



XAPP1274 (v1.2) September 6, 2019

# Native High-Speed I/O Interfaces

## Summary

This application note and associated reference designs are intended to show how to construct high-speed I/O interfaces using the native mode I/O in UltraScale™ and UltraScale+™ devices.

In addition to the information in Chapter 2 of the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [Ref 1], this application note provides a detailed description of how to build high-speed I/O interfaces. This application note describes the UltraScale architecture and building interfaces using the [High Speed SelectIO Wizard](#).

You can download the [Reference Design Files](#) for this application note from the Xilinx® website. For detailed information about the design files, see [Reference Design \(IoWizard\\_RxTxInterface\)](#) and [Reference Design \(IoWizard\\_AsyncDataCapture\)](#).

## Introduction

The [Native I/O Details](#) section provides a detailed insight in the use of native mode for I/O primitives and explains how these should be used and connected.

The [Reference Design \(IoWizard\\_RxTxInterface\)](#) and [Reference Design \(IoWizard\\_AsyncDataCapture\)](#) sections provide design files using the [High Speed SelectIO Wizard](#) for various uses of high-speed SelectIO™ interfaces. These sections can be used as quick-start guidelines.

- Create [Source Synchronous Interfaces](#) using the High Speed SelectIO Wizard (IoWizard\_RxTxInterface). This section describes how to construct source-synchronous 8-channel transmit and receive interfaces. A loopback design uses these interfaces for the KCU105/VCU108 development board equipped with a loopback FMC board.
  - [High Speed SelectIO Wizard Clocking Structures](#)
  - [High Speed SelectIO Wizard Reset Sequence](#): Explains and gives examples of the reset sequence.
- Create [Asynchronous Data Capture Interfaces](#) using the High Speed SelectIO Wizard (IoWizard\_AsyncDataCapture). This section describes how to construct asynchronous data capture 2-channel transmit and receive interfaces. A loopback design uses these interfaces for the KCU105/VCU108 development board equipped with a loopback FMC board.
- [Bitslip](#)

## Features

In conjunction with the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [Ref 1], this application note further explains the native high-speed I/O primitives available in UltraScale devices. Reference designs covering a broad range of high-speed applications are also provided. These reference designs are designed to work on Xilinx development boards to jump-start your system design.

## Native I/O Details

In addition to the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [Ref 1], this section provides additional information about the use and operation of the native mode I/O primitives. The primary data-handling interfaces and their derivatives are described in this section.

Native I/O primitives (RX\_BITSLICE, TX\_BITSLICE, RXTX\_BITSLICE, TX\_BITSLICE\_TRI, and BITSLICE\_CONTROL) are available in a nibble, see [Figure 45](#) in [Appendix A](#). A nibble can contain up to six (lower nibble) or seven (upper nibble) BITSLICES connected to a BITSLICE\_CONTROL primitive. The BITSLICE at position zero (BITSLICE0) in each nibble can be used as a data, clock, or strobe input (see [Figure 46](#) in [Appendix A](#)). Each nibble can create a different receiver and/or transmitter interface in one byte.

The BITSLICES and BITSLICE\_CONTROL functionality are managed using attributes.

## Attributes

When used together, the native mode I/O primitives used for various interfaces to and from the FPGA connections are defined/configured using a set of attributes. [Table 1](#) lists a set of common attributes and required settings for various types of interfaces.

**Table 1: Common Attributes for Data Interfaces**

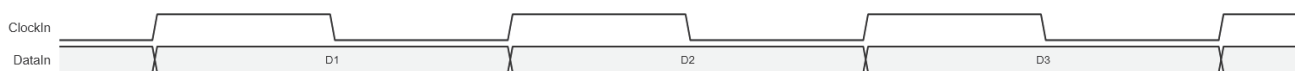
Attribute	Required Settings
<b>TX_BITSLICE or RXTX_BITSLICE</b>	
TX_DATA_WIDTH	Can be 4 or 8
TBYTE_CTL	TBYTE_IN, T
TX_DELAY_TYPE	FIXED, VARIABLE, or VAR_LOAD
TX_DELAY_VALUE	Defines the output delay time, zero or non-zero value.
TX_REFCLK_FREQUENCY	When DELAY_VALUE = 0, leave at default. Else, reference the frequency of the PLL.CLKOUTPHY clock.
TX_DELAY_FORMAT	TIME
TX_UPDATE_MODE	ASYN
ENABLE_PRE_EMPHASIS	FALSE

Table 1: Common Attributes for Data Interfaces (Cont'd)

Attribute	Required Settings
<b>RX_BITSLICE or RXTX_BITSLICE</b>	
RX_DATA_WIDTH	Can be 4 or 8
RX_DATA_TYPE	DATA for BITSlices only receiving data. DATA_AND_CLOCK for BITSlice_0 when receiving a clock.
RX_DELAY_TYPE	FIXED, VARIABLE, or VAR_LOAD
RX_DELAY_VALUE	Defines the output delay time, zero or non-zero value.
RX_DELAY_FORMAT	TIME
RX_REFCLK_FREQUENCY	When DELAY_VALUE = 0, leave at default. Else, reference the frequency of the PLL.CLKOUTPHY clock.
RX_UPDATE_MODE	ASYNCH
FIFO_SYNC_MODE	FALSE
<b>BITSLICE_CONTROL</b>	
RX_GATING	DISABLE
TX_GATING	ENABLE
INV_RXCLK	FALSE, TRUE depending on receiving clock and data timing relationship.
SERIAL_MODE	FALSE, TRUE
LOOPBACK	FALSE

The following sections include interface timing diagrams (Figure 1–Figure 4) and list the attributes (Table 2–Table 5) used to configure dedicated interfaces using native I/O primitives to generate or receive data. This automatic handling of clock and data using the native I/O primitives uses a built-in self-calibrating (BISC) controller that is embedded in each BITSlice\_CONTROL primitive. See the [Built-in Self-Calibration](#) section for further information.

### TX/RX SDR Edge-aligned Interface



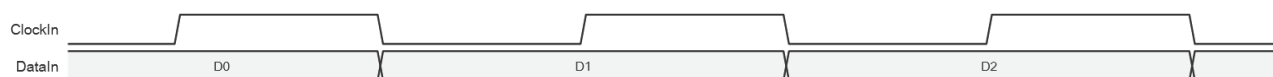
X15799-101316

Figure 1: TX/RX SDR Edge-aligned Interface

Table 2: Attribute Settings for SDR Edge-aligned Interfaces

Attribute	Required Settings
<b>TX_BITSLICE or RXTX_BITSLICE</b>	
TX_OUTPUT_PHASE_90	FALSE (for both data and clock)
<b>BITSLICE_CONTROL</b>	
INV_RXCLK	TRUE
RX_CLK_PHASE_P	SHIFT_0
RX_CLK_PHASE_N	SHIFT_0

## TX/RX SDR Center-aligned Interface



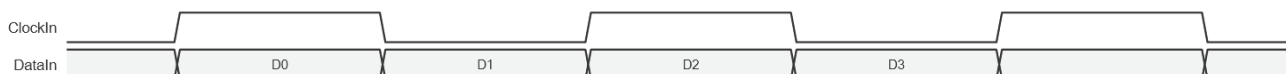
X15798-101316

Figure 2: TX/RX SDR Center-aligned Interface

Table 3: Attribute Settings for SDR Center-aligned Interfaces

Attribute	Required Settings
<b>TX_BITSLICE or RXTX_BITSLICE</b>	
TX_OUTPUT_PHASE_90	FALSE (for both data and clock)
<b>BITSLICE_CONTROL</b>	
INV_RXCLK	FALSE
RX_CLK_PHASE_P	SHIFT_0
RX_CLK_PHASE_N	SHIFT_0

## TX/RX DDR Edge-aligned Interface



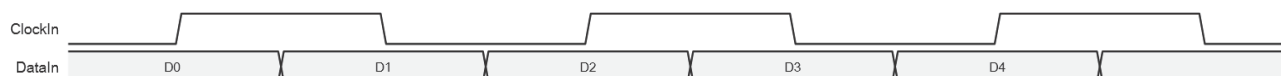
X15797-101316

Figure 3: TX/RX DDR Edge-aligned Interface

Table 4: Attribute Settings for DDR Edge-aligned Interfaces

Attribute	Required Settings
<b>TX_BITSLICE or RXTX_BITSLICE</b>	
TX_OUTPUT_PHASE_90	FALSE (for both data and clock)
<b>BITSLICE_CONTROL</b>	
INV_RXCLK	FALSE
RX_CLK_PHASE_P	SHIFT_90
RX_CLK_PHASE_N	SHIFT_90

## TX/RX DDR Center-aligned Interface



X15796-101316

Figure 4: TX/RX DDR Center-aligned Interface

Table 5: Attribute Settings for DDR Center-aligned Interfaces

Attribute	Required Settings
<b>TX_BITSLICE or RXTX_BITSLICE</b>	
TX_OUTPUT_PHASE_90	FALSE for data and TRUE for the interface clock.
<b>BITSLICE_CONTROL</b>	
INV_RXCLK	FALSE
RX_CLK_PHASE_P	SHIFT_0
RX_CLK_PHASE_N	SHIFT_0

## Clocking Methods

As described in [Appendix A](#), BITSlices must be combined with a BITSlice\_CONTROL primitive to form a nibble. The BITSlice at position zero of each nibble can be used as data, clock, or strobe. If used as clock or strobe clock, these inputs have different functionality over the full I/O bank.

The distinctions between the clock and strobe are listed.

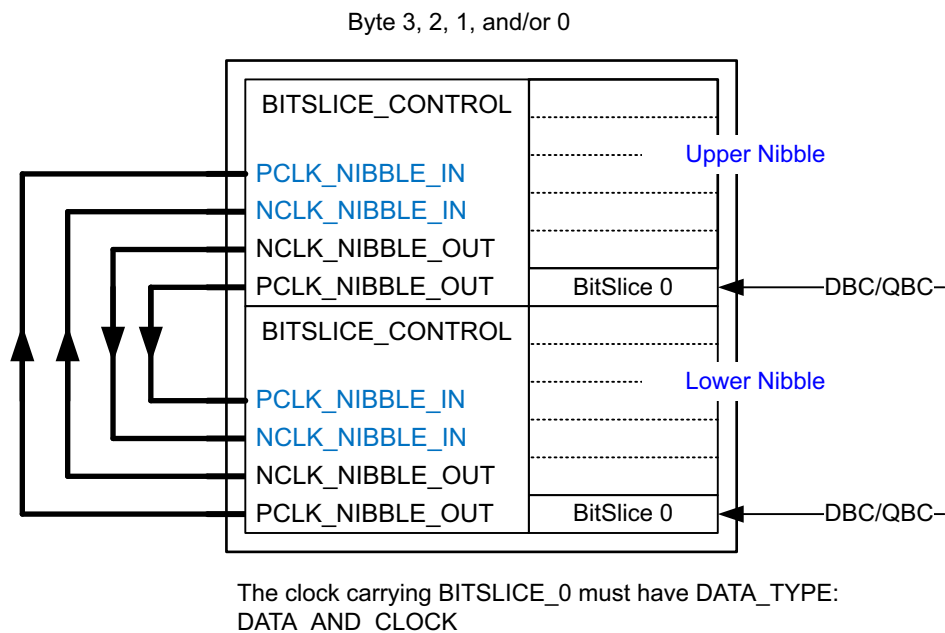
- **Clock:** A regular, repetitive cycling, input signal that can be connected to the BITSlice at position zero and at the same time be used as a PLL/MMCM input. When used for the PLL/MMCM, additional restrictions apply to ensure that the PLL remains locked during operation.
- **Strobe:** Can be a regular, repetitive cycling signal but can also be a repetitive intermittent input signal. Typically, this signal is only connected to the BITSlice at position zero.

The different clock input functions are listed:

- **Dedicated byte clock (DBC):** This is the function of the BITSlice\_0 strobe inputs of the upper and lower nibble in byte\_0 and byte\_3 in an I/O bank. When used, the strobe input can use clock resources between both nibbles in a byte. As further outlined in [Inter-nibble Clocking](#), the clock network in the BITSlices performing the DBC function is called the inter-nibble clock.
- **Quad-byte clock (QBC):** The BITSlice\_0 clock inputs of the upper and lower nibble in byte\_1 and byte\_2 in an I/O bank. When used, this clock input can clock resources in all bytes of an I/O bank. As further outlined in [Inter-byte Clocking](#), the clock network in the BITSlices performing the QBC function is called the inter-byte clock.
- A QBC clock input can also function as a DBC clock.
- **Global clock (GC):** Some of the QBC clock inputs can be used as global clock inputs. These inputs have a dedicated routing path to the MMCM/PLL clock inputs in the I/O bank. When a clock input is used as a global clock input, it can also have a QBC function.

## Inter-nibble Clocking

Inter-nibble clocking propagates clocks applied to DBC or QBC pins between nibbles in the same byte. Each nibble has its own BITSlice\_CONTROL with two input pins, PCLK\_NIBBLE\_IN/NCLK\_NIBBLE\_IN, and two output pins, PCLK\_NIBBLE\_OUT/NCLK\_NIBBLE\_OUT. A clock input in one nibble clocks the data inputs in the other nibble. As shown in Figure 5, both nibbles forming a byte connect the P/NCLK\_NIBBLE\_IN pins of one nibble to the P/NCLK\_NIBBLE\_OUT of the other nibble.



X15793-101316

Figure 5: Inter-nibble Connections in a Byte

The recommended steps for inter-nibble clocking are listed.

- As soon as two nibbles in a byte are used, connect the PCLK\_NIBBLE\_OUT of one nibble to the PCLK\_NIBBLE\_IN of the other nibble. Do the same for NCLK\_NIBBLE\_OUT and NCLK\_NIBBLE\_IN.
- The use of the EN\_OTHER\_nCLK attributes are mutually exclusive. When one pair in a nibble is set to TRUE, the pair in the other nibble MUST be set to FALSE. In some use cases the EN\_OTHER\_nCLK can be {TRUE, FALSE} for one nibble and {FALSE, TRUE} for the other nibble.
- When the upper-nibble BITSlice\_0 is set to either CLOCK or DATA\_AND\_CLOCK, set the EN\_OTHER\_PCLK and EN\_OTHER\_NCLK attributes of the other nibble to TRUE. The EN\_OTHER\_PCLK and EN\_OTHER\_NCLK attributes for the nibble carrying the clock input is set to FALSE. See the previous use cases for exception to this step. When both nibbles are used, always connect the inputs and the outputs of both nibbles together to enable clocking from one to another using attribute settings.

The following example lists the attributes when the upper nibble (BITSlice\_0) is set to CLOCK.

**Upper Nibble Attributes**

EN\_OTHER\_P CLK: FALSE

EN\_OTHER\_N CLK: FALSE

**Lower Nibble Attributes**

EN\_OTHER\_P CLK: TRUE

EN\_OTHER\_N CLK: TRUE

The following example lists the attributes when the lower nibble (BITSlice\_0) is set to CLOCK.

**Upper Nibble Attributes**

EN\_OTHER\_P CLK: TRUE

EN\_OTHER\_N CLK: TRUE

**Lower Nibble Attributes**

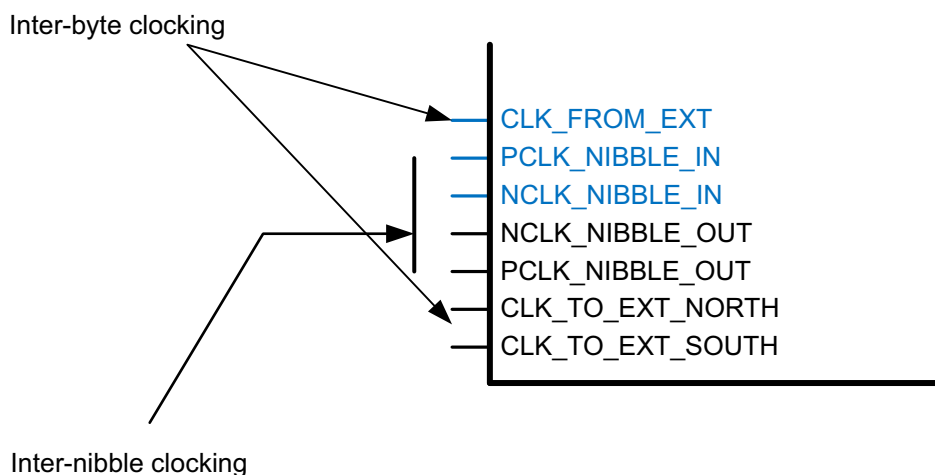
EN\_OTHER\_P CLK: FALSE

EN\_OTHER\_N CLK: FALSE

## Inter-byte Clocking

Inter-byte clocking moves the clock arriving on the QBC pin across the bytes in a bank. The recommended steps for inter-byte clocking are listed.

- The four clock or strobe pairs for the two middle bytes can drive all four byte groups. These pins are called the quad-byte clock (QBC).
- The QBC pin always shares the bit 0 of a nibble.
- The inter-byte clock connections (shown in [Figure 6](#)) are managed using the BITSlice\_CONTROL pins and the functionality is enabled using attributes.
- The attributes EN\_CLK\_TO\_EXT\_NORTH and EN\_CLK\_TO\_EXT\_SOUTH enable clock routing from one BITSlice\_CONTROL.CLK\_TO\_EXT\_xxxxx output to another BITSlice\_CONTROL.CLK\_FROM\_EXT input.



X15794-101316

Figure 6: BITSlice\_CONTROL Inter-byte Connections



**TIP:** When the BITSlice\_CONTROL pin CLK\_FROM\_EXT is not used, it must be tied High.



- 
- The diagram illustrates the QBC (Quad Bus Controller) interface, showing the timing of signals and the internal structure of the QBC blocks.
- Legend:**
- BLUE:** Lower nibble is clock origin.
  - BLACK:** Upper nibble is clock origin.
- Timing Diagram (Left):**
- The timing diagram shows the relationship between the QBC Byte 1 and QBC Byte 2 signals. The signals are represented by black and blue lines. The blue lines indicate the lower nibble (clock origin), and the black lines indicate the upper nibble (clock origin). The diagram shows the signals for QBC Byte 1 and QBC Byte 2, with the blue lines indicating the lower nibble and the black lines indicating the upper nibble.
- QBC Block Diagram (Right):**
- The QBC block diagram shows the internal structure of the QBC blocks, organized into four bytes (Byte 3, Byte 2, Byte 1, and Byte 0). Each byte contains two nibbles (Upper Nibble and Lower Nibble) and a BitSlice 0. The signals are connected to the QBC blocks as follows:
- Byte 3:** CLK\_TO\_EXT\_NORTH, CLK\_FROM\_EXT, CLK\_TO\_EXT\_SOUTH, BitSlice 0.
  - Byte 2:** CLK\_TO\_EXT\_NORTH, CLK\_FROM\_EXT, CLK\_TO\_EXT\_SOUTH, BitSlice 0.
  - Byte 1:** CLK\_TO\_EXT\_NORTH, CLK\_FROM\_EXT, CLK\_TO\_EXT\_SOUTH, BitSlice 0.
  - Byte 0:** CLK\_TO\_EXT\_NORTH, CLK\_FROM\_EXT, CLK\_TO\_EXT\_SOUTH, BitSlice 0.
- The QBC blocks are connected to the QBC- and QBC/GC- signals, which are used to control the QBC interface.

X15795-101316

## I/O or Global Clock

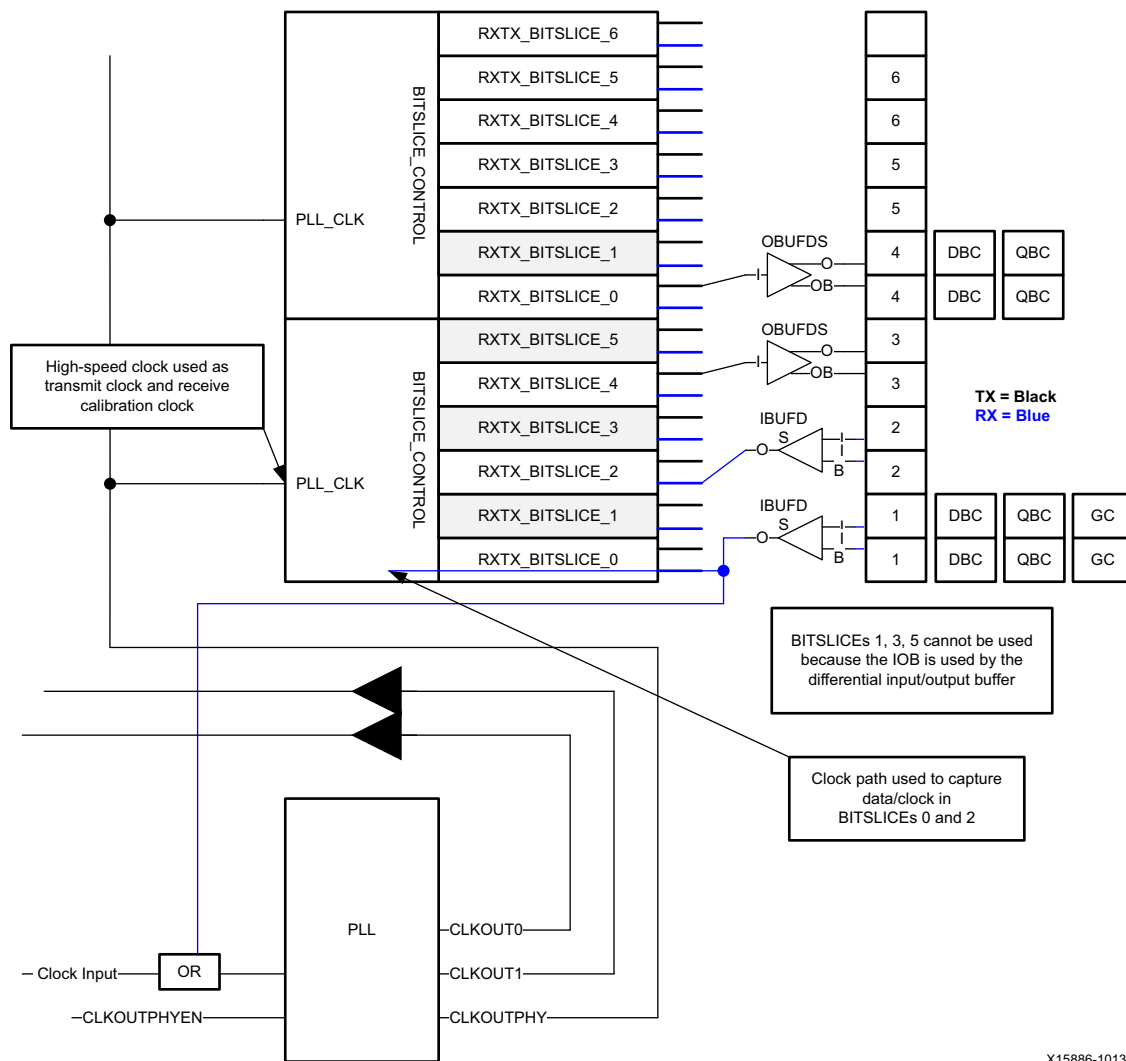
Each BITSlice\_CONTROL has two clock inputs, a REFCLK and a PLL\_CLK. These clock inputs are mutually exclusive, either one can be used, but not both.

