LatticeSC sysCLOCK PLL/DLL User's Guide

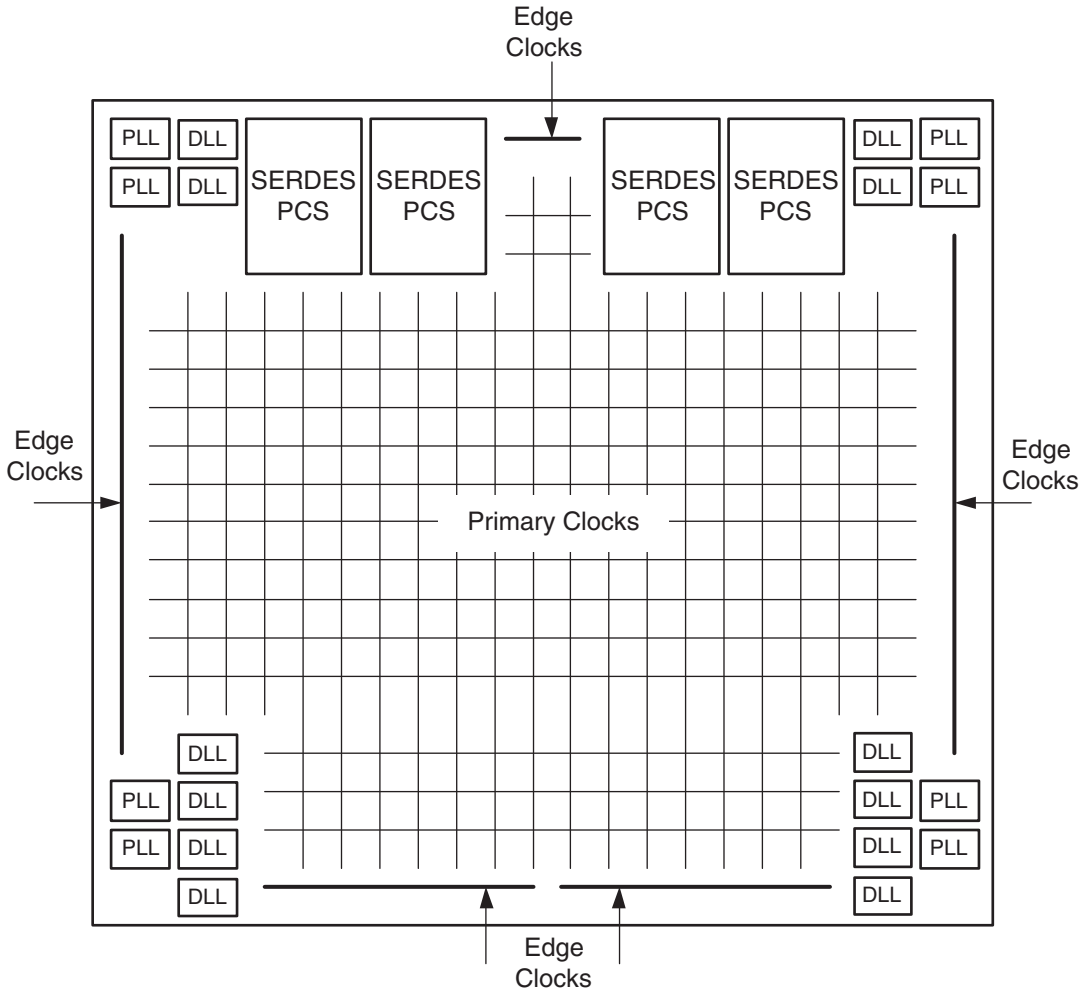
July 2012 Technical Note TN1098

Introduction

This technical note describes the clocking resources available in the LatticeSC architecture. Details are provided for primary clocks, edge clocks, and secondary clocks as well as clock elements such as PLLs, DLLs, Clock Dividers, and several other elements. Performance specifications and tolerances for the clocking elements are found in the LatticeSC/M Family Data Sheet DC and Switching Characteristics section.

Figure 1 provides a high-level view of the clocking structure of the LatticeSC architecture.

Figure 1. LatticeSC Clocking Structure (LFSC3GA25S Device)



General Overview of Clocking Architecture

Clock Networks

There are three types of clock networks in the LatticeSC architecture.

Primary Clock

A Primary clock is the main type of clock on which synchronous FPGA logic functions are performed. These clock networks are low skew global clocks that can connect to all of the synchronous elements in the array. There are up to 48 primary clocks on a LatticeSC device.

Edge Clock

Edge clocks are dedicated I/O clocks and are located on the edge of the array. Edge clocks are connected and arranged by I/O banks or pairs of I/O banks. Edge clocks provide very low skew in a particular bank(s) for creating I/O busses. There are 40 edge clocks arranged in five groups of eight on a LatticeSC device.

Secondary Clock

Secondary clocks use general purpose data routing to create a localized clock network. Secondary clocks are used when primary and edge clocks cannot be used.

sysCLOCK™ PLLs and DLLs

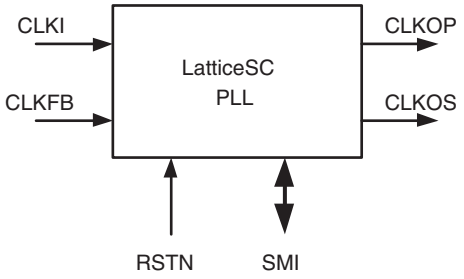
The sysCLOCK PLLs can be used in a variety of clock management applications such as clock injection removal, clock phase adjustment, clock timing adjustment, and frequency synthesis (multiplication and division of a clock). For systems where EMI is a significant factor, LatticeSC PLLs offer spread-spectrum capabilities to help minimize EMI noise. There are eight PLLs arranged in pairs in the corners of a LatticeSC device.

The DLLs are also used in a variety of applications such as clock injection removal, clock delay match and time reference delay (for 90 degree phase shifts). In some modes of operation, the output from the DLL is a clock signal. In other modes of operation, the output is a digital delay control vector (DCNTL[9:0]) that can be used to adjust an input delay element elsewhere on the device. There are 12 DLLs arranged in groups of two and four in corners of a LatticeSC device.

PLL Features

Figure 2 provides a symbolic view of the LatticeSC PLL element.

Figure 2. LatticeSC PLL



Clock Injection Delay Removal

The clock injection delay removal feature of the PLL removes the delay associated with the PLL and clock tree. This feature is typically used to reduce clock to out timing. This feature is performed by aligning the input clock with a feedback clock from the clock tree. Optional delay may also be added to the feedback path to further reduce the clock injection time.

Clock Phase Adjustment

The clock phase adjustment feature of the PLL provides the ability to set a specific phase offset between the two outputs of the PLL. Phase adjustments can be made in 45 degree increments.

Frequency Synthesis

The PLL can be used to multiply up or divide down an input clock.

Spread Spectrum

The PLL supports spread spectrum clocking to reduce peak EMI by using "down-spread" modulation. The spread spectrum operation will vary the output frequency (at 30KHz to 500KHz) in a range that is between its nominal value, down to a frequency that is a programmable 1%, 2% or 3% lower than nominal.

Note: Due to the complex tuning requirements of the PLL in spread spectrum mode, please contact Lattice Technical Support for detailed information on programming the PLL in spread spectrum mode. See the Technical Support Assistance section at the end of this document for contact information.

Additional Features

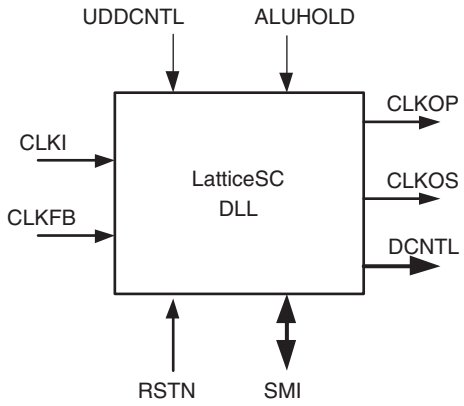
In addition to the major features, the PLL has several other options that can be used in conjunction with the major modes.

- Additional input delay on CLK1 and CLKFB
- Separate output dividers for the two outputs (divide by 1 to 64)
- Additional output delay on CLKOS
- Runtime programmable output dividers and phase offset via SMI interface
- Reset via the GSR or user reset signal

DLL Features

Figure 3 provides a symbolic view of the LatticeSC DLL element.

Figure 3. LatticeSC DLL



Clock Injection Delay Removal

The clock injection delay removal mode of the DLL removes the delay associated with the DLL and clock tree. This feature is typically used to reduce clock to out timing. This feature is performed by adding delay to the input clock to align it to the feedback clock. This delay can also be an output of the DLL on the DCNTL bus to delay other inputs the same amount.

Time Reference Delay

Time reference delay is used to create a 90 degree phase shifted signal. The DLL produces a DCNTL vector that will control a DELAY element to delay an input signal by 90 degrees. This delay will track over process, temperature and voltage changes.

Clock Delay Match

Clock delay match mode accepts two clock inputs and will produce a DCNTL vector that is the delta of the two clocks phases. This DCNTL value can then be used by an input delay element to perform a transfer between the two clocks.

Additional Features

In addition to the major modes, the DLL has several other options that can be used in conjunction with the major modes.

- Additional input delay on CK1 and CLKFB
- Output divider on one of two outputs (divide by 1, 2, or 4)
- Additional phase adjustment on CLKOP and CLKOS
- Runtime programmable output dividers and phase offset via SMI interface
- Reset via the GSR or user reset signal

PLL vs. DLL

The LatticeSC architecture provides both PLLs and DLLs. There are some features that differentiate the PLL from the DLL. Below is a list identifying distinguishing features.

- A PLL can multiply/divide a clock, whereas the DLL can only divide a clock
- A DLL can propagate its functionality to several elements using the delay control vector while the PLL cannot
- The PLL has a finer granularity of divider options than the DLL
- The DLL can more accurately and precisely follow input jitter compared to the PLL
- The PLL has better jitter filtering and stability
- The PLL has better phase accuracy for phase settings
- The DLL can accept a clock input that stops whereas the PLL cannot
- The PLL provides spread spectrum capability
- The DLL can handle non-periodic signals while the PLL cannot

In general, PLLs are better for clocking signals off-chip due to their improved jitter characteristics while DLLs may be better suited for clocks that capture input signals.

Overview of Other Clocking Elements

Clock Dividers (CLKDIV)

Clock dividers are provided to create phase-matched divided-down clocks for divide by 2 or 4. Clock dividers are especially useful for creating the low speed clock used with the I/O Mux/DeMux gearing logic. When using I/O Mux/DeMux gearing logic, these clock dividers also provide a synchronization reset. There are 20 clock dividers on a LatticeSC device.

Dynamic Clock Select (DCS)

A dynamic clock select provides a glitchless switch between two clock sources to a primary clock. This clock multiplexor allows the gating of a clock signal without leaving dedicated clock resources in the device. There are eight dynamic clock select blocks on a LatticeSC device.

Clock Shut Off

A clock shut off element is provided to stop an input signal at the input buffer. It is useful for the DQS in memory interfaces. This element allows clock enable logic to be performed on the edge clock without connecting to a logic element in the array. It is also useful to shut off any or all input signals to the FPGA. This capability is present in every PIO input to the device.

Edge Detection

A simple clock edge detection element is provided directly in the I/O logic. This feature is available on every PIO input. It creates a one-shot output that goes high on the first positive edge after a reset.

Oscillator

An internal oscillator is provided. This is the same oscillator that is used for the master configuration modes when the FPGA sources the configuration clock. This programmable rate oscillator is made available post-configuration for the FPGA design. There is only one oscillator on a LatticeSC device.

Detailed Information for LatticeSC Clocking Components

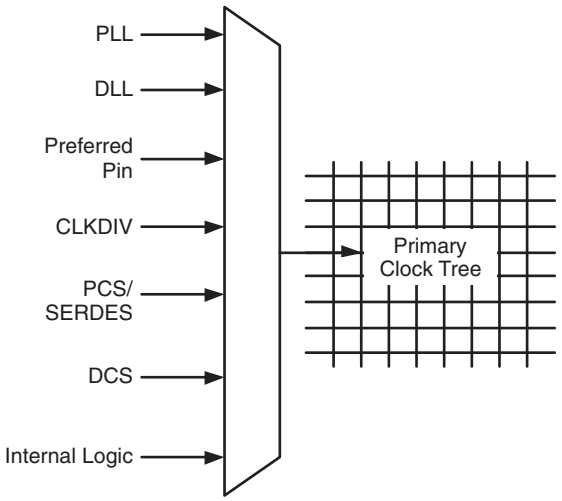
Primary Clock

The primary clock network is the most frequently used clock network of the architecture. This is a low skew clock network that has connectivity to nearly every synchronous element of the device.

The LatticeSC device is divided into four quadrants with each quadrant providing 12 primary clocks for a total of 48 primary clocks on the device. Primary clocks can also cross quadrants in which case the primary clock would be shared by more than a single quadrant. This arrangement allows for 12 primary clocks that cover the entire device not taking into account the quadrant structure.

Primary clocks can be sourced from a variety of FPGA elements as shown in Figure 4. Recovered and reference clocks from the SERDES can drive primary and secondary clocks.

Figure 4. Primary Clock Sources



The ispLEVER® place and route tools will automatically select a clock net as a primary clock resource based on the following rules. The selection rules are ordered by priority from highest to lowest.

1. Device connectivity – if a net in the design is sourced from a port that only drives a primary clock then a primary clock will be used. This is true in the case of the clock dividers which can only drive primary clocks. Also, the SERDES/PCS block provides specific clock outputs which can only drive primary clocks.
2. The user can select a net in the design to use a primary clock resource. This can be done in the pre-map Preference Editor using Project Navigator. The preference can also be entered into the ASCII text preference file using the following syntax.

```
USE PRIMARY NET "<net name>";
```

This preference is only applicable if the source is capable of driving a primary clock. This preference is useful in directing the place and route tools to select a primary clock on a clock that does not have a large number of clock loads.

3. The place and route tool will select clocks with the highest number of clock loads until all primary clock resources are used.

The ispLEVER place and route report indicates which clock signals have been selected to use a primary clock resource.

```
The following four signals are selected as primary clocks:
   txclk (driver: oddr/ucdiv, clk load #: 191)
    rxclk (driver: iddr/ucdiv, clk load #: 184)
    clka (driver: clka, clk load #: 34)
    clkb (driver: clkb, clk load #: 28)
```

Quadrant Clocking

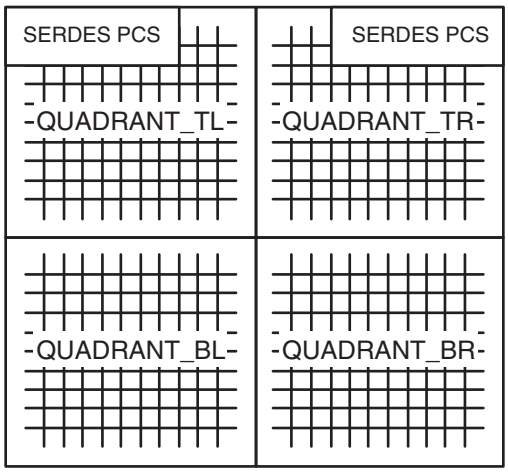
The primary clock network is divided into four quadrants. Each quadrant provides 12 primary clocks for a total of 48 possible primary clocks. Quadrants can be connected and share a primary clock. The user can partition a design into quadrant clocking by assigning a preference to the clock. This preference can be set in either the pre-map preference editor or the preference file. The place and route tool will use the user preference and place logic elements only in the specific quadrant(s) specified. Below is an example of the preference.

```
USE PRIMARY net "clka" QUADRANT_TL;
```

If a quadrant is not specified, the logic will use the entire device and automatically share primary clocks across quadrants. In this case there are only 12 primary clocks in the device.

Figure 5 provides the location and name of each quadrant.

