XILINX®

Connecting Virtex-6 FPGAs to ADCs with Serial LVDS Interfaces and DACs with Parallel LVDS Interfaces

Author: Marc Defossez

Summary

This application note describes how to utilize the dedicated deserializer (ISERDES) and serializer (OSERDES) functionalities in Virtex®-6 FPGAs to interface with analog-to-digital converters (ADCs) that have serial low-voltage differential signaling (LVDS) outputs and with digital-to-analog converters (DACs) that have parallel LVDS inputs. The associated reference design illustrates a basic LVDS interface connecting a Virtex-6 FPGA to any ADCs or DACs with high-speed serial interfaces.

Introduction

Common ADCs have a resolution of 12, 14, or 16 bits. Typically, multiple converters in a single package are translated into multiple serial data channels. These converters can be used in two modes: 1-wire and 2-wire. The ADC Interface section provides a possible solution for both modes. High-speed DACs, however, have one or two (multiplexed data) parallel LVDS input ports (see the DAC Interface section). Generally, each interface port has a bit width equal to the resolution of the DAC.

This application note and accompanying reference design provide a basic interface solution for these type of connections.

ADC LVDS Interface

Many ADCs use a serialized LVDS interface in which digital data is provided to the FPGA over one or two LVDS channels per ADC in the component package. Figure 1 shows the analog input signal along with the input, bit, and frame clocks. As shown in Figure 1, a sample N of the analog signal is converted to digital format and presented at the ADC outputs after a latency period. The analog signal is converted into a digital serial data stream with 12-bit ADC resolution that is provided together with a high-speed bit clock and a sync or frame clock.
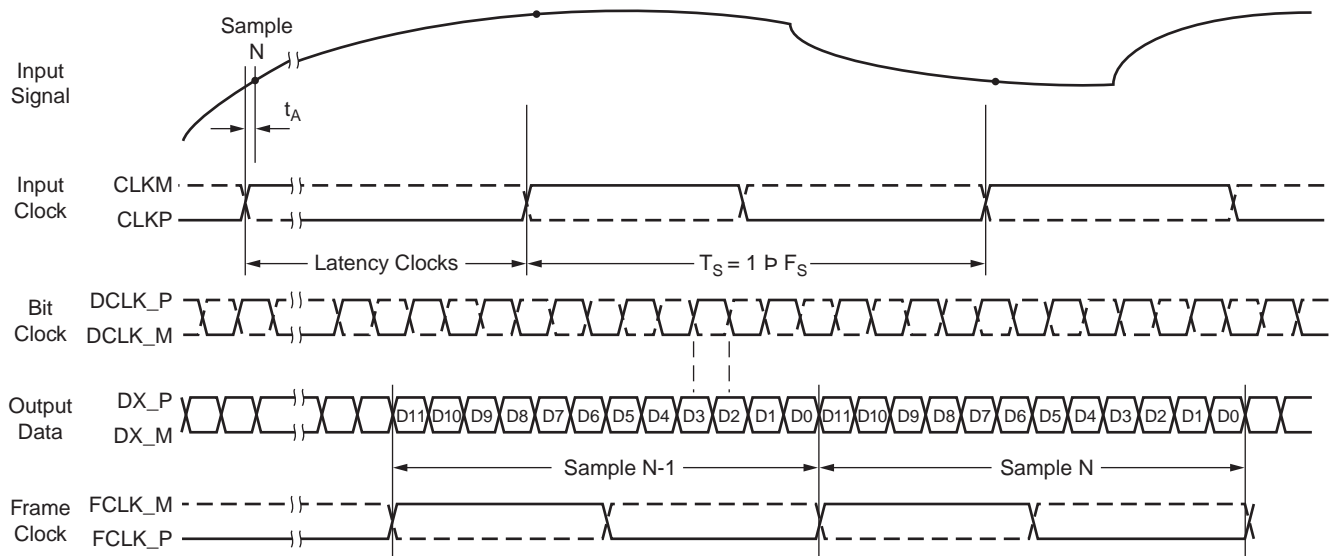
Figure 1: **Single-Channel Converter Setup**

www.BDTIC.com/XILINX

The frame clock is a digitized version of the analog sample clock. It is phase aligned with the serial data, and all data bits of a sample fit into one frame clock period. The high-speed bit clock is presented as a 90° phase-shifted signal to the data and frame clock. With these configurations, the bit clock frequency can be calculated as shown later in this section.

In 1-wire mode, there are as many data channels as converters in the package. In 2-wire mode, the data is split over two data channels per converter.

The frequency of the high-speed bit rate clock is determined by the ADC's resolution and sample rate. Thus, an ADC provides one or two data lanes per converter in the package, but only one bit clock and one frame clock.

The maximum speed of the LVDS I/O is set by the maximum possible speed of the clock toggling the flip-flops in the FPGA logic or the ISERDES components. Thus, the maximum sample speed of a single-channel LVDS ADC with 1-wire interface is limited.

Equation 1 calculates the bit clock rate for a single-channel ADC in 1-wire or 2-wire DDR mode. For example, the bit clock frequency of a 16-bit, 1-wire mode, 150 MSPS device is (16 × 150) / 2 = 1200 MHz, corresponding to a 2.4 Gb/s bit rate. These single-lane (1-wire) bit and clock rates are too high for the LVDS I/O in any FPGA speed grade. The 2-wire interface solution uses two data channels per ADC doubling the data throughput rate while lowering the bit clock rate.

$$Bit\_Clock\ (MHz) = \frac{ADC\_Resolution \times Sample\_Rate}{2 \times Wire\_Interface}$$

Equation 1

When the single-channel, 16-bit, 150 MSPS ADC is used in 2-channel mode, the bit clock rate becomes 600 MHz. This makes it possible to connect such high-speed ADCs to the FPGA.

Figure 2 shows the timing diagram of a 14-/16-bit ADC in 1-wire interface mode. Figure 3 shows the same ADC with a 2-wire interface. Data can be transmitted by the ADC over the LVDS channel with either the most-significant bit (MSB) or the least-significant bit (LSB) first, and arranged with bit or byte alignment.
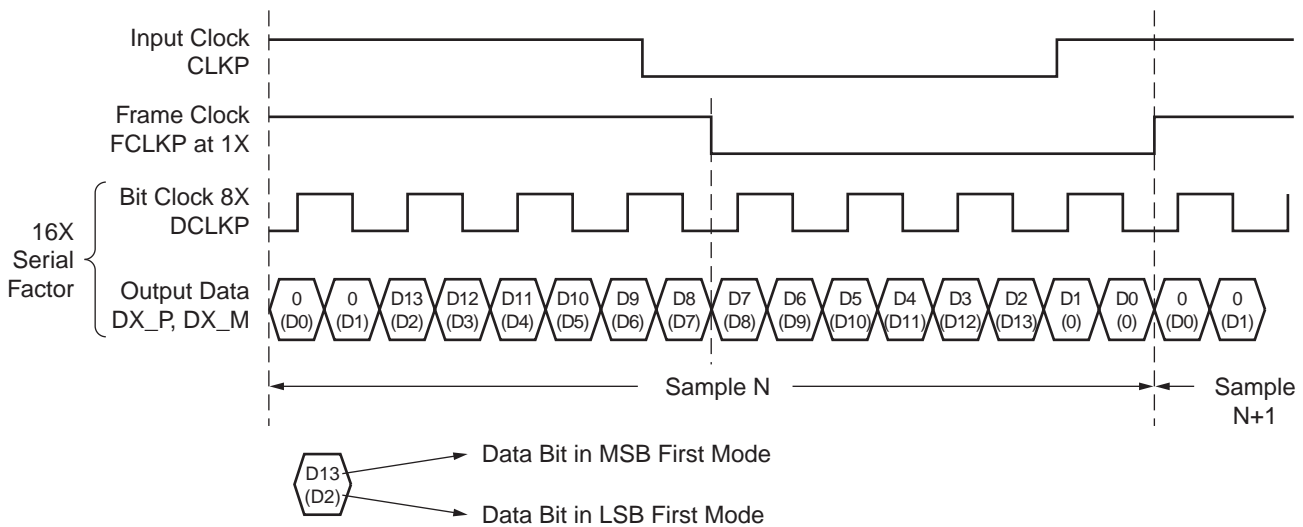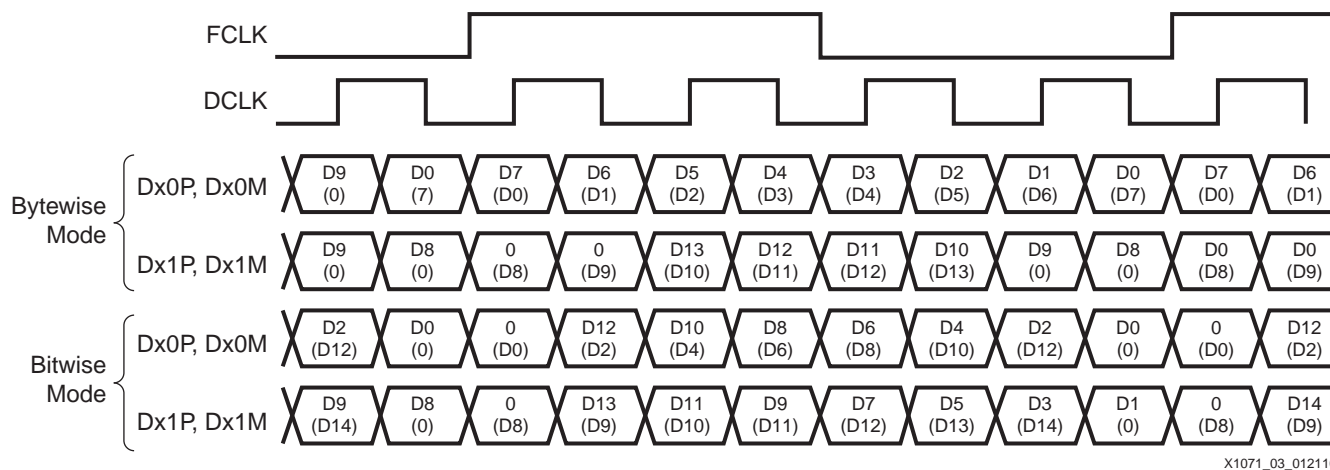

Figure 2: **14-/16-Bit ADC and 1-Wire, 4X Bit Clock Output Waveform**

	FCLK	DCLK											
Bytewise Mode	Dx0P, Dx0M	D9 (0)	D0 (7)	D7 (D0)	D6 (D1)	D5 (D2)	D4 (D3)	D3 (D4)	D2 (D5)	D1 (D6)	D0 (D7)	D7 (D0)	D6 (D1)
	Dx1P, Dx1M	D9 (0)	D8 (0)	0 (D8)	0 (D9)	D13 (D10)	D12 (D11)	D11 (D12)	D10 (D13)	D9 (0)	D8 (0)	D0 (D8)	D0 (D9)
Bitwise Mode	Dx0P, Dx0M	D2 (D12)	D0 (0)	0 (D0)	D12 (D2)	D10 (D4)	D8 (D6)	D6 (D8)	D4 (D10)	D2 (D12)	D0 (0)	0 (D0)	D12 (D2)
	Dx1P, Dx1M	D9 (D14)	D8 (0)	0 (D8)	D13 (D9)	D11 (D10)	D9 (D11)	D7 (D12)	D5 (D13)	D3 (D14)	D1 (0)	0 (D8)	D14 (D9)

X1071_03_012110

*Figure 3:* **14-/16-Bit ADC and 2-Wire, 4X Bit Clock Output Waveform**

The 1-wire interface encounters speed problems sooner than the 2-wire interface. In 1-wire mode, the reference design can support ADC resolutions up to 16 bits with sampling rates up to approximately 80 MSPS. In 2-wire mode, the reference design can support the same ADC resolution ranges with sampling rates up to 160 MSPS.

The reference design has a completely modular design approach, allowing modifications to support different frequencies, resolutions, number of channels, or a combination of all.

The settings of the ADC can be controlled and programmed through a serial peripheral interface (SPI). This interface is not discussed in this application note. However, the reference design includes a sample design that connects the FPGA through a UART-USB link to a PC to allow programming of the ADC across the SPI link.

 www.xilinx.com 3

DAC LVDS Interface

Unlike ADCs, high-speed DACs utilize parallel LVDS data interfaces. DACs use interleaved data bus structures to ensure that clock frequencies are in the LVDS I/O range of the interfacing components. In most cases, when in interleaved mode, the data input buses of two DACs in one package are combined to feed data to one ADC channel. Figure 4 shows how the data and clock can be requested at the DAC inputs. In Figure 4, the data clock is generated by the FPGA to register data using the DAC or DDAC internal clock. The data ports contain parallel data from the FPGA to the DAC, where *n* is the most-significant bit.
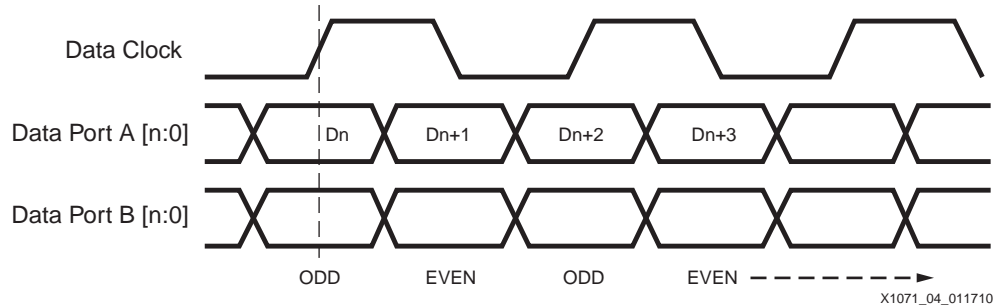

Figure 4:   **Odd and Even Data Sampling**

One LVDS lane per resolution bit is needed to send data to DACs. For interleaved bus connections, the number of I/Os doubles, as shown in Figure 4.

