



XAPP523 (v1.1) May 17, 2017

LVDS 4x Asynchronous Oversampling Using 7 Series FPGAs and Zynq-7000 AP SoCs

Author: Marc Defossez

Summary

This application note describes a method of capturing asynchronous communication using LVDS with SelectIO™ interface primitives. The method consists of oversampling the data with a clock of similar frequency (± 100 ppm). This oversampling technique involves taking multiple samples of the data at different clock phases to get a sample of the data at the most ideal point.

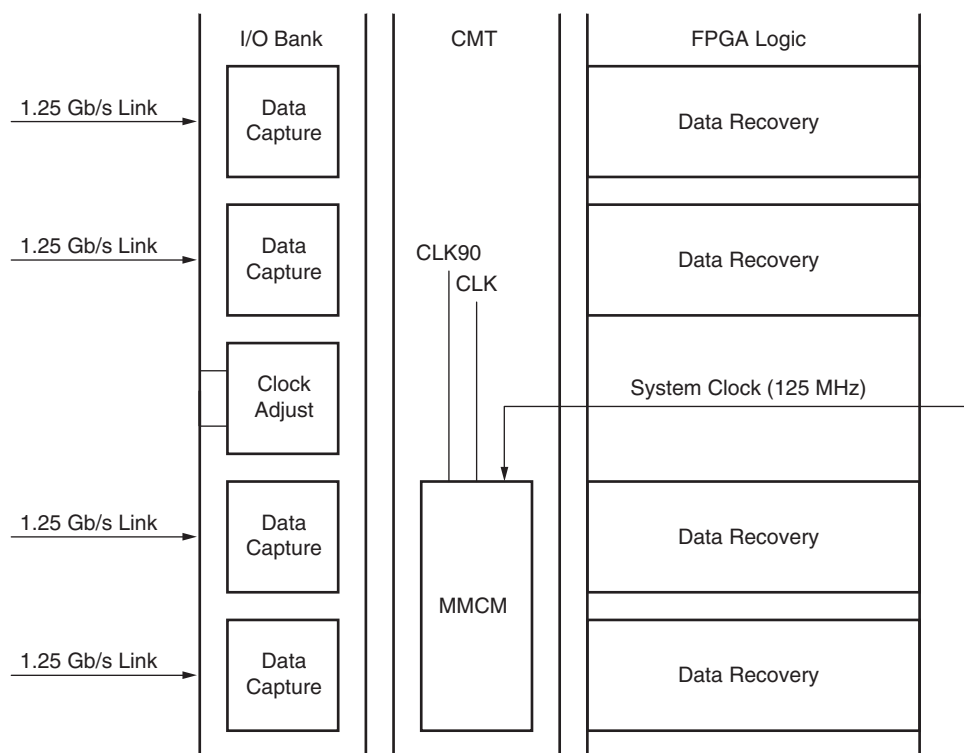
The SelectIO interface in 7 series FPGAs and Zynq®-7000 All Programmable SoCs can perform 4x asynchronous oversampling at 1.25 Gb/s. Oversampling is performed by using ISERDESE2 primitives. Clocks are generated from a mixed-mode clock manager (MMCME2_ADV) through dedicated high-performance paths between the components.

Introduction

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*, also known as data and/or clock recovery. Xilinx® transceivers 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 application note provides a “partial solution” in that no clock is actually recovered, but the arriving data is fully extracted. [Figure 1](#) shows a typical use case.



X523_01_012012

Figure 1: Typical Data Recovery Application

Asynchronous Oversampling

For signal processing, “oversampling” means sampling a signal using a sampling frequency significantly higher than twice the bandwidth (or highest frequency) of the signal being sampled. For the communication interface described in this application note, the “significantly higher” sampling frequency is obtained using different edges of multiple phase-shifted clocks. It is called *asynchronous oversampling* because the clocks used to create the sampling frequency are nominally equal to the data stream frequency.

The circuit discussed here uses a clock (local oscillator) running at the same nominal frequency as the data stream being captured. “Nominal” here means that the local oscillator is either slightly faster or slightly slower than the incoming clock/data stream.

Through the use of a clock manager (MMCME2), high-speed phase-shifted clocks are generated from a slow system clock typically provided by a local clock oscillator (see [Figure 2](#)).

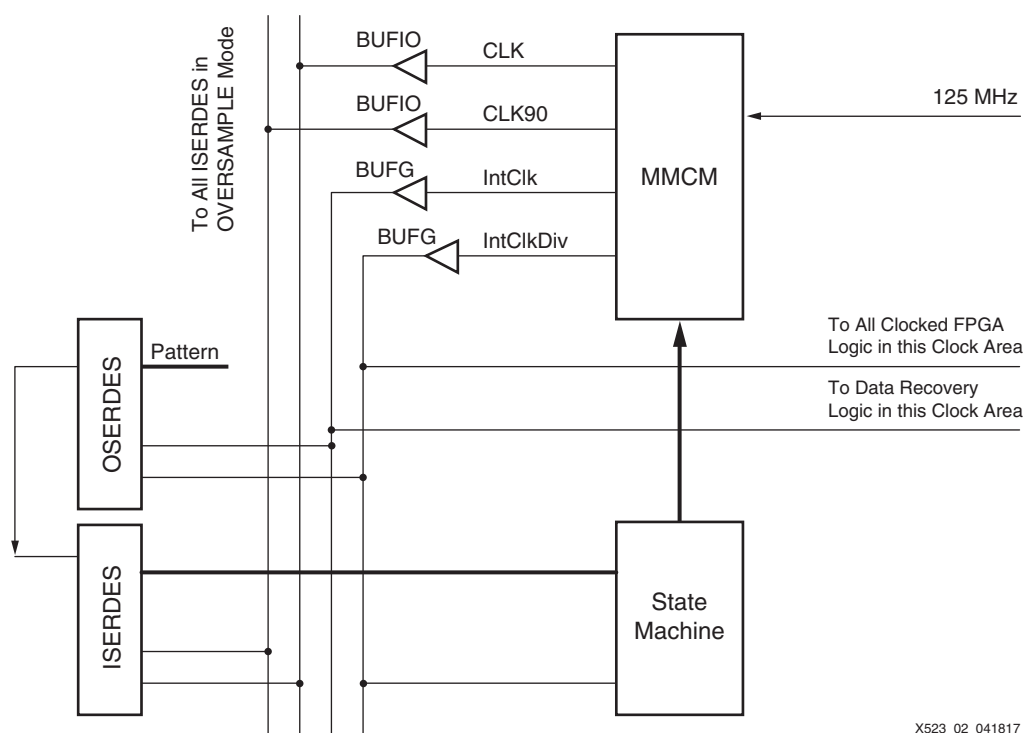


Figure 2: Clock Generation Using MMCME2

The function of the two extra clocks and ISERDES/OSERDES combination shown in Figure 2 is explained in [Clocking and Data Flow, page 10](#). The generated CLK and CLK90 clocks make it possible to oversample an incoming data stream on four edges, meaning that each bit of the DDR data stream can be sampled twice, as shown in Figure 3.

