☒ XILINX®

Source-Synchronous Serialization and Deserialization (up to 1050 Mb/s)
Author: NIck Sawyer

XAPP1064 (v1.1) June 3, 2010

Summary

Spartan®-6 devices contain input SerDes (ISERDES) and output SerDes (OSERDES) blocks. These primitives simplify the design of serializing and deserializing circuits, while allowing higher operational speeds. This application note discusses how to efficiently use these primitives in conjunction with the input delay blocks and phase detector circuitry.

ISERDES and OSERDES Guidelines

Each Spartan-6 FPGA input/output block (IOB) contains a 4-bit input SerDes and a 4-bit output SerDes. The SerDes from two adjacent blocks (master and slave) can be cascaded to make an 8-bit block. This gives the possibility of SerDes ratios from 2:1 to 8:1 on both output and input for both single and double data rate I/O clocks.

Cascading the ISERDES blocks is not an issue when a differential signaling standard is being used because these standards use the two IOBs (master and slave) associated with the two sets of SerDes registers. Thus, using two ISERDES effectively reduces design cost. However, when using a single-ended signaling standard, some care needs to be taken when the design requires either a SerDes ratio of five or more or the phase detector mode. Specifically, two data lines cannot enter the device in adjacent master and slave IOBs when using cascaded SerDes. This limitation is not necessary when the SerDes ratio is four or less and the phase detector mode is not being used because the SerDes is not cascaded. However, by not using the phase detector mode, data loss will occur during calibration and the application will need to account for this loss.

Introduction to Deserialization and Data Reception

A deserializer design and its associated clocking primitives are dependent on the format of the incoming receive data stream. This data tends to fall into three categories.

Case 1

The data stream is a multiple of the rate of the incoming clock, and the clock signal is used as a framing signal for the received data. Multiple changes in the state of the data lines occur during one clock period. A widely used example is the 7:1 interface used in cameras and flat panel TVs and monitors. Other ratios are obviously possible, and the Spartan-6 FPGA ISERDES can support ratios of 2, 3, and 4:1, and also 5, 6, 7, and 8:1 when cascaded. In this example, the received clock is multiplied in a PLL, and the resultant high-speed capture clock is passed to the input logic through the BUFPLL primitive. The BUFPLL capture clock is designed to always be used in single data rate (SDR) mode with respect to the input data. For example, a 150 MHz input clock with accompanying 7:1 data requires the PLL and BUFPLL to operate at 1050 MHz (equals 150 x 7). This high-speed capture clock is used to clock the receive data into the input deserializers and is capable of driving one whole edge of a device. Parallel data is then presented to the FPGA logic at the speed of the original incoming clock. Figure 1 shows this 7:1 data formatting example.
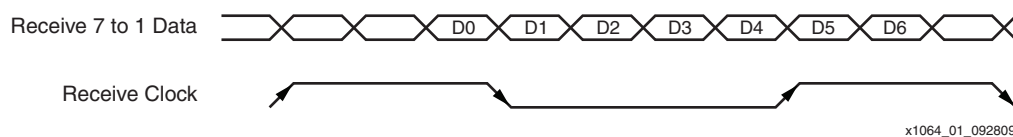
*Figure 1:* **Data Stream Using a Low-Speed Clock with a 7:1 SerDes Ratio**

Case 2

The data stream is a multiply by two of the incoming clock, commonly called Double Data Rate (DDR) reception. A DDR data stream is shown in Figure 2. Each transition of the clock indicates a change in the state of a data line. There are two ways of receiving this kind of data. The first is to use a PLL and a BUFPLL (see Case 1), where the PLL is being used to multiply the incoming clock by two and the BUFPLL allows use of the whole edge of a device. The other method is to use the BUFIO2 primitive, where two BUFIO2s are required to receive DDR data, and the BUFIO2s are only able to drive the same half edge of a device where the clock input is located. The deserialization factor (ratio) can be chosen by the designer (values of 2, 4, 6, and 8:1). The necessary divided clock for the parallel data is generated through one of the BUFIO2 primitives. Two BUFIO2s must be used to multiply the incoming DDR clock by two to generate an SDR capture clock.
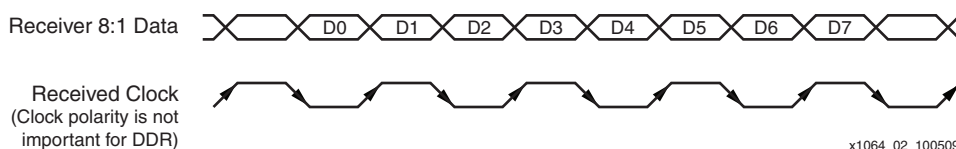
Figure 2: **8:1 Data Stream Using DDR**

Case 3

The data stream is at the same rate as the receiver clock (SDR). A drawing of an SDR data stream is shown in Figure 3. Each data bit changes every two clock transitions, normally on the rising edge of the clock. There are two ways of receiving this kind of data stream. The received clock is multiplied by one in a PLL and the BUFPLL is used to receive data on a whole edge of a device, or a single BUFIO2 or PLL is used to drive the inputs in the half edge of the device where the clock input is situated. The BUFIO2 is also used to divide down the received clock to be used with the deserialized parallel data. Using SDR, ratios of 2, 3, 4, 5, 6, 7, and 8:1 are possible.
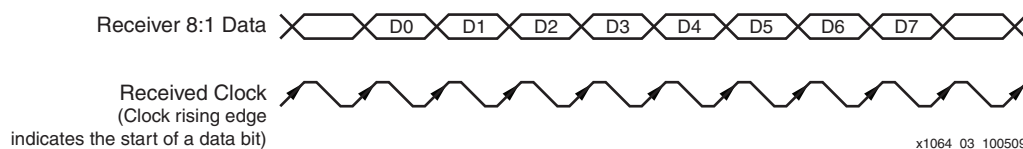
Figure 3: **8:1 Data Stream Using SDR**

Higher Deserialization Factors

Deserialization using factors greater than 8:1 is possible when receiving data. The PLL can be used to generate a third clock, which is intermediate to the high-speed I/O capture clock and the low-speed parallel data clock. Examples of designs using SerDes ratios of 10, 12, 14, and 16:1 are included in the Reference Design Files. Essentially, the input SerDes primitives are still used in 5, 6, 7, and 8:1 modes, receiving data through a high-speed capture clock from the PLL through the BUFPLL. The received parallel data is transferred to the FPGA logic in the intermediate clock domain and then further transferred to the main clock domain using a 2:1 gearbox, also in the FPGA logic. A drawing of the mechanism is shown in Figure 4. In any of these examples of higher deserialization factors where the PLL is used, the receiver clock can be SDR, DDR, or a divided clock.
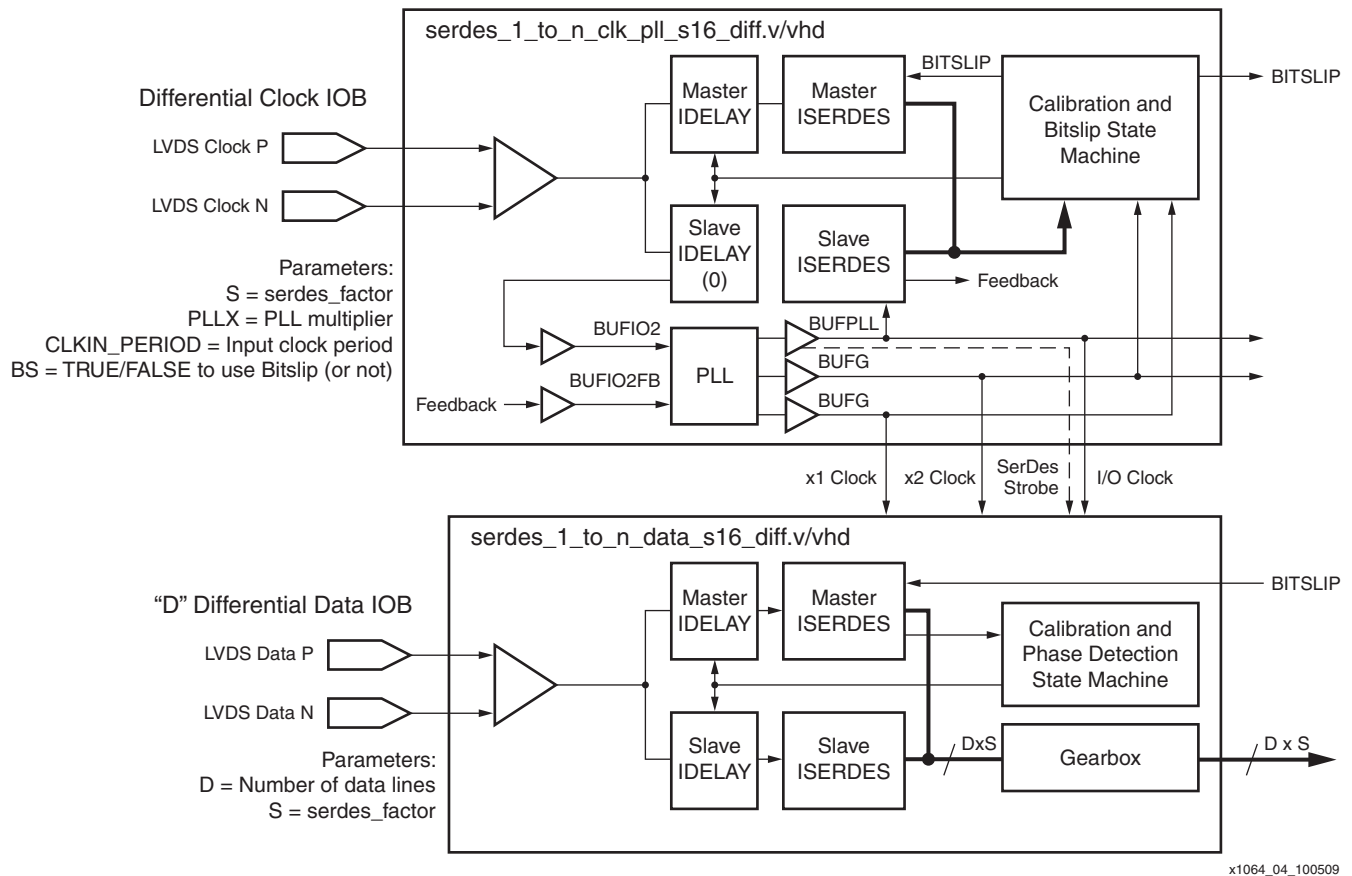

Figure 4: **Receiving Data at Higher SerDes Factors**

Data Reception Using PLL and BUFPLL

The topology for data reception using PLL and BUFPLL is uncomplicated. The receiver clock is multiplied as required in the PLL to generate an internal single data-rate capture clock. The incoming clock signal needs to pass through an input delay block (to balance datapath delays) and a BUFIO2 to reach the PLL. In the 7:1 video example, the received pixel clock must be multiplied by seven. The clock signal is routed from the PLL to a BUFPLL primitive to drive one whole edge of the device. CLKOUT0 and CLKOUT1 are the only outputs of the PLL that are capable of driving high-speed capture clocks to the BUFPLL. The BUFPLL also requires a global clock signal equal to the original non-multiplied source clock, which can be driven from any of the PLL outputs through a global buffer (BUFG), and the LOCKED signal from the PLL, which is required for synchronization inside the BUFPLL.

The three input signals to the BUFPLL allow the BUFPLL to distribute the high-speed receiver clock to the input delay and SerDes primitives in the same edge of the device, along with the required SerDes strobe signal (appropriately aligned) that allows safe transfer of low-speed parallel data to the FPGA logic from the input SerDes.

