LatticeECP3 sysCLOCK PLL/DLL Design and Usage Guide

September 2012 Technical Note TN1178

Introduction

This technical note describes the clock resources available in the LatticeECP3™ device architecture. Details are provided for primary clocks, secondary clocks, edge clocks, and general routing, as well as clock elements such as PLLs, DLLs, clock dividers and more.

The number of PLLs, DLLs and DDR-DLLs for each device is listed in Table 10-1.

Table 10-1. Number of PLLs, DLLs and DDR-DLLs

Parameter	Description	ECP3-17	ECP3-35	ECP3-70	ECP3-95	ECP3-150
Number of GPLLs	General purpose PLL	2	4	10	10	10
Number of DLLs	General Purpose DLL	2	2	2	2	2
Number of DDR-DLLs	DLL for DDR applications	2	2	2	2	2

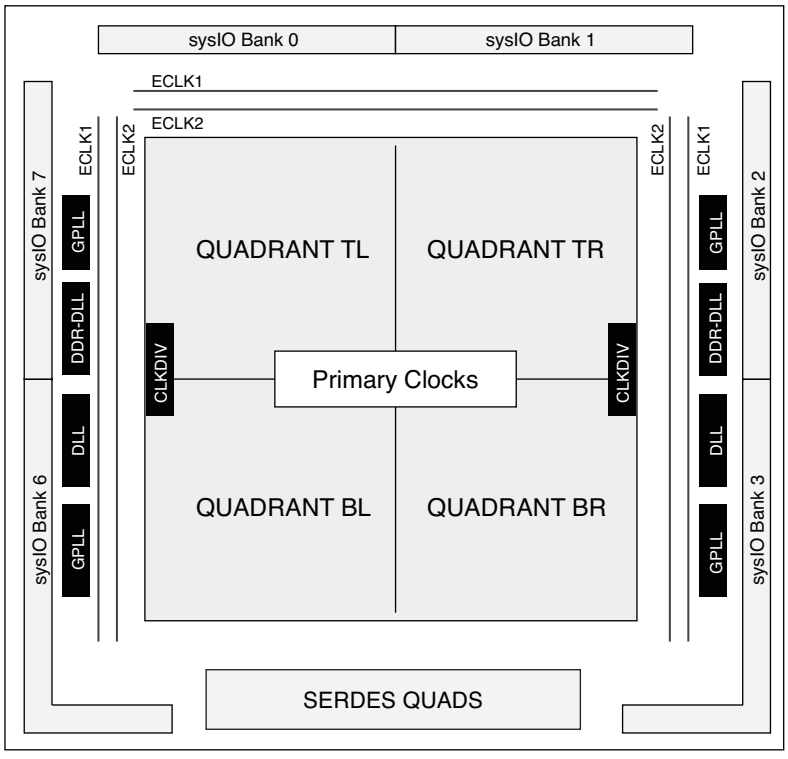
Clock/Control Distribution Network

LatticeECP3 devices provide global clock distribution in the form of eight quadrant-based primary clocks and flexible secondary clocks. Two edge clocks are also provided on the left, right and top edges of the device. Other clock sources include clock input pins, general logic, PLLs, DLLs, DCSs, and clock dividers.

LatticeECP3 Top-Level View

Figure 10-1 provides a view of the primary clocking structure of the LatticeECP3-35 device.

Figure 10-1. LatticeECP3 Clocking Structure (LatticeECP3-35)



Primary Clocks

The LatticeECP3 architecture provides eight primary global clock nets, all eight of which are available to all quadrants; however, each quadrant can have any of these eight nets independent of the others, or they can be tied to other vertically or horizontally adjacent quadrants. Two of these clocks provide the Dynamic Clock Selection (DCS) feature. The six primary clocks without DCS can be specified in the Pre-map Preference Editor as 'Primary Pure' and the two DCS clocks as 'Primary-DCS'.

The sources of primary clocks include the following (refer also to Figure 10-35).

- GPLL outputs
- DLL outputs
- PCS TX CLKs
- CLKDIV outputs
- PCLK PIOs

Secondary Clocks

The LatticeECP3 secondary clocks are a flexible region-based clocking resource. A region is an area over which a secondary clock operates. Each region has eight possible sources to drive secondary clock routing. Note that secondary/regional clock net boundaries do not always coincide with primary/quadrant clock net boundaries; for example, the ECP3-17 has three rows of regions, but two rows of quadrants.

There are eight secondary clock muxes per region. Each mux has inputs from eight different sources. Seven of these are from internal nodes. The eighth input comes from a primary clock pin. The input sources are not necessarily located in the same region as the mux. This structure enables global use of secondary clocks.

The sources of secondary clocks are:

- Dedicated PCLK clock pins:
  - PCLKT0_0
  - PCLKT1_0
  - PCLKT2_0
  - PCLKT3_0
  - PCLKT6_0
  - PCLKT7_0
- Internal nodes

Table 10-2 lists the number of secondary clock regions in LatticeECP3 devices and Figure 10-2 shows how the secondary clocks are organized into regions for a typical LatticeECP3 device.

Table 10-2. Number of Secondary Clock Regions

Parameter	ECP3-17	ECP3-35	ECP3-70	ECP3-95	ECP3-150
Number of regions	16	16	20	20	36
Layout (Vertical x Horizontal)	4x4	4x4	5x4	5x4	6x6

Edge Clocks

The LatticeECP3 device has two Edge Clocks (ECLK) each on the left, right, and top sides of the device. There are no edge clocks located on the bottom side of the device because of the SERDES blocks located there. These clocks, which have low injection time and skew, are used to clock I/O registers. Edge clock resources are designed for high speed I/O interfaces with high fanout capability. In the list of edge clock sources, shown below, there is a

special provision made for signals from the left-side PLLs and DLLs to feed all edge clock banks. This is especially useful in DDR applications which allow I/Os to be distributed across three edges of the device. Refer to Appendix B for detailed information on ECLK locations and connectivity.

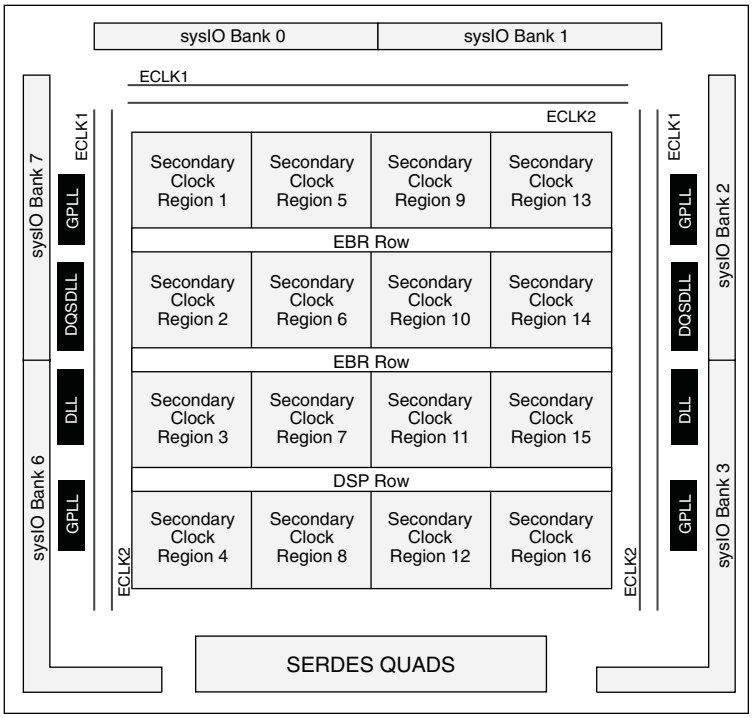
The sources of edge clocks are:

- **For Left and Right Edge Clocks:**
  - Dedicated clock pins
  - GPLL outputs:
    - LatticeECP-35: Both PLLs on each side can drive that same side's edge clock
    - Others: Second and third PLL on each side can drive that same side's edge clock
  - GPLL input pins
  - DLL outputs
  - LEFT side top PLL and DLL can drive both sides ("bridging" capability)
  - Internal nodes
- **For Top Edge Clocks:**
  - Dedicated clock pins
  - Top left or top right GPLL outputs
  - LEFT side top PLL and DLL ("bridging" capability)
  - Internal nodes

Edge clocks can directly drive the secondary clock resources and general routing resources. Refer to Figure 10-36 for detailed information on edge clock routing. The edge clocks on the left and right edges can drive the primary clock resources through the CLKDIV blocks.

Figure 10-2 shows the structure of the secondary clocks and edge clocks for a typical device.

Figure 10-2. LatticeECP3 Secondary Clocks and Edge Clocks (ECP3-35)



General Routing for Clocks

The LatticeECP3 architecture supports the ability to use the general routing, normally used for data routing, as a clock resource. This resource is intended to be used for small areas of the design to allow additional flexibility in linking dedicated clocking resources and building very small clock trees.

The general routing of the LatticeECP3 is optimized for data routing. Clocks can be routed on this resource, but will not have the same performance as the dedicated clocking resources. Due to the large amount of connectivity the place and route of this resource will not have as tight a skew as the dedicated clocking resources. For this reason it is best to limit the distance of a general routing based clock as well as the number of loads.

Additional Connectivity for Dedicated Clock Resources

The dedicated clocking resources in the LatticeECP3 all have preferred entry points for the clock signal. Sometimes these entry points cannot be used by the user. To allow flexible connectivity to the dedicated resources the general routing can also be used an entry point. This is useful when creating a clock gate or clock MUX implemented in FPGA LUTs.

Figures 10-3 and 10-4 provide examples of using general routing to bridge a clock source to a dedicated clock resource. Figure 10-3 uses a LUT such as would be used for a clock gate or MUX. Figure 10-4 uses a PIO, or FPGA I/O input pin, to bridge to a dedicated clock resource. This is the structure that would be found if a non-preferred input pin was selected.