- The REFCLK must be used when the clock source for the BITSlice\_CONTROL is from an MMCM. REFCLK is only supported for RX\_BITSlice.
- The PLL\_CLK must be used when the clock source for the BITSlice\_CONTROL is from a PLL. This is the recommended clocking mode when using native I/O mode.

Using the REFCLK from an MMCM uses clock buffers and routes the clocks over normal clock nets. For example, the maximum clock rate of an MMCM (see the *Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS892) [Ref 2]) is 850 MHz on a -3 speed grade device, which translates into a maximum source-synchronous receiver and transmitter data rate of 850 Mb/s.

The PLL\_CLK from a PLL uses dedicated clock routes between the PLL and the BITSlice\_CONTROL.PLL\_CLK pins. A PLL has a dedicated clock output for the BITSlice\_CONTROL capable of rates up to 2670 MHz (see the *Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS892) [Ref 2]). The PLL\_CLK clock path has better core jitter isolation than the MMCM clock.

Figure 8 shows an example clocking setup for RX\_BITSLICES and TX\_BITSLICES.



X15886-101316

Figure 8: Example Global Clocking Setup

## Receivers

When a data source provides data and a clock or strobe, that clock can be used to capture the delivered data (SERIAL\_MODE = FALSE). The BITSlice\_CONTROL (BISC) makes sure the received clock is correctly edge or center aligned with respect to the data and is kept that way during operation (VT tracking). This REFCLK or PLL\_CLK is used by the BISC controller to calibrate the delay lines, adjust the clock vs. data, and perform VT tracking. The frequency of the REFCLK or PLL\_CLK clock connected to the REFCLK or PLL\_CLK input must equal the received data rate.

The clock provided with the data can also be used to feed the PLL when it is connected to a QBC/GC input pin. The PLL must be fed from an interconnect logic clock source when the clock is connected to a QBC or DBC pin (non-GC). The PLL must be fed from an interconnect logic clock source when the data is delivered with a strobe. A clock source from the interconnect logic can be connected to the CLKIN input of a MMCM or PLL.

### Example 1: Setup for a Source Synchronous Interface

- Data and clock are delivered to the UltraScale device. Assume that the data receive rate is 1250 Mb/s and the receive DDR clock is 625 MHz.
- The clock goes into a QBC/GC pin, is used as a BITSlice input, and fed to the PLL.



**TIP:** To ensure that all RX\_BITSlices start aligned, the RXCLK should be stopped until the RX VTC\_RDY signal is asserted. When the user has control of the TX while the RX is coming out of reset, the CLK to the RX side must be stopped until the RX VTC\_RDY signal is asserted. If the user does not have control of the TX side, then a bitslip module must be implemented in the RX side to ensure all channels are aligned.

- If the RX\_BITSlices are used in 8-bit mode, the parallel data clock or FIFO\_RD\_CLK must equal the incoming clock divided by four (for example,  $625 \text{ MHz}/4 = 156.25 \text{ MHz}$ ). This clock can also be generated by the same PLL used for the PLL\_CLK.
- The BITSlice\_CONTROL attributes are listed.
  - DIV\_MODE = DIV4
  - SELF\_CALIBRATE = ENABLE
  - SERIAL\_MODE = FALSE
  - REFCLK\_SRC = PLLCLK
- The RX\_BITSlice attributes are listed.
  - DATA\_WIDTH = 8
  - REFCLK\_FREQUENCY = 1250
- The CLKOUTPHY clock (generated by the PLL) is the PLL\_CLK input for the BITSlice\_CONTROL and the BISC controller. The PLL\_CLK is used to adjust and tune the clock to the data bits. When the data arrives at 1250 Mb/s, the bit time is 800 ps. To correctly position the clock to the data, the clock used by the BITSlice\_CONTROL must be equal to 1250 MHz.
- When the incoming clock is connected to a GC input pin, this clock is used as data sample clock and can be used as source for the PLL. It's also possible to use an independent clock for the PLL while the incoming clock is used as data sample clock.
- This example uses the received clock as data sample clock and PLL input clock.
- In this example, the received clock is used to sample the data and is also used as an input clock of a PLL that is generating the necessary BITSlice\_CONTROL clock and possibly generating one or two application clocks (for example, FIFO\_RD\_CLK).
  - The input clock is 625 MHz and the CLOCK\_PERIOD = 1.600.
  - The PLL must generate the CLKOUTPHY or PLL\_CLK for the BITSlice\_CONTROL. This clock must run at 1250 MHz for the BISC controller in the BITSlice\_CONTROL to be able to tune and keep the clock correctly in the center of a data bit.
  - For best results (the least jitter), the PLL VCO must run as fast as possible. Refer to the UltraScale device data sheets [Ref 2] for maximum VCO frequencies.

- The PLL attributes are listed.
  - DIVCLK\_DIVIDE = 1
  - CLKFBOUT\_MULT = 2
  - CLKOUTPHY\_MODE = VCO
  - CLKOUT0\_DIVIDE or CLKOUT1\_DIVIDE = 8
- DIVCLK\_DIVIDE and CLKFBOUT\_MULT are set for the VCO to run at 1250 MHz.
- CLKOUTPHY\_MODE = VCO ensures that the VCO clock is used as the BITSLICE\_CONTROL.PLL\_CLK clock.
- CLKOUTn\_DIVIDE sets one or both of the PLL clock outputs to a clock rate that can be used as the application (FIFO\_RD\_CLK) clock of 156.25 MHz.
- There are other attributes that must be set for the UltraScale device. Because they are not specific to this application note, they are omitted from this example.

### **Transmitters**

PLL\_CLK is used as a clock to pass the data from the parallel side to a serial output stream. The frequency of the generated PLL\_CLK must equal the required transmitter serial data stream. The supplied PLL\_CLK is used by the BITSLICE\_CONTROL to generate all necessary clocks to capture the parallel 4 or 8-bit data provided by the interconnect logic and serialize it.

A transmitter, TX\_BITSLICE, can be used as a clock generator when the TX\_BITSLICE.D[7:0] inputs are tied to a regular pattern. The pattern 01010101 generates a clock of 625 MHz when the applied PLL\_CLK is 1250 MHz. This setup generates perfectly synchronized clock/data interfaces. With the TX\_BITSLICE attribute OUTPUT\_PHASE\_90, it is possible to automatically generate a clock that is phase shifted to the accompanying data by 90° or a quarter of its period.



---

**TIP:** The correct clock pattern 01010101 (not 10101010) must be applied to avoid any unnecessary bitslips on the receiver side.

---

### Example 2: Transmitter Delivering Data-end Clock

- Assume that the TX\_BITSLICES operate in 8-bit mode. The clock frequency supplying data to the TX\_BITSLICE.D inputs can be calculated.  
  

$$(PLL\_CLK/2)/DIV_n \text{ with } DIV_n = 4 \text{ for 8-bit or } DIV_n = 2 \text{ for 4-bit.}$$

$$(1250 \text{ MHz}/2)/4 = 156.25 \text{ MHz}$$
- The BITSlice\_CONTROL attributes are listed.
  - DIV\_MODE = DIV4
  - REFCLK\_SRC = PLLCLK
- TX\_BITSLICE attributes are listed.
  - DATA\_WIDTH = 8
  - REFCLK\_FREQUENCY = 1250
- The PLL must generate the CLKOUTPHY or PLL\_CLK for the BITSlice\_CONTROL. This clock must run at 1250 MHz to generate a serial data/clock stream of 1250 Mb/s.
- For best results (the least jitter), the PLL VCO must run as fast as possible. Refer to the *Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS892) [Ref 2] for maximum VCO frequencies.
- The PLL attributes are listed.
  - The input clock is 625 MHz and the CLOCK\_PERIOD = 1.600.
  - DIVCLK\_DIVIDE = 1
  - CLKFBOUT\_MULT = 2
  - CLKOUTPHY\_MODE = VCO
  - CLKOUT0\_DIVIDE or CLKOUT1\_DIVIDE = 8
- DIVCLK\_DIVIDE and CLKFBOUT\_MULT set the VCO to run at 1250 MHz.
- CLKOUTPHY\_MODE = VCO makes that the VCO clock is used as BITSlice\_CONTROL.PLL\_CLK clock.
- CLKOUTn\_DIVIDE sets one or both of the PLL clock outputs to a clock rate that can be used as application clock of 156.25 MHz.
- There are other attributes that must be set for the UltraScale device. Because they are not specific to this application note, they are omitted from this example.

## Built-in Self-Calibration

The implementation of the built-in self-calibrating (BISC) controller in the I/O primitives ensures that extra interconnect logic is not required to calibrate and maintain the clock to data adjustment. The BISC controller in the BITSlice\_CONTROL primitive can be turned on by the SELF\_CALIBRATE attribute (default = TRUE). This attribute must be set together with all necessary attributes for a required I/O interface.

The BISC controller is only used for designs using TIME mode delay lines. TIME mode ensures the stability of clock setting points over any system process, voltage, and temperature (PVT) changes.

The BISC controller ensures the following.

- The output phase of a transmitter interface.
- The input phase shift (SHIFT\_0, SHIFT\_90) of a receiver interface.
- The deskew of all receiver data and clock paths.
- Maintain all clock relationships in the BITSlices.

The BISC controller signals the completion of the calibration to the interconnect logic by asserting the BITSlice\_CONTROL.DLY\_RDY. After calibration completes, the BITSlice\_CONTROL.EN\_VTC input must be asserted (High). This forces the BISC controller to maintain its internal calibration points for voltage and temperature, and the relative position of the clock to the data as seen by the internal sampling registers. The BISC controller signals to the interconnect logic that this step is completed by asserting BITSlice\_CONTROL.VTC\_RDY. Once the DLY\_RDY and VTC\_RDY signals are High, the interconnect logic can enter functional mode.

## RX\_BITSLICE Integrated FIFO

The RX\_BITSLICE integrated FIFO must have the FIFO\_SYNC\_MODE attribute set to FALSE. Other values are not support.

The FIFO allows captured data from the interface clock domain to successfully cross over to the interconnect logic clock domain(s). The FIFO functions as clock domain crossing (CDC) element in a design. The following lists the clock and control inputs of the FIFO.

- FIFO\_WRCLK\_OUT: All capture clocks are generated inside the BITSlice\_CONTROL primitive and dispatched to all connected RX\_BITSLICES. One of the internally generated clocks is a parallel-data clock that is used to write data into the FIFO, (FIFO\_WRCLK) inside the RX\_BITSLICE. A copy of this clock is provided as an output of the BITSlice in position 0 of a nibble. The frequency of the FIFO\_WR\_CLK\_OUT equals the PLL\_CLK divided by 4 (4-bits) or 8 (8-bits).
- FIFO\_RD\_EN: This pin must be High to read the FIFO. When this input pin is left Low, the FIFO output shows new data for every eight FIFO\_WRCLK\_OUT cycles in 8-bit mode and every four FIFO\_WRCLK\_OUT cycles in 4-bit mode. This is because the FIFO\_RD\_EN locks the read pointer of the FIFO while the write pointer advances with each write operation inside the RX\_BITSLICE. When the write pointer reaches the eighth pointer position, it loops

back to position zero and continues. Because the read pointer is locked, an empty pulse (on FIFO\_EMPTY output) is generated and new data arrives on the output pins. One out of eight or four parallel data bytes shows at the RX\_BITSLICE.Q outputs (Figure 9).

- **FIFO\_EMPTY:** This output is High when the FIFO is empty. As soon as data gets written in the FIFO, the signal drops Low and stays Low as long as the write and read FIFO pointers are not equal. The write pointer of the internal FIFOs increments when data is written. Once the write pointer moving away from the read pointer is detected, the FIFO\_EMPTY pin is de-asserted. This can take a synchronization time of one to two FIFO\_RD\_CLK cycles from the first write. Equal write and read pointers indicate that the FIFO is empty.

**Note:** Because FIFO\_WR\_CLK and FIFO\_RD\_CLK alignment might vary, the read pointer and write pointer are not guaranteed to overlap when FIFO\_RD\_EN is disabled. FIFO\_EMPTY is not guaranteed to go High every eight FIFO\_RD\_CLK periods when FIFO\_RD\_EN = 0.

- **FIFO\_RD\_CLK:** The FIFO read clock is supplied by the FIFO\_WRCLK\_OUT of BITSlice\_0 or by a PLL generated clock with a frequency equal to the BITSlice internal FIFO\_WRCLK.

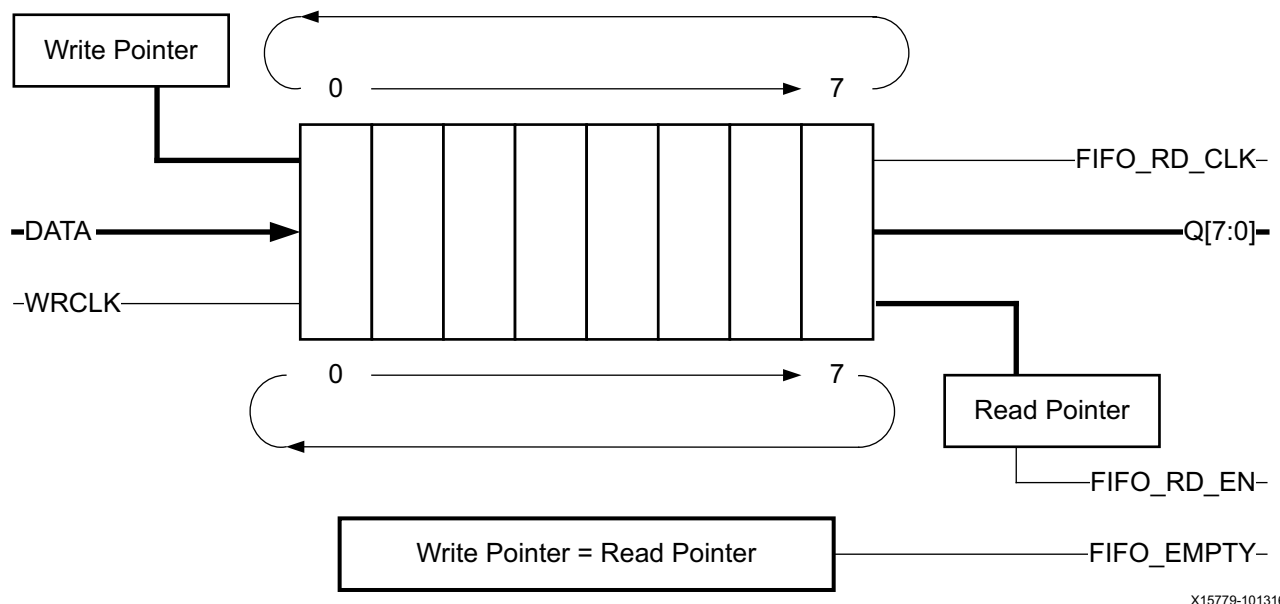


Figure 9: Schematic View of the RX\_BITSLICE FIFO

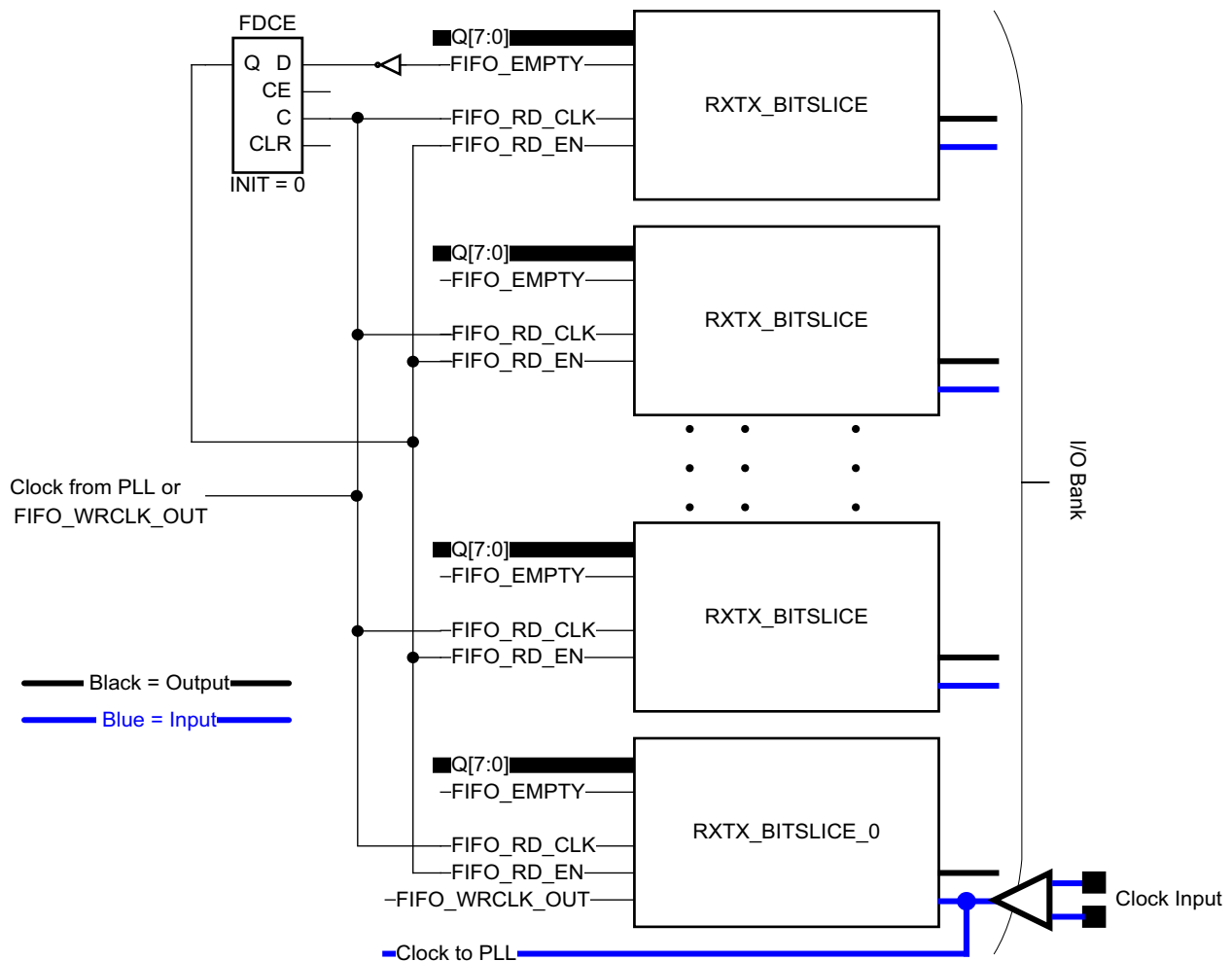
Captured data does not have to be aligned with the serial input channels. Each FIFO can be clocked with the FIFO\_WRCLK\_OUT of the RX\_BITSLICE that is at position zero in the nibble. A PLL generated clock can also be used to clock each FIFO. For each FIFO, connect the FIFO\_EMPTY (inverted) to the FIFO\_RD\_EN. As soon as the FIFO is not empty, the FIFO is enabled and data can be read from using the nibble FIFO\_WRCLK\_OUT or a PLL generated parallel data clock.

When data from each RX\_BITSLICE FIFO of an interface must be presented as aligned, use one of the following methods.

- Single-clock source-synchronous interface.
  - A PLL generated version of the FIFO\_WRCLK clock or the FIFO\_WRCLK\_OUT of the BITSlice that receives the FIFO\_RD\_CLK clock.



- Connect the inverted FIFO\_EMPTY signal of the BITSlice farthest from the BITSlice receiving the clock input (through a flip-flop) to all FIFO\_RD\_EN inputs [Figure 10](#).
- The timing can be challenging when using higher clock rates and adding pipelining along this path can add latency. As a reminder, interconnect logic can only be operated after VTC\_RDY is High.
- The FIFO\_WRCLK\_OUT or a PLL generated clock requires the use of BUFG clock buffers. While it is possible to connect in HDL the FIFO\_WRCLK\_OUT clock straight to the FIFO\_RD\_CLK, the Vivado tools automatically insert a BUFG clock buffer.



**Figure 10: FIFO Handling for Single Clock Interfaces**

- Multi-clock source-synchronous interface.
  - Multi-clock source-synchronous interfaces span more than one I/O bank and each I/O bank receives its own clock.
  - Use the same basic setup as described for the single-clock source-synchronous interface with the following changes.

- Use a NOR gate as input for the flip-flop combining the FIFO\_EMPTY signals of each used RX\_BITSLICE. The NOR gate waits for the last FIFO\_EMPTY to transition Low before triggering the FIFO\_RD\_EN through the flip-flop.
- The FIFO\_RD\_CLK can be a PLL generated clock from one of the input clocks or a FIFO\_WRCLK\_OUT of one of the interfaces or a PLL generated clock from any source as long as the clocks have the same frequency from the FIFO\_WRCLK (mesochronous or the same clocks).
- Using the FIFO\_WRCLK\_OUT or a PLL generated clock uses BUFG clock buffers. The FIFO\_WRCLK\_OUT clock can be connected (in HDL) directly to the FIFO\_RD\_CLK. The Vivado tools automatically insert a BUFG clock buffer.

Refer to the High Speed SelectIO Wizard—Known Issues List [\[Ref 7\]](#) for additional information.

## I/O Bring-up Sequence

To bring up a design in an UltraScale device using native SelectIO primitives, a specific set of steps must be followed to release the reset. This section describes the necessary design steps before applying or releasing the reset ([Figure 11](#)).

**Note:** Strobe clocks (DQS) must be disabled during the reset sequence. Bitflip is required for systems that use the input clock as the strobe clock. The High Speed SelectIO Wizard provides the bitflip functionality.

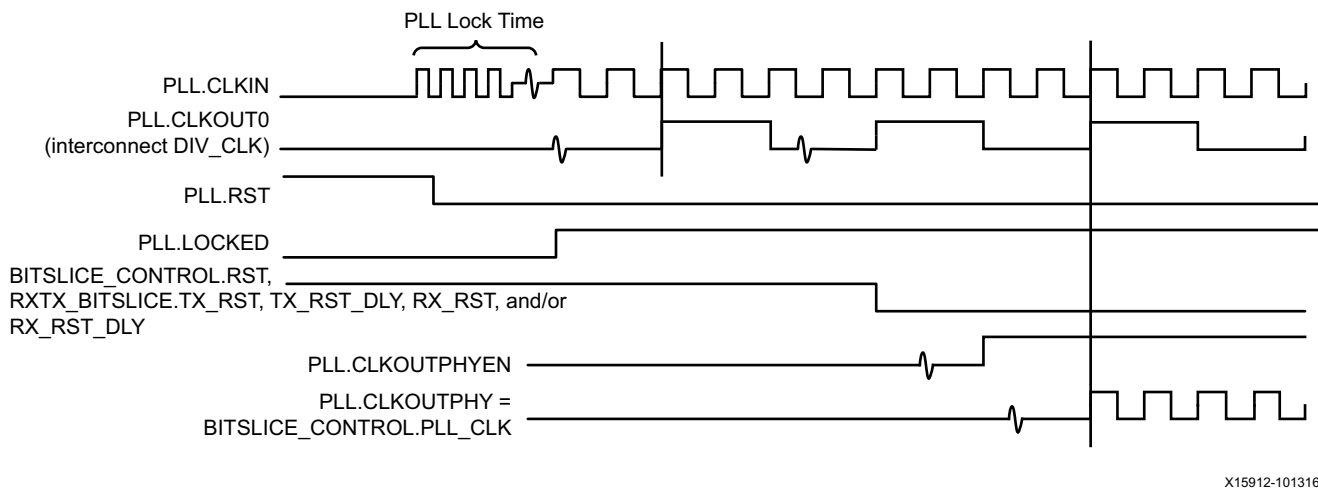


Figure 11: PLL and Reset Bring-up Sequence

### Apply Reset

1. The EN\_VTCs are High for all of the used RX\_BITSLICES, TX\_BITSLICES, and/or RXTX\_BITSLICES.
2. The SELF\_CALIBRATE attribute is set to ENABLE.
3. Assert reset to the PLL.