Although interfacing to DACs requires considerable I/Os, sending data to DACs does not require special knowledge or techniques. The most important issues for DACs are clock and clock-to-data related.

Most DAC devices provide a high-speed clock to the interfacing device (FPGA). The DAC requires that the interfacing device provides data that is phase-aligned or 90° phase-shifted at the DAC input pins. These requirements should also be kept in mind:

Internal clocking of two I/O banks simultaneously requires the use of a mixed-mode clock manager (MMCM) in the FPGA.

To present the clock and phase-aligned or 90° phase-shifted data at the DAC input pins, the clock transmitted to the DAC must be generated the same way that the data is generated. Additionally, the MMCM in the FPGA needs an external feedback path. This configuration is shown in Figure 5 and discussed in detail in DAC Clock, page 16.
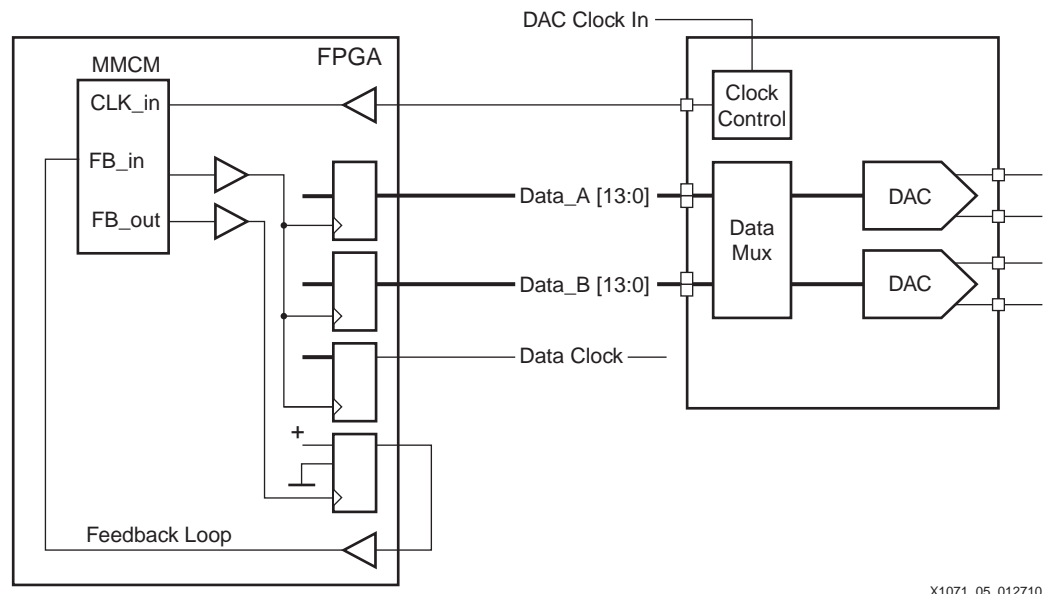
Figure 5:   **Basic Data, Clock, and Clock Feedback Setup**

www.BDTIC.com/XILINX   www.xilinx.com

ADC Interface

This section discusses various aspects of the ADC interface.

Bit Clock

The bit clock rate is determined by Equation 1. For a 16-bit, 150 MSPS ADC, the DDR bit clock rate is 1200 MHz. This rate is too fast for the Virtex-6 device's clock-capable inputs and the regional clocks used in this design. Therefore, the ADC must be used in 2-wire mode. In this mode, ADC data is distributed over two LVDS channels per converter, with the bit clock divided by two, as shown in Equation 1. For example, for a 16-bit, 150 MSPS converter, the bit clock rate is 600 MHz.

Equation 2 is used to calculate the sample rate of an ADC. For an ADC used in 2-wire mode, Wire_Interface must be set to 2 to determine Sample_Rate.

$$Sample\_Rate = \frac{Bit\_Clock\,(MHz) \times (2 \times Wire\_Interface)}{ADC\_Resolution} \qquad \text{Equation 2}$$

Table 1 shows examples of the relationship between the wire interface, the bit clock based on the ADC resolution, and sample clock parameters. Equation 1 and Equation 2 make it easy to calculate values from known parameters.

Table 1: **ADC Parameter Relationship**

Resolution (bits)	Sample Rate (MHz)	Interface Type	Bit Clock (MHz)	Comments
12	80	1-wire	480	OK
12	125	2-wire	375	OK
14	125	1-wire	875	Not OK. 2-wire mode needed
14	150	2-wire	525	OK
16	125	2-wire	500	OK
16	200	2-wire	800	OK. Needs the fastest speed grade

Notes:

1. ADCs with a 14-bit resolution often run in a 16-bit output mode. Two data bits are either dummy bits or used as overflow indicators. Clock rate calculation must then be carried using 16 bits as the resolution.

The bit clock provided by the ADC is 90° out-of-phase with respect to the data and frame signals. The designer must maintain this alignment all the way to the FPGA using good PCB layout. Because the delay from the package pad to the D input of each ISERDES is equal for all signals.

Due to routing and clock buffer delay inside the FPGA grid, the bit clock (DCLK) must be repositioned for capturing data and frame signals, as shown in Figure 6.
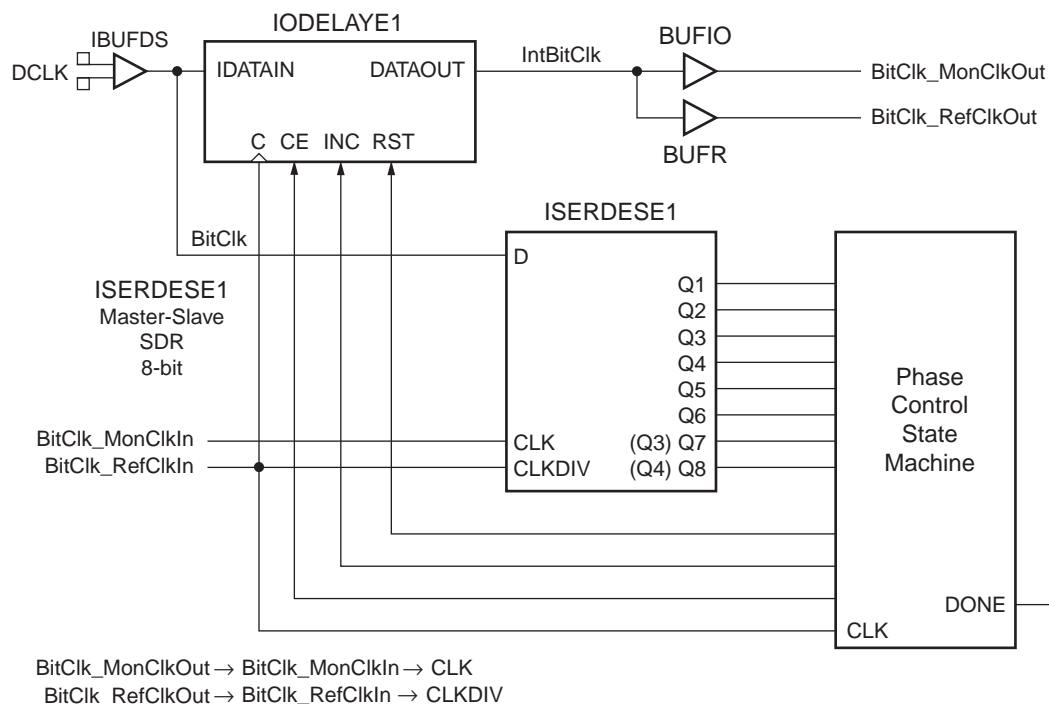
X1071_06_050310

Figure 6: **Bit Clock Alignment Setup**

In variable IDELAY mode, the bit clock (DCLK) from the ADC is routed through an IODELAYE1 from the output of a differential input buffer to a BUFIO and a BUFR (see Figure 6). DCLK becomes BitClk_MonClk (the aligned DCLK), and BitClk_RefClk becomes a reconstructed frame clock (FCLK). DCLK is also applied as data to the D input of the ISERDESE1 in the same I/O tile as the IODELAYE1.

BitClk_MonClk and BitClk_RefClk are routed to the CLK and CLKDIV inputs of all ISERDESE1 components used in the interface. Thus, they are also routed to the ISERDESE1 as inputs to DCLK.

DCLK essentially registers itself in an ISERDES using a delayed version of itself. This technique allows the designer to determine the rising and falling edges of DCLK, and thus position the CLK and CLKDIV input clocks of the ISERDES anywhere in a DCLK cycle using the bit-clock-alignment state machine. This state machine monitors the ISERDESE1 outputs and the deserialized and parallel captured clock bits. When all captured bits are equal (i.e., all 0s or all 1s), the state machine increments or decrements the IODELAYE1 to align the internal clock to the external clock. By incrementing or decrementing the number of delay taps, delay is either introduced or removed from the ISERDESE1 CLK.

Control State Machine Operation

The bit-clock-alignment state machine operates on the rising edge of CLKDIV (BitClk_RefClk) and begins with a preset IODELAY delay tap equal to 16. A Virtex-6 FPGA IODELAYE1 delay line has 32 taps of 78 ps for a total delay of 2.5 ns. The 78 ps tap corresponds to an IDELAYCTRL reference clock of 200 MHz.

When MMCMs and IDELAYCTRLs are locked and ready, the state machine begins by detecting if CLK (BitClk_MonClk) is already or nearly aligned with the input clock (DCLK) (Figure 7).
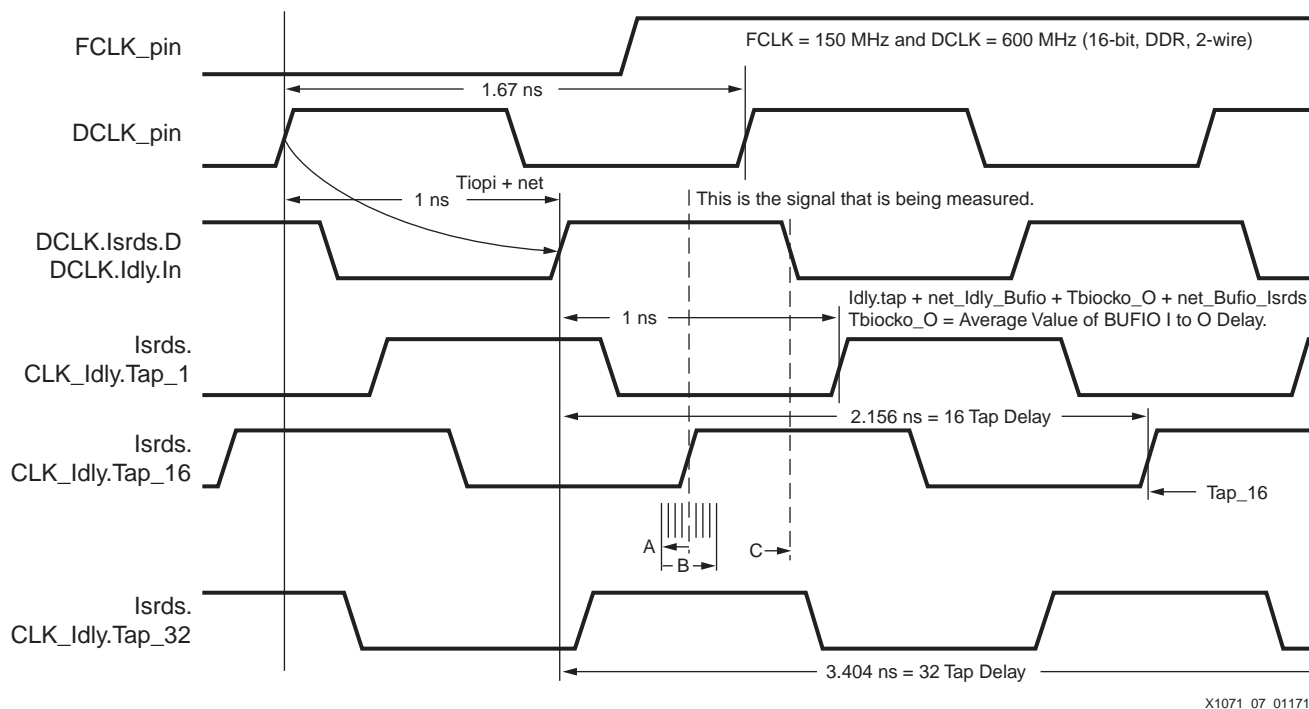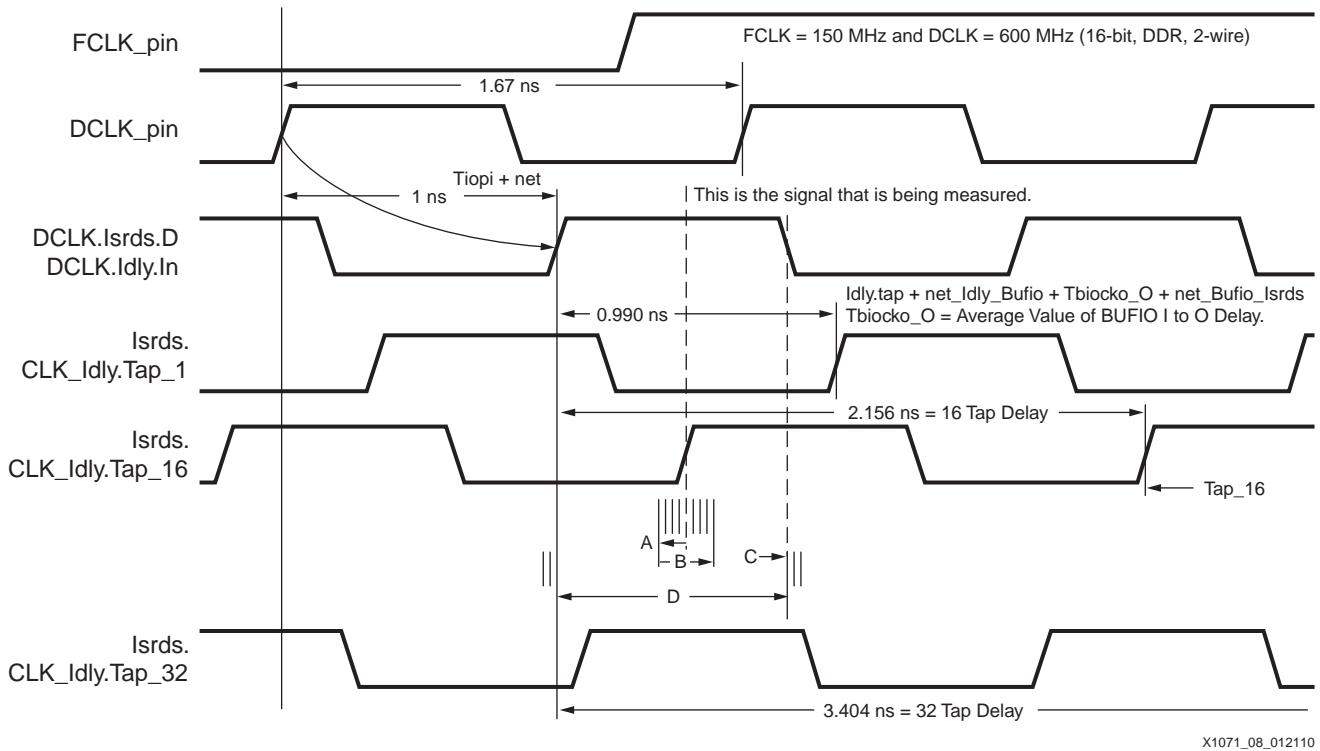
Figure 7: **Clock Alignment when the Edge is Found Immediately**