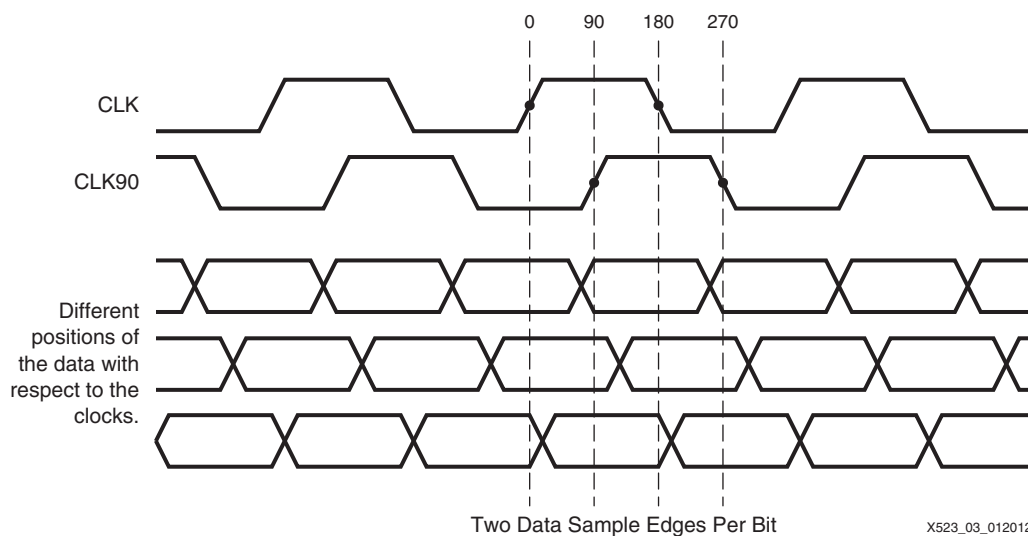
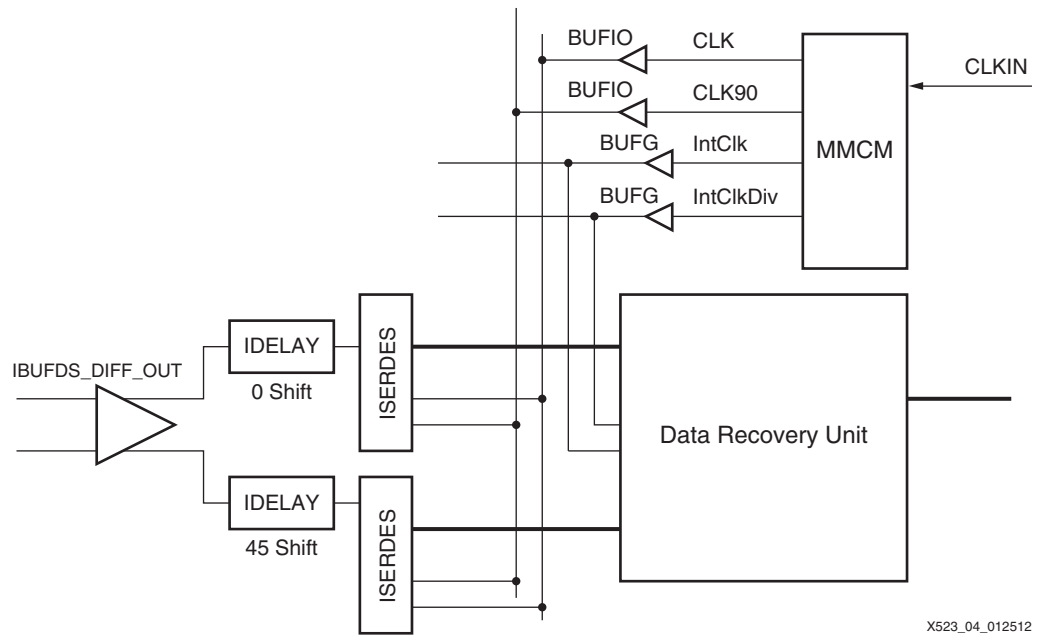


Figure 3: Data Oversampling on Four Clock Edges

If the incoming data stream is split into two branches and one branch is delayed by 45° , it is possible to 4x oversample every data bit. The details of how this circuit is constructed using MMCME2, IODELAYE2, and ISERDESE2 is shown in Figure 4.

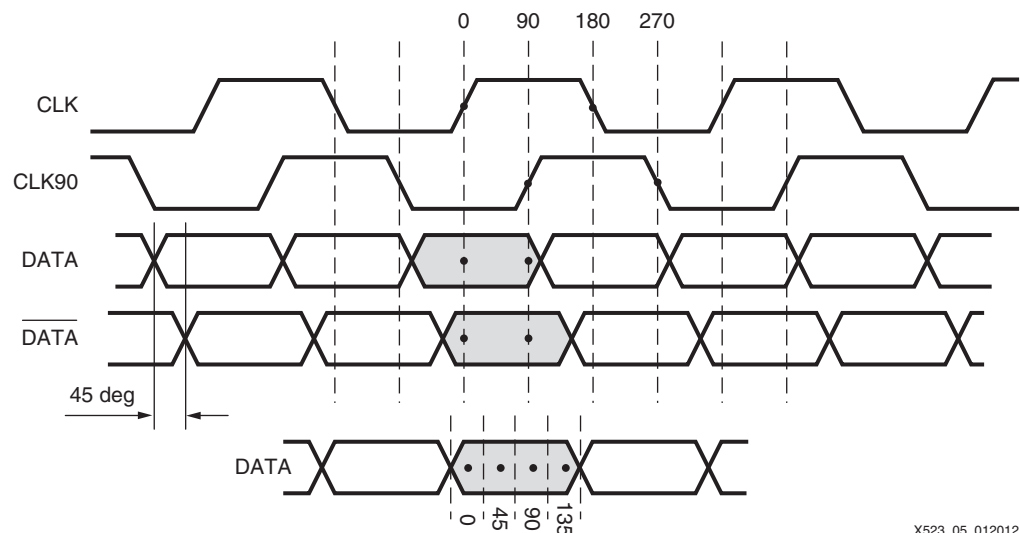


X523_04_012512

Figure 4: MMCM Phase Clock and Phased Data Generation

The MMCME2 generates two clock phases (CLK0 and CLK90). These are routed to the ISERDESE2, where both the positive-transitioning and negative-transitioning edges of the two clocks are used, creating four clock phases. Two copies of the incoming data are created by means of an IBUFDS_DIFFOUT. One branch of the data gets a shift of 45° and the other branch gets no phase shift. The phase shift is obtained by passing the data of both branches through IODELAYE2. This phase-shifted version of the data is passed into a slave ISERDESE2, effectively doubling the sample rate.

Eight clock sample phases for bit oversampling are thus created by using a combination of four clock phases and two data sample phases, as shown in Figure 5.