4. Apply reset to RXTX\_BITSLICE.TX\_RST\_DLY, RXTX\_BITSLICE.RX\_RST\_DLY, TX\_BITSLICE\_TRI.RST\_DLY, RXTX\_BITSLICE.TX\_RST, RXTX\_BITSLICE.RX\_RST, TX\_BITSLICE\_TRI.RST, and/or BITSLICE\_CONTROL.RST.
5. Wait the minimum PLL reset assertion time before releasing the reset. For this timing specification, consult the PLL section of the UltraScale device data sheets [\[Ref 2\]](#).

## Release Reset

1. Hold all the EN\_VTCs High for all of the used RX\_BITSLICES, TX\_BITSLICES, and/or RXTX\_BITSLICES. EN\_VTC of the BITSLICE\_CONTROL should be held Low.
2. Ensure that the SELF\_CALIBRATE attribute is set to ENABLE.
3. Use the following sequence to bring the I/O out of reset.
  - a. Release the reset of the PLL generating the clocks for the interface.
  - b. Wait for the PLL to reach the LOCKED state.
  - c. Release the reset of following primitives: RXTX\_BITSLICE.TX\_RST\_DLY, RXTX\_BITSLICE.RX\_RST\_DLY, TX\_BITSLICE\_TRI.RST\_DLY, RXTX\_BITSLICE.TX\_RST, RXTX\_BITSLICE.RX\_RST, TX\_BITSLICE\_TRI.RST, and/or BITSLICE\_CONTROL.RST
  - d. Wait a minimum of 64 application clock cycles (PLL specification).
  - e. Assert High the CLKOUTPHYEN signal and enable the CLKOUTPHY high-speed PLL output.
4. Continue with the following post-reset sequence.
  - a. Wait until the DLY\_RDY of all the used BITSLICE\_CONTROL primitives are asserted High.
  - b. After all the DLY\_RDY signals are asserted High, assert High the EN\_VTC of all the used BITSLICE\_CONTROLS. To perform this function, it is necessary to use the register interface unit (RIU) RIU\_CLK in a two flip-flop synchronizer circuit.
  - c. Wait until the BITSLICE\_CONTROL VTC\_RDY status output of every used BITSLICE\_CONTROL is asserted High.
  - d. Now, the application in the interconnect logic can be released.
5. The functional mode guidelines for the transmitter require asserting High the TBYTE\_IN[3:0] inputs of the BITSLICE\_CONTROL. This uses the VTC\_RDY signal and a two register synchronizer running from the application clock.
6. The functional mode guidelines for the receiver require asserting High the PHY\_RDEN[3:0] inputs of the BITSLICE\_CONTROL. This uses the VTC\_RDY signal and a two register synchronizer running from the application clock. Follow the actions described in the [RX\\_BITSLICE Integrated FIFO](#) section about reading data from the FIFO.

**Note:** For transmit only interfaces, the PHY\_RDEN[3:0] should be deasserted Low.

---

# High Speed SelectIO Wizard

The native I/O primitives as discussed in Chapter 2 of the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [Ref 1] and the [Native I/O Details](#) section can be assembled into a high-speed I/O interface (PHY). To simplify the effort requires the integrating these SelectIO primitives into high-speed system designs using UltraScale and UltraScale+ devices, Xilinx recommends using the High Speed SelectIO Wizard. This wizard is found in the IP core generator [Ref 6].

From GUI configuration tools, the High Speed SelectIO Wizard creates a top-level Verilog file, with pin-locking information (XDC) for instantiated and configured I/O, and native primitives such as RX\_BITSLICE, TX\_BITSLICE, RXTX\_BITSLICE, BITSlice\_CONTROL, and the PLL present in the UltraScale architecture's physical interface.

This section provides guidelines on how to generate a transmitter and receiver interface with the High Speed SelectIO Wizard and outlines the instructions and guidelines to customize, generate, and set up the interface constraints. It describes the various steps involved in simulation, synthesis, and implementation of an interface.

When using the High Speed SelectIO Wizard, you must know the interface requirements (interface speed, clock-data relationship, and system clock structure) and how the I/O primitives fit in the I/O bank. The interface requirements are design dependent, the primitive knowledge is further outlined in [Appendix A](#).

---

## Source Synchronous Interfaces

The following steps are used to configure and generate the source-synchronous interfaces using the High Speed SelectIO Wizard (IoWizard\_RxTxInterface).

### Pre-core Generation Setup

1. Browse to the folder where you will store the design and run the `ProjectGen.bat` file available for the reference design ZIP file. The BAT file creates a directory structure, [Figure 12](#), like the IDE tool used to create this reference design.
2. Open the file explorer (Windows), Nautilus, or other (Linux) and browse to the `/Libraries` folder in the directory where the design project is generated. Also open a command window (Windows or terminal window (Linux)) and browse to the folder.



---

**TIP:** In Windows, when browsed with the file explorer into the required folder, right click in the address bar and select copy from the popup menu. Click into the command window and type `cd`, then right click and hit paste. Then click on enter.

---

3. When generating a new original design, the command window/terminal shows an empty `/Libraries` folder. When the reference design is used, the `/Libraries` folder should already contain sub-folders with the design library elements already created.

The example process outlined in this application note assumes a new original design.

4. In the /Libraries folder, create a folder called /RxTx\_Intrfce\_Lib. The reference design already uses this folder name. To create an original transmitter or receiver chose a different sub-directory name. This folder is used to put the High Speed SelectIO Wizard created transmitter and receiver design.
5. Change the directory in the command window/terminal into the /RxTx\_Intrfce\_Lib folder.
6. Start the Vivado tools from within the command window/terminal (type: Vivado [enter]).
7. Click on the [Manage IP] icon and select [New IP Location] from the popup selection.
8. A *Create a New Customized IP Location* window pops up. Click [Next].
9. In the window, fill in the following information for this example.
  - Part: xcvu095-ffva2104-2-e (active).
  - Target Language: VHDL (use this even though the High Speed SelectIO Wizard only generates Verilog cores).
  - Target Simulator:
  - Simulator Language: Mixed
  - IP Location:
    - <Path\_To\_Core>/IoWizard\_RxTxInterface/Libraries/RxTx\_Intrfce\_Lib/<sub-folder>. Where the <sub-folder> is:
      - Rx\_8Ch8b\_Ssync\_Intrfce: for the 8 channel, 8-bit receiver.
      - Tx\_8Ch8b\_Ssync\_Intrfce: for the 8 channel, 8-bit transmitter.

Figure 12 shows the folders created in the /Libraries folder.

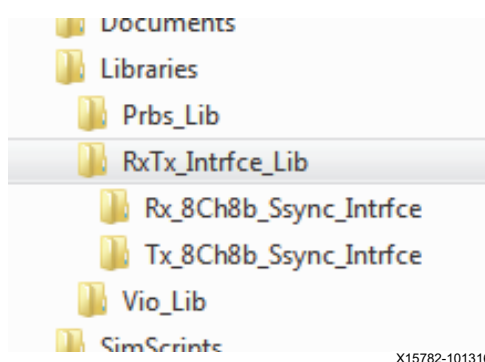


Figure 12: Generated Cores in the /Libraries Folder

10. Click [Finish]. When the folder does not yet exist, a dialog will pop up and ask if you want to create it. Click [OK].
11. Now the *IP Catalog* starts and all files and folders created are generated in the Vivado tools sub-folder where it was started.

12. Under the [Cores] tab, select and double click:

FPGA Features and Design → IO Clocking → High Speed SelectIO Wizard

13. The core generator starts.

14. To configure the core, start with the [Configure and Generate a High Speed SelectIO Wizard Core](#) section.

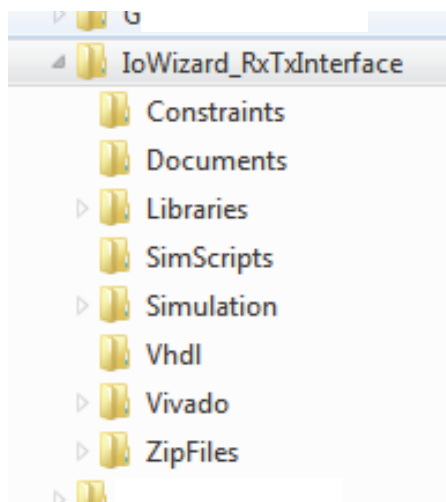
15. When done, click in the Vivado tools window task bar.

[File] → [Close Project]

16. A new core can be generated under the same library folder. If the first core was a transmitter, then next generate a receiver.

17. Restart the process from [step 7](#) and follow the flow up to [step 15](#).

The initial project folder resembles [Figure 13](#).



X16315-101316

**Figure 13: RxDxInterface Initial Project Folders**

To create other cores in the same library folder, keep repeating the process between [step 7](#) and [step 15](#) for each generated core. To generate a core in a different library, for example, the VIO core in this design, close the Vivado tools and start from [step 2](#). The cores generated previously have a set of folders and files, however, there is more than one way to generate cores.

To generate/regenerate the VIO core used in this reference design close the Vivado tools and start at [step 2](#). Store the VIO core in its own library folder (Vio\_Lib) and use the name *VioCore*. For further guidelines about the VIO core, see the following references.

- *Virtual Input/Output v3.0 Product Guide* (PG159) [\[Ref 3\]](#)
- *ChipScope Pro Virtual Input/Output (1.04a)* (DS284) [\[Ref 4\]](#)
- *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [\[Ref 5\]](#)

## Manipulating and Existing Core

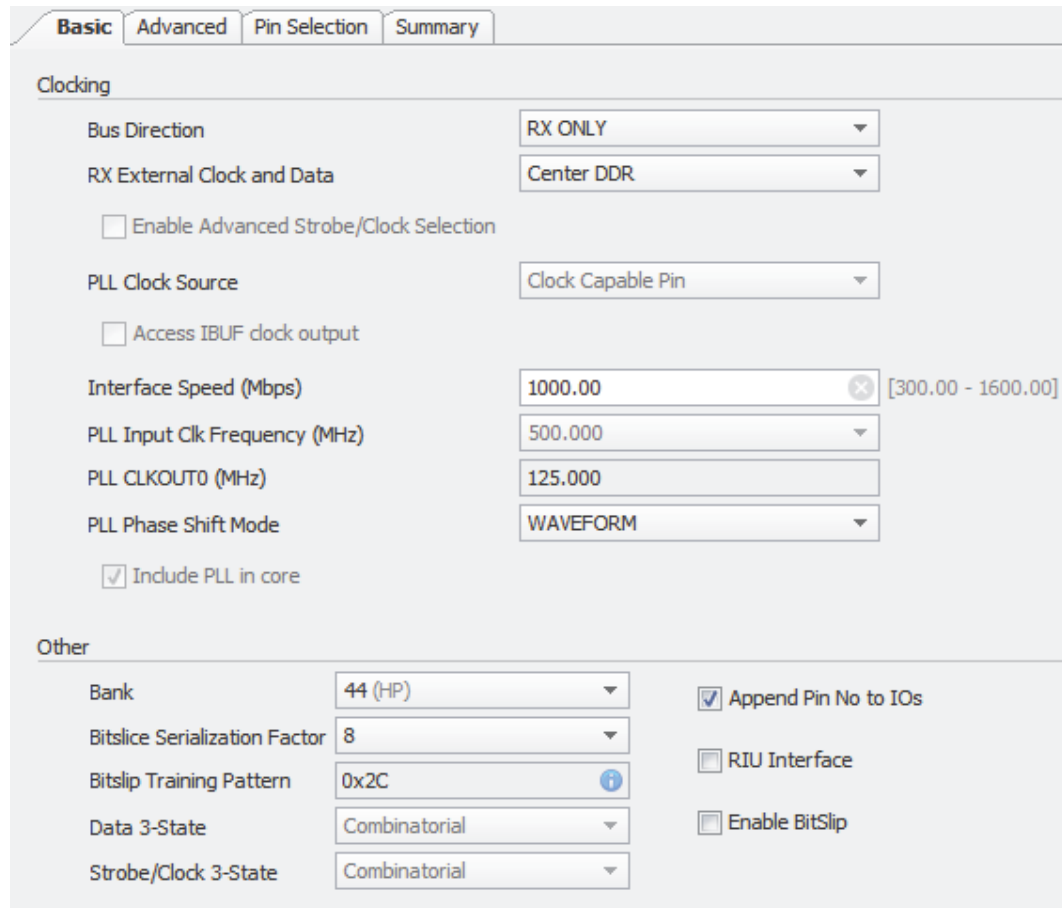
1. Click the [Manage IP] icon in the Vivado tools main window and select [Open IP Location] from the popup menu.
2. Browse to the folder containing the IP core that needs to be changed. For this reference design use the following paths.
  - <Path>\IoWizard\_RxTxInterface\Libraries\RxTx\_Intrfce\_Lib\Rx\_8Ch8b\_Ssync\_Intrfce
  - <Path>\IoWizard\_RxTxInterface\Libraries\RxTx\_Intrfce\_Lib\Tx\_8Ch8b\_Ssync\_Intrfce
3. Hit [Select].
4. The core to modify appears in the *Sources* window on the left.
5. Double click the name of the core to open the High Speed SelectIO Wizard window and start modifications.
6. When all modifications are complete, click [OK] and let the tool generate the core. This step involves clicking [OK] a few times.

## Configure and Generate a High Speed SelectIO Wizard Core

When the previous [step 14](#) in the [Pre-core Generation Setup](#) is complete, configure a transmitter or receiver core using the High Speed SelectIO Wizard.

Assuming that the High Speed SelectIO Wizard is open on screen.

1. Give the core a valid name. Click in the *Component Name* box and enter the name. For example, Component Name: Rx\_8Ch8b\_Ssync\_Intrfce.
2. The next section, including [Figure 14](#) and [Figure 15](#), highlight the selections available on the tabbed sheets of the High Speed SelectIO Wizard. The options available are mentioned with a short functional description. For complete and in-depth discussions on these topics, read the previous section and consult the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [\[Ref 1\]](#).
3. After selecting all the items for the core on the *Configuration* tab, switch to the *Pin Selection* tab.
4. The *Pin Selection* (tabbed sheet) places/fixes the configured interface channels in the selected I/O bank and is organized as shown in [Figure 44](#) of [Appendix A](#). For more background information and guidelines for obtaining better pinouts for a design, read [Appendix A](#).



The image shows the 'Basic' tab of the High Speed SelectIO Wizard. It is divided into two sections: 'Clocking' and 'Other'.

**Clocking Section:**

- Bus Direction: RX ONLY (dropdown)
- RX External Clock and Data: Center DDR (dropdown)
- ☐ Enable Advanced Strobe/Clock Selection
- PLL Clock Source: Clock Capable Pin (dropdown)
- ☐ Access IBUF clock output
- Interface Speed (Mbps): 1000.00 (text input with a range of [300.00 - 1600.00])
- PLL Input Clk Frequency (MHz): 500.000 (dropdown)
- PLL CLKOUT0 (MHz): 125.000 (text input)
- PLL Phase Shift Mode: WAVEFORM (dropdown)
- ☒ Include PLL in core

**Other Section:**

- Bank: 44 (HP) (dropdown)
- Bitslice Serialization Factor: 8 (dropdown)
- BitSlip Training Pattern: 0x2C (text input with an info icon)
- Data 3-State: Combinatorial (dropdown)
- Strobe/Clock 3-State: Combinatorial (dropdown)
- ☒ Append Pin No to IOs
- ☐ RIU Interface
- ☐ Enable BitSlip

X15783-010417

**Figure 14: High Speed SelectIO Wizard Basic Entry Page**

The clocking structure that can be generated using the different GUI options in the wizard are discussed in the [High Speed SelectIO Wizard Clocking Structures](#) section.



Table 6: Descriptions of Pull Downs and Click Boxes in Figure 14

Selection		Description
<b>Clocking</b>		
Bus direction	TX only	Design will only contain TX pins.
	RX only	Design contains only RX pins.
	BIDIR or TX + RX or TX + RX + BIDIR	Design contains a mix of TX, RX, and bidirectional pins.
RX external clock and data	Edge DDR	The supplied clock and data are phase aligned. The clock can only be connected to the QBC/GC pin or pin pair. The supplied clock is used as a data capture clock and as an input clock for one of the PLLs.
	Center DDR	Same as for edge DDR, however, the clock is shifted 90° with respect to the data.
	ASYNC, NONE, or Fractional	No clock is supplied with the data. Consult the <i>High Speed SelectIO Wizard Product Guide</i> (PG188) [Ref 6] for currently supported modes.
	Edge DDR strobe/clock	The applied data and clock are phase aligned, the clock is used to capture the data but is not used for the PLL. The clock fits into any of the QBC/DBC pins in an I/O bank.
	Center DDR strobe/clock	Same as the edge DDR strobe except that the data and clock are 90° phase shifted.
PLL clock source	GC pin	The clock arrives at a GC pin and is used as an input clock for the PLL. For a receiver, this selection is grayed out when an edge or center DDR is chosen as external clock.
	Fabric (driven by BUFG)	The clock for the PLLs must be supplied from within the interconnect logic using a BUFG, BFUGCE, or other clock buffer.
	Access IVUB clock output	Routes the output of the clock input buffer through the IP core and presents it as an output of the core. This way it can be used as input for a second core. PLL. <b>Note:</b> Only usable with ASYNC/NONE.
Interface speed		Specifies the data rate of the interface in Mb/s.
PLL CLK input frequency (MHz)		Selects the PLL input clock from the list. Normally, the correct value is automatically filled in. When edge DDR or center DDR are chosen, the clock selected is data_rate/2.
PLL0 CLKOUT0 (MHz)		Selects the frequency of the CLKOUT0 of PLL0. The frequency of this clock is data_rate/serialization factor.
PLL phase shift mode		Select whether the phase-shifted clock should be modeled into the clock WAVEFORM or LATENCY. No multi-cycle constraint is needed when modeled through LATENCY. The PHASE_SHIFT_MODE property is set in the generated XCD. See the <i>Vivado Design Suite User Guide: Design Analysis and Closure Techniques</i> (UG906) [Ref 11] for details.
Include PLL in core		Select to include the PLL in the core and/or example design.
<b>Other</b>		
Bank		For the device selected, pick an I/O bank from the list of all the available HP I/O and/or HR I/O banks.

Table 6: Descriptions of Pull Downs and Click Boxes in Figure 14 (Cont'd)

Selection		Description
BITSlice serialization factor		Parallel data input/output factor from/to the general interconnect. Accepted values are 4 and 8. Defines the interconnect logic clock frequencies: 4-bit receiver = Capture clock/2 8-bit receiver = Capture clock/4 4-bit transmitter = CLKOUTPHY/4 8-bit transmitter = CLKOUTPHY/8
Bitslip training pattern		Only available when <i>Enable Bitslip</i> is selected. Enter the value that the bitslip state machine should look for in the received BITSlice output. When the pattern is detected, the output of the bitslip will lock to the pattern assuring that new received data will be nibble or byte aligned.
Data 3-state	Combinatorial	Only available for a transmitter core. Uses the T pin of the BITSlice. This pin connects the IOBUF.T pin straight into the device logic and performs as a 3-state block.
	Serialized	Only available for a transmitter core. Uses the TBYTE_IN[3:0] bus to perform a per-bit 3-state on the serial output stream. Further details of this function are available in the <i>UltraScale Architecture SelectIO Resources User Guide</i> (UG571) [Ref 1].
Strobe 3-state Clock 3-state	Combinatorial	Only available for a transmitter core. Uses the T pin of the BITSlice. This pin connects the IOBUF.T pin straight into the interconnect logic and performs as a 3-state block.
	Serialized	Only available for a transmitter core. Uses the TBYTE_IN[3:0] bus to perform a per-bit 3-state on the serial output stream. Further details of this function are available in the <i>UltraScale Architecture SelectIO Resources User Guide</i> (UG571) [Ref 1].
Append pin numbers to I/Os		When selected, the pin names of the generated core are extended with the value of the BITSlice position in the nibble.
RIU interface		Enables the register interface unit (RIU). Further information on the micro-controller/state machine access port is available in the <i>UltraScale Architecture SelectIO Resources User Guide</i> (UG571) [Ref 1].
Enable bitslip		Only available for a receiver. Enables an in-logic bitslip state machine operating as a slip per bit.
Bitslip mode		Only available when <i>Enable Bitslip</i> is selected.

Basic **Advanced** Pin Selection Summary

Clocking Data and Delay

☐ Rx Delay Cascade

RX Delay Mode

RX Delay Type

RX Delay Value  [0 - 1250] (ps)

TX Delay Type

TX Delay Value  [1 - 1250] (ps)

Clock Forward Phase

☐ FIFO Read Enable User Control ☐ Generate RIJ Clock from PLL

☐ Enable PLL CLKOUT1 (MHz) ☐ Enable FIFO WRITE CLKOUT

I/O Standard

☒ Enable N-side RX bitslice

Differential IO Std	<input type="text" value="NONE"/>	Single IO Std	<input type="text" value="NONE"/>
Differential Termination	<input type="text" value="NONE"/>	Single Ended Termination	<input type="text" value="NONE"/>
Differential Tx Pre-Emphasis	<input type="text" value="NONE"/>	Single Ended Tx Pre-Emphasis	<input type="text" value="NONE"/>
Differential Rx Equalization	<input type="text" value="NONE"/>	Single Ended Rx Equalization	<input type="text" value="NONE"/>

X15784-010417

Figure 15: High Speed SelectIO Wizard Advanced Entry Page

Table 7: Descriptions of Pull Downs and Click Boxes in Figure 15

Selection		Description
<b>Clocking Data and Delay</b>		
RX delay cascade		Enables cascading of IDELAY delay lines. Not supported for bidirectional interfaces. Delay cascading is not recommended for interfaces operating at 500 Mb/s or higher.
RX delay mode		TIME: Delay factor is presented in ps. COUNT: Delay factor is presented in number of taps.
RX delay type	FIXED	Fixed delay value is applied on RX data.
	VARIABLE	A given delay value can be incremented or decremented using delay control inputs CE, CLK, LD, and INC.
	VAR_LOAD	A given delay value can be incremented or decremented using values applied to CNTVALUEIN and control inputs CE, CLK, LD, and INC.
RX delay value		When RX delay mode is set to TIME, the delay is given in picoseconds and can be between 0 ps and 1.25 ns. When RX delay mode is set to COUNT, the delay value is presented in the number of taps (0 to 512 taps).
TX delay type	FIXED	Fixed delay value applied on TX data.
	VARIABLE	A given delay value can be incremented or decremented using delay control inputs CE, CLK, and INC.
	VAR_LOAD	A given delay value can be incremented or decremented using values applied to CNTVALUEIN and control inputs CE, CLK, LD, and INC.
TX delay value (ps)		Fixed value applied to the input delay lines (up to 1.250 ns). This value is set to 1 by default. A value of 1 helps align the synchronous toggling of all outputs.
Clock forward phase		Available only for the TX pins (supported values: 0 and 90). Sets the phase between clock forward and TX data.
FIFO read enable user control		This option provides the user with a FIFO enable pin per used BITSlice/channel. When this option is not selected, the core enables the used FIFOs as soon as the FIFO_EMPTY is not true. When this option is selected, you decide to enable or disable one or more FIFOs in the generated interface.
Generate RIU clock for PLL		The RIU clock can come from different sources. One possible source for the RIU clock input comes from using the PLL that generates the BITSlice.CONTROL.PLL_CLK clock. Selecting this option activates the selection of whether the PLL is included in the core or example design.
Enable PLL0 CLKOUT1 (MHz)		Makes the second clock output of the PLL available to the interconnect logic. Select a clock frequency from the drop down list or go with the automatically calculated frequency equal to (data rate/4).
Enable FIFO write CLKOUT		Provides the internal RX_BITSlice FIFO write clock as an output to the interconnect logic.
<b>I/O Standard</b>		
Enable N-side RX_BITSlice		The differential input buffer has a single output connected to the even or P-side bit slices of the I/O logic. When this tick box is selected, the odd or N-side bit slices can be used to make the differential input buffer of type IBUFDS_DIFF_OUT.
Differential I/O standard		Set I/O standard constraints in the XDC file.