Figure 10-3. General Routing Connection to Dedicated Clock Resource from LUT

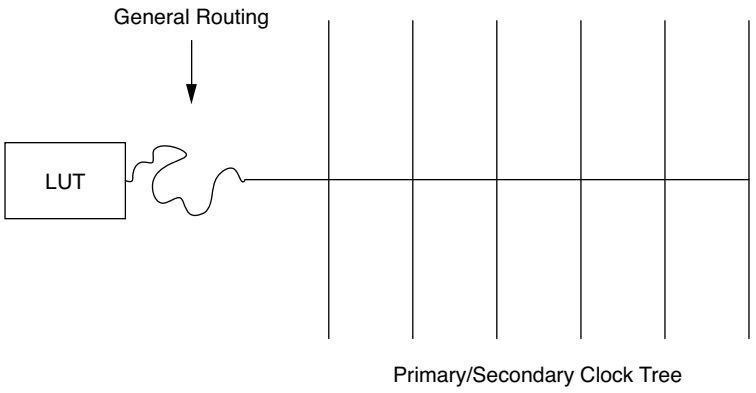
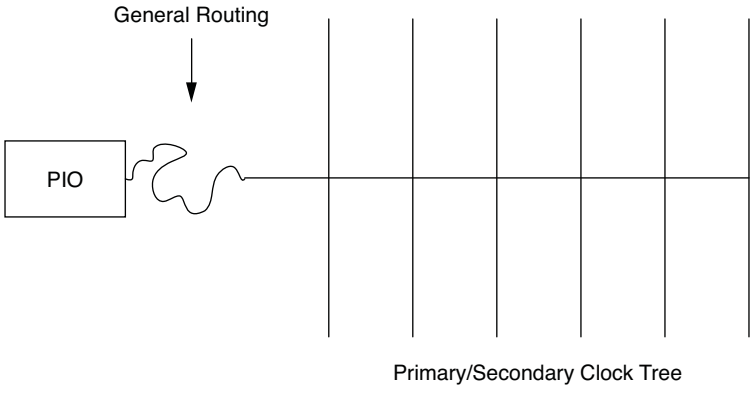


Figure 10-4. General Routing Connection to Dedicated Clock Resource from PIO



Note: General Routing cannot be used in the source path for edge clocks. Edge clocks are high-speed resources which require clean and duty-cycle balanced sources. The Place and Route software in Lattice Diamond® will not allow general routing in the source path to edge clocks.

When using the general routing to reach a dedicated resource the place and route process will issue a warning. Below is an example warning message.

```
WARNING - par: Regional clock signal "myclk" drives a PIO comp.
          Generic routing may have to be used to route to the PIO load of
          this regional clock.
```

These warnings are intended to alert the user that general routing is being utilized to route a portion of the clock before it arrives at a dedicated resource.

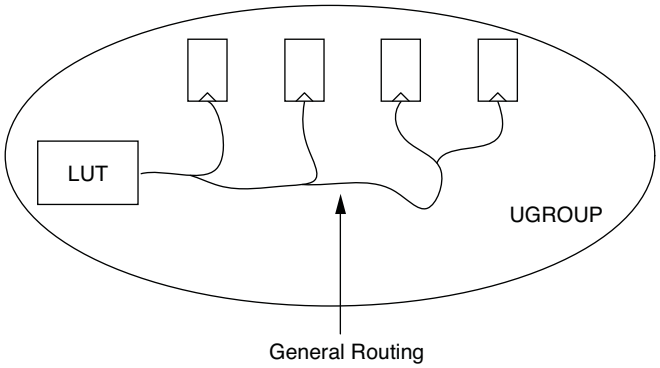
Very Small Clock Domains

General routing can be used to make very small clock domains. As discussed previously, the general routing requires a large number of connections to allow flexible data routing. These additional connections will increase the skew that can occur when implementing clocks. Due to this skewed reality of the general routing it is best to limit the number of loads on the clock.

Small clock domains should be grouped together using a UGROUP preference (see the **Diamond Help >Constraints Reference Guide > Preferences > UGROUP**). A UGROUP preference is a placement constraint which Place & Route will utilize to place all of the components in the group close together. This constraint will attempt to keep the routing distance to a minimum and thereby reduce the amount of clock skew that can occur between the destinations.

Figure 10-5 shows an example of using general routing for a small clock domain.

Figure 10-5. Small Clock Domain Example with General Routing



Static Timing Analysis of General Routing Clocks

All LatticeECP3 designs require the user to run static timing analysis using the Trace process. When using general routing for clocks it is necessary to generate both a setup and hold time Trace report. These setup and hold checks will ensure that any skew induced by the general routing connections will be accounted for by the timing tools and reported to the user.

In the Diamond Strategy for Place & Route Trace make sure the Analysis Option is set to “Standard Setup and Hold Analysis”. This option will create a Trace report which includes both the setup and hold times of the requested number of worst-case paths per preference. The user should examine the general routing based clocks carefully to ensure that these paths meet their timing preferences.

Specifying Clocks in the Design Tools

If desired, designers can specify the clock resources, primary, secondary or edge to be used to distribute a given clock source. Figure 10-6 illustrates how this can be done in the ispLEVER® Design Planner Spreadsheet View (or in Spreadsheet View in the Lattice Diamond design software). Alternatively, the resources can be specified by using corresponding preferences in the preference file.

Global Primary Clock and Quadrant Primary Clock

Global Primary Clock

If a primary clock is not assigned as a quadrant clock, the software assumes it is a global clock.

There are six Global Primary/Pure clocks and two Global Primary/DCS clocks available.

Primary-Pure and Primary-DCS

Primary Clock Net can be assigned to either Primary-Pure (CLK0 to CLK5) or Primary-DCS (CLK6 and CLK7).

Quadrant Primary Clock

Any primary clock may be assigned to a quadrant clock. The clock may be assigned to a single quadrant or to two adjacent quadrants (not diagonally adjacent).

When a quadrant clock net is used, the user must ensure that the registers each clock drives can be assigned in that quadrant without any routing issues.

In the quadrant primary clocking scheme, the maximum number of primary clocks is 32. Note, however, that these are not global primary clocks; the maximum number of global primary clocks is eight.

Primary quadrant clocks can be preferenced in the .lpf file by using this format:

```
USE PRIMARY [PURE | DCS]  [NET | PORT “<net or port name>” [quadrant_type] ];
```

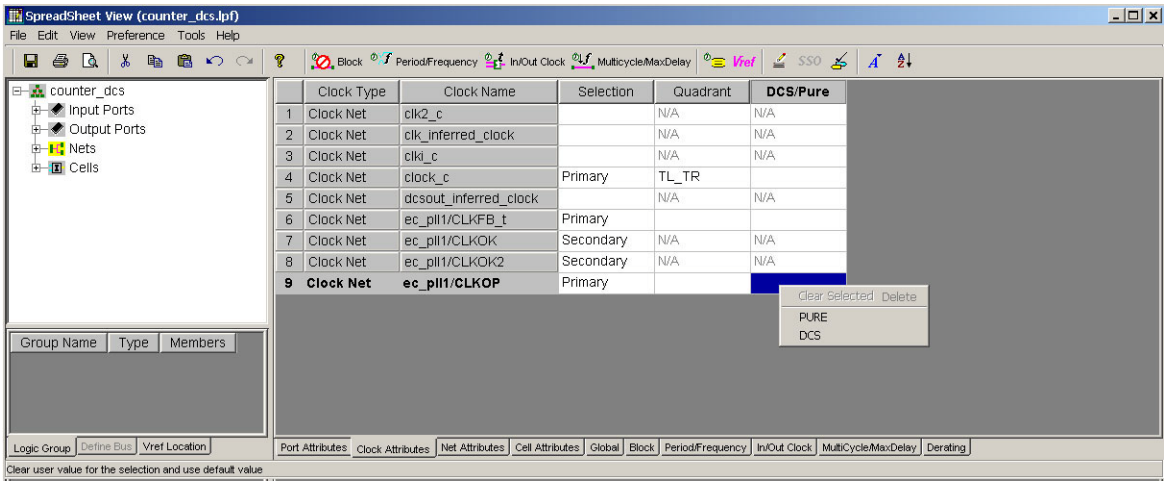
where:

<net or port name> = specific clock net name <string>
[quadrant_type] ::= [QUADRANT_TL | QUADRANT_TR | QUADRANT_BL | QUADRANT_BR]
QUADRANT_TL ::= Top left corner of the FPGA
QUADRANT_TR ::= Top right corner of the FPGA
QUADRANT_BL ::= Bottom left corner of the FPGA
QUADRANT_BR ::= Bottom right corner of the FPGA

One or two quadrants can be specified, for example:

```
USE PRIMARY NET “clk” QUADRANT_TL QUADRANT_TR;
```

Figure 10-6. ispLEVER Design Planner Spreadsheet View (see Appendix C Figure 10-38 for Diamond Equivalent)



Refer to "Appendix A. Primary Clock Sources and Distribution" on page 40 for detailed clock network diagrams.

Global Secondary Clock and Regional Secondary Clocks

Global Secondary Clocks

Secondary clocking is regional. However, if no secondary regions are defined, software will combine all secondary clock regions into a signal global secondary clock structure comprising up to eight secondary clock/CE/LSR sources. If nets are not assigned to secondary regions, the software will automatically assign up to eight clock/CE/LSR nets to use secondary resources in this global secondary clock structure.

Regional Secondary Clocks

Secondary clock resources are regional, meaning there are different clock regions within the device as a whole. Each LatticeECP3 device has a different number of regions based on the size. Below is a basic diagram of the secondary clock region resources for each LatticeECP3.

LatticeECP3-17K / 35K:

R1C1	R1C	R1C3	R1C4
R2C1	R2C2	R2C3	R2C4
R3C1	R3C2	R3C3	R3C4
R4C1	R4C2	R4C3	R4C4

LatticeECP3-70K / 95K:

R1C1	R1C2	R1C3	R1C4
R2C1	R2C2	R2C3	R2C4
R3C1	R3C2	R3C3	R3C4
R4C1	R4C2	R4C3	R4C4
R5C1	R5C2	R5C3	R5C4

LatticeECP3-150K:

R1C1	R1C2	R1C3	R1C4	R1C5	R1C6
R2C1	R2C2	R2C3	R2C4	R2C5	R2C6
R3C1	R3C2	R3C3	R3C4	R3C5	R3C6
R4C1	R4C2	R4C3	R4C4	R4C5	R4C6
R5C1	R5C2	R5C3	R5C4	R5C5	R5C6
R6C1	R6C2	R6C3	R6C4	R6C5	R6C6

Secondary Region Clock Preferencing

In order to use secondary clock regions, the user must create a secondary clock region, then preference their clock to it.

To create a region, the format is:

```
REGION "<region name>" CLKREG "CLKREG_R1C1" <# of Columns to Span> <# of Rows to Span>;
USE SECONDARY NET "<clock net>" <region name>;
```

where:

<region name> = A unique region name given to each region preference.

<clock net> = Specific clock/CE/LSR name.

"CLKREG_R1C1" = The row/column starting point as defined above for each device.

<# of Columns to Span> = Number of columns deep the secondary clock region will encompass. It can be from 1 to 6, depending on which device is used.

<# of Rows to Span> = Number of rows wide the secondary clock region will encompass. It can be from 1 to 6, depending on which device is used.

