LatticeSC flexiPCS/SERDES Design Guide

October 2008 Technical Note TN1145

Introduction

This document has been provided to assist the designer in using the flexiPCS™/SERDES block in the LatticeSC™ FPGA. The LatticeSC/M Family flexiPCS Data Sheet provides details on the features of the flexiPCS including the interface to the FPGA. This document is intended to supplement the data sheet with design-specific information and how to create and use the flexiPCS.

Control Plane Options

The flexiPCS control interface can be implemented in either a static or dynamic mode. A static implementation will setup the flexiPCS during bitstream configuration. Control of the flexiPCS at run time will be limited to the ports available on the flexiPCS interface to the FPGA. A dynamic implementation will provide access to the embedded memory map via the system bus to be able to control the flexiPCS at run time. Both the static and dynamic implementations of the flexiPCS will use an autoconfig file to initialize the flexiPCS during bitstream configuration. The static mode can only be used if the user does not plan to access the flexiPCS memory map at run time and does not plan on using quad alignment of multiple flexiPCS quads.

Autoconfig File

The autoconfig file is produced when creating the flexiPCS module in IPexpress™. This file is unique to each instance created in IPexpress. The file contains the default register settings for the memory map of the flexiPCS to provide the user with modes selected in IPexpress. This file is used during simulation as well as bitstream generation. The file is found in the same location as the IPexpress module. It must be copied to the location of the simulation environment as well as the ispLEVER® project.

System Bus Interface

The LatticeSC provides an on-chip bus which interconnects all of the embedded blocks including PLLs, DLLs, certain MACO™ blocks, and the flexiPCS/SERDES. The interconnection to the system bus and the flexiPCS uses dedicated wires. This interconnection is used to load the contents of the autoconfig file during bitstream configuration, issue run time transactions via the system bus, and pass clocking and multi-channel alignment information when performing multi-quad flexiPCS data alignment.

The system bus has dedicated ports for each flexiPCS site on the device. When creating the system bus in IPexpress the user must select the flexiPCS sites for which the system bus is to be connected.

Multi-Quad/Chip Alignment

The system bus and dedicated interconnection to flexiPCS quads are used to perform multi-quad and multi-chip alignment. For multi-quad alignment there are up to four alignment groups that are defined using the system bus. Once these groups are defined the clocking and alignment information is passed on the dedicated interconnect lines.

For multi-chip alignment the same interconnect is used with the addition of six top level ports on the system bus which pass information to the other chip. A chip is either defined as master or a slave which is selected when creating the system bus in IPexpress. There are slave ports and master ports which must be used at the top level of the design. These top level ports have dedicated pin locations on the device.

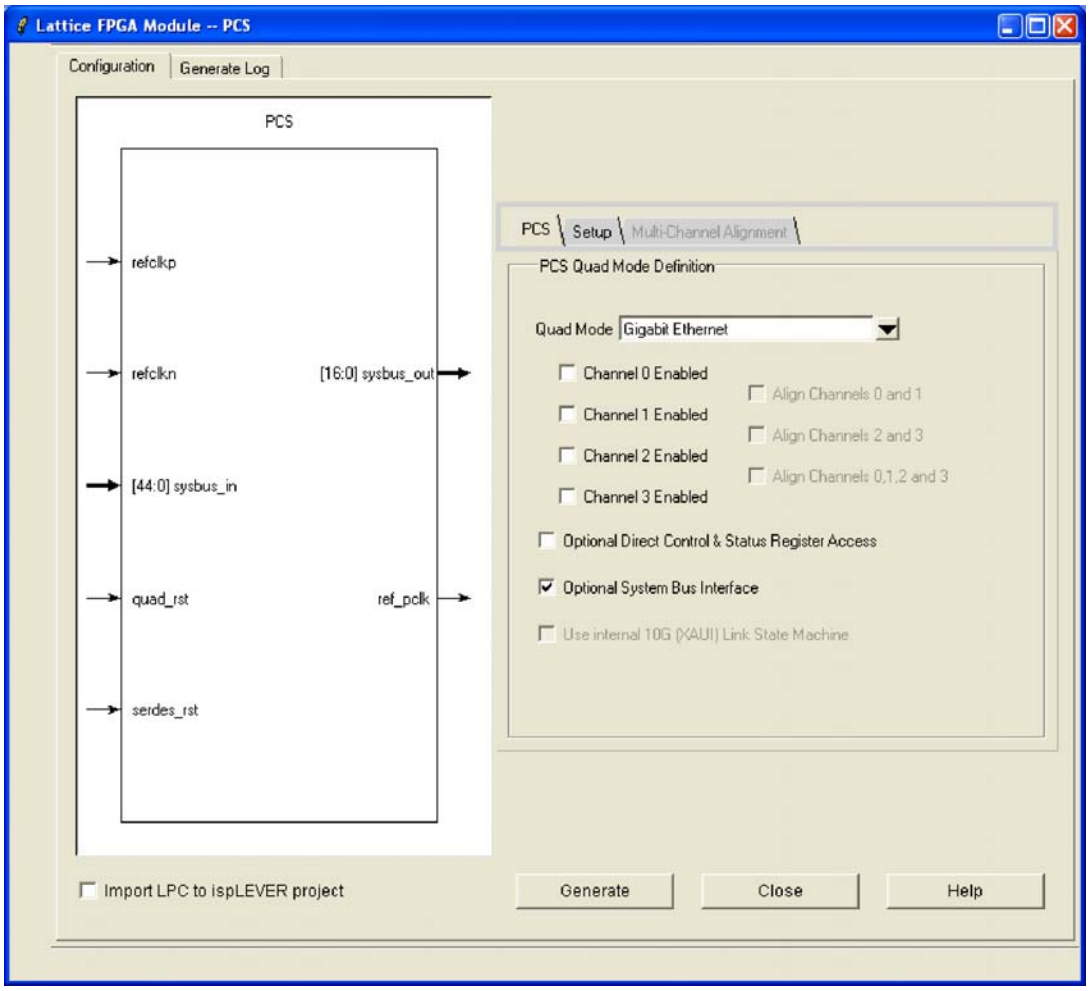
Creating a flexiPCS/SERDES in IPexpress

The flexiPCS/SERDES module is customized and created using IPexpress. The user selects the type of protocol, which channels to be enabled, multi-channel alignment options, and clocking options.

Note: The LatticeSC library element for the flexiPCS is named PCSA. Do not use the File Name PCSA when creating the flexiPCS in IPexpress. This will cause a naming conflict in the synthesis tool.

- **Quad Mode** – Selects the protocol to use for the entire flexiPCS quad. Selections include 8-bit SERDES Only, 10-bit SERDES Only, SONET, Gigabit Ethernet, Fibre Channel, XAUI, Serial RapidIO, PCI-Express, Generic 8b10b. This selection sets the protocol mode for the entire flexiPCS quad (all four channels).
- **Channel Enable** – Select which channels to enable of the flexiPCS quad. In the autoconfig file any channel that is enabled will be powered up. Channels can later be powered down using the run time memory map.
- **Align Channels** – Select a static multi-channel alignment mode. This is the default multi-channel setup in the autoconfig file. This alignment mode can be later changed using the run time memory map. Selecting a channel alignment grouping will enable the Multi-Channel Alignment tab.
- **Optional Direct Control & Status Register Access** – Each protocol mode has option ports that can be selected. These ports are defined in the LatticeSC/M Family flexiPCS Data Sheet per protocol mode.
- **Optional System Bus Interface** – This option controls whether the system bus ports are available on the flexiPCS for the system bus. The system bus must be used if using the flexiPCS memory map or using multi-quad alignment.
- **Use Internal 10G (XAUI) Link State Machine** – Select this to use the 10G (XAUI) link state machine to control the word aligner. This option is only available when Generic 8b10b is selected in the Quad Mode pull-down menu.