Table 7: Descriptions of Pull Downs and Click Boxes in Figure 15 (Cont'd)

Selection	Description
Differential termination	Pick values from the list. For the correct value for the chosen I/O standard, see the <i>UltraScale Architecture SelectIO Resources User Guide</i> (UG571) [Ref 1].
Differential TX pre-emphasis	
Differential RX equalization	
Single I/O standard	Set I/O standard constraints in the XDC file.
Single-ended termination	Pick values from the list. For the correct value for the chosen I/O standard, see the <i>UltraScale Architecture SelectIO Resources User Guide</i> (UG571) [Ref 1].
Single-ended TX pre-emphasis	
Single-ended RX equalization	

This section discusses the options to configure an interface using the High Speed SelectIO Wizard GUI for the transmitter and receiver cores. Figure 16 shows the information necessary to make a receiver and a transmit interface with the following requirements.

- 1250 Mb/s source-synchronous receiver with eight data channels of 8-bits and a 90° phase-shifted free running clock.
- 1250 Mb/s transmitter with eight data channels of 8-bit with generated 90° phase-shifted clock.
- The RIU interface and CLKOUT1 must be present.
- The receiver and transmitter must be placed in the same I/O bank.
- The transmitter operates on a clock from somewhere in the interconnect logic.
- Both receiver (Table 8) and transmitter (Table 9) must be able to run at different data rates. This last requirement requires that the generated HDL code must be altered to use both PLLs in an I/O bank.
- Not all options listed are used in the generated reference designs.

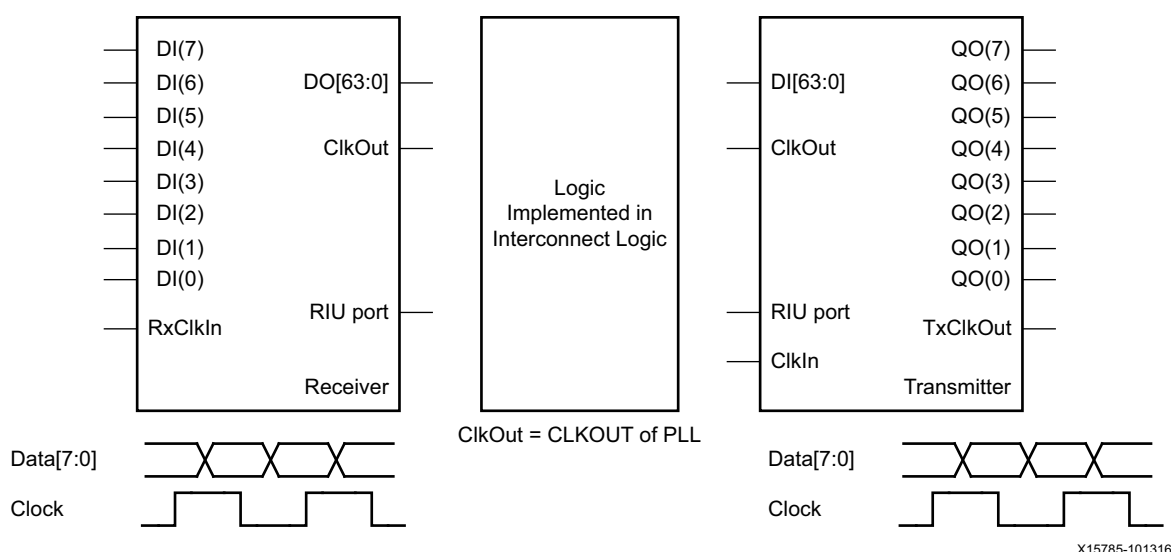


Figure 16: Requirements for the Receiver and Transmitter Interfaces

**Table 8: Receiver Requirements**

Component name	Wix_8ch8b_Rx_Intrfce
Bus direction	RX_ONLY
Serialization factor	8
RX external clock and data	Center DDR strobe. Arriving clock is used to capture the data.
PLL clock source	GC pin (clock input for the PLL)
Interface speed	1250 Mb/s
PLL clock input	625 MHz (auto-selected and grayed out)
PLL0 CLKOUT0	156.25 MHz (auto calculated)
Enable PLL0 CLKOUT1	Tick the box and provide clock speed (312.5 MHz)
RIU interface	Click the box
RX delay type	FIXED
RX delay value	0
Bank	68
Differential I/O standard	Provided in the pin selection (second tab). XDC or HDL must reflect that it is a differential I/O standard.
Pin selection	Byte_2 and byte_3 in the second tab. No choice since the center DDR/GC is chosen, clock input is fixed to the QBC/GC input at byte_2. Data inputs: Four channels in byte_2 and four in byte_3.

When all the data is entered, click OK and let the Vivado tools generate the HDL for the interface. Click [Generate] in the new pop-up menu.

**Table 9: Transmitter Requirements**

Component name	Wix_8ch8b_Tx_Intrfce
Bus direction	TX_ONLY
Serialization factor	8
PLL clock source	Interconnect logic
Interface speed	1250 Mb/s
PLL clock input	125 MHz
PLL0 CLKOUT0	156.25 MHz
Enable PLL0 CLKOUT1	Click the box and provide clock speed (312.5 MHz)
RIU interface	Click the box.
TX delay type	FIXED
TX delay value	1
Clock forward phase	90
Bank	68

Table 9: Transmitter Requirements (Cont'd)

Differential I/O standards	Provided in the second tab ( <i>Pin Selection</i> ). XDC or HDL must reflect that it is a differential I/O standard.
Pin selection	Byte_0 and Byte_1 in second tab. The DBC differential pin is selected to act as clock output (Clk forward), four data channels in byte_0 and four data channels in byte_1.

When all data is entered, click OK and let the Vivado tools generate the HDL for the interface. Click [Generate] in the new pop up menu.

Follow this process for both the transmitter and receiver core. If necessary, also follow it for the VIO core. At this point, all IP cores are generated, but a design using these cores is necessary. The top-level design using all cores is discussed in the next section.

## Top-level Design using High Speed SelectIO Wizard Cores

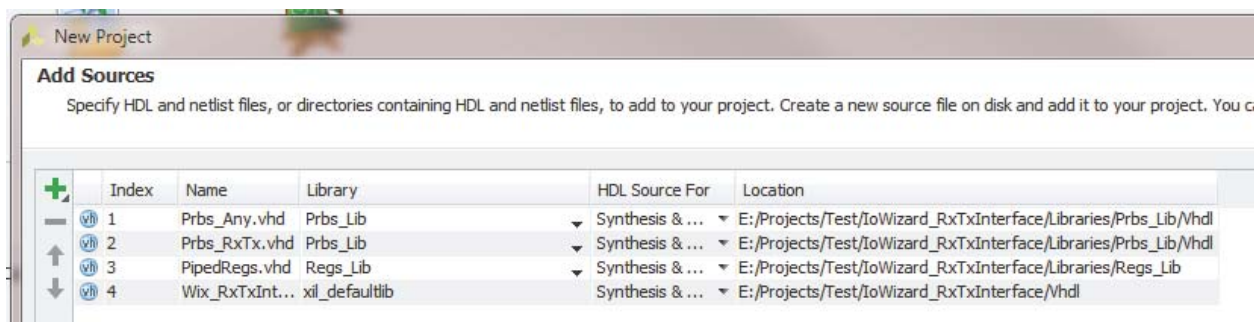
1. Open the file explorer (Windows), Nautilus or other (Linux), and browse to the `/Vivado` folder in the directory where the design project is generated. Also open a command window (Windows or terminal window (Linux)) and browse to the same directory.
2. When a design is new, the command window/terminal shows an empty `/Vivado` folder. When using the reference design, the `/Vivado` folder already contains the sub-folders with design.
3. This example assumes a new design.
4. In the command window/terminal pointing into the `/Vivado` folder, type `vivado` to start the Vivado tool.
5. When the Vivado tools are started, create a new project or open an existing project to add one or multiple SelectIO interfaces. This example assumes that a new project is created.
6. When there is already a Vivado project available, open it using the following process.
  - a. Select on the main page [Open Project]
  - b. Browse down until the `<file_name>.xpr` file is shown.
  - c. Select that file and click [OK].
7. When no project exists or when creating a new project, click the *Create New Project* icon on the start page in the Vivado tools.
  - a. Click Next.
  - b. Enter the project name.
  - c. Check the project location. If the location is not correct, change it to the correct location.



**TIP:** When creating a new project, check the box (in the Vivado tool) to create a sub-folder containing the just created project. Sub-folders are very convenient when testing multiple versions of the same project/design.

- d. Click Next.

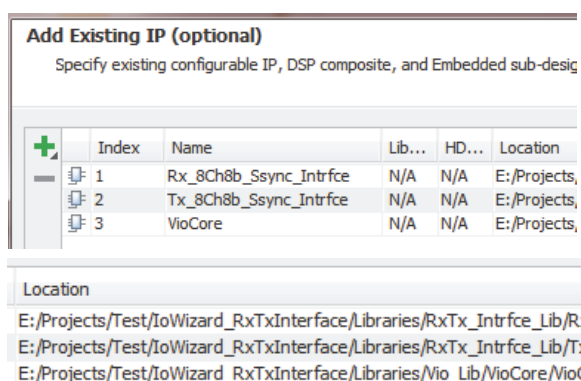
- e. On the following page, check the RTL Project option.
- f. The Add Sources window opens.
  - Click on the + sign and select Add Files.
  - Add files in the window until it resembles [Figure 17](#).
  - Do not forget to set the library for each of the files as shown in [Figure 17](#).



X16325-101316

*Figure 17: Add Sources Example Folder*

- g. Click [Next]
- h. The Add Existing IP (optional) window opens.
  - Add the previously generated cores (transmitter, receiver, and VIO).
  - Click on the + sign and select Add Files.
  - Add files in the window until it resembles [Figure 18](#).



X16326-101316

*Figure 18: Add Existing IP Example Folder*

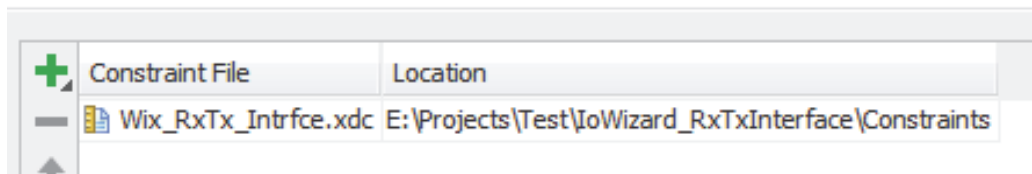
- i. Click [Next].



- j. The Add Constraints (optional) windows opens.
  - The reference design is a constraints file in the /Constraints folder. If using it, add it here.
  - Click on the + sign and select Add Files.
  - After adding the file, the window should resemble [Figure 19](#).

### Add Constraints (optional)

Specify or create constraint files for physical and timing constraints.

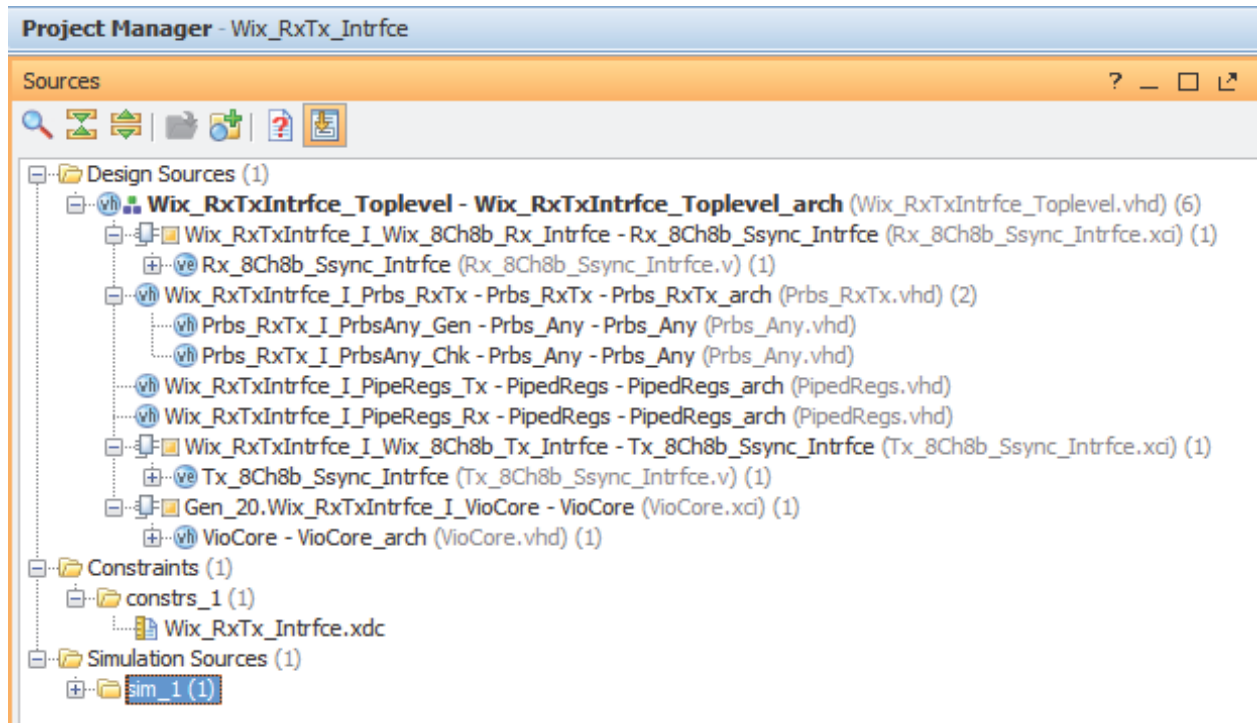


X16327-101316

Figure 19: Add Constraints Example Folder

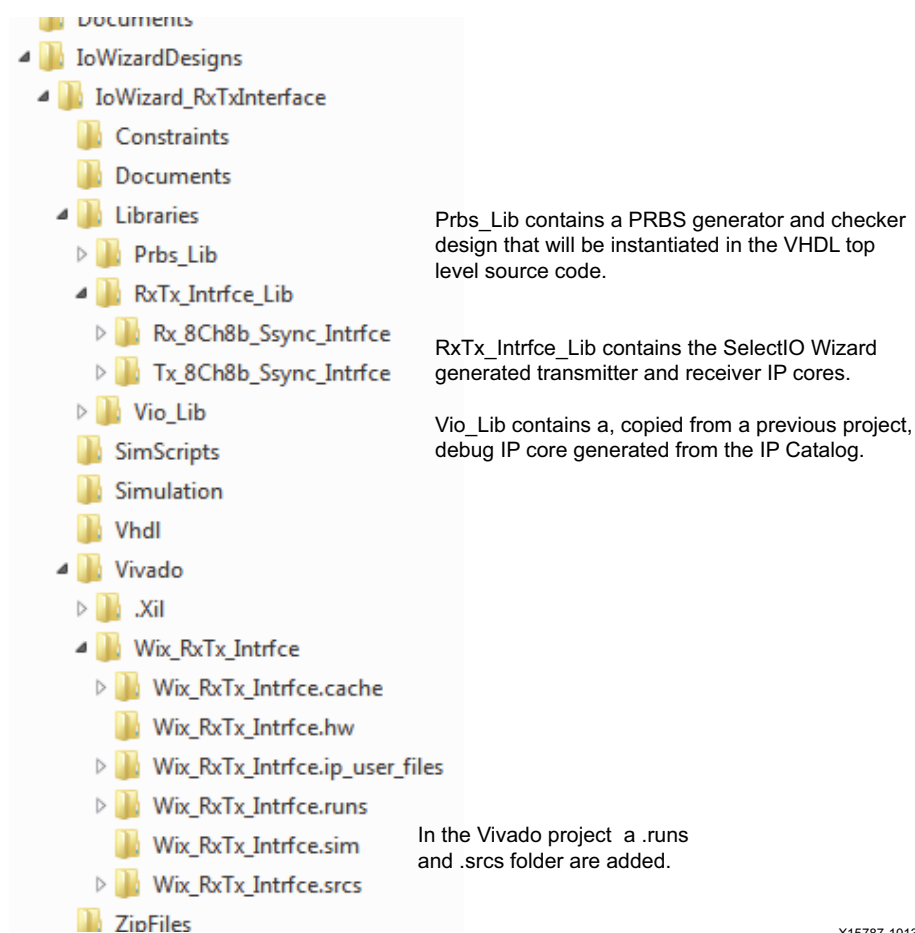
- k. Click [Next].
  - l. Select the UltraScale device. This example selected the XCVU095-FFVA2104 (-2 speed grade), which also fits on the development board.
  - m. Click Finish.
8. The Vivado tools create the necessary files and folders that become the design entry mode.
  9. Select *Run Synthesis*, *Run Implementation*, and *Generate Bitstream*.

An example of the *Sources* window of the Vivado IDE tools is shown in Figure 20 and an example of the folders created is shown in Figure 21.



X15786-101316

Figure 20: The Sources Window in the Vivado Tools



X15787-101316

**Figure 21: IP and Vivado Tools Folders after Generating the Core**

At this point, both the receive and transmitter interfaces (and the VIO core) generated by the High Speed SelectIO Wizard and each of the other design files are synthesized and implemented in the UltraScale device.

The block diagram of the reference design is shown in [Figure 22](#). [Figure 22](#) uses the generated interfaces example and other details from [Figure 16](#) to show a block diagram of both interfaces as constructed in hardware using the High Speed SelectIO Wizard.

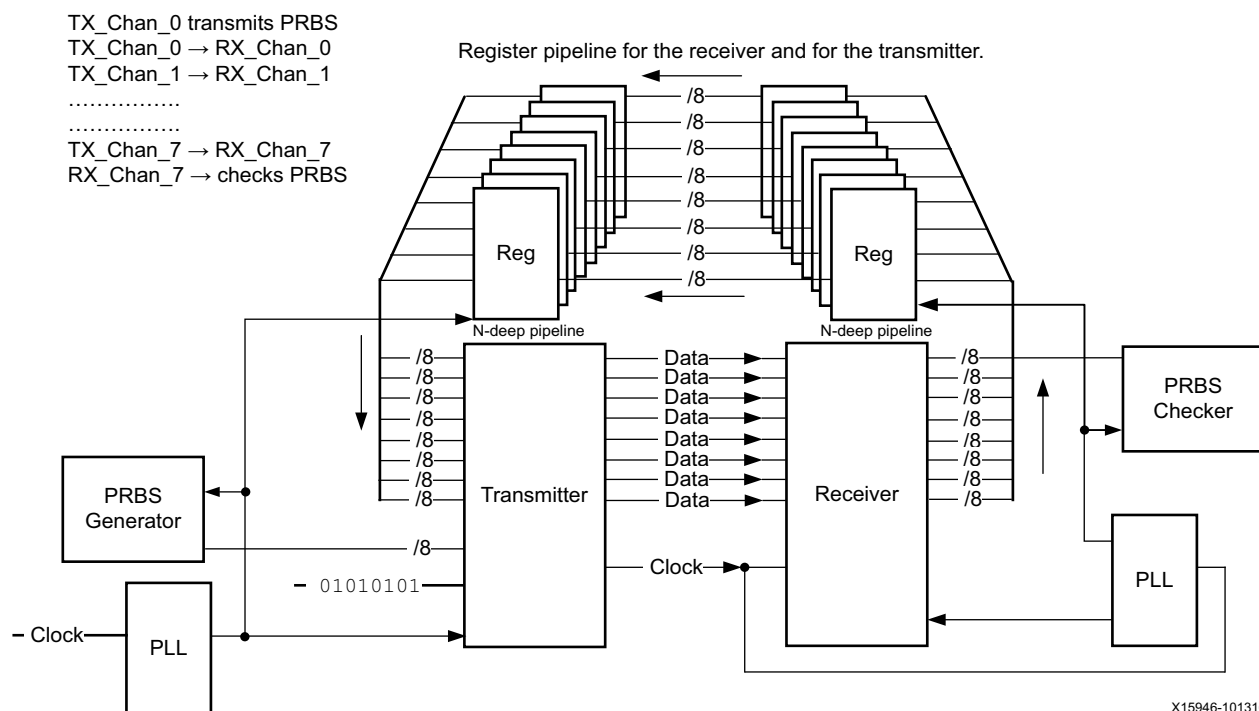


Figure 22: Receiver and Transmitter Interface Generated by the High Speed SelectIO Wizard

## Implementing the Interface

The generated IP cores are instantiated in the top-level HDLs written for this design in VHDL. This section describes how the top level VHDL file and top level XDC files are generated before synthesis and implementation.

1. Start a new HDL file with a text editor.
  - The top level VHDL source code is stored in the `/Vhdl` folder under the project root folder.
  - The files generated by the High Speed SelectIO Wizard and those necessary for completion of the top-level design (`.xdc`, `.vho`) are found in the IP cores folders under the `/Libraries` sub-folder.
2. The top level HDL file in the reference design with this application note shows how the IP cores are instantiated and how the extra source code is added between both cores and other necessary design items.
3. Beside HDL source files, the design also needs constraints (pin placement, timing). The High Speed SelectIO Wizard generates a constraints file for each of the generated cores. The generated XDC files contains the pin placement following the entries done for each transmit and/or receive channel.
4. Using a text editor, open the generated XDC files and copy the contents into a design top-level XDC file. For the reference design, the IP core XDC files are found in the following library locations.

```
../Libraries/RxTx_Intrfce_Lib/Rx_8Ch8b_Ssync_Intrfce/Rx_8Ch8b_Ssync_Intrfce.xdc
../Libraries/RxTx_Intrfce_Lib/Tx_8Ch8b_Ssync_Intrfce/Tx_8Ch8b_Ssync_Intrfce.xdc
```

5. Add a top-level design XDC file with timing and other constraints to the Vivado project. Along with the timing and other constraints, copy the contents of both the High Speed SelectIO Wizard and the generated XDC files.
  - a. Rename the port labels to the port labels used on the toplevel. For example, the generated core gets a clock input label of `clkIn_p`, but the top-level HDL design uses the pin name `ClkIn_p_pin`. Rename the copied syntax.
  - a. Define the IOSTANDARD, on-chip termination, and if used, BIAS and/or equalization.