For example, we will create a region in a LatticeECP3-150K device and assign a clock to it.

```
REGION "D_QUAD_R" CLKREG "CLKREG_R1C4" 5 3;

USE SECONDARY NET "clk1" D_QUAD_R;
```

This will create a secondary clock region that is five columns deep, by three rows wide, starting at R1C4 and clk1 will reside only in that area. Any logic that is clocked by clk1 will be placed within that region. If it cannot be placed in that region a DRC error will occur in place and route. The diagram of this region looks like this:

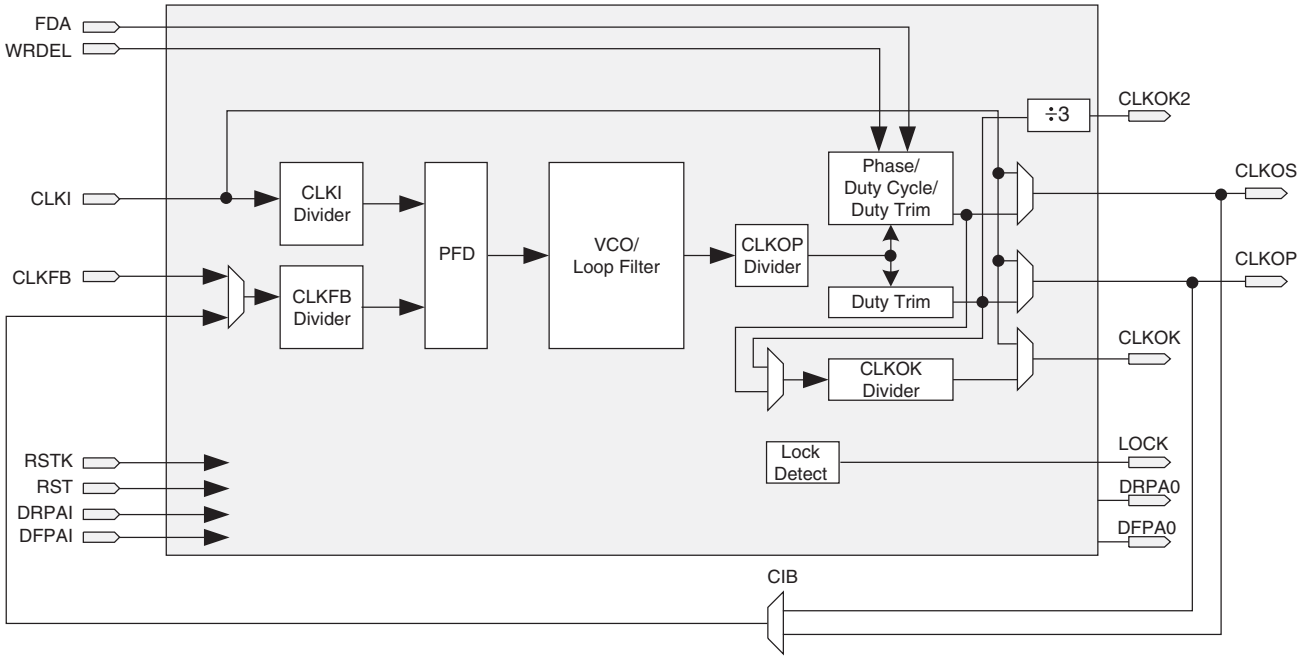
R1C1	R1C2	R1C3	R1C4	R1C5	R1C6
R2C1	R2C2	R2C3	R2C4	R2C5	R2C6
R3C1	R3C2	R3C3	R3C4	R3C5	R3C6
R4C1	R4C2	R4C3	R4C4	R4C5	R4C6
R5C1	R5C2	R5C3	R5C4	R5C5	R5C6
R6C1	R6C2	R6C3	R6C4	R6C5	R6C6

sysCLOCK™ PLL

The LatticeECP3 PLL provides features such as clock injection delay removal, frequency synthesis, phase/duty cycle adjustment, and dynamic delay adjustment. Figure 10-7 shows the block diagram of the LatticeECP3 PLL.

Generally, the best way to add a PLL to a design is by using IPexpress™; the user simply provides information to the tool via a GUI, and the tool performs all calculations and design rule checks. It then generates a package that can be added to your design in the HDL language of your choice.

Figure 10-7. LatticeECP3 PLL Block Diagram



Functional Description

PLL Divider and Delay Blocks

Input Clock (CLKI) Divider

The CLKI divider is used to control the input clock frequency into the PLL block.This is also referred to as the M-Divider. The divider setting directly corresponds to the divisor of the output clock. The input and output of the input divider must be within the input and output frequency ranges specified in the LatticeECP3 Family Data Sheet. This is checked by IPexpress.

Feedback Loop (CLKFB) Divider

The CLKFB divider is used to divide the feedback signal. This is also referred to as the N-Divider. Effectively, this multiplies the output clock, because the divided feedback must speed up to match the input frequency into the PLL block. The PLL block increases the output frequency until the divided feedback frequency equals the input frequency. The input and output of the feedback divider must be within the input and output frequency ranges specified in the LatticeECP3 Family Data Sheet. This is checked by IPexpress.

Output Clock (CLKOP) Divider

The CLKOP divider serves the dual purposes of squaring the duty cycle of the VCO output and scaling up the VCO frequency into the 500MHz to 1000MHz range to minimize jitter. This is also referred to as the V-Divider.

CLKOK Divider

The CLKOK divider acts as a source for the global clock nets. This is also referred to as the K-Divider. It divides the CLKOP or CLKOS signal (user selectable) of the PLL by the value of the divider to produce lower frequency clock.

CLKOK2 Divider

The CLKOK2 divider always works off the CLKOP signal and has a fixed value of 3. The CLKOK2 signal can be used for generating 140 MHz from 420 MHz to support SPI4.2 or for other uses. The first rising edge of CLKOK2 is aligned to the first falling edge of CLKOP (after reset) and the falling edge of CLKOK2 is aligned to the third rising edge of CLKOP. This will show up as a skew between CLKOP and CLKOK equal to one-half of the CLKOP period. CLKOK2 is available to be routed as a primary clock.

Phase Adjustment and Duty Cycle Select (Static Mode)
Users can program CLKOS with Phase and Duty Cycle options. Phase adjustment can be done in 22.5° steps. The duty cycle resolution is 1/16th of a period except 1/16th, 15/16th and 16/16th duty cycle options, which are not supported to avoid minimum pulse violation.

Dynamic Phase Adjustment (DPHASE) and Dynamic Duty Cycle (DDUTY) Select
The Phase Adjustment and Duty Cycle Select can be controlled in dynamic mode. When this mode is selected, both the Phase Adjustment and Duty Cycle Select must be in dynamic mode. If only one of the features is to be used in dynamic mode, users can manually set the other control inputs to the fixed logic levels of their choice.

Duty Trim Adjustment
With the LatticeECP3 device family, the duty cycle can be fine-tuned with the Duty Trim Adjustment.

Fine Delay Adjust
This optional feature is controlled by the input port, WRDEL. See information on the WRDEL input in the next section of this document.

PLL Inputs and Outputs

CLKI Input

The CLKI signal is the reference clock for the PLL. It must conform to the specifications in the LatticeECP3 Family Data Sheet in order for the PLL to operate correctly. The CLKI can be sourced from a dedicated dual-purpose pin or from routing. Please note it is not recommended to use a DCS output as a clock source to this input as a loss of PLL lock can occur.

RST Input

The PLL reset occurs under two conditions. First, at power-up, the internal power-up reset signal from the configuration block resets the PLL. Second, the user-controlled PLL reset signal RST, provided as part of the PLL module, can be driven by an internally generated reset function or an external pin. This RST signal resets all internal PLL counters, flip-flops (including the M, N, V, K and CLKOK2 Dividers) and the charge pump. When RST goes inactive, the PLL will start the lock-in process, and will take t_{LOCK} time to complete the PLL lock. Figure 10-8 shows the timing diagram of the RST input. Figure 10-9 shows the timing relationship between the RST input and the CLKI divider output. RST is active high. The RST signal is optional; if unused, tie the input LOW. RST asserts asynchronously and deasserts synchronously.

Figure 10-8. RST Input Timing Diagram

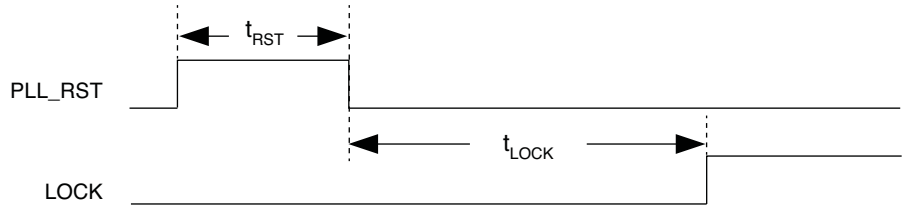
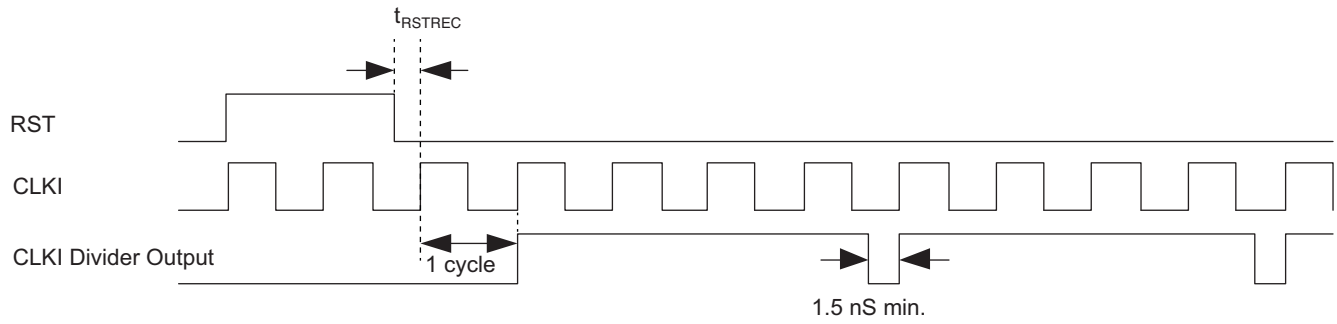


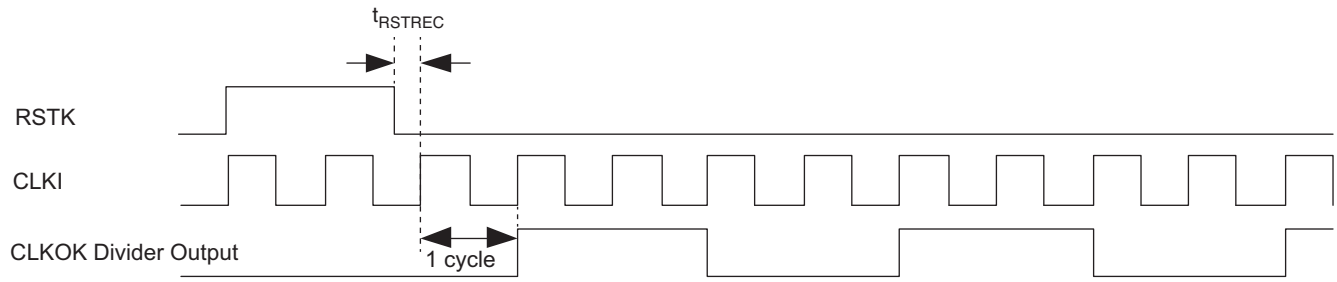
Figure 10-9. RST Input and CLKI Divider Output Timing Diagram (Example: CLKI_DIV = 4)



RSTK Input

RSTK is the reset input for the K-Divider (refer to Figure 10-10). This K-Divider reset is used to synchronize the K-Divider output clock to the input clock. LatticeECP3 has an optional gearbox in the I/O cell for both outputs and inputs. The K-Divider reset is useful for the gearbox implementation. RSTK is active high.

