlast` as such, `tlast` is provided purely as a check that the core is in sync with the system and its use is optional. For further details on AXI4-Stream Interfaces see [Ref 1] and [Ref 2].

Basic Handshake

Figure 3-3 shows the transfer of data in an AXI4-Stream channel. `tvalid` is driven by the source (master) side of the channel and `tready` is driven by the receiver (slave). `tvalid` indicates that the value in the payload fields (`tdata` and `tlast`) is valid. `tready` indicates that the slave is ready to receive data. When both `tvalid` and `tready` are TRUE in a cycle, a transfer occurs. The master and slave set `tvalid` and `tready` respectively for the next transfer appropriately.

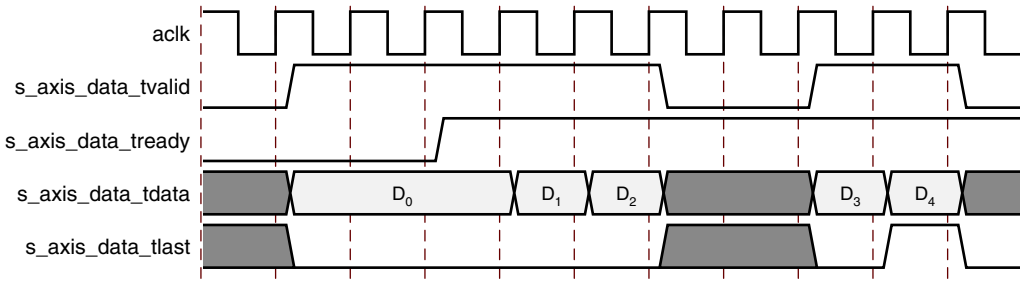

Figure 3-3: **Data Transfer in an AXI-Stream Channel**

The full flow control of AXI4-Stream aids system design because the flow of data is self-regulating. Data loss is prevented by the presence of back pressure (`tready`), so that data is only propagated when the downstream datapath is ready to process it.

The core has one output channel: M_AXIS_DATA. If the output is prevented from off-loading data because `m_axis_data_tready` is low then data accumulates in the core. When the core internal buffers are full the core stops further operations. When the internal buffers fill, the `tready` (`s_axis_data_tready`) is deasserted to prevent further input. This is the normal action of back pressure.

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 4]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 5]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 6]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7]

Customizing and Generating the Core

This section includes information about using Xilinx® tools to customize and generate the core in the Vivado® Design Suite.

If you are customizing and generating the core in the Vivado IP Integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 4] for detailed information. IP Integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value you can run the `validate_bd_design` command in the Tcl console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 5] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 6].

The customization parameters on the Vivado Integrated Design Environment (IDE) for the Convolutional Encoder are as follows.

Component Name

The component name is used as the base name of the output files generated for the core. Names must begin with a letter and must be composed from the following characters: a to z, 0 to 9 and "_".

Data Rates and Puncturing

Punctured

This parameter determines whether the core is punctured.

Not Punctured:	Only the output rate can be modified. Its value can be integer values from 2 to 7, resulting in a rate 1/2 to rate 1/7 encoder.
Punctured:	Both the input rate and output rate can be modified. The input value can range from 2 to 12, resulting in a rate n/m encoder where n is the input rate and n<m<2n.

Punctured Codes

The two puncture pattern codes used to remove bits from the encoded data prior to output. The length of each puncture code must be equal to the puncture input rate, and the total number of bits set to 1 in the two codes must equal the puncture output rate (m) for the codes to be valid. A 0 in any position indicates that the output bit from the encoder is not transmitted. See Figure 3-1 for an example of a rate 3/4 punctured encoder. For the puncturing case, the data can be output in a single bitstream or in the dual-channel mode where the data is output in parallel two bits at a time.

Convolution

Constraint Length

The length of the constraint register plus 1 in the encoder. This value can be in the range 3 to 9, inclusive.

Convolution Codes

Convolution codes are used to generate the encoder outputs. For a punctured core, only two are required, because the encoder internal to the punctured core is always rate 1/2. The codes can be entered in binary, octal or decimal and have length equal to the constraint length.

Optional Pins

Check the boxes of the optional pins that are required. Select only pins that are genuinely required, as each selected pin uses more FPGA resources and can result in a less-than-maximum operating frequency.

User Parameters

Table 4-1 shows the relationship between the fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl Console).