Figure 1. IPexpress flexiPCS Module

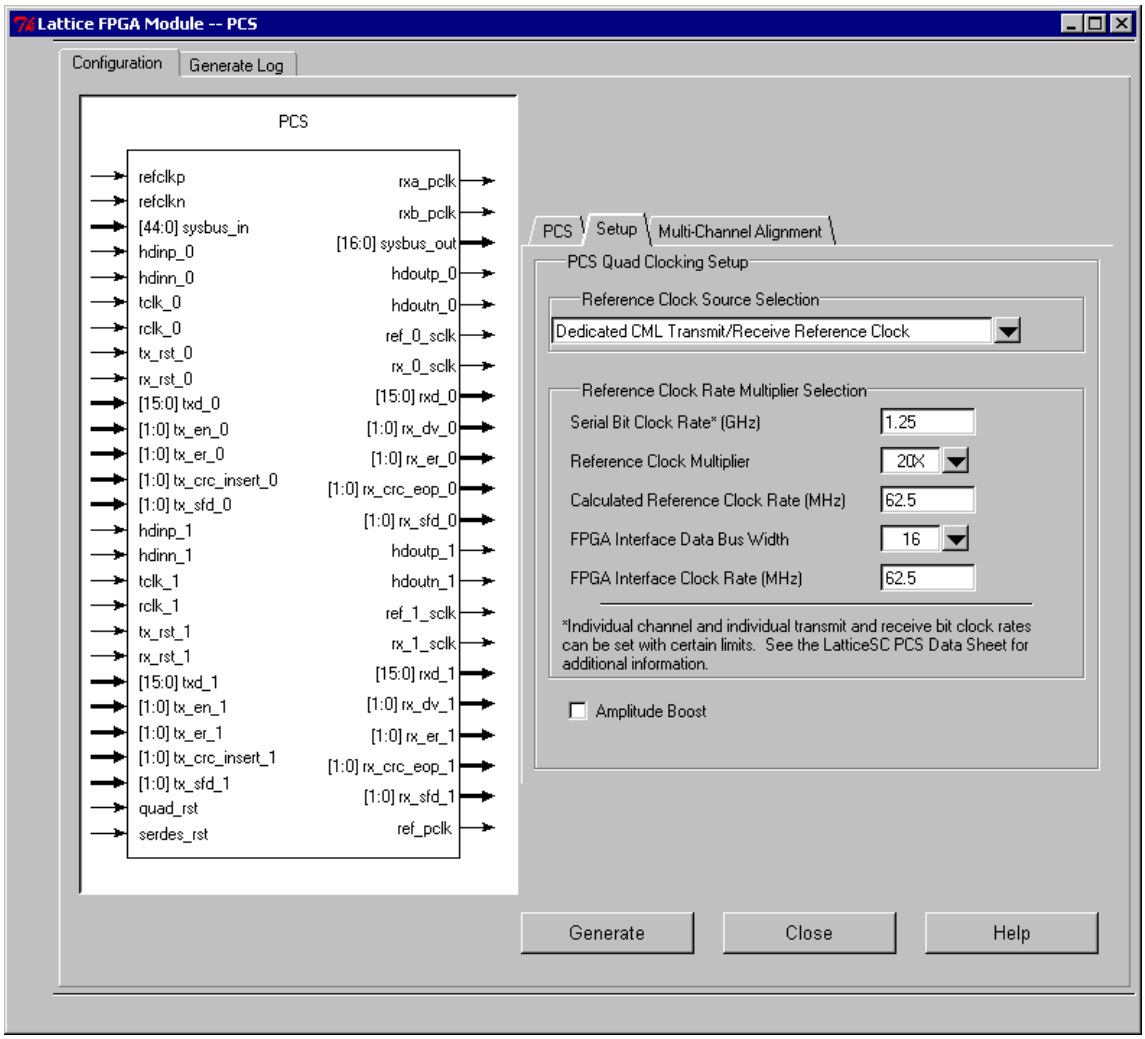


Setup Tab

- **Reference Clock Source Selection** - Selects the source of the reference clock for the Tx and Rx clocks to the SERDES. This option controls the ports on the module and how they are connected to the SERDES. If “All Reference Clocks” is selected the user will need to set the autoconfig file to the correct default for reference clock selection in quad register 0x29. The default value written in the autoconfig file is 0x00 which is the external reference clock.
- **Reference Clock Rate Multiplier Selection** - This section of controls allows the user to customize the clock rates to match the desired serial bit rate, clock source, and FPGA interface. The SERDES supports various multipliers and the FPGA interface supports various interface widths depending on the selected protocol. The SERDES reference clock rate and FPGA interface clock rate are automatically calculated dependent on the Serial Bit Clock Rate, Reference Clock Multiplier and FPGA Interface Data Bus Width.

  Once a high speed bit rate has been selected in ispLEVER, the SERDES are optimized for that particular rate. This is done via attributes that are placed on the flexiPCS module which control performance settings in the bitstream. If the user needs to modify the rate at which the SERDES operates, IPexpress should be used to regenerate the module so that new attributes can be assigned for the new high speed rate. Switching between full-rate and half-rate modes is the exception to this requirement. In this instance, a new flexiPCS module is not required.
- **Amplitude Boost** - This option will increase the amplitude of the transmitter while increasing the power slightly.

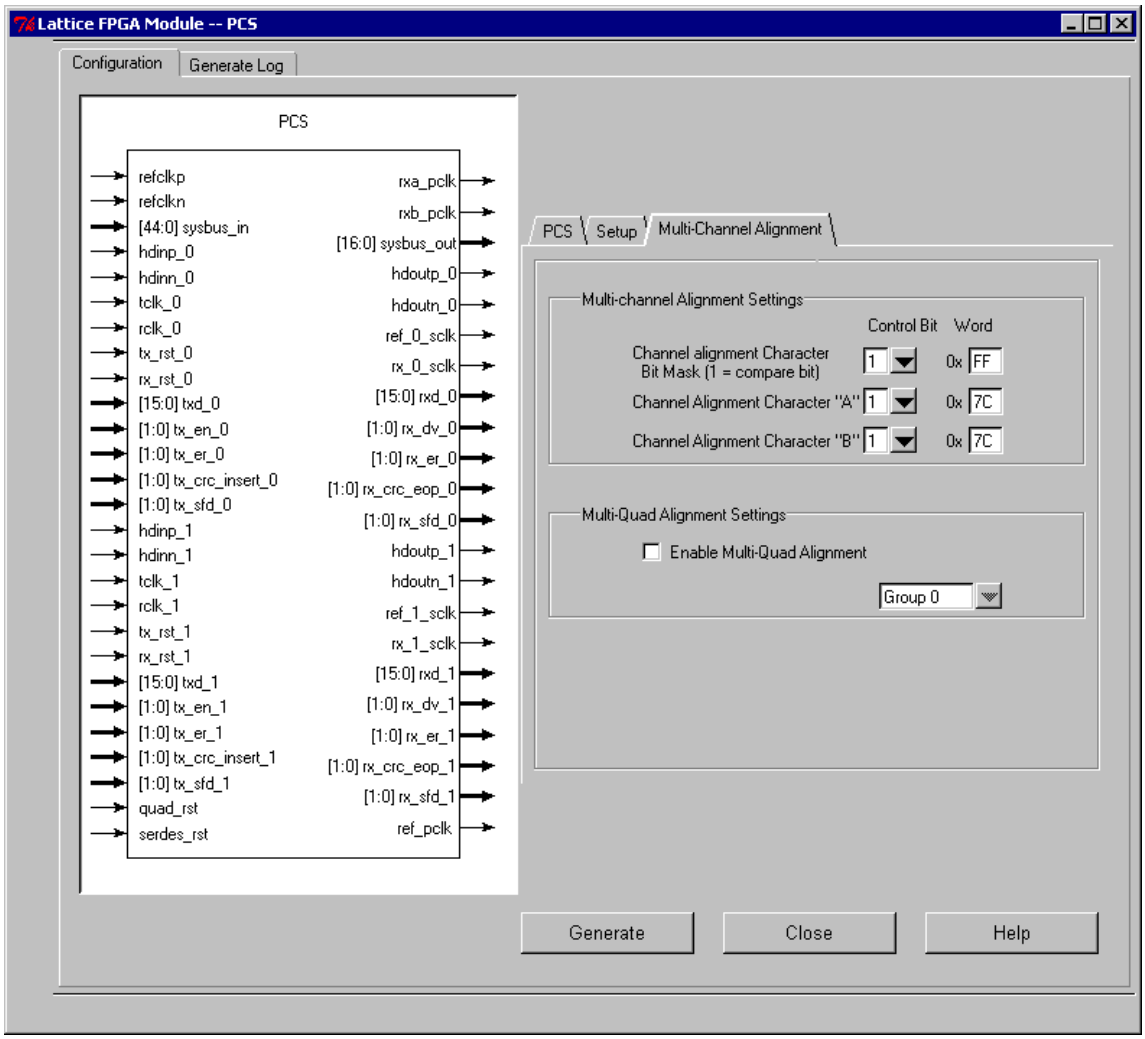
Figure 2. IPexpress flexiPCS Module Setup Tab



Multi-Channel Alignment Tab

- **Multi-channel Alignment Settings** - This section of controls allows the user to select the alignment character that will be used to align the data streams.
- **Multi-Quad Alignment Settings** - This section of controls enables multi-quad alignment and selects the group clock associated with the flexiPCS quad.

Figure 3. IPexpress flexiPCS Module Multi-Channel Alignment Tab



IPexpress will create several files using the file name specified. The following files will be present in the project path directory specified in IPexpress.

- **PCSA.v (Verilog only)** – This file is used for both synthesis and simulation and contains the black box definition and call to the simulation model for the flexiPCS/SERDES. This file is the same for all flexiPCS modules.
- **<user name>.v or <user name>.vhd** – This is the module generated by IPexpress that is customized to the user's options. This file is used in both synthesis and simulation.
- **<user name>.readme** – This file contains a sample code for the user to use in the top level of the design. First there is a sample module definition using the SERDES port and attributes. Next there is a sample instance for the user to copy.

<user name>.txt - This is the autoconfig file which contains default settings for the flexiPCS memory map. This file can be modified by the user to add different options that are not covered by the IPexpress such as CML buffer settings. This file must be copied to the simulation and ispLEVER project directory.

Clocking

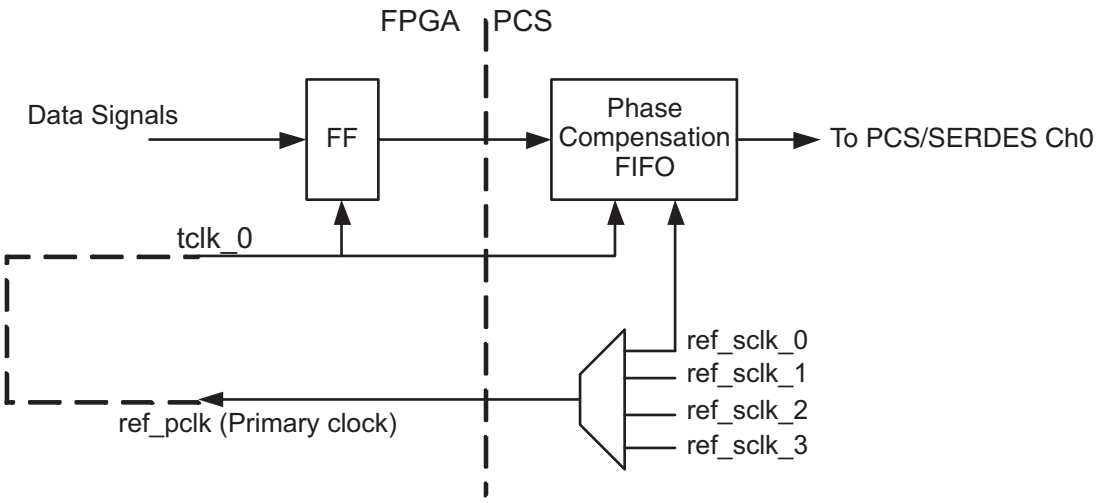
The flexiPCS/SERDES requires a reference clock which will be multiplied x5, x10, or x20 to create the high speed bit rate. If the source of the refclk is from the FPGA then it is necessary to ensure the refclk is coming from a primary clock. Primary clocks have the least amount of jitter and should only be used for the refclk source.

The high speed bit rate will then be divided back down to create the lower speed clocks for the flexiPCS to FPGA interface. This section provides the recommended interface clocking for the various modes of the flexiPCS.

Transmit Clocking

The transmit direction clocking recommendation is the same for all modes of the flexiPCS/SERDES. Figure 4 provides the recommended clocking diagram for the transmit interface of a single channel.