Figure 10-10. RSTK Input and CLKOK Divider Output Timing Diagram (Example: CLKOK_DIV = 4)



CLKFB Input

The feedback signal to the PLL, which is fed through the feedback divider, can be derived from the Primary Clock net (CLKOP), a preferred pin, directly from the CLKOP divider or from general routing. External feedback allows the designer to compensate for board-level clock alignment. Please note it is not recommended to use a DCS output as a clock source to this input as a loss of PLL lock can occur.

CLKOP Output

The sysCLOCK PLL main clock output, CLKOP, is a signal available for selection as a primary clock.

CLKOS Output with Phase and Duty Cycle Select

The sysCLOCK PLL auxiliary clock output, CLKOS, is a signal available for selection as a primary clock. The CLKOS is used when phase shift and/or duty cycle adjustment is desired. The programmable phase shift allows for different phase in increments of 22.5°. The duty select feature provides duty select in 1/16th of the clock period. This feature is also supported in Dynamic Control Mode.

CLKOK Output with Lower Frequency

The CLKOK is used when a lower frequency is desired. It is a signal available for selection as a primary clock.

CLKOK2 Output

The CLKOK2 divider always works off the CLKOP signal and has a fixed value of 3. The CLKOK2 signal can be used for generating 140 MHz from 420 MHz to support SPI4.2 or for other uses. The first rising edge of CLKOK2 is aligned to the first falling edge of CLKOP (after reset) and the falling edge of CLKOK2 is aligned to the third rising edge of CLKOP. This will show up as a skew between CLKOP and CLKOK equal to one-half of the CLKOP period. CLKOK2 is available to be routed as a primary clock.

LOCK Output

The LOCK output provides information about the status of the PLL. After the device is powered up and the input clock is valid, the PLL will achieve lock within the specified lock time. Once lock is achieved, the PLL lock signal will be asserted. If, during operation, the input clock or feedback signals to the PLL become invalid, the PLL will lose lock. However, when the input clock completely stops, the LOCK output will remain in its last state, since it is internally registered by this clock. It is recommended to assert PLL RST to re-synchronize the PLL to the reference clock. The LOCK signal is available to the FPGA routing to implement generation of RST. ModelSim® simulation models take two to four reference clock cycles from RST release to LOCK high.

Dynamic Phase and Dynamic Duty Cycle Adjustment

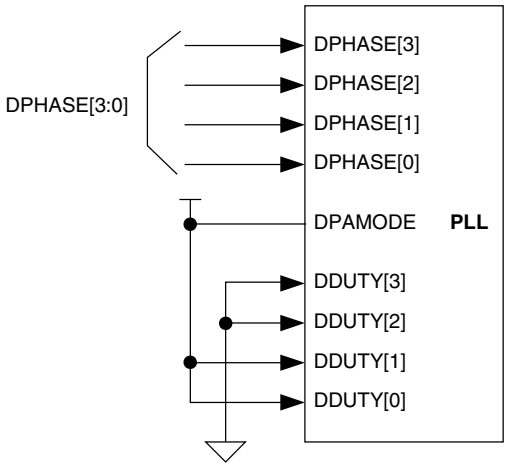
The DPHASE[3:0] port is used with the Dynamic Phase Adjustment feature to allow the user to connect a control signal to the PLL. The DDUTY[3:0] port is used with the Dynamic Duty Adjustment feature to allow the user to connect a control signal to the PLL. The DPHASE and DDUTY ports are listed in Table 10-3.

The Dynamic Phase and Dynamic Duty Cycle Adjustment features will be discussed in more detail in later sections of this document.

Table 10-3. Dynamic Phase and Duty Cycle Adjust Ports

Port Name	I/O	Description
DPHASE[3:0]	I	Dynamic Phase Adjust inputs
DDUTY[3:0]	I	Dynamic Duty Cycle Adjust inputs

Figure 10-11. Example of Dynamic Phase Adjustment with a Fixed Duty Cycle of 3/16th of a Period



Dynamic Phase Adjustment/Duty Cycle Select

Phase Adjustment settings are described in Table 10-4.

Table 10-4. Phase Adjustment Settings

DPHASE[3:0]	Phase (°)
0000	0
0001	22.5
0010	45
0011	67.5
0100	90
0101	112.5
0110	135
0111	157.5
1000	180
1001	202.5
1010	225
1011	247.5
1100	270
1101	292.5
1110	315
1111	337.5

Fine Delay Ports

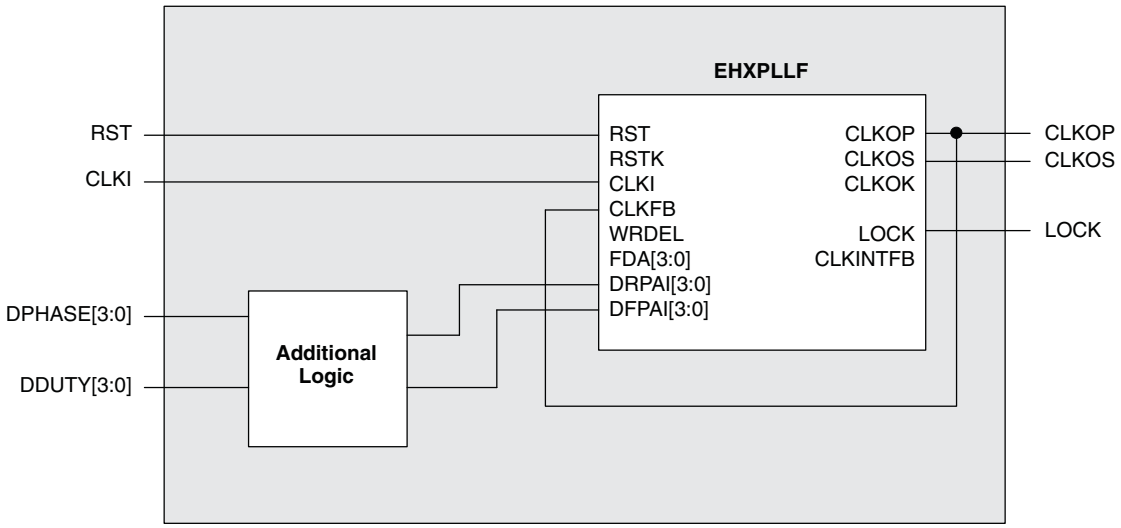
When selecting “Dynamic Mode" and enabling CLKOS, the FINEDELB0-3 ports appear. These ports allow the user to lag the CLKOS output clock with respect to the feedback clock in increments of t_{PA} * FINEDELB(0:3). The t_{PA} values are located in the LatticeECP3 Family Data Sheet.

The FINEDELA port is enabled with the FINEDELA checkbox on the GUI. This port allows you to push CLKOS ahead of the feedback clock by ~70 ps. This signal is an active high pulse.

LatticeECP3 PLL Modules

When the user creates a PLL module using IPexpress, the module will consist of a wrapper around the PLL library element and any additional logic required for the module. Figure 10-12 is a diagram of a typical PLL module. The module port names can be different than the library element is some cases. The user will see the module port names in the IPexpress window and also in the source code file for the generated module. These are the ports that will be connected in the user's design. IPexpress also creates an instantiation template file that shows the user how to instantiate the PLL module in their design. The user can import the *.LPC file (in ispLEVER™, or *.IPX in Diamond) into their project or the generated source code file.

Figure 10-12. LatticeECP3 Typical PLL Module Generated by IPexpress

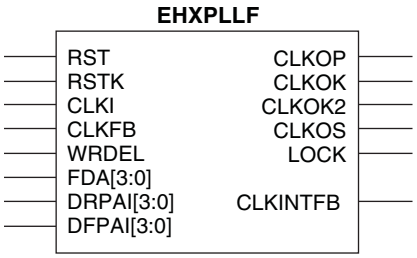


The PLL module shown in Figure 10-12 represents an example where the user has chosen to use the CLKOP and CLKOS ports, with a PLL reset signal, PLL lock signal, and dynamic phase and dynamic duty cycle. It also uses CLKOP feedback so the software will connect the CLKOP signal to the CLKFB port and use the primary clock tree to route this signal. The user would connect their signals to the CLKI, RST, DPHASE[3:0], DDUTY[3:0], CLKOP, CLKOS, and LOCK signals.

LatticeECP3 PLL Library Definition

One PLL library element is used for LatticeECP3 PLL implementation. Figure 10-13 shows the LatticeECP3 PLL library symbols.

Figure 10-13. LatticeECP3 PLL Library Symbol



EPLLD Design Migration from LatticeECP2 to LatticeECP3

The EPLLD generated for LatticeECP2 can be used with minor changes. If the configuration does not include Dynamic Phase and Duty Options, the migration is fully supported. If Dynamic Phase and Duty Options are included, the user must tie the DPAMODE port to ground.

Dynamic Phase/Duty Mode

This mode sets both Dynamic Phase Adjustment and Dynamic Duty Select at the same time. There are two modes, “Dynamic Phase and Dynamic Duty” and “Dynamic Phase and 50% Duty”.

- **Dynamic Phase and 50% Duty**
  This mode allows users to set up Dynamic Phase inputs only. The 50% Duty Cycle is handled internally by the ispLEVER software. The DDUTY[3:0] ports are user-transparent in this mode.
- **Dynamic Phase and Dynamic Duty**

This mode allows designers to use both DDPHASE[3:0] and DDUTY[3:0] ports to input dynamic values.

To use Dynamic Phase Adjustment with a fixed duty cycle other than a 50%, simply set the DDUTY[3:0] inputs to the desired duty cycle value. Figure 10-11 illustrates an example circuit.

Example: Assume a design uses dynamic phase adjustment and a fixed duty cycle select and the desired duty cycle in 3/16th of a period. The setup should be as shown in Figure 10-11.

Duty Cycle Select settings are described in Table 10-5.

