Product Obsolete/Under Obsolescence

Application Note: Virtex-II Series

XAPP671 (v1.1) January 7, 2005

High Speed Data Recovery Using Asynchronous Data Capture Techniques

Author: Catalin Baetoniu and Tze Yi Yeoh

Summary

This application note describes using asynchronous data capture techniques as a method for high-speed data recovery in Virtex™-II and Virtex-II Pro™ devices. The reference designs accompanying this application note show how data is recovered in an interface running at 622 Mb/s DDR with 0.3UI of jitter. The reference design also shows successful recovery of data even when the receive sampling clock is out of phase with the serial data — up to ±12 unit intervals (UI).

Introduction

In high-speed source-synchronous applications, clock and data recovery are essential. The most prevalent method of clock and data recovery using Xilinx devices is oversampling incoming data with multiple phases of the clock generated by the DCM. However, using the DCM as a means of data recovery is not always suitable at very high frequencies due to the DCM's inherent jitter. The additional jitter from the DCM causes a corresponding reduction in the data-valid window limiting the highest possible performance.

Asynchronous data capture techniques allow the user to perform data recovery without using the DCM, thus achieving higher performance. Using asynchronous data capture techniques, parallel bus interfaces (e.g., SPI4.2) are implemented in Virtex-II Pro devices running at up to one Gb/s DDR.

Reference Design Overview

This section provides key information about the reference designs accompanying this application note. The three reference designs provided in xapp671.zip show various asynchronous data capture techniques for implementation in Virtex-II and Virtex-II Pro devices. Table 1, Table 2, and Table 3 outline these reference designs.

Table 1: **Reference Design Specifics for des32.zip**

Design File	des32.zip
Device Implemented	XC2VP7-6 FF896
Device Family Supported	Virtex-II Pro family only
Input Specifications	• 622 Mb/s data transfer rate(1) • 0.3 UI maximum jitter • ±12 UI maximum accumulated phase transient • 622 MHz LVDS input reference clock • 32 independent channels
Resource Utilization	• 2305 slices • Four BUFGMUXs • 16 block RAMs
HDL Used	VHDL
Synthesis Tool	Synplify 7.2.2
Implementation Tool	Xilinx ISE v5.2i SP#3
Simulation Tool	Modelsim 5.7c

Notes:

1. There is no lower limit on input frequency. However, the absolute jitter value must not exceed the best case timing constraint as discussed in Best-Case Timing Constraint.

Table 2: **Reference Design Specifics for pbd_4chan_vlog.zip**

Design Files	pbd_4chan_vlog.zip
Device Implemented	XC2V1000-5-FG456
Device Family Supported	Virtex-II and Virtex-II Pro family
Input Specifications	• 622 Mb/s data transfer rate(1) • 0.3 UI maximum jitter • ±12 UI maximum accumulated phase transient • 311MHz LVDS input reference clock • Four independent channels
Resource Utilization	• 385 slices • Two BUFGMUXs • Two block RAMs
HDL Used	Verilog
Synthesis Tool	Synplify 7.2.2
Implementation Tool	Xilinx ISE v5.2i SP#3
Simulation Tool	Modelsim 5.7c

Notes:

1. There is no lower limit on input frequency. However, the absolute jitter value must not exceed the best case timing constraint as discussed in Best-Case Timing Constraint.

Table 3: **Reference Design Specifics for pbd_4chan_vhdl.zip**

Design Files	pbd_4chan_vhdl.zip
Device Implemented	XC2V1000-5-FG456
Device Family Supported	Virtex-II and Virtex-II Pro family
Input Specifications	• 622 Mb/s data transfer rate(1) • 0.3 UI maximum jitter • ±12 UI maximum accumulated phase transient • 311 MHz LVDS input reference clock • Four independent channels
Resource Utilization	• 385 slices • Two BUFGMUXs • Two block RAMs
HDL Used	VHDL
Synthesis Tool	Synplify 7.2.2
Implementation Tool	Xilinx ISE v5.2i SP#3
Simulation Tool	Modelsim 5.7c

Notes:

1. There is no lower limit on input frequency. However, the absolute jitter value must not exceed the best case timing constraint as discussed in Best-Case Timing Constraint.

Design Overview

The reference designs illustrate the concept of using asynchronous data capture to perform data recovery. The basic components in the reference designs include:

- Clock generation module
- Data sampling delay lines
- Data recovery state machine
- Output elastic buffer

Figure 1 illustrates the major components of a one channel reference design. The channels are replicated to produce the appropriate bus width.