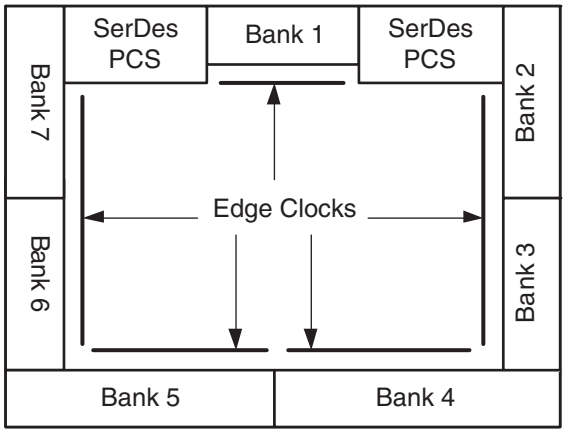
Figure 5. Primary Clock.Quadrants



Edge Clock

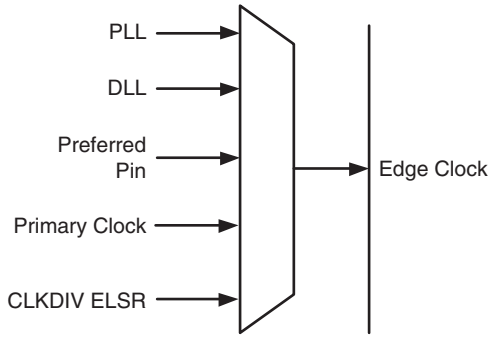
The edge clock network is arranged around the perimeter of the device, connecting to the I/O logic. Edge clocks are localized low skew clocks intended for clocking high speed data in and out of the I/O logic. Edge clocks are tied to the bank based I/O structure as shown in Figure 6.

Figure 6. LatticeSC Edge Clock Banking



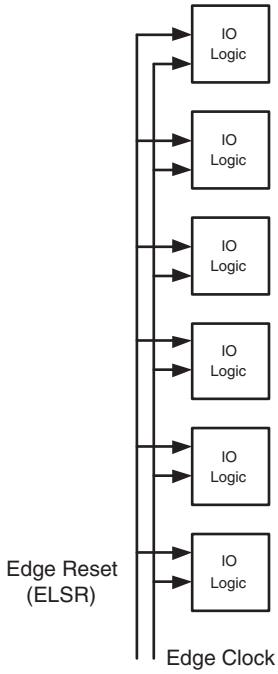
Edge clocks in banks, 1, 4, and 5 provide low skew connectivity to elements in their bank. There are eight edge clocks available in each of these banks. Edge clocks in banks 2/3 and 6/7 span the entire side of the device. There are eight edge clocks that are shared in banks 2/3 and another eight in banks 6/7. For wide I/O data buses it is best to use a low skew clock on the outside of the device. The edge clocks on the bottom and the side can be driven by the same source from a PLL/DLL. Primary clocks can also drive edge clocks as shown in Figure 7.

Figure 7. Edge Clock Sources



Edge clock routes can also be used for high speed synchronous resets for the I/O logic. The Mux/DeMux gearing of the I/O logic requires a synchronous reset to synchronize the entire bus. An edge clock local set/reset (ELSR) can be used to route this reset with its very low skew as shown in Figure 8. This allows all of the I/O logic blocks to be reset in the same high speed clock cycle.

Figure 8. Edge Set/Reset (ELSR)



Edge clocks can also be used for high speed clock inputs to be divided down by either a PLL, DLL, or CLKDIV. For example, an edge clock could be used to source a 1GHz clock to a PLL to divide down the clock rate for the FPGA design.

The ispLEVER place and route tools will automatically select a clock net as an edge clock resource based on the following rules. The rules are ordered by priority from highest to lowest.

1. Device connectivity – if it is possible for an edge clock to be used, the place and route tools will use an edge clock. This relies on three factors.

   a. The source of the clock must be capable of driving an edge clock.
   b. **All** destinations of the clock must be capable of receiving an edge clock.
   c. There are edge clock resources available in the bank structure.

2. The user preference "USE EDGE NET "<net name>";" allows the user to force a net to be implemented using an edge clock. This will only work if the first rule of device connectivity is fulfilled.

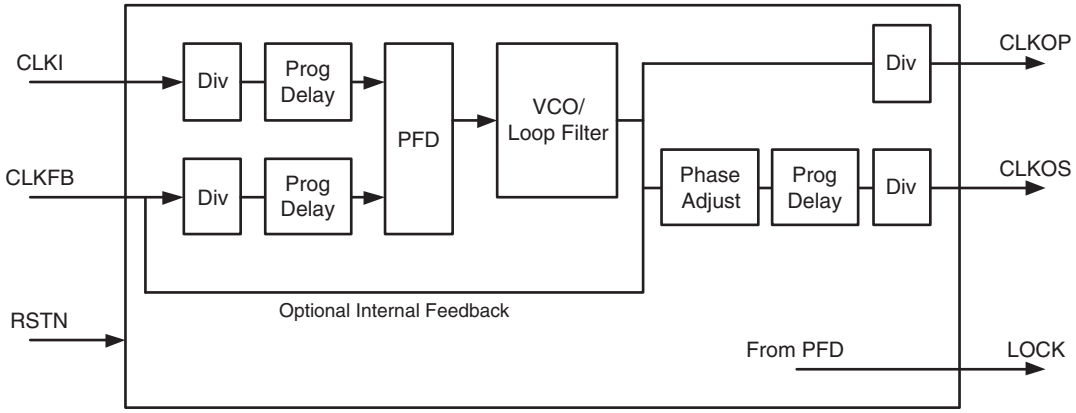
Secondary Clock

Secondary clocks are created using a set of extendable horizontal spines and vertical branches which are created automatically by ispLEVER. These clocks are used when primary and edge clocks cannot be used. Secondary clocks are buffered every six rows or six columns of the array and do not have the same skew control as the edge or primary clock network.

PLL

The LatticeSC PLL provides features such as clock injection delay removal, frequency synthesis, phase adjustment, and spread spectrum capabilities. At the center of the PLL is a voltage controlled oscillator (VCO), phase/frequency detector (PFD) to compare its two inputs, CLKI and CLKFB, and a loop filter. At each input and output there are dividers to control the VCO and output rates. A block diagram for the PLL is shown in Figure 9.

Figure 9. LatticeSC PLL Block Diagram



The LatticeSC PLL allows the user to select an input frequency and output frequencies for its two outputs, CLKOP and CLKOS. Dividers provide the ability to synthesize different rates based on the input frequency. A mixture of phase control and programmable delays provide the ability to determine the timing relationships of the clock signals.

Ports and Descriptions

The LatticeSC PLL is created and configured by IPexpress™ (included in the ispLEVER design tools). The following is a list of user port names and descriptions for the PLL. IPexpress will wrap the element shown in Figure 9 to create a customized PLL module based on user selections.

CLKI Input

The CLKI signal is the reference clock for the PLL. The CLKI input can be sourced from any type of FPGA routing and pin. The PLL CLKI input has a preferred pin per PLL which will be discussed later in this document. The preferred pin provides the lowest latency and best case performance. If the input clock CLKI stops, then the PLL will drift to a frequency in the kHz range.

CLKFB Input

The CLKFB input is only available if the user selects to use a user clock signal for the feedback. If internal feedback or CLKOS/CLKOP is used for the feedback, this connection will be made inside the module.

The PLL will align the input clock phase with the feedback clock phase to remove clock injection delay. The CLKFB input also supports a divider for use in frequency synthesis. The PLL CLKFB input has a preferred pin per PLL which will be discussed later in this document. The preferred pin provides the lowest latency and best case performance.

If internal feedback is selected, the PLL feedback is connected internally directly from the VCO. If the feedback input clock CLKFB stops, the PLL drifts to a frequency at the high range of the PLL operating frequency. A reset is required to recover from this state.

CLKOP Output

An output of the PLL based on the VCO rate which can be divided. The CLKOP output can drive primary, edge, and secondary clock routing.

CLKOS Output

An output of the PLL based on the VCO rate which can be divided and/or phase shifted. The CLKOS output can drive primary, edge, and secondary clock routing.

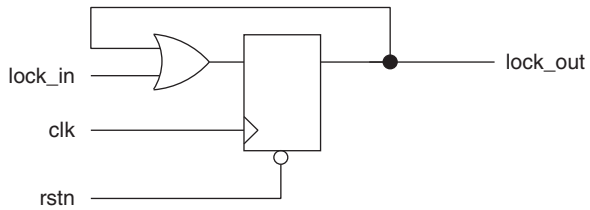
LOCK Output

Once the PLL becomes frequency locked, the PFD tries to match the phase of the CLKI input clock to the CLKFB feedback clock. The clock signals to the PFD are used to determine if they are close enough in phase for the PLL to be considered locked by the lock detect circuitry coming from the PLL loop. When this occurs the PLL LOCK signal will go high as an indicator. If the CLKI and CLKFB input stops, the LOCK output may go low and the VCO will free-run and drift. At least one edge for CLKI must occur without CLKFB in order for the LOCK signal to go low.

Depending on certain input conditions of the clock frequency, the relative input clock edge and PLL update edge relationships may cause the digital detect logic of the PLL LOCK signal to transition low for several clock cycles. While the digitally produced lock signal may transition low for several clock cycles, the analog core of the PLL maintains operation within all of its specifications. This LOCK signal transition may indicate a false out-of-lock occurrence if not properly filtered by the user design.

The described false out-of-lock signal is due to a synchronization state within the digital lock detect circuit itself and is not indicative of a true, analog out-of-lock condition of the PLL. To compensate for this potential false indication, it is recommended that the designer implement a simple circuit which sets the PLL LOCK output high upon lock and keeps it high until a global reset is issued. An example circuit is shown in Figure 10. A caveat with this solution is in the unlikely event that the clock source or PLL falls out of lock or is disabled for any reason, the example circuit shown in Figure 10 will still indicate that the PLL is locked which may not necessarily be the case.

Figure 10. PLL LOCK Circuit



RSTN Input

Active low reset input to reset the VCO and divider logic to operate at a low operating frequency. The PLL can optionally be reset by the GSR as well. It is recommended that if the PLL requires a reset that the reset is not the same as the FPGA logic reset. Typically logic requires that a clock is running during a reset condition. If the data path reset also resets the PLL the source of the logic clock will stop and this may cause problems in the logic.

SMI (Serial Management Interface)

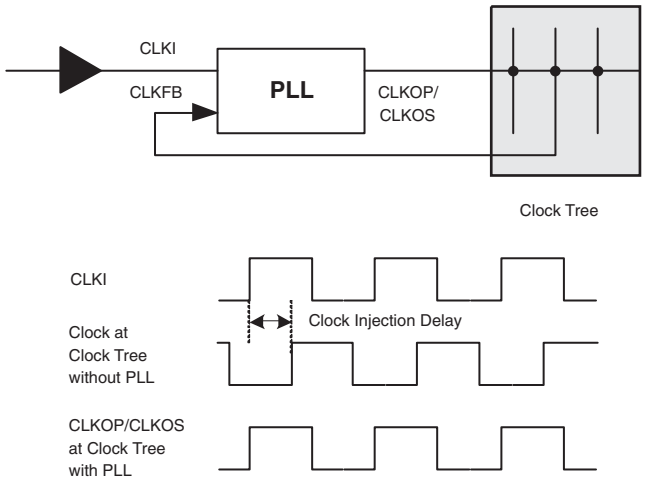
The PLL supports a run time control interface for modifying the behavior of the PLL in the system. Such parameters as output dividers and phase offset can be changed while in the system without requiring a reconfiguration. This control interface is known as the Serial Management Interface (SMI). More information on using the SMI will be found later in this document.

PLL Modes of Operation

PLL Clock Injection Removal

The PLL can be used to reduce clock injection delay. Clock injection delay is the delay from the input pin of the device to a destination element such as a flip-flop. The PLL will align the CLKI with the CLKFB. If the CLKFB signal comes from the clock tree (CLKOP, CLKOS) then the delay of the PLL and the clock tree will be removed from the overall clock path. Figure 11 shows a circuit example and waveform.

Figure 11. Clock Injection Delay Removal via PLL



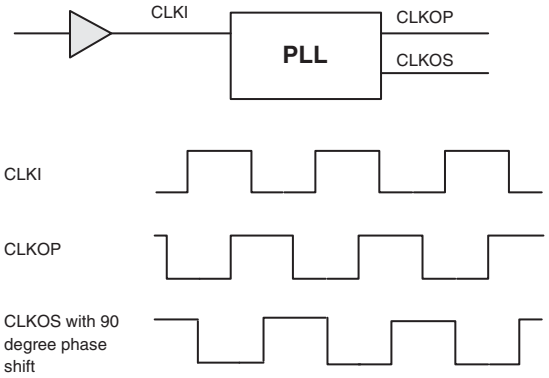
Removing clock injection delay is beneficial for decreasing clock-to-out timing of the device.

PLL Clock Phase Adjustment

The PLL can create fixed phase relationships in 45 degree increments. Creating fixed phase relationships are useful for forwarded clock interfaces where a specific relationship between clock and data is required. There are two configurations for phase adjustments, CLKOP to CLKOS and CLKI to CLKOS.

CLKOP to CLKOS: Figure 12 shows a circuit example and waveform of a clock phase adjustment between CLKOP and CLKOS using internal feedback.

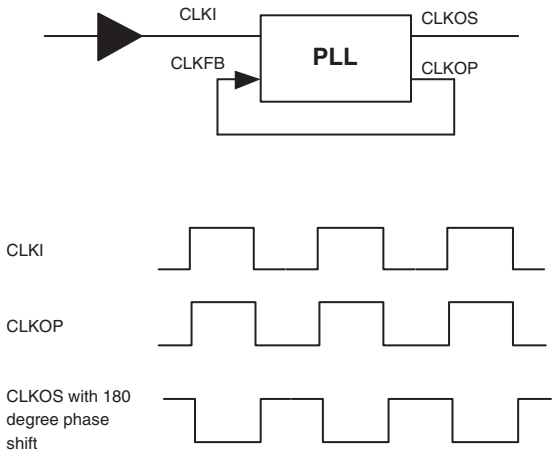
Figure 12. Clock Phase Adjustment PLL CLKOP leads CLKOS



In this PLL configuration CLKOP will lead CLKOS by the determined amount of phase adjustment. CLKOP can be used as the feedback clock in this configuration, but not CLKOS. If CLKOS is used as the feedback clock the phase adjustment will be removed by the nature of the PLL.

CLKI to CLKOS: Figure 13 shows a circuit example and waveform of a clock phase adjustment between CLKI and CLKOS.

Figure 13. Clock Phase Adjustment PLL CLKI leads CLKOS



In this PLL configuration, CLKI will lead CLKOS by the determined amount of phase adjustment. CLKOP is required to be the feedback clock connected to CLKFB. This feedback path will align the outputs to the CLKI input if no phase adjustment is specified. If phase adjustment is added the CLKOS output will be delayed by this amount.

Additional Delay Capabilities

In addition to the phase adjustment capability of the PLL there are two other methods for controlling the clock edges of the PLL outputs.

Fixed Delays: The CLKI, CLKFB, and CLKOS ports support a fixed delay capability. These delays can be added to further shift the clock edges in time. CLKI and CLKFB provide coarse (0-3ns) and fine (0-700ps) delays. CLKOS provides only fine delay. Adding delay to CLKI shifts the entire PLL out in time affecting both CLKOP and CLKOS. Adding delay to CLKFB shifts all of the PLL outputs negatively and removes delay from the PLL. The CLKOS delay shifts only the CLKOS output port. This can change the overall delay of the PLL if CLKOS is used as the feedback clock source.

Note: In order to add delay to CLKOS, the CLKOS port must be using the output path that runs through the CLKOS output divider. If the added delay is used to create a relationship with CLKOP, then the CLKOP port must also run through the CLKOP output divider. There is a delay matching circuit in the CLKOP path to match a 0ps setting on CLKOS for this purpose.