As shown in Figure 7, step A and step B are taken backwards and forwards, respectively, to ensure measuring is outside the possible jitter window at a clock edge.

If an edge is found during this operation, a counter is started and the state machine steps forward until a new edge is seen (point C). At this point, the counter represents the width of one-half of the clock period. When the edge-detecting phase takes place, the level (High or Low) of the clock is recorded as status bits. The status bits indicate what needs to be done after the clock edges are detected:

- Status bits = `010`: BitClk_MonClk is 180° out-of-phase with respect to the clock at the ISERDESE1.D (DCLK) input. To correct this phase misalignment, BitClk_MonClk must be reset to a known state.

- Status bits = `101`: ISERDESE1.CLK and the clock at the ISERDESE1.D input are in phase, and nothing needs to be done.

The most likely scenario is that no edge is found during the initial edge detection steps. The state machine steps forward until a new edge is detected (point C in Figure 8). At this point, a counter is started, and the state machine steps backwards, decreasing the delay line until a new edge is detected (point D in Figure 8). At this point, the counter represents one-half the clock period width, and the clock edges can be aligned as indicated by the status bits.

FCLK_pin

DCLK_pin

1.67 ns

FCLK = 150 MHz and DCLK = 600 MHz (16-bit, DDR, 2-wire)

DCLK.Isrds.D
DCLK.Idly.In

1 ns   Tiopi + net

This is the signal that is being measured.

Isrds.
CLK_Idly.Tap_1

0.990 ns

Idly.tap + net_Idly_Bufio + Tbiocko_O + net_Bufio_Isrds
Tbiocko_O = Average Value of BUFIO I to O Delay.

Isrds.
CLK_Idly.Tap_16

2.156 ns = 16 Tap Delay

Tap_16

A
B
C
D

Isrds.
CLK_Idly.Tap_32

3.404 ns = 32 Tap Delay

X1071_08_012110

Figure 8:   **Probable Clock Alignment Procedure**

This clock alignment method measures half of a clock period and can align a clock as low as 400 MHz (78 ps/tap $\times$ 32 taps = 2.496 ns). A margin must be added to this frequency to account for the fact that ISERDESE1.CLK is not always perfectly positioned. Therefore, assuming a 110 MSPS, 16-bit DDR ADC in 2-wire mode, 440 MHz is the lowest practical clock frequency.

If ADCs with slower sample rate clocks are used, half the clock period is larger than the 16 taps (1.25 ns or one-half the delay line). In this case, two possible situations arise for the state machine to find the clock edges:

• Only one edge of the input bit clock is found, and the ISERDESE1.CLK edge must be aligned to that edge. If the status bits equal `110`, the aligned clock is 180° out-of-phase. If the status bits equal `001`, the aligned clock is in phase with the input clock.

• No edge of the input bit clock is found. If the status bits equal `111`, the clock must be moved to the 0-tap delay. If the status bits equal `000`, the clock must be moved to the 32-tap delay.

Frame Clock

The LVDS frame clock from the ADC is a slow-running clock that is phase aligned with the data. It is a digitized version of the sample clock with a known and regular pattern. The frame clock is used to train and align the data captured in the FPGA. Figure 9 is a block-level view of the frame clock alignment circuit.
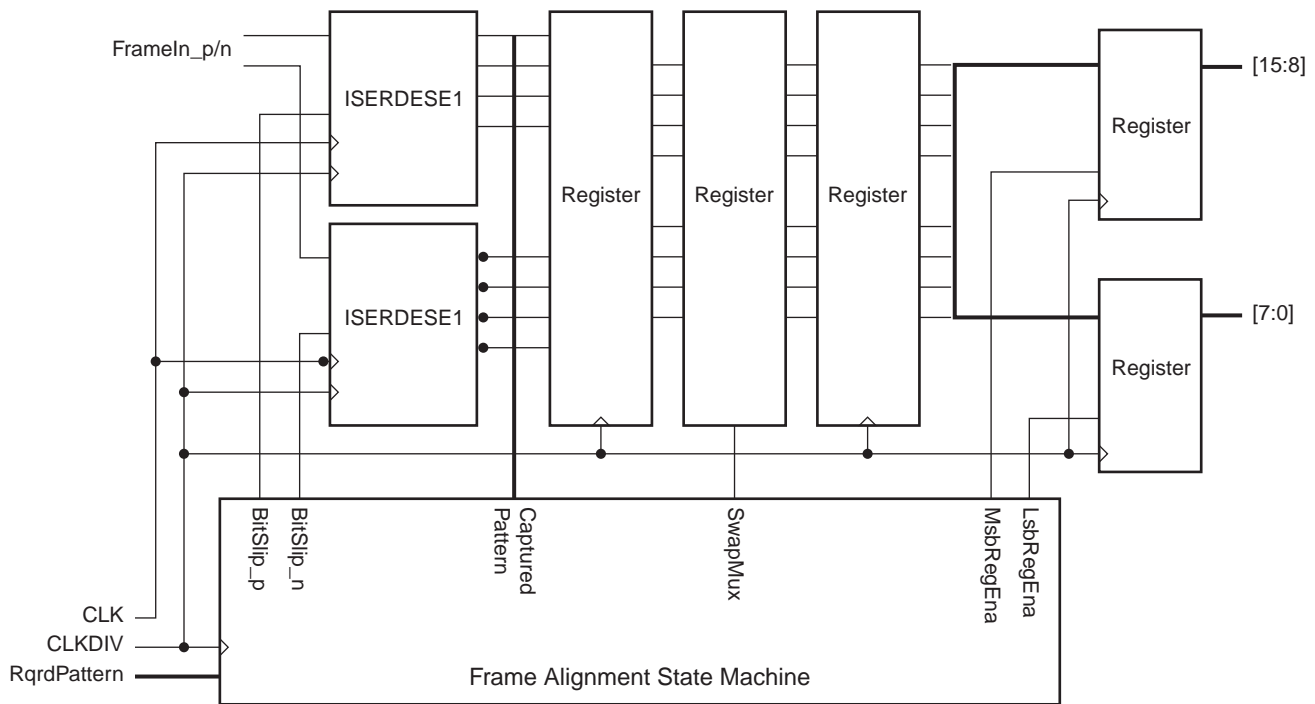
Figure 9: **Frame Clock Pattern Capture and Adjustment Circuit**

Two ISERDESE1 components are used in Networking single data rate (SDR) mode. Each ISERDESE1 is clocked on a different edge of the aligned bit clock (DCLK). Alignment of the frame clock and the data starts when the bit clock (DCLK) is properly aligned.

The output of the two ISERDES components is compared to a fixed value representing the frame clock pattern. When this fixed value is not equal to the output of the ISERDESE1, a Bitslip operation is initiated for both the frame clock and data. The Bitslip operation is applied until the output of the frame clock ISERDESE1 components matches the given frame clock pattern. When this output is equal to the programmed pattern value, the Bitslip operation is stopped for both frame clock and the data. The data and frame clock signals within the FPGA are then considered valid. However, the data and frame clock signals are not always valid.

After the execution of a Bitslip operation, the construction of the ISERDESE1 sometimes allows the ISERDESE1 to output the same value as from the preceding cycle. For normal source-synchronous designs, where training patterns can be executed in the data, this is not an issue. The only consequence is that an extra Bitslip operation is needed to achieve synchronization.

If the ADC interface runs the frame synchronization without a training pattern, the ISERDESE1 that outputs the same value twice puts the whole synchronization process in jeopardy. When double outputs appear, the frame capturing circuit operates as if it is synchronized to the frame pattern, when in reality it has not.

The result of this erroneous frame synchronization appears in the captured data. The data capturing ISERDESE1 is shifted along with the frame ISERDESE1. Thus, when the frame assumes synchronization, the data values are wrong. Measures are taken in the interface logic to avoid this issue.

Frame Pattern Capturing

The frame clock is captured as a data stream with repeating data. Because the frame channel and data channel are phase-aligned, the frame clock is well suited for use as a training pattern or synchronization lane for the real data lanes. When it is possible to adjust the captured frame

www.BDTIC.com/XILINX

pattern so that it corresponds to a given pattern (i.e., the pattern it normally must have), the received data is ensured to have the correct format.

When the ADC interface operates in 2-wire mode, the frame patterns appear as shown in Figure 10 (16-bit, DDR, 2-wire). When the interface aligns the captured frame data so that the fixed pattern is detected, the data channels also receive correct data.
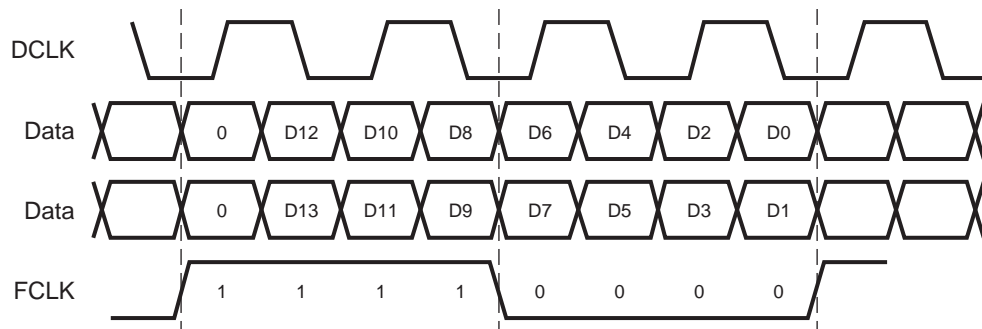
X1071_10_011710

Figure 10: **Frame Clock Pattern**

It is unknown what bit is captured first. Therefore, it is necessary to capture eight bits, then compare that eight-bit value with the required value. If the values are not equal, a Bitslip operation is started.

The DDR input is captured by two ISERDESE1 components configured in SDR mode. One ISERDESE1 captures data on the rising edge of CLK, and the other captures data on the falling edge of CLK.

To perform a valid Bitslip operation, the operation must first be applied to one of the two components.

When the captured value in both ISERDESE1 components does not equal the desired frame pattern, a Bitslip operation must then be applied to the second ISERDES1. This can be thought of as a *ping-pong* Bitslip operation. When the captured value equals the desired frame pattern value, the Bitslip operation is stopped, and correct frame and data values are captured.

The frame capturing logic is organized the same way as the data capturing logic. Thus, a register, multiplexer, register, and word-assembling register are placed after the ISERDESE1 to allow the frame pattern to be checked by an application. This logic is not necessary. The only logic required is the frame state machine.

Bitslip Operation

When a Bitslip operation is performed, the serial-to-parallel data captured by CLK in a parallel register (operated by CLKDIV) is not registered at the correct moment, and the serial-to-parallel data is captured one CLK cycle too late. The result is the serial-to-parallel register shifts one extra bit into the register, while losing a bit at the other end. Therefore, the data captured in a parallel register appears as if the data is shifted by one bit.

Figure 11 shows the capturing of bits without a Bitslip operation. When eight bits are captured in the serial-to-parallel register, the data is transferred to the parallel output register ISERDESE1 (Q outputs). The serial-to-parallel register is clocked by a high-speed CLK, and the parallel register is clocked by CLKDIV. To capture eight bits, the frequency of CLKDIV is CLK/4, so that the correct data is loaded at the right time.