X523_05_012012

Figure 5: Sample Edges

IDELAY Tap Setting Calculation Example

The following list outlines the logical flow of timing assumptions and calculations leading to the IDELAY tap settings:

1. Assume that the incoming data stream runs at 1.25 Gb/s; the bit time is therefore 800 ps.
2. The CLK and CLK90 clocks therefore run at 625 MHz, or 1.6 ns.
3. The edges of both clocks arriving at 0°, 90°, 180°, and 270° have positions at 0, 400, 800, and 1200 ps.
4. To shift 45°, one branch of the data must be delayed by 200 ps.
5. The tap delay of an IDELAY component is controlled by an IDELAYCTRL component. The IDELAYCTRL component in this design is clocked at 310 MHz, so a single-tap delay is 52 ps. (Refer to Note 1 in the Input/Output Delay Switching Characteristics table in [DS182](#), *Kintex-7 FPGAs Data Sheet: DC and Switching Characteristics*.)
6. The 200 ps desired delay for 45° phase shift is divided by 52 ps per tap to give 3.8 or 4 taps.

Therefore, the IDELAY_VALUE of the first IDELAY must be set to 0, and the IDELAY_VALUE of the second (slave) IDELAY must be set to 4.

7 Series ISERDESE2 Oversampling Mode

The ISERDESE2 component in 7 series FPGAs is an improved version of similar components in previous FPGA families (ISERDES in Virtex-5 FPGAs and ISERDESE1 in Virtex-6 FPGAs). The ISERDESE2 component can implement (i.e., be configured as) different functions:

- In its most basic function, the ISERDESE2 provides the functionality of an IDDR flip-flop.
- A second and more complex function is that of a dedicated serial-to-parallel converter with specific clocking and logic features designed to facilitate the implementation of high-speed source-synchronous applications (NETWORKING mode).
- A third function is MEMORY mode, where the ISERDES is configured as a dedicated interface for different types of memories (QDR, DDR3, etc.).
- In its fourth and final function, the ISERDESE2 can be used in OVERSAMPLING mode. Here, ISERDESE2 is used to capture two phases of DDR data. In this mode, the ISERDESE2 is thus used as a dual set of IDDR flip-flops.

For detailed descriptions of the ISERDESE2 functionality, see [UG471](#), *7 Series FPGAs SelectIO Resources User Guide*.

For convenience, [Figure 6](#) shows the ISERDESE2, with oversampling mode configuration. In earlier implementations, the oversampling design was implemented in FPGA logic using SLICE flip-flops. With 7 series FPGAs, this functionality is implemented in the ISERDESE2.

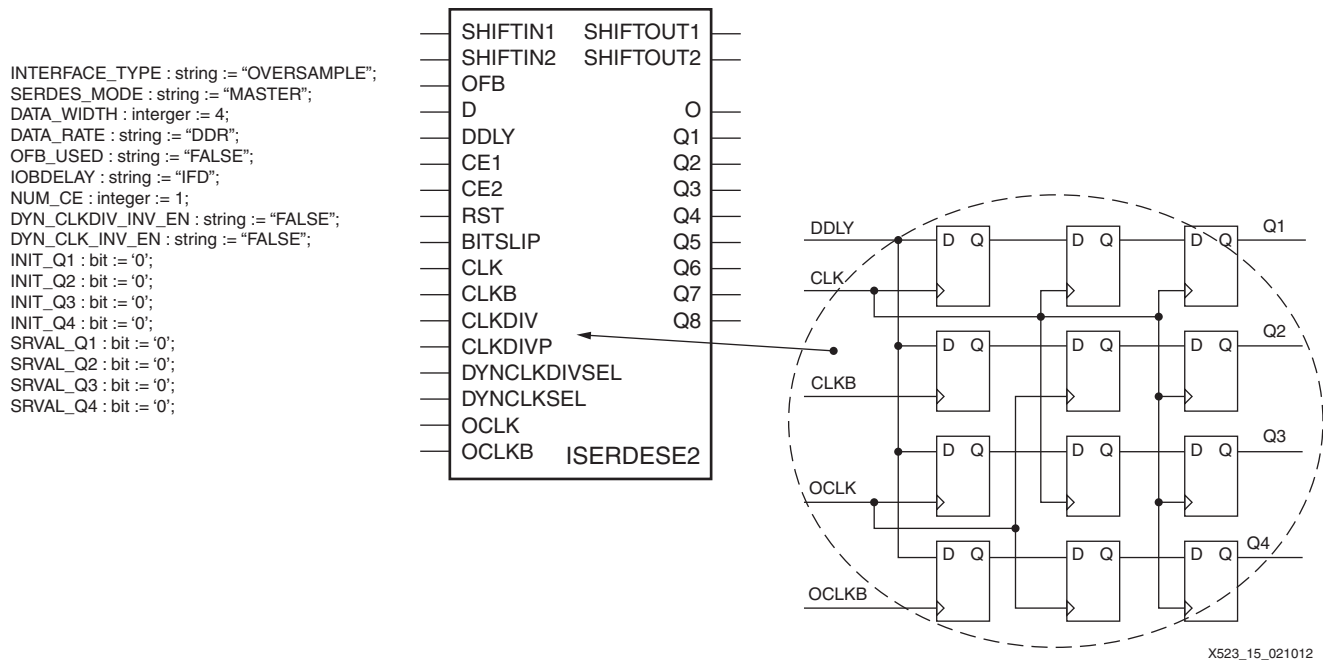


Figure 6: ISERDESE2 in OVERSAMPLING Mode Configuration

Data Recovery Unit

An ISERDESE2 used in networking mode requires a high-speed sampling clock (CLK) to capture the serial data stream in the ISERDESE2. It also needs a low-speed function of CLK (CLKDIV) to present the captured data in parallel format at the outputs of the ISERDESE2. The internal circuits in the ISERDESE2 ensure that this conversion from the CLK level to the CLKDIV level works as a clock-domain crossing (CDC) circuit.

The output of the ISERDESE2 components in oversampling mode is generated from the high-speed sampling clocks (CLK/CLKB and OCLK/OCLKB). These clocks are only available for clocking of ISERDESE2 and/or OSERDESE2.

The CDC operation must thus be implemented in registers in the FPGA logic. The details of how this is done are discussed in [Clocking and Data Flow, page 10](#). The CDC registers and some comparing logic are implemented in the data recovery unit (DRU) and are clocked by the CLK input. The low-speed clock (CLKP or CLKDIV) clocks the rest of the DRU.

The locations of the sampling and comparison points relative to the data stream coming into the FPGA are shown in [Figure 7](#). There are two streams of data, one offset from the other by 200 ps (4 IDELAY taps). For this application, the speed of the arriving data stream is 1.25 Gb/s. Through use of the IBUFDS_DIFFOUT primitive, one data stream is the complement of the other data stream, similar to the input data of IBUFDS_DIFF_OUT (i.e., differential signaling).