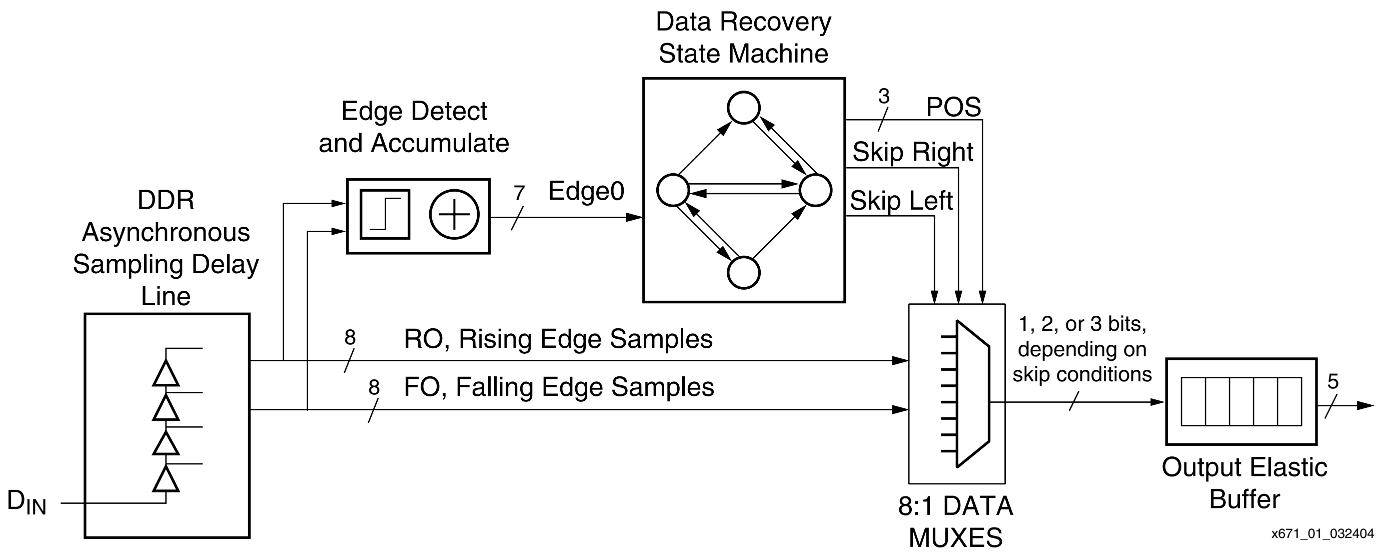
Data Recovery State Machine
Edge Detect and Accumulate
DDR Asynchronous Sampling Delay Line
7
Edge0
3
POS
Skip Right
Skip Left
8
RO, Rising Edge Samples
8
FO, Falling Edge Samples
1, 2, or 3 bits, depending on skip conditions
5
D_IN
8:1 DATA MUXES
Output Elastic Buffer
x671_01_032404

Figure 1: **Block Diagram for One Channel**

A single global sampling clock (both edges in the DDR case) samples delayed versions of incoming data using a multi-tap delay line. The data recovery state machine continuously selects the valid sample using edge information from the multi-tap delay line and forwards the correct sample to the output elastic buffer.

The `des32.zip` reference design is delivered as three VHDL files: `des.vhd`, `des32.vhd`, and `des32_tb.vhd`. The `des.vhd` file includes the logic for one channel: the data sampling delay lines, part of the data recovery state machine, and the output elastic buffer. The main file `des32.vhd` is the top design file. It contains the clock generation module, LVDS input buffers for all 32 channels, and instantiates 32 one-bit RPM tiles. A portion of the data recovery state machine common to each two channel pair (the block RAM) is also instantiated at this level. The `des32_tb.vhd` file is a VHDL testbench used for design verification. The `des32.zip` reference design is only for implementation in Virtex-II Pro devices because Virtex-II Pro devices have different CLB column spacing between block RAMs than Virtex-II devices. Furthermore, implementing this design in Virtex-II devices require, for timing reasons, placing a transparent latch between flip-flops crossing between CLK0 and CLK180 domain (see Using Virtex-II Devices).

The `pbd_4chan_vlog.zip` reference design is delivered as five Verilog files: `tap_ctrl_lut.v`, `tap_sm.v`, `data_recovery_2ch.v`, `top.v`, and `testb_lfsr.v`. The `tap_ctrl_lut.v` file includes the logic for one channel, the data sampling delay lines, and the output elastic buffer. The `tap_sm.v` file contains the data recovery state machine. Since a block RAM is shared between two channels, `data_recovery_2ch.v` is used to instantiate two instances of tap_ctrl_lut and one instance of tap_sm. Finally, the main file `top.v` instantiates two instances of data_recovery_2ch for a final tally of four channels. The Verilog testbench `testb_lfsr.v` file outputs an LFSR pattern for design verification. The `pbd_4chan_vlog.zip` reference design can be implemented in both Virtex-II and Virtex-II Pro devices.

The `pbd_4chan_vhdl.zip` reference design is the VHDL version of `pbd_4chan_vlog.zip` and is functionally identical to `pbd_4chan_vlog.zip`. There are three VHDL files: `tap_ctrl_lut.vhd`, `top.vhd`, and `testb_lfsr.vhd`. The `tap_ctrl_lut.vhd` file includes the logic for one channel (the data sampling delay lines, part of the data recovery state machine, and the output elastic buffer). The `top.vhd` file is the top design file. It has the clock generation module, LVDS input buffers for four channels and instantiates four tap_ctrl_lut components. The portion of the data recovery state machine common to each pair of two channels (e.g., block RAM) is also instantiated at this level. The `testb_lfsr.vhd` file is a VHDL testbench for design verification. Unlike `des32.zip`, this reference design can be implemented in Virtex-II devices as well as Virtex-II Pro devices.

References to specific signals used in the reference designs are made throughout this application note. In every case, references to signals in the `des32.zip` reference design are mentioned first with the corresponding signals in `pbd_4chan_verilog.zip` and `pbd_4chan_vhdl.zip` mentioned in parenthesis. For example, CLK311 (REC_CLK) where CLK311 is the 311 MHz clock signal in `des32.zip` and REC_CLK is the corresponding clock signal in `pbd_4chan_verilog.zip` and `pbd_4chan_vhdl.zip`.