**TIP:** The labels in the **pin names** column (tab 2) are not translated into the generated core files. Be sure to check the pin labels in the generated files with those used in the top-level design.

6. Synthesize and implement the design.

## Reference Design (IoWizard\_RxTxInterface)

You can download the [Reference Design Files](#) for this application note from the Xilinx website.

Table 10 shows the reference design matrix.

Table 10: IoWizard\_RxTxInterface Reference Design Matrix

Parameter	Description
<b>General</b>	
Developer Name	Jim Tatsukawa and Marc Defossez
Target Device	UltraScale FPGAs
Source code provided	Yes
Source code format	VHDL and Verilog
Design uses code and IP from existing Xilinx application note and reference designs or third party	Yes
<b>Simulation</b>	
Functional simulation performed	Yes
Timing simulation performed	N/A
Test bench used for functional and timing simulations	Yes
Testbench format	VHDL and Verilog
Simulator software/version used	QuestaSim_10.3d
SPICE/IBIS simulations	No
<b>Implementation</b>	
Synthesis software tools/versions used	Vivado 2015.4 or later
Implementation software tools/versions used	Vivado 2015.4 or later
Static timing analysis performed	Yes

Table 10: IoWizard\_RxTxInterface Reference Design Matrix (Cont'd)

Parameter	Description
<b>Hardware Verification</b>	
Hardware verified	Yes
Hardware platform used for verification	KCU105 and VCU108

### Reference Design Directory Setup (IoWizard\_RxTxInterface)

The directory structure for the reference designs are shown in Figure 23 and Figure 24.

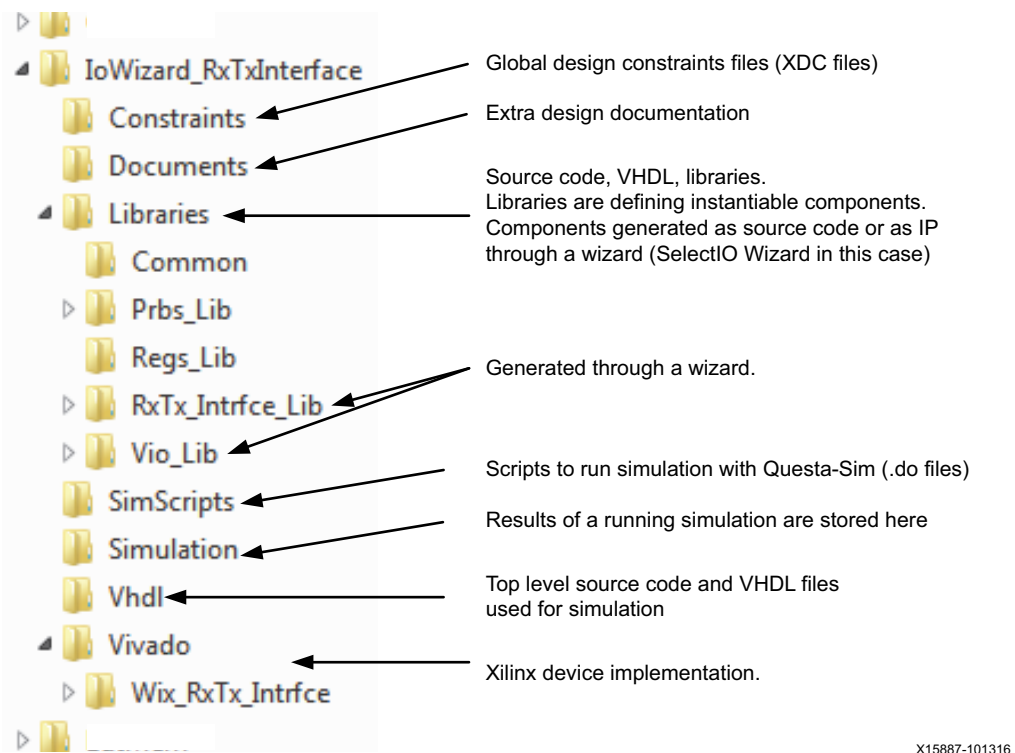
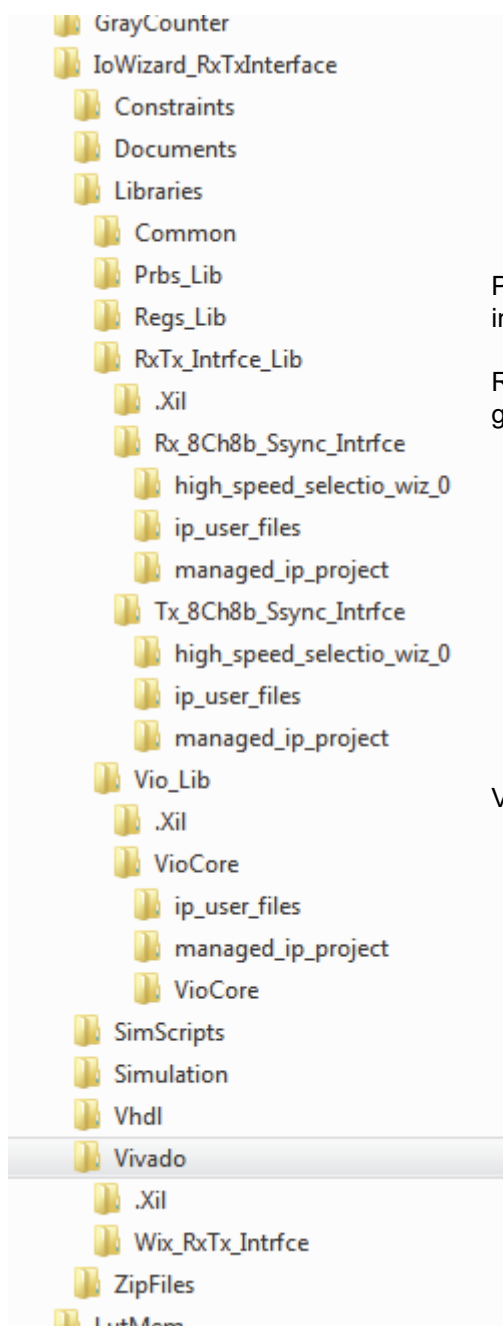


Figure 23: Directory Structure



Prbs\_Lib contains a PRBS generator and checker design that is instantiated in the VHDL top level source code.

RxTx\_Intrfce\_Lib contains the high-speed SelectIO Wizard generated transmitter and receiver IP cores.

Vio\_Lib contains a debug IP core generated from the IP catalog.

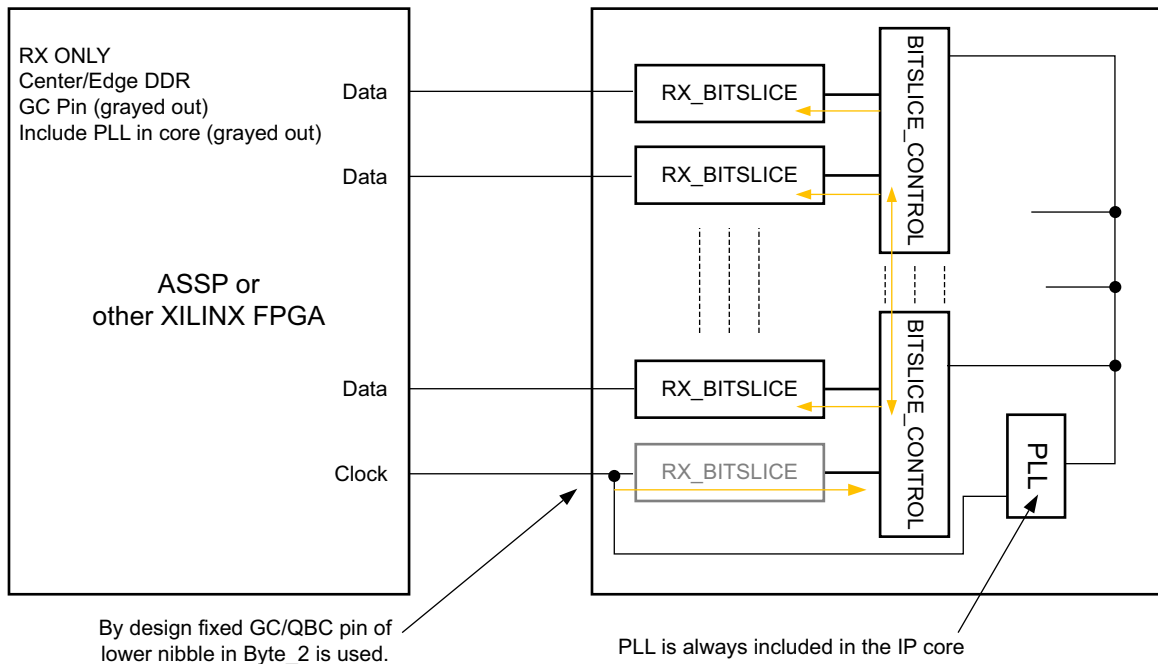
Vivado project for the top-level design implementation.

X16328-101316

Figure 24: Directory Structure Further Described

## High Speed SelectIO Wizard Clocking Structures

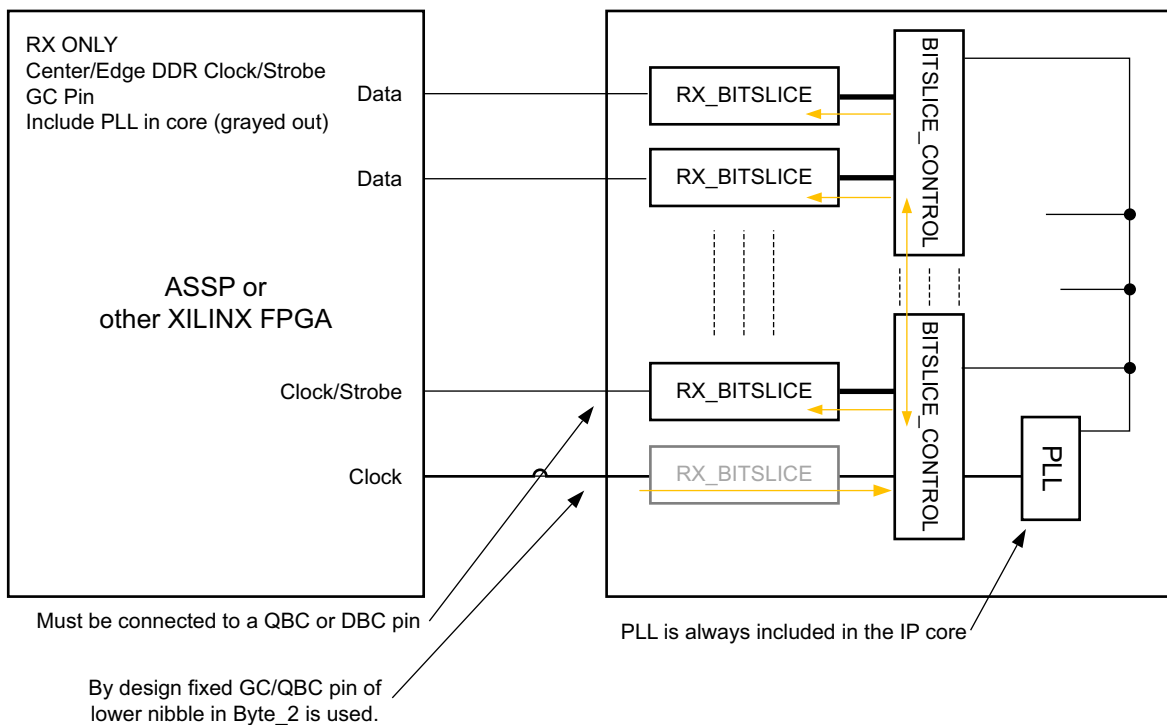
Figure 25 through Figure 28 outline some of the clocking structures used by the High Speed SelectIO Wizard.



X16803-101316

Figure 25: RX Only—Center/Edge DDR Interface

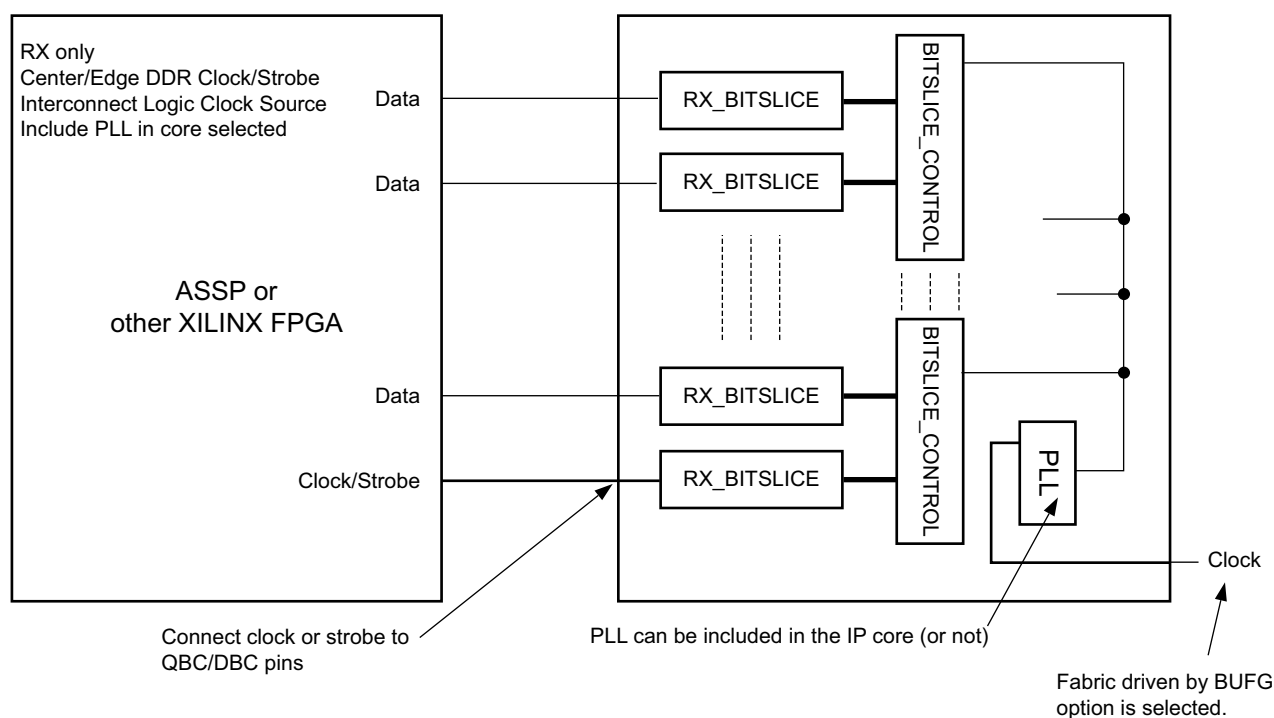
The black clock lines show the clock path in the FPGA to the PLL.CLKIN input. The orange clock lines show the clock path for data capturing. The clock input pin GC/QBC is used as dual function, being a global clock (GC) input for the PLL and a data capture clock capable to capture data in the four bytes of an I/O bank (QBC).



X16804-101316

Figure 26: RX Only—Center/Edge DDR Clock/Strobe with Global Clock Option





X16805-101316

**Figure 27: RX Only—Center/Edge DDR Clock/Strobe with Fabric Clock Option**

As in [Figure 25](#) the black and orange lines show the different clocks through the [Figure 27](#).

The global clock pin is used a single function while the data capture clock or strobe must be connected to one of the other quad or byte clock inputs in the I/O bank.

A clock or strobe applied to a quad byte clock input (QBC) is used to capture data in all four bytes of an I/O bank.

A clock or strobe applied to a dedicated byte clock (DBC) input can only be used to capture data in the byte where the pin is located.

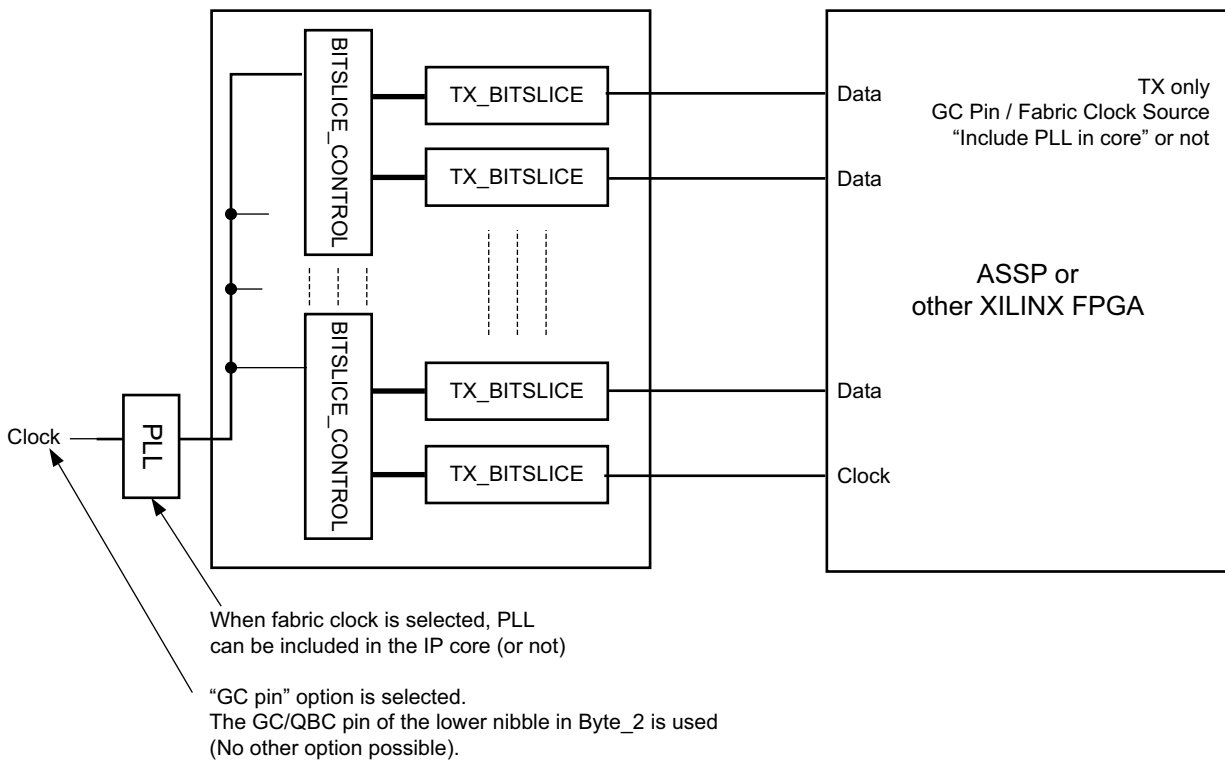


Figure 28: TX Only Interface

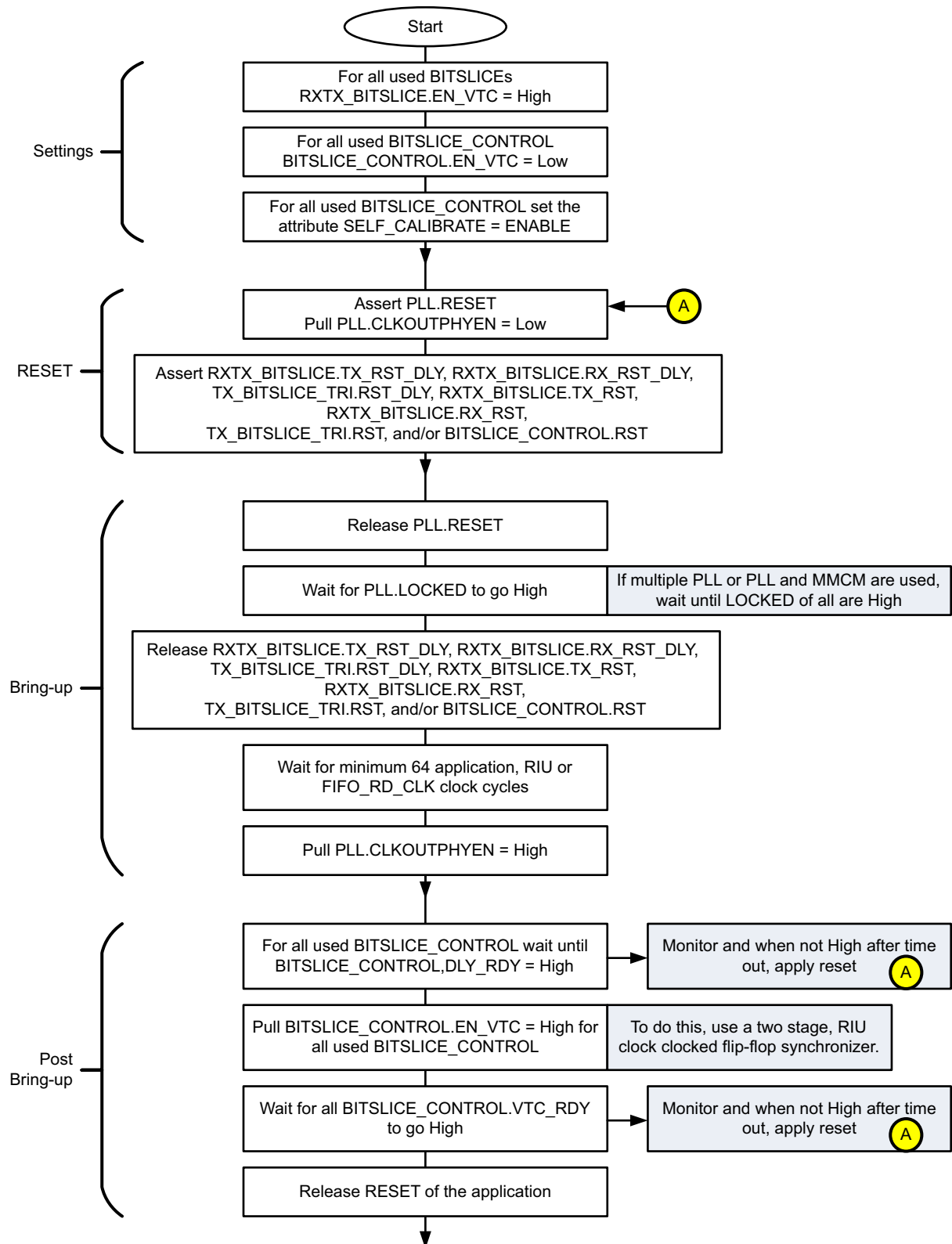
Each output of a transmitter interface can be used to generate a clock or strobe. The easiest way to generate a clock, generating a 50/50 clock, is to connect the inputs of a TX\_BITSLICE to a fixed pattern.