When using the PLL for data reception, PLL deskew is required. The feedback clock signal is routed from an I/O clock destination at the input SerDes primitive of the clock input pin back to the PLL using a BUFIO2FB primitive. This mechanism forces the multiplied clock to be in the same phase as the original received clock.

The mechanism for centering and capturing data reliably is based on the IODELAY2 primitive, used in input delay mode only. This mechanism is discussed in the Delaying Input Data and Clocks section. The block diagram of the receiver is shown in Figure 5.
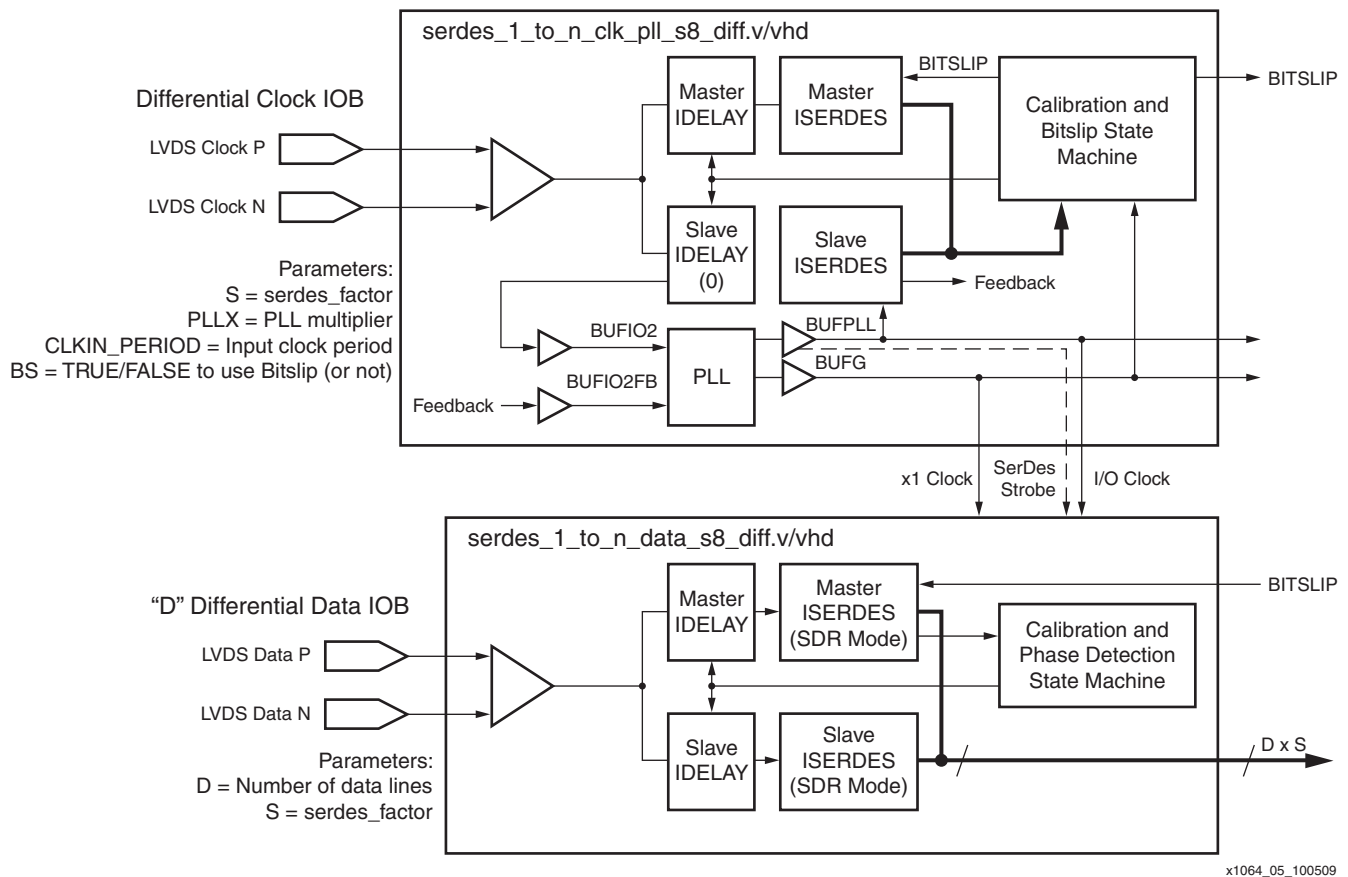

Figure 5: **Data Reception Using PLL and BUFPLL**

DDR Data Reception Using Two BUFIO2s

The topology for DDR data reception using two BUFIO2s uses the incoming clock to directly capture data without the use of a PLL. The incoming clock signal is fed through a delay block to balance the data and clock delays. In the case of a differential signal, as shown in Figure 6, the true and complement signals are fed through master and slave input delays (both set to zero) and then to a pair of BUFIO2 primitives. The first BUFIO2 accepts both true and complement input clocks and uses these to generate the appropriate divided clock and SerDes strobe for the input SerDes primitives. For example, if the receiver clock is 311 MHz (622 Mb/s data) and the design requires an 8:1 SerDes reduction, the BUFIO2 being driven by true and complement receiver clocks with its divide parameter set to eight actually divides the input clock by 4 to 77.75 MHz. The resultant I/O clock is routed to the input SerDes primitives along with the other

(inverted) I/O clock generated from the other BUFIO2. These two clocks are doubled in the input SerDes to give a 622 MHz sampling clock for the 622 Mb/s data.

The BUFIO2s need to be located in the same half edge as the clock input, and when using input delays, it is not possible to simultaneously use the alternate BUFIO2s in the other half edge. Reception of data buses when using input delays is therefore limited to the half edge where the clock input is located.

The mechanism for centering and capturing data reliably is based on the IODELAY2 primitive, used in input delay mode only. This mechanism is discussed in the Delaying Input Data and Clocks section. The block diagram of the receiver is shown in Figure 6.
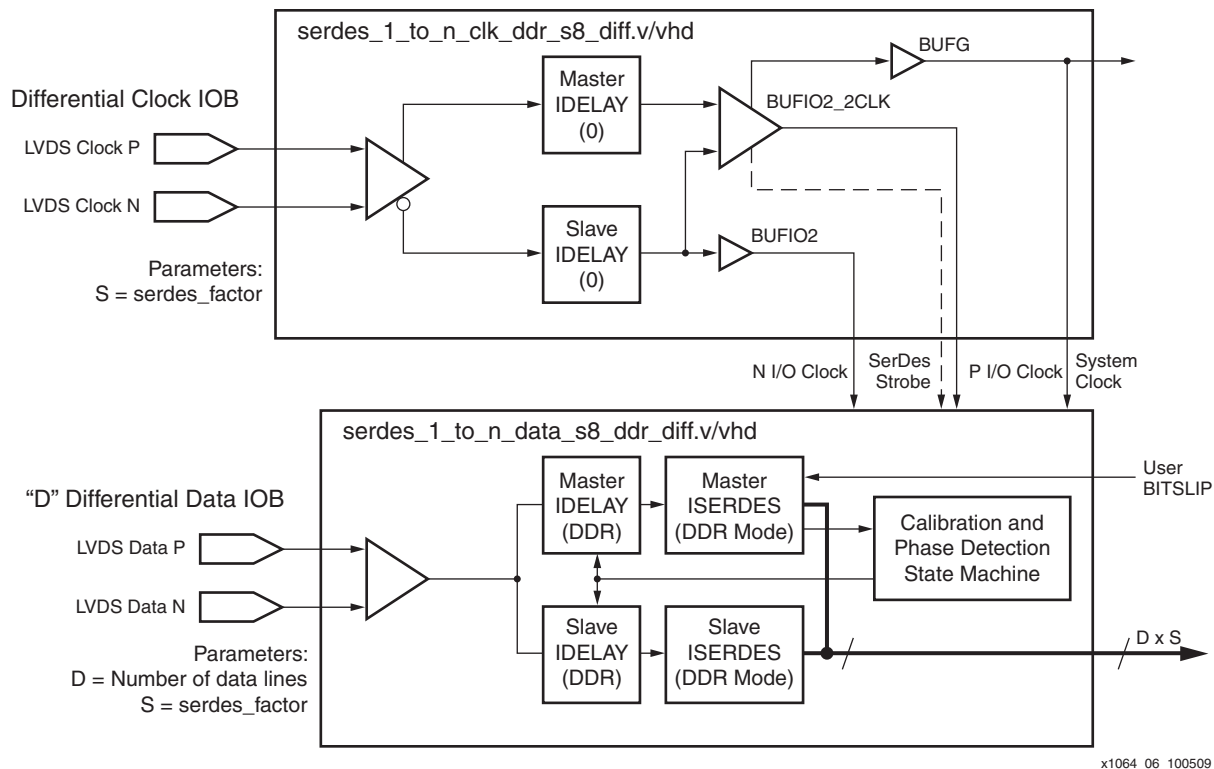

Figure 6: **DDR Data Reception Using Two BUFIO2s**

SDR Data Reception Using BUFIO2

The topology for SDR data reception using BUFIO2 uses the incoming clock directly to capture data. The clock signal is fed through a delay block (set to 0) to a BUFIO2. The BUFIO2 uses this input clock to generate the appropriate divided clock and SerDes strobe for the input SerDes primitives. For example, if the receiver clock is 525 MHz (525 Mb/s data) and the design requires an 8:1 SerDes reduction, then the BUFIO2 with its divide parameter set to eight divides the input clock by 8 to 65.625 MHz. The resultant I/O clock is routed to the input SerDes primitives for data capture. The limiting factor in this case is the maximum clock frequency allowed through the clock input pin.

The BUFIO2 needs to be located in the same half edge as the clock input. When using input delays, it is not possible to simultaneously use the alternate BUFIO2 in the other half edge. Reception of data buses is therefore limited to the half edge, the location of the clock input when using input delays.

The mechanism for centering and capturing data reliably is based on the IODELAY2 primitive, used in input delay mode only. This mechanism is described in the Delaying Input Data and Clocks section. The block diagram of the receiver is shown in Figure 7.
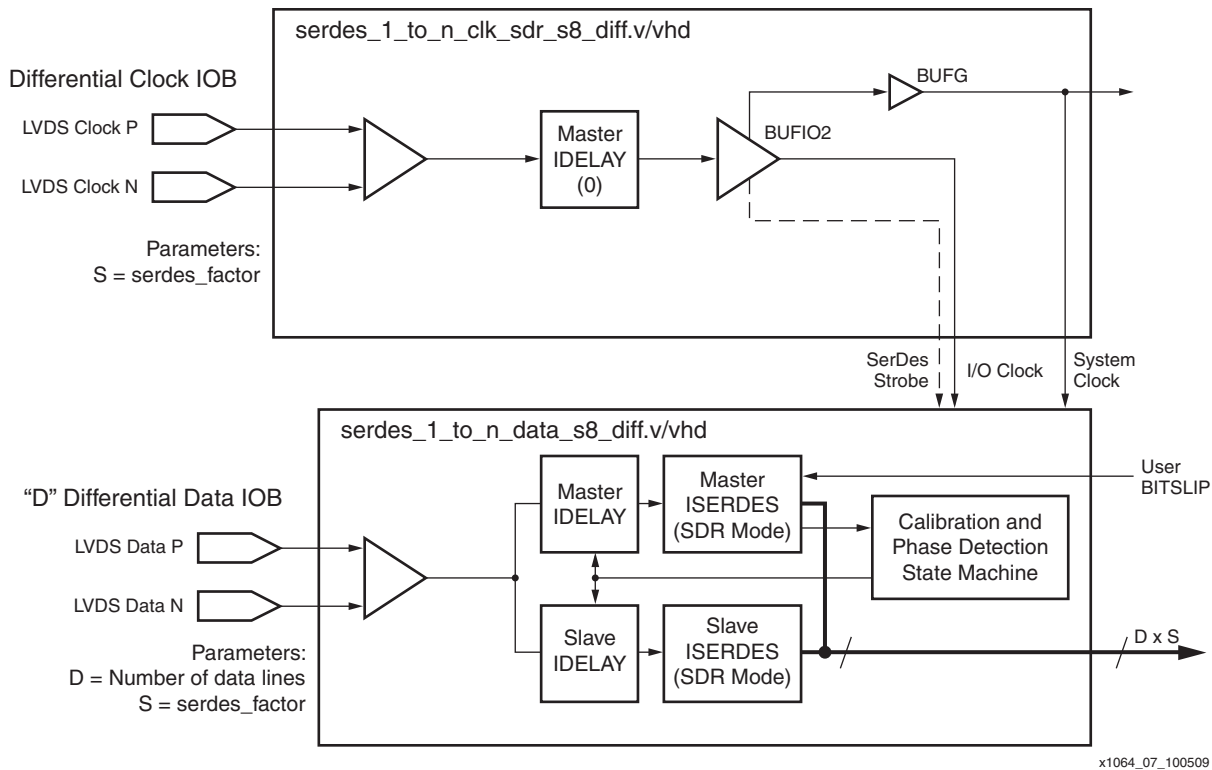
Figure 7: **SDR Data Reception Using BUFIO2**