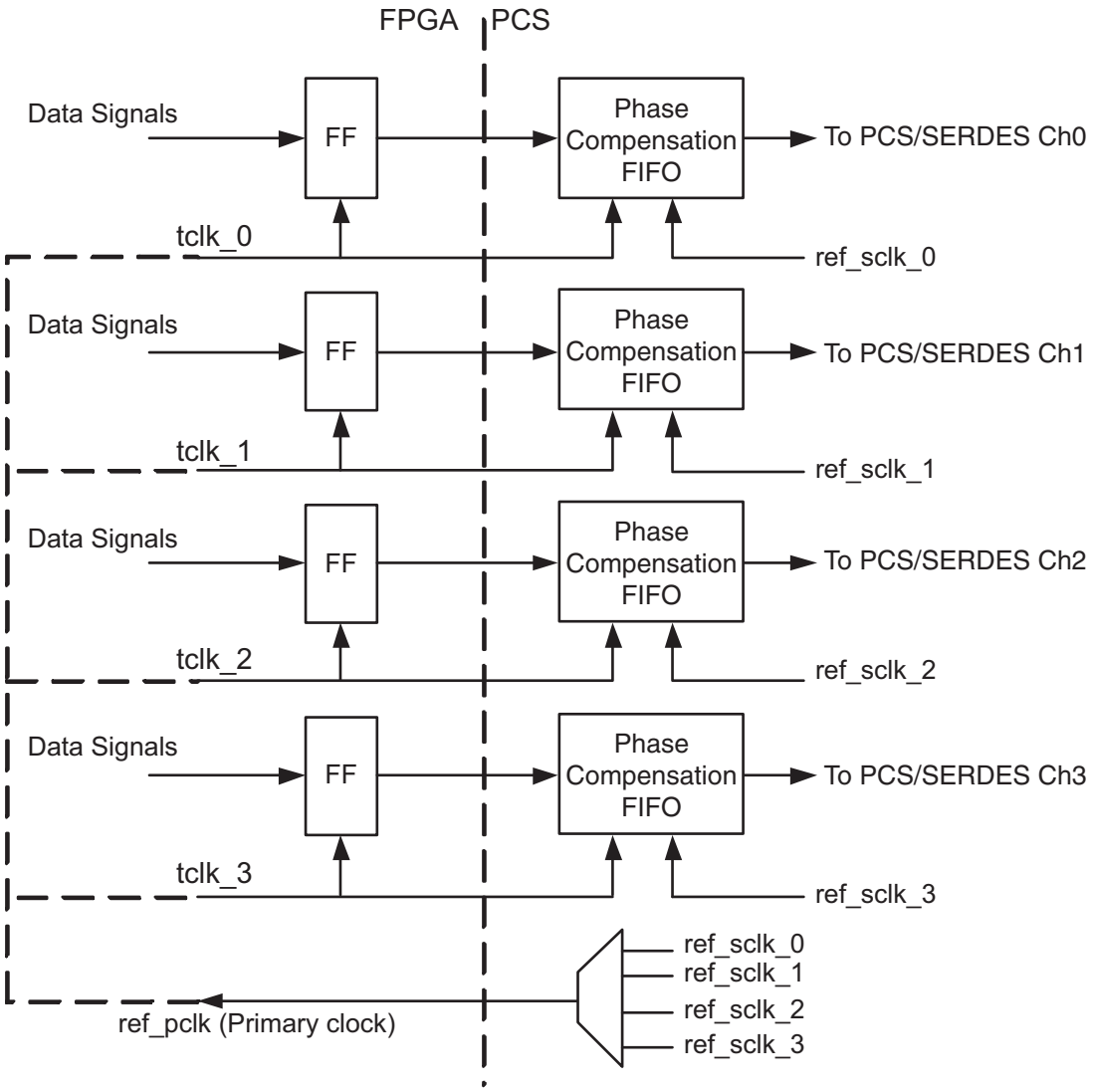
Figure 4. flexiPCS Interface Transmit Clocking for a Single Channel



The ref_pclk output clock is derived from the reference clock. There is a mux controlled via a memory map register which controls the source of this clock based on channel. This mux is useful if a transmit channel is powered down and needs to switch to another channel.

Figure 5 provides the recommended clocking diagram for the transmit interface using all four channels.

Figure 5. flexiPCS Interface Transmit Clocking for all Four Channels



In the transmit channel there is a single PLL to create the high speed bit rate clock. Each channel then divides down the high speed clock to the data rate of the FPGA interface. Since all of the channels are running at the same rate in the transmit direction all of the channels can use the same clock from one channel. Again, the mux controlling the source of ref_pclk is only important if a transmit channel is powered down.

Note: If a mixture of half-rate and full-rate channels are used in a quad, then the secondary clock must be used per channel as shown in Figure 10.

Transmit Clock Source Requirements

The FPGA interface phase compensation FIFOs are intended only to provide a clean timing interface to the PCS logic and cannot be used to arbitrate between different clock frequencies. Therefore the tclk_[0-3] ports must be connected to clocks that are frequency locked to the reference clock. Typically, they are fed from the locked reference clock outputs from the PCS to the FPGA as detailed in Figures 4 and 5.

For applications where the source of the tclk clock is not provided per the recommendations detailed in Figures 4 and 5, the clocks connected to the tclk_[0-3] ports may not be created from a DLL to PLL combination or a PLL to PLL combination as these combinations may create short term frequency deviations that will overflow or underflow the FPGA interface phase compensation FIFOs. In these applications, it is recommended that the FPGA interface

phase compensation FIFOs interface be implemented as detailed in Figures 4 and 5 followed by a larger FPGA FIFO that will better tolerate any short term frequency deviations created by an analog component to analog component combination.

An example of a clocking scheme that violates this requirement is providing a SERDES recovered clock as input to a PLL with the output of the PLL connected to a tclk_[0-3] port. Since the SERDES recovered clock is created by a SERDES PLL, this combination is considered a PLL to PLL combination.

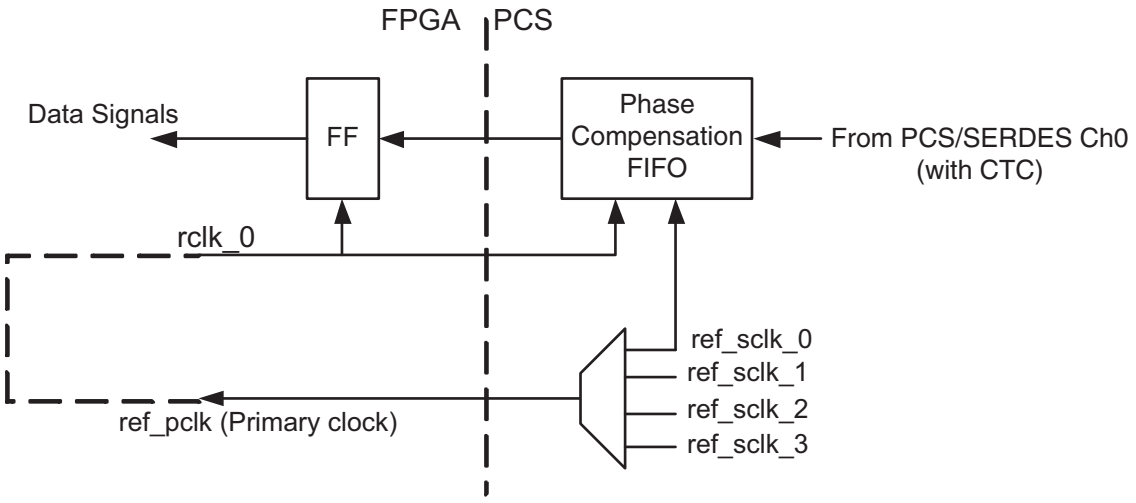
Receive Clocking

The receive direction clocking recommendation is different for certain modes of the flexiPCS.

Receive Clocking with the CTC

Figure 6 provides the recommended clocking diagram for the receive interface of a single channel when using the clock tolerance compensation (CTC) block.

Figure 6. flexiPCS Interface Receive Clocking for a Single Channel with CTC

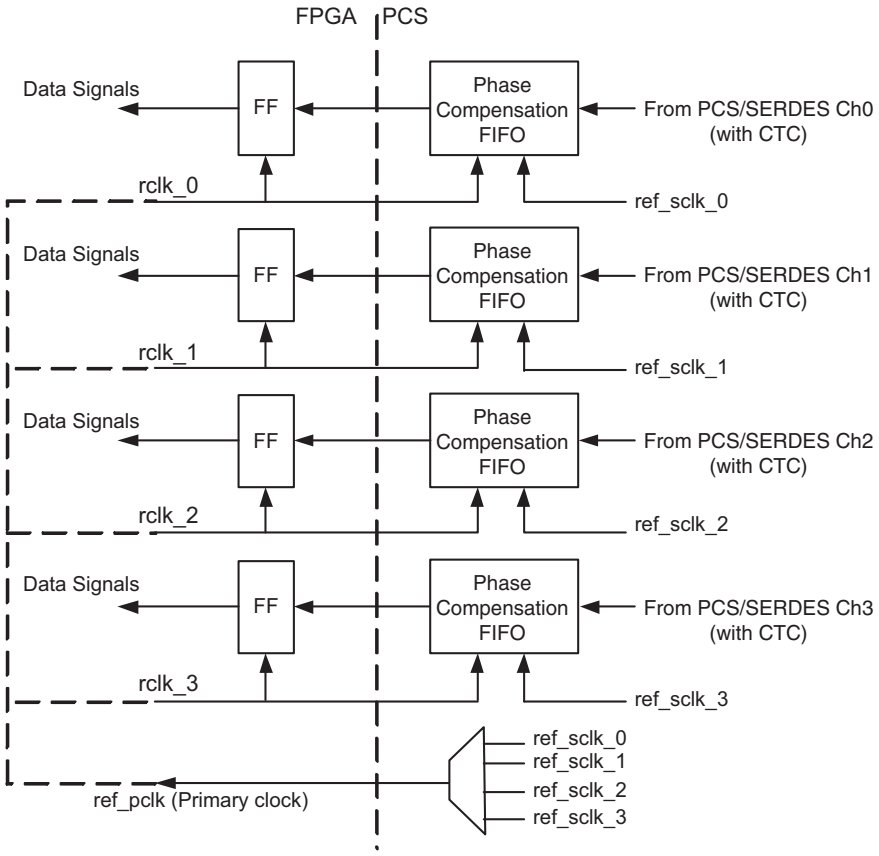


The CTC block is written to using the recovered clock derived from received data. The CTC block is read using the local reference clock which is the transmit clock. Since the CTC is read using the transmit clock the receive interface is the exact same as the transmit interface when using the CTC.