In the example in Figure 11, data capture starts with the bit of value 7. Bits are shifted into the serial-to-parallel register at the CLK clock rate. The vertical stacked blocks represent that register. These blocks show that the bit of value 7 is shifted in first and then ends at the bottom. The last bit shifted in is the bit with value D.

Next at the CLKDIV rising edge, the entire serial-to-parallel register is transferred to a parallel output storage register. That register then contains the value DCBA0987. Consecutive new data is shifted into the serial-to-parallel register and transferred to the parallel output register. Thus, after each rising edge of CLKDIV, the ISERDESE1 output reflects the eight bits shifted into the serial-to-parallel register with CLK.
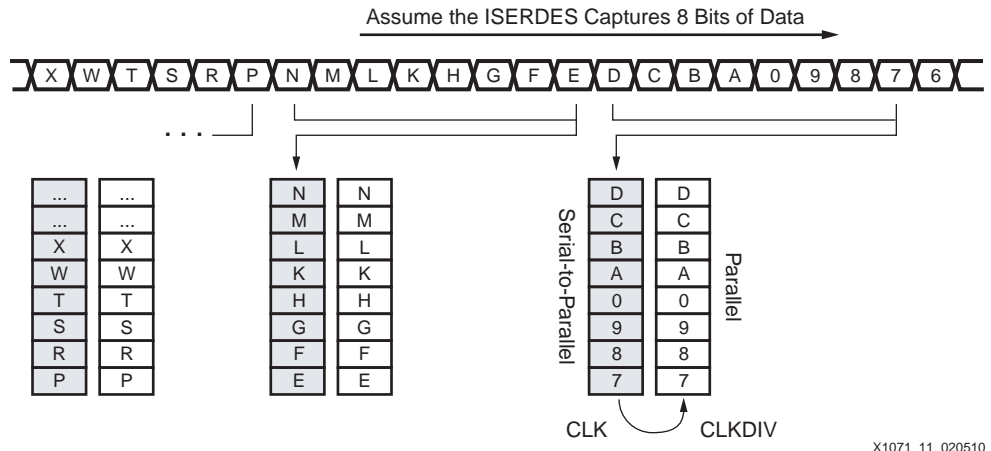

Figure 11: **Pattern Capturing without a Bitslip Operation**

Figure 12 shows the same behavior except that a Bitslip operation is executed when the second byte is captured in the serial-to-parallel register. Thus, the data is not transferred to the parallel register after the eighth bit is received, but after the ninth bit is received. It appears as if the pattern is shifted by one bit. The first bit shifted in is lost at the bottom (end) of the serial-to-parallel register.
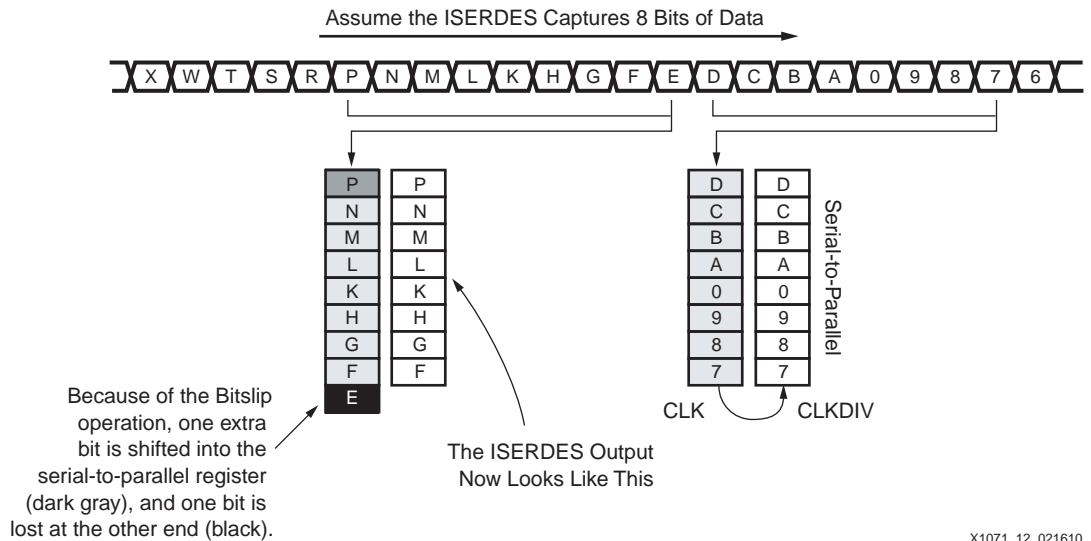

Figure 12: **Pattern Capturing with Bitslip Operation**

Sometimes, the ISERDES outputs the same data twice. This action has no effect on regular source-synchronous interfaces that synchronize through data training patterns. Only synchronization is delayed. However, the ADC interface is significantly impacted by an ISERDES outputting the same data twice. The ADC interface is synchronized using the frame clock signal. All other signals and data inputs are shifted along with the frame pattern while it is synchronizing. Therefore, when something happens to the frame signal, the data is impacted. The captured data is corrupted when the frame circuit erroneously assumes it is synchronized to the frame pattern.

Because the ISERDES Bitslip operation does not occur for both ISERDES components simultaneously, one ISERDES can output the same data twice before the other ISERDES. Thus, synchronization is not possible.

Figure 13 shows the duplicate outputting of data.
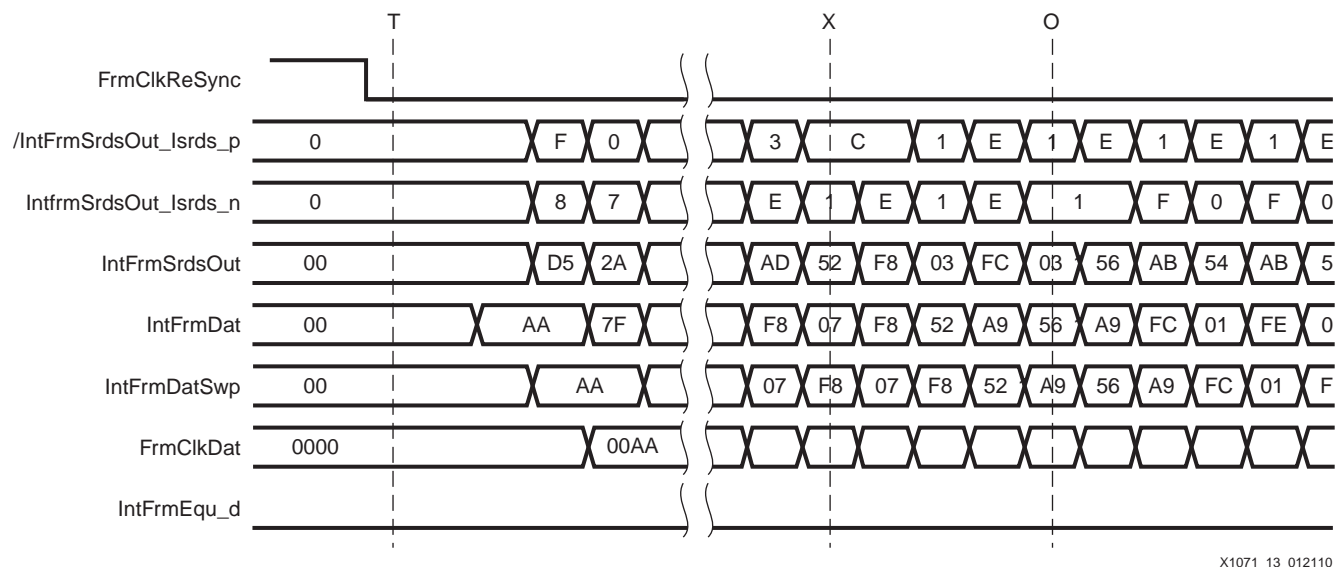
Figure 13: **ISERDES Outputting the Same Value Twice**

These operations are illustrated in Figure 13:

- After a couple of Bitslip operations, the output of the ISERDES components is 3 and E, resulting in the byte AD.
- The next output is C and 1, resulting in a byte 52.
- The ISERDES indicated by "_p" receives a Bitslip request and executes it. The result should be 1 and E, resulting in A9. Instead, the ISERDES outputs C a second time, and the resulting byte is now F8 (C and E). A nibble of the received byte did not receive the Bitslip request when it should have and is delayed one CLKDIV period, resulting in a mixed-up byte.
- On the next CLKDIV edge, the earlier Bitslip operation occurs at the output of the ISERDES, resulting in 1 and 1 or 03.

Data

The ADC can provide digitized data in 1-wire or 2-wire mode. The 1-wire interface transmits data to the FPGA using a single differential LVDS pair, while 2-wire mode uses two LVDS channels per converter.

Converters use different bit or byte organizations when transmitting data over one or two LVDS data pairs. Data can be arranged MSB or LSB first and be bit or byte oriented. Many ADCs can be configured via the SPI interface to use a selection of these modes.

To accommodate for as many ADCs as possible, the data interface can be configured to use these parameters. The basic data interface has two LVDS data inputs. Therefore, it can be configured as a single data channel in 2-wire mode or a dual data channel in 1-wire mode.

The data output of a basic interface follows the 1-wire or 2-wire selection. In 1-wire mode, the 32-bit output bus is split into two 16-bit output buses. The lower 16 bits are Channel 0, and the upper 16 bits are Channel 1. In 2-wire mode, the 32-bit output represents the same data in the upper and lower 16-bit segments. In the reference design, the upper 16 bits (MSBs) are used (Figure 14).
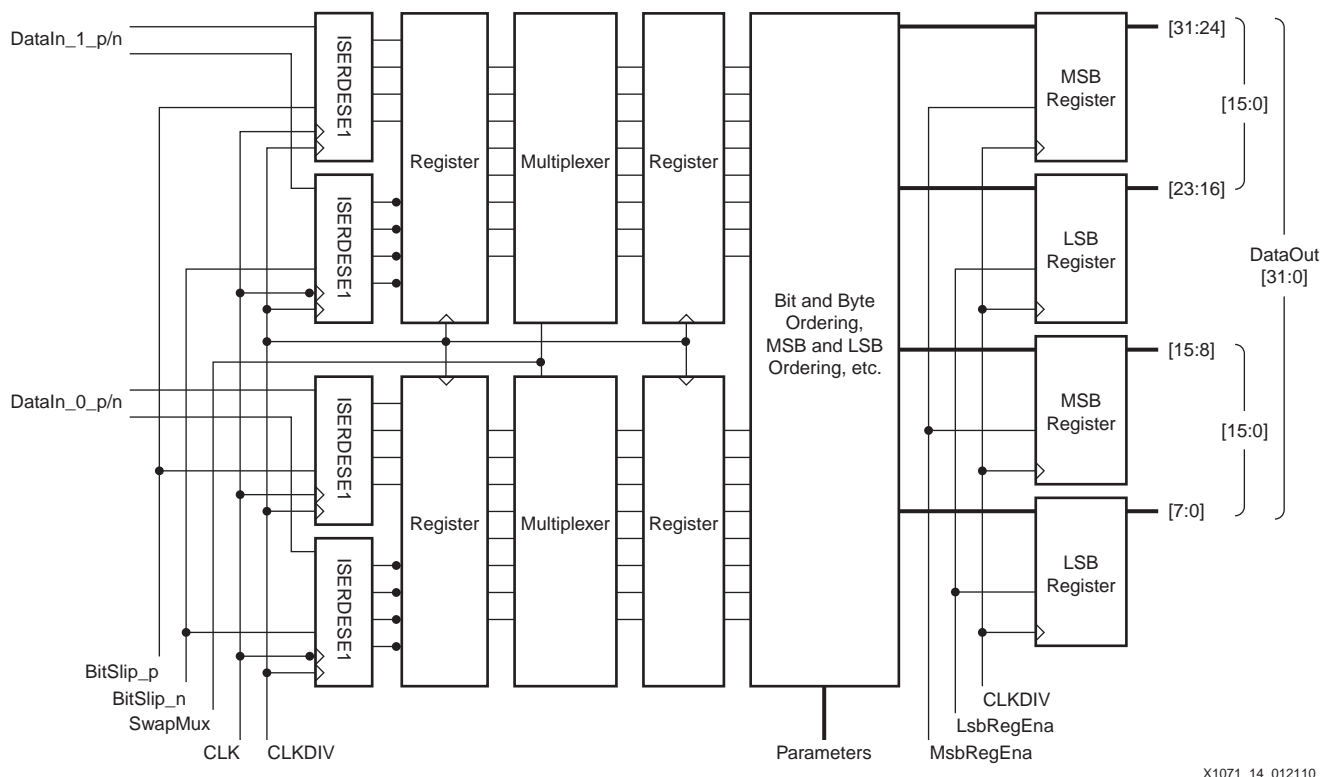
Figure 14: **Block Diagram of a Basic Data Interface**

Alignment is done using the frame clock as pattern training data. The frame clock is phase-aligned with the data from the ADC. When the frame clock is received and the frame pattern is recognized, the received data is also aligned because it is shifted with the frame signal.

Data Inversion and Bit Swap Multiplexer

The DDR data from the ADC is received by two ISERDESE1 components, each in SDR mode. This configuration is possible when a differential input buffer is used with a differential output into the FPGA logic (IBUFDS_DIFF_OUT).

Data in the ISERDESE1 serial-to-parallel registers must be captured on the rising and falling edges of CLK.