Delaying Input Data and Clocks

The Spartan-6 FPGA data capture mechanism is based on the input delay primitives (IODELAY2). The individual delay taps are not constant over PVT and therefore require regular calibration. Mechanisms to perform the regular calibration that allow continuous data capture are described in this section and shown in Figure 8. In this sort of data capture, it is important that the capture clock and input data delays are closely matched. To achieve this, the input clock must be routed through an input delay that is set to zero before being routed to either a BUFIO2 or a PLL through a BUFIO2. With this method, the insertion delay in the clock and data paths are equal, and the data delays can be varied to ensure data capture occurs in the middle of the data eye.

Assuming that the received clock and data arrive edge aligned, the data delay needs to be set to precisely half of the capture clock period so that the data is sampled in the middle of the eye. To achieve this, the built-in calibration function of the input delay primitive must be used. When a calibration command (CAL) is issued to the input delay (by asserting CAL High for one low-speed clock cycle), the input delay internally determines how many delay tap elements are required to delay the data by half a bit period, and then sets the delay line to be equal to half of this value. The high-speed capture clock itself is used as the frequency reference. For example, the calibration circuit determines the incoming capture clock, and therefore the incoming bit period is equal to 24 delay taps at the current voltage and temperature. Setting the number of delay taps equal to 12 delays the data by one half of a bit period, allowing successful data capture. Some time later, the next calibration finds that the values have changed to 26 and 13 respectively, and the input delay values are then updated automatically.

Received data is lost while the calibration process is occurring in the input delay. In some protocols, this is not a problem, but where the data is continuous, this causes issues. A further or phase-detector mode is included within the input delays to allow calibration to occur without data loss.

In phase-detector mode, an input data signal is supplied to two input delay primitives in parallel. These primitives are referred to as master and slave. When in this mode, the slave input delay is configured to control the behavior of both itself and the master input delay. Received data is taken from the master delay, and the output of the slave delay is usually ignored unless the deskew feature of the phase detector is required. The deskew feature is described further in Phase Detector and Board Deskew section.

Using the previous example, a calibration command is issued to the slave input delay, the block calculates the number of taps equal to one capture clock period, for instance 24, and then sets the master input delay to half of this value, in this instance, 12. The update of the master input delay value is synchronized to the input data stream, so no data loss occurs either during calibration or when the delay value is updated.

In DDR mode, the capture clock that is measured and used for calibration is actually the two I/O clock signals combined together. For example, an incoming 311 MHz clock used to capture DDR data is doubled in the input SerDes and in the input delay primitives, thus producing a 622 MHz clock, which is therefore equal to the incoming bit rate.
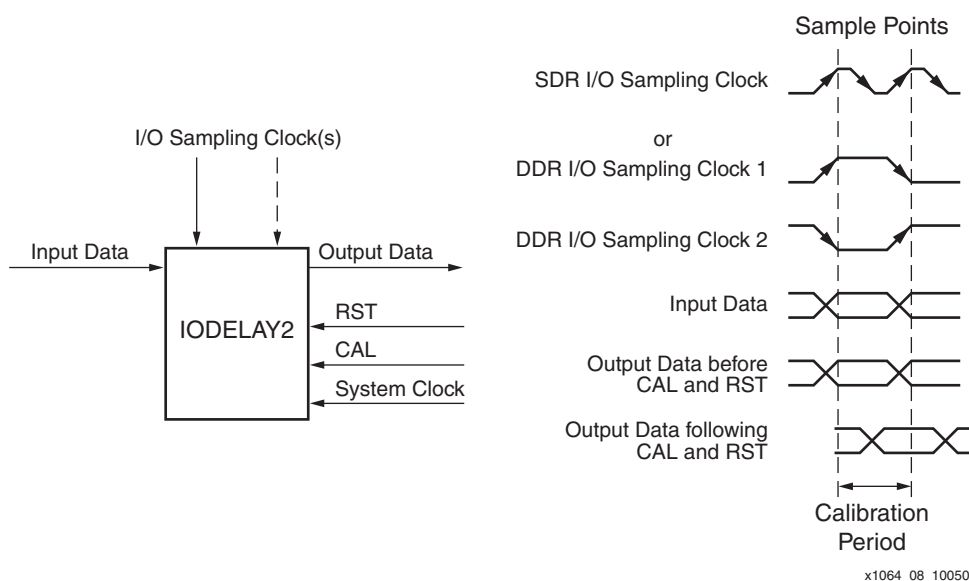

x1064_08_100509

Figure 8: **Input Delay Primitives**

Reset (RST) and CAL originate from a designer's state machine. Asserting CAL High for one clock period causes a calibration to occur, this determines how many delay taps at the current PVT are equal to one bit period. Asserting RST High for one clock period causes half the resulting value to be loaded into the delay line. In phase-detector mode, performing a CAL function does not affect data integrity. RST only needs to be asserted High following the initial CAL function, further CAL functions do not require an RST.

Phase Detector and Board Deskew

The Spartan-6 FPGA phase detector has dual definitions. Phase-detector mode is the mode where a slave input delay effectively controls a master input delay during calibration, allowing the master delay to pass data through without modification (apart from delay) and is used to avoid data loss.

The phase detector generally refers to the possibility of using dedicated logic inside the input SerDes primitives to allow reception of data that is skewed for some reason from its associated input clock. The function of the phase-detector logic, which requires control from an external state machine, is to adjust the input delay appropriately to ensure that the receiver sampling clock is in the center of the received data eye. This allows maximum performance of the sampling circuitry and higher error-free data reception bit rates.

The effects of a phase-controller operation are shown in Figure 9. The I/O sampling clock is always SDR–whether coming from a BUFPLL, a BUFIO2, or when doubled by using two BUFIO2s. The initial offset of the input data is shown in the first pair of traces. Data coming through the master input delay, which is calibrated, is delayed by half a clock period, or as close to half a clock period as a discrete delay line can achieve. The data coming through the slave is initially delayed by zero, which is effectively the same as delaying it by one bit period as long as the results, VALID and INCDEC, are pipelined accordingly. For clarity, the data pipeline stage through the slave is not shown in the timing diagram.

As shown in the first pair of traces in Figure 9, the sampling point is not in the middle of the data eye. As well, the sample taken from the master delay (inside the master input SerDes) is the same as the one taken from the slave delay (inside the slave input SerDes). Following a change in state of the input data, the phase detector determines that both signals tested are the same value and indicates this by using a pair of output pins from the master input SerDes called VALID and INCDEC. VALID is asserted High whenever a valid transition is detected. INCDEC then indicates the direction to adjust the delays to move the sample clock closer to the center of the data eye. The designer's state machine acts on this data and commands the input delay primitives to increment or decrement appropriately, using the pins CE and INCDEC on the slave input delay. CE is asserted High for one system clock period, with INCDEC set appropriately for the required direction to adjust the delay.

The second pair of traces in Figure 9 shows the result of a decrement command issued to the slave input SerDes. The delay is reduced by one tap. The master delay becomes ½ MAX – 1 (where MAX is the value found by calibration described in Delaying Input Data and Clocks, and MAX is not the maximum possible delay of 256 taps) and the slave input delay is now MAX – 1. The sampling circuitry finds that the two samples (master and slave) are the same and issues another decrement command to the slave input delay.

The result following this decrement is shown in the third set of traces in Figure 9, and since the two samples are still the same, another decrement command is issued to the slave input delay.

The fourth pair of traces in Figure 9 shows the master delay at ½ MAX – 3, the slave delay at MAX – 3, and that the two samples taken by the input SerDes primitives are now different. This indicates to the controlling state machine that an increment command to the slave input delay must be issued. This command moves the result back to the state shown by the third pair of traces in Figure 9.

The circuit moves around these two points because they correspond to the ideal situation where the sampling clock is in the middle of the data eye. In actual silicon, there is jitter on the incoming clock and data, but the principle of operation remains the same, and the sample point is maintained near the middle of the eye over time. All the design examples include a state machine based on a 32-bit filter to control the delays appropriately in the presence of jitter.

Periodically, the input delays need to be recalibrated to accommodate any changes in delay that occur over voltage and temperature. The calibration command does not affect the reception of data but does affect the current position of the sample point found by the phase detector. The value of MAX changes with recalibration, but the phase detector continues its operation transparently with the new value of MAX and, therefore, ½ MAX is loaded into both delay primitives.
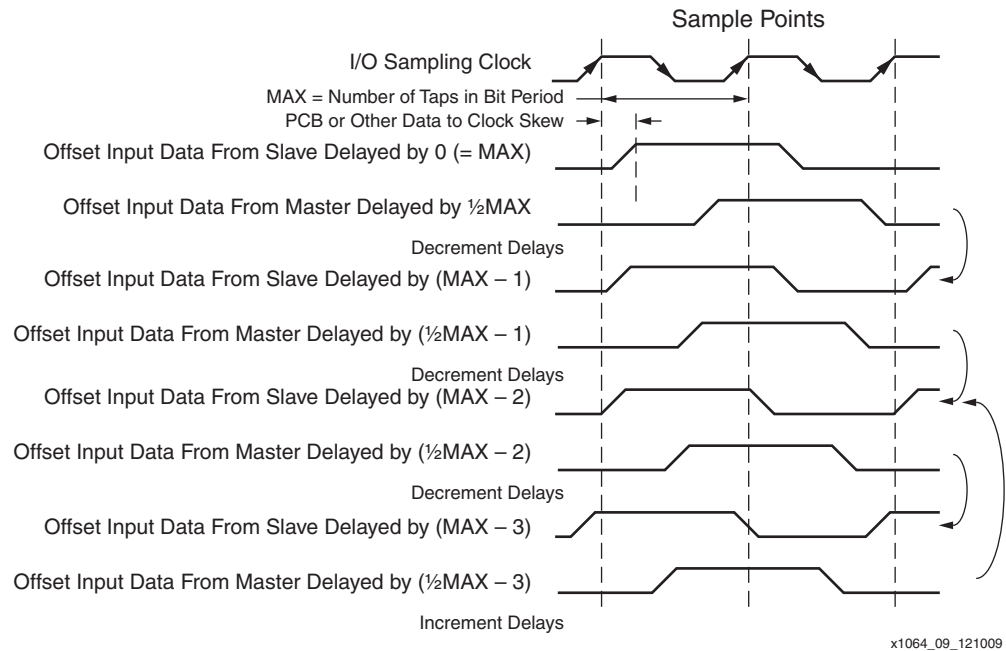
*Figure 9:* **Input Data Deskew Using the Phase Detector**

Introduction to Serialization and Data Transmission

As with data reception, the design of the serializer and its associated clocking primitives depends on the desired format of the transmitted data stream and forwarded clock. The design depends upon whether the required output forwarded clock and data stream change state at the same time and can be generated from the same transmit clock, or if the required output forwarded clock is SDR and therefore changes state twice for each data bit transition.