Note: The same ref_pclk signal is used for the transmit (tclk_*) and receive (rclk_*) clocks.

Figure 7 provides the recommended clocking diagram for the receive interface for all four channels when using the CTC block.

Figure 7. flexiPCS Interface Receive Clocking for All Four Channels with CTC



Just like the single channel, the CTC block is written to using the recovered clock derived from received data. The CTC block is read using the local reference clock. Therefore the receive interface is the exact same as the transmit interface when using the CTC. All four transmit clocks are the same rate so the same clock can be used for all four channels simplifying the clocking.

Note: If a mixture of half-rate and full-rate channels are used in a quad, then the secondary clock must be used for the Tx and Rx clocks per channel.

Receive Clocking with the MCA

If using the Multi-Channel Aligner (MCA) and not the CTC, the receive clocking will now remain in the recovered clock domain from the data. When using multi-channel alignment, all channels of the alignment group must run at the same rate. In this case any of the group channel's recovered clock can be used to read data from the MCA. Figure 8 provides a clocking diagram when using the MCA (and not the CTC).

Figure 8. flexiPCS Interface Receive Clocking Diagram with MCA

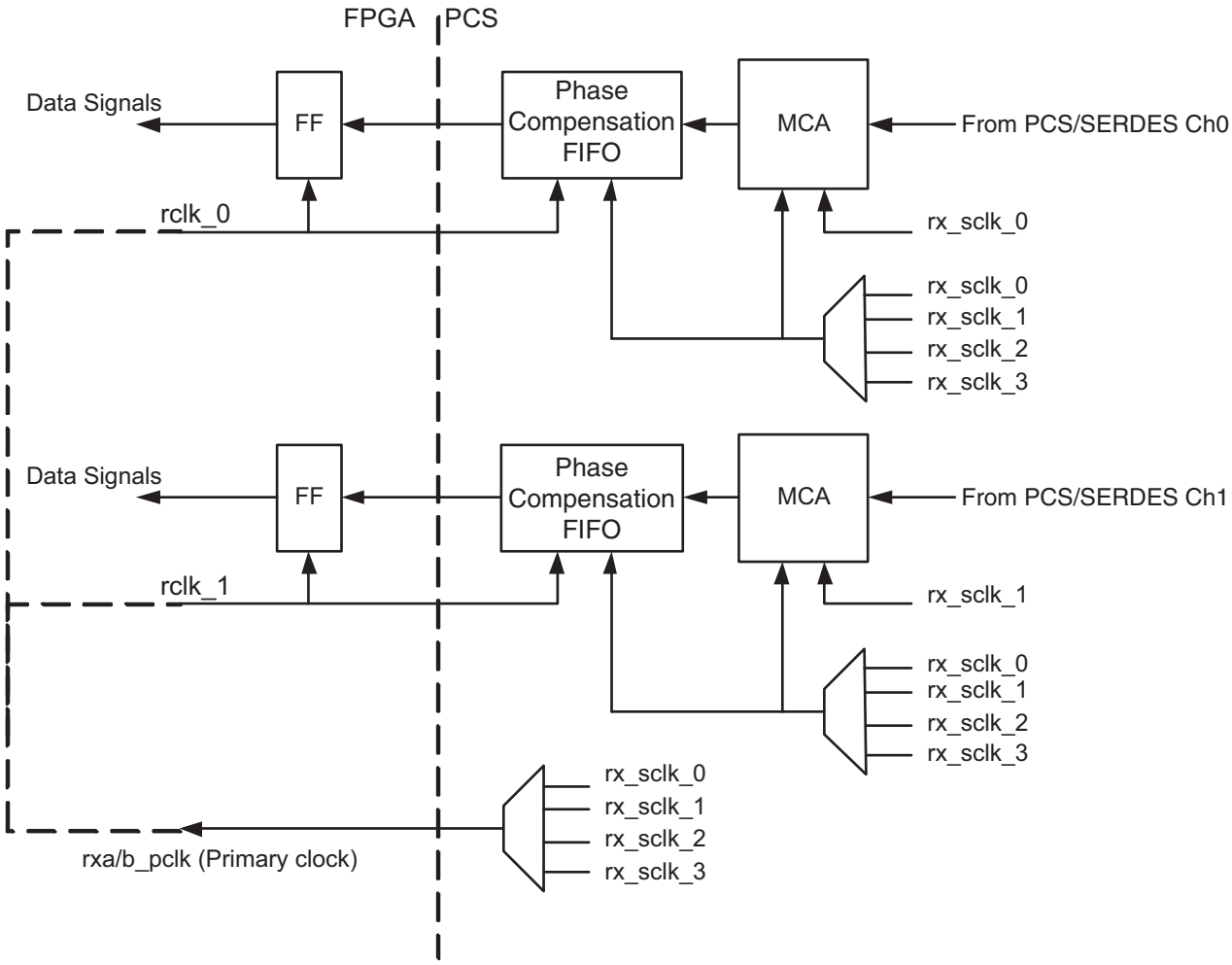


Figure 8 shows a clocking example using two channels. The same clocking interface is used for three or four channel alignment. There are two primary clock outputs for the recovered clock, rxa_pclk and rxb_pclk. So each quad can support two alignment groups and still use primary clocks. This allows a quad with all four channels aligned, three channels aligned and one not aligned, or two groups of two channels aligned.

There is a separate mux selection for the source of rxa_pclk and rxb_pclk. Each clock output can be driven by any of the quads recovered clocks. Likewise for the MCA there is a mux to control the read clock of the MCA. It is important to make sure that the rxa/b_pclk is driven by the same recovered clock as the MCA read clock for the alignment group. For example, if rx_sclk_0 is selected for the alignment group of channel0 and channel1, then rx_sclk_0 should be selected for the rxa/b_pclk which clocks the interface for this channel grouping.

Note: In this example, all channels in an alignment group must use the same rate (full-rate or half-rate), therefore individual clocks cannot be used.

Receive Clocking without CTC and MCA

If not using the MCA or CTC the receive data will be clocked using the recovered clock per channel. Again, each quad supports two primary output clocks for the recovered clock, rxa_pclk and rxb_pclk. If up to two channels are used then the primary clocks are recommended. If more than two channels are used then the user will be required to use secondary clocks for some channels.

Figure 9 shows a clocking diagram using the primary output recovered clock.

Figure 9. flexiPCS Interface Receive Clocking Using Primary Output Recovered Clock

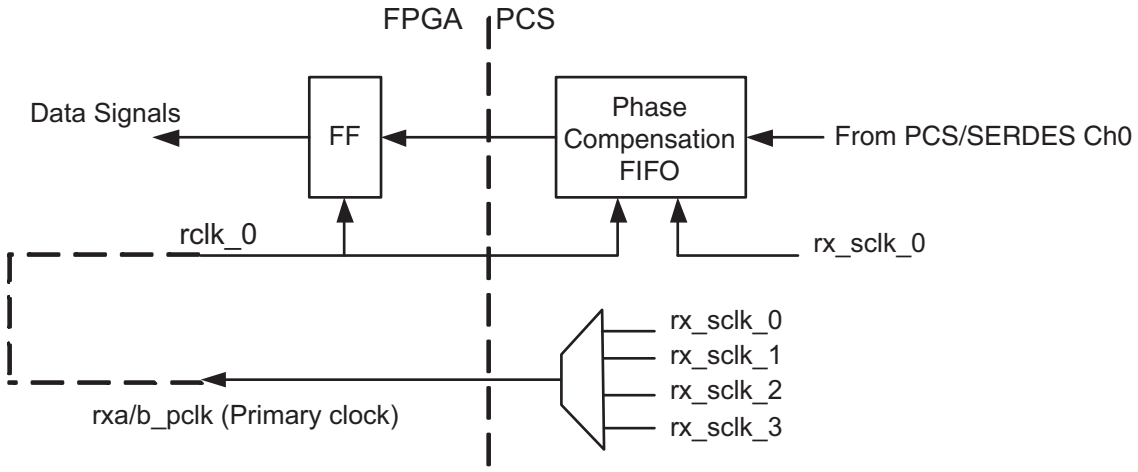
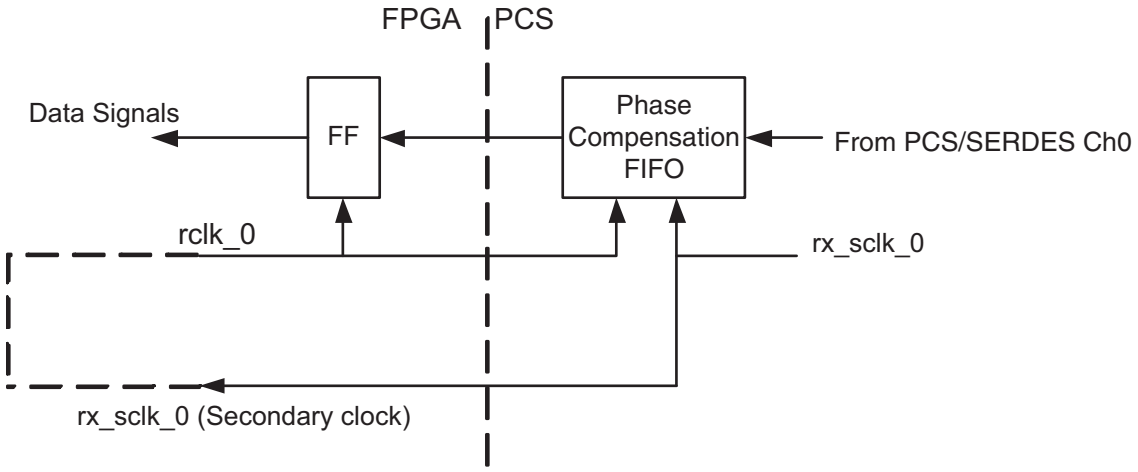


Figure 10. flexiPCS Interface Receive Clocking Using Secondary Output Recovered Clock



Recovered Clock Source Requirements

The FPGA interface phase compensation FIFOs are intended only to provide a clean timing interface to the PCS logic and cannot be used to arbitrate between different clock frequencies. Therefore the rclk_[0-3] ports must be connected to clocks that are frequency locked to the reference clock. Typically, they are fed from the clock outputs from the PCS to the FPGA as detailed in Figures 6, 7, 8 and 9.