Table 10-5. Duty Cycle Select Settings

DDUTY[3:0]	Duty Cycle (1/16th of a Period)	Comment
0000	0	Not Supported
0001	1	Not Supported
0010	2	
0011	3	
0100	4	
0101	5	
0110	6	
0111	7	
1000	8	
1001	9	
1010	10	
1011	11	
1100	12	
1101	13	
1110	14	
1111	15	Not Supported

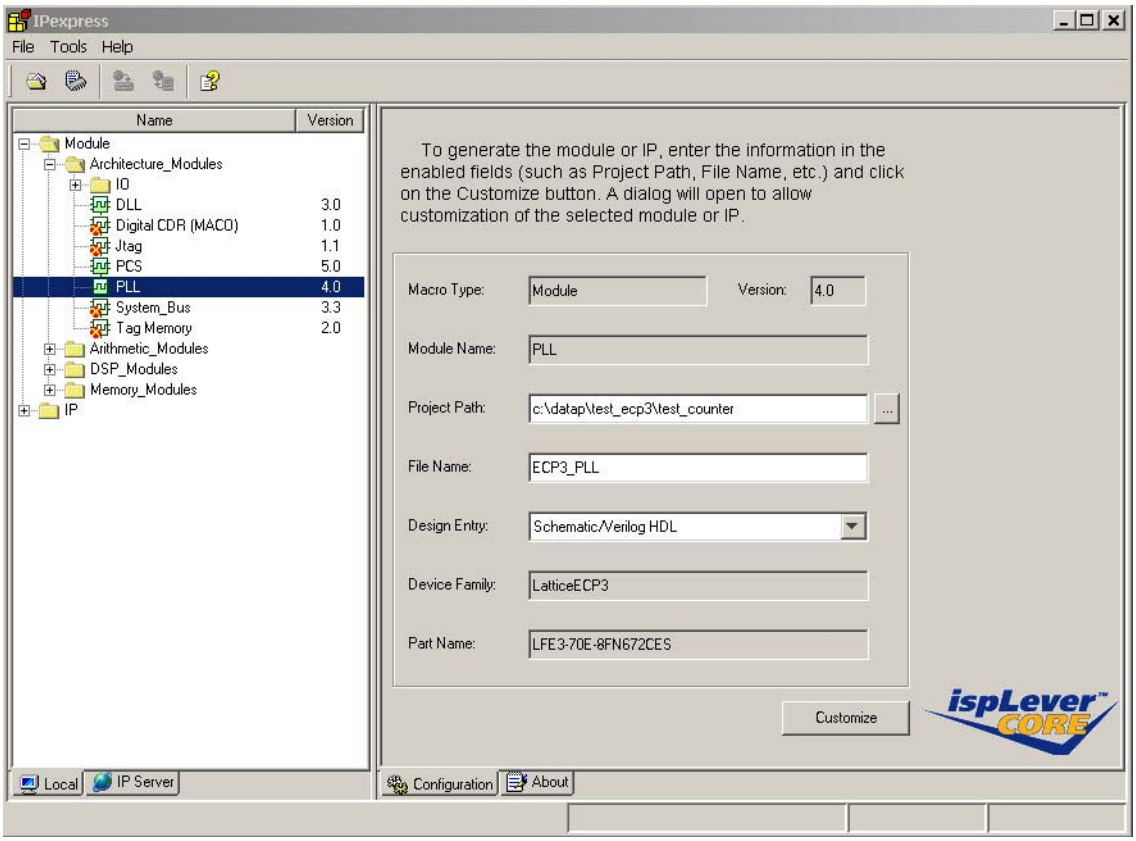
Note: PHASE/DUTY_CTNL is selected in the GUI ‘PLL Phase & Duty Options’ box and if it is set to ‘Dynamic Mode’, then both DPHASE[3:0] and DDUTY[3:0] inputs must be provided. If one of these inputs is a fixed value, the inputs must be tied to the desired fixed logic levels.

PLL Usage in IPexpress

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

Figure 10-14 shows the main window when PLL is selected. The only entry required in this window is the module name. Other entries are set to the project settings. Users may change these entries, if desired. After entering the module name of choice, clicking on **Customize** will open the Configuration Tab window as shown in Figure 10-15.

Figure 10-14. ispLEVER IPexpress Main Window (see Appendix C Figure 10-39 for Diamond Equivalent)



Configuration Tab

The Configuration Tab lists all user accessible attributes with default values set. Upon completion, clicking **Generate** will generate source and constraint files. Users may choose to use the *.LPC file (for ispLEVER, or *.IPX file for Diamond) to load parameters.

Configuration Modes

There are two modes that can be used to configure the PLL in the Configuration Tab, Frequency Mode and Divider Mode.

Frequency Mode: In this mode, the user enters input and output clock frequencies and the software calculates the divider settings. If the output frequency entered is not achievable the nearest frequency will be displayed in the 'Actual' text box. After input and output frequencies are entered, clicking the **Calculate** button will display the divider values.

Divider Mode: In this mode, the user sets the divider settings with input frequency. Users must choose the CLKOP Divider value to maximize the f_{VCO} and achieve optimum PLL performance. After input frequency and divider settings are set, clicking the **Calculate** button will display the frequencies. Figure 10-15 shows the Configuration Tab.

Figure 10-15. ispLEVER LatticeECP3 PLL Configuration Tab (see Appendix C Figure 10-40 for Diamond Equivalent)

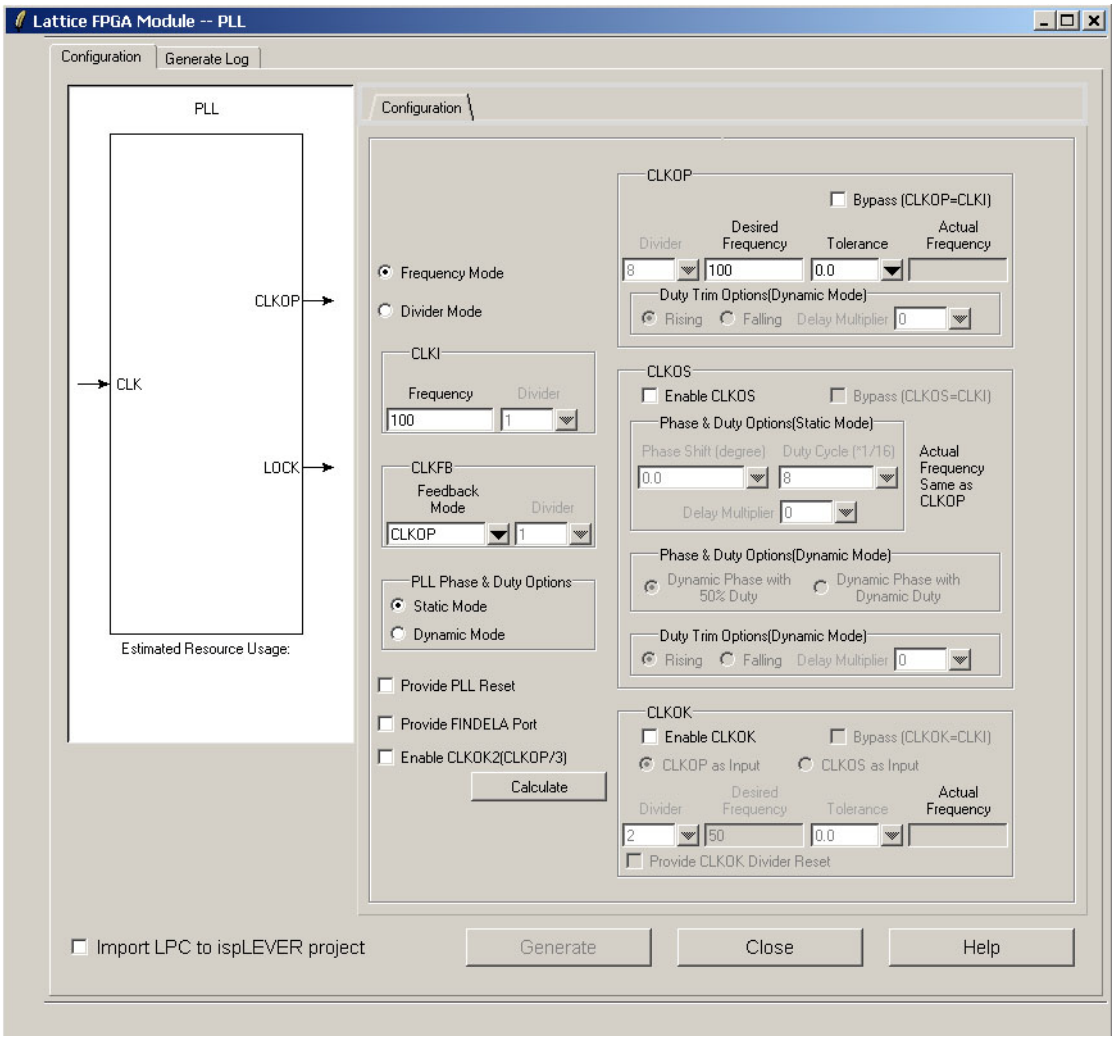


Table 10-6 describes the user parameters in the IPexpress GUI and their usage.

Table 10-6. User Parameters in the IPexpress GUI

User Parameters		Description	Range
Frequency Mode		User desired CLKI and CLKOP frequency	ON/OFF
Divider Mode		User desired CLKI frequency and dividers settings	ON/OFF
CLKI	Frequency	Input Clock frequency	2 MHz to 500 MHz
	Divider	Input Clock Divider Setting (Divider Mode)	1 to 64
CLKFB	Feedback Mode	Feedback Mode	Internal, CLKOP, CLKOS, User Clock
	Divider	Feedback Clock Divider Setting (Divider Mode)	1 to 64
PLL Phase & Duty Options	Static Mode	CLKOS Phase/Duty in Static Mode	ON/OFF
	Dynamic Mode	CLKOS Dynamic Mode Phase/Duty Setting	ON/OFF

Table 10-6. User Parameters in the IPexpress GUI (Continued)