The even bits (0, 2, 4, 6, 8, 10, 12, and 14) are captured on the rising edge of CLK, and the odd bits are captured on the falling edge of the clock.

At the output of the differential input buffer, the p-branch represents the through value of the bit, and the n-branch represents the inverted value of the data bit. Therefore, the n-side captured data must be inverted (Figure 16). The inversion can be done at the IOB of Virtex-6 FPGAs. The disadvantage to the inversion at the IOB is, at high speed, a phase shift is created between the p-side data and the n-side data due to the delay of the inverter in the IOB. Thus, the best location for the inversion is at the outputs of the ISERDESE1 (Figure 15). The needed inverter can be absorbed in the used logic.
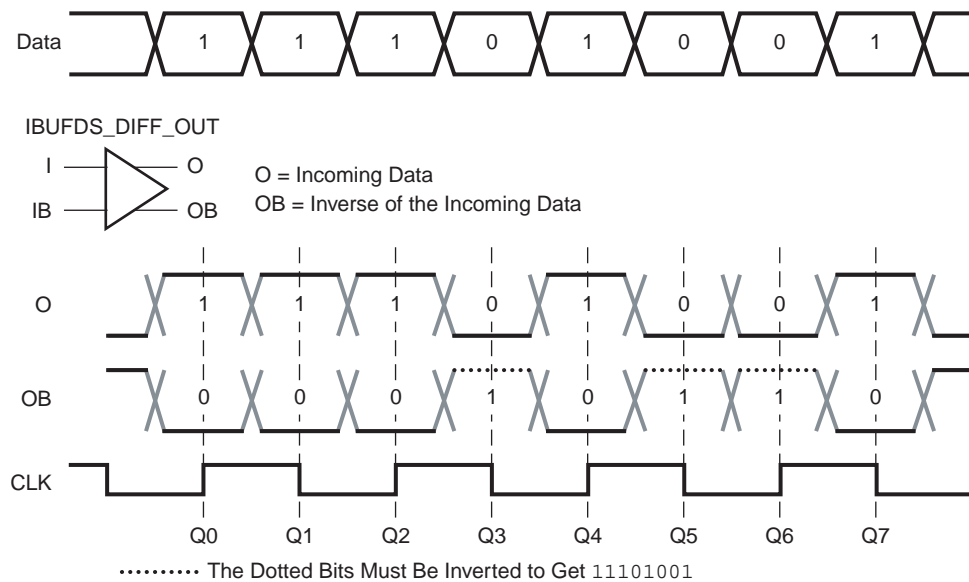
Figure 15:   **The Inverter at the ISERDESE1 Output**

It cannot be guaranteed that the even bits are captured on the rising clock edge and the odd bits are captured on the falling clock edge. The even bits could be clocked into the p-side ISERDESE1 on the falling clock edge and the odd bits could be captured on the rising clock edge. Figure 16 illustrates this scenario and shows how the multiplexer reorders the bits.
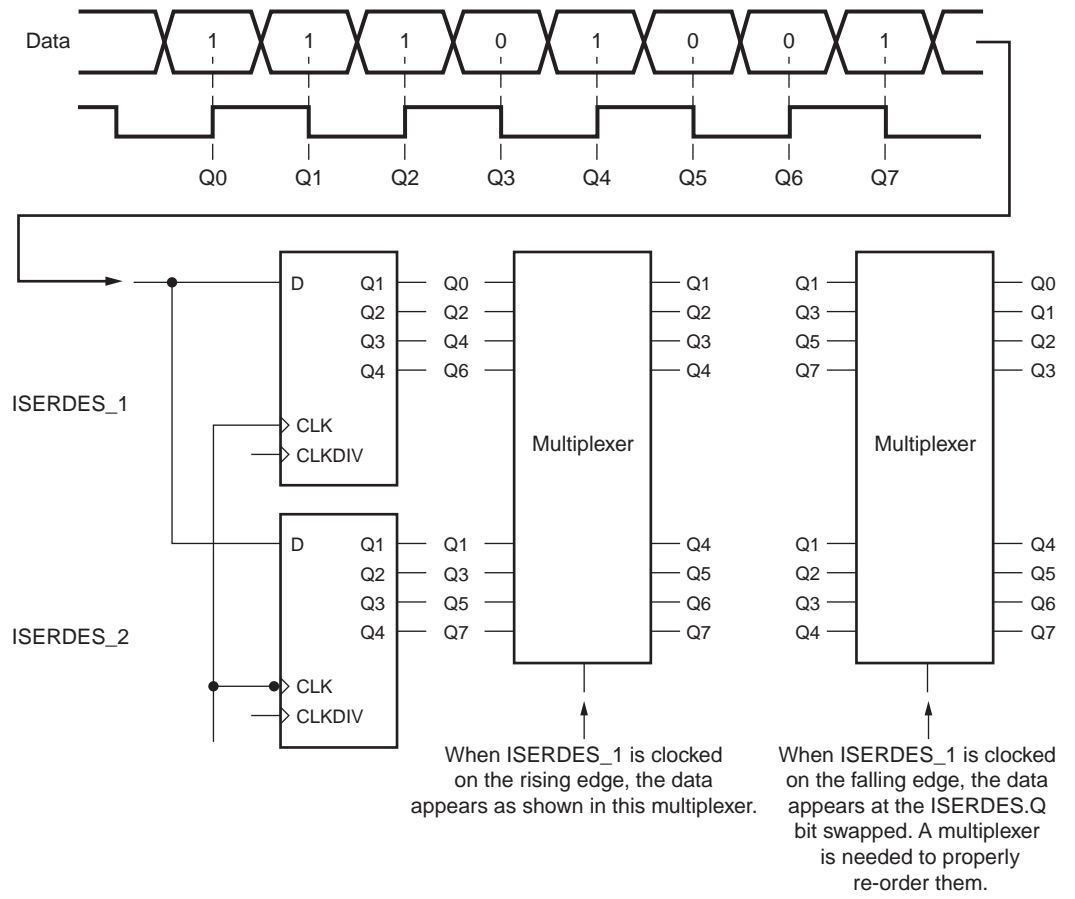
X1071_16_012110

Figure 16: **Captured Data and the Function of the Multiplexer**

www.xilinx.com

DAC Interface

This section discusses the DAC interface.

DAC Clock

A high-speed DAC delivers a clock at the rate it expects to get data back from the interface. Because most high-speed DACs require at least two I/O banks, the clock from the DAC is provided to the MMCM in the FPGA. This provides several advantages:

• The MMCM reduces jitter, when present, from the incoming DAC clock.

• The MMCM can provide all clocks needed for the DAC interface.

• The MMCM, through an external feedback loop, can phase align or phase shift (90° or other) the data provided to the DAC at the input pins of the DAC on the PCB.

Figure 17 shows an MMCM setup. The high-speed clock from the DAC is routed to a clock-capable I/O input in one of the I/O banks used for data and clock connections to the DAC. The MMCM is placed closely to FPGA logic behind the I/O bank and spans the needed clock areas (an RLOC or LOC attribute might be needed).
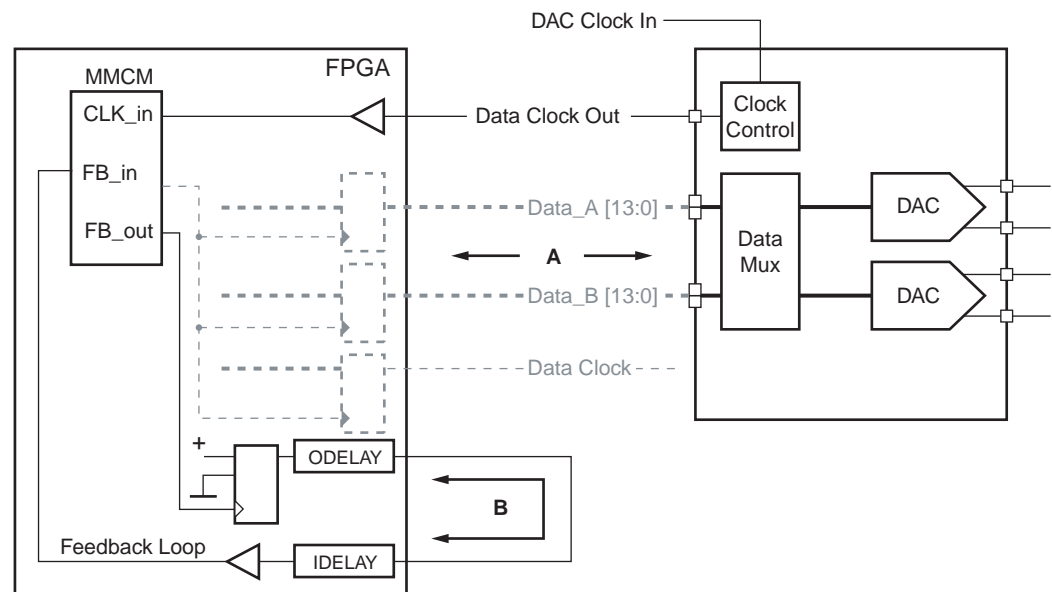
X1071_17_042010

Figure 17: **Basic MMCM and Feedback Setup**

The feedback loop needed by the MMCM is brought onto the PCB via an LVDS-configured I/O best placed in the middle of both I/O banks (at the bottom I/O of the top-oriented bank or at the top I/O of the bottom-oriented bank).

The feedback path on the PCB must have the same length or twice the length, depending the DAC used, as the data connections from the FPGA output pins to the DAC input pins. The feedback signal is taken back into the FPGA via an LVDS-configured, clock-capable I/O. As explained next, dynamic tuning of the feedback path is also possible when using ODELAY, IDELAY, or both in variable mode (see Figure 17).

Some DACs have features that allow the user to adjust the clock feedback loop, and thus the data presence at the DAC input pins. These features can also be added in the FPGA if adjustments are necessary. Figure 18 shows an IODELAYE1 component used as ODELAY in the output path of the clock feedback and used as IDELAY in the input path of the feedback. Either both components (ODELAY and IDELAY) or one of the two (ODELAY or IDELAY) can be used.

Both delay lines can be controlled by a PicoBlaze™ or other processor, or by a state machine.

The user wires all CNTVALUEIN inputs of the IODELAYE1 components into a five-bit bus connected to an output of a PicoBlaze processor. The user wires the RST input of each IODELAYE1 together in a bus (eight bits control eight RST input pins).

The tap delay value on CNTVALUEIN is loaded into the IODELAYE1 component on the rising edge of CLK when RST is High. When RST is released, the new CNTVALUEIN value is applied and is part of the delay chain.
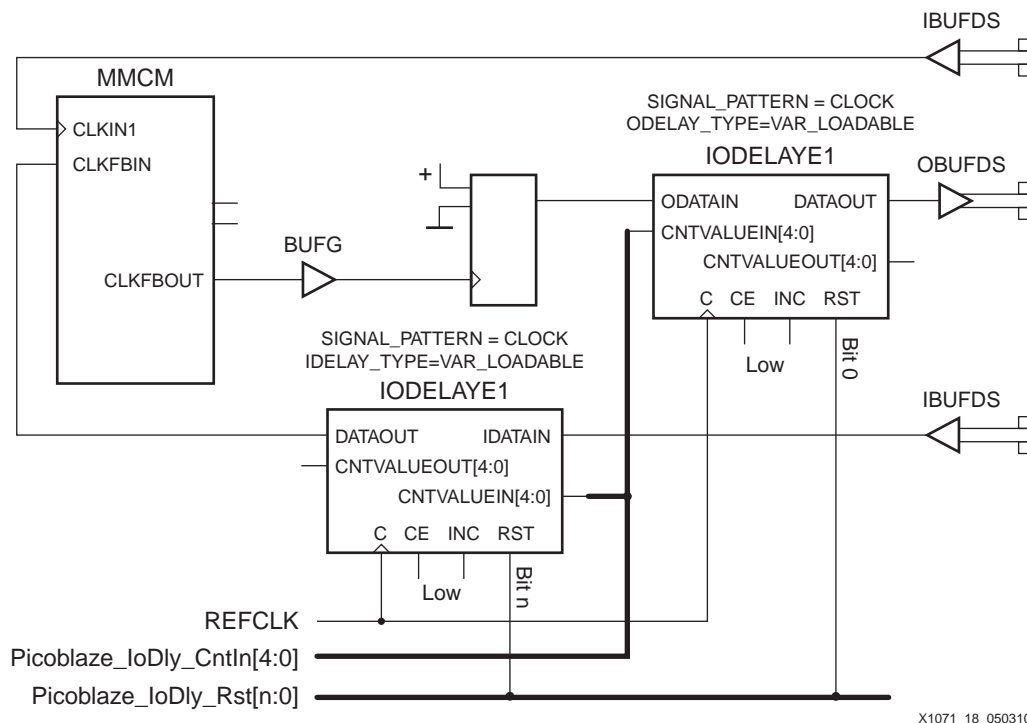

Figure 18: **IODELAY and ODDR Use in the MMCM Feedback Loop**

In Virtex-6 FPGAs, the output delay blocks are also dynamically tunable. An ODELAY can provide the same function as the IDELAY. In addition, the ODELAY can be directly loaded by a delay value or counter value, which reduces the adjusting time of the interface. Both solutions can be used at the same time.

To account for the switching parameters of the Virtex-6 FPGA I/O (OSERDES), the feedback loop can pass though an ODDR (Figure 18).

Data Clock

In some DACs, data must arrive at the DAC input pins phase-aligned to or 90° phase-shifted from the DAC clock. For these devices, the techniques discussed in DAC Clock, page 16 can be used.