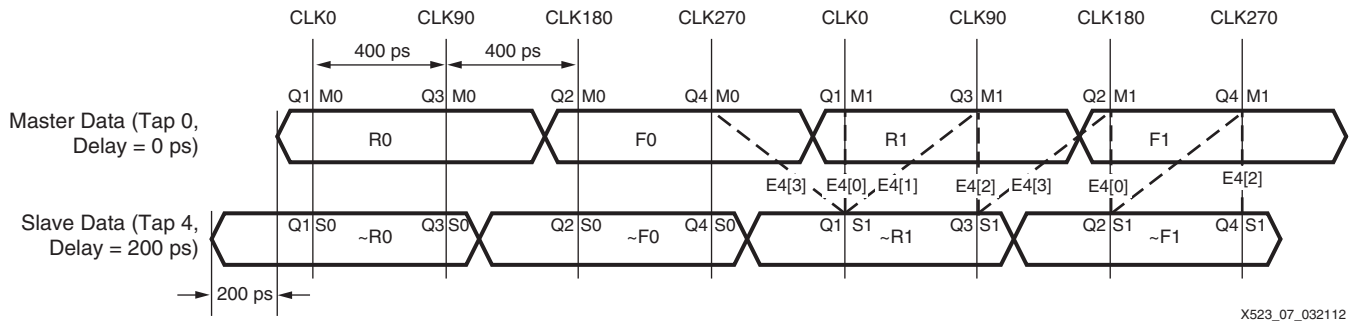


Figure 7: Data Stream Sample and Comparison Points

The data is sampled through four clock phases 400 ps or 90° apart and named CLK0, CLK90, CLK180, and CLK270, as shown in [Figure 3, page 3](#). Sampling points occur where the clocks intersect the data streams. These points are named according to the format:

$Qx [M \text{ or } S]x$

Where:

Qx = the ISERDESE2 outputs Q1, Q2, Q3, or Q4

Mx or Sx = the source ISERDESE2 (M = master, S = slave) of the data outputs (Qx)

For example, sample point Q1M1 shows where CLK0 samples the data and creates an output at port Q1 of the master ISERDESE2.

The lines labeled E4[0] through E4[3] that connect the sample points show where the DRU is comparing data and looking for a data edge. The formulas for the four comparisons are shown in [Equation 1](#) through [Equation 4](#).

$$E4[0] = [Q1M1 \text{ xor } Q1\overline{S1}] \text{ or } [Q2M1 \text{ xor } Q2\overline{S1}] \quad \text{Equation 1}$$

$$E4[1] = [Q3M1 \text{ xor } Q1\overline{S1}] \text{ or } [Q4M1 \text{ xor } Q2\overline{S1}] \quad \text{Equation 2}$$

$$E4[2] = [Q2M1 \text{ xor } Q3\overline{S1}] \text{ or } [Q4M1 \text{ xor } Q4\overline{S1}] \quad \text{Equation 3}$$

$$E4[3] = [Q1M1 \text{ xor } Q4\overline{S0}] \text{ or } [Q2M1 \text{ xor } Q3\overline{S1}] \quad \text{Equation 4}$$

These comparison points, *relative to the original data stream*, are actually 200 ps apart. For example, [Equation 1](#) (E4[0]) *xor*-compares Q1M1 against Q1S1 and Q2M1 against Q2S1. These comparisons are shown by two gray dashed lines each labeled E4[0].

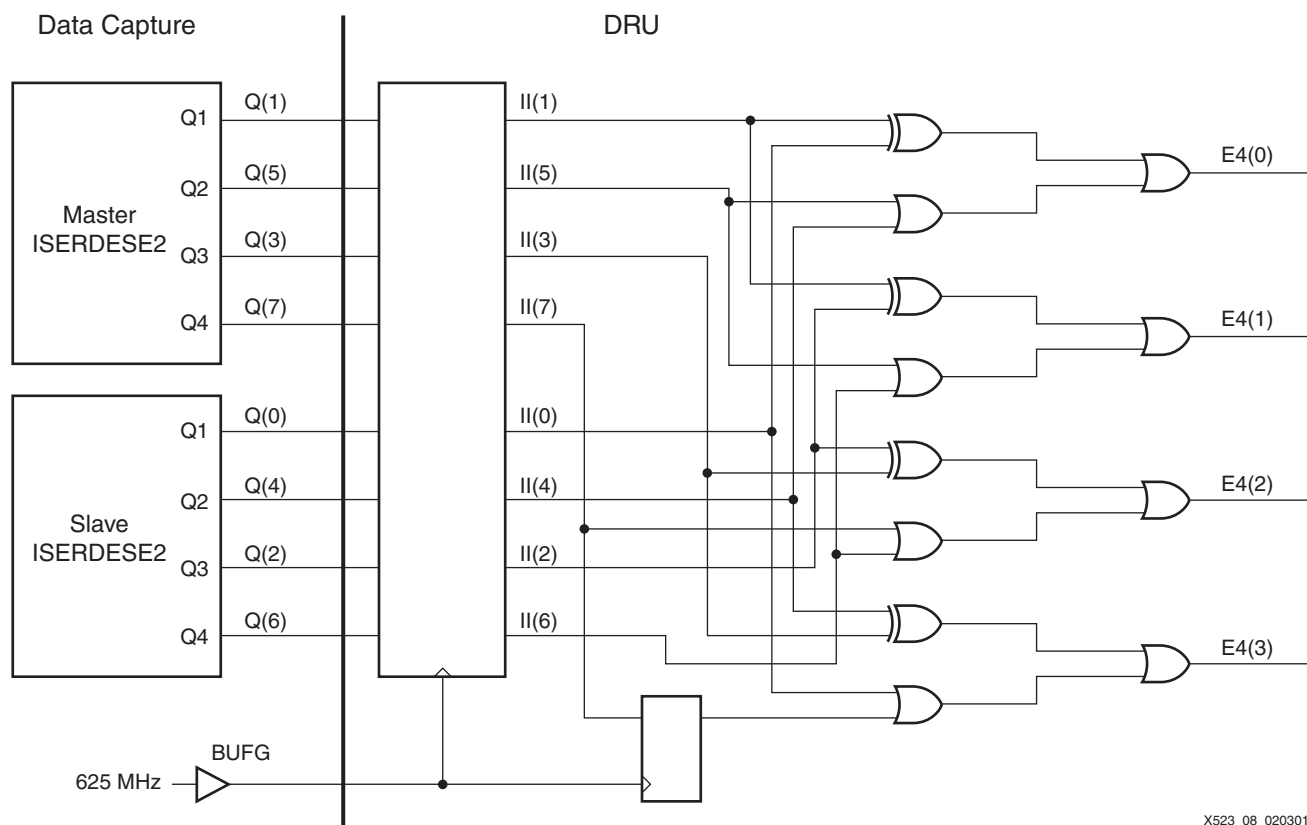
Referring to [Figure 7](#) and looking first at the Q1M1 *xor* Q1S1 comparison, it can be seen that both points are sampled by CLK0. However, the Q1S1 sample is delayed 200 ps (by the action of IDELAYE2) relative to Q1M1, thus allowing comparison of two samples 200 ps apart. Similarly, Q2M1 and Q2S1 are both sampled by CLK180, but again, the sample points are separated by 200 ps due to the action of IDELAYE2 on the slave data stream.

If either the CLK0 or CLK180 sample points produces an *xor* result of 1—that is, the levels of the sampled data do not match—it can be concluded that there is an edge (a level transition) between those two sample points. The first E4[0] sample point comparison occurs in rising-edge zones R1 and $\overline{R1}$, while the second E4[0] sample point occurs in falling-edge zones F1 and $\overline{F1}$. Thus, both comparisons would match, and the *xor* outputs for these tests would both be 0. The DRU state machine would know that there are no data transition edges there.