User Parameters		Description	Range
CLKOP	Bypass	Bypass PLL: CLKOP = CLKI	ON/OFF
	Desired Frequency	User enters desired CLKOP frequency	4 MHz to 500 MHz
	Divider	CLKOP Divider Setting (Divider Mode)	2, 4, 8, 16, 32, 48, 64, 80, 96, 112, 128
	Tolerance	CLKOP tolerance users can tolerate	0.0, 0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0
	Actual Frequency	Actual frequency achievable. Read only	—
	Rising	Rising Edge Trim	ON/OFF
	Falling	Falling Edge Trim	ON/OFF
	Delay Multiplier	Number of delay steps	0 to 7
CLKOS	Enable	Enable CLKOS output clock	ON/OFF
	Bypass	Bypass PLL: CLKOS = CLKI	ON/OFF
	Phase Shift	CLKOS Static Phase Shift	0°, 22.5°, 45°..337.5°
	Duty Cycle	CLKOS Static Duty Cycle	2 to 14
	Phase and Duty Options	Dynamic Phase with 50% Duty	ON/OFF
		Dynamic Phase with Dynamic Duty	ON/OFF
	Rising	Rising Edge Trim	ON/OFF
	Falling	Falling Edge Trim	ON/OFF
	Delay Multiplier	Number of Delay steps	0 to 3
CLKOK	Enable	Enable CLKOS output clock	ON/OFF
	Bypass	Bypass PLL: CLKOK = CLKI	ON/OFF
	Desired Frequency	User enters desired CLKOK frequency	0.03125 to 250 MHz
	CLKOK input	Select input source for CLKOK	CLKOP/CLKOS
	Frequency	User enters desired CLKOK frequency	31.25 kHz to 250 MHz
	Divider	CLKOK Divider Setting	2, 4, 6, ... 126, 128
	Tolerance	CLKOK tolerance users can tolerate	0.0, 0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0
	Actual Frequency	Actual frequency achievable. Read only	—
CLKOK2	Enable	Enable CLKOK2 output clock	ON/OFF
Provide PLL Reset		Provide PLL Reset Port (RESET)	ON/OFF
Provide CLKOK Divide Reset		Provide CLKOK Reset Port (RSTK)	ON/OFF
Provide FINDELA Port		Provide CLKOS Fine Delay Port (WRDEL)	ON/OFF
Import LPC to ispLEVER Project		Import .lpc file to ispLEVER project	ON/OFF
Import IPX into Diamond Project		Import .lpc file to Diamond project	ON/OFF

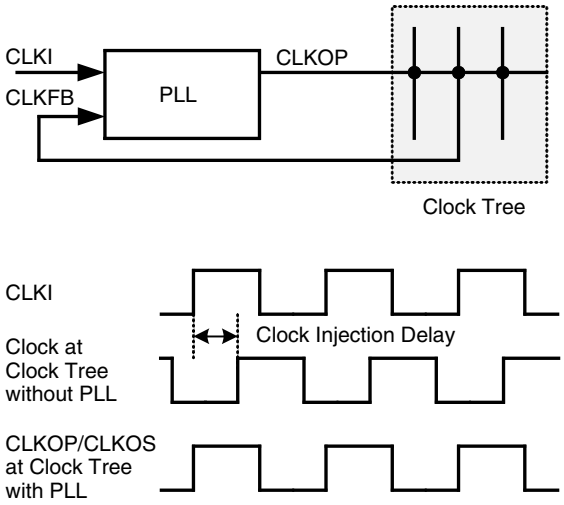
PLL Modes of Operation

PLLs have many uses within a logic design. The two most popular are Clock Injection Removal and Clock Phase Adjustment. These two modes of operation are described below.

PLL Clock Injection Removal

In this mode the PLL is 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 phase detector of the PLL aligns the CLKI with CLKFB. If the CLKFB signal comes from the clock tree (CLKOP), then the PLL delay and the clock tree delay is removed. Figure 10-16 Illustrates an example block diagram and waveform.

Figure 10-16. Clock Injection Delay Removal Application

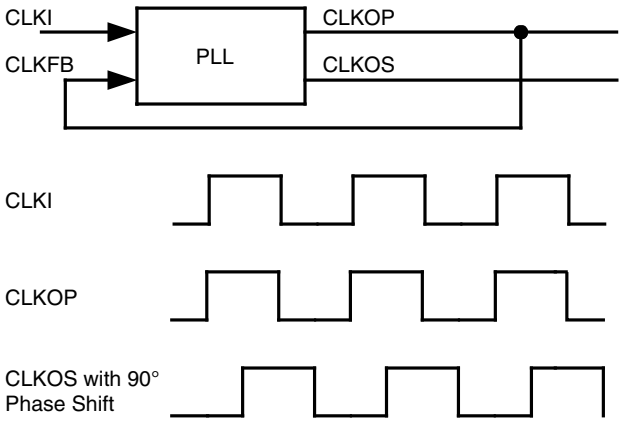


PLL Clock Phase Adjustment

Refer to Figure 10-17. In this mode the PLL is used to create fixed phase relationships in 22.5° increments. Creating fixed phase relationships is useful for forward clock interfaces where a specific relationship between clock and data is required.

The fixed phase relationship can be used between CLKI and CLKOS or between CLKOP and CLKOS.

Figure 10-17. CLKOS Phase Adjustment from CLKOP



IPexpress Output

There are two IPexpress outputs that are important for use in the design. The first is the <module_name>.[v|vhd] file. This is the user-named module that was generated by the tool to be used in both synthesis and simulation flows. The second is a template file, <module_name>_tmpl.[v|vhd]. This file contains a sample instantiation of the module. This file is 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, IPexpress sets attributes in the HDL module that are specific to the data rate selected. Although these attributes can be easily changed, they should only be modified by regenerating the package in IPexpress so that the performance of the PLL 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.

Notes on PLL Usage

The GPLL CLKOP should be used as the feedback source to optimize PLL performance.

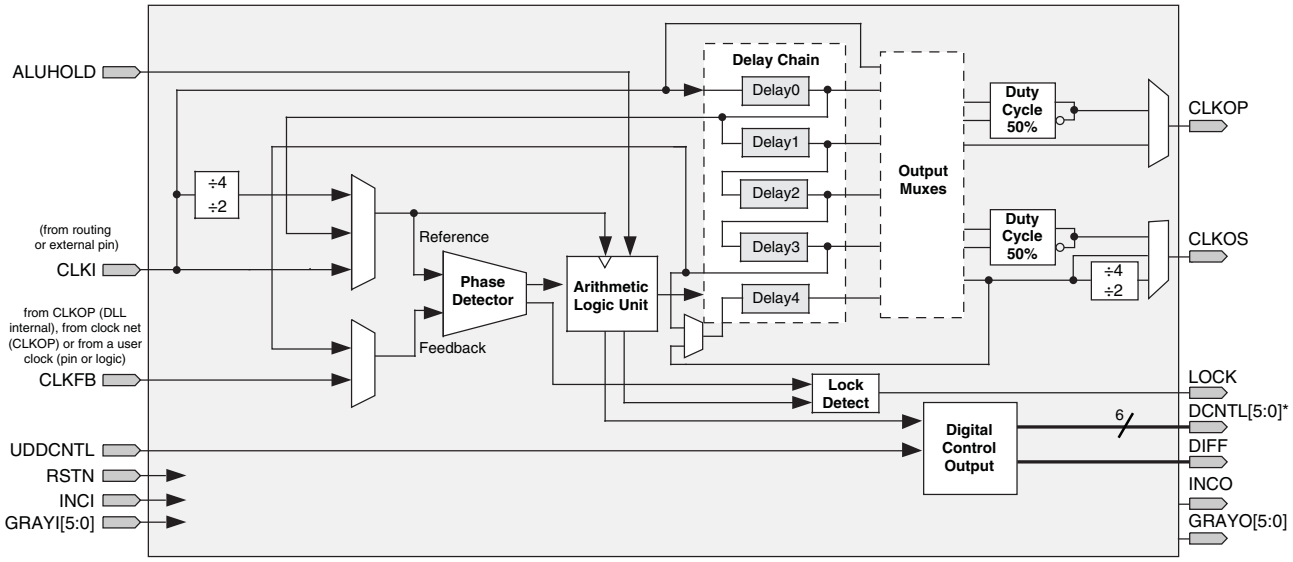
Most designers use PLLs for clock tree injection removal mode and the CLKOP should be assigned to a primary clock. This is done automatically by the software unless otherwise specified by the user.

CLKOP can route only to CLK0 to CLK5, while CLKOS/CLKOK can route to all primary clocks (CLK0 TO CLK7).

sysCLOCK DLL

The LatticeECP3 DLL provides features such as clock injection delay removal, time reference delay (90º 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[5:0]). The delay control vector is connected to a Slave Delay Line (DLLDEL) element. Figure 10-18 provides a block diagram of the LatticeECP3 DLL.

Figure 10-18. LatticeECP3 DLL Block Diagram



* This signal is not user accessible. This can only be used to feed the slave delay line.

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

DLL Overview

The LatticeECP3 DLL is created and configured by IPexpress. The following is a list of port names and descriptions for the DLL. There are two library elements used to implement the DLL: CIDDLLB (Clock Injection Delay), and TRDDLLB (Time Reference Delay). IPexpress will wrap one of these library elements to create a customized DLL module based on user selections.

DLL Inputs and Outputs

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 provides the lowest latency and best case performance.

CLKFB Input
The CLKFB input is available only if the user chooses 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. In Clock Injection Delay Removal mode, the DLL will align the input clock phase with the feedback clock phase by delaying the input clock.

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

DCNTL[5:0] Output
This output of the DLL is used to delay a signal by a specific amount. The DCNTL[5:0] vector can only connect to a Slave Delay Line element.

DIFF Output
Active high difference indicator. Active when DCNTL output is different than the internal setting and an update is needed.

UDDCNTL Input
This input is used to enable or disable updating of the DCNTL[5:0]. 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[5:0] outputs are continuously updated.

ALUHOLD Input
This active high input stops the DLL from adding and subtracting delays to the CLKI signal. The DCNTL[5:0], 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. If the CLKI input stops the LOCK output will remain asserted. Since the clock is stopped, there is no clock to deassert 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. The LOCK output transitions are glitch-free.

RSTN
Active low reset input to reset the DLL. The DLL can optionally be reset by the GSRN as well. It is recommended that if the DLL requires a reset, the reset should not be 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. RSTN asserts asynchronously and deasserts synchronously.

GRAYO Output
Gray-coded digital control bus to other DLLs. This bus, together with the GRAYI bus and INCO and INCI signals, enables DLLs to be safely cascaded. The buses are Gray-coded to prevent glitches in the transfer of the control signals. The INCO / INCI signal accompanies the GRAYO / GRAYI bus, and indicates when an incremental adjustment is being passed.

GRAYI Input
Gray-coded digital control bus from another DLL in time reference mode. See description of the GRAYO output bus, above.

INCO Output
Active high incremental indicator to other DLLs. See description of the GRAYO output bus, above.

INCI Input
Active high incremental indicator from another DLL. See description of the GRAYO output bus, above.

DLL Attributes

The LatticeECP3 DLL utilizes several attributes that allow the configuration of the DLL through source constraints, IPexpress and preference files. The following section details these attributes and their usage.

DLL Lock on Divide by 2 or Divide by 4 CLKOS Output
The LatticeECP3 DLL allows 'divide by 2' or 'divide by 4' CLKOS outputs. Two optional 'divide by 2' and 'divide by 4' blocks are placed at the CLKI input as well as the CLKOS and this enables the use of divided CLKOS in the DLL feedback path. This allows the DLL to perform clock injection removal on a 'divide by 2' or 'divide by 4' clock, which is useful for DDRX2 and DDRX4 modes of I/O buffer operation.