**TIP:** The D0 input of the TX\_BITSLICE will be output as the first serial bit.

## High Speed SelectIO Wizard Reset Sequence

The reset sequence (Figure 29) is described in the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [Ref 1] and used in the High Speed SelectIO Wizard.



X16807-101316

Figure 29: Reset Sequence

# Asynchronous Data Capture Interfaces

This section describes a method of capturing asynchronous communication using LVDS with native SelectIO interface primitives. Native SelectIO primitives are used along with the High Speed SelectIO Wizard. The method consists of a master-slave phase tracking algorithm that captures and tracks data bits using an unrelated sample clock generated from a PLL.

## Introduction to Asynchronous Data Capture

Synchronizing the clock and data is the most common method of achieving communication between devices using low-voltage differential signaling (LVDS). This means that the clock is transmitted on one differential channel and the data on one or several other differential pairs. At the receiver, the clock (after synchronization) is used to capture the data. This is known as source-synchronous communication.

When transmitting data without a separate accompanying clock signal, the clock used to capture the data must be recovered at the receiver side from the incoming data stream. This is called asynchronous communication.

Asynchronous communication systems tend to recover the clock to capture the data from the incoming data stream. This is known as clock recovery and Xilinx serial transceivers (GTH/GTY) use this principle. Data recovery allows a receiver to extract data from the incoming clock/data stream and then move the data into a new clock domain. Sometimes, the recovered clock is used for onward data treatment or transmission.

The circuit described in this section provides a *partial solution* in that no clock is actually recovered, but the arriving data is fully extracted using an unrelated clock generated from a PLL and a dedicated data tracking algorithm. This method is called *phase tracking*.

## Implementation Principle

In 7 series devices, asynchronous data capture uses a 4X oversampling technique (see *LVDS 4x Asynchronous Oversampling Using 7 Series FPGAs* (XAPP523) [Ref 8]). The ISERDES2 (in 7 series devices) can be configured in a mode where it acts as two independent DDR sampling registers. 4x oversampling can be achieved using the dedicated clock generation from a MMCM and IDELAY settings.

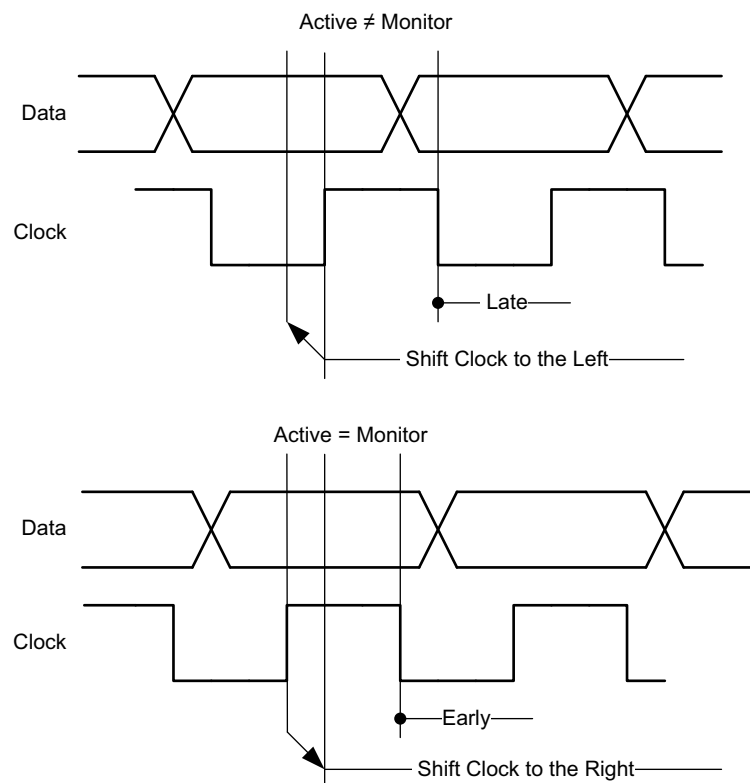
Oversampling using UltraScale architecture I/O logic is not possible because it is completely different from 7 series I/O. Oversampling cannot not be used in native mode or component mode. However, asynchronous data sampling in native mode can use *phase tracking*.

### Theory of Operation

Data is sampled by a DDR clock running at a frequency of half a unit interval (UI). The DDR clock is frequency related, but phase unrelated, to the captured serial data stream. To operate correctly, a data stream must be captured. Active or real data is sampled on the rising edge of the DDR clock, while monitor data is sampled on the falling edge of the clock. The sampling

process compares contiguously the active data to the monitor data and adjusts (when necessary) the position of the clock edge in the data bit to the optimal sample point, the middle of the data bit.

- When the active sampled data and the monitor data are not equal (sampled data and monitor data are two different data bits), then the rising clock edge (active edge) must be shifted to the left. In Figure 30, see the Active  $\neq$  Monitor diagram. The monitor samples late.
- When the active sampled data and the monitor data are equal (sampled data and monitor data are the same bit), then the rising clock edge (active edge) must be shifted to the right. In Figure 30, see the Active = Monitor diagram. The monitor samples early.
- The logic controlling the sampling will continuously shift the clock left and right to keep the position of the rising data sampling edge as close as possible to the middle of the data bit.
- Data wander or voltage and temperature adjustment are slow adjustment processes. Contiguously shift does not imply that the clock shifts up and down by each clock cycle, but the clock gradually shifts over time.



X17580-101316

Figure 30: Data Sampling Points

When data is wandering or changes due to voltage or temperature, one of these situations occurs and requires that the clock is shifted in one or the other direction to keep the active (rising) clock edge in the center of the data bit. This mechanism ensures that data and phase are tracked by the unrelated clock and always return the correct data value.

## Device Implementation

The data sampling clock is generated by a PLL, which means that the clock cannot be phase shifted as described in the [Theory of Operation](#). To obtain the same result, the captured data is to be shifted using the delay lines in the BITSLICEs.

The PLL generated clock is a DDR clock with a frequency equal to the data rate divided by two. The generated clock is the DDR clock used in source-synchronous designs. To obtain the half-UI sampling scheme discussed in the [Theory of Operation](#), data must be captured in two BITSLICEs, and one of them must have a starting delay (Figure 31).

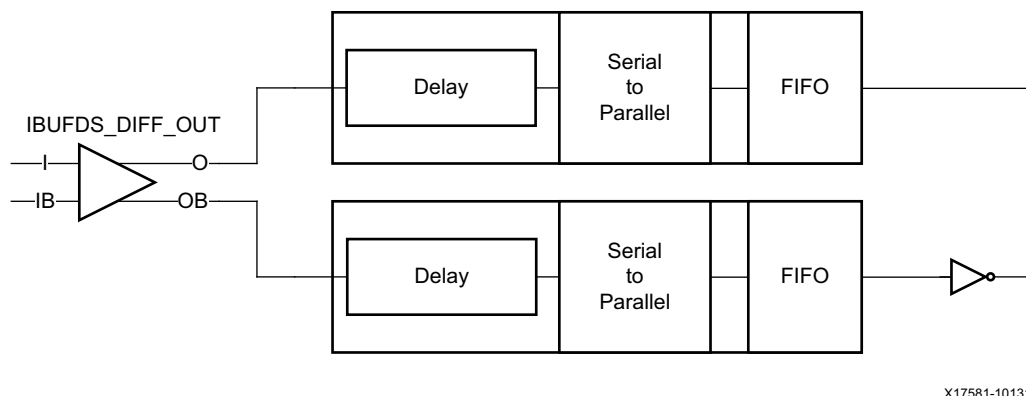
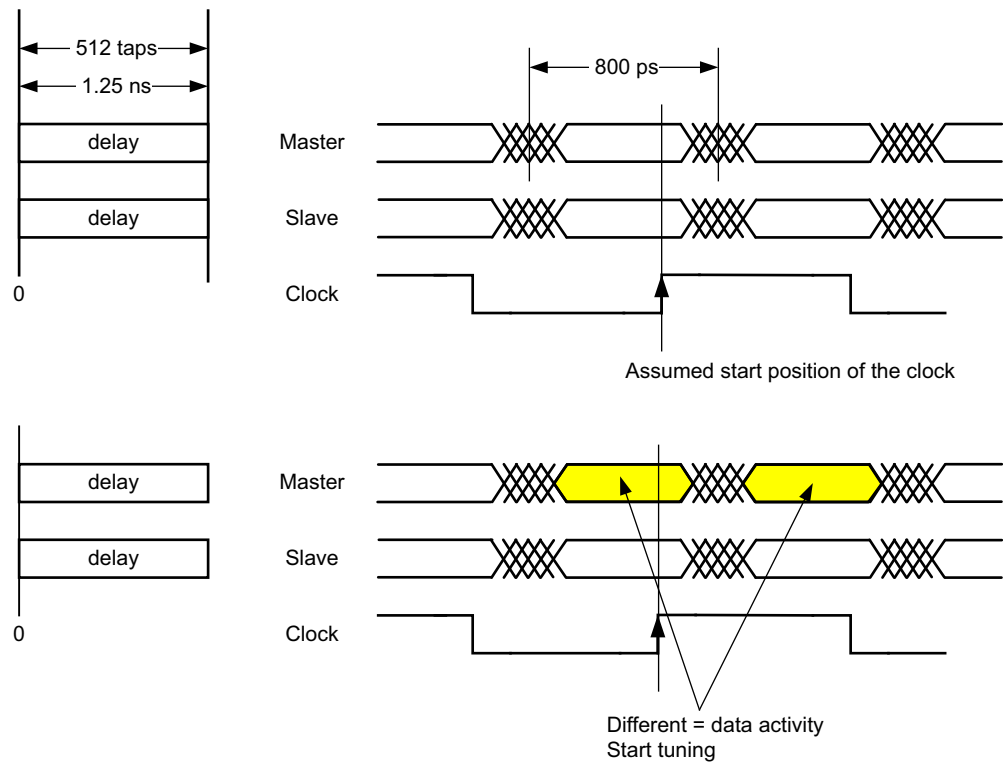


Figure 31: Data Capture Setup

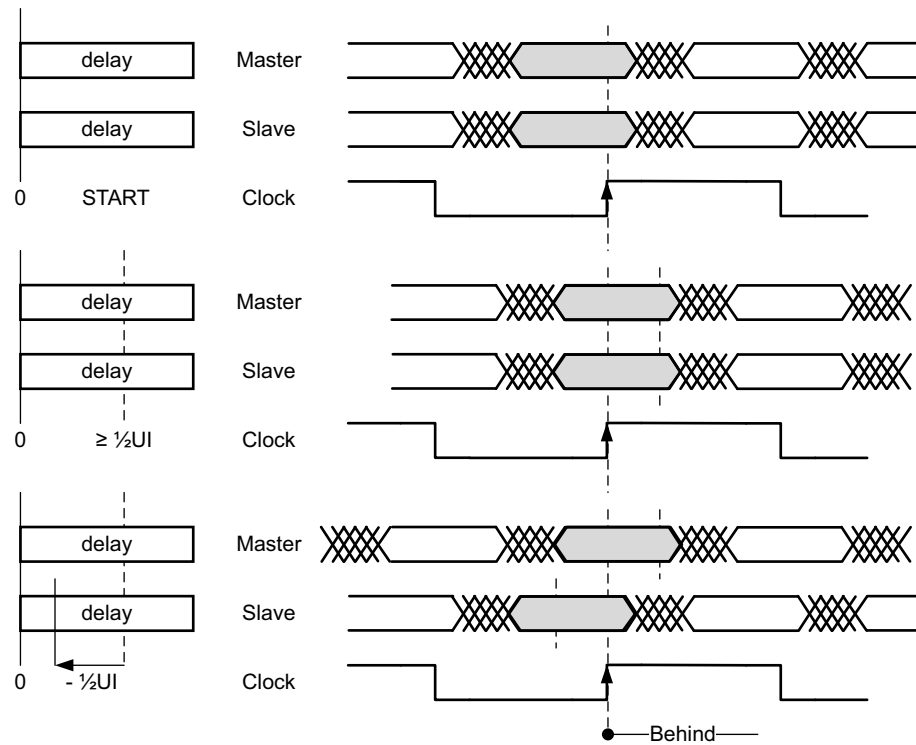
The setup functions are listed.

- The input data stream is captured in a p-side (master) and n-side (slave) BITSLICE.
- The PLL generated DDR clock is passed through the BITSLICE\_CONTROL having an unknown position with respect to the captured data (as shown in the top of [Figure 32](#)).
- The delay lines are used in COUNT mode. The amount of taps is calculated and the design works with these fixed values. Otherwise, the RIU interface of the BITSLICE\_CONTROL determines the exact tap delay per BITSLICE.
- Example: Asynchronous data capturing of a SGMII stream at 1250 Mb/s.
  - The DDR clock to capture the data would be 625 MHz.
  - The clock period is 1.6 ns, with a bit time of 800 ps.
  - Sampling must occur at  $\frac{1}{2}$ UI or 400 ps.
  - The amount of delay-line taps that fit in 400 ps can be measured using the BITSLICE\_CONTROL.RIU register interface by writing 0x0C to the register 0x38 and afterwards reading from the register 0x39 which provides the number of delay taps required for one UI. This information is used by the receiver logic for the phase tracking algorithm to know the wrap-around boundary.
- The master receiver channel checks for data activity.



**Figure 32: Initial Settings of the Delay Lines**

- Activity (bottom of [Figure 32](#)) shows how the bits captured by the channel are different and that the adjustment procedure of the data capturing channels can start. Activity is checked at the output of the BITSlice configured in 4-bit mode.
- The delay lines are incremented while the captured data in the master and slave channels are compared. When data is equal for  $\frac{1}{2}$  UI or after a UI, the master and slave positions will be set by incrementing or decrementing the slave delay line by  $\frac{1}{2}$  UI to appear as though the data is captured by a  $\frac{1}{2}$  UI clock. [Figure 33](#) and [Figure 34](#) show the positions of master and slave for different capture starting points.
- In reference to [Figure 33](#), it is assumed that the sample clock starts at the far end of the data bits captured in master and slave.
  - The delay lines are incremented by eight taps for a number of times.
  - A counter keeps track of the amount of increments.
  - Master and slave captured data are compared. As long as they are equal, the delays are incremented.
  - When the master delay line is incremented by  $\frac{1}{2}$ UI or more, the slave delay line is set back for  $\frac{1}{2}$ UI.

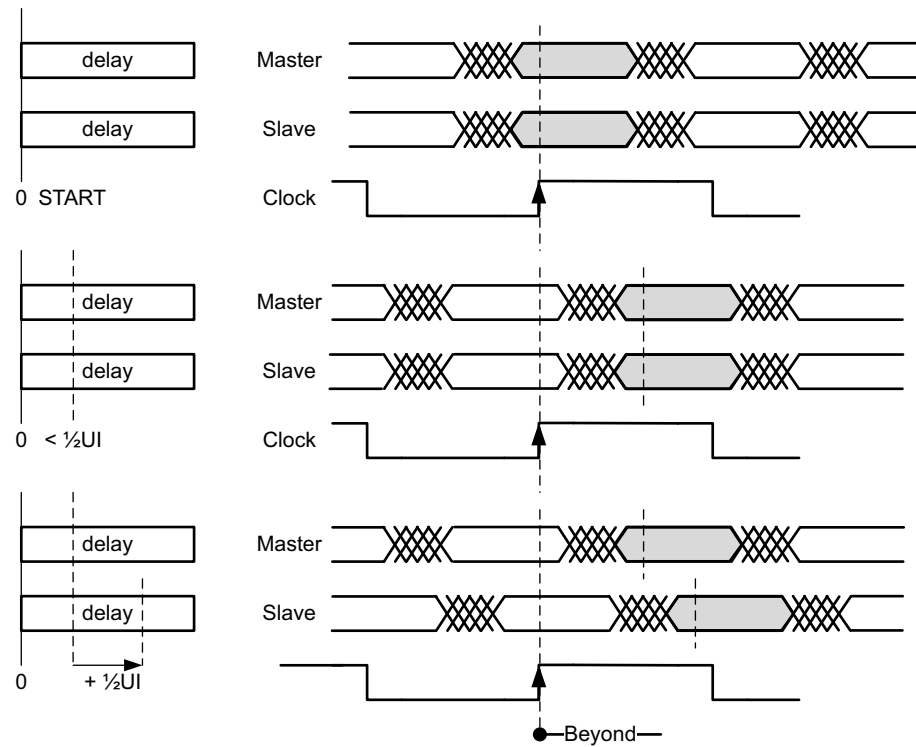


X17584-101316

**Figure 33: Half UI Adjustment Using Delay Lines, Slave Behind Master**

- In reference to [Figure 34](#), it is assumed that the sample clock starts at the beginning of the data bits captured in master and slave.
  - The delay lines are incremented by eight taps for a number of times.
  - A counter keeps track of the amount of increments.
  - Master and slave captured data are compared. As long as they are equal, the delays are incremented. In this case, the delays incrementation stops when passing through the jitter area between two bits. The master and slave data are no longer equal. The track counter keeps running.
  - When the master delay line is incremented by a value smaller than  $\frac{1}{2}UI$ , the slave delay line is set forward for  $\frac{1}{2}UI$ .

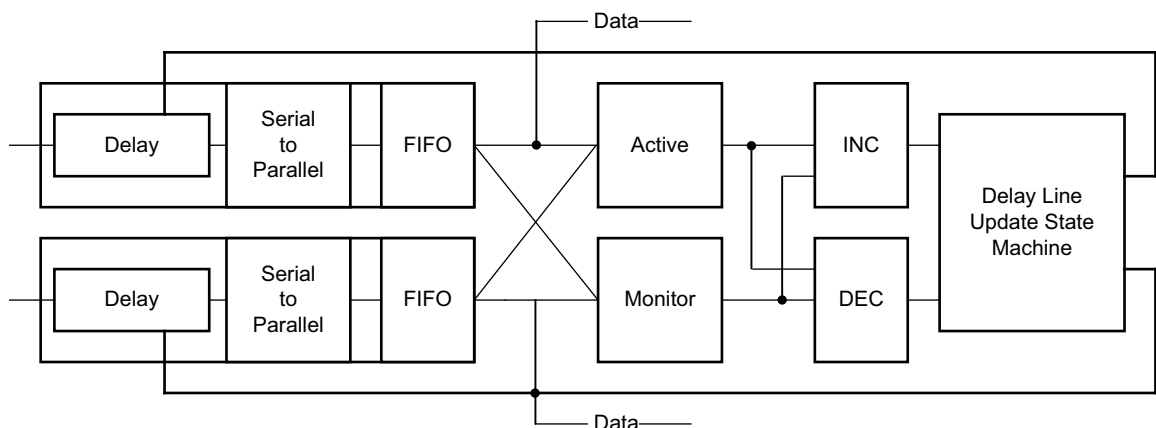




X17585-101316

Figure 34: Half UI Adjustment Using Delay Lines, Slave Beyond Master

- Incrementing or decrementing delay lines occurs in eight taps (specified in the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [Ref 1].
- At this stage, a state machine keeps the position of the data to the clock adjusted by incrementing or decrementing the delay lines.
- Delay lines wrap around, from 0 to bit-delay or bit-delay to 0. When this occurs the master and slave switch position, so that the adjustment can continue running.
- Figure 35 shows the adjust and tracking design. The data signals are fed into a set of registers before the data is passed into the four-to-ten gearbox design.



X17583-101316

Figure 35: Alignment and Tracking Circuit

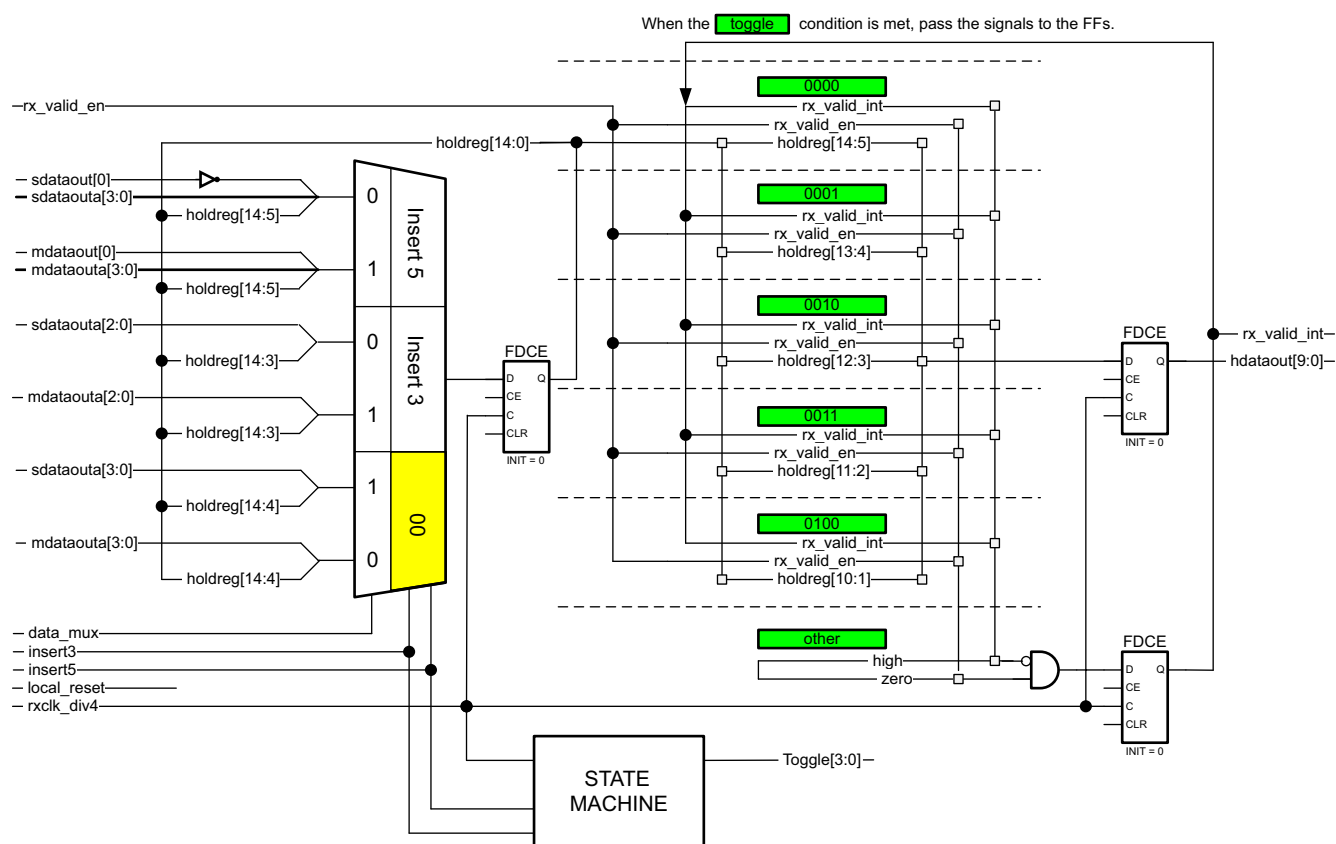
### ***Additional Required Logic***

Communication protocols, like SGMII, transmit 8B/10B encoded data using asynchronous data capturing techniques. The asynchronous receiver is capturing these 10-bit oriented streams four bits at a time. The ten-bit patterns must be restored after data is properly captured. A gearbox circuit is required to restore the patterns. Gearbox scaling receives 4-bit patterns up to the 10-bit patterns as they are transmitted.