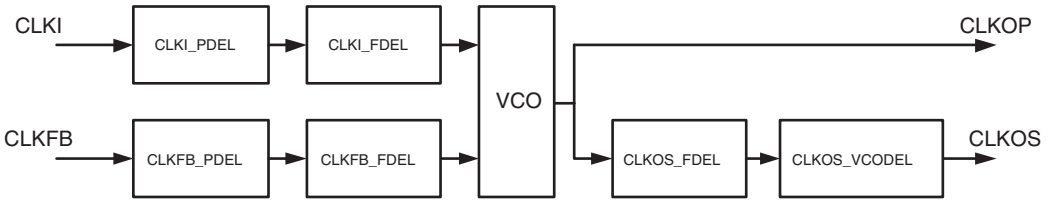
CLKOS VCO Delay: The CLKOS port supports a VCO delay which allows the CLKOS port to be delayed a specific number of VCO clock cycles. This is accomplished by holding the CLKOS output port in reset for the number of specified VCO clock cycles after a reset or configuration. CLKOS can be delayed 0 to 31 VCO clock cycles. The VCO rate can be found with the following equation (assuming internal feedback).

$$\text{VCO Rate} = \frac{\text{CLKI Rate}}{\text{CLKI\_DIV}} \text{ X } \text{CLKFB\_DIV}$$

CLKI_DIV and CLKFB_DIV can be found by looking in the generated HDL from IPexpress.

Figure 14 provides a simplified block diagram of the PLL additional delayed capabilities.

Figure 14. PLL Fixed Delay Block Diagram



The setting of these additional delays is discussed later in this document under the Advanced Preference Support section.

Considerations for use of various phase adjustments:

1. The Coarse Phase Delay on CLKI and CLKFB are not PVT compensated.
2. The accuracy of the PLL Clock Phase Adjustment and the CLKOS VCO Delay are much more accurate over PVT than the Fine Delay Capability.
3. The PLL Clock Phase Adjustment and the CLKOS VCO Delay change as the clock frequency changes - the Fine Phase Delay remains fixed. Therefore, the Fine Phase Delay should only be used for minor modifications to the clock delays.

PLL Frequency Synthesis

The PLL provides dividers on all of the clock inputs and outputs. A divider on the CLKFB works as a multiplier. For example, a divide by 2 on the CLKFB produces a VCO rate 2x that of CLKI. Using these dividers and feedback paths, the PLL can create new clock rates from a single CLKI rate. There is only a single VCO which will run at a given rate. The CLKOP and CLKOS outputs will need to be divided down rates of the VCO rate. Each divider can divide the clock signal by up to 64.

IPexpress removes the calculation of the dividers from the user. The user enters the input frequency and the desired output frequency and the IPexpress performs the calculations of the divider values. The VCO frequency must always remain within its operating frequency range provided in the LatticeSC/M Family Data Sheet. The dividers can be changed by a preference or at run time via the SMI interface. These topics are discussed later in this document.

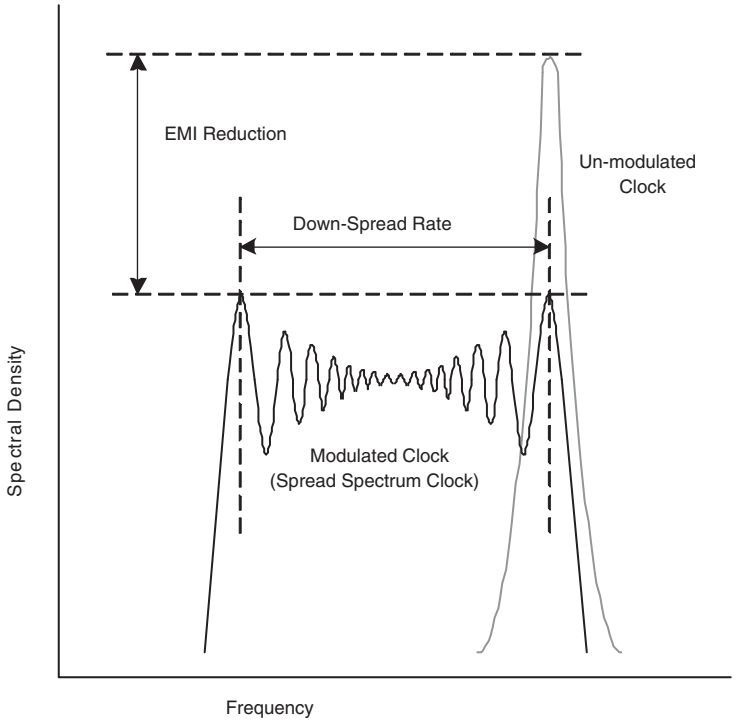
Spread Spectrum

Spread Spectrum is the technique of modulating the operating frequency of a circuit slightly to spread its radiated emissions over a range of frequencies rather than just one tone. This reduction in the maximum emission for a given frequency helps meet FCC requirements. The LatticeSC PLL includes a Spread Spectrum feature for designers who want to reduce the EMI emission in their systems. Spread Spectrum in this case is dynamically changing the PLL output frequency to disperse the PLL output clock energy across a wider frequency range. The frequency change is controlled, thus changing most of the "jitter" component into a controlled deterministic jitter.

Note: Due to the complex tuning requirements of the PLL in spread spectrum mode, please contact Lattice Technical Support for detailed information on programming the PLL in spread spectrum mode. See the Technical Support Assistance section at the end of this document for contact information.

The LatticeSC PLL supports "down-spread" modulation, the output frequency is always lower than or equal to the nominal output frequency. The PLL does not support "up-spread" or "center-spread" modulation. Figure 15 provides a plot of down-spread modulation. When enabled, the output frequency will vary in a range that is 0.5%, 1.0%, or 1.5% below its nominal value. This is accomplished by removing a clock cycle periodically.

Figure 15. Down-Spread Modulation Plot



For information regarding enabling spread spectrum in the LatticeSC PLL, contact Lattice Technical Support. Contact information is found at the end of this document.

Programmable Loop Bandwidth

The LatticeSC PLL provides a programmable bandwidth. This bandwidth is the measure of the PLL's ability to track jitter in the incoming reference clock. A low bandwidth PLL will filter out high frequency jitter while a high bandwidth PLL will track high frequency jitter and provide a shorter lock time. IPexpress supports a medium (default) and high bandwidth setting for the PLL. IPexpress uses the frequency of the VCO and bandwidth setting to determine settings for the loop filter resistance and current. After the PLL settings have been calculated in IPexpress, the VCO frequency and bandwidth are displayed for the user.

Creating a PLL using ispLEVER

IPexpress is used to create and configure a PLL. The user will use the graphical user interface to select parameters for the PLL. The result is an HDL model to be used in the simulation and synthesis flow.

Configuration Tab

The GUI will automatically determine and set the divider values for the PLL to create a VCO rate and CLKOP/CLKOS output rate based on user criteria.

CLKI Frequency – The rate (MHz) of the CLKI input.

CLKOP/CLKOS Desired Frequency – The rate at which the user wants the CLKOP/CLKOS output to run.

CLKOP/CLKOS Tolerance (%) – The accuracy required for the actual frequency from the desired frequency. The GUI is actually calculating how to use the internal dividers for CLKI, CLKFB, CLKOP, and CLKOS to create the output rates from the input rate. It may not be possible to create the exact frequency desired by the user. The tolerance field informs the GUI of the accuracy required for each clock output compared to the other output clock. This provides flexibility for the GUI to find a solution. If a solution is not possible the GUI will inform the user to make changes to the selections.

CLKOS Phase Shift – The CLKOS output can be phase shifted in reference to the CLKOP output. This field specifies the CLKOS to CLKOP phase offset in 45 degree increments. CLKOP will lead CLKOS by the amount of phase shift selected.

CLKFB Feedback Mode – Sets the feedback mode of the PLL to either internal, CLKOP, CLKOS, or User Clock. If internal feedback is selected minimal clock injection delay is removed. If CLKOP/CLKOS is selected then the clock tree injection delay for the specific output clock will be removed. If User Clock is selected then the user will be provided with the CLKFB port on the PLL.

CLKFB Frequency – If User Clock is selected for Feedback Mode the GUI needs to know the input frequency of the CLKFB port to calculate the divider values.

Advanced Tab

Provide RSTN port – The RSTN port allows the user to reset the PLL VCO and dividers via a user signal.

Enable GSR to Reset PLL – Enabled, the PLL VCO and dividers will be reset via the GSR. No user signal is required. When applying a reset to the PLL, the VCO operates at a low frequency.

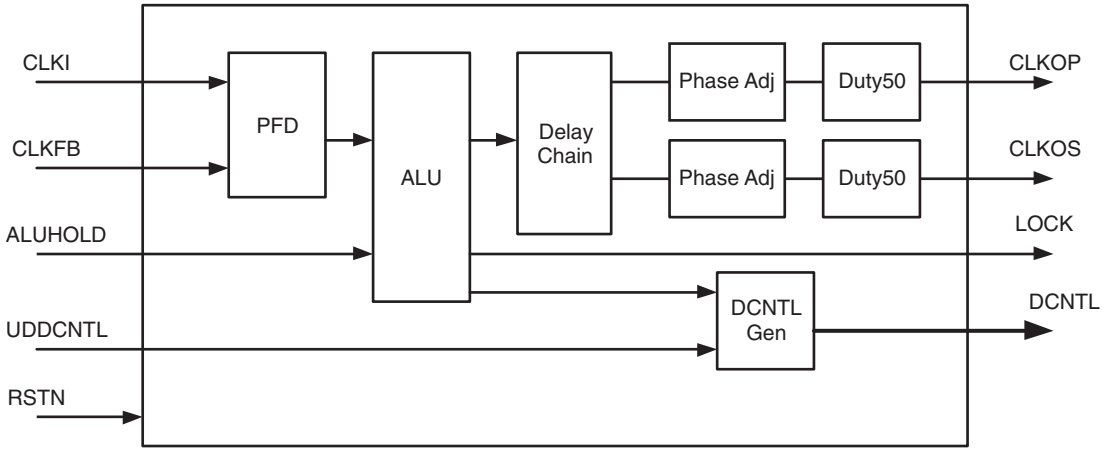
Provide SMI Ports – The SMI ports allow the user to change PLL behavior for output dividers and phase offset in system.

Use High Bandwidth – Enables IPexpress to create a high bandwidth loop filter in PLL.

DLL

The LatticeSC DLL provides features such as clock injection delay removal, delay match, time reference delay (90 degree phase delay), and output phase adjustment. The DLL performs clock manipulation by adding delay to the CLKI input signal to create specific phase relationships. There are two types of outputs of the DLL. The first are clock signals similar to the PLL CLKOP and CLKOS. The other type of output is a delay control vector (DCNTL[9:0]). The delay control vector is connected to a DELAY element located in the I/O logic which matches the delay cells in the DLL. This delay vector allows the DLL to dynamically delay an input signal by a specific amount. Figure 16 provides a block diagram of the LatticeSC DLL.

Figure 16. LatticeSC DLL Block Diagram



Clock injection delay removal and output phase adjustment both use only the clock outputs of the DLL. Time reference delay and delay match modes use the delay control vector output. Specific examples of these features will be discussed later in this document.

Ports and Descriptions

The LatticeSC DLL is created and configured by IPexpress. The following is a list of port names and descriptions for the DLL. There are four library elements used to implement the DLL. These include CIDDLLA (clock injection delay), SDCDLLA (clock injection delay with DCNTL), CIMDLLA (clock delay match), and TRDDLLA (time reference delay). IPexpress will wrap one of these library elements to create a customized DLL module based on user selections.

CLKI Input
The CLKI signal is the reference clock for the DLL. The CLKI input can be sourced from any type of FPGA routing and pin. The DLL CLKI input has a preferred pin per DLL which will be discussed later in this tech note. The preferred pin provides the lowest latency and best case performance.

CLKFB Input
The CLKFB input is only available if the user selects to use a user clock signal for the feedback or in clock delay match mode. If internal feedback or CLKOS/CLKOP is used for the feedback this connection will be made inside the module.

In clock injection delay removal mode the DLL will align the input clock phase with the feedback clock phase by delaying the input clock.

In clock delay match mode the DLL will calculate the delta between the CLKI and CLKFB signals. This delay value is then output on the DCNTL vector.

The DLL CLKFB input has a preferred pin per DLL which will be discussed later in this user's guide. The preferred pin provides the lowest latency and best case performance.

CLKOP Output
An output of the DLL based on the CLKI rate. The CLKOP output can drive primary, edge, and secondary clock routing.

CLKOS Output
An output of the PLL based on the CLKI rate which can be divided and/or phase shifted. The CLKOS output can drive primary, edge, and secondary clock routing.

DCNTL[9:0] Output
This output of the DLL is used to delay an I/O input signal a specific amount. The DCNTL[9:0] vector connects to a DELAY element that is located in the I/O logic. The DLL can then control multiple input delays from a single DLL. The DLL uses the exact same DELAY elements as the I/O logic and will have the same compensation to process, voltage and temperature conditions.

There are two 10-bit delay control vector routes available in each bank. Delay control vectors can be shared across banks as well.

The DCNTL[9:0] bus can also be driven by FPGA user logic instead of the DLL. For more information on using the DCNTL bus, refer to TN1088, LatticeSC PURESPEED™ I/O Usage Guide.

UDDCNTL Input
This input is used to enable or disable updating of the DCNTL[9:0]. To ensure that the signal is captured by the synchronizer in the DLL block, it must be driven high for a time equal to at least two clock cycles when an update is required. If the signal is driven high and held in that state, the DCNTL[9:0] outputs are continuously updated.

ALUHOLD Input
This active high input stops the DLL from adding and subtracting delays to the CLKI signal. The DCNTL[9:0], CLKOP, and CLKOS outputs will still be valid, but will not change from the current delay setting.

LOCK Output

Active high lock indicator output. The LOCK output will be high when the CLKI and CLKFB signal are in phase. This is true for both external and internal feedback configurations.

If the CLKI input stops the LOCK output will remain asserted. The clock is stopped so there is no clock to de-assert the LOCK output. Note that this is different from the operation of the PLL where the VCO continues to run when the input clock stops.

RSTN Input

Active low reset input to reset the DLL. The DLL can optionally be reset by the GSR as well. It is recommended that if the DLL requires a reset that the reset is not the same as the FPGA logic reset. Typically logic requires that a clock is running during a reset condition. If the data path reset also resets the DLL the source of the logic clock will stop and this may cause problems in the logic.

SMI (Serial Management Interface)

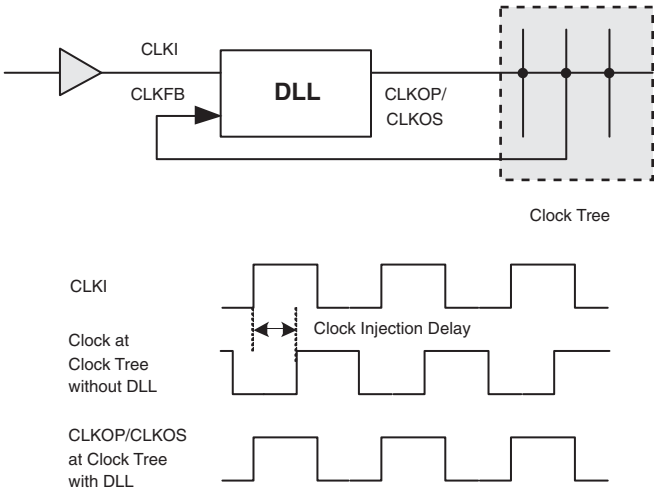
The DLL supports a run time control interface for modifying the behavior of the DLL in the system. Such parameters as output dividers and phase offset can be changed while in the system. This control interface is known as the Serial Management Interface (SMI). More information on using the SMI will be found later in this tech note.

DLL Modes of Operation

DLL Clock Injection Removal

The DLL can be used to reduce clock injection delay (CIDDLLA). Clock injection delay is the delay from the input pin of the device to a destination element such as a flip-flop. The DLL will add delay to the CLKI input to align CLKI to CLKFB. If the CLKFB signal comes from the clock tree (CLKOP, CLKOS) then the delay of the DLL and the clock tree will be removed from the overall clock path. Figure 17 shows a circuit example and waveform.

