Summary

This application note and associated reference design show how to use the High Speed SelectIO™ Wizard to generate an asynchronous receiver using the native mode I/O in UltraScale™ and UltraScale+™ devices.

A three-channel transmitter and receiver design can be implemented on a board such as the KCU105 board [Ref 1]. The reference design has been modified to provide easier pinout changes by using the High Speed SelectIO Wizard. The reference design targets the KCU105 board, but the design can be modified for other boards by following the guidelines described in this application note.

Download the reference design files for this application note from the Xilinx website. For detailed information about the design files, see Asynchronous Reference Design.

Introduction

Asynchronous data capture allows data to be captured when using differential inputs like LVDS without a forwarded clock as commonly used in synchronous interfaces. Instead, a state machine uses a reference clock and the variable delays within the RX_BITSLICE to recapture the clock from the data stream. The clock recovery algorithm requires the RX_BITSLICE to be directly controlled, which requires the High Speed SelectIO Wizard to be configured and connected to the clock recovery algorithms.

These are the restrictions in asynchronous data capture:

• 1300 Mb/s (-2 and 3 speed grades) or 1250 Mb/s (-1 speed grade)
• Inputs must use differential inputs

These are the restrictions in the reference design:

• For porting purposes, the receiver (RX) and transmitter (TX) must be placed in different byte groups
• The receiver must use ASYNC mode DATA_WIDTH = 4
• The transmitter must use DATA_WIDTH = 8
• Changes to pins might require changes to VHDL
Asynchronous Data Capture

• A differential 125 MHz clock is used to control the virtual I/O (VIO) interface for general-purpose I/Os

RECOMMENDED: Drive clocks directly from global clock inputs that connect to the Quad byte clock strobe pins. The pin name contains the text string “GC_QBC.” Directly connect global clock inputs to the PLL within the bank. Using the PLL with a cascaded MMCM clock source or sharing clocks across multiple banks might limit the performance.

This application note is organized by the following topics:

• Asynchronous Data Capture
• Asynchronous Reference Design
• Simulation
• Hardware Operation

Asynchronous Data Capture

Synchronous interfaces transfer data with a clock that has a known phase and frequency related to the data being transferred. For example in a DDR interface, the clock is normally phase-aligned with data being captured on both the rising and falling edges of the clock.

For asynchronous interfaces, sometimes only data is transmitted. In other cases, a clock is transmitted with data, but the exact clock/data phase relationship is unknown. To recapture the asynchronous data in these cases, the receiver’s reference clock must match the frequency of the data within ±100 ppm. For example, if the transmitter transmits data at 1250 Mb/s and the receiver requires a clock of 625 MHz, the reference clock can vary by ±0.125 MHz. For this reference design, the receiver’s circuitry uses the RX_BITSLICE (native mode) along with control circuitry to track the phase alignment of the asynchronous data. This data recovery method is called phase tracking.

The phase tracking is loosely based on the Alexander Bang-Bang Phase Detector. As shown in Figure 1, a monitor is adjusted to be aligned to the edge of the data transitions based on using the active and monitor data. When monitor is at the edge of data, active is centered due to the separation of half a bit time (BtVal/2).
For this phase tracking to work, the delay tap value for a single bit time must be read from the BITSLICE_CONTROL. For a design operating at 1250 Mb/s, the bit time is 800 ps. A register interface unit (RIU) state machine records this delay value as BtVal, which is then used by the phase tracking algorithm. For example, if the average tap delay is 5 [ps/tap], then BtVal = 160 for this example calculation.

In UltraScale and UltraScale+ devices, differential pairs can use both RX_BITSLICEs, as shown in Figure 2. A state machine is used to control the delays for both the master and slave RX_BITSLICEs so that one of the bitslices samples the data at the center of a bit time (active) while the other bitslice samples data at the edge of the bit time (monitor).

**Figure 1: Monitor Phase Tracking Adjustments (Active Delay > Monitor Delay)**

**Figure 2: Differential Channel For Asynchronous Phase Tracking**
In UltraScale devices, the phases are controlled by adjusting the CNTVALUEIN for the different RX_BITSLICEs, which inserts the delays into the datapath. Increasing the delay value has the effect of moving the sampling window negatively, as shown in Figure 3.

![Figure 3: Data Sampling Window With RX_BITSLICE Delays](image)

Communication protocols use 8B/10B encoding to transmit data requiring a gearbox to convert the 4-bit patterns to 10-bit patterns. The state machine uses the 4-bit to 10-bit gearbox to additionally ensure data is not lost due to the phase tracking.