For applications where the source of the rclk clock is not provided per the recommendations detailed in Figures 6, 7, 8 and 9, the clocks connected to the rclk_[0-3] ports may not be created from a DLL to PLL combination or a PLL to PLL combination as these combinations may create short term frequency deviations that will overflow or underflow the FPGA interface phase compensation FIFOs. In these applications, it is recommended that the FPGA interface phase compensation FIFOs interface be implemented as detailed in Figures 6, 7, 8 and 9 followed by a larger FPGA FIFO that will better tolerate any short term frequency deviations created by an analog component to analog component combination.

An example of a clocking scheme that violates this requirement is providing a SERDES recovered clock as input to a PLL with the output of the PLL connected to an rclk_[0-3] port. Since the SERDES recovered clock is created by a SERDES PLL, this combination is considered a PLL to PLL combination.

Resets

There are five resets available on the flexiPCS/SERDES. Each of the resets are responsible for resetting a different portion of the flexiPCS/SERDES. Each of these resets can be used either through the port of the flexiPCS or through a memory map register bit.

- **quad_rst** - This active-high async reset is responsible for resetting the entire flexiPCS/SERDES including the memory map and PLL. It is not recommended that this reset be used since it will reset the memory map leaving the flexiPCS/SERDES in an unprogrammed state (clearing the autoconfig initializations).
- **serdes_rst** - This active-high async reset is responsible for resetting the SERDES block including the PLL. It is recommended that this reset is held active until the reference clock to the flexiPCS/SERDES is stable. This reset should also be held active while any SERDES settings (other than full-rate to half-rate) are changed. The default auto configuration files will hold the reset active during device provisioning. serdes_rst should not be tied to the global FPGA user reset for normal functional operation.
- **rx_rst_[0:3]** - This active-high async reset is available per channel and is responsible for resetting the receive data path including the word aligner, state machines, MCA, and CTC.
- **tx_rst_[0:3]** - This active-high async reset is available per channel and is responsible for resetting the transmit data path including all state machines and serializers.
- **mca_resync_[01:23]** - This active-high async reset is responsible for resetting the MCA and forcing a new alignment. If using the MCA, this reset should be toggled after the de-assertion of the rx_rst_[0:3].

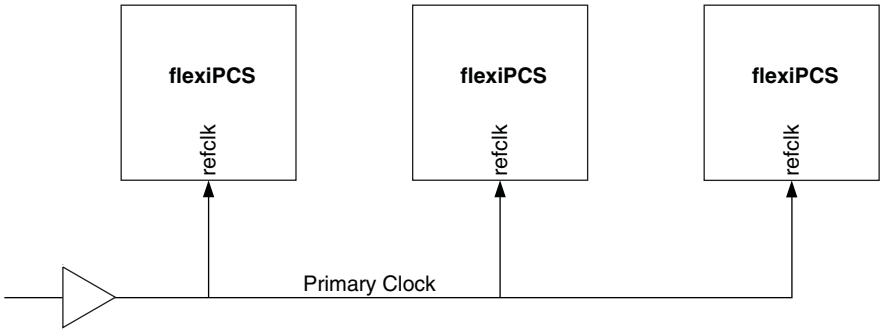
Multi-Channel Transmit Alignment

The flexiPCS has the ability to align all of the transmit SERDES channels in the device to a tolerance of 200ps across the device. Applications such as SFI-5 have tight transmit alignment tolerance requirements that can benefit from this capability. In order to achieve this level of transmit alignment there are a few details for the user to understand and implement relative to the transmit reference clock and the transmit reset.

Low Skew Reference Clock

In order to align the transmit SERDES channels, the high-speed clock must be very low skew across all of the channels. Each flexiPCS quad contains its own high speed transmit PLL. The reference clock that feeds all of the PLLs must have very low skew to all of the quads in the transmit group. Using the FPGA sourced reference clock on a primary clock route will provide this very low skew path. This FPGA-sourced reference clock path is balanced across all flexiPCS quads on the device.

Figure 11. Low Skew Transmit Reference Clock Network

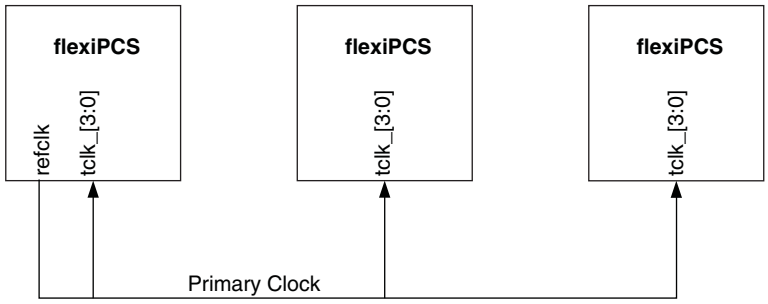


The designer can choose to use the FPGA source reference clock when creating the flexiPCS module in IPexpress. To force the usage of a primary clock, use the USE PRIMARY NET preference on the FPGA sourced reference clock net.

Low Skew Transmit FPGA Interface Phase Compensation FIFO Input Clock

In order to align the transmit data to the transmit phase compensation FIFOs, the corresponding transmit input clocks must have very low skew. Connecting all the tclk_[3:0] clock inputs to a single source primary clock route will provide this very low skew path. To force the usage of a primary source clock, the user must select one of the flexiPCS ref_pclk output clocks as the tclk_[3:0] clock source.

Figure 12. Low Skew Transmit Clock Network



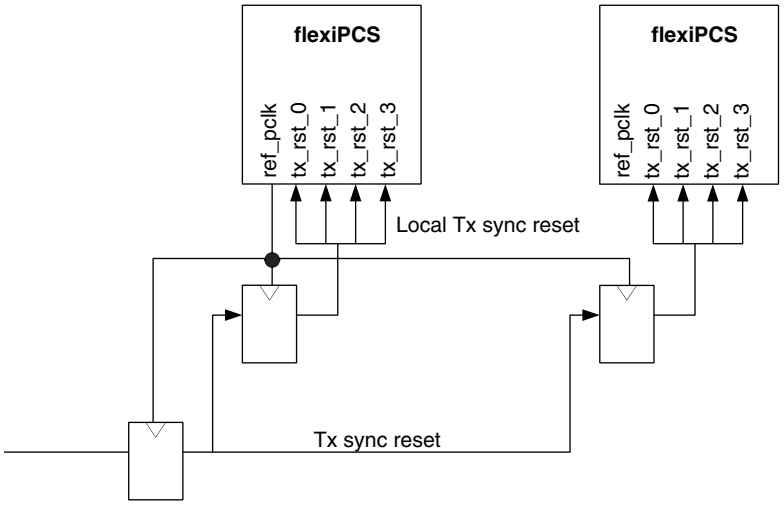
Transmit Reset Synchronization

To make sure all channels come out of transmit reset on the same clock edge there are two things the user must implement in the design. Quad registers associated with the flexiPCS blocks need to be set into a transmit synchronization mode with x4 multi-channel alignment mode specified on the associated quads. There also must be a synchronized tx_rst provided to the associated quads.

First the flexiPCS must turn on transmit synchronization by setting Quad Interface Register 0x30 bit 5 to a 1. Next the user must set the flexiPCS in x4 multi-channel alignment mode. Typically the multi-channel alignment mode is only used for the receive path. In this setup the multi-channel aligner is also used to force a transmit synchronization as well. Setting Quad Interface Register 0x19 bit 1 to a 1 will set the multi-channel aligner in x4 mode. Setting the MCA in x4 mode will not impact the receive path unless the mca_align_en ports or register controls are active. Data will simply pass through the MCA in the receive path and not be channel-aligned. The two quad registers, 0x30 and 0x19, can be set in the configuration .txt file for the flexiPCS blocks affected.

In the FPGA design all of the transmit resets for the flexiPCS need to be released with very low skew. To provide a very low skew transmit reset signal the following circuit should be implemented, where the ref_pclk uses a primary clock net.

Figure 13. TX Lane Reset Synchronization Network



By using a local flip-flop to drive each flexiPCS's transmit reset lines the resets will be balanced to the tolerance of the primary clock (~100ps across the device).

The user must wait to release the transmit reset until all of the flexiPCS transmit PLLs in the group have achieved lock. This typically takes on the order of 1ms after the release of serdes_rst.

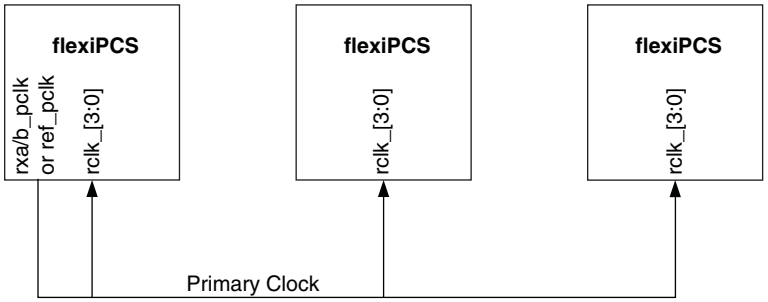
Multi-Channel Receive Alignment

As described in the Multi-Channel Alignment section of the LatticeSC/M Family flexiPCS Data Sheet, the multi-channel aligner can be used to align as many as 64 receive channels across two LatticeSC devices. This section describes how to implement clocking and receive reset Synchronization in multi-Channel alignment. Note that even though certain flexiPCS modes (10-bit and 8-bit SERDES-only modes) do not support multi-channel alignment, some of the techniques described below will minimize skew among receive data channels at the FPGA interface.

Low Skew Receive FPGA Interface Phase Compensation FIFO Input Clock

In order to align the receive data out of the receive phase compensation FIFOs, the corresponding receive input clocks must have very low skew. Connecting all the rclk_[3:0] clock inputs to a single source primary clock route will provide this very low skew path. To force the usage of a primary source clock, the user must select either rxa_pclk or rxb_pclk in non-CTC modes of the flexiPCS, or ref_pclk in CTC-modes of the flexiPCS.