Detailed Block Description

Clock Generation Module

The clock generation module in `des32.zip` uses a 622 MHz input reference to generate two 311 MHz clocks (0° and 180° phase) used for data sampling and one 155 MHz clock for driving the data recovery state machines. Another 124.4 MHz clock is generated as a test read clock for the output interface. It is very important for the first three clocks to be in phase with each other as data is moved between flip-flops in these three clock domains.

The phase of the 124.4 MHz clock is not critical since it is decoupled from the input data rate by the output elastic buffer. The frequency of the 124.4 MHz clock must be exactly 1/5 of the reference clock, otherwise overruns or underruns will occur in the elastic buffer. Therefore, the 124.4 MHz clock is generated by dividing the 622 MHz input reference clock by a factor of five.

In the `pbd_4chan_vlog`.zip and `pbd_4chan_vhdl.zip` reference designs, the input 311 MHz clock is buffered onto the global clock tree and fed directly into the system.

Generally, DCMs are used to produce clocks with a certain phase relationship between them. In the des32 reference design, the three clocks CLK311, nCLK311, and CLK155 are produced without using a DCM. This is accomplished by using controlled placement and routing of a few critical components and nets. It is possible because there is no phase requirement between the input reference clock and the three output clocks. The input 622 MHz clock drives a small number of flip-flops directly using local routing. This very high-speed clock does not use a BUFG and reaches all the flip-flops it is driving with minimal skew. The clock routing is guaranteed to remain unchanged by the use of directed routing constraints. The other four critical nets drive the four BUFGs. The skew between these clocks is also minimized by the use of directed routing constraints.

This implementation uses the minimum amount of resources (no DCMs and minimal numbers of BUFGs) and produces less skew and jitter on the clocks than when using a DCM.

Data Sampling Delay Lines

The input data for each channel is sampled asynchronously using 8-tap delay lines. There are two delay lines per channel: one for rising edge samples and one for falling edge samples. Each delay line takes eight LUTs. The input nets from the IOBs to the inputs of the two delay lines and the nets connecting the LUTs in each delay line are the only asynchronous portion of the design. The input net from the IOB must reach both delay lines with very little skew. By using directed routing constraints ensures the consistent routing of these nets for all the channels. Figure 2 shows a LUT based data sampling delay line.