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter	User parameter	Default Value
Punctured	punctured	False
Input Rate	input_rate	1
Output Rate	output_rate	2
Puncture Code 0	puncture_code0	0
Puncture Code 1	puncture_code1	0
Constraint Length	constraint_length	7
Convolution Code Radix	convolution_code_radix	Binary
Convolution Code 0	convolution_code0	1001111
Convolution Code 1	convolution_code1	1101101
Convolution Code 2	convolution_code2	0000011
Convolution Code 3	convolution_code3	0000011
Convolution Code 4	convolution_code4	0000011
Convolution Code 5	convolution_code5	0000011
Convolution Code 6	convolution_code6	0000011
Dual Output	dual_output	False
TREADY	tready	True
ACLKEN	aclken	False

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 5].

System Generator for DSP

The Convolutional Encoder core is also available through Xilinx System Generator for DSP, a design tool that enables the use of the Simulink® model-based design environment for

FPGA design. The Convolutional Encoder core is one of the DSP building blocks provided in the Xilinx blockset for Simulink. The core can be found in the Xilinx Blockset in the Communication section. The block is called "Convolutional Encoder 9.0." See the *System Generator for DSP User Guide* (UG640) [Ref 11] for more information.

The controls in the System Generator GUI work identically to those in the Vivado Design Suite IDE, although the layout differs.

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite.

Required Constraints

This section is not applicable for this IP core.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

Simulation

For comprehensive information about Vivado simulation components, as well as information about using supported third party tools, see the *Vivado® Design Suite User Guide: Logic Simulation* (UG900) [Ref 7].

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado® Design Suite User Guide: Designing with IP* (UG896) [Ref 5].

Test Bench

This chapter contains information about the test bench provided in the Vivado® Design Suite.

Demonstration Test Bench

When the core is generated using the Vivado Design Suite, a demonstration test bench is created. This is a simple VHDL test bench that exercises the core.

The demonstration test bench source code is one VHDL file: `<component_name>/demo_tb/tb_<component_name>.vhd` in the Vivado IP catalog output directory. The source code is comprehensively commented.

Using the Demonstration Test Bench

The general steps for using the demonstration test bench are:

1. Compile the netlist and the demonstration test bench into the work library. See your simulator documentation for detailed information.
2. Simulate the demonstration test bench.
3. View the test bench signals in your simulator waveform viewer to see the operations of the test bench.

Demonstration Test Bench in Detail

The demonstration test bench performs the following tasks:

- Instantiates the core
- Generates a clock signal
- Drives the core input signals to demonstrate core features
- Checks that the core output signals follow AXI protocol rules (data values are not checked to keep the test bench simple)
- Provide signals showing the separate fields of AXI `tdata` signals

The demonstration test bench drives the core input signals to demonstrate the features and modes of operation of the core. The operations performed by the demonstration test bench are appropriate for the configuration of the generated core and are a subset of the following operations:

1. An initial phase where the core is initialized and no operations are performed.
2. Drive random data into the core to demonstrate convolutional encoding.
3. Demonstrate the use of AXI handshaking signals `tvalid` and `tready`.
4. Demonstrate the effect of asserting `aresetn`.
5. If `aclken` is present: demonstrate the effect of toggling `aclken`.

Customizing the Demonstration Test Bench

It is possible to modify the demonstration test bench to use different source data. Source data is generated randomly and driven into the core by the procedures of the `s_data_stimuli` process. The clock frequency of the core can be modified by changing the `CLOCK_PERIOD` constant.

Migrating and Upgrading

This appendix contains information about migrating a design from ISE® to the Vivado® Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

Migrating to the Vivado Design Suite

For information about migrating to the Vivado Design Suite, *see the ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 8].

Parameter Changes

The Vivado Design Suite IP update feature can be used to update an existing XCO file from v7.0 and v8.0 to v9.0, but the update mechanism alone does not create a core compatible with v7.1. Table A-1 shows the changes to XCI parameters from v7.0 and v8.0 to v9.0.