When this optional clock divider is used only in the CLKOS output path, it allows the DLL to output two time-aligned clocks at different frequencies. When the divider is set to divide by 2 or divide by 4, a 'dummy' delay is inserted in the CLKOP output path to match the clock to Q delay of the CLKOS divider.

DLL Lock Time Control
The DLL will lock when the CLKI and CLKFB phases are aligned. In a simulation environment, the lock time is fixed to 100µs (default). This value can be changed through an HDL parameter or preference (for the back annotation simulation). The DLL contains a parameter named LOCK_DELAY which accepts an integer value for the total time in µs 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),
```

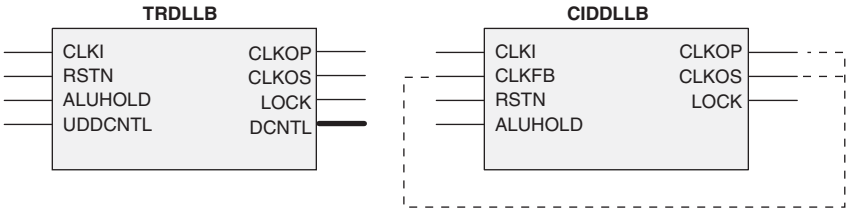
VHDL:

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 "EHXPLLF" LOCK_DELAY=200;
```

DLL Library Symbols

Figure 10-19. DLL Library Symbols



DLL Library Definitions

The Lattice library contains library elements to allow designers to utilize the DLL. These library elements use the DLL attributes defined in the "DLL Attributes" section.

The two modes of operation are presented as library elements as listed below.

Table 10-7. DLL Library Elements

Library Element Name	Mode of Operation	Description
TRDLLB	Time Reference Delay DLL	This mode generates four phases of the clock, 0°, 90°, 180°, 270°, along with the control setting used to generate these phases.
CIDDLLB	Clock Injection Delay DLL (Four Delay Cell Mode)	This mode removes the clock tree delay, aligning the external feedback clock to the reference clock. It has a single output coming from the fourth delay block.

DLL Library Element I/Os

Table 10-8. DLL Library Element I/O Descriptions

Signal	I/O	Description
CLKI	I	Clock input pin from dedicated clock input pin, other I/O or logic block.
CLKFB	I	Clock feedback input pin from dedicated feedback input pin, internal feedback, other I/O or logic block. This signal is not user selectable.
RSTN	I	Active low synchronous reset. From dedicated pin or internal node.
ALUHOLD	I	“1” freezes the ALU. For TRDLLA and CIDDLLA.
UDDCNTL	I	Active high synchronous enable signal from CIB for updating digital control to PIC delay. It must be driven high at least two clock cycles.
DCNTL[5:0]	O	Digital delay control code signals.
CLKOP	O	The primary clock output for all possible modes.
CLKOS	O	The secondary clock output with finer phase shift and/or division by 2 or by 4.
LOCK	O	Active high phase lock indicator. Lock means the reference and feedback clocks are in phase.

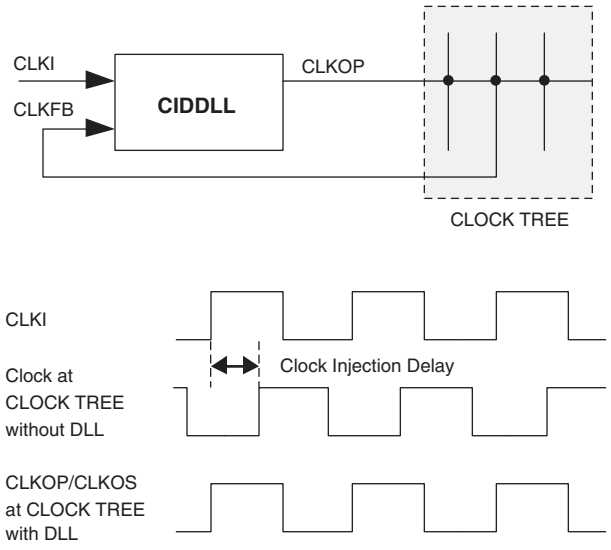
Note: Refer to the LatticeECP3 Family Data Sheet for frequency specifications.

DLL Modes of Operation

Clock Injection Removal Mode (CIDDLLB)

The DLL can be used to reduce clock injection delay (CIDDLLB). 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 10-20 shows a circuit example and waveform.

Figure 10-20. Clock Injection Delay Removal via DLL

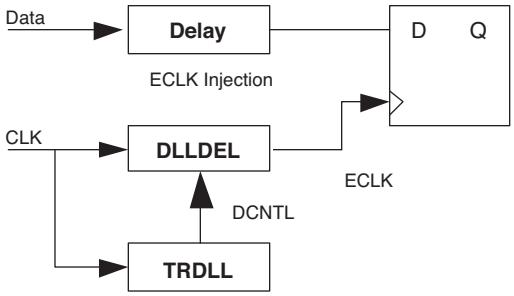


Clock injection removal mode can also provide a DCNTL port. When using the DCNTL, the DLL delay will be limited to the range of the DCNTL vector. Therefore, IPexpress will restrict the CLKI rate from 300MHz to 500MHz.

Time Reference Delay Mode (TRDLLB: 90-Degree Phase Delay)
The Time Reference Delay (TRDDLLB) mode of the DLL is used to calculate 90 degrees 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 10-21 provides a circuit example of this mode.

Figure 10-21. 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 a Slave Delay Line (DLLDELB) to delay the signal by 90 degrees of the full period of CLKI.

DLL Usage in IPexpress

IPexpress is used to create and configure a DLL. The IPexpress graphical user interface, shown in Figure 10-13, allows users to select parameters for the DLL. Parameters are described in Table 10-9. The result is an HDL model to be used in the simulation and synthesis flow.

Figure 10-22. ispLEVER LatticeECP3 IPexpress DLL Configuration Tab (see Figure 10-41 for Diamond Equivalent)

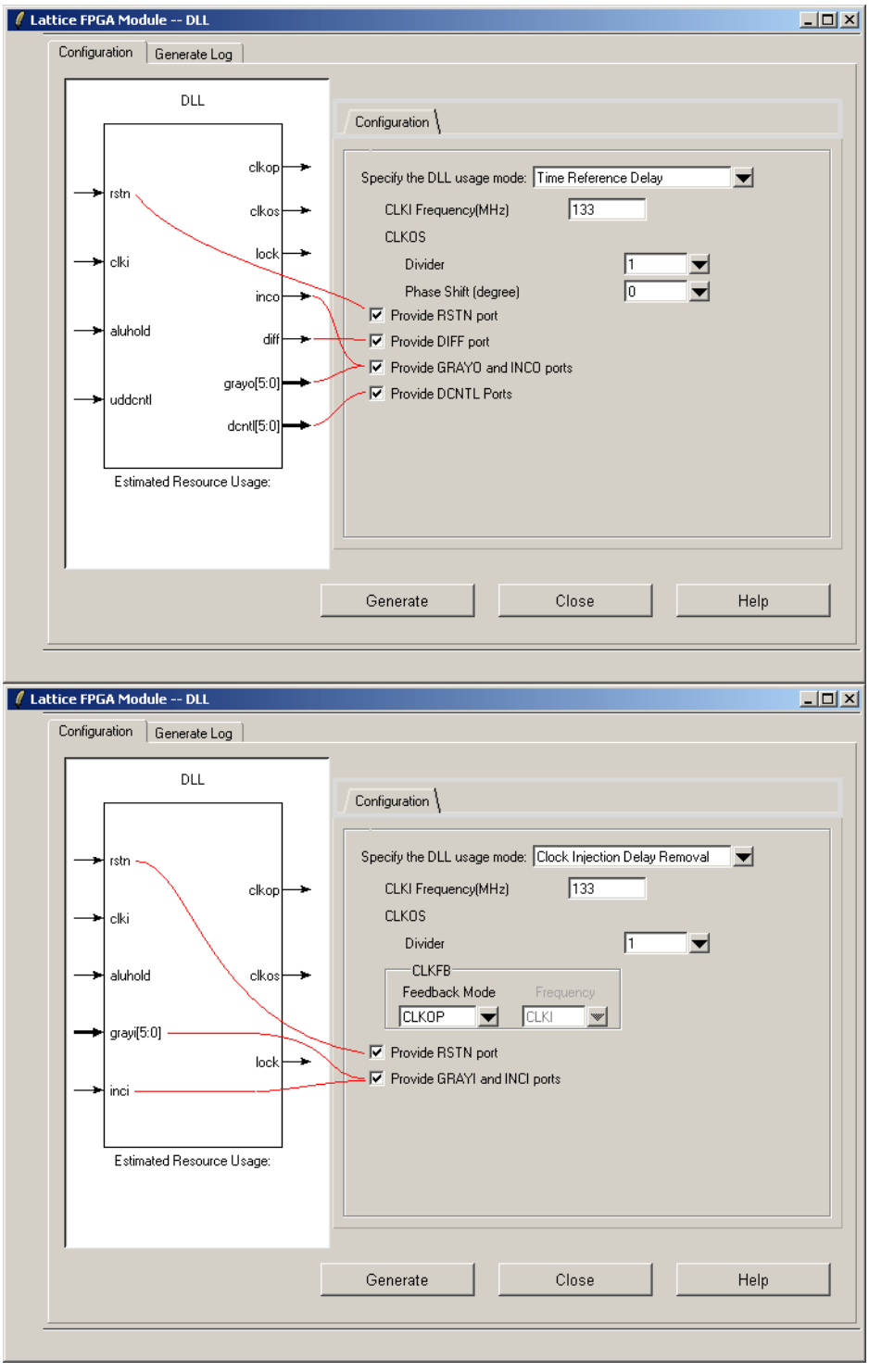


Table 10-9. User Parameters in the IPexpress DLL GUI

User Parameter	Description	Range	Default
DLL Usage Mode	User desired operation mode	Time Reference Delay, Clock Injection Delay Removal	Clock Injection Delay Removal
CLKI Frequency (MHz)	Input CLKI frequency	133-500 MHz	133 MHz
CLKOS Divider	Output CLKOS Divider Setting	1, 2, 4	1
CLKOS Phase Shift (degrees)	Output CLKOS Phase Shift Setting	0° to 360° in 11° steps	0°
CLKFB Feedback Mode	Feedback Clock mode (source of feedback)	CLKOP, CLKOS, User Clock	CLKOP
CLKFB Frequency	Feedback Clock source frequency (CLKI divided by 1, 2 or 4)	CLKI, CLKI/2, CLKI/4	CLKI
Provide RSTN Port	Provide reset port (active-LO)	ON/OFF	ON
Provide DIFF Port	Provide DIFF port	ON/OFF	OFF
Provide GRAYO Port (Time Ref Delay mode only)	These inputs/outputs are used to safely cascade PLLs/DLLs. Refer to text for details.	ON/OFF	OFF
Provide GRAYI Port (Clk Inj Dly Rmvl mode only)		ON/OFF	OFF
Provide INCO Port (Time Ref Delay mode only)	These inputs/outputs are used to safely cascade PLLs/DLLs. Refer to text for details.	ON/OFF	OFF
Provide INCI Port (Clk Inj Dly Rmvl mode only)		ON/OFF	OFF
Provide DCNTL Port	Provide Delay Control Vector output port	ON/OFF	OFF
Import PIX to Diamond Project (Diamond only)	Import .IPX file to project	YES/NO	NO

PLL/DLL Cascading

It is possible to connect several arrangements of PLLs and DLLs. There are three possible cascading schemes:

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

It is not possible to connect the DLL to a PLL. The DLL produces abrupt changes on its output clocks when changing delay settings. The PLL sees this as radical phase changes that prevent the PLL from locking correctly. A DLL in Static Delay Mode can, however, be used to set fine phase delays, and it is generally best to do this with the DLL placed in front of the PLL.