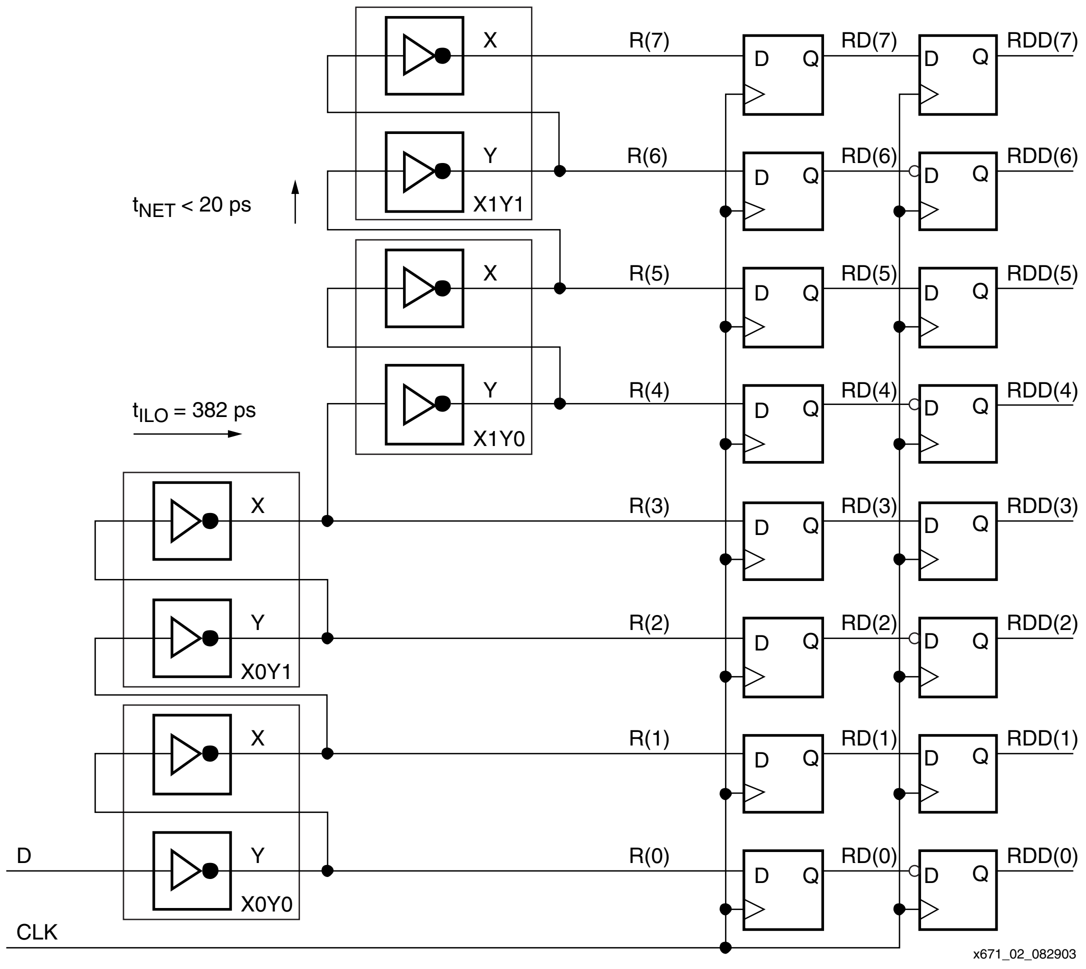

Figure 2: **LUT Based Data Sampling Delay Line**

An improper relative position of an IOB pair to RPM tile can lead to a different routing for this net. If the delays for the two loads of this net are not equal, the design will not work. This situation is indicated through directed routing constraint failures. It is important to check the place-and-route report after each implementation.

The timing analysis used to determine the tap delays and length of the delay chain are based on two basic timing constraints:

Best-Case Timing Constraint

The best-case timing constraint is the minimum value for the entire delay line. It must be longer than the amount of data jitter. This constraint determines the number of taps.

Worst-Case Timing Constraint

The maximum value for the tap delay must be shorter than the data valid time window (equivalent to the Nyquist sampling theorem for analog signals). The worst-case timing constraint determines the selection of the delay line elements.

Mathematically, the constraints are expressed as:

$$t_{JIT} < t_{TAPMIN} \times ((n - 2k) + 1) \qquad \text{Eq. 1}$$

and

$$t_{VAL} > k \times t_{TAPMAX} \qquad \text{Eq. 2}$$

In this example *n* is the number of taps, *k* is the minimum number of data samples in the data-valid time window, t_{TAPmin} and t_{TAPmax} are the required minimum and maximum tap delays, t_{JIT} is the amount of jitter, and t_{VAL} is the data valid time window.

A minimum and maximum tap delay value, t_{TAPmin} and t_{TAPmax} are the result of the derating factor causing a variation in the delay values used in the analysis. Derating factors are due to variations in process, voltage, and temperature (PVT) and are generally assumed to be approximately 40%. The timing analysis for this particular example is explored in Appendix A.

Figure 3 illustrates the two constraints.

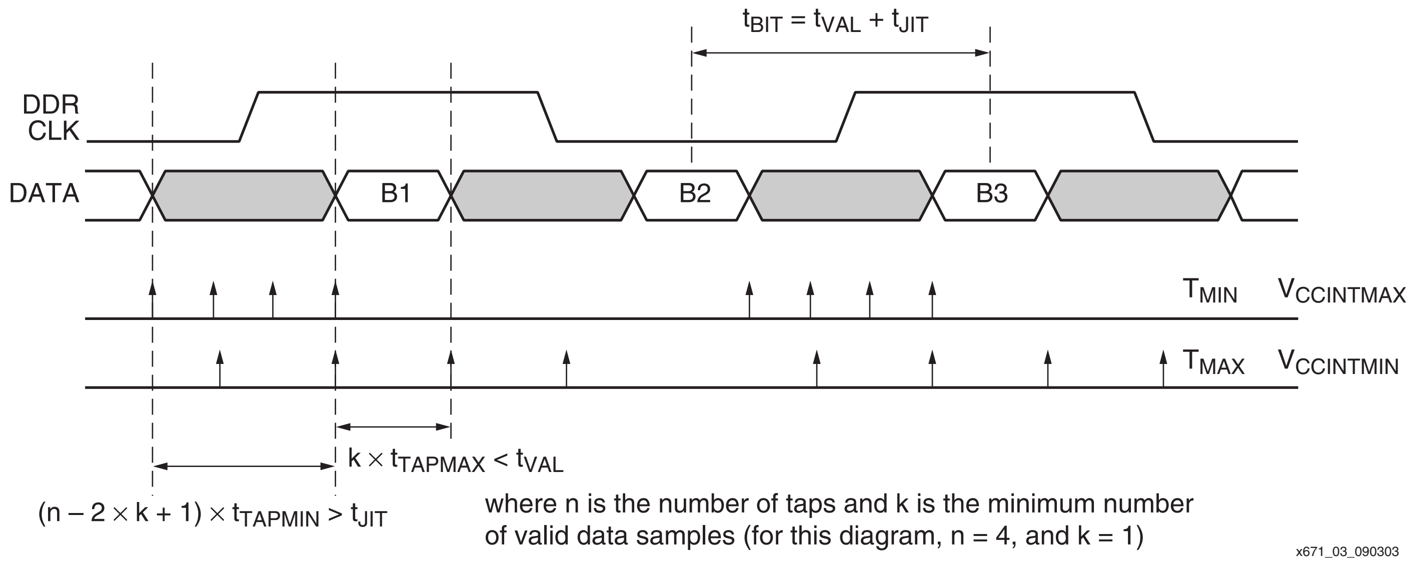
tBIT = tVAL + tJIT
DDR CLK
DATA
B1
B2
B3
TMIN
VCCINTMAX
TMAX
VCCINTMIN
k × tTAPMAX < tVAL
(n – 2 × k + 1) × tTAPMIN > tJIT
where n is the number of taps and k is the minimum number of valid data samples (for this diagram, n = 4, and k = 1)
x671_03_090303

Figure 3: **Timing Diagram**

For functional simulation, the delays through the LUTs are modeled using the t_{TAP} and the DERATING_FACTOR generic parameters.