There are many ways to construct a gearbox, in this reference design the gearbox is built as a rotating 15-bit shift register where the LSB bits are refreshed every clock cycle with new captured data.

The 15-bit register is updated three times with 4 bits and one time with 3 bits from the master or slave. At the switching point of master to slave, or vice versa, direct data from the master and slave must be loaded, and then the five LSB bits are updated.

A small state machine running with the shift register selects the required 10-bits from the 15-bit shift register output. See [Figure 36](#).



**Figure 36: 4-bit to 10-bit Gearbox**

The 10-bit output of the gearbox (and also the output of the reference design) can be used as is or applied to the 10B/8B conversion logic. This conversion logic is generated using the IP-catalog in the Vivado Design Suite.

## Interface Details

Although that the reference design contains transmitters, only the details of the receiver data capture are discussed in this section. The implementation for the transmitters is the same as the transmitter implementation for source-synchronous designs. The only difference is that no clock is transmitted or forwarded.

### ***BITSLICE\_CONTROL***

The RX\_BITSLICE only receives data. The clock for capturing the data must be generated by a PLL in the UltraScale architecture-based device. To apply this PLL generated clock to the RX\_BITSLICE as a sample clock, the BITSLICE\_CONTROL-SERIAL\_MODE attribute must be set to TRUE. With this set, the BITSLICE\_CONTROL.PLL\_CLK clock input is used as the data sampling clock.

To have BISC run at boot up of the circuit, enable the SELF\_CALIBRATE. The finalized initial BISC run is flagged by DLY\_RDY and VTC\_RDY. Both signals are used as status inputs in the reset and bring-up logic.

### **BITSLICE**

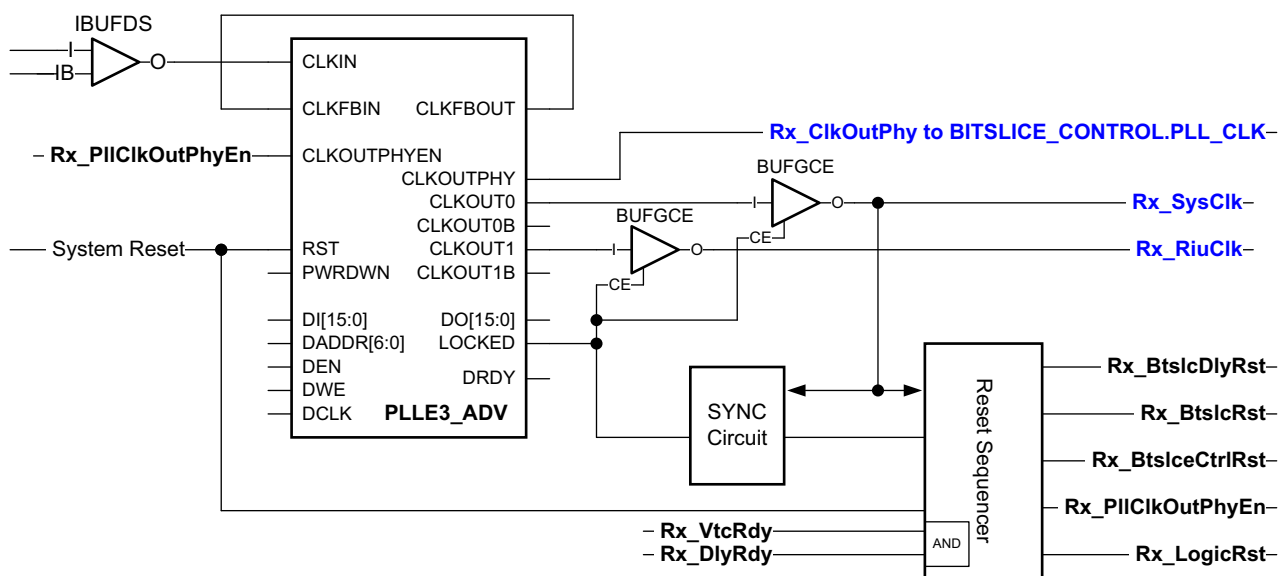
The BITSLICES connected to a BITSLICE\_CONTROL function correctly with this parameter set.

- RX\_DATA\_WIDTH = 4
- DATA\_TYPE = SERIAL
- DELAY\_FORMAT = COUNT: Change the delay line on a per tap base.
- DELAY\_TYPE = VAR\_LOAD: Use CNTVALUEIN[8:0] and LD to alter the delay lines.
- DELAY\_VALUE = 0: Start the delay line from delay-tap zero.

### **Clocking**

The data sample clock for the RX\_BITSLICES must be generated by a PLL (Figure 37). The PLL must be one of the two PLLs in the I/O bank containing the asynchronous data capturing interface. The input or source clock for the PLL can come from the following clocks.

- A global clock (GC) input in the same I/O bank that is used for the data inputs of the design. This setup generates the best results because the clock, data, RX\_BITSLICES, BITSLICE\_CONTROL, and PLL are all part of the same power-rail in the UltraScale architecture-based device. This setup is used to qualify the reference design, a SGMII interface.
- A clock generated from an MMCM, a clock input in a different I/O bank than that used for the asynchronous data sampling interface, or other. This setup using the MMCM is not included in the reference design. The results of this setup depend on the clock source used to feed the PLL and are not characterized.



X17587-101316

Figure 37: PLL and Reset Sequencer

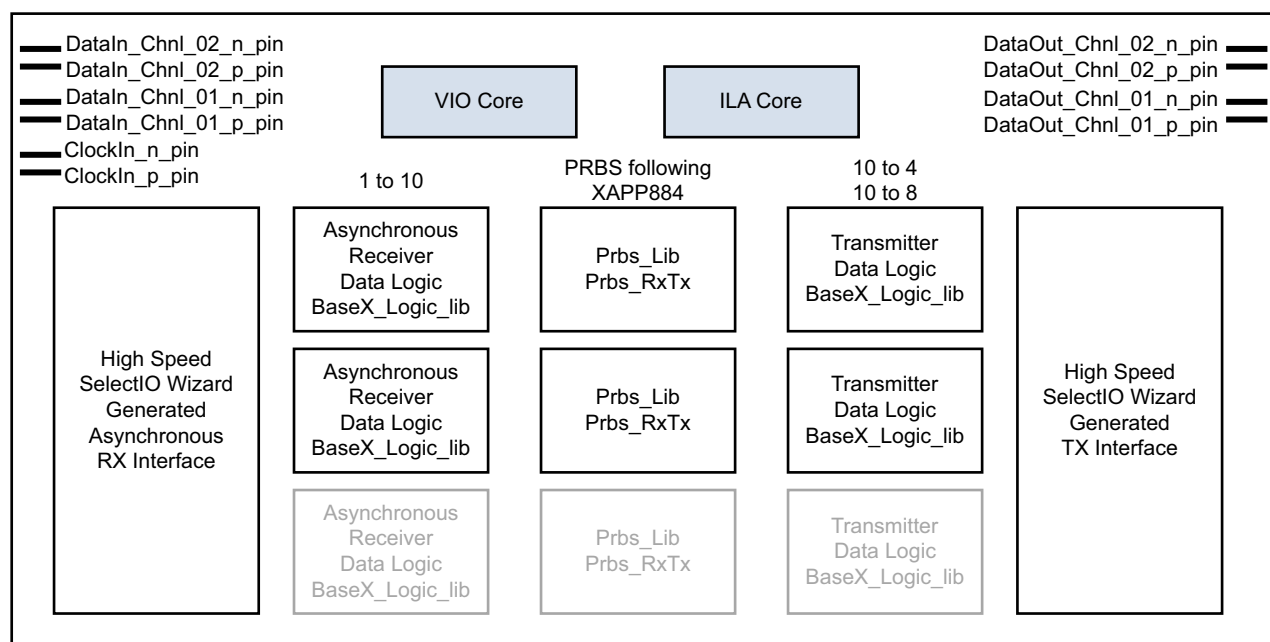
The PLL.CLKOUTPHY clock output connects, without a clock buffer, directly to the BITSlice\_CONTROL.PLL\_CLK input clock. Since this clock is used as sample clock, it must run at a speed equal to a data rate related DDR clock. For example, for an SGMII receiver running at 1250 Mb/s, the generated PLL clock must be 625 MHz.

The second PLL in the I/O bank can be used as a clock generator for the asynchronous data transmitters. An asynchronous data transmitter is similar to a source-synchronous transmitter. The only difference for an asynchronous transmitter interface is that no clock is forwarded together with the data.

The clock generated by a PLL for the BITSlice\_CONTROL, the PLL\_CLK clock input of a transmitter, must run at the same frequency as the required serial transmitted data rate. For example, for an SGMII transmitter at 1250 Mb/s, the PLL must generate a PLLCLKOUTPHY of 1250 MHz.

### Setup of the Interface

Figure 38 shows in a block diagram how the design is constructed. The different elements in the diagram are found as libraries or folders in the reference design structure shown in Figure 41.



X17588-101316

Figure 38: Asynchronous Data Capture Design Block Diagram

## Cores Generated using the High Speed SelectIO Wizard

The portion of an I/O design can generate the necessary I/O cores by using instantiated primitives and following the guidelines outlined in this application note or by using the High Speed SelectIO Wizard. The High Speed SelectIO Wizard can only generate the I/O portion including the reset sequencer and PLLs. The rest of the design is implemented in the internal interconnect logic and is not part of the generated cores.

To create full transmit and receive channels, two separate cores must be generated.

- Receiver(s)
  - The primitives for this kind of receiver must be configured in SERIAL mode, which is translated by the wizard using the clocking option [ASYNC/NONE].
  - For a source-synchronous design, a phase aligned or 90° shifted bit clock is provided with the data. This clock supplied to any quad byte clock (QBC) or dedicated byte clock (DBC) can be used as a data-capture clock. For asynchronous data capturing designs (SERIAL mode), no clock is supplied with the data. Data must be captured by an unrelated, frequency correct DDR clock, where the frequency of the clock must match  $\frac{1}{2}$  the data rate. For example, when data at a rate of 1250 Mb/s must be captured, an unrelated clock of 625 MHz must be supplied to the I/O primitives. One of the two PLLs in the I/O bank can generate this clock.

- Transmitter
  - A general transmitter can be constructed for source synchronous or other design constructs. The transmitter does not generate a clock as a data capture clock for use on the receiver side. Transmitters in this design case only need to generate data.
  - The clock required for generation of the data must have the same frequency as the required data rate. For example, when a data rate of 1250 Mb/s is required, the clock for the transmitter primitives must be 1250 MHz. This clock can be generated from one of the two I/O bank available PLLs.
  - For the best performance, the input clock for these PLLs should be either a clock supplied to the GC/QBC pin in the I/O bank or a clock supplied from the internal interconnect logic. [Figure 39](#) shows a transmitter and receiver channel and their respective PLLs.

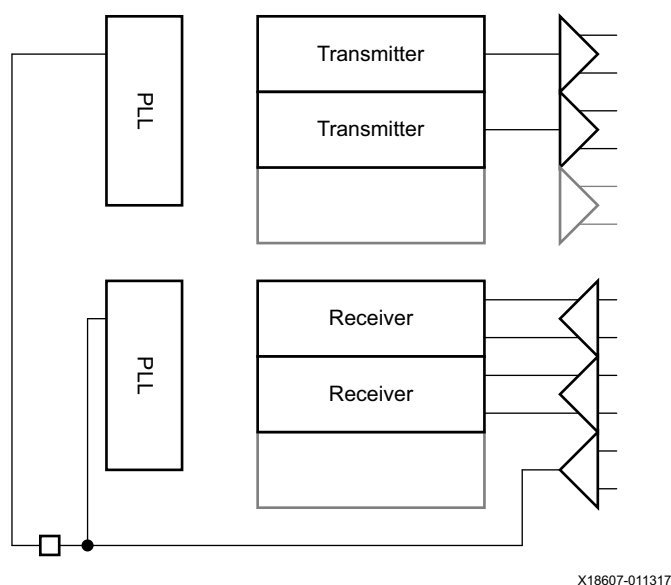


Figure 39: Transmitter and Asynchronous Receiver with PLLs

Transmitters and receivers use their own PLL. This means that the High Speed SelectIO Wizard options [BIDIR, TX+RX, or TX+RX+BIDIR] cannot be used, because using these options assumes that both transmitters and receivers operate PLLs at the same frequency. For this asynchronous design, two IP cores are generated, one core for the receivers and one core for the transmitters. The input clock at the GC/QBC pins is passed through one core to serve as PLL clock input for the PLL of the other IP core.

An IP core comprises a nibble of a byte. The lower nibble of the byte is occupied by the receivers and the upper nibble of that same byte is occupied by the transmitters ([Figure 39](#)). By choosing the correct pin selection boxes in the **Pin Selection** tab of the High Speed SelectIO Wizard.

Because three differential RX/TX channels can fit in a byte, an I/O bank (four bytes) can contain up to 12 RX/TX channels. For maximum performance and minimum jitter, an unrelated clock is supplied to the GC/QBC pin of the I/O bank. By using an unrelated clock, one of the RX/TX channels must be replaced by a clock input, limiting the I/O bank to 11 differential RX/TX channels.

By following the [I/O Assignment Rules](#), the receiver's IP core also contains the unrelated clock input (single-ended or differential) and passes this clock through the core to the transmitter IP core. The setup is shown in [Figure 39](#).

To fill an entire I/O bank with RX/TX channels, three IP cores must be generated and properly placed in the I/O bank, see the High Speed SelectIO Wizard for placement. The following cores must be generated.

- A nibble with two RX data channels and a clock input placed into the byte\_2 location of the I/O bank, where the GC/QBC pin or pin-pair is located.
- A nibble with three RX data channels. This nibble can be used for bytes 0, 1, and 3.
- A nibble with three TX channels. One channel is not used in byte 2.

**Note:** To create a design occupying multiple bytes, create an RX core containing all receiver nibbles and a core containing all transmitter nibbles in the High Speed SelectIO Wizard. Assemble both cores in a top-level HDL file. See the [Reference Design \(IoWizard\\_AsyncDataCapture\)](#) section.

The reference design is designed to fit in byte 2 of an I/O bank and comprises two data channels and a clock input (differential).

- High Speed SelectIO Wizard Generated receiver core.
- The RX core fits in the lower nibble of byte 2. It operates from its own PLL, while the input clock for both PLLs is supplied by a clock input at the QBC\GC pin in byte 2 (BITSlice\_0, lower nibble). The necessary reset sequencer is also generated by the High Speed SelectIO Wizard.
- The following settings and tick box activations generate the receiver core.

<Name\_of\_Rx\_Interface> configuration in the High Speed SelectIO Wizard v3.1.1

- [Basic]

[Bus Direction]: RX ONLY

[Rx External Data and Clock] : ASYNC/NONE

[PLL clock source]: Clock capable pin

Tick: Access IBUF clock output

[Interface Speed]: 1250 Mb/s

[PLL input Clock]: 625 MHz

[PLL CLKOUT0]: 325.50 MHz

[PLL phase shift mode]: Waveform

[Bank]: 68

[Serialization Factor]: 4

Tick: RIU interface

- [Advanced]

[RX Delay Mode]: COUNT

[RX delay type]: VAR\_LOAD

[RX Delay Value]: 0

Tick: Enable PLL CLKOUT1 156.250 MHz

Tick: Generate RIU clock from PLL

Tick: Enable n-side RX BITSlice

[Differential I/O std]: LVDS

- [Pin Selection]

- Select byte group 2

- Select pins 26, 28, and 30. This enables automatically pins 27, 29, and 31. Pins 26 and 27 are clock inputs. Pins 28, 29, and 30, 31 are receiver data channels.

- Click [OK] to generate the receiver core.

The reference design is designed to fit into byte 2, where the clock input is fit into that byte and thus it is necessary to remove one TX channel (keep two).

- High Speed SelectIO Wizard generated transmitter core.

- The TX core fits in the upper nibble of byte 2. It operates from its own PLL, while the input clock for both PLLs is supplied by a clock input at the QBC\GC pin in byte 2 (BITSlice\_0, lower nibble). The clock is passed through the receiver core and then used as an input for the PLL in the transmitter. The necessary reset sequencer is also generated by the High Speed SelectIO Wizard.

- The following settings and tick box activations generate the transmitter core.

<Name\_of\_Tx\_Interface> configuration in the High Speed SelectIO Wizard v3.1.1

- [Basic]

[Bus Direction]: TX ONLY

[PLL Clock Source]: Fabric (driven by the BUFG)

[Interface Speed]: 1250 Mb/s

[PLL input clock]: 625 MHz

[PLL CLKOUT0]: 156.250 MHz

Tick: Include PLL in core

[Bank]: 68

[Serialization factor]: 8

Tick: RIU interface



- [Advanced]

[TX Delay Type]: FIXED

[TX Delay Value]: 1

Tick: Enable PLL CLKOUT1 125.00 MHz

[Differential I/O std]: LVDS

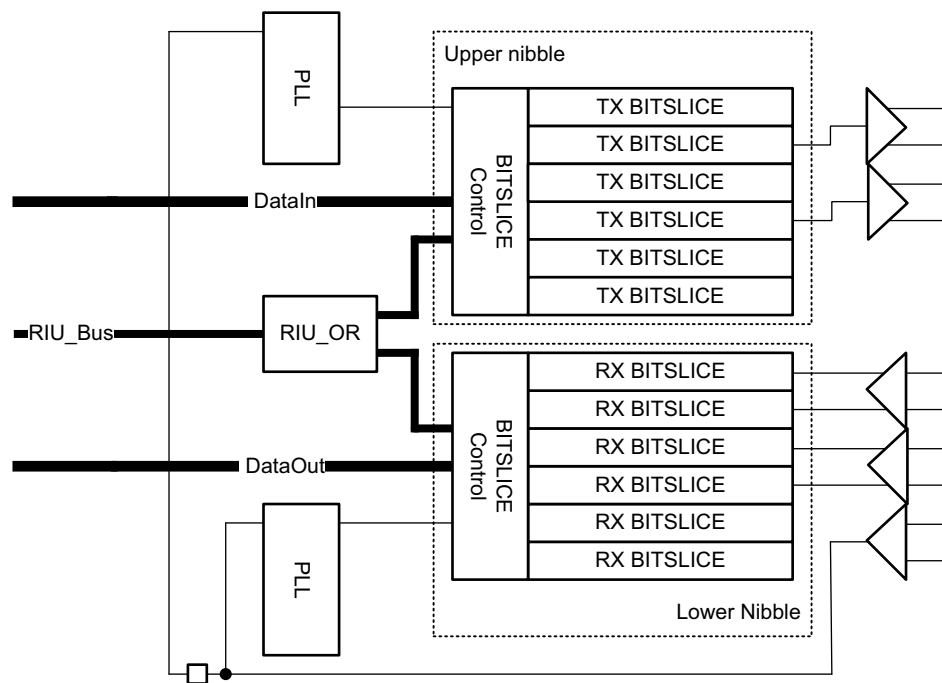
- [Pin Selection]

- Select byte group
- Select pins 34, 35, 36, and 37
  - Change the signal type on each of the pins to differential.
  - 34, 35, and 36, 37 are transmitter data channels.

- Click [OK] to generate the transmitter core.

A delay calibration state machine uses the RIU interface to read the delay of a single tap of the receiver. This process is necessary to start the interface with a correctly placed clock and to track for voltage and temperature variations.

Although only the RX side RIU interface is used, both cores are generated with an RIU interface (obligation). On the level where both cores are combined, an RIU\_OR component is added to allow the state machine of the receiver can get the tap delay value. The full and assembled RX/TX interface is shown in [Figure 40](#).

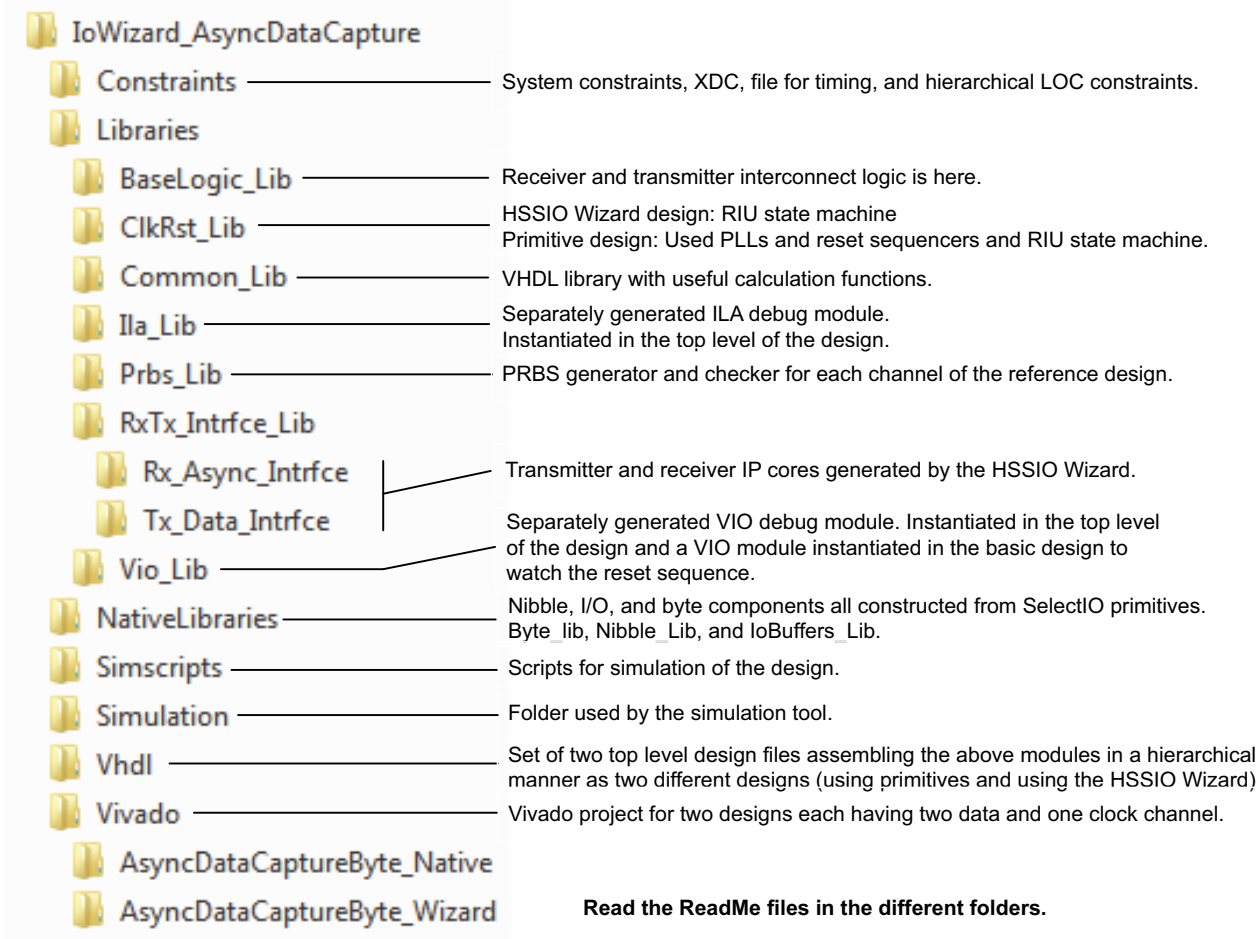


X18608-011317

Figure 40: Assembled Transmitter and Asynchronous Receiver IP Cores



**IMPORTANT:** The High Speed SelectIO Wizard in Vivado versions 2016.3 and 2016.4 does not have the tick box option to pass the input clock from the GC/QBC pin(s) to the output of the IP core. To use this function, follow the guideline documents linked from the High Speed SelectIO Wizard—Known Issues List (AR68404) [Ref 7].