A contrasting example is shown by [Equation 4](#), which *xor*-compares Q1M1 against Q4S0 and Q2M1 against Q3S1. Q1M1 is sampled by CLK0 from the master data stream, while Q4S0 is sampled by CLK270 from the slave (phase-delayed) data stream and is then stored for an extra cycle in the DRU. CLK270 and CLK0 are 400 ps (90°) apart, but because the slave data is delayed by 200 ps, the Q1M1 and Q4S0 sample points are really only 200 ps apart, again relative to the original data stream. Similarly, Q2M1 is sampled by CLK180, and Q3S1 is

sampled by CLK90. Again, the sample points are 200 ps apart relative to the original data stream. For each comparison, one sample point falls in a rising-edge zone and the other falls in a falling-edge zone. These two comparisons would produce an *xor* result of 1, indicating that an edge (level transition) exists somewhere between the two sample points in each comparison.

Figure 8 shows what Equation 1 through Equation 4 look like in logic and how the data flows out of the ISERDESE2 and into that logic. A stage of registers between the ISERDESE2 and the logic facilitates the timing. This also shows how the Q4 output of the slave ISERDESE2 is stored from the previous sample set to be compared with the new sample set.



X523_08_0203012

Figure 8: Edge Detection Circuit

At this point, it should be clear how the data comes into the FPGA and is then fed into the DRU for edge detection. The next step in the DRU is to process the comparison data. This simple state machine, based upon where the data edge was and where it moves to, then chooses a sample point away from the data edge.

The ideal sample point can be expected to move around because of voltage and temperature variations, jitter, and offset between the source and receiver clocks. This means that the comparison point equations are always changing value, and the state machine is always updating based on these changing results. Figure 9 and Table 1 describe the flow of the state machine from one set of data to the next.

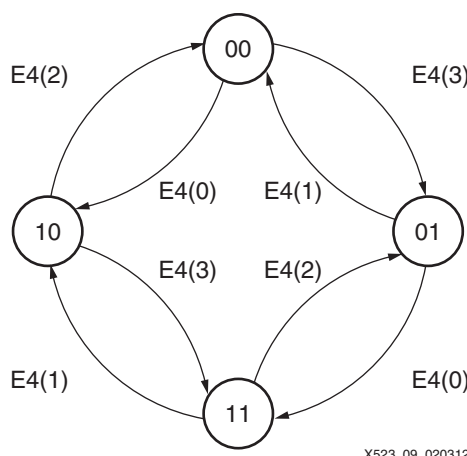


Figure 9: Data Select with Edge Information

Table 1: Data Select Map

EQ	DO
00	Use samples Q(0) and Q(4)
01	Use samples Q(1) and Q(5)
11	Use samples Q(2) and Q(6)
10	Use samples Q(3) and Q(7)

In [Table 1](#), the EQ column shows the current position of the state machine with input from [Equation 1](#) through [Equation 4](#). The DO column shows what sample set is used in the interconnect logic. Remembering that each ISERDESE1 in oversampling mode acts as two sets of IDDR flip-flops, DO indicates which IDDR flip-flops should be used as the ideal sample points.

[Figure 9](#) shows, for each given state or sample set, where the state machine would go next. For example, assume the state machine starts in state 01, which uses the Q(1) and Q(5) signals. This maps to Q1 and Q2 out of the ISERDESE1 master, which would be CLK0 and CLK180, respectively.

Then, if the data edge were to move to the left, the center point would be shifted from CLK0/CLK180 to CLK90/CLK270. This would be seen by E4(3) changing its value from 0 to 1. When this happens, the state machine moves from state 00 to state 01.

Bit Skip

When an edge moves to the left of the first data bit sample, *bit skip* occurs. Bit skip also occurs when an edge moves to the right of the last data bit sample.

When an edge is detected to the left of the last sample, the new current sample is moved from the last sample to the right corresponding to the first sample of the next data. As shown in [Table 1](#), when the state machine is in state 10, it samples Q(3) and Q(7). The state machine then changes to state 00, sampling Q(0) and Q(4). However, a sample of data was already taken in the previous state when the state machine was 10. As a result, during the first 00 state of the state machine, one bit of the sampled bits is dropped. This is called a negative bit skip. Negative bit skip outputs five bits per clock.

When the edge is detected to the right of the first sample, the new current sample is moved to the left, which corresponds to the last sample of the next data. As shown in [Table 1](#), when the state machine is at 00, it samples Q(0) and Q(4). The state machine then changes to state 10. In this state, it samples Q(3) and Q(7). However, no sample of data was taken during states 00

and 10. As a result, during state 10 of the state machine, the last sample is taken along with current samples, causing seven bits to be output. This is called a positive bit skip. Positive bit skip outputs seven bits per clock.

From [Figure 9](#), therefore, it can be seen that bit skip occurs when there is a transition between states 00 and 10. When there are no bit-skip conditions, the sampled data output has one bit per clock in the SDR mode and two bits per clock in the DDR mode.

Therefore, for six bits of parallel data:

- The number of bits for a negative bit skip condition is five.
- The number of bits for a positive bit skip condition is seven.
- The number of bits for no bit skip condition is six.

Clocking and Data Flow

The data flow from the pins of the FPGA to the 10-bit-wide interface presented to the FPGA interconnect logic takes several steps, all under the control of several clock sources. The clock sources are generated through an MMCME2 component. The setup of this design is shown in [Figure 10](#). The numbers above each section represent clocking transfer points discussed in the numbered sections following [Figure 10](#).