Other manufacturers, such as Analog Devices, provide DACs that not only deliver a clock to the FPGA, but also require the interface to deliver a clock and data signal (phase-aligned or phase-shifted) as shown in Figure 19.
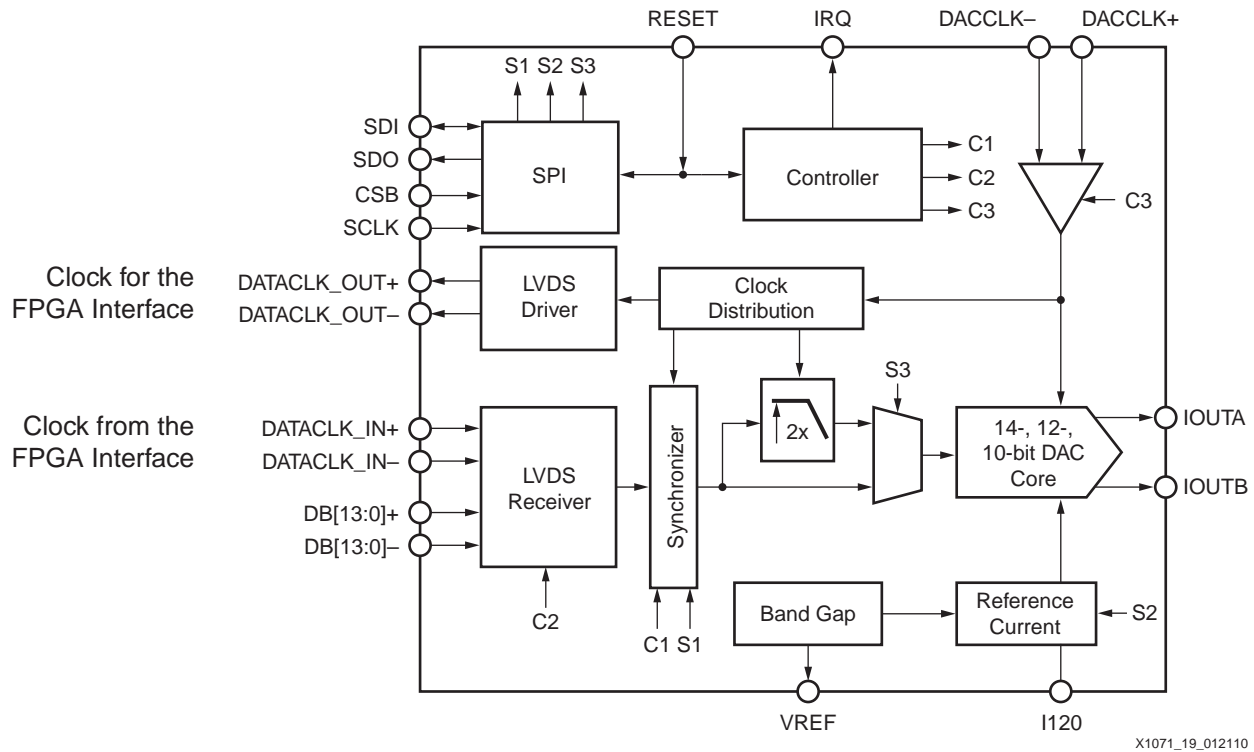
Figure 19: **DAC Requiring a Data Clock**

In this case, the data and clock must be generated the same way in the FPGA interface to guarantee a strict relationship between the two. This strict relationship can only be achieved when the clock uses the same logic as the data. Thus, the clock must be generated using an OSERDESE1. This clock generation technique, shown in Figure 20, is an extended version of that used with an ODDR.

Clock Generation Using an OSERDESE1

The generated clock cycle can be fixed or programmable. When the clock cycle is fixed, the OSERDESE1 data inputs should be tied High or Low to match the desired clock cycle. When the clock cycle needs to be programmable, the OSERDESE1 inputs can be connected to a ROM. The ROM can be cycled through its contents when new clock formats are needed.

One rising CLKDIV edge is enough to start clock generation (refer to OSERDESE1 for more information). In nearly all designs, CLKDIV is a true clock signal in the FPGA.

The clock output is usually in phase with the data. Timing for all OSERDESE1 components in an FPGA is the same. When OSERDESE1 components are used to generate data and clock, the outputs of the FPGA are in phase.

Often, it is required to apply a known phase shift to the clock as opposed to the data. For a fixed clock output, when the 1 and 0 inputs are swapped, the generated clock phase shifts over 180°. When an OSERDESE1 is clocked using a phase-shifted clock from the MMCM, the clock can be phase shifted over 90°. The MMCM should generate a 90° phase-shifted clock for the OSERDESE1.CLK input. In output delay mode, an IODELAYE1 located in front of the ISERDESE1 can dynamically shift the output to where it is needed.

Figure 20 shows how an OSERDESE1 in Master-Slave DDR configuration can be used to generate the data clock.
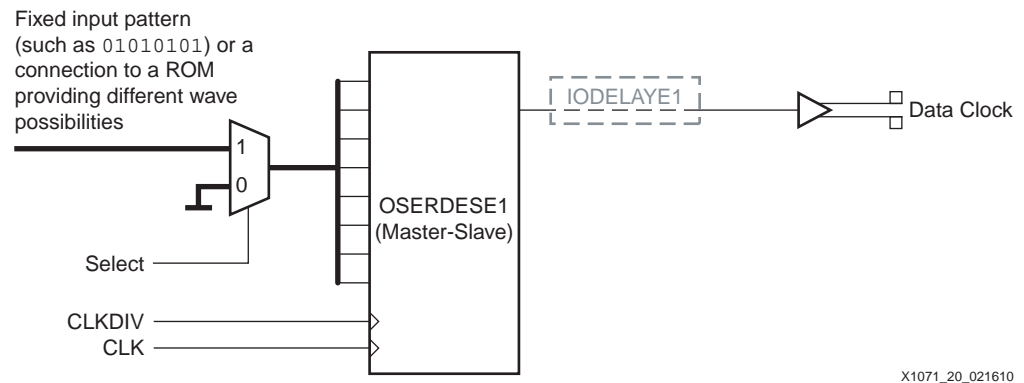

Figure 20:   **Clock Generation Using an OSERDESE1**

OSERDESE1

An OSERDESE1 is basically a parallel input register followed by a parallel-to-serial shift register. Data is loaded into the parallel register on the rising edge of CLKDIV and shifted out at the CLK rate.

An internal state machine controls the connection between the two registers and the data transfers from the parallel register into the shift register. The CLK, CLKDIV, and state machine operation are related by the input width (DATA_WIDTH) and use mode (SDR or DDR).
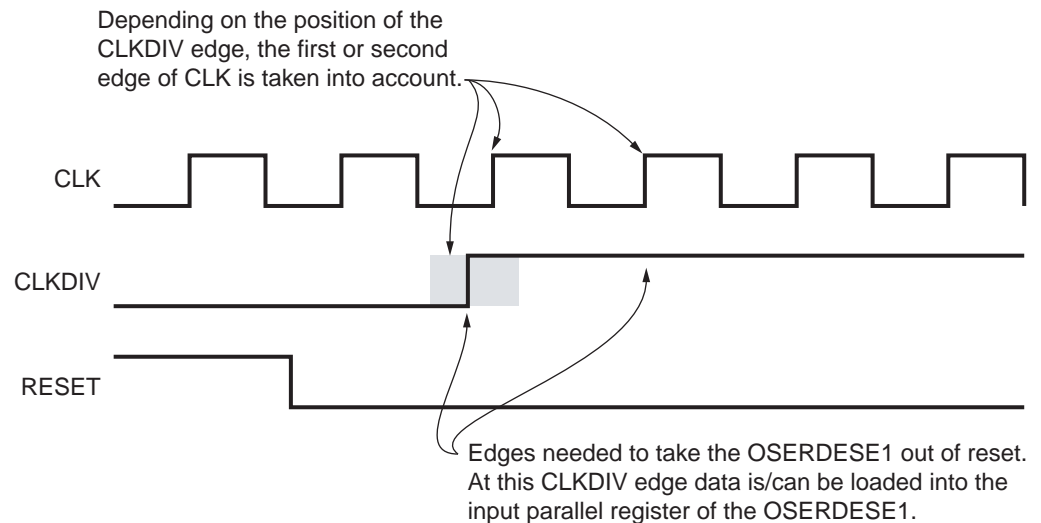
An OSERDESE1 can be set up as a:

- Master or master-slave
- 4-, 6-, 8-, or 10-bit input in DDR mode (8- and 10-bit inputs are only available in the master-slave configuration)
- 2-, 3-, 4-, 5-, 6-, 7-, or 8-bit input in SDR mode
- High-Z capable output buffer (OBUF)

For the DAC interface, the OSERDESE1 is used in master-slave, 8-bit DDR mode. The CLKDIV rate must be set at one-fourth the CLK rate.

When using the OSERDESE1, these points should be considered:

- The OCE input is not an enable for the data inputs (parallel input register) of OSERDESE1. The OCE pin enables the MSB output registers on the serial shift output side.
- After releasing the OSERDES reset, a rising CLKDIV edge followed by a rising CLK edge is needed before something can happen, as shown in Figure 21.

Depending on the position of the
CLKDIV edge, the first or second
edge of CLK is taken into account.

CLK

CLKDIV

RESET

Edges needed to take the OSERDESE1 out of reset.
At this CLKDIV edge data is/can be loaded into the
input parallel register of the OSERDESE1.
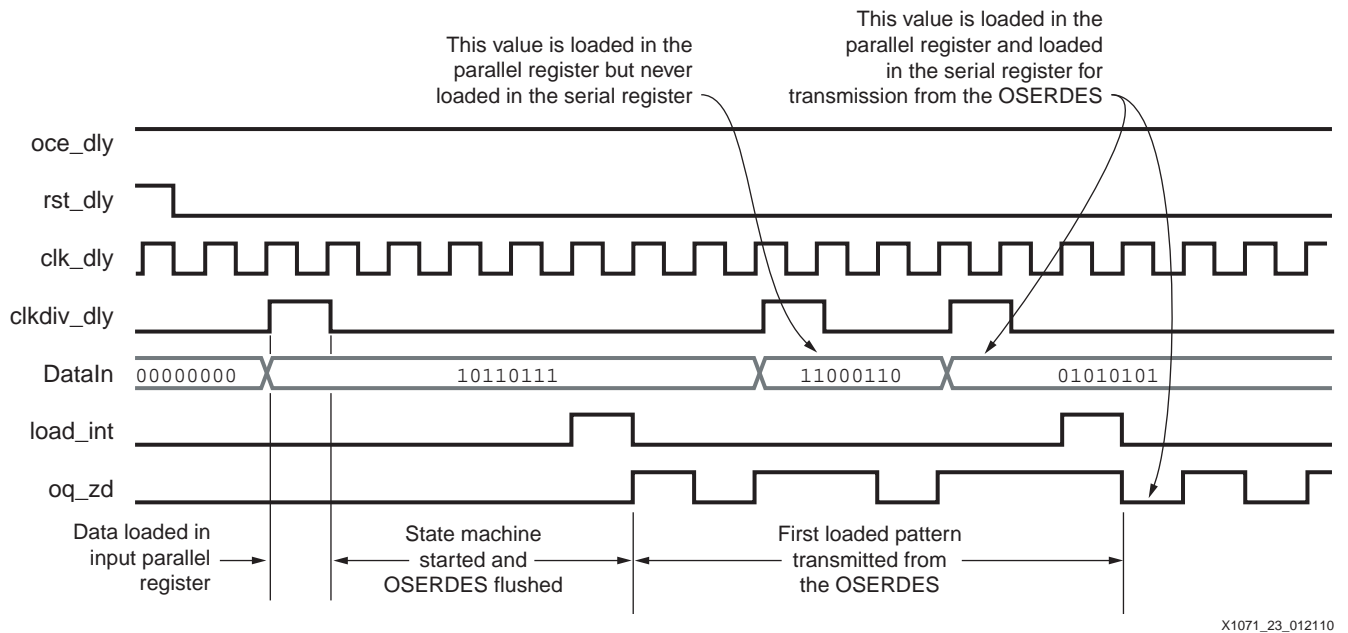
X1071_21_012110

Figure 21: **Bringing the OSERDESE1 to the Reset State**

- Data does not immediately flow out of the ISERDESE1 after being loaded into the ISERDESE1.

- The rising CLKDIV edge has loaded data in the parallel input register, but the internal state machine that transfers this data in the parallel-to-serial shift register must first generate a load pulse. Because this happens just after reset is released, the ISERDESE1 transmits all 0s until the load pulse occurs.

- The internal state machine that transfers data from the parallel input register into the parallel-to-serial register depends on the DATA_WIDTH attribute. After reset is released, the OSERDESE1 internal state machine always flushes out the same number of bits. The number of bits depends on the DATA_WIDTH and DATA_RATE attributes of the OSERDESE1.

- After the first load following reset, data is shifted in regular patterns.