Certain output standards, such as LVDS, are only available on the top and bottom edge of Spartan-6 devices.

Case 4

The required output forwarded clock and data stream change state at the same time and can be generated from the same transmit clock. Figure 10 shows a widely used example of this data stream. The forwarded clock is the 7:1 interface used in cameras and flat panel TVs and monitors. Other ratios are obviously possible, and the output SerDes supports ratios of 2, 3, 4:1 and also when cascaded 5, 6, 7, 8:1. This case also includes the forwarding of a DDR clock, which is shown in Figure 11, with a 6:1 SerDes ratio. The SDR internal transmitter clock can be generated either through a PLL and BUFPLL, a single BUFIO2, or two BUFIO2s, depending on the frequency source for the internal transmitter clock.

x1064_10_121409
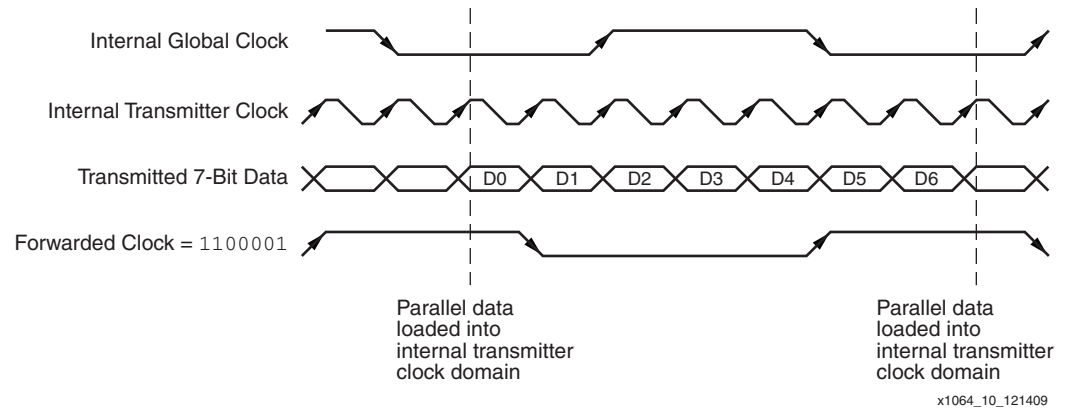
Figure 10: **Output Data Stream Using a Forwarded Low-Speed Clock with a 7:1 SerDes Ratio**


x1064_11_121409
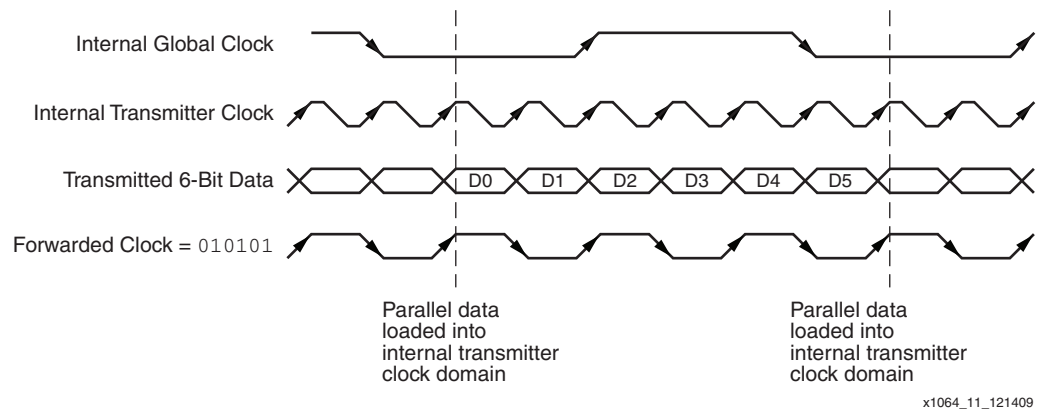
Figure 11: **Output Data Stream Using a DDR Forwarded Clock**

Case 5

The required output forwarded clock is SDR, where it changes state twice for each data bit transition. There are two methods to solve this case. The first method, shown in Figure 12, uses a single internal transmitter clock to generate an SDR forwarded clock by sending the pattern `0101` etc., and transmit each data bit twice at the same rate. The data only appears to change once for every two clock transitions. This method can use a PLL plus BUFPLL, a BUFIO2, or two BUFIO2s, depending on the frequency source for the internal transmitter clock. The disadvantage of this method is that the effective output SerDes ratio is a maximum of four rather than eight when using cascaded SerDes. The advantage is that only one BUFPLL is used. The second method, shown in Figure 13, requires two transmitter clocks to be generated through a PLL and two BUFPLLs. One transmitter clock is used to generate an SDR forwarded clock, and the other (which is half the speed of the first) is used to generate the forwarded data. The advantage of this method is that the full cascaded output SerDes ratio of eight is available, but at the cost of using both BUFPLLs on a given edge of the device and an extra global buffer.
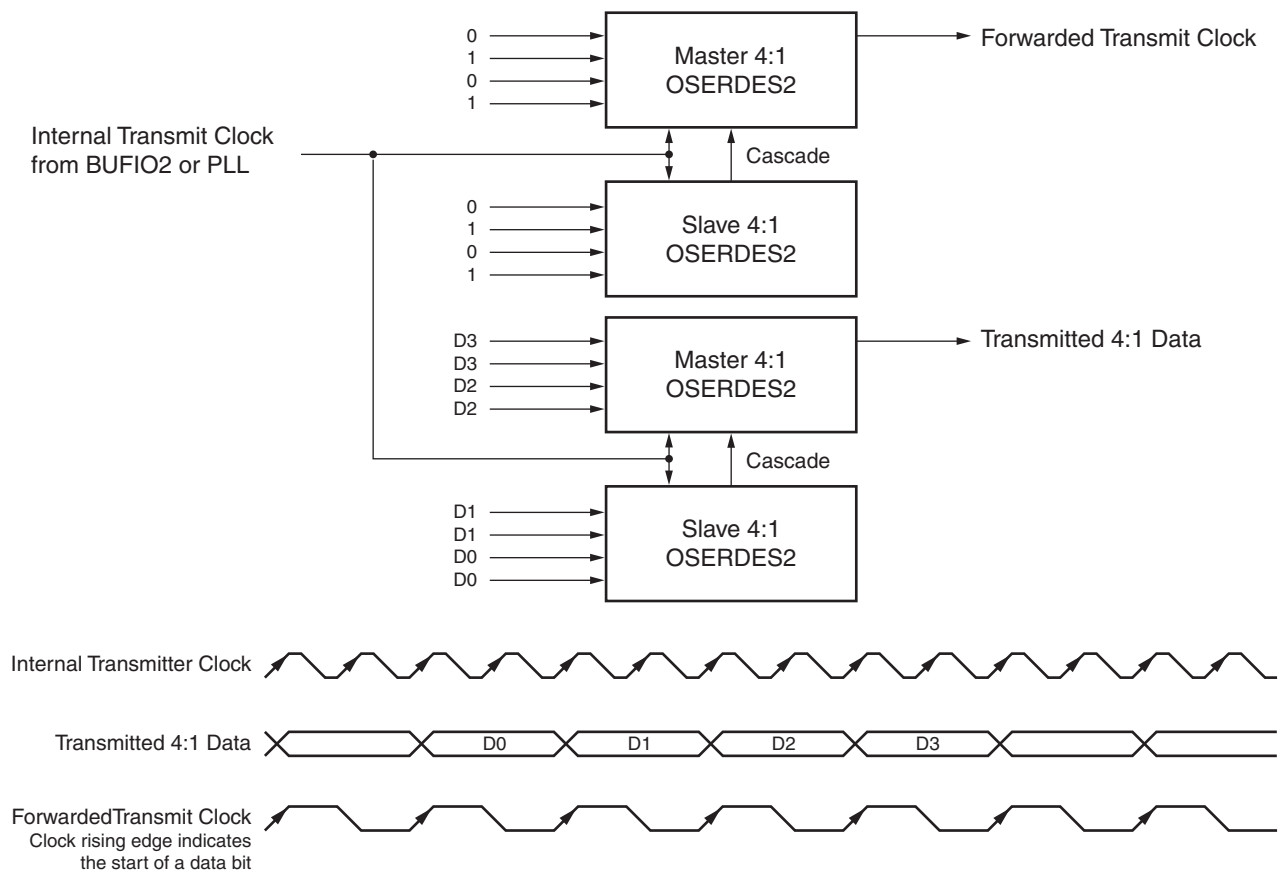
Figure 12: **Output Data Stream Using an SDR Forwarded Clock with One Transmitter Clock**
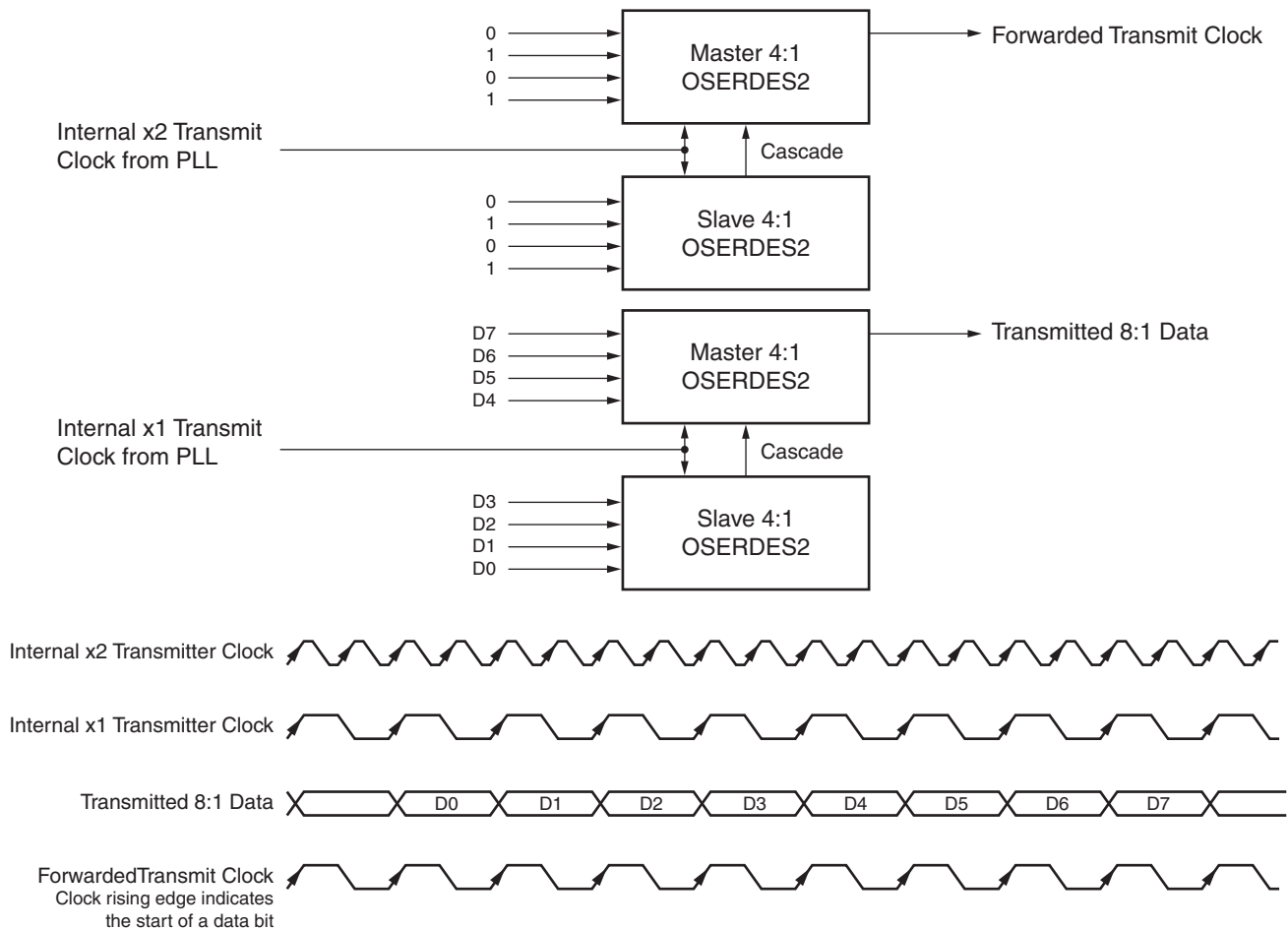
Figure 13: **Output Data Stream Using an SDR Forwarded Clock with Two Transmitter Clocks**