Figure 17. Clock Injection Delay Removal via DLL

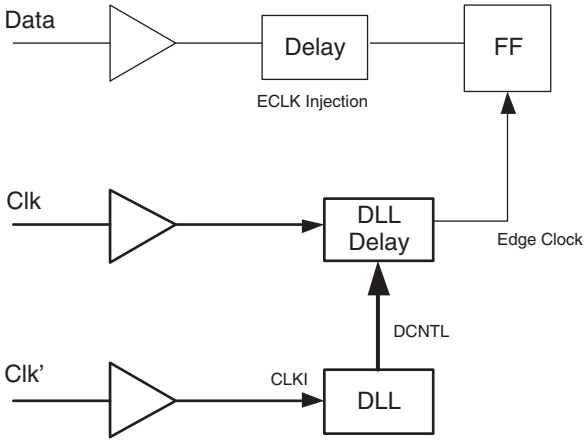


Clock injection removal mode can also provide a DCNTL port. In this mode the delay added to the CLKI signal is output on the DCNTL port so that other input signals can be delayed the same amount. This is very useful if several clocks are used in the same circuit to minimize the number of DLLs required. When using the DCNTL the DLL delay will be limited to the range of the DCNTL vector. Therefore, IPexpress will restrict the CLKI rate from 300MHz to 700MHz.

DLL Time Reference Delay (90 Degree Input Phase Delay)

The Time Reference Delay (TRDDLLA) mode of the DLL is used to calculate 90 degrees worth of delay to be placed on the DCNTL vector. This is a useful mode in delaying a clock 90 degrees for use in clocking a DDR type interface. Figure 18 provides a circuit example of this mode.

Figure 18. Time Reference Delay Circuit Example



In this mode, CLKI accepts a clock input. The DLL produces a DCNTL vector that will delay an input signal by 90 degrees of a full period of the CLKI signal. This DCNTL vector can then be connected to an I/O delay element to delay the signal by 90 degrees of the full period of CLKI. More information on using the time reference delay mode of the DLL can be found in TN1088, LatticeSC PURESPEED I/O Usage Guide.

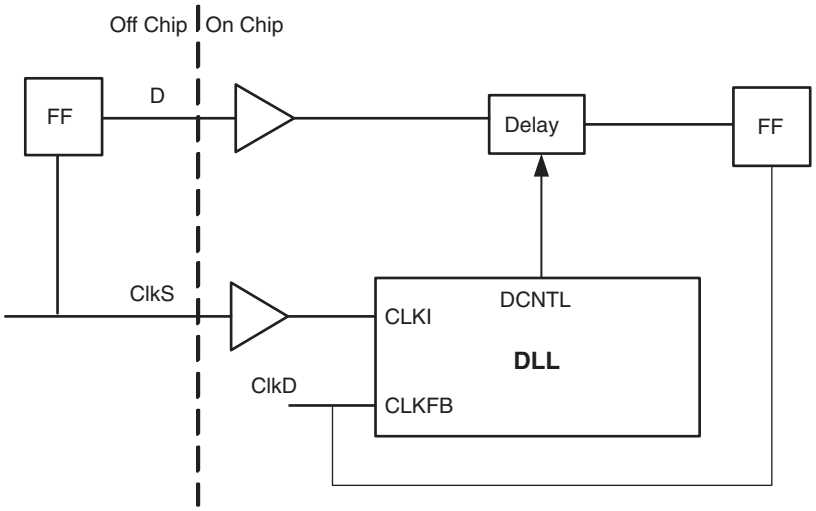
For applications requiring delays slightly less than or slightly greater than 90 degrees, the DCNTL 90 degree phase shift can be incremented or decremented by setting the DLL SMI offset 0x8 [7:0] register setting. This register setting will either add (DLL SMI offset 0x8 bit 7 = 0) or subtract (DLL SMI offset 0x8 bit 7 = 1) t_{FDEL} delays from the default 90 degree phase shift. The number of t_{FDEL} delays to be added or subtracted is specified by DLL SMI offset 0x8 bits [0:6].

Note: The specification of t_{FDEL} is located in the DC and Switching Characteristics section of the LatticeSC/M Family Data Sheet.

DLL Clock Delay Match

The Clock Delay Match (CIMDLLA) mode of the DLL allows two synchronous clock inputs to be used to create a DCNTL vector that is the delta of the delay between the two clocks. In this circuit the DLL will compare the phase difference between the CLKI and CLKFB inputs to create a delay vector. This is a useful mode when data is transmitted off one clock (CLKI) and captured on a different clock (CLKFB) that is synchronous to each other. By delaying the data inside the I/O logic before being latched by CLKFB the data signal will be delayed the same amount as the difference in the clock delay. Figure 19 provides a circuit example of the mode.

Figure 19. Clock Delay Match Circuit Example



In Clock Delay Match mode the maximum delay is limited to the extent of the DCNTL bus (3.9ns).

Additional DLL Features

In addition to the major modes of operation the DLL has several features that are available in all modes.

CLKI and CLKOS Dividers

The CLKI and CLKOS ports support a divide by 2 or divide by 4 clock divider. When using the CLKOS divider CLKOP is compensated to maintain the correct phase alignment. The CLKI divider setting is not available when creating the module using the IPexpress. Setting this divider will be discussed later in this document in the Advanced Preference Support section.

CLKOS Phase Shift

The CLKOS output also allows the user to set a phase offset in reference to the CLKOP output. The CLKOP output will lead the CLKOS output by the amount of phase set on CLKOS.

Duty Cycle Correction

The DLL can also correct the duty cycle of the CLKI input signal to create a "50/50" output clock. This feature is available on both the CLKOP and CLKOS outputs of the DLL. Duty cycle correction can be enabled in the Pre-Map Preference Editor.

Creating a DLL using ispLEVER

IPexpress is used to create and configure a DLL. The user will use the graphical user interface to select parameters for the DLL. The result is an HDL model to be used in the simulation and synthesis flow.

Configuration Tab

Usage Mode – Select the mode of the DLL (Time Reference Delay, Clock Injection Delay Removal, or Clock Delay Match). This selection will enable or disable further options in the GUI.

CLKI Frequency – The rate (MHz) of the CLKI input.

CLKOS Divide – Set the divider for the CLKOS output to be either no divide, divide by 2, or divide by 4.

CLKOS Phase Shift – Set the phase offset of the CLKOS to the CLKOP output. CLKOP will lead CLKOS by the amount of phase shift selected.

CLKFB Feedback Mode – Sets the feedback mode of the DLL to either internal, CLKOP, CLKOS, or User Clock. If internal feedback is selected then minimal clock injection delay is removed. If CLKOP/CLKOS is

selected then the clock tree injection delay for the specific output clock will be removed. If User Clock is selected then the user will be provided with the CLKFB port on the DLL.

CLKFB Frequency – This is a read-only field to display to the user the rate of the CLKFB input.

Provide RSTN port – The RSTN port allows the user to reset the DLL through a user signal.

Enable GSR to reset DLL – If selected the DLL will be reset via the GSR. No user signal is required.

Provide SMI ports – The SMI ports allow the user to change DLL behavior for output dividers and phase offset in system.

Provide DCNTL port – In Clock Injection Delay Mode it is possible to obtain the delay added to the CLKI port on the DCNTL port. This can then be used to delay other clock inputs by the same amount by connecting the DCNTL vector to DELAY elements.

PLL/DLL Cascading

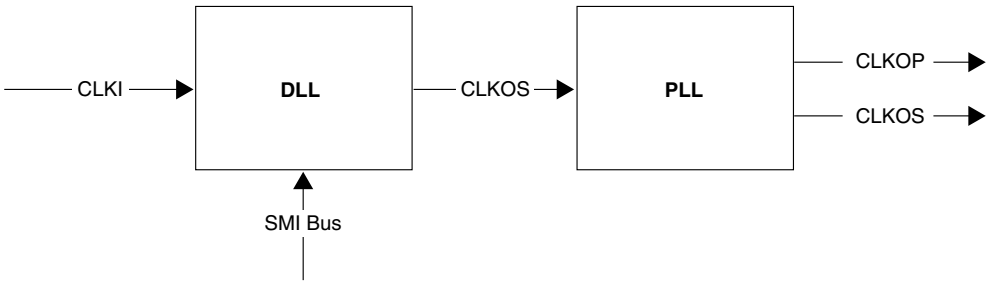
It is possible to connect together several arrangements of PLLs and DLLs or fine phase shifts of an output clock signal. There are four possible cascading schemes:

- PLL to PLL
- PLL to DLL
- DLL to DLL
- DLL to PLL

Cascading the DLL to the PLL

The DLL can be used to drive the PLL to create fine phase shifts of an input clock signal. Figure 20 provides a diagram of the cascading configuration where the DLL shifts all outputs for CLKOP and CLKOS out in time.

Figure 20. DLL to PLL



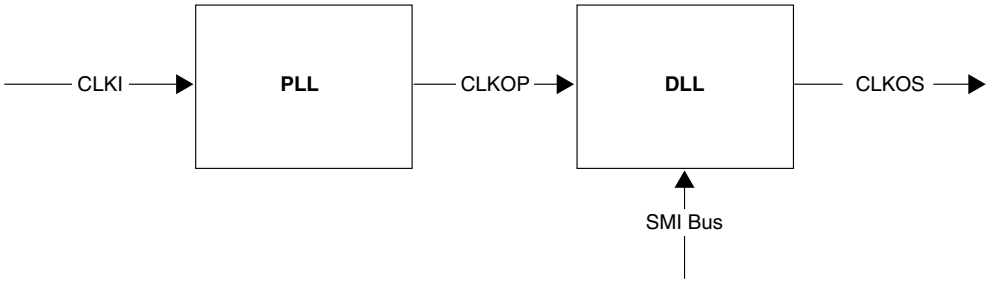
Note: This figure shows the PLL using internal feedback. It is also possible to use external feedback using either CLKOP or CLKOS.

By cascading the DLL to the PLL, the DLL can add very fine delay to the CLKI input resulting in fine phase shifting at the output of the PLL. As the DLL provides finer phase resolution than the 45 degree setting of the PLL, this application is useful in clock forwarding applications where the data and clock relationship can be slightly offset. The PLL is used at the output to filter the jitter of the incoming clock.

In this application the DLL must be programmed to use static delay. Typically the DLL is changing its delay setting to maintain a relationship between the CLKI and CLKFB. In this system there is no DLL CLKFB and the delay setting will remain constant. This is necessary since the PLL will not be able to tolerate abrupt phase changes that could be created by the DLL. When using this cascading scheme it is best to use edge clock routing from the CLKOS output clock of the DLL to the PLL input. This can be accomplished by using a USE EDGE NET preference.

Figure 21 provides a diagram of the cascading configuration where the DLL shifts only CLKOP out in time.

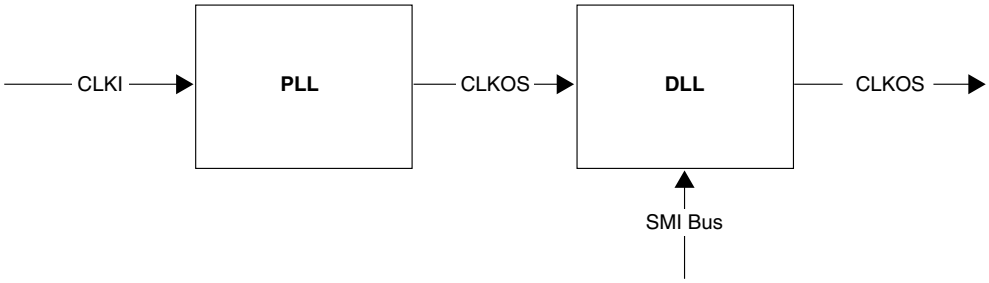
Figure 21. PLL to DLL



Note: This figure shows the PLL using internal feedback. It is also possible to use external feedback using either CLKOP or CLKOS.

Figure 22 provides a diagram of the cascading configuration where the DLL shifts only CLKOS out in time.

Figure 22. PLL to DLL

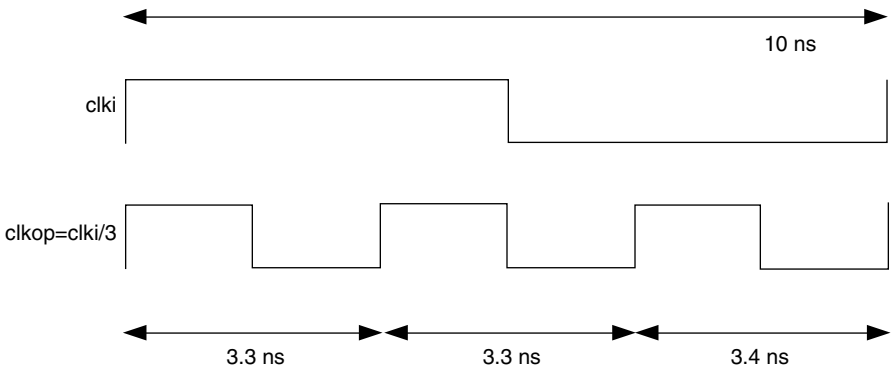


Note: This figure shows the PLL using internal feedback. It is also possible to use external feedback using either CLKOP or CLKOS.

Cascading PLLs

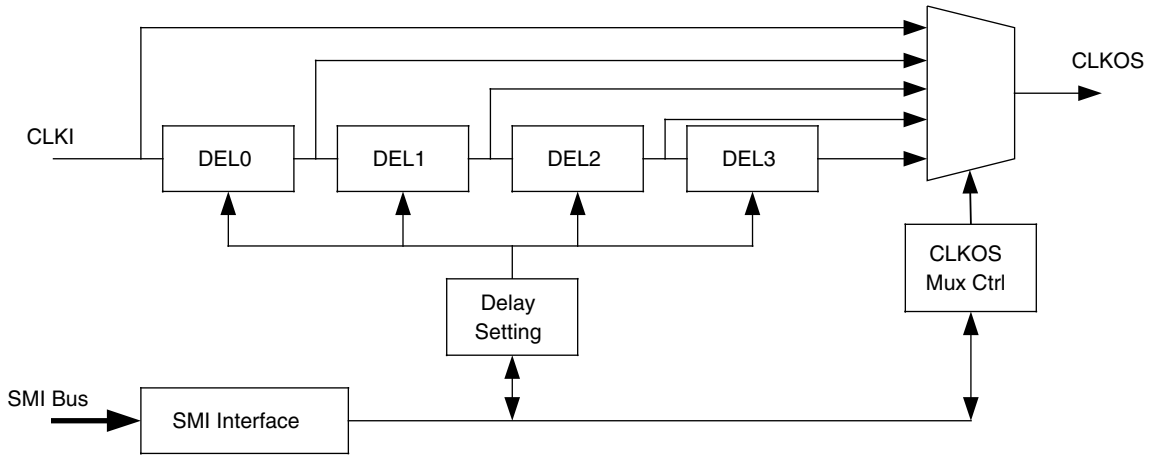
When cascading PLLs, the second PLL may not lock during simulation due to rounding errors caused by certain values of clkop. For example, when clkop=clki/3 and clkop is used as the input to the second PLL, the second PLL does not lock because the time period is not the same. In this situation, it is recommended to use an input frequency that will avoid this issue.

Figure 23. Cascading PLLs



The DLL creates delay through a series of delay blocks as shown in Figure 24.

Figure 24. DLL Delay Architecture



The user must select how much delay is added in the CLKI to CLKOS path. This selection is done using two controls. The first control selects how many delay blocks are used in the DLL path. The second control selects the delay of each delay block. Notice that the same delay value is used in each delay block. The total delay of the DLL can be found using the following formula.

Total DLL Delay = Static DLL delay (t_{DLL}) + (Static Delay block delay (t_{FDEL}) * Delay Setting Value * Number of Delay blocks selected by the CLKOS Mux Ctrl)

Note: Delay Setting Value = (DLL SMI offset 0x6 [0:7] register setting) where 0 ð (DLL SMI offset 0x6 [0:7] register setting) ð 0x8F (decimal 143).

Note: The specification of t_{DLL} and t_{FDEL} are located in the DC and Switching Characteristics section of the LatticeSC/M Family Data Sheet.

Creating and Configuring the DLL

IPexpress is used to create a Time Reference Delay DLL configuration with a SMI. The feedback clock selection in IPexpress is not important since the feedback path is not used in this application. The SMI will be required to control the CLKOS mux as well as the delay setting. Refer to the Serial Management Interface (SMI) section of this document to learn how to use the SMI. The SMI is the only method for setting the static delay value.