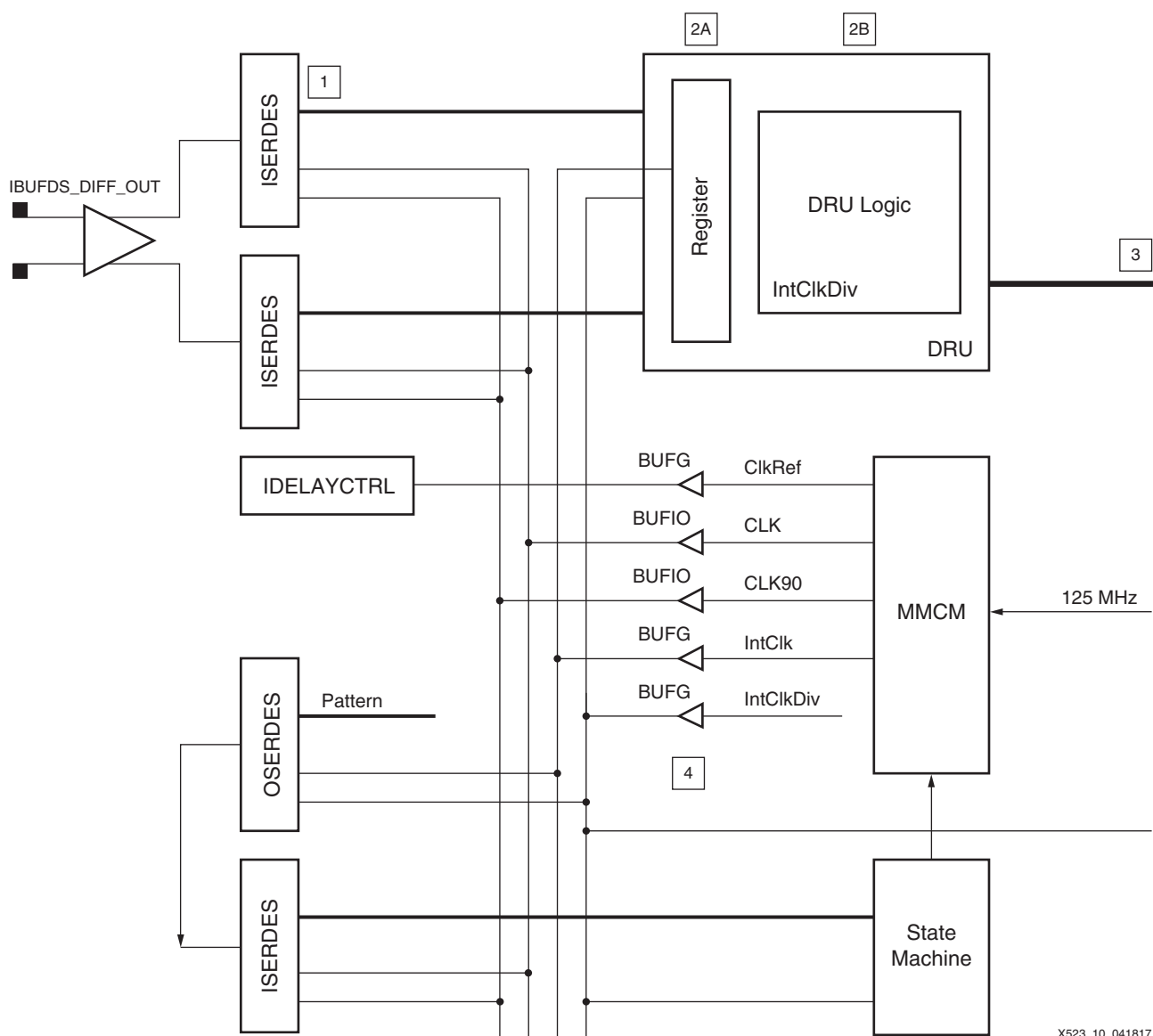


Figure 10: MMCM Clock Resources

1. Data enters FPGA at 1.25 Gb/s

The ISERDESE2 uses the CLK and CLK90 clocks running at 625 MHz. These are applied through single-region BUFIO clocks, which is a requirement for this design.

Capturing of the ISERDESE2 outputs is done using SLICE flip-flops in the BUFG clock domain. This cross-domain clocking forces a 600 ps mandatory constraint between the outputs of the ISERDESE2 running in the BUFIO clock domain and the registers capturing the data in the BUFG clock domain. (These constraints are supplied in the HDL source code.)

2. DRU

- a. The CLK and CLK90 clocks applied to the ISERDESE2 are available only for clocking of devices in the I/O banks. Data must be transferred from the ISERDESE2 into SLICE flip-flops. The SLICE flip-flops capturing the data must run at a separate 625 MHz clock.

To move from the BUFIO clock network to the BUFG clock network, the use of clock domain crossing (CDC) techniques is mandatory. In this case, the clocks for the

ISERDESE2 and the clocks for the DRU logic are phase-aligned using a state machine. It is also very important that the delay from ISERDESE2 to the registers in the DRU does not exceed 600 ps.

- b. In this stage of the DRU, data from the 625 MHz BUFG clock to the 312.5 MHz BUFG clock is handled. These clocks are in phase with each other, and thus there are no special requirements.

3. Data/clock presentation

The 10-bit data from the DRU is presented to the user interconnect logic at a 312.5 MHz clock rate with a clock enable.

4. Clock alignment state machine

BUFIO and BUFG have an undefined phase relationship with each other (length of routing, delay of clock buffers, etc.). To transfer data between both clock domains, CDC logic is needed or, as used here, the clocks must be phase-aligned. To perform clock phase alignment, a calibration scheme is set up. The clock alignment circuit uses the knowledge that all I/Os in an FPGA I/O bank are identical and therefore have identical timing.

- An OSERDESE2 is loaded with a fixed data pattern and is clocked by the clocks from the BUFG clock tree (IntClk, IntClkDiv). The output of the OSERDESE2 is a clock pattern at the IntClk (625 MHz, BUFG) clock rate. Through the feedback path, the clock pattern is captured by the neighbor ISERDESE2 running from the BUFIO clock tree. The data-capturing ISERDESE2 runs on this same clock tree.
- Using this technique, it is possible to measure the phase relationship between the two clocks. Using the independent phase-shift capability of the MMCM with a small state machine, the BUFG clocks are phase-shifted to phase-match the BUFIO clocks. Along with the CLK clock (625 MHz), the CLK90 clock (625 MHz) is phase-shifted, and with the IntClk clock (625 MHz), the IntClkDiv clock (312.5 MHz) is phase-shifted.

The phase calibration process is illustrated in [Figure 11](#).

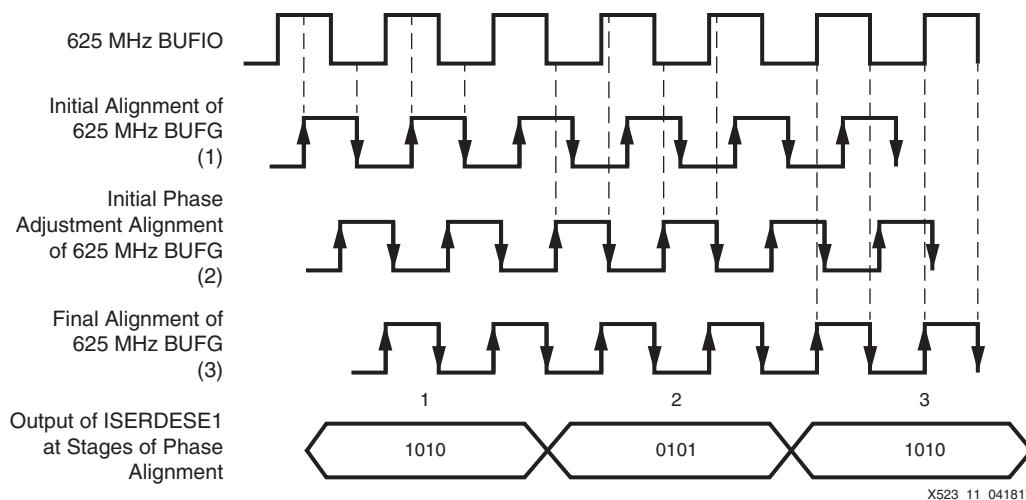


Figure 11: BUFG to BUFIO Phase Alignment Waveform Example

To allow for this clocking scheme, correct configuration of the MMCME2 is necessary. An example of the MMCME2 configuration is provided in the next section.

MMCME2 Clock Generation

The calculations in this section are done assuming that the input for the MMCME2 is 125 MHz and the output clocks need to sample a 1.25 Gb/s (625 MHz DDR) data stream. The component used for this calculation example is a Kintex™-7 FPGA, -2 speed grade.