Higher Serialization Factors

Serialization using factors greater than 8:1 is possible when transmitting data, by using the PLL to generate a third clock, which is intermediate to the high-speed transmit clock and the low-speed parallel data clock. Examples of designs using SerDes ratios of 10, 12, 14, and 16:1 are included in the Reference Design Files. Essentially, the output SerDes primitives are still used in 5, 6, 7, and 8:1 modes, transmitting data through a high-speed clock from the PLL and the BUFPLL. The parallel data for transmission is transferred from the FPGA logic to the output SerDes in the intermediate clock domain, having been transferred from the main clock domain to the intermediate clock domain using a 2:1 gearbox also in FPGA logic. A drawing of the mechanism is shown in Figure 14. The external transmitter clock can be SDR, DDR, or a divided clock in any case where the PLL is used.
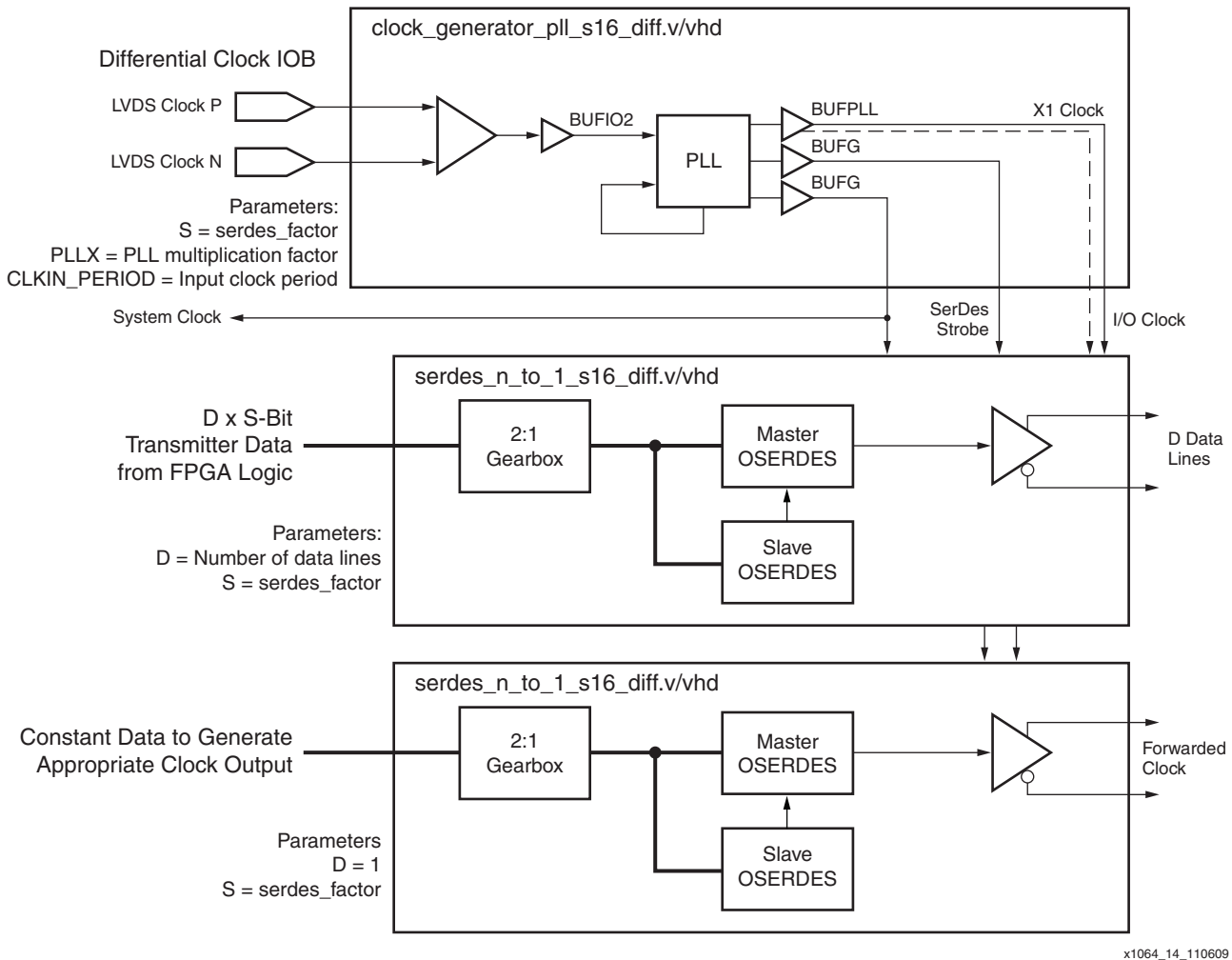
Figure 14: **Transmitting Data at Higher SerDes Factors**

Data Transmission Using PLL and BUFPLL

The topology for data transmission using PLL and BUFPLL is uncomplicated. The transmitter source clock is multiplied as required in the PLL to generate an internal SDR transmitter clock. In the 7:1 video example, the internal pixel clock is multiplied by 7. This clock is routed from the PLL to a BUFPLL primitive to drive one whole edge of the device. LVDS transmission is only possible on the top and bottom edges. The only outputs of the PLL that are capable of driving high-speed clocks to the BUFPLL are CLKOUT0 and CLKOUT1. The BUFPLL also requires a global clock signal equal to the original non-multiplied source clock (which can be driven from any of the PLL outputs through a global buffer (BUFG)), and the LOCKED signal from the PLL (which is required for synchronization inside the BUFPLL).

The three input signals allow the BUFPLL to distribute the high-speed transmission clock to the output SerDes primitives in the same edge of the device, along with the required SerDes strobe signal (appropriately aligned) that allows safe capture of low-speed parallel data from the FPGA logic into the output SerDes. This parallel data is then serialized for output using the high-speed transmission clock. The forwarded clock output is similarly generated by sending a constant value to the output SerDes associated with the clock line. For example, a forwarded DDR clock associated with 8:1 data requires the pattern `10101010`. A forwarded clock for 7:1 video applications requires `1110000` or `1111000`. The necessary circuit and output waveforms are shown in Figure 15.

There is no need for any PLL deskew when only using the PLL for data transmission, thus, internal feedback is used by connecting the feedback OUT pin of the PLL directly back to the feedback IN pin.
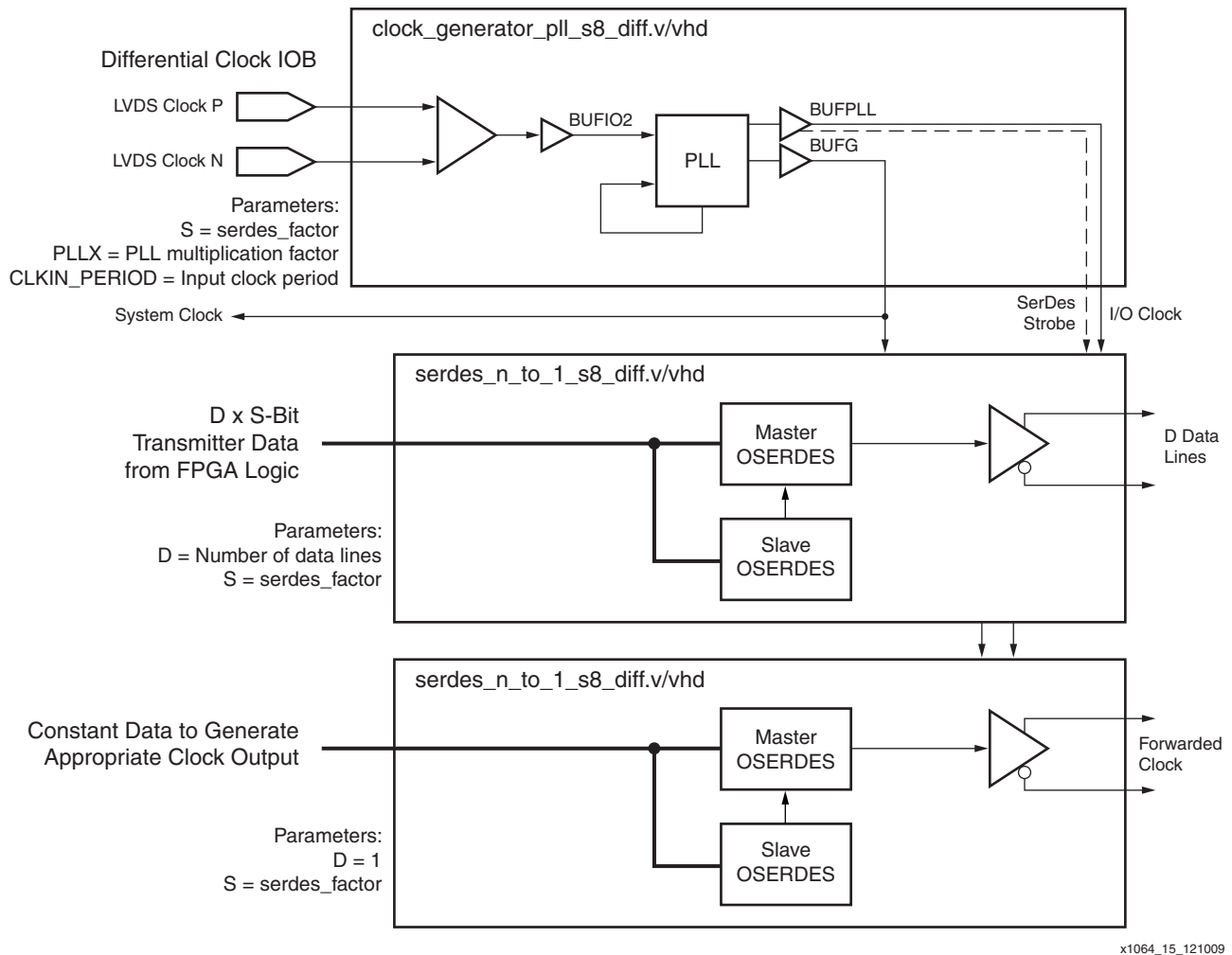