Figure 14. Low Skew Receive Clock Network



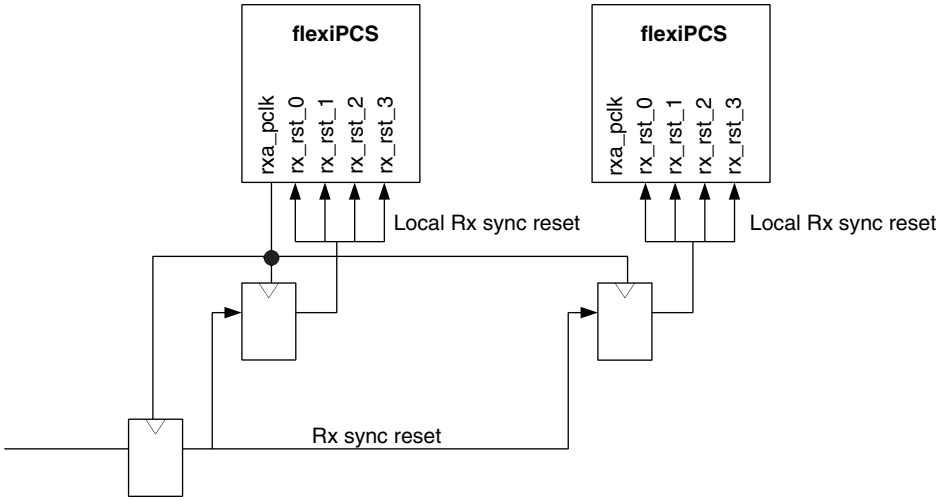
Receive Reset Synchronization

To make sure that all the receive phase compensation FIFOs come out of receive reset at zero phase, the user must set the flexiPCS in a x4 multi-channel alignment mode. Set Quad Interface Register 0x19 bit 1 to a 1 to configure the multi-channel aligner in x4 mode.

In 8-bit or 10-bit SERDES only applications (such as SFI-5), there may be a need to minimize the data skew across receive channels. Here, it is also recommended to configure the flexiPCS in x4 multi-channel alignment mode by setting Quad Interface Register 0x19 bit 1 to a 1. The MCA in x4 mode will not impact the receive path unless the mca_align_en ports or register controls are active. Data will simply pass through the MCA in the receive path and not be channel aligned. Within a quad, it is also recommended to source the read clock of each channel's multi-channel aligner to a common clock source. This ensures that a single clock reads all the multi-channel aligners' data synchronously within a flexiPCS quad. This can be achieved by configuring Quad Interface Register 0x01. For example, to configure all flexiPCS quad's multi-channel aligners read clocks to be sourced from channel 0, set Quad Interface Register 0x01 to 0x00.

In the FPGA design all of the receive resets for the flexiPCS need to be released with very low skew. To provide a very low skew receive reset signal the following circuit should be implemented.

Figure 15. RX Lane Reset Synchronization Network



By using a local flip-flop to drive each flexiPCS's receive reset lines the resets will be balanced to the tolerance of the primary clock (~100ps across the device).

Lane Resynchronization Upon Loss of CDR Lock

When multi-channel alignment is used with or without CTC in a LatticeSC flexiPCS, certain events at the RX SERDES inputs can cause a CDR loss of lock on one or more channels. Such an event may be caused by powering-up another SERDES linked to the flexiPCS RX SERDES after the LatticeSC device's bitstream has already been loaded. A CDR loss of lock event may cause the CTC and/or Receive FPGA Interface Phase Compensation FIFO pointers to mismatch across channels. The CTC and Phase Compensation FIFO logic does not automatically resynchronize when the CDR loss of lock event disappears, so FPGA logic is needed to resynchronize the CTC and receive Phase Compensation FIFO.

RX Lane Resynchronization Logic

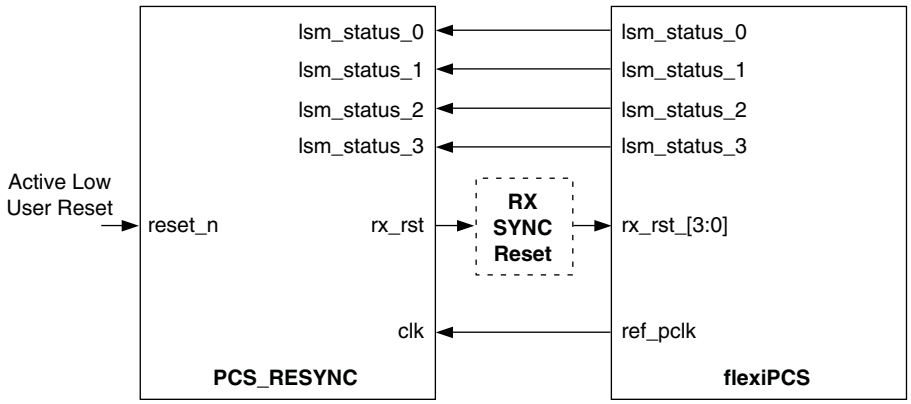
In packet mode applications, the FPGA re-synchronization logic can monitor the lsm_status_[3:0] output PCS pins and issue an RX lane reset pulse whenever any of the lsm_status_[3:0] signals go low (typically as a result of too many generic 8b10b code violations caused by CDR loss of lock), followed by all lsm_status_[3:0] going high.

In 8-bit or 10-bit SERDES only applications, the CDR receive loss of lock (rlol) interrupt bit (CIR address 0x97, bit 6) per channel can be monitored. An RX lane reset pulse should be issued whenever any channel rlol interrupt goes high, followed by all channels (within the alignment group) rlol interrupts going low.

In SONET applications, the rx_oof_[3:0] signals can be monitored instead and the re-synchronization logic should issue an RX lane reset pulse whenever any of the rx_oof_[3:0] signals goes high, followed by all rx_oof_[3:0] going low.

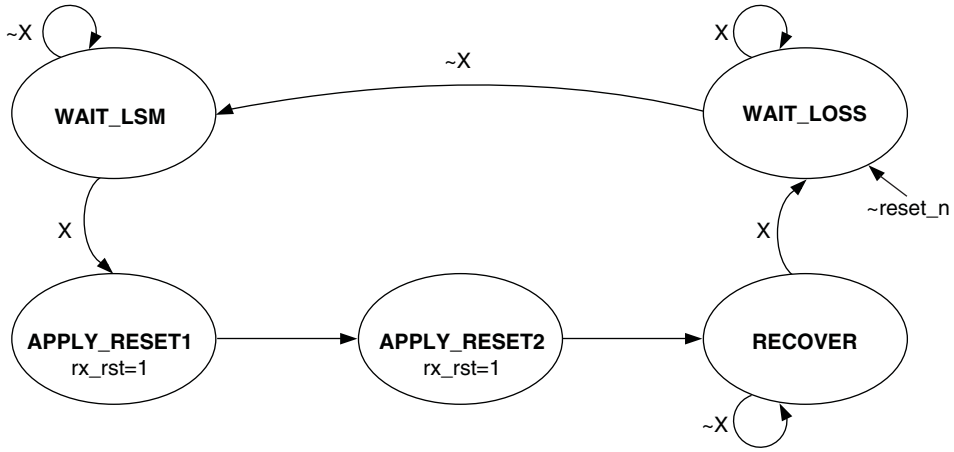
A block diagram of a typical lane re-synchronization block (PCS_RESET) connected to the flexiPCS (packet mode) is shown in Figure 16. A state diagram showing when the rx_rst output signal is asserted and a typical timing diagram follow.

Figure 16. Rx Lane Resynchronization Block Diagram



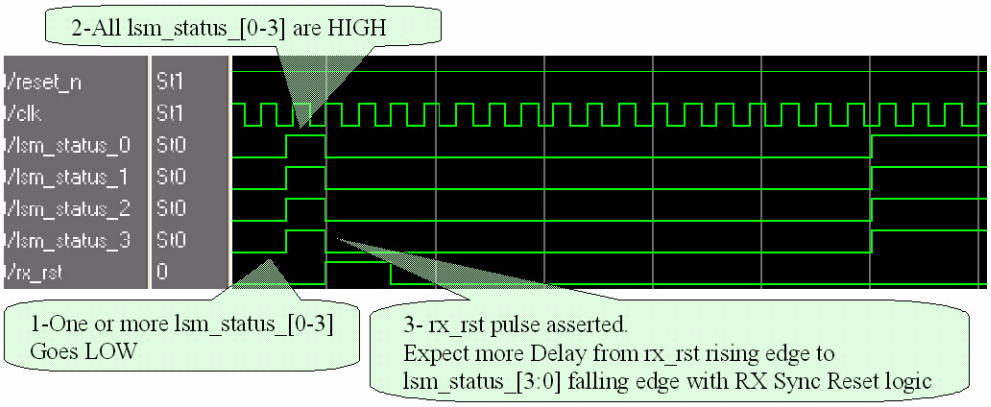
Note: Unused lsm_status inputs should be tied high.

Figure 17. Rx Lane Resynchronization State Diagram



Note: X = lsm_status_0 and lsm_status_1 and lsm_status_2 and lsm_status_3.

Figure 18. Rx Lane Resynchronization Timing Diagram



PCI Express Receiver Detection

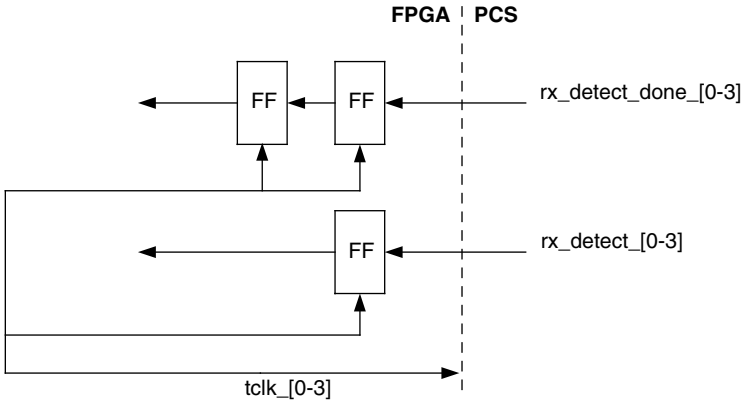
The flexiPCS supports the PCI Express defined receiver detection feature. This allows the CML output buffer to identify a termination hanging off the line. The PCI Express IP core for the LatticeSCM device provides the PCI Express receiver detection support as part of the LTSSM module. If the PCI Express IP core is not being used the

receiver detection circuit needs to be driven by user logic in the FPGA. The procedure for performing receiver detection is documented in the LatticeSC/M Family flexiPCS Data Sheet in the PCI Express section. In addition, there are two items that need to be noted.