Once the DLL is created the user will need to program the CLKOS mux and delay setting at run time. The following steps will need to be performed. Reference the SMI Interface DLL memory map found later in this document.

1. Force the DLL to use a static delay value by setting SMI offset 0x3 bit6 to 1
2. Select the number of delay blocks to use in the clock path using the CLKOS Mux Ctrl (SMI offset 0x1 bits [5:7] as specified in the DLL memory map).
3. Set the Delay Setting using SMI offset 0x6 [0:7] as specified in the DLL memory map.

These values can be changed at any time once the FPGA design has been loaded and the SMI interface is available. Run time changes to the static delay value are not represented in TRACE or timing simulation. TRACE and timing simulation will only show the static delay of the DLL.

The PLL configuration for accepting a clock from the DLL is not specific. Any PLL configuration can be used.

IPexpress Output

There are two outputs of IPexpress that are important for use in the design. The first is the <module_name>.[v|vhd] file that is produced. This is the user-named module that was generated by the tool to be used in both synthesis and simulation flows. The second file is a template file <module_name>_tmpl.[v|vhd]. This file contains a sample

instantiation of the module. This file is only provided for the user to copy/paste the instance and is not intended to be used in the synthesis or simulation flows directly.

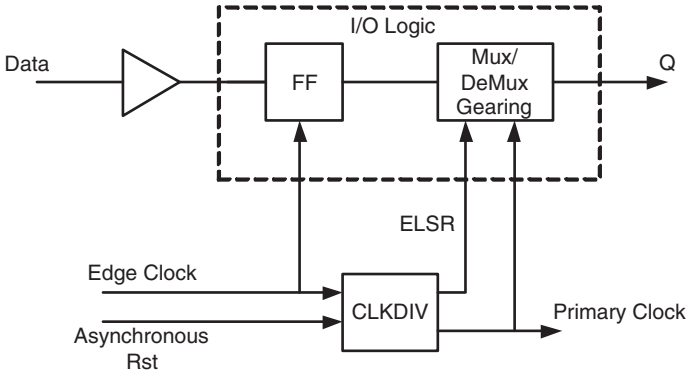
For the PLL/DLL, IPexpress sets attributes in the HDL module created that are specific to the data rate selected. Although these attributes can be easily changed they should only be modified by re-running the GUI so that the performance of the PLL/DLL is maintained. After the map stage in the design flow FREQUENCY preferences will be included in the preference file to automatically constrain the clocks produced from the PLL/DLL.

Clock Dividers

The clock divider (CLKDIV) can divide a clock by 2 or 4 and drives a primary clock network. The clock dividers are useful for providing the low speed FPGA clocks for I/O shift registers (x2, x4) and DDR (x2, x4) I/O logic interfaces. To guarantee a synchronous transfer in the I/O logic the CLKDIV input clock must come from an edge clock and the output drive a primary clock. In this case they are phase matched.

A CLKDIV can also be used to create a low skew edge local set/reset (ELSR) using edge clock routing. This is especially useful for synchronously resetting the I/O logic when Mux/DeMux gearing is used in order to synchronize the entire data bus as shown in Figure 25. Using the low skew characteristics of the edge clock routing a reset can be provided to all bits of the data bus to synchronize the Mux/DeMux gearing.

Figure 25. CLDIV Example Circuit with I/O Logic



There are special connections for the edge clock CLKI and ELSR ports of the CLKDIV element. These connections are provided later in this document in the Design Implementation section.

Creating a CLKDIV

There are two methods for creating a CLKDIV for use in the FPGA design. The first method is via IPexpress. When creating SDR and DDR I/O interfaces using Mux/DeMux gearing in IPexpress the user can select to use a CLKDIV. In this method the CLKDIV becomes part of the I/O interface and all attributes are set for the specific interface.

The second method is to instantiate the CLKDIV library element directly in the HDL code. Figure 26 provides the CLKDIV library element definition.

Figure 26. CLKDIV Library Element

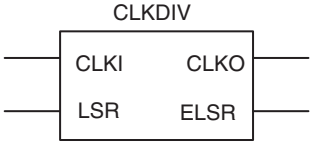


Table 1. CLKDIV Port Definition

Name	Description
CLKI	Clock Input
LSR	Asynchronous Active High Reset Input
CLKO	Clock Output
ELSR	Synchronous Edge Reset Output clocked out on CLKI

Table 2. CLKDIV Attribute Definition

Name	Description	Value	Default
DIV	Divider	1,2,4 (1: off)	1
GSR	GSR Enable	Enable/Disable	Disable

CLKDIV Instantiation in HDL

VHDL Example

```
component CLKDIV
-- synopsys translate_off
generic (DIV : in Integer);
-- synopsys translate_on
port (CLKI: in  std_logic; LSR: in  std_logic;
      CLKO: out  std_logic; ELSR: out  std_logic);
end component;

attribute DIV : string;
attribute DIV of I : label is “2”;

I: CLKDIV
-- synopsys translate_off
generic map (DIV=> 2)
-- synopsys translate_on
port map (CLKI=>EClk, LSR=>Rst, CLKO=>SClk, ELSR=>ERst);
```

Verilog Example

```
// synopsys translate_off
defparam I.DIV = 2;
// synopsys translate_on
CLKDIV I (.CLKI(EClk), .LSR(Rst), .CLKO(SClk), .ELSR(ERst))
/* synthesis DIV=”2” */;
// exemplar attribute I DIV 2
```

Dynamic Clock Select (DCS)

DCS is a global clock buffer incorporating a smart multiplexer function that takes two independent input clock sources and avoids generating glitches or runt pulses on the output clock, regardless of where the enable signal is toggled.

The DCSs are located in pairs at the center of each edge. There are eight of them in each LatticeSC device.

The output of the DCS drives only primary clock. Figure 27 shows the block diagram of the DCS.

Figure 27. DCS Library Element

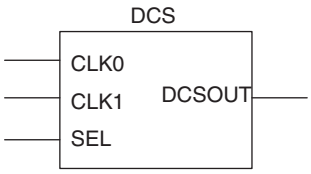


Table 3. DCS Port Definitions

I/O	Name	Description
Input	SEL	Input Clock Select (see Table 4)
	CLK0	Clock input 0
	CLK1	Clock Input 1
Output	DCSOUT	Clock Output

There are eight different modes user can select from. Table 4 describes how each mode is configured.

Table 4. DCS Modes of Operation

Attribute Name	Description	Output		Value
		SEL=0	SEL=1	
DCSMODE	Rising edge triggered, latched state is high	CLK0	CLK1	POS
	Falling edge triggered, latched state is low	CLK0	CLK1	NEG
	Sel is active high, Disabled output is low	0	CLK1	HIGH_LOW
	Sel is active high, Disabled output is high	1	CLK1	HIGH_HIGH
	Sel is active low, Disabled output is low	CLK0	0	LOW_LOW
	Sel is active low, Disabled output is high	CLK0	1	LOW_HIGH
	Buffer for CLK0	CLK0	CLK0	CLK0
	Buffer for CLK1	CLK1	CLK1	CLK1

DCS Timing Diagrams

Each mode performs its unique operation. The clock output timing is determined by input clocks and the edge of SEL signal. Figure 28 describes the timing of each mode.

Figure 28. Timing Diagrams by DCSMODE

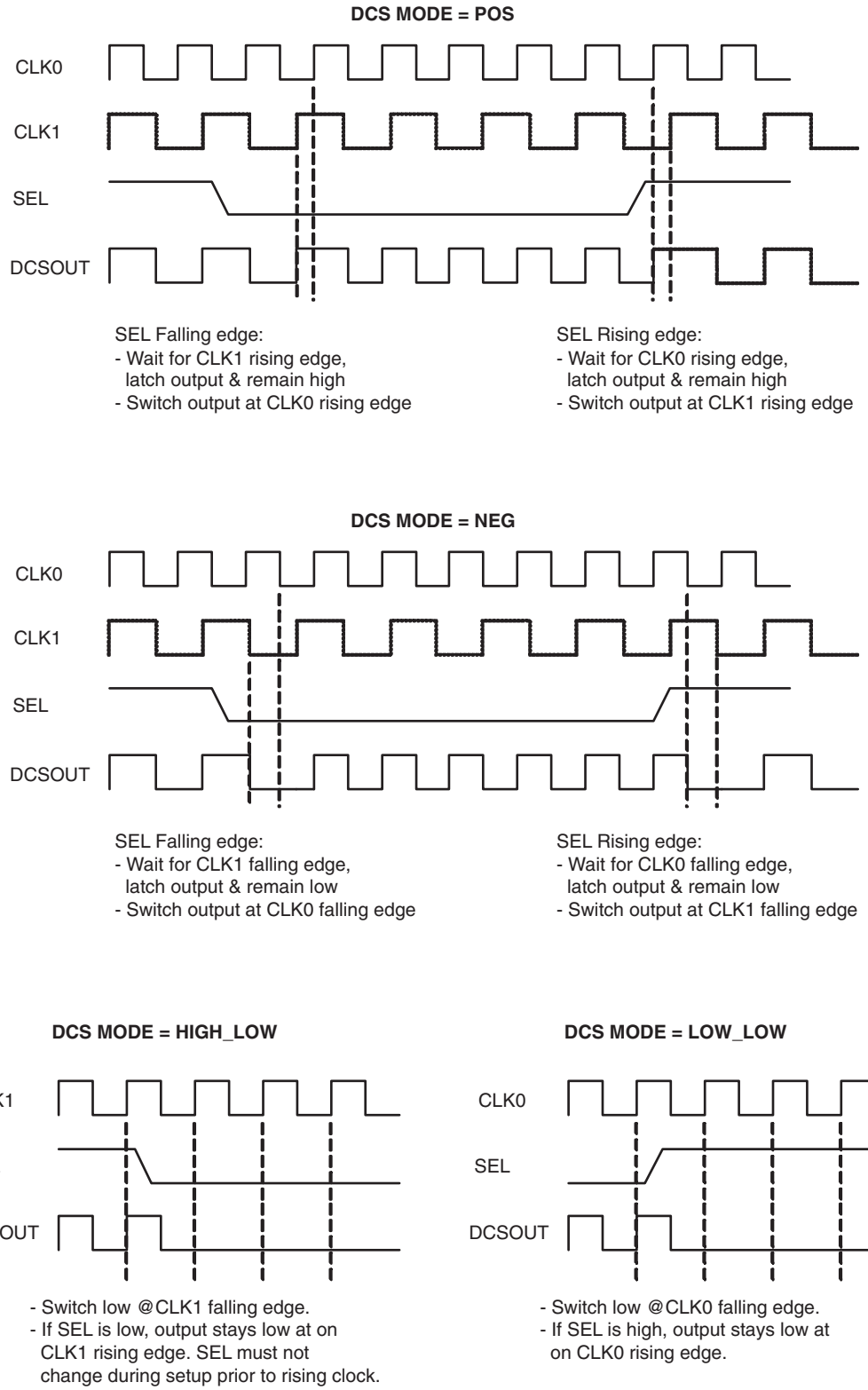
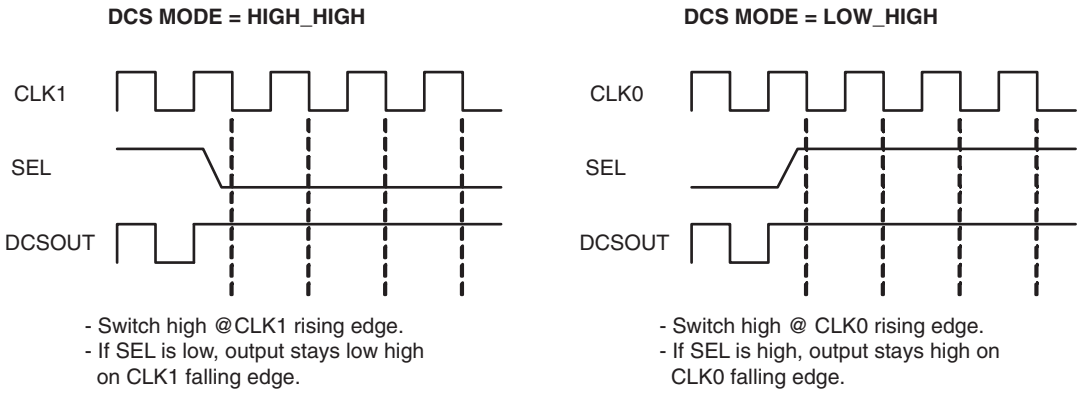


Figure 28. Timing Diagrams by DCSMODE (Cont.)



DCS Instantiation in HDL

VHDL Example

```
component DCS
-- synthesis translate_off
generic  (
  DCSMODE : string :=  "POS"
-- synthesis translate_on
PORT (
CLK0 : IN    std_logic;
CLK1 : IN    std_logic;
SEL : IN    std_logic;
DCSOUT : OUT   std_logic);
END COMPONENT;

attribute DCSMODE : string;
attribute DCSMODE of I: label is "POS";

I : DCS
-- synthesis translate_off
generic map (
DCSMODE  => "POS")
-- synthesis translate_on
port map (
SEL => clksel,
CLK0 => dcsclk0,
CLK1 =>  dcsclk1,
DCSOUT =>  dcsclk );
```

Verilog Example

```
// synopsys translate_off
defparam I.DCSMODE = "POS";
// synopsys translate_on
DCS I (.SEL(clksel), .CLK0(dcsclk0), .CLK1(dcsclk1), .DCSOUT(dcsout))
/* synthesis DCSMODE="POS" */;
// exemplar attribute I DCSMODE "POS"
```

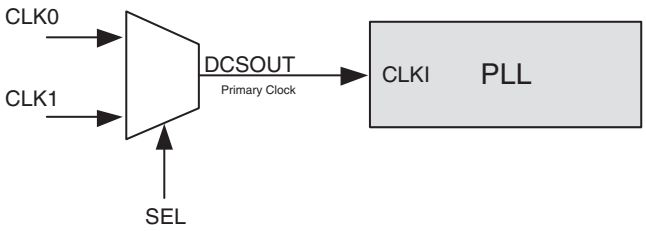
Dynamic Clock Switching at PLL/DLL Inputs

Input Reference Clock Switchover

A dynamically controlled 2:1 mux is included in the reference clock path to allow for dynamic switching of the reference clock. The intent of this feature is to allow the PLL/DLL to switch between two reference input clocks. This can be used in systems with clock redundancy and in dual clock domain systems where one of the clocks may stop running for some reason.

A user signal from the FPGA core is used to switch the reference clock from CLK0 to CLK1. A circuit example of the PLL reference clock switchover is shown in Figure 29. The two clocks from pads are inputs to a 2:1 mux that is controlled by the clock switchover circuit. The output of the mux drives into the Reference clock input of the PLL/DLL block.

Figure 29. PLL Reference Clock Switchover Circuit Example



Clock Shut Off (CLKCNTL)

The clock control block CLKCNTL is used to synchronously stop and asynchronous start a primary, edge, or secondary clock that is sourced by an input buffer. This element allows a clock to be gated by a logic signal without requiring the use of a LUT. This feature is useful in DDR memory interfaces to perform the DQS postamble correction. Figure 30 provides the block diagram of the CLKCNTL library element.