- Figure 22 shows an OSERDESE1 in 8-bit DDR mode (master-slave). After reset is released and data is loaded into the OSERDES on the rising CLKDIV edge, the data appears at the output in four CLK cycles. It takes four CLK cycles because eight bits are loaded, and the controller first shifts the eight previous bits (all 0s), out at the DDR CLK rate.
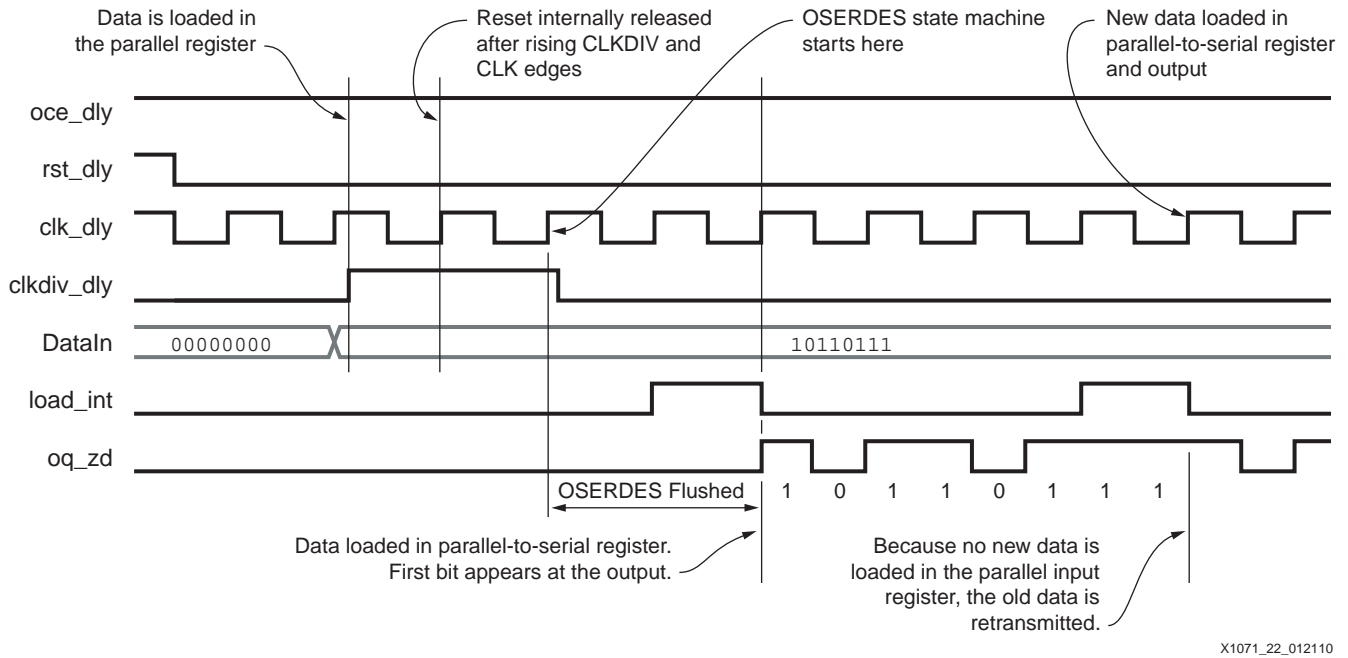
Figure 22: **OSERDESE1: First Data Out After Release of Reset**

- New data can be loaded into the OSERDESE1 at any time. However, not all loaded data appears at the output. The state machine only transfers data from the input parallel register into the output shift register, after it completes the ongoing shift operation, as shown in Figure 23. This phenomena most often occurs in a sloppy design in SDR mode.

Figure 23: **OSERDESE1: Data Flow While Continuing Operation**

- The state machine generates shift register load pulses at the CLK rate depending on the DATA_WIDTH and DATA_RATE attributes.

- Data is not erased from the input parallel register after being loaded into the parallel-to-serial output register. Therefore, when no new data is loaded into the parallel register with every subsequent load pulse, the parallel-to-serial register is loaded with the same data, as shown in Figure 22.

Data

A DAC requires parallel data. Therefore, at least one data bus must be constructed with the same width as the DAC. Each bit of the bus is represented by one (master) or two (master-slave) OSERDES components.

The input of each OSERDES is a nibble (four bits) or a byte (eight bits). Therefore, each OSERDES must load the number of bits (a nibble or byte) that represent, at the output data bus to the DAC, the number of words with the same resolution as the DAC, as shown in Figure 24.

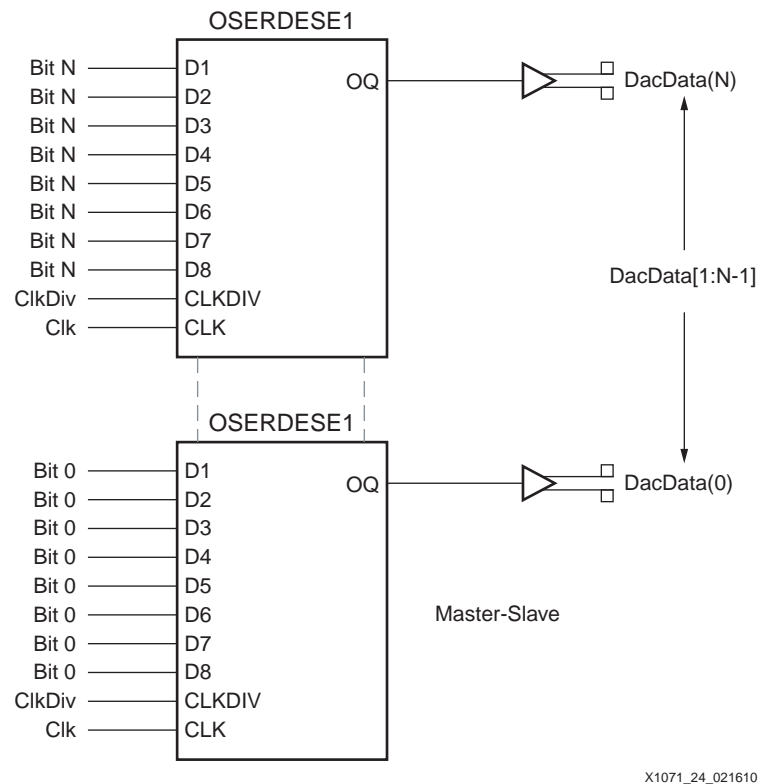*Figure 24:* **OSERDES Bit Arrangement for a DAC Data Input Bus**

Assuming that a 1.2 GSPS, 14-bit DAC connected to the FPGA provides a 600 MHz clock, the OSERDES must be used in DDR mode. An MMCM clock resource must generate a 600 MHz CLK and a 150 MHz CLKDIV, when the OSERDES is used in 8-bit mode (master-slave).

The application needs to provide 14×8 bits = 112 bits at a 150 MHz rate. This requirement is not difficult when using clock domain crossing data buffers in block or distributed memory as temporary storage. The distribution of bits output from the memory to the inputs of the OSERDES is done in the routing network of the FPGA, as shown in Figure 25 and Figure 26. In the example in Figure 25, the DAC has a 16-bit resolution and the application delivers data in a 16-bit bus format. In the example in Figure 26, the DAC has 14-bit resolution and the back-end design delivers data in a 32-bit format.

In Figure 25, eight 16-bit buses are routed and eventually registered to 16 8-bit buses. In this case, the bit routing order must be done as follows. For bit *n*, *n* is [15:0] of input bus *x*, where *x* is [7:0] →bit *m*. Value *m* is the data width input of an OSERDES [7:0] of the OSERDES for DAC bit *r*, where *r* is the resolution of DAC [15:0].

For example:

- Bit (5) of Bus_7 --> Bit (7) of OSERDES input for DAC bit (5).

- Bit (5) of Bus_6 --> Bit (6) of OSERDES input for DAC bit (5).

- Bit (5) of Bus_5 --> Bit (5) of OSERDES input for DAC bit (5).

www.BDTIC.com/XILINX

- Bit (5) of Bus_4 --> Bit (4) of OSERDES input for DAC bit (5).
- Bit (5) of Bus_3 --> Bit (3) of OSERDES input for DAC bit (5).
- Bit (5) of Bus_2 --> Bit (2) of OSERDES input for DAC bit (5).
- Bit (5) of Bus_1 --> Bit (1) of OSERDES input for DAC bit (5).
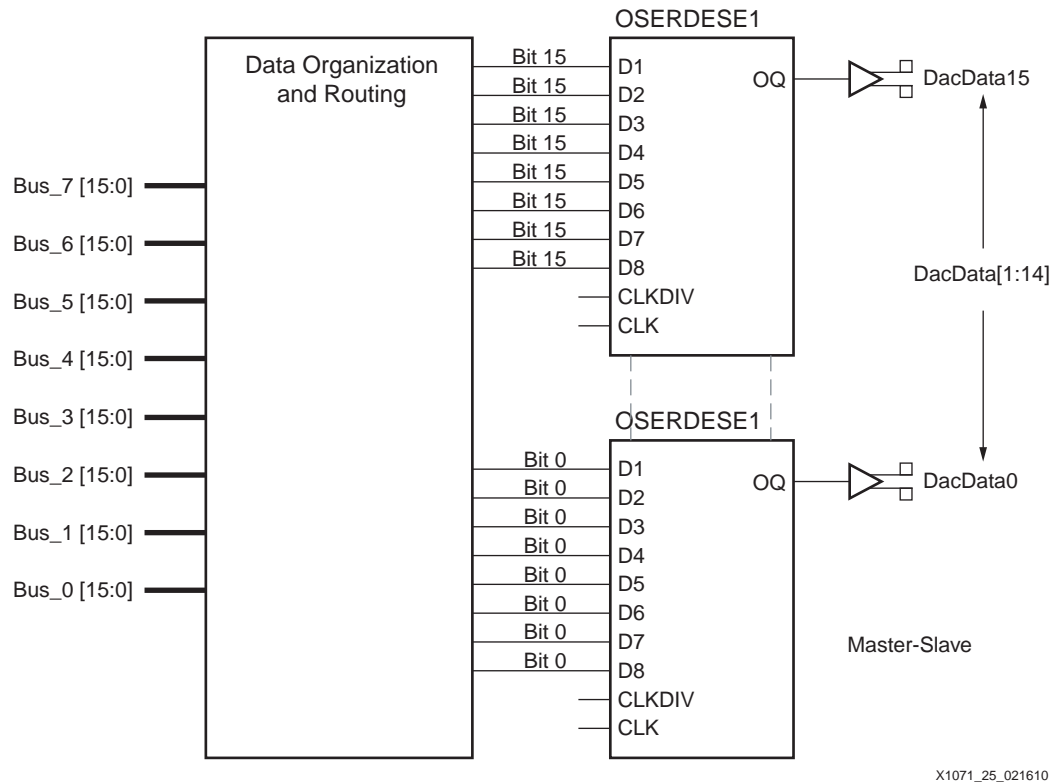- Bit (5) of Bus_0 --> Bit (0) of OSERDES input for DAC bit (5).

Figure 25: **Distribution of Data to the OSERDES Inputs (Example 1)**

Figure 26 shows a second example where the application delivers data for the DAC in 32-bit bus format. The used DAC has a resolution of 14 bits, meaning the lower 16 bits from MSB bus are used.

The connection of the application buses to the inputs of the OSERDES can be linear with the application bus order (DataBus_3[111:96] = OSERDES 13 and 12, down to DataBus_0[31:0] = OSERDES 3, 2, 1, and 0 inputs) or it can be a custom order. In all cases, no logic is needed, and the FPGA routing resources make sure that the right connections are realized.
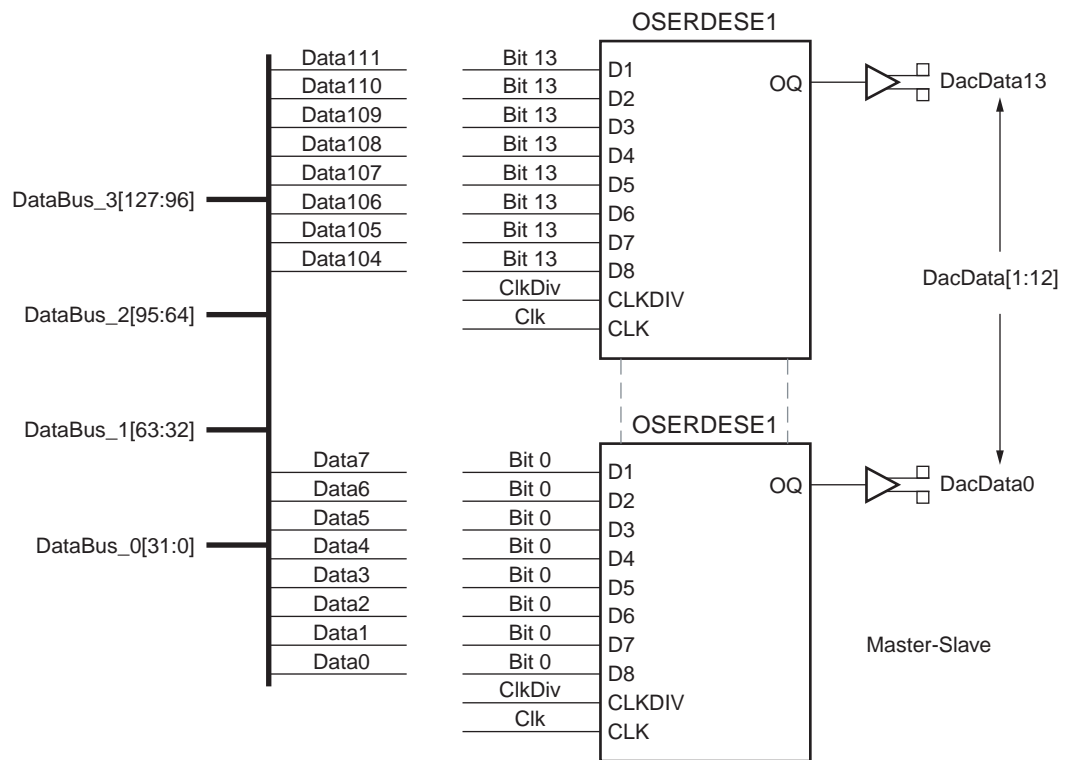
Figure 26: **Distribution of Data to the OSERDES Inputs (Example 2)**