The PCI Express receiver detection circuit will only function if the flexiPCS mode is set to be PCI Express (quad 18 bit 5). It is possible to change this bit at run time if the flexiPCS is not operating in this mode during traffic, but rather only for receiver detection.

The other important item to note is that the status signals coming back from the flexiPCS are asynchronous. In order to properly sample these signals in FPGA logic they will need to be synchronized. Figure 19 provides a simple synchronization circuit that can be used for the receiver detection status lines form the flexiPCS. Notice that the rx_detect_done signal is registered two times to guarantee enough time for the rx_detect line to stabilize.

Figure 19. PCI Express Receiver Detection Usage



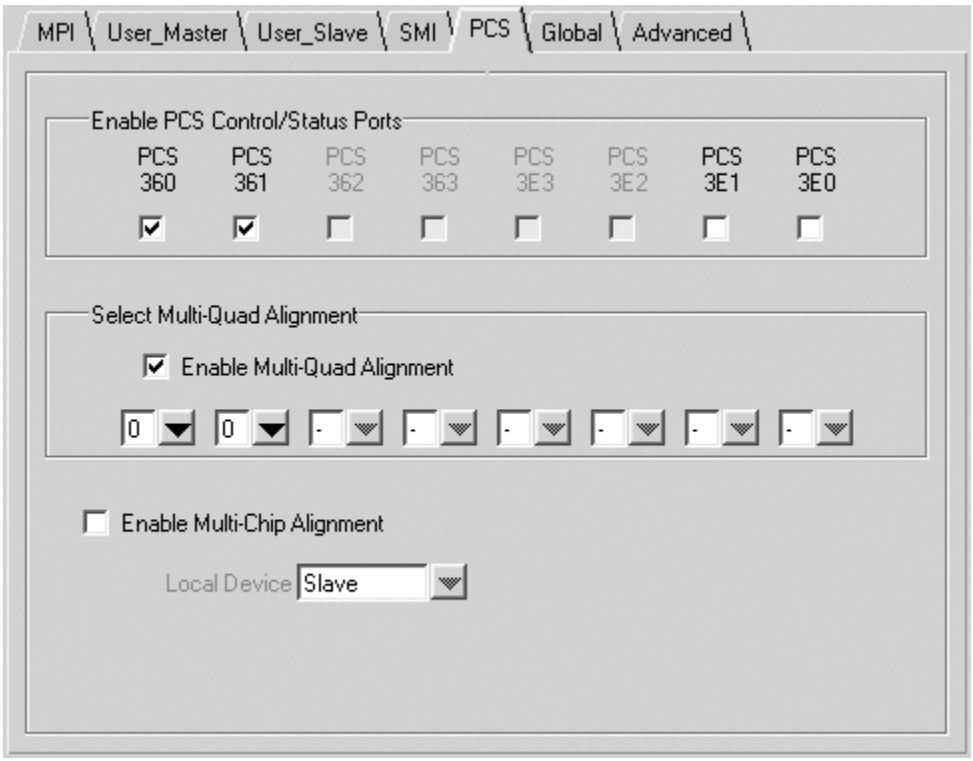
Creating a Sysbus for the flexiPCS

The System Bus is tied to all of the flexiPCS/SERDES quads with dedicated routing. If the user wants to access the memory map of the flexiPCS at run time the system bus will need to enable ports connected to the flexiPCS quads. When creating a system bus in IPexpress the user must select which physical quads on the device to enable the ports. The system bus is also responsible for passing multi-quad and multi-chip alignment information. Options for multi-quad alignment are also selected in the system bus (as well as other selections in the flexiPCS quad).

Like the flexiPCS, the system bus also creates an autoconfig file to load the embedded memory map of the system bus in simulation and bitstream generation.

In IPexpress for the system bus there is a tab labeled “PCS”. On this page the user will select the physical flexiPCS locations to enable ports on the system bus. Figure 20 provides a view of the flexiPCS tab of the system bus.

Figure 20. flexiPCS Tab of System Bus Creation in IPexpress



Only flexiPCS quads that are available on the device are allowed to be selected. Once a flexiPCS location is selected a flexiPCS specific port will exist for the particular flexiPCS quad. For example, PCS360 has dedicated ports on the system bus named pcs360_in[16:0] and pcs360_out[44:0]. Likewise the flexiPCS has ports sysbus_in[44:0] and sysbus_out[16:0] that connect to these system bus ports.

If more than one quad is selected then the user may enable multi-quad alignment options. Once enabled the user must select an alignment group for each flexiPCS quad. In the example in Figure 20 both flexiPCS quads PCS360 and PCS361 are enabled in a multi-quad alignment group 0.

The user also has the option of enabling multi-chip alignment. When using multi-chip alignment all flexiPCS quads in group 0 will be aligned with the other chip's group 0. The user must select if the current chip is a master or slave device for the alignment group.

Note: It is also possible to use the same system bus module for the master and the slave and later change the system bus autoconfig file to force one to be the master. This is easier than creating two system bus modules. Modifying the autoconfig file will be discussed later in this document.

Simulation of the flexiPCS/SERDES

The flexiPCS/SERDES simulation support is provided by a pre-compiled model for ModelSim® and NC-Verilog®. The simulation model that is pre-compiled is a behavioral model for the SERDES and the RTL of the flexiPCS. Table 1 provides the location for each of the simulation models.

Table 1. Simulation Model Locations

Simulator	Model Location
ModelSim (Lattice OEM Version)	ISPTools \ modelsim\lattice\sc\pcsa_mti_work
ModelSim	ISPTools \ ispFPGA\maco\bin\<platform>\ pcsa-mti-<version>_<platform>-V2-1.zip
NC-Verilog	ISPTools \ ispFPGA\maco\bin\<platform>\ pcsa-ncv_<version>_<platform>-V2-1.zip

Models that are distributed in .zip files need to be decompressed before the model can be used.

Autoconfig Files

Both the system bus and the flexiPCS use autoconfig files to initialize memory maps. The autoconfig files will be loaded automatically by the simulation model for both the system bus and the flexiPCS. All autoconfig files (system bus and all unique flexiPCS autoconfig files) will need to be located in the current working directory of the simulation (or added to the simulator's search path). An error message will be displayed if an autoconfig file is not found.

It is possible to edit the text based autoconfig file. Using the system bus or flexiPCS memory map as a reference the autoconfig file consists of lines that write 8-bit values into the memory map. The addressing of the autoconfig file is a relative offset from the base of the flexiPCS quad or the system bus.

System Bus Autoconfig File

The system bus uses a "irr" command which stands for Inter-quad Interface Register. The memory map for the Inter-quad Interface Registers can be found in the LatticeSC/M Family flexiPCS Data Sheet. These registers control the multi-quad and multi-chip alignment controls of the system bus.

The syntax is "irr" <address> <data value>. The address is the base of the iir registers which is located at 0x3EF00. Below is an example command what will write 0x01 to address 0x3EF0a.

```
iir 0a 01
```

flexiPCS Autoconfig File

The flexiPCS uses two commands, quad and ch#. A quad command writes to a quad based register. A ch# command writes to a channel based register. Address offsets for both Quads and Channels can be found in the LatticeSC/M Family flexiPCS Data Sheet. The syntax for a quad write is "quad <address> <data value>. The syntax for a ch# write is "chN <address> <data value> where N can be any channel 0:3. Below are example commands.

```
ch0 13 03
ch1 13 03
ch2 13 03
ch3 13 03
quad 28 50
```

The base address of a flexiPCS quad is not known until the flexiPCS is located to a particular site on the device.

Example Autoconfig File

```
# This file is used by the simulation model as well as the ispLEVER bitstream
# generation process to automatically initialize the PCS quad to the mode
# selected in the IPexpress. This file is expected to be modified by the
# end user to adjust the PCS quad to the final design requirements.
# channel_0 is in "SERDES Only(8/16-bit)" mode
# channel_1 is in "SERDES Only(8/16-bit)" mode
# channel_2 is in "Disabled" mode
# channel_3 is in "Disabled" mode

ch0 13 03  # Powerup Channel
ch1 13 03  # Powerup Channel
quad 28 40  # Reference clock multiplier
quad 29 01  # default
# quad 02 00  # ref_pclk source is ch0
# quad 19 00  # FPGA bus width is 8-bit/10-bit (default)
quad 18 60 # 8-bit SERDES Only
ch0 14 90  # 16% pre-emphasis
ch0 15 10  # +6dB equalization
ch1 14 90  # 16% pre-emphasis
ch1 15 10  # +6dB equalization
quad 30 04  # Set TX Sync Bit

# These lines must appear last in the autoconfig file.  These lines apply the correct
# reset sequence to the PCS block upon bitstream configuration
quad 41 00  # de-assert serdes_rst
quad 40 ff  # assert datapath reset for all channels
quad 41 03  # assert MCA reset
quad 41 00  # de-assert MCA reset
quad 40 00  # de-assert datapath reset for all channels
```

REFCLK Source in Simulation

When creating an external reference clock in simulation the model is expecting to see a CML type input. The clock source must be created so that the model sees both a P and N clock signal. There must be 0 delay between the P and N so using a NOT gate on one to create the other will not work.

Resets in Simulation

The flexiPCS simulation model will toggle the quad_rst before the autoconfig file is loaded. This is to reset the entire model including the memory map before the autoconfig file contents sets the specific mode. This procedure is not required on the actual silicon device.

Clocks in Simulation

The flexiPCS model will take approximately 40µs for the transmit PLL to lock to the reference clock. This is longer than in the actual silicon. For this reason all of the clocks from the PCS will not be stable for at least 40µs after the reference clock is stable and the serdes_rst is deasserted.