Data Recovery State Machine

As data is sampled by the tap delay lines, the data recovery state machine processes these samples and ultimately produces valid output bits.

The first step consists of creating seven edge-detect signals by pairwise XOR-ing the eight delay line samples. The edge detects for the rising-edge and the falling-edge samples are then OR-ed together to produce iEDGE. In the process, the falling-edge signals are also moved into the CLK311 (REC_CLK) clock domain. Each time there is a data transition, either in the rising-edge samples or the falling-edge samples, at least one bit in iEDGE is set. The iEDGE signal is then moved into the CLK155 (REC_CLK/2) clock domain as the signal EDGE, and accumulated over a period determined by the control signal CE.

The accumulated edge signal aEDGE is the output of this module and is processed by the state machine. The state machine then produces a position signal, POS, to be fed back into `des.vhd` (tap_ctrl_lut). Two 8:1 multiplexers use the POS signal to select one of the rising edge and one of the falling edge samples as the valid data output.

The actual state machine is implemented in a dual-port block RAM, shared by two channels. A control signal, CE, controls the period of time when edges are accumulated. CE is generated for each channel. The period of CE is determined by the MAXCNT constant. The state machine uses the EDGE information provided by the delay lines to adjust POS. The current sampling point is updated every time CE is "1".

Two versions of the state machine are provided in the reference design: a behavioral and a structural version. A behavioral version is included where the transitions are described with *case* and *if* statements. In the structural version the state machine is implemented in a block RAM. While both could be simulated and synthesized, the first one is intended for simulation and the second one for synthesis.

The simulation version of the state machine is in a more readable format. This format is used for debugging and/or modifying the state machine behavior. While this version can be synthesized, it is not efficient since it would map into a large number of slices. For synthesis, use the block RAM since it is initialized with data producing exactly the same functionality as the behavioral version. This second version can also be used for simulation and/or synthesis. Changes to the state machine functionality are made in the behavioral section (maintaining the exact same code structure and indenting). The FixDes32.exe utility included in the reference design will re-annotate the attributes and the generics to update the BRAM initial values to reflect the new functionality (see Using FixDes32 to Modify State Machine Behavior). This method enables the implementation of any state machine behavior with the same number of inputs and outputs using the same low number of logic resources without affecting the device utilization, routing, and timing performance.

Output Elastic Buffer

The output elastic buffer converts the 1-bit 622 Mb/s data stream into a 5-bit 124.4 MHz output (for `des32.zip`) or 8-bit 77 MHz output (for `pbd_4chan_vlog.zip` and `pbd_4chan_vhdl.zip`). Each channel has its own 16 5-bit (or 8-bit) word buffer. Since the 622 Mb/s input data stream can be out of phase with the input reference clock by as much as ±12 UI, they are virtually asynchronous. For this reason, instead of the normal two bits for each CLK311 (REC_CLK) period, it is possible to have three bits/clock (if the clock is slower than the data) or just one bit/clock (when the clock is faster than the data).

The POS signal produced by the state machine is brought back into the CLK311 clock domain and pipelined accordingly to drive three 8:1 multiplexers producing the R8, F8, and M8 signals. R8 is the current valid rising edge sample, F8 is the current valid falling edge sample, and M8 a delayed version of R8 used for the case when three bits/clock are required. The skip conditions (one or three bits/clock instead of the normal two bits/clock) are indicated by the state machine using the SKIP_LEFT and SKIP_RIGHT signals. Based on the value of these signals either three bits (SKIP_RIGHT condition), two bits (normal condition), or just one bit (SKIP_LEFT condition) are shifted into the DATA5 (DATA8) shift register.

The elastic buffer itself has 16 5-bit (8-bit) locations. At reset, the read and write pointers point to the middle of the buffer. The initial state of the buffer is half full and the largest possible phase offset is accommodated before overrun or underrun occurs. In des32, the elastic buffer accommodates a maximum accumulated phase error of ±40 bits. In `pbd_4chan_vlog.zip` and `pbd_4chan_vhdl.zip` the maximum accumulated phase error increases to ±64 bits.

Modifying the Reference Design

The reference designs provided in xapp671.zip were designed to particular input specifications. This section describes how to modify the reference designs given different input specifications. Once the modifications are made, the timing in the provided constraints files will be different.

Tap Delay

The LUT based delay line routing can be modified for a different tap delay based on the input frequency and jitter characteristics of the application. The two methods outlined use an interleaved tap delay line or a dual speed LUT-based delay line.

Interleaved Tap Delay Line

For applications where t_{ILO} is as large as a tap delay, use an interleaved solution. In this example, the interconnect delay shown at the bottom of the diagram is chosen to approximate ½ of a LUT delay, t_{ILO}. When the LUTs are interleaved as shown in Figure 4, the tap delay t_{TAP} equals ½ t_{ILO}.