X17591-121816

Figure 41: Asynchronous Data Capture Reference Design Folder

The generated reference design comprises all the logic for three RX/TX data channels or two RX/TX data channels and a clock input. The design fits into an I/O byte of an UltraScale architecture-based device. The upper nibble contains the transmitters and the lower nibble contains the receivers.

Using the High Speed SelectIO Wizard, a receiver core can be created for the native mode logic for the asynchronous data capture, and a transmitter core can be created for the native mode logic for the data transmitter. For more information on the PHYs, see the [High Speed SelectIO Wizard](#) section. Each generated (RX and TX) core contains all necessary components (PLL, BITSlice\_CONTROL, BITSlices, and bring-up logic) to form an RX and TX PHY. Behind each receiver channel, there is alignment logic and tuning for a one to 10-bit gearbox. In front of each transmitter fits a ten-to-four or ten-to-eight gearbox. The transmitter gearbox is fed by the output of a PRBS generator, while the receiver gearbox feeds a PRBS decoder. The results of PRBS transmission and reception are shown in the Vivado tools using a connected ILA and VIO interface.

## I/O Assignment Rules

Each UltraScale architecture-based device's SelectIO bank contains four bytes, and each byte contains an upper and lower nibble. The lower nibble has six and the upper nibble has seven usable RXTX\_BITSLICES. The RX\_BITSLICES and TX\_BITSLICES are subsets of this RXTX\_BITSLICE. A differential I/O uses two of these BITSlices. There can be three differential receiver or transmit channels in a nibble. All used BITSlices in a nibble must be connected to the BITSlice\_CONTROL of that nibble.

A transmitter and receiver core can be generated with the High Speed SelectIO Wizard. Each core needs a PLL generating the necessary clocks for the PHY and logic. Because the BITSlice\_CONTROL has one PLL\_CLK input, the transmitters and receivers must fit in their own nibble is shown in [Figure 42](#).

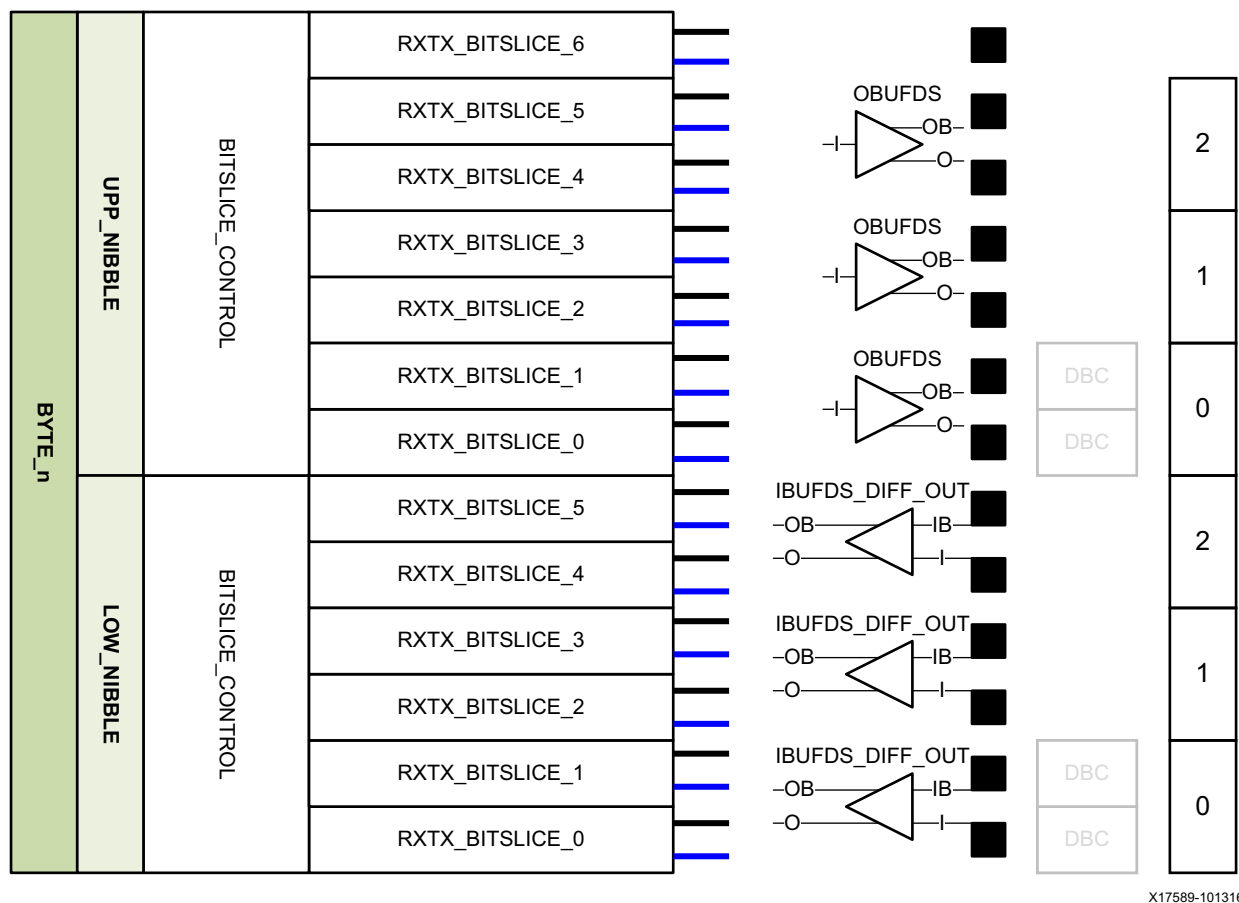
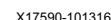


Figure 42: Byte Setup as Three Data Channels

[Figure 42](#) shows a byte configured as three transmit and three receive channels. As described in the [Clocking](#) section, the clock input for an asynchronous data capturing interface should be in the same I/O bank. However, this would mean housing three data channels in one byte. By using a differential input as the clock input, the byte can then contain two RX/TX data channels and one clock input as shown in [Figure 43](#).



If a full I/O bank (four bytes) must be configured for asynchronous data sampling, then three bytes can contain three RX/TX data channels and one byte can contain two RX/TX data channels and the differential clock input. A fully configured I/O bank can be the home of eleven RX/TX data channels and one differential clock input. The clock input **MUST** be fixed onto a GC input pin pair. In most cases, the clock input is fixed onto the only pair in the I/O bank with QBC/GC (Byte\_2, lower-nibble) capabilities.

The High Speed SelectIO Wizard does not work with nibbles and bytes but with pin numbers. Consult the ASCII Package Files section in the *UltraScale and UltraScale+ FPGAs Packaging and Pinouts Product Specification* (UG575) [Ref 9].

1. Download the ASCII file for the specific device/package combination.
2. The ASCII file has different columns. In the bank column, browse down to the I/O bank selected by the design.

3. The byte and nibble combinations are listed in a memory byte group column as [byte number]U (upper nibble) and/or L (lower nibble). For example using I/O bank 44:
  - Top byte 3U, 3L
  - Top middle byte: 2U, 2L
  - Bottom middle byte: 1U, 1L
  - Bottom byte: 0U, 0L
4. Select the byte or bytes the interface must fit into.
5. Match byte and nibble information to the I/O pin numbers of the ASCII file, and enter them in the High Speed SelectIO Wizard.

The *Pin Name* column in the ASCII file lists all functionality of a selected IO pin. For example, the extracted ASCII file for a xcku040ffa1156 shows the following.

```

Pin                AJ21
Pin Name           IO_L13P_T2L_N0_GC_QBC_44
Memory Byte Group  2L
Bank               44
I/O Type           HP
Super Logic Region NA
No-Connect         KU11P
  
```

This is the P-side of the 13th pin pair in high-performance (HP) I/O bank 44. The pin fits in the lower nibble of byte\_2 (2L) and the function of the pin, besides normal I/O, is the global clock (GC) and/or quad byte Clock (QBC)

Consult the *Pin Definitions* section in the *UltraScale and UltraScale+ FPGAs Packaging and Pinouts Product Specification* (UG575) [Ref 9] for information on multi-function pins.

## Reference Design (IoWizard\_AsyncDataCapture)

You can download the [Reference Design Files](#) for this application note from the Xilinx website. [Table 11](#) shows the reference design matrix.

**Table 11: Asynchronous Data Capture Reference Design (IoWizard\_AsyncDataCapture)**

Parameter	Description
<b>General</b>	
Target Device	XCV095-2
Source Code provided	Yes
Source Code Format	VHDL and Verilog
IP used	Yes
<b>Simulation</b>	
Functional simulation performed	Yes, on the hierarchical parts of the design
Timing simulation performed	No
Testbench format	VDHL

Table 11: Asynchronous Data Capture Reference Design (IoWizard\_AsyncDataCapture) (Cont'd)

Parameter	Description
Simulator software/version	QuestaSim-10.3b
SPICE/IBIS simulations	No
<b>Implementation</b>	
Synthesis tool/version	Vivado 2016.2 or later
Implementation tool/version	Vivado 2016.2 or later
Static timing analysis performed	Yes
<b>Hardware Verification</b>	
Hardware verified	Yes
Hardware platform used for verification	UC SGMII Validation platform

## Asynchronous Data Capture Reference Design Directory Setup

The directory structure for the reference designs (ILA and VIO cores) are shown in [Figure 41](#).

---

## Bitslip

The Xilinx application note, *Bitslip in Logic* (XAPP1208) [\[Ref 10\]](#) outlines managing bitslip in a native I/O primitive.

---

## Conclusion

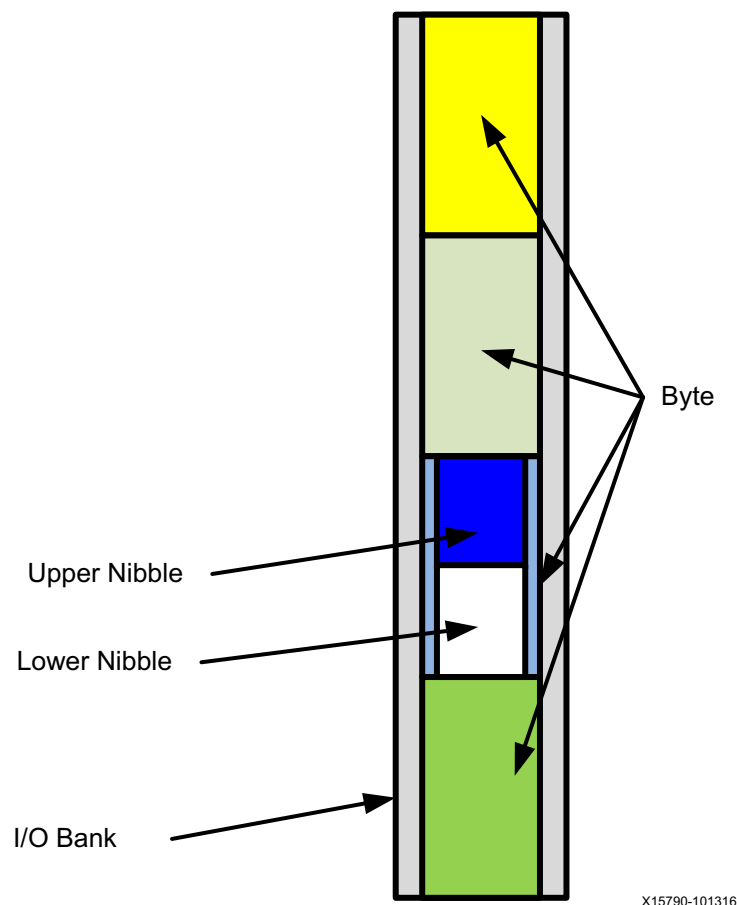
This application note and the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [\[Ref 1\]](#) supplies a further understanding of the functions available when using the UltraScale architecture SelectIO resources in native mode. The High Speed SelectIO Wizard is discussed and examples are given for generating a high-speed native I/O interfaces. In the asynchronous data capture interface reference design, a phase-tracking algorithm is used to capture high-speed asynchronous data. These different examples are used in real applications to show how to generate I/O interfaces using the High Speed SelectIO Wizard.

## Appendix A

When used in native mode, RX\_BITSLICE, TX\_BITSLICE, or RXTX\_BITSLICE are the smallest elements of an UltraScale device I/O. These BITSLICES are combined in bigger elements referred to as a nibble. Two nibbles are combined in a byte. A byte contains a lower and an upper nibble. The lower nibble has six BITSLICES and the upper nibble has seven BITSLICES.

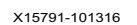
Each nibble, either six or seven BITSLICES, needs a BITSLICE\_CONTROL primitive. This BITSLICE\_CONTROL primitive contains all logic to monitor and adjust the clock, perform voltage and temperature compensation (VTC), and other tasks for connected BITSLICES.

Every UltraScale device has exactly the same nibble, byte, and I/O bank composition, see [Figure 44](#). The entire generation of generic interfaces can be used in one or multiple UltraScale devices.

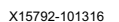


**Figure 44: Layout of an UltraScale Device I/O Bank**

[Figure 45](#) shows a generic nibble. There are two sizes of nibbles, a lower nibble with six BITSLICES and an upper nibble with seven BITSLICES. [Figure 46](#) shows the combination of both nibbles into one byte. The nibble and byte display RXTX\_BITSLICES. This is the basic BITSLICE structure and from the RXTX\_BITSLICE two derivatives are created, the RX\_BITSLICE and TX\_BITSLICE. A TX\_BITSLICE\_TRI BITSLICE is used in a serial transmitted bit stream for per bit 3-state functionality.



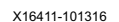
*Figure 45:* **Layout of a Nibble**



**Figure 46: Layout of a Byte in an I/O Bank**



Figure 47 shows a block diagram of an upper nibble in a byte.

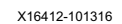


**Figure 47: Upper Nibble in a Byte**

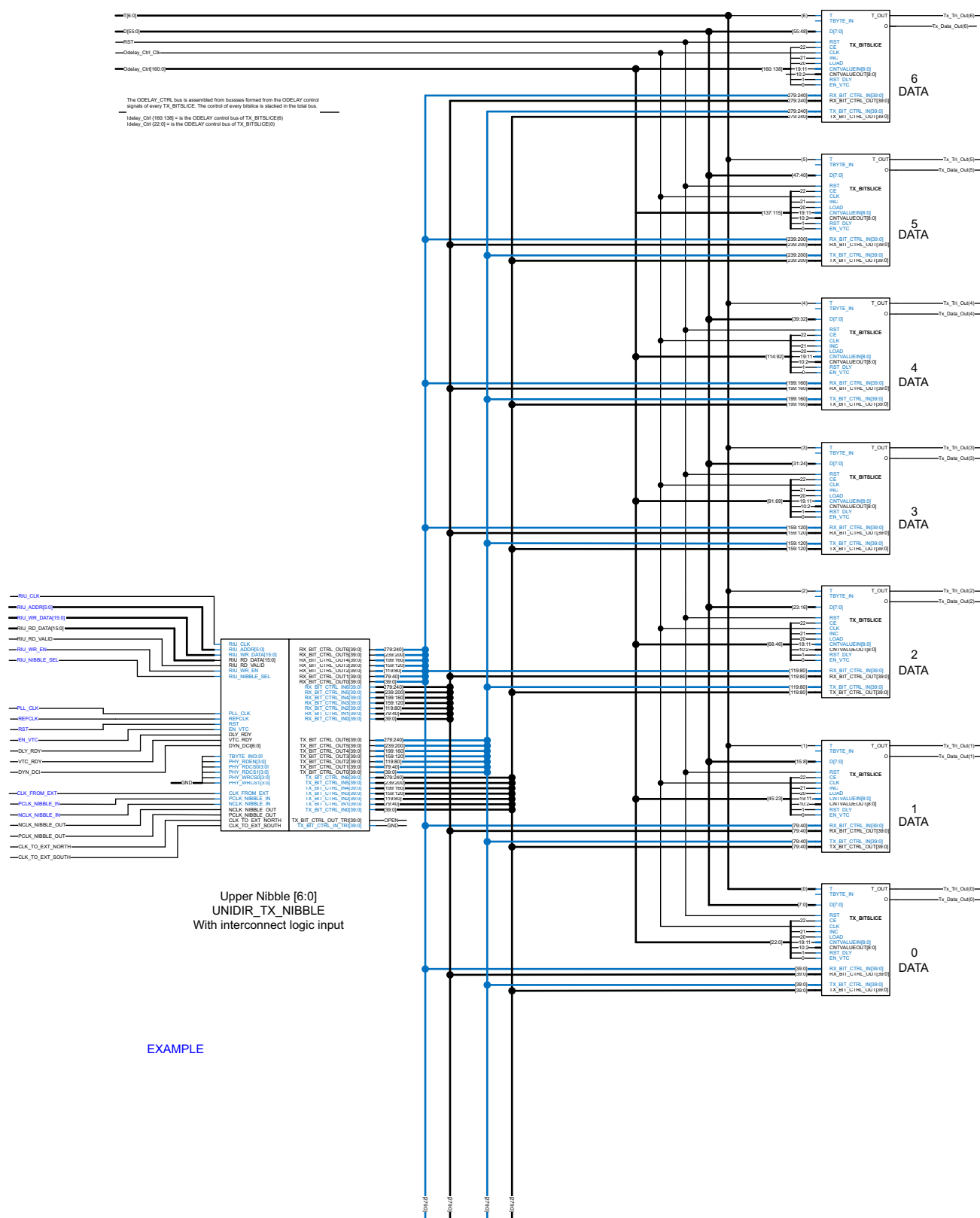
Figure 48 shows a detailed view of a fully configured receiver upper nibble.

Figure 49 shows a detailed view of a fully configured transmitter upper nibble using a 3-stated input (where the T-inputs are used).

Figure 50 shows a detailed view of a fully configured transmitter upper nibble using a 3-stated input from a serial 3-state option (where TBYTE\_IN inputs at the BITSlice\_CONROL are used).

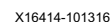


66



X16413-101316

Figure 49: Fully Configured Transmitter Upper Nibble using a 3-stated Input



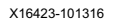
68

The diagram illustrates the timing relationship between the ATmega328P and the ATmega16 during an SPI transfer. The ATmega328P's SS pin is connected to the ATmega16's CS pin. The ATmega328P's SCK pin is connected to the ATmega16's SCK pin. The ATmega328P's MOSI pin is connected to the ATmega16's MISO pin. The ATmega16's MISO pin is also connected to the ATmega328P's MISO pin. The diagram shows the timing of the SS, SCK, and CS signals, as well as the data transfer between the two microcontrollers. The timing parameters are defined as follows:

- $t_1$ : Time from SS to SCK
- $t_2$ : Time from SCK to CS
- $t_3$ : Time from CS to SCK
- $t_4$ : Time from SCK to MISO
- $t_5$ : Time from MISO to CS
- $t_6$ : Time from CS to MISO

The diagram also shows the ATmega16's internal state (Q) and the ATmega328P's internal state (Q). The ATmega16's internal state is shown as a series of pulses, and the ATmega328P's internal state is shown as a series of pulses. The diagram is a timing diagram for the SPI interface between the ATmega328P and the ATmega16.

**Figure 51: RXTX\_BITSLICE Receiver in 8-bit Mode**



Timing diagram for RX\_TX\_BITSlice showing clock and data signals. The diagram is divided into two sections. The top section shows the internal TX\_BITSlice clock generation from the FPGA Logic Clock. The bottom section shows the output signals RX\_TX\_BITSlice\_O Clock and Data\_0 and Data\_1. Annotations include:

- $T = \text{period of clock} = (\text{Data Rate}/2)/\text{DIV4}$
- Frequency = Bit rate of the data
- AA = 01010101 TX\_BITSlice generating a clock
- BB = 10111011 TX\_BITSlice generating phase aligned data
- CC = 11001100 TX\_BITSlice generating 90-degrees shifted data
- 1/13/16T (OUTPUT\_PHASE\_90 = FALSE) : RX\_TX\_BITSlice Generated Clock
- 1/13/16T (OUTPUT\_PHASE\_90 = FALSE)
- 1/14/16T (OUTPUT\_PHASE\_90 = TRUE)

**Figure 53: RXTX\_BITSLICE Transmitter in 8-bit Mode**

## References

1. *UltraScale Architecture SelectIO Resources User Guide* ([UG571](#))
2. UltraScale device data sheets:
  - *UltraScale Architecture and Product Overview* ([DS890](#))
  - *Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* ([DS892](#))
  - *Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* ([DS893](#))
3. *Virtual Input/Output v3.0 Product Guide* ([PG159](#))
4. *ChipScope Pro Virtual Input/Output (1.04a)* ([DS284](#))
5. *Vivado Design Suite User Guide: Programming and Debugging* ([UG908](#))
6. *High Speed SelectIO Wizard Product Guide* ([PG188](#))
7. High Speed SelectIO Wizard—Known Issues List ([AR64126](#))
8. *LVDS 4x Asynchronous Oversampling Using 7 Series FPGAs* ([XAPP523](#))
9. *UltraScale and UltraScale+ FPGAs Packaging and Pinouts Product Specification* ([UG575](#))
10. *Bitslip in Logic* ([XAPP1208](#))
11. *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* ([UG906](#))

## Revision History

The following table shows the revision history for this document.

Date	Version	Revision
09/06/2019	1.2	Updated second bullet in <a href="#">I/O or Global Clock</a> . Removed REFCLK from <a href="#">Transmitters</a> . In <a href="#">RX_BITSLICE Integrated FIFO</a> , updated second bullet and added note after third bullet. Added note about strobe clocks after first paragraph in <a href="#">I/O Bring-up Sequence</a> . Updated <a href="#">step 1</a> in <a href="#">Release Reset</a> . Updated second bullet after <a href="#">Figure 32</a> . Updated fourth bullet in <a href="#">BITSLICE</a> . Updated second paragraph after <a href="#">Figure 41</a> .
02/28/2017	1.1	Updated the <a href="#">Summary</a> and <a href="#">Introduction</a> and added the <a href="#">Asynchronous Data Capture Interfaces</a> section and reference design. Updated <a href="#">Figure 14</a> , <a href="#">Figure 15</a> , and <a href="#">Figure 41</a> . Added <a href="#">Cores Generated using the High Speed SelectIO Wizard</a> . Updated <a href="#">Automotive Applications Disclaimer</a> .
06/01/2016	1.0	Initial Xilinx release.

---

## Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at [www.xilinx.com/legal.htm#tos](http://www.xilinx.com/legal.htm#tos); IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at [www.xilinx.com/legal.htm#tos](http://www.xilinx.com/legal.htm#tos).

### **Automotive Applications Disclaimer**

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

© Copyright 2016–2019 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.