Figure 30. CLKCNTL Block Diagram and Example Waveform

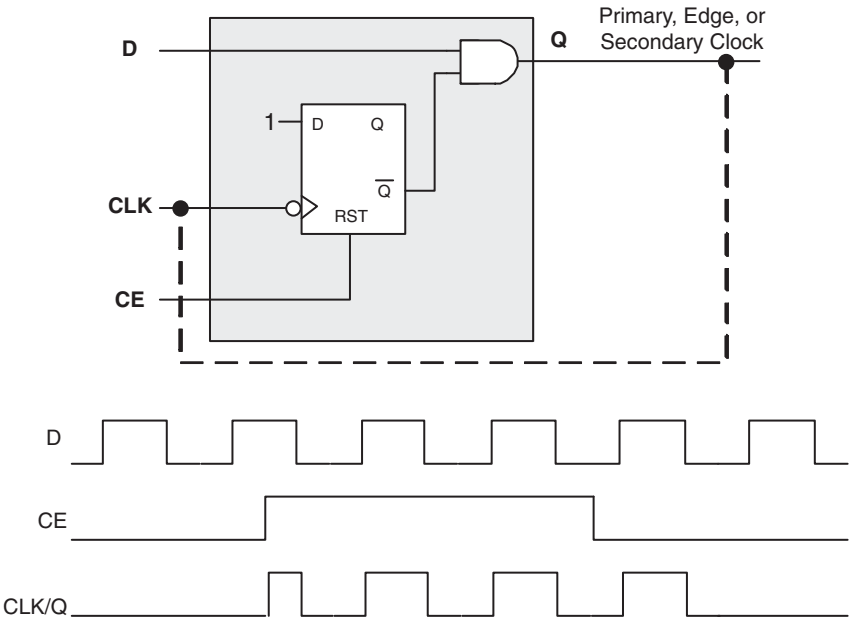


Table 5. CLKCNTL Library Element

I/O	Name	Description
Input	D	Input Clock Signal
	CLK	Output Clock Signal Feedback
	CE	Async Active High Clock Enable Sync Active Low Clock Disable (synchronous to CLK falling edge)
Output	Q	Output Clock Signal

The CLKCNTL is implemented in the I/O logic and is available on every I/O pin. It can also be used to force any or all inputs to the device low synchronously.

A DQS Postamble Solution can be created using IPexpress which uses the CLKCNT element to perform the DQS shutoff. After the postamble occurs on DQS pad, it is possible for spurious transitions or runt pulses to occur due to the tristate condition. Incorrect data could then be transferred into next stage of registers if rising edge of clock occurs after postamble. The CLKCNTL element is used to stop the edge clock so that incorrect data is not latched.

CLKCNTL Instantiation in HDL

VHDL Example

```
component CLKCNTL
PORT (
D : IN    std_logic;
CLK : IN    std_logic;
CE : IN    std_logic;
Q : OUT   std_logic);
END COMPONENT;


I : CLKCNTL
port map (
D => clkin,
CLK => clkout,
CE => enable,
Q => clkout );
```

Verilog Example

```
CLKCNTL I (.D(clkin), .CLK(clkout), .CE(enable), .Q(clkout));
```

Clock Edge Detection (CLKDET)

The clock edge detection (CLKDET) element as shown in Figure 31 is a single shot rising edge detector that is available in the I/O logic of every I/O pin. This element has many uses an example of which is its use for DQS preamble detection for DDR memory interfaces.

Figure 31. CLKDET Block Diagram

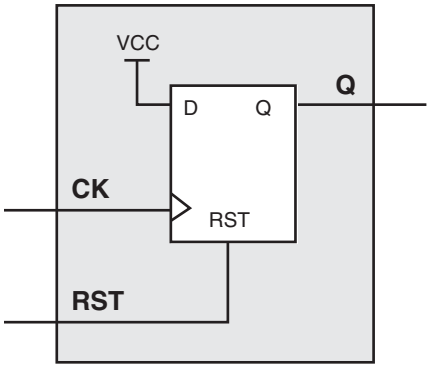


Table 6. CLKDET I/O Definition

I/O	Name	Description
Input	CK	Clock input
	RST	Async Reset Input
Output	Q	Detection Output

Table 7. CLKDET Attribute Definition

Attribute	Name	Description
CLKMODE	ECLK	Input clock is Edge Clock
	SCLK	Input clock is System Clock

CLKDET Instantiation in HDL

VHDL Example

```
component CLKDET
-- synthesis translate_off
generic  (
  CLKMODE : string :=  "ECLK"
-- synthesis translate_on
PORT (
CK : IN    std_logic;
RST : IN    std_logic;
Q : OUT   std_logic);
END COMPONENT;

attribute CLKMODE : string;
attribute CLKMODE of I: label is "ECLK";

I : CLKDET
port map (
CK => clkin,
RST => clkrst,
Q => clkdet );
```

Verilog Example

```
// synopsys translate_off
defparam I.CLKMODE = "ECLK";
// synopsys translate_on
CLKDET I (.CK(clkin), .RST(clkrst), .Q(clkdet));
```

```
/* synthesis CLKMODE="ECLK" */;
// exemplar attribute I CLKMODE "ECLK"
```

Oscillator

The internal oscillator (OSCA) is the source of the configuration clock in master config modes. It may also be used after configuration as a general-purpose clock. After configuration, the frequency is set with the DIV attribute on the OSCA library element. The OSCA has a frequency range of 1 to 130MHz with tolerance of -30% to +45%.

During configuration the rate of the oscillator is set via the SYSCONFIG preference.

```
SYSCONFIG MCCLK_FRQ=4; // Mhz
```

Post configuration the rate of the oscillator is set via the DIV attribute on the library element. The library element is shown in Figure 32.

Figure 32. Oscillator Library Element

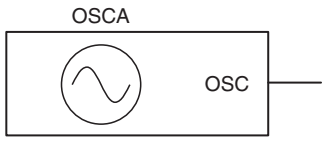


Table 8. Oscillator Attribute Definition

Attribute	Description	Values	Default
DIV	Divider Value	1,2,4,8,16,32,64,128	1 (about 130 MHz)

OSCA Instantiation in HDL

VHDL Example

```
component OSCA
-- synthesis translate_off
generic  (
  DIV : integer := 8
-- synthesis translate_on
PORT (
OSC : OUT    std_logic);
END COMPONENT;

attribute DIV : integer;
attribute DIV of I: label is 8;

I : OSCA
port map (
OSC => clk);
```

Verilog Example

```
// synopsys translate_off
defparam I.DIV = 8;
// synopsys translate_on
OSCA I (.OSC(clk));
/* synthesis DIV = 8 */;
// exemplar attribute I DIV 8
```

Clock Boosting

Clock boosting uses programmable clock delays inside the PFU, PIC, and EBRs to be configured by ispLEVER to trade short data delays with long data delays to increase overall system performance. For example, if a data path is meeting timing on a very short routing path and missing timing on a long routing path a clock delay can be added to the long data route to help in meeting timing. The trade off will be on the hold side of the short data path. The ispLEVER clock boosting tool understands this timing trade off and will set the clock delays to optimize system performance. The ispLEVER clock boosting tool also verifies that any clock boosting performed does not introduce any hold time issues. For more information on clock boosting, please refer to TN1131, Clock Boosting in Lattice SC/M FPGAs.

It is also possible to manually set these clock delays in the preference file. The CLKDELAY preference allows the user to set a clock delay on a CELL (PFU or PIC) or ASIC (EBR). Below are two examples. Valid values for CLKDELAY are DEL0 (no delay), DEL1, DEL2, and DEL3. See the LatticeSC/M Family Data Sheet for delay values for DEL1, DEL2, and DEL3.

```
CLKDELAY CELL "ff_a" DEL1

CLKDELAY ASIC "EBR1" PIN CLKIDEL DEL3
```

Design Implementation

The following sections detail how to use preferences, set locations and choose input pins for the clocking elements.

PLL, DLL, and CLKDIV Locations

Figure 33 shows the locations, site names, and connectivity of the PLLs, DLLs, CLKDIVs and edge clocks. Please note, only the PCLK pins with an arrow going directly to the CLKDIV can drive that CLKDIV directly. The dots indicate connectivity to the CLKDIV for either the PLL or DLL, but not the ECLK line as a whole. For example, PCLKT3_0 cannot drive CLKDIV2C, only CLKDIV2C, only PCLKT2_2, DLL_URCC (P) and DLL_URCD (S) outputs can. Table 9 shows the CLKDIVs and their available sources.

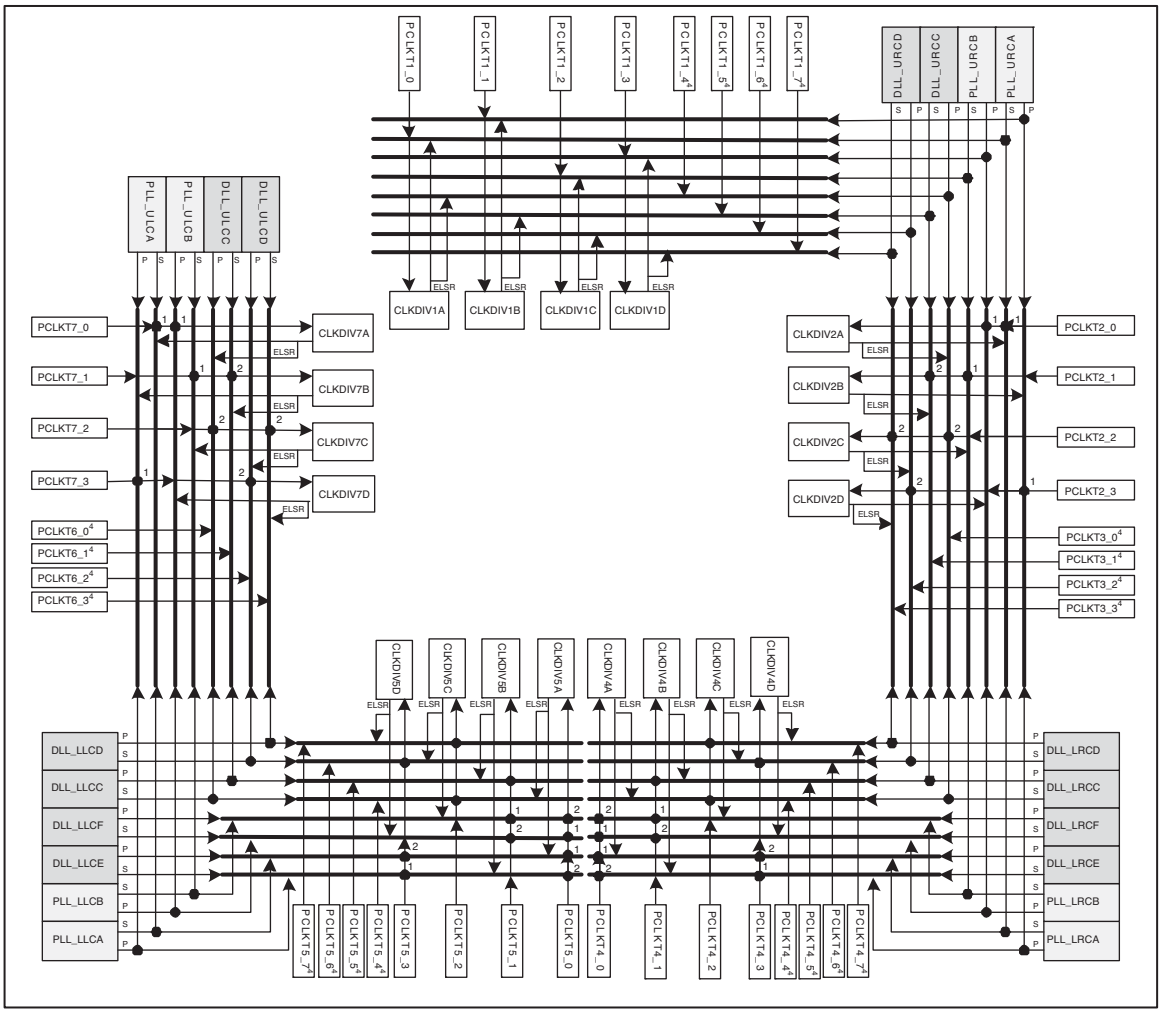
Table 9. Valid CLKDIV Drivers

CLKDIV	CLKDIV Input Source				
CLKDIV1A	PCLKT1_0				
CLKDIV1B	PCLKT1_1				
CLKDIV1C	PCLKT1_2				
CLKDIV1D	PCLKT1_3				
CLKDIV2A	PCLKT2_0	PLL_URCA (S)	PLL_URCB (P)		
CLKDIV2B	PCLKT2_1	PLL_URCB (S)	DLL_URCC (S)		
CLKDIV2C	PCLKT2_2	DLL_URCC (P)	DLL_URCD (S)		
CLKDIV2D	PCLKT2_3	PLL_URCA (P)	DLL_URCD (P)		
CLKDIV4A	PCLKT4_0	DLL_LRCE (S)	PLL_LRCA (S)	PLL_LRCB (P)	DLL_LRCF (P)
CLKDIV4B	PCLKT4_1	DLL_LRCF (S)	PLL_LRCB (S)	DLL_LRCC (P)	
CLKDIV4C	PCLKT4_2	DLL_LRCC (S)	DLL_LRCD (P)		
CLKDIV4D	PCLKT4_3	PLL_LRCA (P)	DLL_LRCE (P)	DLL_LRCD (S)	
CLKDIV5A	PCLKT5_0	DLL_LLCE (S)	PLL_LLCA (S)	PLL_LLCB (P)	DLL_LLCF (P)
CLKDIV5B	PCLKT5_1	DLL_LLCF (S)	PLL_LLCB (S)	DLL_LLCC (P)	
CLKDIV5C	PCLKT5_2	DLL_LLCC (S)	DLL_LLCD (P)		
CLKDIV5D	PCLKT5_3	PLL_LLCA (P)	DLL_LLCE (P)	DLL_LLCD (S)	
CLKDIV7A	PCLKT7_0	PLL_ULCA (S)	PLL_ULCB (P)		
CLKDIV7B	PCLKT7_1	PLL_ULCB (S)	DLL_ULCC (S)		

Table 9. Valid CLKDIV Drivers

CLKDIV	CLKDIV Input Source				
CLKDIV7C	PCLKT7_2	DLL_ULCC (P)	DLL_ULCD (S)		
CLKDIV7D	PCLKT7_3	PLL_ULCA (P)	DLL_ULCD (P)		

Figure 33. PLL, DLL, CLKDIV, and Edge Clock Locations and Connectivity



1. This connection dot shows a connection from a PLL source to the CLKDIV.
2. This connection dot shows a connection from a DLL source to the CLKDIV.
3. PCLK inputs can drive an edge clock where indicated by an arrowhead, and certain ones can also drive the CLKDIV as indicated in Table 9.
4. These PCLK pins CANNOT drive CLKDIVs.

As shown in Figure 33 the specific PLLs, DLLs, and CLKDIV ELSRs can only drive specific edge clocks. Care must be taken when choosing a PLL, DLL, or CLKDIV to make sure there is no contention of edge clock resources. Table 10 is a tabular description of the information in Figure 33. Each row in the table represents the shared resources of a single edge clock.