This gearbox also provides the additional purpose of compensating up to the 200 ppm difference between the data rate and reference clock. The gearbox is implemented as a 15-bit shift register and can gradually insert or remove an extra bit of data using oversampling and consequently ensuring no data is lost. For the gearbox, both the write and read side are clocked by a fabric clock running at the frequency of the parallel data (1250 Mb/s ÷ 4 = 312.5 MHz).

### Asynchronous Reference Design

Download the reference design files for this application note from the Xilinx website.

Table 1 shows the reference design matrix.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General</strong></td>
<td></td>
</tr>
<tr>
<td>Developer name</td>
<td>Jim Tatsukawa and Marc Defossez</td>
</tr>
<tr>
<td>Target devices</td>
<td>UltraScale FPGAs</td>
</tr>
<tr>
<td>Source code provided</td>
<td>Yes</td>
</tr>
<tr>
<td>Source code format</td>
<td>VHDL and Verilog</td>
</tr>
<tr>
<td>Design uses code and IP from existing Xilinx application note and reference designs or third party</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Simulation</strong></td>
<td></td>
</tr>
<tr>
<td>Functional simulation performed</td>
<td>Yes</td>
</tr>
<tr>
<td>Timing simulation performed</td>
<td>Yes</td>
</tr>
<tr>
<td>Test bench used for functional and timing simulations</td>
<td>Yes</td>
</tr>
</tbody>
</table>

---

XAPP1330 (v1.1) August 3, 2018

www.xilinx.com
The reference design accompanying this application note uses the High Speed SelectIO Wizard to generate the native mode interfaces for the TX and RX as shown in Figure 4.

![Asynchronous Block Diagram](XAPP1330-v1.1-052418)

**Figure 4: Asynchronous Block Diagram**

The design files are organized as shown in Figure 5. In the download files, a TCL script (top.tcl) has been added to rebuild the project.
The reference design uses four cores:

- **Tx_Data_Interface** - High Speed SelectIO Wizard for transmitter setup of TX_BITSLICE
- **Rx_Async_Interface** - High Speed SelectIO Wizard for receiver setup of RX_BITSLICE
- **vio_0** - Hardware interface to control resets and for getting status of reference design
- **ila_0** - Hardware debug interface for capturing data during operation

The constraint file has additional settings for the specific demonstration board being targeted and might not be needed for all boards. The High Speed SelectIO Wizard provides additional constraints such as placement and select I/O settings.

The phase alignment algorithm requires RIU access to the BITSLICE_CONTROL, which is why the RX and TX interfaces must be kept in different byte groups and the design can be used without any changes. For designs that must place the RX and TX interfaces within the same byte group, the RIU_OR must be used within the `Byte_TopWizard_RxTx.vhd` design file. The RIU_OR has been commented out in the `Byte_TopWizard_RxTx.vhd` design file. Because the RIU_OR has been removed, the transmitter and receiver can be placed in different banks.
While changing pinouts for any device is simplified by using the wizard and using the same signal names for the ports, a number of interface signals need to be updated based on the instantiation template. The port names can be seen in the instantiation template for the given IP source. For VHDL, the component definition and the instantiation need to be updated.

These are the RX port names based on byte group:

- \texttt{riu_addr_bg<#> \Rightarrow IntRiu_Addr}, \quad -- \text{in [5:0]}
- \texttt{riu_wr_data_bg<#> \Rightarrow IntRiu_WrData}, \quad -- \text{in [15:0]}
- \texttt{riu_rd_data_bg<#> \Rightarrow IntBase_Riu_Rd_Data_Rx}, \quad -- \text{out [15:0]}
- \texttt{riu_valid_bg<#> \Rightarrow IntBase_Riu_Valid_Rx}, \quad -- \text{out}
- \texttt{riu_wr_en_bg<#> \Rightarrow IntRiu_Wr_En}, \quad -- \text{in}
- \texttt{riu_nibble_sel_bg<#> \Rightarrow IntRiu_Nibble_Sel}, \quad -- \text{in [1:0]}

These are the RX port names based on nibble (bitslice control):

- \texttt{dly_rdy_bsc<#> \Rightarrow IntBase_Rx_Dly_Rdy}, \quad -- \text{out}

These are the RX port names based on bitslice:

- \texttt{rx_en_vtc<#> \Rightarrow IntBase_Rx_Bs_En_Vtc}, \quad -- \text{in}
- \texttt{fifo_rd_clk<#> \Rightarrow IntBase_Rx_Fifo_Rd_CLK}, \quad -- \text{in}
- \texttt{fifo_rd_en<#> \Rightarrow IntBase_Rx_Fifo_Rd_En_0(1)}, \quad -- \text{in}
- \texttt{fifo_empty<#> \Rightarrow IntBase_Rx_Fifo_Empty_0(1)}, \quad -- \text{out}
- \texttt{rx_ce<#> \Rightarrow Low}, \quad -- \text{in}
- \texttt{rx_inc<#> \Rightarrow Low}, \quad -- \text{in}
- \texttt{rx_load<#> \Rightarrow IntBase_Idly_Load_0(1)}, \quad -- \text{in}
- \texttt{rx_cntvaluein<#> \Rightarrow IntBase_Idly_CntValueIn_0(17 downto 9)}, \quad -- \text{in [8:0]}
- \texttt{rx_cntvalueout<#> \Rightarrow IntBase_Idly_CntValueOut_0(17 downto 9)}, \quad -- \text{out [8:0]}