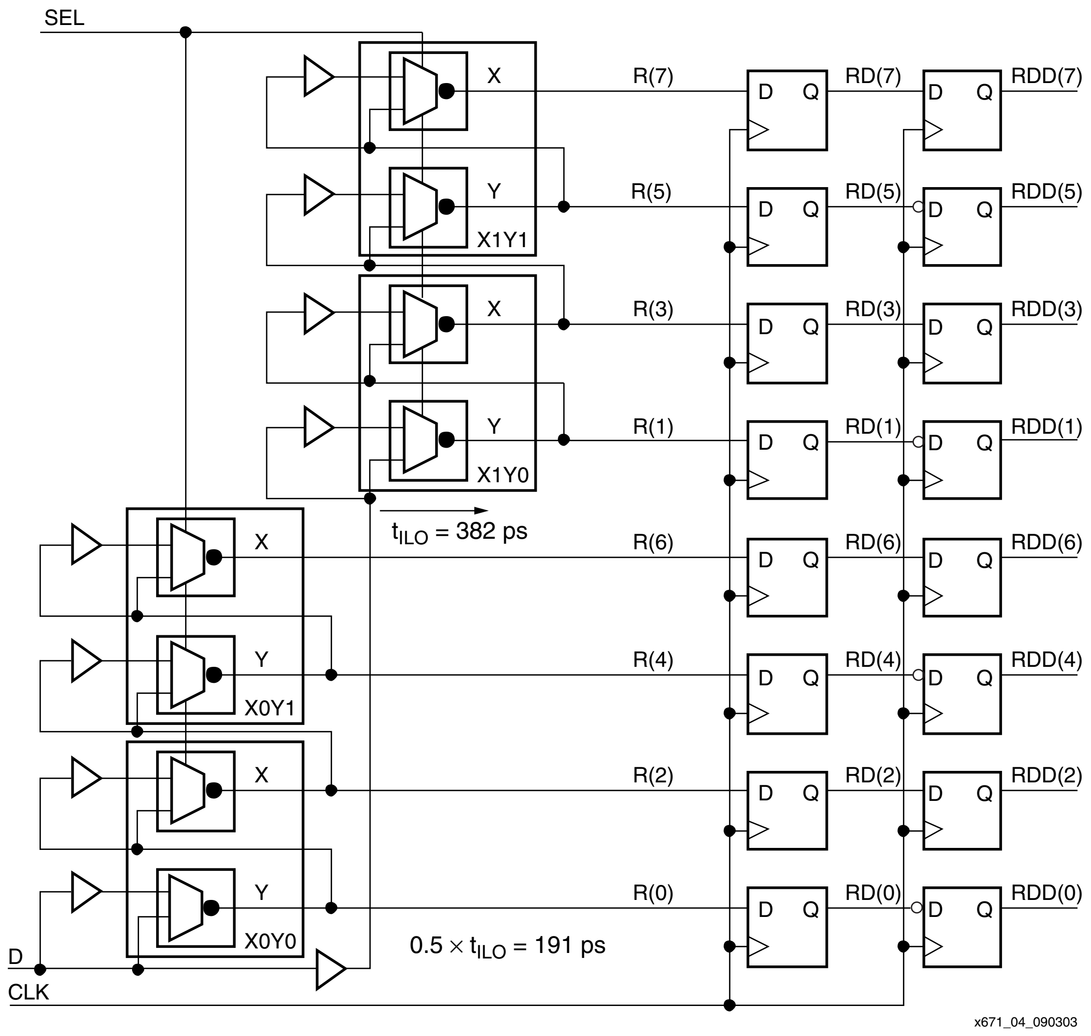

Figure 4: **Interleaved LUT-Based Delay Line**

Dual-Speed LUT-Based Delay Line

In the example in Figure 5, the interconnect delay, t_{NET}, is added on to the LUT delay, t_{ILO}, to provide an alternative longer tap delay, t_{TAP}. The SEL input is used to select between fast ($t_{TAP} = t_{ILO}$) and slow ($t_{TAP} = t_{ILO} + t_{NET}$) delay modes. The DEL input is used to insert an extra delay at the input of the delay chain and shift all eight sampling points to a different bit area.