As shown in [Figure 11](#), one MMCME2 is used as clock source for the interface. This means that the MMCME2 must deliver these clocks:

- **ClkRef** ideally running at 312.5 MHz, but limited to 310 MHz by the IDELAY_CTRL component parameters (REFCLK frequency = 300 MHz ±10 MHz)
- **CLK** at 625 MHz via BUFIO
- **CLK90** inverted CLK at 625 MHz via BUFIO
- **IntClk** at 625 MHz via BUFG
- **IntClkDiv** at 312.5 MHz via BUFG

From [DS182](#), *Kintex-7 FPGAs Data Sheet: DC and Switching Characteristics*, the MMCM switching characteristics for the -2 speed grade are:

- MMCM_F_{IN_MIN} = 10 MHz
- MMCM_F_{IN_MAX} = 933 MHz
- MMCM_F_{VCO_MIN} = 600 MHz
- MMCM_F_{VCO_MAX} = 1440 MHz
- MMCM_F_{OUT_MIN} = 4.69 MHz
- MMCM_F_{OUT_MAX} = 933 MHz
- MMCM_F_{PFD_MIN} = 10 MHz
- MMCM_F_{PFD_MAX} = 500 MHz (Bandwidth set to High or Optimized)

Formulas:

$$D_{MIN} = \text{RoundUp} \left(\frac{F_{IN}}{F_{PFD_MAX}} \right) = 1 \quad \text{Equation 5}$$

$$D_{MAX} = \text{RoundDown} \left(\frac{F_{IN}}{F_{PFD_MAX}} \right) = 12 \quad \text{Equation 6}$$

$$M_{MIN} = \left(\text{RoundUp} \left(\frac{F_{VCO_MIN}}{F_{IN}} \right) \right) \cdot D_{MIN} = 5 \quad \text{Equation 7}$$

$$M_{MAX} = \text{RoundDown} \left(\left(\frac{D_{MAX}}{F_{VCO_MIN}} \right) / F_{IN} \right) = 138 \quad \text{Equation 8}$$

$$M_{IDEAL} = \left(\frac{D_{MIN} \cdot F_{VCO_MAX}}{F_{IN}} \right) = 11.52 \quad \text{Equation 9}$$

To let the PLL inside the MMCME2 run in optimal conditions, F_{VCO} must be maximized while the dynamic phase detector (F_{PFD_MAX}) cannot be exceeded.

The VCO frequency is given by [Equation 10](#).

$$F_{VCO} = F_{IN} \cdot \left(\frac{M}{D} \right) \quad \text{Equation 10}$$

D is calculated as 1 and the M value must be between 5 and 138. With an input clock of 125 MHz, if M is taken as 10, the VCO frequency is 1250 MHz. (Taking M as 12 provides a VCO frequency of 1500 MHz, which is too high).

$$F_{VCO} \Rightarrow F_{IN} \cdot \left(\frac{M}{D} \right) \Rightarrow 125 \text{ MHz} \cdot \frac{10}{1} = 1250 \text{ MHz} \quad \text{Equation 11}$$

The MMCME2 output clocks are calculated in Equation 12.

$$F_{OUT} = F_{IN} \cdot \left(\frac{M}{D \cdot O} \right) \quad \text{Equation 12}$$

Where:

O is the divider factor of the output counter of the MMCME2 clock output.
 D is the value used in the MMCME2 attributes.

- The CLK0 output of the MMCME2 accepts a real number value as divider. This is perfect for generating 310 MHz from 1250 MHz.
 - The D value is 4.0322.
- The CLK1 and CLK2 outputs of the MMCME2 are used to generate the 625 MHz clocks that get distributed to the ISERDESE2 via the BUFIO clock buffers.
 - The D value for both clocks is 2.
- Clock outputs CLK3 and CLK4 are then used to generate the IntClkDiv clock of 312.5 MHz and the IntClkv clock of 625 MHz. These clock outputs must be phase-shifted. Therefore, the attribute for the phase-shifting operation of the MMCM must be turned on.
 - The D value for CLK3 (312.5 MHz) is 4.
 - The D value for CLK4 (625 MHz) is 2.
 - The attribute to turn on for both is CLKOUTn_USE_FINE_PS.
- n = clock output (3 or 4).

Reference Design

The reference design files are available for download at:

<https://secure.xilinx.com/webreg/clickthrough.do?cid=184349>

Table 2 shows the reference design checklist.

Table 2: Reference Design Checklist

Parameter	Description
General	
Developer Name	Marc Defossez
Target Device	Kintex-7 and Virtex-7 FPGAs Recommended speed grades: -2 and -3, and only high-performance I/O banks in -1 devices. (The -1 speed grade needs special care. View the source code for recommendations.)
Source Code Provided	Yes
Source Code Format	VHDL
IP Used	No
Simulation	
Functional Simulation Performed	Yes
Timing Simulation Performed	No
Testbench Format	VHDL
Simulator Software/Version	ISIM_13.3 or later
SPICE/IBIS Simulations	No

Table 2: Reference Design Checklist (Cont'd)

Parameter	Description
Implementation	
Synthesis Tool/Version	XST 13.3 or later
Implementation Tool/Version	ISE Design Suite 13.3 or later
Static Timing Analysis Performed	Yes
Hardware Verification	
Hardware Verified	Yes
Hardware Platform Used for Verification	KC705 board

Table 3 shows a summary of the reference design utilization. The reference design contains a dual receiver. For demonstration and test purposes, a set of PRBS transmitters and a set of PRBS reception blocks is added on the KC705 board. Only the design utilization of the dual receiver is listed. The XC7K325T-2-FFG900 device is used for implementation.

Table 3: Reference Design Utilization

Component	Percentage (%)	Total Number	Number Utilized
Flip-Flops	1	407,600	191
Slices:		203,800	
As LUT	1		174
As memory	1		17
As SRL	1		17
As route-through			2
BUFG/BUFGCTRL	9	32	3
ISERDESE2	1	500	5
OSERDESE2			1
IDELAYE2	1	350	4
IDELAYCTRL	1	10	1
MMCME2 ⁽¹⁾	1	10	1

Notes:

- One MMCME2 is needed per I/O bank. One I/O bank can fit up to 19 receiver channels. One set of ISERDESE2/OSERDESE2 must be kept free for the clock phase adjustment (CDC logic).

The design uses floorplanning constraints in the UCF file. This implementation technique makes each receiver use the same amount of FPGA logic and allows easy multiplication of the design in any 7 series FPGA. For more information about this, refer to the documentation inside the reference design ZIP file.

Receiver UI and Jitter Tolerance

Given the DRU method used, two valid sampling points are required at all times. This means the starting point is 0.500 UI. The oversampling is based on evenly spaced sampling points. Any error in that spacing causes the receiver jitter eye requirement to increase.

$$\text{Receiver Jitter Eye Requirement} = \text{Eye Requirement of DRU} + \text{Sampling Phase Error}$$

$$0.625 \text{ UI} = (0.500 \text{ UI}) + (0.125 \text{ UI})$$

Sampling phase error comprises all of the effects of taking a 125 MHz clock and multiplying it up to 625 MHz, then feeding it to the two BUFIOs with a phase shift and the IODELAYE2's 200 ps phase shift.