Table 10. Edge Clock Shared Resources

Bank	Edge Clock	PLL/DLL	PLL/DLL	PIO	ELSR	General Routing/ Primary Clocks[1]
1	1	PLL_URCA.P		PCLKT1_1	CLKDIV1B.ELSR	2 connections
	2	PLL_URCA.S		PCLKT1_0	CLKDIV1A.ELSR	2 connections
	3	PLL_URCB.P		PCLKT1_3	CLKDIV1D.ELSR	2 connections
	4	PLL_URCB.S		PCLKT1_2	CLKDIV1C.ELSR	2 connections
	5		DLL_URCC.P	PCLKT1_4	CLKDIV1A.ELSR	2 connections
	6		DLL_URCC.S	PCLKT1_5	CLKDIV1B.ELSR	2 connections
	7		DLL_URCD.P	PCLKT1_6	CLKDIV1C.ELSR	2 connections
	8		DLL_URCD.S	PCLKT1_7	CLKDIV1D.ELSR	2 connections
2/3	1	PLL_URCA.P	PLL_LRCA.P	PCLKT2_1	CLKDIV2B.ELSR	2 connections
	2	PLL_URCA.S	PLL_LRCA.S	PCLKT2_0	CLKDIV2A.ELSR	2 connections
	3	PLL_URCB.P	PLL_LRCB.P	PCLKT2_3	CLKDIV2D.ELSR	2 connections
	4	PLL_URCB.S	PLL_LRCB.S	PCLKT2_2	CLKDIV2C.ELSR	2 connections
	5	DLL_LRCC.P	DLL_URCC.P	PCLKT3_0	CLKDIV2A.ELSR	2 connections
	6	DLL_LRCC.S	DLL_URCC.S	PCLKT3_1	CLKDIV2B.ELSR	2 connections
	7	DLL_LRCD.P	DLL_URCD.P	PCLKT3_2	CLKDIV2C.ELSR	2 connections
	8	DLL_LRCD.S	DLL_URCD.S	PCLKT3_3	CLKDIV2D.ELSR	2 connections
4	1	PLL_LRCA.P	DLL_LRCE.S	PCLKT4_1	CLKDIV4B.ELSR	2 connections
	2	PLL_LRCA.S	DLL_LRCE.P	PCLKT4_0	CLKDIV4A.ELSR	2 connections
	3	PLL_LRCB.P	DLL_LRCF.S	PCLKT4_3	CLKDIV4D.ELSR	2 connections
	4	PLL_LRCB.S	DLL_LRCF.P	PCLKT4_2	CLKDIV4C.ELSR	2 connections
	5		DLL_LRCC.P	PCLKT4_4	CLKDIV4A.ELSR	2 connections
	6		DLL_LRCC.S	PCLKT4_5	CLKDIV4B.ELSR	2 connections
	7		DLL_LRCD.P	PCLKT4_6	CLKDIV4C.ELSR	2 connections
	8		DLL_LRCD.S	PCLKT4_7	CLKDIV4D.ELSR	2 connections
5	1	PLL_LLCA.P	DLL_LLCE.S	PCLKT5_1	CLKDIV5B.ELSR	2 connections
	2	PLL_LLCA.S	DLL_LLCE.P	PCLKT5_0	CLKDIV5A.ELSR	2 connections
	3	PLL_LLCB.P	DLL_LLCF.S	PCLKT5_3	CLKDIV5D.ELSR	2 connections
	4	PLL_LLCB.S	DLL_LLCF.P	PCLKT5_2	CLKDIV5C.ELSR	2 connections
	5		DLL_LLCC.P	PCLKT5_4	CLKDIV5A.ELSR	2 connections
	6		DLL_LLCC.S	PCLKT5_5	CLKDIV5B.ELSR	2 connections
	7		DLL_LLCD.P	PCLKT5_6	CLKDIV5C.ELSR	2 connections
	8		DLL_LLCD.S	PCLKT5_7	CLKDIV5D.ELSR	2 connections
6/7	1	PLL_ULCA.P	PLL_LLCA.P	PCLKT7_1	CLKDIV7B.ELSR	2 connections
	2	PLL_ULCA.S	PLL_LLCA.S	PCLKT7_0	CLKDIV7A.ELSR	2 connections
	3	PLL_ULCB.P	PLL_LLCB.P	PCLKT7_3	CLKDIV7D.ELSR	2 connections
	4	PLL_ULCB.S	PLL_LLCB.S	PCLKT7_2	CLKDIV7C.ELSR	2 connections
	5	DLL_LLCC.P	DLL_ULCC.P	PCLKT6_0	CLKDIV7A.ELSR	2 connections
	6	DLL_LLCC.S	DLL_ULCC.S	PCLKT6_1	CLKDIV7B.ELSR	2 connections
	7	DLL_LLCD.P	DLL_ULCD.P	PCLKT6_2	CLKDIV7C.ELSR	2 connections
	8	DLL_LLCD.S	DLL_ULCD.S	PCLKT6_3	CLKDIV7D.ELSR	2 connections

1. There are two total connections per bank to edge clock routing from general routing or primary clocks.

Similarly, Table 11 is a tabular description of the primary clock shared resources. Each row in the table represents the shared resources of a single primary clock.

Table 11. Primary Clock Shared Resources

SERDES	PLL	DLL	PIO	CLKDIV	DCS
PCS36000.FF_SYSCLK_P1	PLL_ULCA.P			CLKDIV7A	
PCS36000.FF_RXCLK_P1	PLL_ULCA.S			CLKDIV7B	
PCS36000.FF_RXCLK_P2	PLL_ULCB.P			CLKDIV7C	
PCS36100.FF_SYSCLK_P1	PLL_ULCB.S			CLKDIV7D	
PCS36100.FF_RXCLK_P1		DLL_ULCC.S	PCLKT6_3		
PCS36100.FF_RXCLK_P2		DLL_ULCC.P	PCLKT7_3		
PCS36200.FF_SYSCLK_P1		DLL_ULCD.S	PCLKT6_2		
PCS36200.FF_RXCLK_P1		DLL_ULCD.P	PCLKT7_2		
PCS36200.FF_RXCLK_P2			PCLKT6_1		DCSLB
PCS36300.FF_SYSCLK_P1			PCLKT7_1		DCSLA
PCS36300.FF_RXCLK_P1		DLL_LLCE.S	PCLKT6_0		
PCS36300.FF_RXCLK_P2		DLL_LLCE.P	PCLKT7_0		
PCS36300.FF_RXCLK_P2		DLL_LLCF.S	PCLKT7_0		
PCS36300.FF_RXCLK_P1		DLL_LLCF.P	PCLKT6_0		
PCS36300.FF_SYSCLK_P1			PCLKT7_1		DCSLA
PCS36200.FF_RXCLK_P2			PCLKT6_1		DCSLB
PCS36200.FF_RXCLK_P1	PLL_LLCA.P		PCLKT7_2		
PCS36200.FF_SYSCLK_P1	PLL_LLCA.S		PCLKT6_2		
PCS36100.FF_RXCLK_P2	PLL_LLCB.P		PCLKT7_3		
PCS36100.FF_RXCLK_P1	PLL_LLCB.S		PCLKT6_3		
PCS36100.FF_SYSCLK_P1		DLL_LLCC.S			
PCS36000.FF_RXCLK_P2		DLL_LLCC.P			
PCS36000.FF_RXCLK_P1		DLL_LLCD.S			
PCS36000.FF_SYSCLK_P1		DLL_LLCD.P			
PCS3E000.FF_SYSCLK_P1	PLL_URCA.P			CLKDIV2A	
PCS3E000.FF_RXCLK_P1	PLL_URCA.S			CLKDIV2B	
PCS3E000.FF_RXCLK_P2	PLL_URCB.P			CLKDIV2C	
PCS3E100.FF_SYSCLK_P1	PLL_URCB.S			CLKDIV2D	
PCS3E100.FF_RXCLK_P1		DLL_URCC.S	PCLKT3_3		
PCS3E100.FF_RXCLK_P2		DLL_URCC.P	PCLKT2_3		
PCS3E200.FF_SYSCLK_P1		DLL_URCD.S	PCLKT3_2		
PCS3E200.FF_RXCLK_P1		DLL_URCD.P	PCLKT2_2		
PCS3E200.FF_RXCLK_P2			PCLKT3_1		DCSRA
PCS3E300.FF_SYSCLK_P1			PCLKT2_1		DCSRB
PCS3E300.FF_RXCLK_P1		DLL_LRCE.S	PCLKT3_0		
PCS3E300.FF_RXCLK_P2		DLL_LRCE.P	PCLKT2_0		
PCS3E300.FF_RXCLK_P2		DLL_LRCF.S	PCLKT2_0		
PCS3E300.FF_RXCLK_P1		DLL_LRCF.P	PCLKT3_0		
PCS3E300.FF_SYSCLK_P1			PCLKT2_1		DCSRB
PCS3E200.FF_RXCLK_P2			PCLKT3_1		DCSRA
PCS3E200.FF_RXCLK_P1	PLL_LRCA.P		PCLKT2_2		
PCS3E200.FF_SYSCLK_P1	PLL_LRCA.S		PCLKT3_2		

Table 11. Primary Clock Shared Resources (Continued)

SERDES	PLL	DLL	PIO	CLKDIV	DCS
PCS3E100.FF_RXCLK_P2	PLL_LRCB.P		PCLKT2_3		
PCS3E100.FF_RXCLK_P1	PLL_LRCB.S		PCLKT3_3		
PCS3E100.FF_SYSCLK_P1		DLL_LRCC.S			
PCS3E000.FF_RXCLK_P2		DLL_LRCC.P			
PCS3E000.FF_RXCLK_P1		DLL_LRCD.S			
PCS3E000.FF_SYSCLK_P1		DLL_LRCD.P			
			PCLKT5_3	CLKDIV5A	DCSBB
			PCLKT5_2	CLKDIV5B	DCSBA
			PCLKT5_1	CLKDIV5C	DCSBB
			PCLKT5_0	CLKDIV5D	DCSBA
			PCLKT4_3	CLKDIV4A	DCSBA
			PCLKT4_2	CLKDIV4B	DCSBB
			PCLKT4_1	CLKDIV4C	DCSBA
			PCLKT4_0	CLKDIV4D	DCSBB
			PCLKT1_3	CLKDIV1A	DCSTA
			PCLKT1_2	CLKDIV1B	DCSTB
			PCLKT1_1	CLKDIV1C	DCSTB
			PCLKT1_0	CLKDIV1D	DCSTA

PLLs, DLLs, and CLKDIVs can be located to a specific site using the LOCATE preference. Below is an example for each type.

```
LOCATE COMP "mypll/mypll_0_0" SITE "PLL_URCA";

LOCATE COMP "mydll/mydll_0_0" SITE "DLL_LLCC";

LOCATE COMP "myclkdiv" SITE "CLKDIV7A";
```

Preferred Input Pins

There are preferred pins for the PLL, DLL, CLKDIVs, primary clocks, and edge clocks.

Preferred pins provide the best routing for reduced jitter and clock injection delay. These preferred pins can be found in the LatticeSC/M Family Data Sheet pinout section under the Dual-Function column for a given package. All preferred clock pins can be differential or single ended.

I/O pins can be located to a specific pin using the LOCATE or IOBUF preference. Below is an example of each preferences.

```
LOCATE COMP "pll_in" SITE "D3";

IOBUF PORT "pll_in" IO_TYPE=LVDS SITE=D3;
```

There are two types of preferred pins, PLL/DLL pins and Primary/Edge/CLKDIV pins.

PLL/DLL Preferred Pins

Preferred pins for the PLLs and DLLs use dedicated routes to the inputs of the PLL and DLL. If the PLL or DLL is located via the LOCATE preference the ispLEVER place and route tools will select the preferred pin automatically if available. Likewise, if a preferred pin is selected as the input to a PLL or DLL the ispLEVER place and route tools will select the PLL or DLL associated with the preferred pin automatically. If the preferred pin for the PLL/DLL is not available an edge clock is the best second choice for performance.

Table 12 maps the data sheet preferred pins name to the site location of the PLL and DLL. These preferred pins for the CLKI and if necessary CLKFB input to the PLL and DLL. Only the true pin of each differential pair is shown in the table. For more pinout information, refer to the LatticeSC/M Family Data Sheet.

Table 12. Preferred PLL/DLL Pins

Preferred Pin Label	PLL CLKI	DLL CLKI
ULC_PLLT_IN_A	PLL_ULCA	
ULC_PLLT_IN_B	PLL_ULCB	
ULC_DLLT_IN_C		DLL_ULCC
ULC_DLLT_IN_D		DLL_ULCD
LLC_PLLT_IN_A	PLL_LLCA	
LLC_PLLT_IN_B	PLL_LLCB	
LLC_DLLT_IN_C		DLL_LLCC
LLC_DLLT_IN_D		DLL_LLCD
LLC_DLLT_IN_E		DLL_LLCE
LLC_DLLT_IN_F		DLL_LLCF
LRC_PLLT_IN_A	PLL_LRCA	
LRC_PLLT_IN_B	PLL_LRCB	
LRC_DLLT_IN_C		DLL_LRCC
LRC_DLLT_IN_D		DLL_LRCD
LRC_DLLT_IN_E		DLL_LRCE
LRC_DLLT_IN_F		DLL_LRCF
URC_PLLT_IN_A	PLL_URCA	
URC_PLLT_IN_B	PLL_URCB	
URC_DLLT_IN_C		DLL_URCC
URC_DLLT_IN_D		DLL_URCD

Primary, Edge, and CLKDIV Preferred Pins

Preferred pins for the primary clocks, edge clocks, and CLKDIVs provide the lowest latency routes into each of these elements.

Table 13 maps the data sheet preferred pin name to a primary clock, edge clock, or CLKDIV site. Only the true pin of each differential pair is shown in the table.

Table 13. Preferred Primary, Edge, and CLKDIV Pins

Preferred Pin Label	CLKDIV	Primary	Edge
PCLKT7_0	CLKDIV7A	Input	Bank 6/7
PCLKT7_1	CLKDIV7B	Input	
PCLKT7_2	CLKDIV7C	Input	
PCLKT7_3	CLKDIV7D	Input	
PCLKT6_[0:3]		Input	
PCLKT5_0	CLKDIV5A	Input	Bank 5
PCLKT5_1	CLKDIV5B	Input	
PCLKT5_2	CLKDIV5C	Input	
PCLKT5_3	CLKDIV5D	Input	
PCLKT5_[4:7]			
PCLKT4_0	CLKDIV4A	Input	Bank 4
PCLKT4_1	CLKDIV4B	Input	
PCLKT4_2	CLKDIV4C	Input	
PCLKT4_3	CLKDIV4D	Input	
PCLKT4_[4:7]			
PCLKT3_[0:3]		Input	Bank 3/2
PCLKT2_0	CLKDIV2A	Input	
PCLKT2_1	CLKDIV2B	Input	
PCLKT2_2	CLKDIV2C	Input	
PCLKT2_3	CLKDIV2D	Input	
PCLKT1_0	CLKDIV1A	Input	Bank 1
PCLKT1_1	CLKDIV1B	Input	
PCLKT1_2	CLKDIV1C	Input	
PCLKT1_3	CLKDIV1D	Input	
PCLKT1_[4:7]			

Clocking Timing Constraints (Preferences)

Clocks in the LatticeSC architecture need to be constrained for ispLEVER to implement the design to the desired performance. The following is a list of preferences that can be used to constrain clocks. For additional information on using these preferences and the Preference Editor see the ispLEVER Help System.

Note that PLLs and DLLs automatically produce frequency preferences for their outputs which will be added to the preference file after the map process.

FREQUENCY – Specifies the minimum operating frequency for all sequential output to sequential input elements clocked by the specified clock.

```
FREQUENCY port "clkin" 200 MHz;

FREQUENCY net "rxclk" 156 MHz;
```

PERIOD – Specifies the minimum period for all sequential output to sequential input elements clocked by the specified clock. Exactly the same as the FREQUENCY preference but uses ns instead of MHz.

```
PERIOD port "clkin" 20 ns;

PERIOD net "rxclk" 6.41ns;
```

MULTICYCLE – Allows for relaxation of previously defined PERIOD or FREQUENCY constraints on a path.

```
MULTICYCLE from cell "block_a*" to cell "block_b*" 2x;
```

CLKSKEWDIFF – Allows the user to specify the clock skew between two top-level input clocks. This skew will then be used by ispLEVER when timing the clock domain transfers between the two clocks.

```
CLKSKEWDIFF CLKPORT "clk1" CLKPORT "clk2" 2ns;
```

Modifying the PLL/DLL through Advanced Preference Support

The PLL and DLL have several features that can be controlled by preferences. This allows the behavior of the PLL and DLL to change without having to regenerate the module. Preference changes to the PLL and DLL must be included in the preference file prior to the map process. The Preference Editor can be used to change several of these features.

The ASIC preference is used to modify non-FPGA based elements. The type of the element needs to be defined in the ASIC preference. This type is the name of the library element used for the element. The PLL always uses the EHXPLLA type. The DLL uses four different library elements depending on the mode of operation. The library element name can be found in the <module_name>.[v|vhd] file.

The following is a list of features that can be controlled via preferences. These features can also be changed in EPIC.

Divider Settings – The output dividers can be changed for the PLL CLKOP/CLKOS and the DLL CLKI/CLKOS. For the PLL the output divider can only be changed if that particular output is not the source of the CLKFB.

```
ASIC "pll/pll_0_0" TYPE "EHXPLLA" CLKOP_DIV=4 CLKOS_DIV=8;

ASIC "dll/dll_0_0" TYPE "CIMDLLA" CLKOS_DIV=2 CLKI_DIV=2;
```

Table 14 provides a list of divider attributes and possible values. Note for the DLL, if CLKOS is used as the feedback clock then the CLKI_DIV and CLKOS_DIV must be equal.