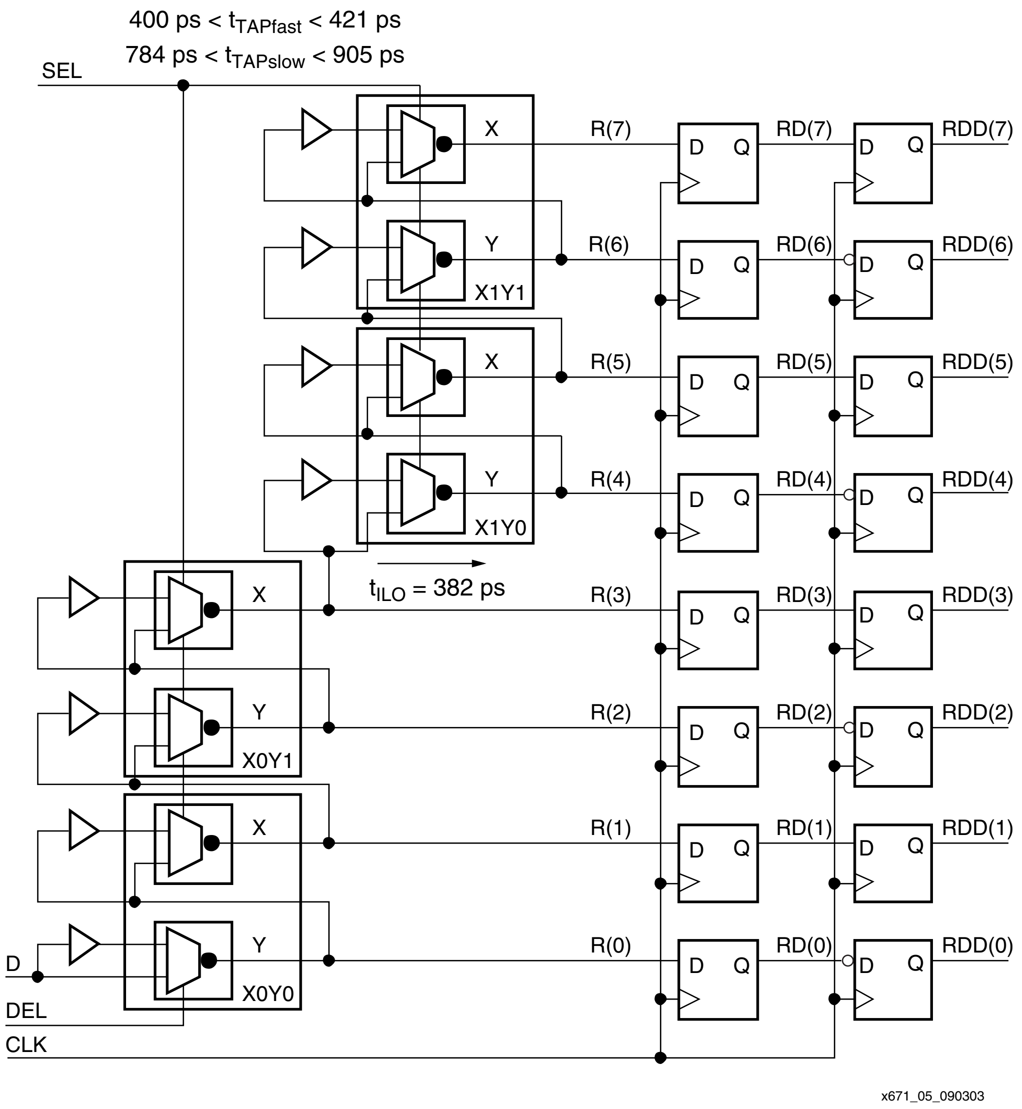

Figure 5: **Dual-Speed LUT-Based Delay Line**

Channel Width

The des32 reference design currently supports up to a 32-bit wide interface. To support an interface smaller than 32 bits wide, change the bus width of the input data ports, D_{Ip} and D_{In} (lines 31 and 32 in `des32.vhd`). Since the input data ports are location constrained in the UCF file, modify the UCF file to reflect the number of input channels used.

In pbd_4chan_vlog and pbd_4chan_vhdl, instantiate more or less channels as required. Also, since the input data ports are location constrained in the UCF file, modify the UCF file to reflect the number of input channels used.

Edge Detect State Machine Behavior

For the most part, the edge-detect state machine is accurate for a wide variety of input conditions. However, the behavior of the state machine could be modified in the following ways:

Modify MAXCNT

The MAXCNT constant, currently set to 16 in `des32.vhd`, determines the number of clock periods accumulating edge information before the state machine changes the current sampling point. The actual number of 622 Mb/s bits accumulated is 4 × MAXCNT since the state machine runs at 155 MHz. In a high-jitter environment, a lower MAXCNT constant allows the state machine to adapt quickly to edge changes.

Edge-Detect Algorithm

The current threshold for edge detection is one event per MAXCNT period. A higher threshold for edge detection enables the state machine to make a more intelligent decision on where to position the POS pointer. For example, a higher threshold level makes the state machine immune to a single noise spike. In this case, the current sampling is not affected by misplaced data edges and only responds to a systematic phase change in the data stream. If a higher threshold is required for edge detection, modify the code to replace the simple one-bit aEDGE accumulator with a set of seven histogram-like counters. They accumulate the number of edge events occurring over a MAXCNT interval. The EDGEO output signal is the output of the comparison of the seven counters with a given threshold. This provides flexibility to the data recovery state machine at a higher device utilization cost (at 155 MHz the speed hit is not large).

Behavior of the Edge-Detect State Machine

Some applications can require changes to the behavior of the state machine. The BRAM INIT attributes and generic parameters should not be directly modified. Instead, modify the behavioral description of the state machine to develop and test changes, and then use the FixDes32 tool to back-annotate the changes into INIT strings.

Using FixDes32 to Modify State Machine Behavior

FixDes32 is a utility written in Delphi to convert the behavioral description of the state machine in `des32.vhd` into INIT attributes and generic parameters for implementation in a block RAM. To run this utility, the file containing the state machine must be in the same directory as the `FixDes32.exe` application and must be named `des32.vhd`. The code structure and indenting must be maintained when modifying the behavioral state machine RTL code. The behavioral section of the state machine code (from the '$IFDEF' statement to '$ELSE' statement) must be commented out with the '--;' keyword. The FixDes32 utility uses this keyword to determine the behavioral state machine. Once the state machine file is prepared, run FixDes32.exe (double click on `FixDes32.exe` in Windows) and the tool automatically back-annotates the INIT strings with the changes.

Backend Interface

In the `des32.vhd` RTL code provided in the reference design, the 5-bit output ports of the output elastic buffers are OR-ed together and connected to a TEST output port to prevent the logic from being minimized by the synthesis tool. Modify the read interface of the elastic buffers to create the desired interface to the backend portion of the design. Similarly, modify the 8-bit output ports in pbd_4chan_vlog and pbd_4chan_vhdl as appropriate.