Unused Inputs in Simulation

All unused inputs at the FPGA/PCS interface should be tied off and all PCS/SERDES chip-level inputs should be driven from the testbench.

Implementation in ispLEVER

The following sections will discuss implementation details in ispLEVER.

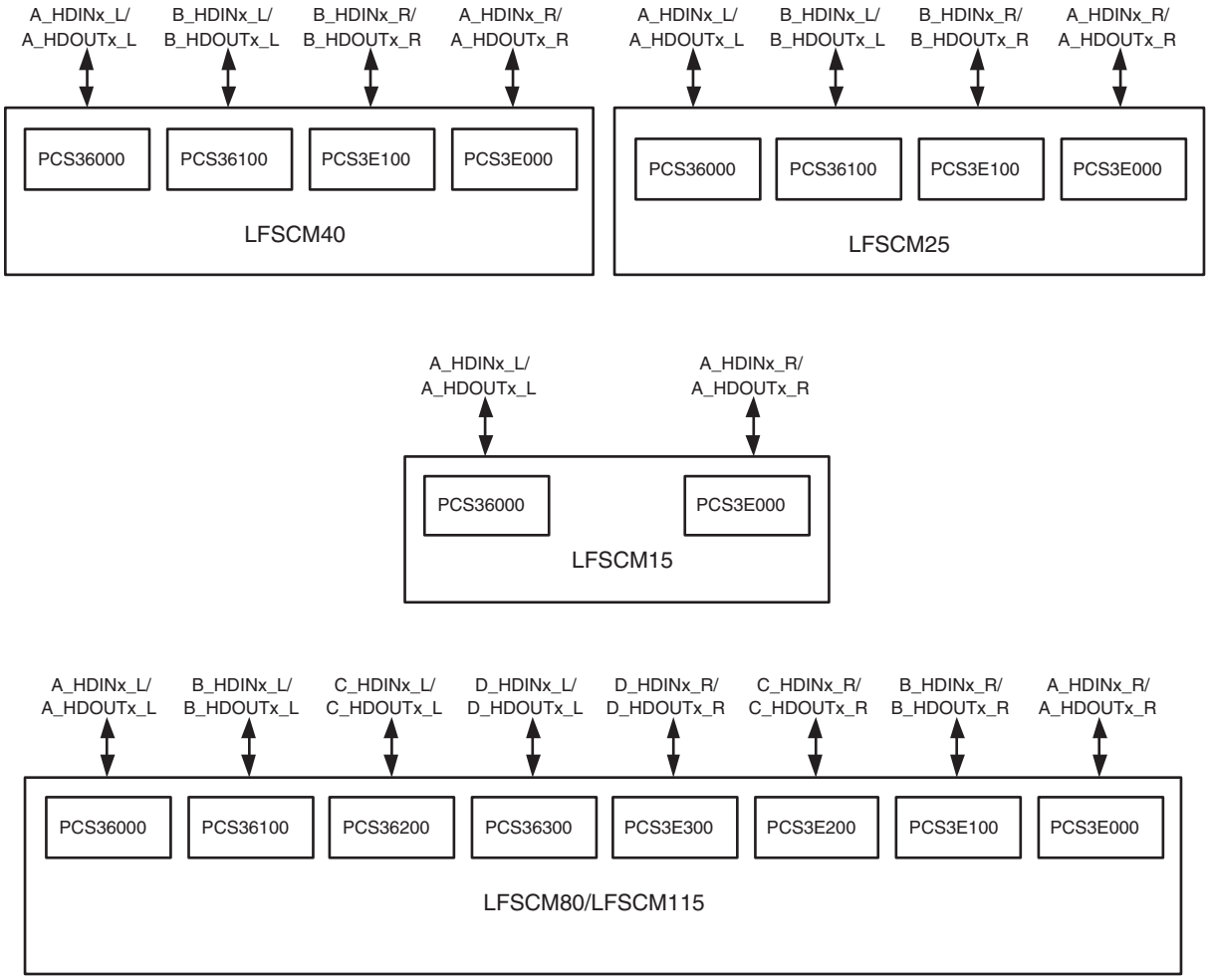
Synthesis

IPexpress will create the module named by the user as well as a PCSA.v file. The PCSA.v file needs to be included in synthesis to provide the module definition for the ports.

Selecting the flexiPCS Quad Location

There are several flexiPCS quad locations on a LatticeSC device. As the device array grows so does the number of flexiPCS quad locations. Figure 21 provides a diagram for the flexiPCS quad locations and site names for each device.

Figure 21. flexiPCS Locations per Device



The user must make note that all flexiPCS/SERDES quads are not bonded out to pins on the selected package. Verify in the package listing that the HDIN and HDOUT pins are available for the selected quad before selecting the location.

The user selects a location by using a LOCATE preference in the .lpf file. Below is an example.

```
LOCATE COMP "my_pcs/pcsa_inst" SITE "PCS36000" ;
```

If the user is using a system bus in the design the site specified in the LOCATE must match the connection made to the system bus.

There is a hardware requirement when inner flexiPCS quads are used outer quads need to be powered up. For example, if PCS3E100 was to be used then PCS3E00 will need to be powered up. There is a current source for each side. This current source runs through each quad and therefore if an inner quad is to be used then all of the outer quads leading up to the inner quad need to be powered up. This requirement is handled by the ispLEVER software. If only an inner quad is used the required outer quads will be powered up via the bitstream. Note that locating the individual SERDES I/O is not supported in the ispLEVER Design Planner. Instead, the location of the dedicated SERDES I/O is driven by the selected flexiPCS/SERDES quad.

Assigning Timing Constraints

The flexiPCS interface to the FPGA fabric is synchronous across a FIFO interface for both transmit and receive. The user supplies the clock to either read (receive) or write (transmit) to this FIFO. The entire interface is constrained by providing a FREQUENCY constraint on the interface clock. In this manner it is the same as other synchronous elements of the FPGA.

Figures 10 and 11 provide example timing report (Trace report) segments of the flexiPCS interface to the FPGA.

Figure 22. Transmit flexiPCS Interface Report

```
Passed:  The following path meets requirements by 1.292ns

 Logical Details: Cell type Pin type   Cell/ASIC name  (clock net +/-)

   Source:         FF      Q      txd_0  (from ref_pclk +)
   Destination:    PCSA    Port   pcs/pcsa_inst(ASIC)  (to ref_pclk +)

   Delay:        1.344ns  (23.7% logic, 76.3% route), 1 logic levels.

 Constraint Details:

     1.344ns physical path delay SLICE_4 to pcs/pcsa_inst meets
     3.205ns delay constraint less
    -0.187ns skew and
     0.756ns FF_TXD_0_0_SET requirement (totaling 2.636ns) by 1.292ns

 Physical Path Details:

 Name   Fanout Delay (ns)   Site     Resource
REG_DEL  ---   0.318  R14C7C.CLK to  R14C7C.Q0 SLICE_4 (from ref_pclk)
ROUTE     1    1.026  R14C7C.Q0  to  000.FF_TXD_0_0 txdZ0Z_0 (to ref_pclk)
                  --------
                    1.344   (23.7% logic, 76.3% route), 1 logic levels.

 Clock Skew Details:

 Source Clock:
          Delay              Connection
         1.653ns PCS36000.FF_SYSCLK_P1 to R14C7C.CLK

 Destination Clock :
          Delay              Connection
         1.840ns PCS36000.FF_SYSCLK_P1 to PCS36000.FF_TCLK0
```

In this example the transmit interface is using ref_pclk with a FREQUENCY preference of 312MHz.

Figure 23. Receive flexiPCS Interface Report

```
Passed:  The following path meets requirements by 1.220ns

 Logical Details:  Cell type  Pin type  Cell/ASIC name  (clock net +/-)

   Source:          PCSA       Port      pcs/pcsa_inst(ASIC)  (from rxa_pclk ?)
   Destination:     FF         Data in   dout_p_4  (to rxa_pclk +)

   Delay:               1.636ns  (36.1% logic, 63.9% route), 1 logic levels.

 Constraint Details:

       1.636ns physical path delay pcs/pcsa_inst to SLICE_2 meets
       3.205ns delay constraint less
       0.201ns skew and
       0.148ns M_SET requirement (totaling 2.856ns) by 1.220ns

 Physical Path Details:

   Name  Fanout Delay (ns)      Site              Resource
FB_RXD_0_4 ---  0.591 36000.FF_RCLK0 to 000.FB_RXD_0_4 pcs/pcsa_inst (from rxa_pclk)
ROUTE       1   1.045 000.FB_RXD_0_4 to R17C7B.M0 rxd_4 (to rxa_pclk)
                --------
                 1.636   (36.1% logic, 63.9% route), 1 logic levels.

 Clock Skew Details:

 Source Clock:
           Delay              Connection
          1.840ns PCS36000.FF_RXCLK_P1 to PCS36000.FF_RCLK0

 Destination Clock :
           Delay              Connection
          1.639ns PCS36000.FF_RXCLK_P1 to R17C7B.CLK
```

In the receive interface example the interface is constrained with a FREQUENCY preference of 312MHz using the rxa_pclk.

More Information

More information on the flexiPCS, System Bus and LatticeSC architecture can be found in the following documents.

- LatticeSC/M Family Data Sheet
- LatticeSC/M Family flexiPCS Data Sheet
- TN1085, LatticeSC MPI/System Bus

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
January 2007	01.0	Initial release.
March 2007	01.1	Added PCI Express Receiver Detection section.
October 2007	01.2	Changed the name of the PCS FIFO bridge from FIFO to phase compensation FIFO
		Added the “Transmit Clock Source Requirements” and “Recovered Clock Source Requirement” sections.
January 2008	01.3	Updated bulleted list in Creating a flexiPCS/SERDES in IPexpress text section.
		Updated screen shot for IPexpress flexiPCS Module.
July 2008	01.4	Added information in the Resets section for serdes_rst.
		Globally changed serdes_reset to serdes_rst.
		Added new sections for Multi-Channel Transmit Alignment, Multi-Channel Receive Alignment, and Lane Resynchronization Upon Loss of CDR Lock.
October 2008	01.5	Added statement about SFI-5 in Multi-Channel Transmit Alignment section.
		Added information about CDR receive loss of lock for 8-bit or 10-bit SERDES in RX Lane Resynchronization section.