Other connection schemes between an application and the OSERDES inputs can be used similar to these shown in Figure 25 and Figure 26. The OSERDES can also be used with other data widths and then the connection to an application can be completely different from Examples 1 and 2.
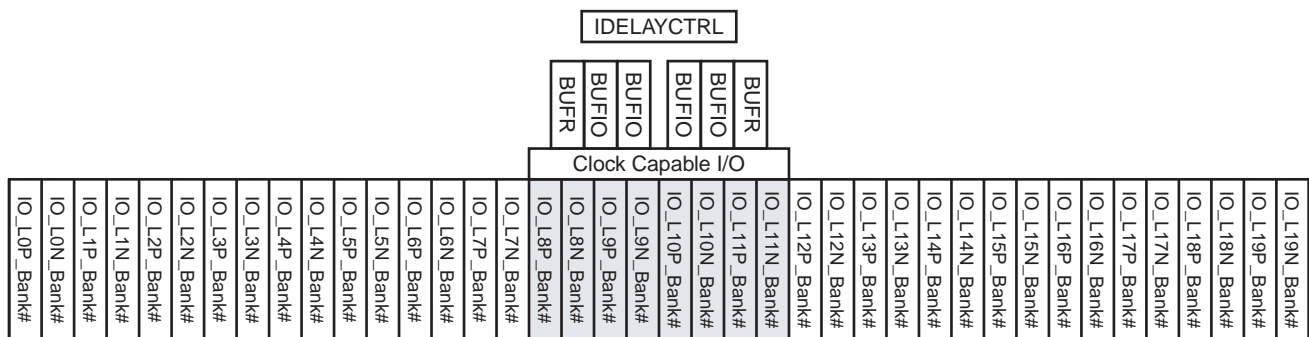
PCB Guidelines

This section discusses in brief component placement and provides short, general guidelines for PCB designers.

Component Placement

Different circuit components should be placed as close as possible to each other on the PCB, and should be aligned according to the pinout of the components. Components should be positioned to minimize the number of turns, corners, and vias. The flexibility of the FPGA pinout can assist in optimizing the PCB routing. A straight, short connection improves all possible parameters of a PCB layout, including:

• Signal integrity

• Transmission line effects

• Capacitance and inductance

• Operating frequency

Transmission line effects are important when distances between components are lengthy. All transmission lines should be terminated properly to control reflections. Virtex-6 FPGAs have I/Os arranged in banks. An I/O bank from the Virtex-6 FPGA accommodates 20 differential I/Os or 40 single-ended I/Os, as shown in Figure 27. Four differential (eight single-ended) I/Os can also be used as clock-capable I/O inputs or outputs.

Figure 27: **I/O Bank of Virtex-6 FPGAs**

Guidelines and Recommendations

This section summarizes some key guidelines for PCB designers. For an in-depth discussion on PCB design and layout, refer to UG373, *Virtex-6 FPGA PCB Design Guide*.

• Spend sufficient time when placing components for the layout.

• Keep trace lengths as short as possible.

• Spend time determining the number of PCB layers and how the layer stackup is realized.

• If possible, during PCB layout, keep the length of a track shorter than the travel and reflection times of the signal on the trace. If this is not possible, take transmission line theory into account.

• Match the lengths of all differential traces (data and clock).

• When making turns with differential traces, balance the number of left and right turns. When making a turn with a differential trace, the inner trace becomes shorter than the outer trace of the pair. When using more turns in one direction, one trace of the differential pair is longer than the other (without direct correction possibilities).

• Spread traces after routing over the available space of the PCB to minimize crosstalk.

• Do not route traces into 90° or 180° turns. Such turns increase the effective width of the trace, contributing to parasitic capacitance. At very fast edge rates, these discontinuities

www.xilinx.com

can cause significant signal integrity problems. Instead use round, circular turns. If this is not possible, use 45° corners.

- Follow the signal return path guidelines.
- Use guard traces where needed.
- Remember the importance of ground planes.
- When available, use the thermal pad connection at the bottom side of the ADC or DAC packages to improve the operating stability of the device.

Reference Design

The reference design files can be downloaded at http://www.xilinx.com/member/adc_dac_lvds_interface/index.htm (the user must first accept the License agreement). Table 2 lists the reference design matrix, which provides additional information about the design.

Table 2: **Reference Design Matrix**

Parameter	Description
General	
Developer Name	Xilinx
Target Device	Virtex-6 FPGA (XC6VLX240T and ML605 demonstration board)
Source Code provided	Yes
Source Code format	VHDL
IP used	No
Simulation	
Functional simulation performed	Yes, one for each hierarchical block
Timing simulation performed	No
Testbench format	VHDL
Simulator Software / version	ModelSim, version SE_6.5b
SPICE / IBIS simulations	No
Implementation	
Synthesis tool / version	XST version in ISE® software v11.4
Implementation tool / version	ISE software v11.4
Static Timing Analysis performed	Yes
Hardware verification	
Hardware verified	Yes
Hardware platform used for verification	ML605 demonstration board with FMC_101 loopback interface

Reference Design Utilization Summary

The provided reference design includes the logic for a quad-channel, serial LVDS ADC and a quad-channel LVDS DAC. The design is controlled by an eight-bit PicoBlaze microcontroller.

The reference design shows how ADC and DAC devices can be connected to a Virtex-6 FPGA using its I/O features (ISERDES, OSERDES, and LVDS).

The design is built in the XC6VLX240T-2-FF1156 FPGA on an ML605 board. It uses an FMC-101 cable loopback board to demonstrate the communication between the DAC and the ADC interfaces constructed in the FPGA.

The DAC interface built in the FPGA simulates a real DAC device. Some additional clock signals make it possible to communicate with the ADC interface in the same FPGA. The ADC interface is a complete interface because it is used to connect to real ADC devices. An application design is added to both interfaces, making it possible to transmit and receive data from and to a PC.

Table 3 lists the device utilization summary reported by the place-and-route tools for the reference design.

Table 3: **Device Utilization Summary**

| Slice Logic Utilization	Used	Available	Utilization (%)
Number of Slice Registers	1,083	301,440	1%
Number Used as Flip-Flops	1,083	-	-
Number of Slice LUTs	1,406	150,720	1%
Number Used as Logic	977	150,720	1%
Number Used as Memory	323	58,400	1%
Number of Occupied Slices	710	37,680	1%
Number of LUT Flip-Flop Pairs Used	1,575	-	-
Number of Bonded IOBs	106	600	17%
Number of RAMB36E1/FIFO36E1s	6	416	1%
Number of BUFG/BUFGCTRLs	8	32	25%
Number of ILOGICE1/ISERDESE1s	24	720	3%
Number of OLOGICE1/OSERDESE1s	25	720	3%
Number of BUFIODQSs	2	72	2%
Number of BUFRs	4	36	11%
Number of IDELAYCTRLs	2	18	11%
Number of IODELAYE1s	14	720	1%
Number of MMCM_ADVs	2	12	16%

Reference Design Directory Setup

The design is generated using a new Eclipse-based editor and project manager for HDL designs. The project and files can also be opened and edited using a normal text editor.

The basic Eclipse tool (Galileo, version 3.5) is installed with the SimplifIDE editor and project management tool. SimplfIDE HDL (VHDL/Verilog) is a plug-in that can be installed on top of Eclipse.

Because the design hierarchy is set up in the HDL management tool, the design follows some specific directory settings. These are extended versions of the rules normally used by the designer of the reference design.

Figure 28 shows the top-level hardware suite and directory setup.
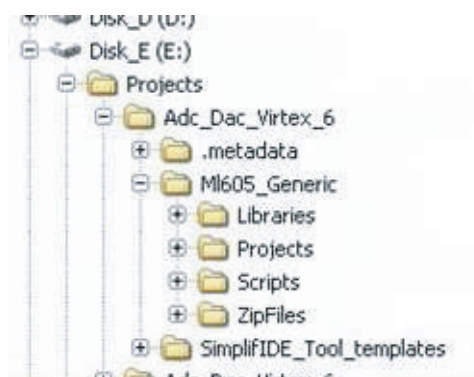
X1071_28_011710

Figure 28: **Top-Level Hardware Suite and Project Directory Structure**

The `.metadata` directory, created by Eclipse, contains all Eclipse workspace settings. The `Ml605_Generic` directory is the created hardware suite containing all subdirectories for possible multiple projects. The `Libraries` subdirectory contains libraries spanning multiple projects. The `Projects` subdirectory contains any different projects. The `Scripts` subdirectory contains scripting files for the SimplifIDE project management and editor. The `SimplifIDE_Tool_templates` directory contains templates on how to set up the directories in projects and libraries.

To use Eclipse and SimplifIDE, the "Hardware Profile" of the Eclipse project should be imported by pointing to the `.metadata` directory.

Figure 29 shows the directory structure of the project. In this case, `Adc_Dac` is a generic ADC and DAC interface project with back-end features, such as:

• Small, LUT-based clock domain crossing data buffers.

• Large, block RAM-based clock domain crossing data buffers.

• An eight-bit PicoBlaze microcontroller design featuring SPI and I^{2}C interfaces with a USB (RS-232) connection to a PC.

Top-Level Design File
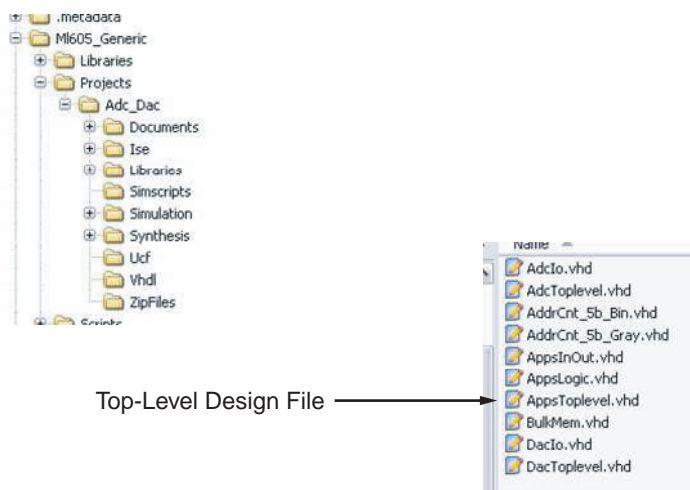
X1071_29_012110

Figure 29: **Directory Setup Structure of the Reference Design**

Table 4 lists all subdirectories in the project.

Table 4: **Project Subdirectories**

Subdirectory	Description
Documents	Documentation on the global ADC/DAC project and the ML605 board.
Ise	Xilinx implementation of the top-level design.
Libraries	The different elements of the design (hierarchy).
SimScript	Simulation scripts for the top level of the design.
Simulation	Used by the simulator when simulating (compiling) the entire design.
Synthesis	Used by the synthesis tools. One subdirectory is created per used synthesis tool.
Ucf	User Constraints File directory for fitting the design to the ML605 board.
Vhdl	Top-level HDL files of the design.
ZipFiles	Zipped versions of projects.

The entire design is done using flexible design blocks, represented in hierarchical design blocks. Each block is designed as a standalone block into its own library.

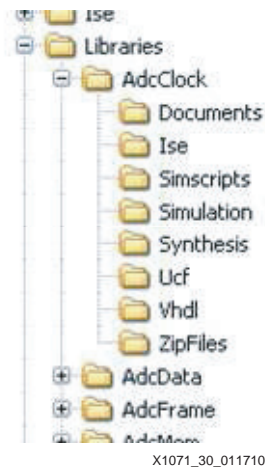Figure 30 shows the setup of the libraries in the project.

X1071_30_011710

Figure 30: **Design Library Setup Structure**

Table 5 defines the subdirectories for the libraries in the project. All libraries are set up the same way.

Table 5: **Library Subdirectories**

Subdirectory	Description
Documents	Documentation about the functioning of the HDL in the library.
Ise	Xilinx implementation of the top-level design.
Libraries	The different elements of the library design.
SimScript	Simulation scripts for the top level of the design.
Simulation	Used by the simulator when simulating the entire design.
Synthesis	Used by the synthesis tools. One subdirectory is created per used synthesis tool.
Ucf	User Constraints File subdirectory.

Table 5: **Library Subdirectories** *(Cont'd)*

Subdirectory	Description
`Vhdl`	HDL design files of the design.
`ZipFiles`	Zipped files for project backups.

Additional information about the directory setup and all design documentation is contained in PDF files in the `/Documents` directories in the project.

References and Resources

This section provides supplemental information useful to this application note:

- DS152, *Virtex-6 FPGA Data Sheet: DC and Switching Characteristics*
- UG361, *Virtex-6 FPGA SelectIO Resources User Guide*
- UG362, *Virtex-6 FPGA Clocking Resources User Guide*
- UG363, *Virtex-6 FPGA Memory Resources User Guide*
- UG365, *Virtex-6 FPGA Packaging and Pinout Specifications*
- Eclipse: http://www.eclipse.org/downloads/
- SimplifIDE: http://www.eclipseplugincentral.com/ (search for SimplifIDE)

Conclusion

Virtex-6 FPGA interfaces use the ISERDES and OSERDES features to provide a flexible and versatile platform for building high-speed LVDS signaling interfaces to all the latest ADC and DAC devices on the market.

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
06/23/10	1.0	Initial Xilinx release.

Notice of Disclaimer

www.BDTIC.com/XILINX