x1064_15_121009

Figure 15: **Data Transmission Using PLL and BUFPLL**

When one FPGA is performing both data reception and data transmission of a similar standard, the PLL and BUFPLL can be shared between the transmitter and receiver. One PLL can drive one or two BUFPLLs with the same clock. These PLLs can be on different edges of the device; however, each PLL is associated with different edges of the device. Possible connections are shown in Figure 16. In devices with four or less PLLs, any PLL can feed the BUFPLLs on any edge. In devices with five PLLs, the middle PLL in the top half of the device cannot feed any of the BUFPLLs. The input clock to a PLL has to come from a clock input pin through a BUFIO2 primitive to a PLL in the same vertical half (top or bottom) as the clock pin and BUFIO2. When designing with feedback where deskew is required, the feedback must come through a BUFIO2 primitive adjacent to the BUFIO2 that is driving the clock towards the PLL. Feedback can only come from a BUFIO2FB in the same vertical half (top or bottom) of the device as the PLL.
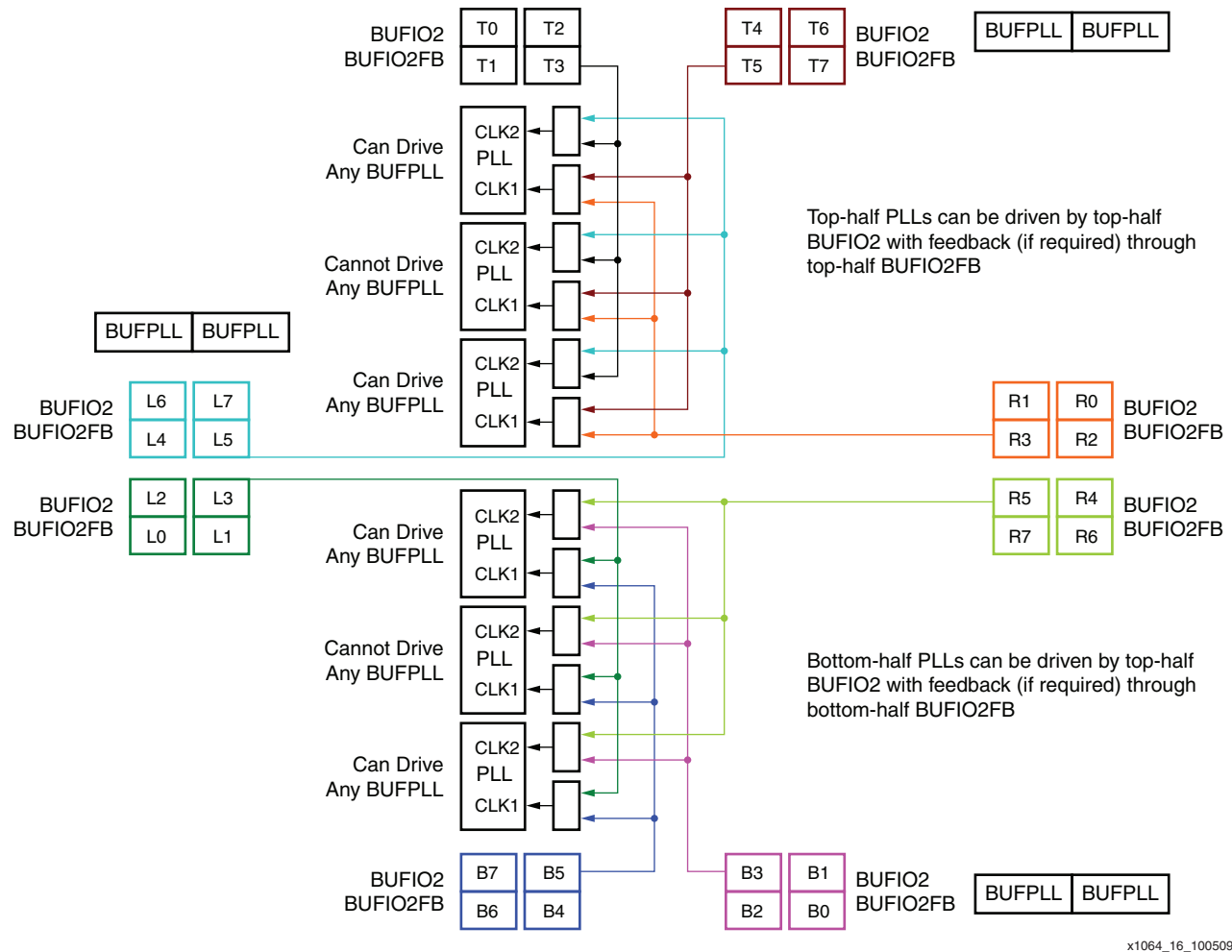
x1064_16_100509

*Figure 16:* **PLL to BUFPLL Connections**

Data Transmission of an SDR Forwarded Clock Using a PLL and Two BUFPLLs

The topology for this mechanism is uncomplicated. The transmitter source clock is multiplied as required in the PLL to generate two internal SDR transmitter clocks, one of which is twice the frequency of the other. These two clocks are routed from the PLL to two BUFPLL primitives on the same edge of the device. Each can drive the entire edge. However, LVDS transmission is only possible on the top and bottom edges. The only outputs of the PLL capable of driving high-speed clocks to the BUFPLL are CLKOUT0 and CLKOUT1. The BUFPLL for data transmission also requires a global clock signal equal to the original non-multiplied source clock, which can be driven from any of the PLL outputs through a global buffer (BUFG), and the LOCKED signal from the PLL, which is required for synchronization inside the BUFPLL. The BUFPLL for clock transmission requires a BUFG of twice the frequency of the original clock source, which again can be generated from any of the PLL outputs.

The three input signals to the BUFPLL allow distribution of the high-speed transmission clock to the output SerDes primitives on the same edge of the device, along with the required SerDes strobe signal (appropriately aligned) to allow safe capture of low-speed parallel data from the FPGA logic into the output SerDes. This parallel data is then serialized for output using the high-speed transmission clock. The forwarded clock output is generated in a similar manner but at twice the internal clock rate, by sending a constant value to the output SerDes associated with the clock line. For example, a forwarded SDR clock associated with 8:1 data requires the pattern `10101010`. The necessary circuit and output waveforms are shown in Figure 17.

There is no need for PLL deskew when only using it for data transmission, so internal feedback can be used by connecting the feedback out pin of the PLL directly back to its feedback pin.
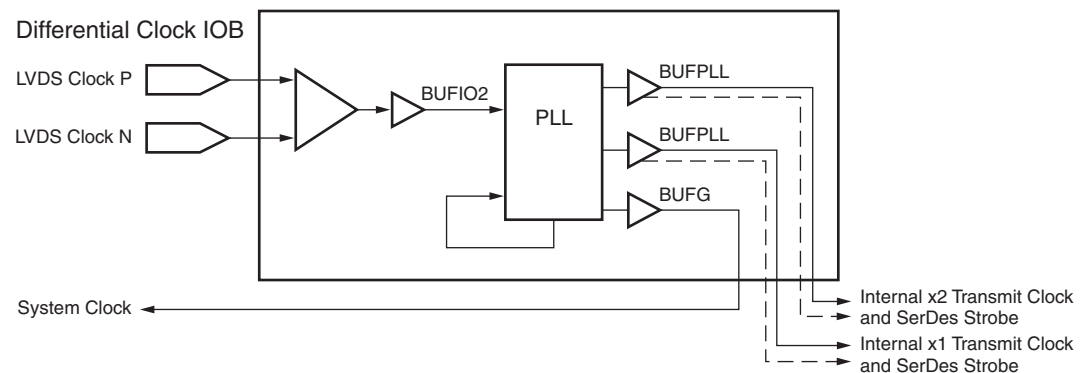
x1064_17_092809

Figure 17: **SDR Data Transmission through a PLL and BUFPLL**

Data Transmission Using Two BUFIO2s

Where a source transmitter clock is available that is half the required bit rate for transmission, two BUFIO2s can be used to generate the output data and forwarded clock. For example, a 311 MHz input clock can generate data at 622 Mb/s and a forwarded clock that is also 311 MHz.

Each Spartan-6 FPGA input clock pin is connected to the non-inverting input of one primary BUFIO2 and the inverting input of another primary BUFIO2 as long as the delay primitive is not used. Two clocks that are 180° apart in phase are therefore available. These two clocks are capable of being doubled inside each output SerDes configured for DDR operation when the output SerDes is in the same half side as the clock input.

In addition, the input clock can also feed the alternate BUFIO2 sites on the other half of the device, so it is possible to drive the whole side of a device by using four BUFIO2s, assuming the input delay primitive is not used, which is normally the case for data transmission.

One of the BUFIO2s is also used to generate a low-speed clock for the internal parallel data and the required SerDes strobe capture signal for the output SerDes primitives. The low-speed clock output is divided by the required SerDes factor and then distributed through a global buffer. For example, the incoming 311 MHz clock is divided by eight when 8:1 data transmission is required.

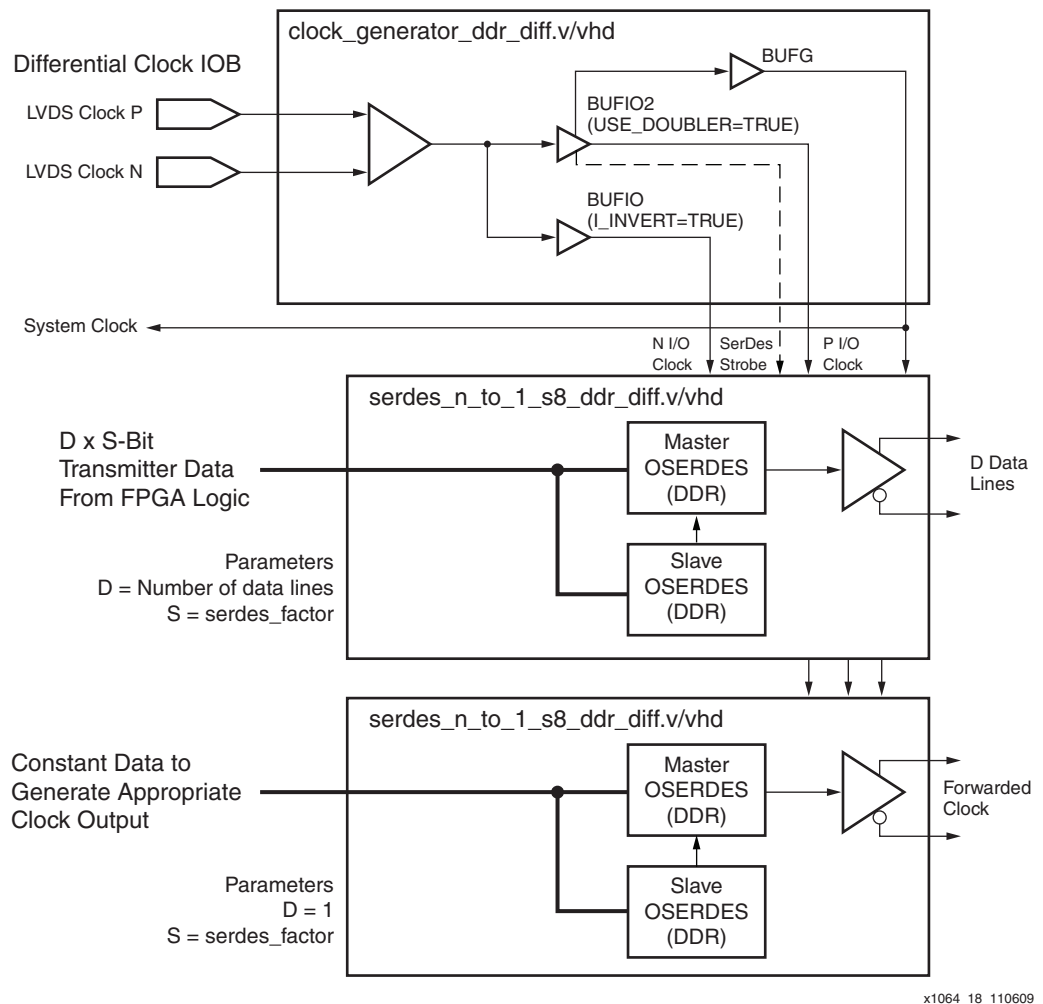The circuit and waveforms for this example are shown in Figure 18.

x1064_18_110609

Figure 18: **Data Transmission through Two BUFIO2s**

Data Transmission Using a BUFIO2

Where a source transmitter clock is available that is equal to the required bit rate for transmission, a single BUFIO2 can be used to generate the output data and a DDR forwarded clock. For example, a 622 MHz input clock can generate data at 622 Mb/s and a forwarded clock that is 311 MHz.