Sampling phase error includes:

- MMCME2_ADV jitter for the exact setting in the reference design
- MMCME2_ADV phase error between CLK0 and CLK90
- MMCME2_ADV DCD
- IODELAYE2 delay accuracy (ability to create a 200 ps phase shift)
- IODELAYE2 pattern-dependent jitter
- Any offset between the two paths of the master and slave ISERDESE2

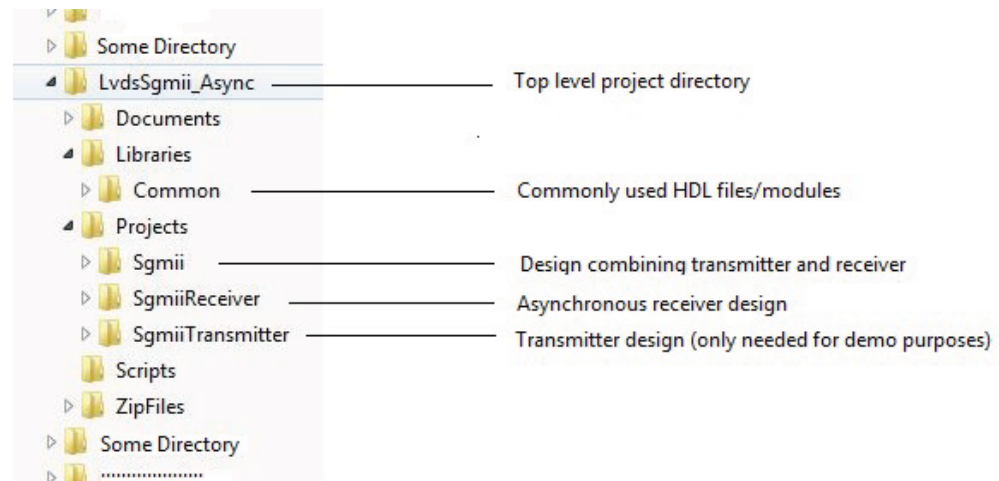
Sampling phase error does *not* include:

- Any other frequency of the clock or setting of the MMCME2_ADV
- Signal integrity losses (ISI, board jitter, etc.)
- Internal device jitter

Process, voltage, and temperature characterization are performed to qualify the interface. The allowable overall jitter tolerance is 0.375 UI.

Reference Design Directory Setup

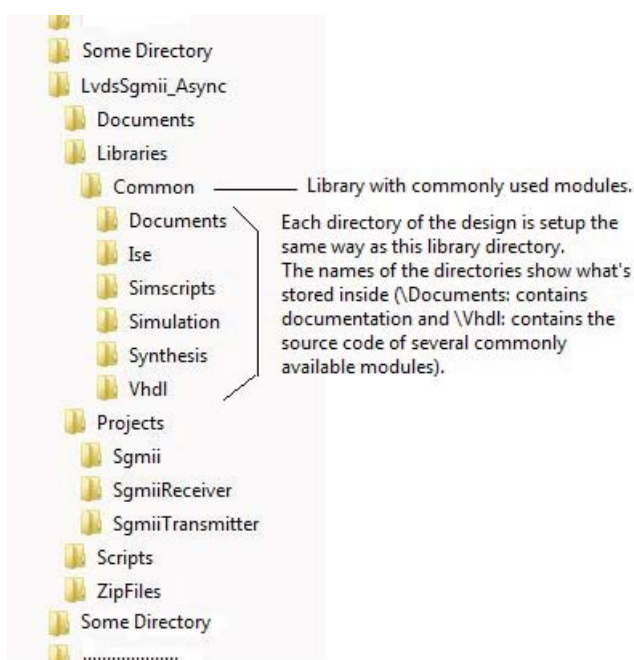
The design is highly hierarchical. This provides great flexibility and allows easy reuse of design modules. The top structure of the design is shown in [Figure 12](#).



X523_12_012012

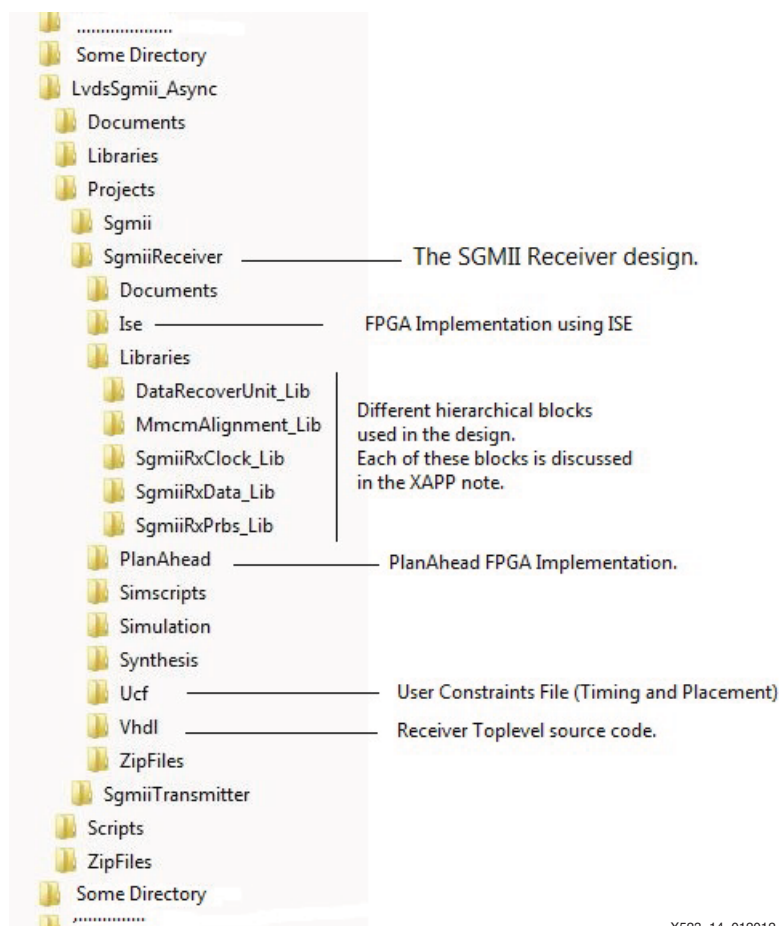
Figure 12: Top-Level Setup of the Design

[Figure 13](#) and [Figure 14](#) show the directory structure of the `Common` and `SgmiiReceiver` folders, respectively.



X23_13_012012

Figure 13: Directory Structure of Common Library



X523_14_012012

Figure 14: SgmiiReceiver Design with Directory Structure of the Libraries

`SgmiiReceiver` is the only part of the design needed for custom implementation. This is also the part discussed in this application note.

Conclusion

Xilinx 7 series FPGAs can implement asynchronous communication using SelectIO interface resources, allowing transceivers to be reserved for other uses. This implementation also helps to reduce cost by making smaller FPGA selection possible.

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
04/06/12	1.0	Initial Xilinx Release.
05/17/17	1.1	Added Zynq-7000 SoC to document title and Summary . Removed "GT" before "transceivers" in Introduction and Conclusion . Updated OSERDES and ISERDES connections in Figure 2 . Updated OSERDES connection in Figure 10 . In Figure 11 , updated BUFIO rate to 625 MHz.

Notice of Disclaimer

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>.