Table A-1: **Parameter Changes from v7.0 to v8.0 and 9.0**

Version 7.0	Version 8.0 and 9.0	Notes
component_name	component_name	Unchanged
constraint_length	constraint_length	Unchanged
convolution_code0	convolution_code0	Unchanged
convolution_code1	convolution_code1	Unchanged
convolution_code2	convolution_code2	Unchanged
convolution_code3	convolution_code3	Unchanged
convolution_code4	convolution_code4	Unchanged
convolution_code5	convolution_code5	Unchanged
convolution_code6	convolution_code6	Unchanged
	convolution_code_radix	Added in V8 for ease of use in entering convolutional codes.
dual_output	dual_output	Unchanged
input_rate	input_rate	Unchanged

Table A-1: **Parameter Changes from v7.0 to v8.0 and 9.0 *(Cont'd)***

Version 7.0	Version 8.0 and 9.0	Notes
output_rate	output_rate	Unchanged
puncture_code0	puncture_code0	Unchanged
puncture_code1	puncture_code1	Unchanged
	tready	Added in v8 for AXI control signal handshaking
ce	aclken	Renamed from ce to aclken for AXI standardization
fd_in		Replaced with AXI control signals
nd		Replaced with AXI control signals
rdy		Replaced with AXI control signals
rfd		Replaced with AXI control signals
rfdd		Replaced with AXI control signals
synchronous_clear		aresetn always present on core

Port Changes

Table A-2: **Port Changes from v7.0 to v8.0 and 9.0**

Version 7.0	Version 8.0 and 9.0	Notes
clk	aclk	Rename only
ce	aclken	Rename only
sclr	aresetn	Rename and change on sense (now active-Low). Must now be asserted for at least 2 cycles
fd_in		v8.0 does not require a pulse at the start of each block. s_axis_data_tvalid is used to detect this automatically.
nd	s_axis_data_tvalid	
rfd	s_axis_data_tready	
rffd		Input data stream can be sampled when s_axis_data_tready is asserted
rdy	m_axis_data_tvalid	

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

Parameter Changes

No changes.

Port Changes

No changes.

Other Changes

No changes.

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the Convolutional Encoder core, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

This product guide is the main document associated with the Convolutional Encoder. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx® Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use keywords such as:

- Product name
- Tool messages
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Record for the Convolutional Encoder

AR: 54496

Technical Support

Xilinx provides technical support in the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, navigate to the Xilinx Support web page.

- Additional files based on the specific issue might also be required. See the relevant sections in this debug guide for guidelines about which files to include with the WebCase.

Debug Tools

There are debug tools available to address Convolutional Encoder design issues.

Vivado Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx devices.

The Vivado logic analyzer is used with the logic debug IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 10].

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado lab tools are a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the Vivado lab tools for debugging the specific problems.

General Checks

Ensure that all the timing constraints for the core were properly incorporated from the example design and that all constraints were met during implementation.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the `LOCKED` port.
- If your outputs go to 0, check your licensing.

AXI4-Stream Interface Debug

If data is not being transmitted or received, check the following conditions:

- If the transmit `s_axis_data` is stuck Low, the core cannot send data.
- If the receive `m_axis_data` is stuck Low, the core is not receiving data.
- Check that the `ACLK` inputs are connected and toggling.
- Check that the AXI4-Stream waveforms are being followed Figure 3-3.
- Check core configuration.

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

References

These documents provide supplemental material useful with this product guide:

1. *Xilinx Vivado AXI Reference Guide* (UG1037)
2. *AMBA® AXI4-Stream Protocol Specification* (ARM IHI 0051A)
3. *LogiCORE IP Viterbi Decoder Project Guide* (PG027)
4. *Vivado® Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
5. *Vivado Design Suite User Guide: Designing with IP* (UG896)
6. *Vivado Design Suite User Guide: Getting Started* (UG910)
7. *Vivado Design Suite User Guide - Logic Simulation* (UG900)
8. *ISE® to Vivado Design Suite Migration Guide* (UG911)
9. *Vivado Design Suite User Guide: Designing with IP* (UG896)
10. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
11. *System Generator for DSP User Guide* (UG640)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/18/2015	9.0	UltraScale+ device support added.
04/02/2014	9.0	• Added link to resource utilization figures • Added User Parameter table (Table 4-1)
12/18/2013	9.0	• Revision number advanced to 9.0 to align with core version number. • Template updated. • Added UltraScale™ architecture support.
03/20/2013	2.0	• Updated for core v9.0, and for Vivado Design Suite-only support. • Major revisions made to Debugging Appendix.
01/18/2012	1.0	Initial Xilinx release for AXI interfaces; previous non-AXI version of the data sheet is DS248.

Please Read: Important Legal Notices