The input clock pin is connected to its associated BUFIO2, which can drive all the associated output SerDes in the same half side of a device. In addition, the input clock can also feed an alternate BUFIO2 site in the other half side of the device. It is possible to drive the whole side of a device by using two BUFIO2s, assuming the input delay primitive is not used, which is normally the case for data transmission.

The BUFIO2 is also used to generate a low-speed clock for the internal parallel data and the required SerDes strobe capture signal for the output SerDes primitives. The low-speed clock output will be divided by the required SerDes factor and then distributed through a global buffer. For example, the incoming 622 MHz clock can be divided by eight for a 8:1 data transmission.

The circuit and waveforms for this example are shown in Figure 19.
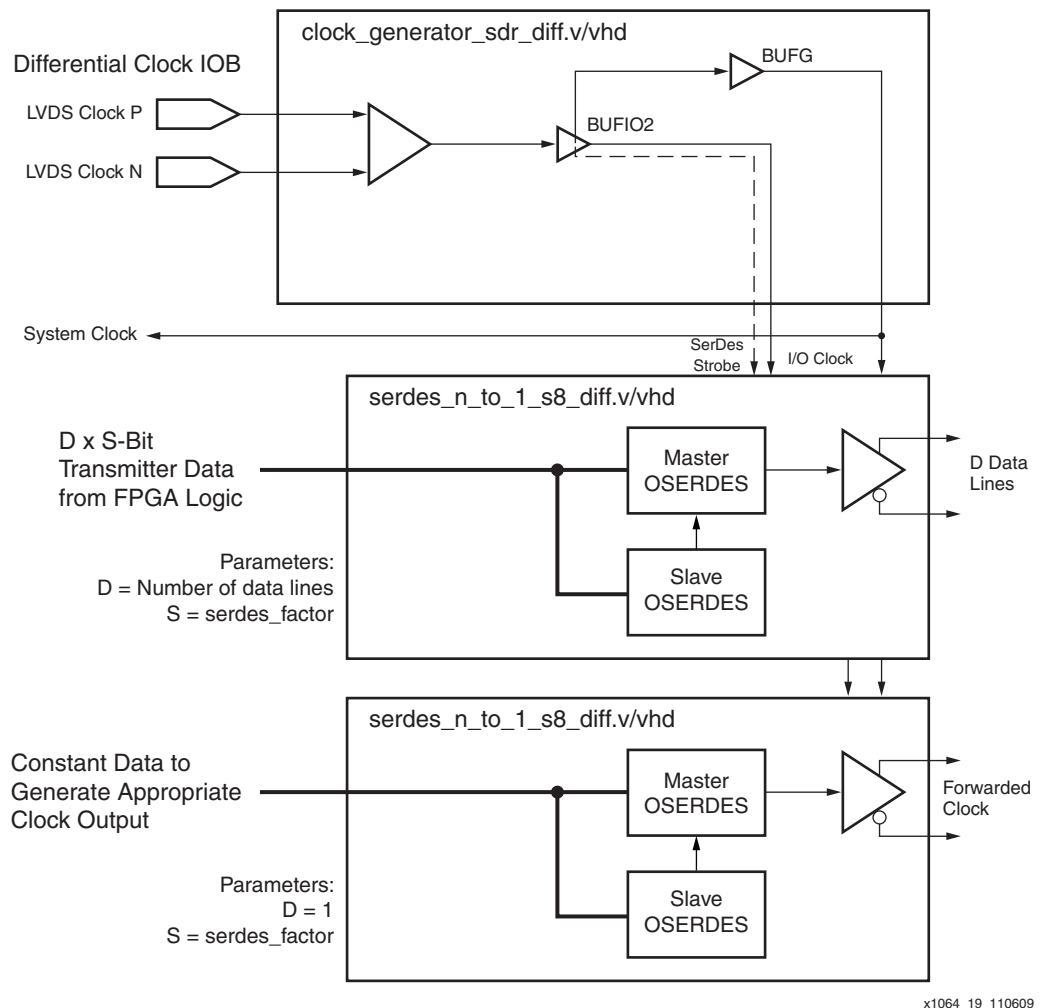

x1064_19_110609

Figure 19: **Data Transmission through Two BUFIO2s**

www.BDTIC.com/XILINX

Data Transmission of an SDR Forwarded Clock Using Two BUFIO2s

The topology for this mechanism combines the BUFIO2s used in single and double-rate modes for transmission of data together with an associated SDR forwarded clock.

The incoming local transmitter source clock is connected to two BUFIO2s, which can be used to generate a doubled clock inside the output SerDes associated with the forwarded clock, whereas the output SerDes associated with the output data lines uses one undoubled clock.

For example, if a 622 MHz clock is available, this can be distributed to the data output SerDes primitives to generate data at 622 Mb/s. It is also available by using both BUFIO2 clocks and DDR mode in the clock output SerDes to regenerate the 622 MHz clock through a constant `10101010` pattern. The BUFIO2 is configured to provide a divided clock by setting its divide parameter equal to the SerDes ratio desired. For example, with the 622 MHz input clock, a division by eight enables 8:1 output SerDes operation, with an internal system clock of 77.75 MHz.

The circuit and waveforms for this example are shown in Figure 20.
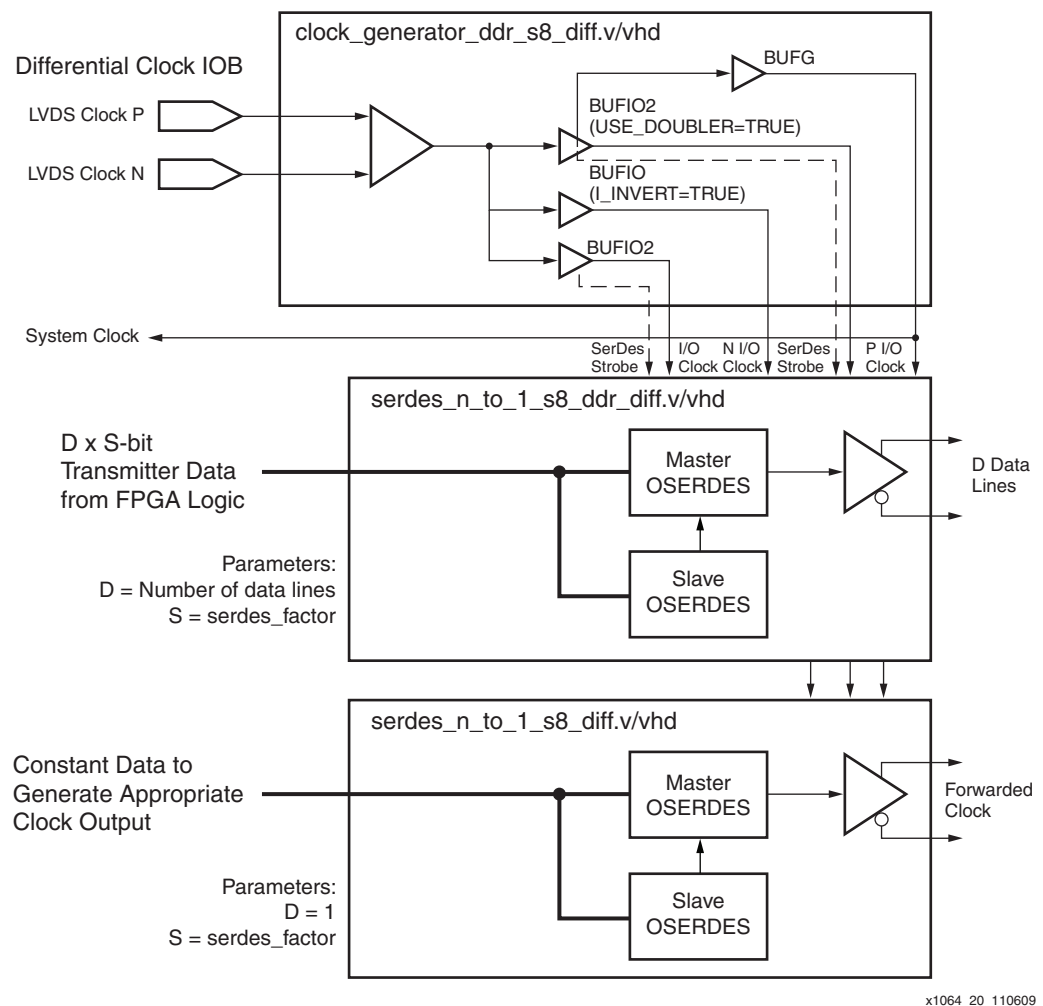
x1064_20_110609

Figure 20:   **SDR Data Transmission through Two BUFIO2s**

Design Considerations

Transmitter Use with Single-Ended Data and Clocks

All of the transmitter examples used differential clock and data signals. When transmission requires a single-ended clock or data signal, then certain restrictions apply for SerDes factors greater than four. The design file (`serdes_n_to_1_s8_se.v/vhd`) contains the necessary serializer logic for a single-ended output (either forwarded clock or data) for SerDes factors from 2 to 8. When the SerDes factor is 4 or less, only one OSERDES2 is used, and device pins adjacent to each other can be used to form a data bus. When the SerDes factor is 5 to 8, then two OSERDES2s are used and the pin next to the active output (which must be a master or _p pin) is blocked from use as a synchronous output since the necessary logic is already occupied.

Receiver Use with Single-Ended Data and Clocks

All of the receiver examples used differential clocks and data signals. When the received clock and/or data is a single-ended signal, then certain restrictions apply.

Differential Data Signals with a Differential Clock

All of the design files and techniques given are valid.

Single-Ended Data Signals with a Differential Clock

The clock is received as in the previous examples. The data is received using the design file `serdes_1_to_n_data_s8_se.v/vhd`. When the SerDes factor is 4 or less and the phase-detector option is not selected, then the receiver only uses one ISERDES2, and device pins adjacent to each other can form a data bus. When the SerDes factor is 5 to 8 or the phase detector function is selected, two ISERDES2s are required and the pin next to the active input (which must be a master or _p pin) is blocked from use as a synchronous input since the necessary logic is already occupied.

Differential Data Signals with a Single-Ended Clock

The data is received as in the previous examples. When PLL clocking is required, the clock is received using the design file `serdes_1_to_n_clk_pll_s8_se.v/vhd`. When the SerDes factor is 4 or less (or not required at all), the clock receiver only uses one ISERDES2, and the adjacent clock pin can be used. When the SerDes factor is 5 to 8, two ISERDES2s are required and the clock pin next to the active clock input (which must be a master or _p clock pin) cannot be used as a synchronous input since the necessary logic is already occupied. However, it can be used a clock input.

The file `serdes_1_to_n_clk_sdr_s8_se.v/vhd` is used when SDR BUFIO2 data reception is required. When the SerDes factor is 4 or less (or not required at all), the clock receiver only uses one ISERDES2, and the adjacent clock pin can be used. When the SerDes factor is 5 to 8, then two ISERDES2s are required and the clock pin next to the active clock input (which must be a master or _p clock pin) cannot be used as a synchronous input since the necessary logic is already occupied. However, it can be used a clock input.

DDR BUFIO2 data reception adds complexity. A single-ended clock pin can only feed one BUFIO2 through an input delay, limiting access to the two BUFIO2s required for DDR reception. In this case, the solution is to feed the incoming single-ended clock simultaneously to two clock input pins. The master (_p) pin feeds one BUFIO2 directly through an input delay, and the slave (_n) pin inverts the clock inside the IOB and feeds the second BUFIO2 through a second input delay. The design file for this example is `serdes_1_to_n_clk_ddr_s8_se.v/vhd`.