Migrating to a Different Pinout/Package/Device

It is relatively easy to retarget the design for a different pinout/package/device by following a few basic rules:

- In Virtex-II Pro devices, avoid using CLB rows straddling the PPC405 and the top and bottom four CLB rows (because of the MGTs the routing resources are slightly different).

- Always pair the tiles, each pair shares a block RAM. It is possible to locate these block RAMs in the two block RAM columns inside the RPM tile area to create a very compact design without wasted resources.
- In des32, there are left and right versions of the channel tile; the generic parameter LEFT (`des.vhd`, line 26) determines the floorplanning and the directed routing constraints.
- The location of the IOBs, the rloc_origin attribute of the RPM, and the LEFT generic parameter must be consistent. Use the FPGA Editor and the place and route (PAR) report to check the final result. Directed routing errors point to placement inconsistencies.

Using Virtex-II Devices

The pbd_4chan_vlog and pbd_4chan_vhdl reference designs are both implemented in a Virtex-II device (XC2V1000-5-FG456). There is a timing budget issue when the data path is moving from the CLK0 domain to CLK180 domain at 311 MHz. In order to meet timing, a transparent latch is inserted in the path crossing the two clock domains (lines 208-215 in `tap_ctrl_lut.v`). This latch adds a half clock cycle into the timing budget. The path through the latch is constrained appropriately in the UCF file (lines 5-8 in `top.ucf`). To guarantee placement and routing of the design, it must be implemented in a XC2V1000 or larger device.

Testbench

In des32, the testbench file is `des32_tb.vhd`. The file shows how to instantiate the core and how to simulate the design for jitter, skew, and different timing derating conditions. The data rate is determined by t_{HCLK} (half of the transmit clock period). The t_{JMAX} parameter controls the amount of jitter added to each data transition with respect to the reference transmit clock. While the real life jitter distribution is Gaussian, a uniform distribution is used in the testbench to increase the probability of extreme timing conditions. Finally, t_{DELTA} is a parameter controlling the simulation of skew and wander conditions. For every TxCLK period, a value of t_{DELTA} is added to or subtracted from the transmit clock period. Every 10 µs the behavior is changed between addition and subtraction. The net effect is a continuous wander condition between the transmit data and the receive clock. The amplitude of this wander is determined by the t_{DELTA} constant and the 10 µs period controlling the alternation between fast and slow transmit clock modes.

It is also possible to model the timing behavior of the delay lines with the t_{TAP} and the DERATING_FACTOR generic parameters. The actual tap delay in simulation is $t_{TAP} \times$ DERATING_FACTOR.

A simple 10-bit pattern is used for testing. It is easy to check for this pattern at the 5-bit output of the interface but more complex tests using CRC data streams can be developed.

In pbd_4chan_vlog, the testbench file is `testb_lfsr.v`. The testbench generates an LFSR test pattern and uses that pattern to detect errors. This testbench does not simulate jitter, skew, and phase transient accumulation.

In pbd_4chan_vhd, the testbench file is `testb_lfsr.vhd`. The testbench generates an LFSR test pattern and uses that pattern to detect errors. This testbench can also simulate jitter, skew, and phase transient accumulation similar to des32.

Conclusion

In many high-speed parallel bus interfaces, data recovery techniques using asynchronous data capture is quickly becoming a necessity. Asynchronous data capture is an alternative method of performing data recovery without the use of a DCM. This application note has presented overview of asynchronous data capture techniques. The techniques are demonstrated in two examples using a 32-channel link and a 4-channel link running at 622 Mb/s implemented in a Virtex-II Pro device, and a 4-channel link running at 622 Mb/s implemented in a Virtex-II device.

Appendix A

Timing Analysis

Two basic timing constraints determine the tap delays and length of the delay chain: Best-Case Timing Constraint and Worst-Case Timing Constraint. Figure 3, page 6 illustrates the two constraints.

This example has an input specification data rate of 622 Mb/s, data to clock jitter of 0.30 UI, and the maximum accumulated phase transient at the receiver of ±12 UI. Based on the information given, the total bit time is 1607 ps, the data valid window is 1125 ps, and jitter time is 482 ps. The example uses a LUT delay line with t_{ILO} = 320 ps and t_{NET} = 75 ps. Solving for t_{TAPMAX}:

$t_{TAPMAX} = (t_{ILO} + t_{NET})/2$

$t_{TAPMIN} = t_{TAPMAX} \times 40\%$ (using a 40% derating factor)

gives t_{TAPMAX} = 395 ps and t_{TAPMIN} = 158 ps

Starting with the worst-case conditions, the parameters n and k are calculated based on t_{TAPMAX}.

Since the edge-detect state machine requires at least two valid data samples, k = 2. Evaluating Equation 2 (page 5):

$t_{DATAMIN} > t_{TAPMAX} \times 2$

1125 ps > 790 ps (335ps margin, at least two valid data samples)

Therefore, the worst-case timing constraint is met.

The best-case timing constraint is evaluated next. Using an 8-tap delay line:

$t_{JITMAX} < t_{TAPMIN} \times 5$

482 ps < 790 ps

The jitter condition is also met.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/24/04	1.0	Xilinx Confidential version.
01/07/05	1.1	Initial Xilinx release.