These are the TX port names based on nibble (bitslice control)

- \texttt{vtc_rdy_bsc<#> \Rightarrow open}, \quad -- : \text{OUT STD_LOGIC;}
- \texttt{en_vtc_bsc<#> \Rightarrow IntBase.Tx_Bsc_En_Vtc}, \quad -- : \text{IN STD_LOGIC;}
- \texttt{dly_rdy_bsc<#> \Rightarrow IntBase.Tx_Dly_Rdy_bsc4}, \quad -- : \text{OUT STD_LOGIC;}

For the TX ports, if the design spans multiple nibbles, the \texttt{dly_rdy_bsc<#>} for each of the nibbles should be combined using the Stat_Tx_Dly_Rdy (Byte_TopWizard_RxTx.vhd). For example, if the TX interface spans nibble 0 and nibble 1, the Tx_Data_Interface should be adjusted as follows for \texttt{dly_rdy_bsc#} connections:

\begin{verbatim}
Stat_Tx_Dly_Rdy <= IntBase.Tx_Dly_Rdy_bsc0 and IntBase.Tx_Dly_Rdy_bsc1;
Byte_TopWizard_RxTx_I_TX_Data_Interface : Tx_Data_Interface
port map (  
  Tx_dly_rdy_bsc0 => IntBase.Tx_Dly_Rdy_bsc0,  
  Tx_dly_rdy_bsc1 => IntBase.Tx_Dly_Rdy_bsc1,  
  ...  
);
\end{verbatim}

The following lists pin selection guidelines for simplifying RX and TX designs:

- Each byte group must be either RX or TX.
- Empty byte groups are allowed. For example, an interface can use byte group 0 and 2.
- Byte groups spanning each interface must be either RX or TX. If byte groups 0 and 2 are used for the RX, byte group 1 must either be an RX or left empty.
Figure 6 shows how the bitslices are organized within a bank.

<table>
<thead>
<tr>
<th>Bitslice</th>
<th>Nibble (Bitslice Control)</th>
<th>Byte Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
For the reference design to work, the TX and RX wizard settings are shown in Figure 7 to Figure 11.

**Figure 7:** High Speed SelectIO Wizard TX Setup - Basic Tab
Figure 8: High Speed SelectIO Wizard TX Setup - Advanced Tab
Figure 9: High Speed SelectIO Wizard TX Setup - Pin Selection Tab
The settings shown in Figure 10, Figure 11, and Figure 12 should be used for the RX.

**Figure 10:** High Speed SelectIO Wizard RX Setup - Basic Tab
Figure 11: High Speed SelectIO Wizard RX Setup - Advanced Tab

COUNT must be used

Dynamic alignment requires VAR_LOAD setting

Equalization might be needed for AC-coupled differential inputs
**Figure 12:** High Speed SelectIO Wizard RX Setup - Pin Selection Tab
To allow the reference clock to be moved to different banks, an extra clock buffer (BUFGCE) has been added as shown in Figure 13. To free up the clock routes, the CLOCK_DEDICATED_ROUTE has been added. In this specific case, the clock is only routed to the PLLs and a divided clock, which is used as a control clock that does not require any phase relationship to the input pins.

Figure 13: Reference Clock Connections to PLLs
Simulation

A simple testbench has been provided. The reference design uses registers that are not supported by simulation. For proper simulation, set C_InSimulation = "true" in the top-level design file Byte_Top_RxTx_Prbs_Wiz_Vio_Top.vhd.

```
Byte_Top_RxTx_Prbs_Wiz_Vio_Top_I_RxTx_Prbs : entity
xil_defaultlib.Byte_Top_RxTx_Prbs_Wiz
  generic map (
    C_InSimulation              => "true",
    C_SimDevice                 => "ULTRASCALE"
  )
```

The reference design accesses a reserved BITSLICE_CONTROL RIU that is not supported in simulation. In Riu_StateMach.vhd, when C_InSimulation = "true", Rx_BtVal is set to 160 (0xA0). C_InSimulation = "true" fixes the Rx_BtVal in post-synthesis and post-implementation simulations.