Single-Ended Data and Clock Signals

The data is received using the design file `serdes_1_to_n_data_s8_se.v/vhd`. When the SerDes factor is 4 or less and the phase detector option is not selected, the receiver uses only one ISERDES2, and device pins adjacent to each other can form a data bus. When the SerDes factor is 5 to 8 or the phase detector function is selected, two ISERDES2s are required and the pin next to the active input (which must be a master or _p pin) cannot be used as a synchronous input since the necessary logic is already occupied.

The file `serdes_1_to_n_clk_sdr_s8_se.v/vhd` is used when SDR BUFIO2 data reception is required. When the SerDes factor is 4 or less (or not required at all), the clock receiver only uses one ISERDES2, and the adjacent clock pin can be used. When the SerDes factor is 5 to 8, then two ISERDES2s are required and the clock pin next to the active clock input (which must a master or _p clock pin) cannot be used as a synchronous input since the necessary logic is already occupied. However, it can be used a clock input.

DDR BUFIO2 data reception adds complexity. A single-ended clock pin can only feed one BUFIO2 through an input delay, limiting access to the two BUFIO2s required for DDR reception. In this case, the solution is to feed the incoming single-ended clock simultaneously to two clock input pins. The master (_p) pin feeds one BUFIO2 directly through an input delay, and the slave (_n) pin inverts the clock inside the IOB and feed the second BUFIO2 through a second input delay. The design file for this example is `serdes_1_to_n_clk_ddr_s8_se.v/vhd`.

Receiver Timing Analysis

Timing analysis for the receiver consists of subtracting the various sources of timing errors and uncertainty from the bit period in picoseconds (ps) equivalent to the bit rate. The value remaining after this analysis is the margin available to the system. A positive number indicates that the system has sufficient margin and will function properly.

The receiver skew margin (RSKM) is a specification that often appears in data sheets for ASSPs or other devices that perform a similar deserialization function. This value is generated by subtracting only the sources of uncertainty that exist inside the receiver from the bit period, and then dividing the result by two. An illustration of RSKM is shown in Figure 21.
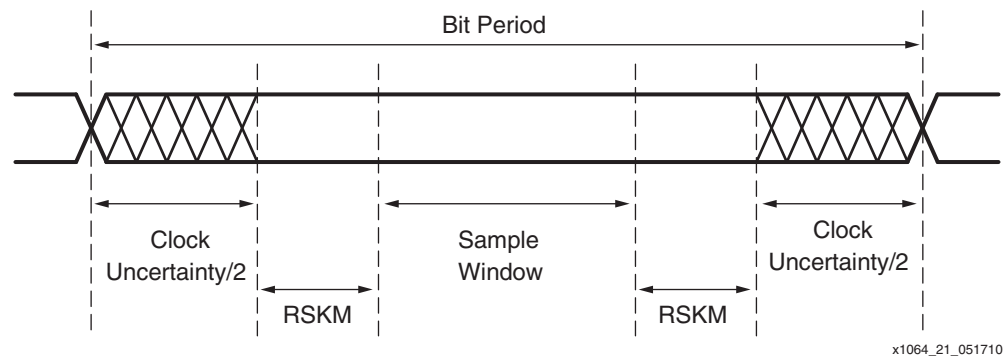
x1064_21_051710

Figure 21: **Receiver Skew Margin**

Uncertainty Without Phase Detector

For the interfaces described in this application note that use calibration, but do not use the phase detector, the sources of uncertainty are:

- All mismatch and silicon variations are bundled into one parameter, denoted as T_{SAMP_BUFIO2}, which is guaranteed by characterization to be better than 480 ps for all Spartan-6 devices with LVDS signalling. This number includes the setup and hold window of the device, which is the time that the data must be present and valid relative to the internal synthesized clock at the IOB flip-flops (assuming that the input data lines are calibrated to be half of the unit interval (UI) delay).

- Package Skew

  This number varies with the placement of the input lines in the package and is available from TRACE when the design is analyzed.

- Clock Skew

  The BUFIO2 clocks are designed as full clock trees, the IOB skew is very small. An accurate number for a given device and placement can be obtained using FPGA Editor.

- Jitter and Timing Uncertainty

  The clocking wizard (available in the Core Generator™ tool in the ISE® design suite) generates a value for the jitter accumulated in the transmitter PLL.

Uncertainty With Phase Detector

For the interfaces described in this application note that use the phase detector, the sources of uncertainty are:

- Accuracy of the Phase Detector Mechanism

  The design of the phase detector and the state machine in the FPGA logic give a sampling point that is within ±2 delay taps of the ideal sampling point. The phase error of the PLL generated sampling clock is not a factor.

- Package Skew

  This number varies with the placement of the input lines in the package and is available from TRACE when the design is analyzed.

- Jitter and Timing Uncertainty

  The clocking wizard (available in the Core Generator™ tool in the ISE® design suite) generates a value for the jitter accumulated in the transmitter PLL.

The input delay line is made up of groups of eight tap delays that are used up to 32 times in an active delay line (equals 256 tap delays total). DS162: *Spartan-6 FPGA data sheet* specifies the maximum values to reach each tap. Table 1 shows some calcuation examples.

Table 1: **Example Calculation of Maximum Individual Delay Between Taps**

	Example Delay	**Calculation**	**Total Delay Between 4 Successive Taps**
T_{TAP1}	61 ps	Delay from T_{TAP1} to T_{TAP5}	170 ps
T_{TAP2}	77 ps	Delay from T_{TAP2} to T_{TAP6}	215 ps
T_{TAP3}	140 ps	Delay from T_{TAP3} to T_{TAP7}	203 ps
T_{TAP4}	166 ps	Delay from T_{TAP4} to T_{TAP8}	258 ps
T_{TAP5}	231 ps	Delay from T_{TAP5} to T_{TAP1}	170 ps
T_{TAP6}	292 ps	Delay from T_{TAP6} to T_{TAP2}	215 ps

Table 1: **Example Calculation of Maximum Individual Delay Between Taps** *(Cont'd)*

	Example Delay	Calculation	Total Delay Between 4 Successive Taps
T_{TAP7}	343 ps	Delay from T_{TAP7} to T_{TAP3}	203 ps
T_{TAP8}	424 ps	Delay from T_{TAP8} to T_{TAP4}	258 ps

An analysis of the example in Table 1 shows that the worst-case span of four taps is 258 ps. This example would suggest that based on phase-detector usage, a delay value of ±129 ps delay should be used for any calculations.

Another RSKM calculation example (Table 2) uses an input clock running at 135 MHz and multiplied to 945 MHz in a PLL to clock in 945 Mb/s data.

Table 2: **Example Calculation**

Bit Period at 945 Mb/s	1058 ps
Package skew (refer to TRACE for precise values)	−120 ps
PLL Jitter (from the clocking wizard)	−112 ps
Phase Detector accuracy	−129 ps
Total	697 ps
RSKM = Total/2	349 ps

Transmitter Timing Analysis

For the interfaces described in this application note, the sources of uncertainty are:

- Package Skew

  This number varies with the placement of the input lines in the package and is available from TRACE when the design is analyzed.

- Jitter and Timing Uncertainty

  The clocking wizard (available in the Core Generator tool in the ISE design suite) generates a value for the jitter accumulated in the transmitter PLL.

Reference Design Files

Design files for the majority of the examples explained in this application note are available in both Verilog and VHDL at (https://secure.xilinx.com/webreg/clickthrough.do?cid=140956). The name of the appropriate file is included in the figures for different applications shown throughout this document. Also included are some example top-level files and example timing constraints for popular applications, such as the 7:1 interface used in flat panel displays and cameras.

Each of the data input and output modules can be parameterized for both input width (number of input pins) and depth (required SerDes factor), and there are versions for both singe-ended and differential I/O. The data receiver modules also contain a signal to indicate whether the generation of phase-detector logic is required in the example where it is required to deskew the input bus. The phase-detector mode is always used to allow input delay calibration to occur without any data loss occurring, and adding the phase-detector logic allows reliable operation at higher bit rates, and where the incoming data has an unknown phase to the incoming clock. The parallel data generated from the receiver modules is the width multiplied by the depth, for example receiving a 6-bit bus of 7:1 data will output 42 bits for each system clock cycle.

General information about the reference design is shown in Table 3. The device utilization is shown in Table 4.

Table 3: **Reference Design Checklist**

General Information	
Developer name	Xilinx
Target devices	Spartan-6 FPGAs
Source code provided	Yes
Source code format	VHDL, Verilog
Design uses code/IP from an existing reference design, application note, 3rd party, Core Generator	No
Simulation	
Functional simulation performed	Yes
Timing simulation performed	No
Testbench used for functional and timing simulations provided	Yes
Testbench format	VHDL, Verilog
Simulator software used	MXE
SPICE/IBIS simulations	No
Implementation	
Synthesis software tools used	XST 11.3
Implementation software tools used	ISE 11.3
Static timing analysis performed	Yes
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	SP601 and FMC101

Table 4: **Device Utilization**

Design Files	IODELAY2s	PLLs	BUFPLLs	BUFIO2	BUFGs	Slices	ISERDES2	OSERDES2
Clock Generator Designs								
clock_generator_ddr_s8_diff	0	0	0	2 or 4	1	0	0	0
clock_generator_pll_s16_diff	0	1	1	1	2	0	0	0
clock_generator_pll_s8_diff	0	1	1	1	1	0	0	0
clock_generator_sdr_s8_diff	0	0	0	3	1	0	0	0
Clock Receivers								
serdes_1_to_n_clk_ddr_s8_diff	2	0	0	2	1	< 10	0, 1, or 2	0
serdes_1_to_n_clk_ddr_s8_se	2	0	0	2	1	< 10	0, 1, or 2	1
serdes_1_to_n_clk_pll_s16_diff	2	1	1	1	2	< 10	0, 1, or 2	2
serdes_1_to_n_clk_pll_s8_diff	2	1	1	1	1	< 10	0, 1, or 2	3
serdes_1_to_n_clk_pll_s8_se	1 or 2	1	1	1	1	< 10	0, 1, or 2	4
serdes_1_to_n_clk_sdr_s8_diff	1	0	0	1	1	< 10	0, 1, or 2	5
Data Receivers								

 www.xilinx.com

Table 4: **Device Utilization** *(Cont'd)*

Design Files	IODELAY2s	PLLs	BUFPLLs	BUFIO2	BUFGs	Slices	ISERDES2	OSERDES2
serdes_1_to_n_data_ddr_s8_diff	2	0	0	0	0	~4 per input line	2	0
serdes_1_to_n_data_ddr_s8_se	1 or 2	0	0	0	0		1 or 2	0
serdes_1_to_n_data_s16_diff	2	0	0	0	0		2	0
serdes_1_to_n_data_s8_diff	2	0	0	0	0		2	0
serdes_1_to_n_data_s8_se	1 or 2	0	0	0	0		1 or 2	0
Data Transmitters								
serdes_n_to_1_ddr_s8_diff	0	0	0	0	0	0	0	2
serdes_n_to_1_ddr_s8_se	0	0	0	0	0	0	0	1 or 2
serdes_n_to_1_s16_diff	0	0	0	0	0	0	0	2
serdes_n_to_1_s8_diff	0	0	0	0	0	0	0	2
serdes_n_to_1_s8_se	0	0	0	0	0	0	0	1 or 2

Conclusion

Spartan-6 FPGAs perform in a wide variety of applications requiring various serialization and deserialization factors up to 16-to-1, at speeds up to 1050 Mb/s, depending on the application, speed grade, and package.

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
12/23/09	1.0	Initial Xilinx release.
06/03/10	1.1	Added Receiver Timing Analysis and Transmitter Timing Analysis sections.

Notice of Disclaimer