Table 14. Divider Attributes

Attribute	Values
PLL	
CLKOP_DIV	1-64
CLKOS_DIV	1-64
DLL	
CLKI_DIV	1, 2, 4
CLKOS_DIV	1, 2, 4

CLKOS VCO Delay: The CLKOS port supports a VCO delay which allows the CLKOS port to be delayed a specific number of VCO clock cycles. This is accomplished by holding the CLKOS output port in reset for the number of specified VCO clock cycles after a reset or configuration. CLKOS can be delayed 0 to 31 VCO clock cycles.

```
ASIC "pll/pll_0_0" TYPE "EHXPLLA" CLKOS_VCODEL=4;
```

Programmable Delay Settings: The PLL has programmable delays that can be added to certain ports. The PLL can add programmable fixed delay to the CLKI, CLKFB, and CLKOS ports.

The PLL uses coarse delays (PDEL) and fine delays (FDEL). For a range of values for each PDEL value see the LatticeSC/M Family Data Sheet. Table 15 provides a list of delay attributes and possible values. For ports that support both coarse and fine delays the delay is the summation of both.

Remember, to use the PLL CLKOS fine delay the CLKOS output divider path needs to be used.

```
ASIC "pll/pll_0_0" TYPE "EHXPLLA" CLKI_PDEL=DEL0 CLKFB_PDEL=DEL1;

ASIC "pll/pll_0_0" TYPE "EHXPLLA" CLKOS_FDEL=DEL0 CLKOS_MODE=DIV CLKOP_MODE=DIV;
```

Table 15. Fixed Delay Attributes

Attribute	Values
CLKI_PDEL	DEL0, DEL1, DEL2, DEL3
CLKI_FDEL	100, 200, ..., 700
CLKFB_PDEL	DEL0, DEL1, DEL2, DEL3
CLKFB_FDEL	100, 200, ..., 700
CLKOS_FDEL	100, 200, ..., 700

Note. FDEL values are an approximation of ps.

In addition to the programmable fixed delays the PLL CLKOS output also supports a VCO delay which allows the clock to be delayed by a specific number of VCO clock cycles. The attribute CLKOS_VCODEL has possible values of 0, 1, 2, 3, ...,31. Note that this delay works in increments of the VCO clock period, not CLKOS period.

```
ASIC "pll/pll_0_0" TYPE "EHXPLLA" CLKOS_VCODEL=4;
```

CLKOP vs. CLKOS Phase Adjustment: The PLL and DLL can produce a fixed phase relationship between the CLKOP and CLKOS outputs.

The DLL only provides phase offsets between CLKOP and CLKOS in Time Reference Delay mode. There are three attributes that can be adjusted as shown in Table 16. The CLKOS_PHASE and CLKOS_FPHASE are cumulative to create the total phase offset for CLKOS.

Table 16. DLL Phase Adjustment Attributes

Attribute	Values (degrees)
CLKOP_PHASE	0, 90, 180, 270, 360
CLKOS_PHASE	0, 90, 180, 270, 360
CLKOS_FPHASE	0, 11, (for 11.25), 22 (for 22.5), 45

The PLL uses the PHASE_ADJ single keyword to control the CLKOP to CLKOS phase offset. PHASE_ADJ has possible values of 0, 45, 90, etc. The PHASE_ADJ attribute controls the phase offset at the output of the VCO. If the CLKOP or CLKOS output is divided down using the output dividers this will change the ultimate phase offset at the output of the PLL.

Note that for both the PLL and DLL phase changes on output ports that are used as the CLKFB source are not valid.

Spread Spectrum: The PLL spread spectrum capabilities can be enabled and the down spread drift set via a preference. The SPREAD keyword enables or disables the spread spectrum feature. Valid values are ENABLED/DISABLED. The SPREAD_DRIFT keyword sets the down-spread drift percentage. Valid values are 1, 2, and 3. The output frequency will vary in a range that is below its nominal value down to a frequency that is 1%, 2%, or 3% lower than nominal. This option uses values of 0.5%, 1.0%, and 1.5% in the front-end GUI referring to the average frequency. A 0.5% selection yields SPREAD_DRIFT=1 for example.

```
ASIC "pll/pll_0_0" TYPE "EHXPLLA" SPREAD=ENABLED SPREAD_DRIFT=1;
```

GSR: The PLL and DLL can be reset by the GSR if enabled. The GSR keyword can be set to ENABLED/DISABLED.

```
ASIC "pll/pll_0_0" TYPE "EHXPLLA" GSR=DISABLED;
```

PLL/DLL Lock Time Control

The PLL and DLL will lock when the CLKI and CLKFB phases are aligned. In a simulation environment the lock time has a fixed delay of 100µs. This value can be changed through an HDL parameter or preference (for the back annotation simulation). The PLL/DLL contains a parameter named LOCK_DELAY which accepts an integer value for the total time in us until the lock output goes high. Below is an example of how to set this value for front-end simulation.

Verilog

```
defparam mydll.mypll_0_0.LOCK_DELAY=500;
mydll dll_inst(.CLKI(clkin), .CLKOP(clk1), .CLKOS(clk2),
```

VHDL

Not supported

For back annotation simulation LOCK_DELAY needs to be set in the preference file. Below is an example for the PLL.

```
ASIC "pll/pll_0_0" TYPE "EHXPLLA" LOCK_DELAY=200;
```

Serial Management Interface (SMI)

The PLL and DLL can connect to the system bus via the serial management interface (SMI). The SMI allows a subset of the PLL and DLL features to be modified during run time. The DLL must be created in Time Reference Delay mode to use this capability. The SMI interface is a broadcast protocol that is driven from the system bus or from FPGA logic and runs at a maximum of 50 MHz. The SMI bus is connected via FPGA routing to all of the SMI targets that are instantiated in the FPGA design. Each target is given an offset address. When a user accesses the system bus at this specific offset address the SMI bus will transact with the given target. It is also possible to given several SMI targets the same offset address to broadcast data to all of them. For more information on the SMI protocol and the System Bus see TN1085, LatticeSC MPI/System Bus.

When connecting the SMI of the PLL or DLL to the system bus, all of the connections can be directly connected and the system bus will handle all of the protocol timing. If the user is creating their own SMI master interface to drive the PLL or DLL's target interface, the protocol for the SMI needs to be handled by the user. Information on the specifics of the SMI can be found in TN1085, LatticeSC MPI/System Bus. Note that the smi_rstn port of the SMI needs to toggled low once the bistream has been loaded. When connected to the system bus this is handled by the release of the GSR. For a user's master interface the smi_rstn port needs to be toggled low after the release of GSR for a few SMI clock cycles.

The PLL and the DLL each have a memory map that describes registers accessible from the SMI. The PLL and DLL are first configured via the FPGA bitstream. Once the configuration is loaded into the FPGA the registers available on the SMI bus are loaded with the configuration set in the bitstream. A subset of these settings can now be changed at run time. For example, the phase offset can be changed as well as the output dividers.

The SMI capabilities of the PLL and DLL are not supported in simulation. SMI transactions can be simulated, but the PLL and DLL will not respond to any changes during simulation.

The user must set the SMI offset address for the PLL or DLL. Below are examples for setting the PLL target address to 0x410 in the preference file.

```
ASIC "pll/pll_0_0" TYPE "EHXPLLA" SMI_OFFSET="0x410";
```

When making modifications to the PLL or the DLL a reset must be issued following the write transaction. The reset must be performed via the RSTN port or GSR. Be sure to include a RSTN port on the PLL or DLL when creating the module if the SMI interface will be utilized.

Table 17 provides the memory map for the PLL. This memory map begins at the base address specified by the SMI_OFFSET value.

Table 17. PLL Memory Map

Register	Bits	Description
0x0	[0:7]	Reserved (Do not change these bits)
0x1	[0:7]	Reserved (Do not change these bits)
0x2	[0:1]	CLKOS Output Mode (refer to Figure 7) 00 - Bypass VCO (CLKI to CLKOS) 01 - Fine Delay Only - The clock path will go from the VCO through the fine delay block and to the CLKOS port. The divider will be bypassed. 10 - VCO - The clock path will go from the VCO directly to the CLKOS port. The fine delay and divider will be bypassed. 11 - Divider and Fine Delay - The clock path will go through both the fine delay and divider.
	[2:7]	CLKOS Divider Can only be changed if CLKOS is not the source of CLKFB. Assert RSTN after change to reset divider. 000000 - Divide by 1 100000 - Divide by 2 010000 - Divide by 3 ... 011111 - Divide by 63 111111 - Divide by 64
0x3	[0:1]	CLKOP Output Mode (refer to Figure 7) 00 - Bypass VCO (CLKI to CLKOS) 01 - CLKOS Fine Delay Match Only - The clock path will go from the VCO through the CLKOS fine delay matching block and to the CLKOP port. This is to ensure the phase relationship between the CLKOS and CLKOP. The divider will be bypassed. 10 - VCO - The clock path will go from the VCO directly to the CLKOP port. The fine delay and divider will be bypassed. 11 - Divider and Fine Delay Match - The clock path will go through both the fine delay match and divider.
	[2:7]	CLKOP Divider Can only be changed if CLKOP is not the source of CLKFB. Assert RSTN after change to reset divider. 000000 - Divide by 1 100000 - Divide by 2 010000 - Divide by 3 ... 011111 - Divide by 63 111111 - Divide by 64

Table 17. PLL Memory Map (Continued)

Register	Bits	Description
0x4	[0:2]	CLKI Fine Delay 000 to 0ps 100 to 100 ps 010 to 200 ps 110 to 300 ps 001 to 400 ps 101 to 500 ps 011 to 600 ps 111 to 700 ps
	[3:5]	CLKFB Fine Delay 000 to 0ps 100 to 100 ps 010 to 200 ps 110 to 300 ps 001 to 400 ps 101 to 500 ps 011 to 600 ps 111 to 700 ps
	[6:7]	Reserved (Do not change these bits)
0x5	[0:2]	CLKOS Fine Delay 000 to 0ps 100 to 100 ps 010 to 200 ps 110 to 300 ps 001 to 400 ps 101 to 500 ps 011 to 600 ps 111 to 700 ps
	[3:5]	Phase Offset. CLKOP will lead CLKOS by this amount. Can only be changed if CLKOS is not the source of CLKFB. The Phase Offset is the offset at the output of the VCO. If the CLKOP and CLKOS output is divided down by the output dividers specified in the netslist, the phase offset changes at the output of the PLL. 000 to 0 degrees 100 to 45 degrees 010 to 90 degrees 110 to 135 degrees 001 to 180 degrees 101 to 225 degrees 011 to 270 degrees 111 to 315 degrees
	[6:7]	Reserved (Do not change these bits)
0x6	[0]	CLKOS VCO Delay Enable 1 - Enable VCO Delay 0 - No additional VCO Delay
	[1:6]	VCO Delay Control. Assert RSTN after change to any of these bits. 000000 - CLKOS aligned with CLKOS 100000 - CLKOS delayed by 1 VCO clock 010000 - CLKOS delayed by 2 VCO clocks ... 011111 - CLKOS delayed by 30 VCO clocks 111111 - CLKOS delayed by 31 VCO clocks
	[7]	Reserved (Do not change this bit)
0x7	[0:7]	Reserved (Do not change these bits)

Table 17. PLL Memory Map (Continued)

Register	Bits	Description
0x8	[0:1]	Reserved (Do not change these bits)
	[2]	Spread Spectrum Enable 0 - Disabled 1 - Enabled
	[3:7]	Reserved (Do not change these bits)
0x9	[0:7]	Reserved (Do not change these bits)
0xa	[0:7]	Reserved (Do not change these bits)
0xb	[0]	Power down the PLL 1 - Powered Up 0 - Powered Down
	[1:7]	Reserved (Do not change these bits)

Table 18 provides the memory map for the DLL. This memory map begins at the base address specified by the SMI_OFFSET value.

Table 18. DLL Memory Map

Register	Bits	Description
0x0	[0:4]	Reserved (Do not change these bits)
	[5:7]	CLKOP Phase Shift. CLKOP will be delayed by this amount 000 - 90 degrees 100 - 180 degrees 010 - 270 degrees 110 - 360 degrees 001 - 0 degrees
0x1	[0:4]	Reserved (Do not change these bits)
	[5:7]	CLKOS Phase Shift. CLKOS will be delayed by this amount if the Force Static Delay (register 0x3 bit 6) is set to 0x1. 000 - 90 degrees 100 - 180 degrees 010 - 270 degrees 110 - 360 degrees 001 - 0 degrees CLKOS will be delayed by this amount if the Force Static Delay (register 0x3 bit 6) is set to (0x0). 000 - 4 delays 100 - 3 delays 010 - 2 delays 110 - 1 delay 001 - 0 delays
0x2	[0:1]	Reserved (Do not change these bits)
	[2:3]	CLKOS Fine Phase Shift. Added to CLKOS Phase Shift to create total delay if 0x3 bit 5 is set. 00 - 0 degrees 01 - 45 degrees 10 - 22.5 degrees 11 - 11.24 degrees
	[4:5]	Reserved (Do not change these bits)
	[6:7]	CLKOS Divider 00 - Divide by 1 10 - Divide by 2 01 - Divide by 4 11 - 50% Duty Cycle Correction

Table 18. DLL Memory Map (Continued)

Register	Bits	Description
0x3	[0:4]	Reserved (Do not change these bits)
	[5]	CLKOS Fine Phase Shift Enable 0 - No fine phase shift 1 - Add fine phase shift (register 0x2 bits [2:3]) to CLKOS Phase shift
	[6]	Force Static Delay 0 - ALU controlled 1 - Static Delay (see register 0x6 bits [0:7])
	[7]	Hold
0x4	[0:7]	Reserved (Do not change these bits)
0x5	[0:7]	Reserved (Do not change these bits)
0x6	[0:7]	Delay setting. Only valid if the Force Static Delay (register 0x3 bit 6) is set to 0x1. Delay setting = (0x6 [0:7] register setting) * t_{FDEL} where 0 ð (0x6 [0:7] register setting) ð 0x8F (decimal 143). Note: The specification of t_{FDEL} is located in the DC and Switching Characteristics section of the LatticeSC/M Family Data Sheet.
0x7	[0:7]	Reserved (Do not change these bits)
0x8	[0:6]	Adjust DCNTL phase shift to INDEL by the number of t_{FDEL}s specified. Note: The specification of t_{FDEL} is located in the DC and Switching Characteristics section of the LatticeSC/M Family Data Sheet).
	[7]	0 - Add value to DCNTL phase shift 1 - Subtract value from DCNTL phase shift
0x9	[0:7]	Reserved (Do not change these bits)

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
February 2006	01.0	Initial release.
January 2007	01.1	Added additional information on the PLL Lock operation.
		Added information on cascading DLLs to PLLs.
June 2007	01.2	Updated DLL to PLL diagram.
		Added Cascading PLLs section and related diagram.
		Updated PLL Memory Map text (Register 0x2, Bits [2:7] and Register 0x3, Bits [2:7]).
August 2007	01.3	Added Primary Clock Shared Resources table.
October 2007	01.4	Updated DLL Memory Map
		Updated Clock Phase Adjustment PLL CLKOP Leads CLKOS diagram.
November 2007	01.5	Corrected references for DLL SMI offset register. Corrected OSCA Instantiation in VHDL example. Modified paragraph below Figure 33. Added table for edge clock resource sharing. Corrected Table 16, entries 0x3 through 0x8.
February 2008	01.6	Updated PLL, DLL, CLKDIV, and Edge Clock Locations and Connectivity diagram.
		Updated Edge Clock Shared Resources table.
July 2008	01.7	Updated Serial Management Interface (SMI) text section.
		Updated PLL Memory Map table, bits [0:1] for registers 0x2 and 0x3.
March 2009	01.8	Updated Lock Output text section.
		Added PLL LOCK Circuit diagram.
September 2009	01.9	Updated Edge Clock Shared Resources table.
June 2010	02.0	Updated PLL, DLL, CLKDIV, and Edge Clock Locations and Connectivity figure.
July 2012	02.1	Document updated with new corporate logo.
		Added Valid CLKDIV Drivers table.
		PLL, DLL, CLKDIV, and Edge Clock Locations and Connectivity diagram – Updated footnotes.