After simulation is complete, set C_InSimulation = "false" to ensure the Rx_BtVal is being read from the BITSLICE_CONTROL. The design needs to be re-synthesized afterwards so that the Rx_BtVal is read from the Rx_RdData connected to the BITSLICE_CONTROL.

To simplify the RIU state machine and allow for more portability, the bit time for Rx_BtVal is used as the bit time for all of the receivers, because each receiver should be running at the same data rate. In Byte_TopWizard_RxTx.vhd, Int_Base_Rx_BtVal is used for each BaseX_TxRxLogic instantiation.

Hardware Operation

The reference design has been set up for the KCU105 board using the FMC HPC connector. Attach the XM107 FMC loopback card to the J22 FMC connector. The XM107 connects LA[16:0] to LA[33:17]. See the FMC XM107 Loopback Card User Guide (UG539) [Ref 2].

The design requires two clock sources. The VIO interface is directly connected to the 100 MHz clock to control the VIO interface and generate the input/output signals for controlling the design.

The transmitter and receiver channels require a 625 MHz clock which is connected to the Si570 when SI570_CLK_SEL = 1. To control the clocks, use the serial port as described in the KCU105 Board User Guide (UG197) [Ref 1] using the system controller. By using the serial port, the system controller can set the KCU105 Si570 user clock frequency to 625 MHz.

For some systems the FMC might also need to be powered using the serial port. Follow the instructions in the board user guide, for example, KCU105 Board User Guide (UG917). Hardware operation should follow the reset sequence using the VIO outputs, as shown in Figure 14.
The reset sequence has been simplified to reset when IntResetIn has been asserted High. The reset and enable controls are still available, as shown in Figure 14.

When operating correctly, GernatePrbsValid<03|02|01> and PrbsErrorDetec<03|02|01> continue to toggle, indicating the interface is running.
Check PrbsErrDetSticky_<03|02|01> and PrbsErrCnt_<03|02|01> for errors.

**Figure 15: Virtual I/O (VIO) Inputs**

[Description of the diagram and the highlighted areas]

- **[17:9] Bt_Val**
- **[9:0] State Machine**
- **GeneratPrbsValid with Activity**
- **PrbsErrCnt should be running for normal operation.**
- **PrbsErrDetSticky & PrbsErrDetect are Low With No Activity**
- **LOCKED status for TX and RX PLL's**
- **Output from reference design to assert TX/RX logic in reset during startup**
- **LOCKED status for TX and RX PLL's**
To make sure the interface is working, use the InjectError_Chnl_<03|02|01> to inject an error on the given channel. As shown in Figure 16, PrbsErrDetSticky remains High if any errors occur. In the example, InjectError_Chnl_02 has been asserted High. When the error occurs, PrbsErrorDetect_02 momentarily goes High with the activity also indicating a change in values. When enough errors occur, PrbsErrCnt terminates with the activity stopping.

![Virtual I/O (VIO) Inputs After Injected Errors](image)

**Figure 16:** Virtual I/O (VIO) Inputs After Injected Errors

After the error has occurred, follow the reset sequence. PrbsClearSticky_<03|02|01> and PrbsRst_Chnl_<03|02|01> will clear the error flags and error counter.

**Note:** If PrbsErrCnt_<03|02|01> does not have any activity and is H000_0000_0000_0000, check to make sure the FMC supplies are set. Check the user guide for the given board on how to set the FMC supply.

### Conclusion

The asynchronous reference design works with the native primitives to control the RX_BITSLICE and BITSLICE_CONTROL primitives. As a result, when the pins are modified in the High Speed SelectIO Wizard, the port connections need to be updated. By following the guidelines in this application note, the reference design pinouts can be modified. Additional instructions on using the design have also been provided.
Documentation Navigator and Design Hubs

Xilinx Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the **Design Hubs** page.

*Note:* For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

References

1. *KCU105 Board User Guide (UG917)*
2. *FMC XM107 Loopback Card User Guide (UG539)*
3. *LogiCORE IP High Speed SelectIO Wizard Product Guide (PG188)*
4. *UltraScale Architecture SelectIO Resources (UG571)*

Revision History

The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>03/29/2018</td>
<td>1.0</td>
<td>Initial Xilinx release.</td>
</tr>
<tr>
<td>08/03/2018</td>
<td>1.1</td>
<td>Removed all references to <em>Native High-Speed I/O Interfaces</em> (XAPP1274). Added <em>Asynchronous Data Capture</em>. Updated description of phase alignment algorithm and first bullet of pin selection guidelines in <em>Asynchronous Reference Design</em>. Updated description of clocks before Figure 13. Updated Figure 4, Figure 7, Figure 8, Figure 10, Figure 12, and Figure 15.</td>
</tr>
</tbody>
</table>
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 https://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 https://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 2018 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.