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. 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 easily be 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.

DLLDEL (Slave Delay Line)

The Slave Delay line is designed to generate the desired delay in DDR/SPI4 applications. The delay control inputs (DCNTL[5:0]) are fed from the general purpose DLL outputs. The library element definitions are described in Figure 10-23 and Table 10-10.

Figure 10-23. DLLDELB Library Symbol

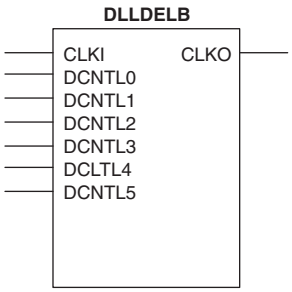


Table 10-10. DLLDELB I/O

Name	I/O	Description
CLKI	I	Clock Input
DCNTL[5:0]	I	Delay Control Bits
CLKO	O	Clock Output

DLLDELB Declaration in VHDL Source Code

```
COMPONENT DLLDELB
      PORT     (
               CLKI :IN std_logic;
               DCNTL0 :IN std_logic;
               DCNTL1 :IN std_logic;
               DCNTL2 :IN std_logic;
               DCNTL3 :IN std_logic;
               DCNTL4 :IN std_logic;
               DCNTL5 :IN std_logic;
               CLKO :OUT std_logic
               );
END COMPONENT;

begin
DLLDELBinst0: DLLDELB1
      PORT MAP (
               CLKI => clkisig,
               DCNTL0 => dcntl0sig,
               DCNTL1 => dcntl1sig,
               DCNTL2 => dcntl2sig,
               DCNTL3 => dcntl3sig,
               DCNTL4 => dcntl4sig,
               DCNTL5 => dcntl5sig,
               CLKO => clkosig
               );

end
```

DLLDELB Usage with TRDLLB - Verilog - Example

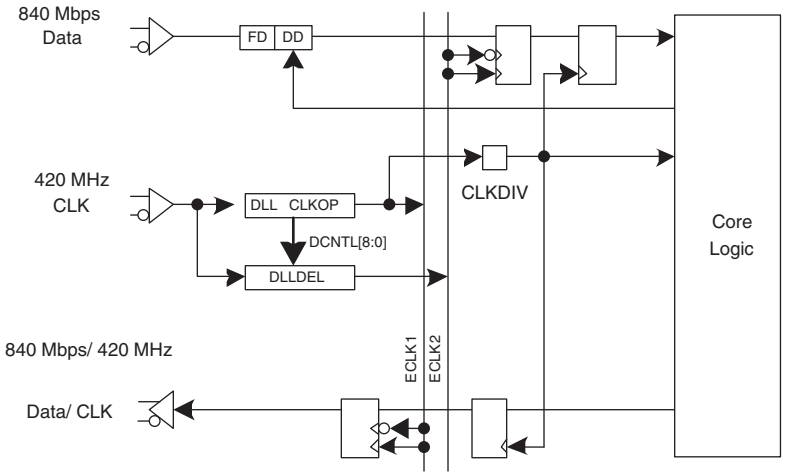
Note: DLL0(TRDLLB) must be generated by IPexpress as a sub-module

```
module ddldel_top (rst,d,clkin,clkout,aluhold,uddcntl,q);

input rst,d,clkin,aluhold,uddcntl;
output clkout,q;

wire [5:0]DCntl_int;
reg qint;

DLL0 dllinst0 (.clk(clkin), .aluhold(aluhold), .uddcntl(uddcntl), .clkop(), .clkos(),
           .dcntl(DCntl_int),.lock());
DLLDELB delinst0 (.CLKI(clkin),.DCNTL0(DCntl_int[0]),.DCNTL1(DCntl_int[1]),
            .DCNTL2(DCntl_int[2]), .DCNTL3(DCntl_int[3]), .DCNTL4(DCntl_int[4]),
            .DCNTL5(DCntl_int[5]), .CLKO(clk90)); //synthesis syn_black_box

assign clkout = clk90;
assign q = qint;

always@(posedge clk90 or negedge rst)
      if (~rst)
         qint =1'b0;
      else
         qint = d;

endmodule
```

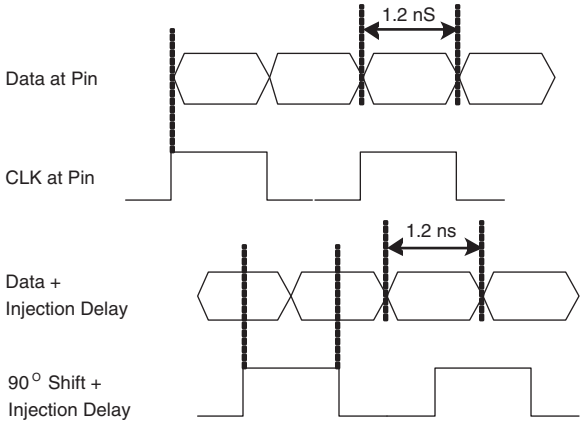
DLLDELB Application Example

Figure 10-24 shows an example DLLDEL application. As shown in the timing diagram, DLLDEL shifts the clock by 90 degrees to center both edges in the middle of data window.

Figure 10-24. SPI4.2 and DDR Registers Interface Application



FD: Fixed Delay
DD: Dynamic Delay
Users can select the delay setting in IPexpress.



DQSDLL and DQSDEL

There is another combination of DLL and Slave Delay Line, DQSDLL and DQSDEL, in the LatticeECP3 device family. This pair is similar in design and function to DLL and DLLDEL, but usage is limited to DDR implementation. For additional information, see TN1180, LatticeECP3 High-Speed I/O Interface.

Clock Dividers (CLKDIV)

The clock divider divides the high-speed clock by 1, 2, 4 or 8. All the outputs have matched input to output delay. CLKDIV can take as its input the edge clocks and the CLKOP of the PLL or DLL. The divided outputs drive the primary clock and are also available for general routing or secondary clocks. The clock dividers are used for providing the low speed FPGA clocks for shift registers (x2, x4, x8) and DDR/SPI4 I/O logic interfaces.

CLKDIV Library Definition

Users can instantiate CLKDIV in the source code as defined in this section. Figure 10-25 and Tables 10-11 and 10-12 describe the CLKDIVB definitions.

Figure 10-25. CLKDIV Library Symbol

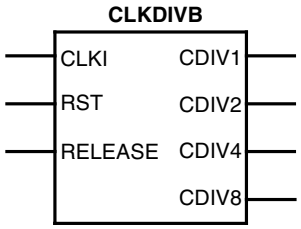


Table 10-11. CLKDIVB Port Definition

Name	Description
CLKI	Clock Input
RST	Reset Input, asynchronously forces all outputs low.
RELEASE	Releases outputs synchronously to input clock.
CDIV1	Divided BY 1 Output
CDIV2	Divided BY 2 Output
CDIV4	Divided BY 4 Output
CDIV8	Divided BY 8 Output

Table 10-12. CLKDIVB Attribute Definition

Name	Description	Value	Default
GSR	GSR Enable	ENABLED/DISABLED	DISABLED

CLKDIV Declaration in VHDL Source Code

```
COMPONENT CLKDIVB
-- synthesis translate_off
      GENERIC (
         GSR : in String);
-- synthesis translate_on
      PORT (
         CLKI,RST, RELEASE:IN    std_logic;
         CDIV1, CDIV2, CDIV4, CDIV8:OUT   std_logic);
END COMPONENT;

      attribute GSR : string;
      attribute GSR of CLKDIVinst0 : label is “DISABLED”;

begin

CLKDIVinst0:         CLKDIVB
-- synthesis translate_off
     GENERIC MAP(
         GSR         => “disabled”
         );
-- synthesis translate_on
      PORT MAP(
         CLKI        => CLKIsig,
         RST         => RSTsig,
         RELEASE     => RELEASEsig,
         CDIV1       => CDIV1sig,
```

```
        CDIV2       => CDIV2sig,
        CDIV4       => CDIV4sig,
        CDIV8       => CDIV8sig
        );
```

CLKDIV Usage with Verilog - Example

```
module clkdiv_top(RST,CLKI,RELEASE,CDIV1,CDIV2,CDIV4,CDIV8);

input CLKI,RST,RELEASE;
output CDIV1,CDIV2,CDIV4,CDIV8;

CLKDIVB CLKDIBinst0 (.RST(RST),.CLKI(CLKI),.RELEASE(RELEASE),
     .CDIV1(CDIV1),.CDIV2(CDIV2),.CDIV4(CDIV4),.CDIV8(CDIV8));

defparam CLKDIBint0.GSR = "DISABLED"

endmodule
```

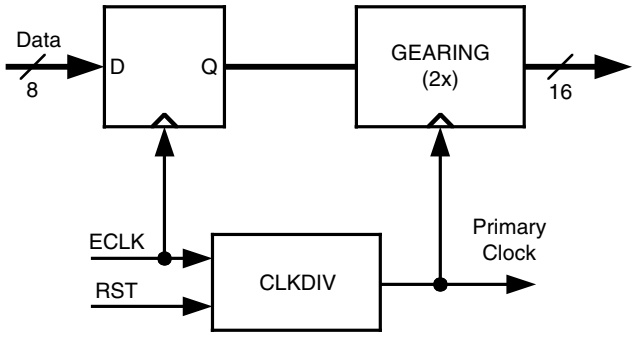
CLKDIV Example Circuits

The clock divider (CLKDIV) can divide a clock by 2, 4 or 8 and drives a primary clock network. 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. Divide by 8 is provided for slow speed/low power operation.

To guarantee a synchronous transfer in the I/O logic, the CLKDIV input clock must be driven by an edge clock and the output must drive a primary clock. In this case, they are phase matched.

It is especially useful to synchronously reset the I/O logic when Mux/DeMux gearing is used in order to synchronize the entire data bus as shown in Figure 10-26. 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.

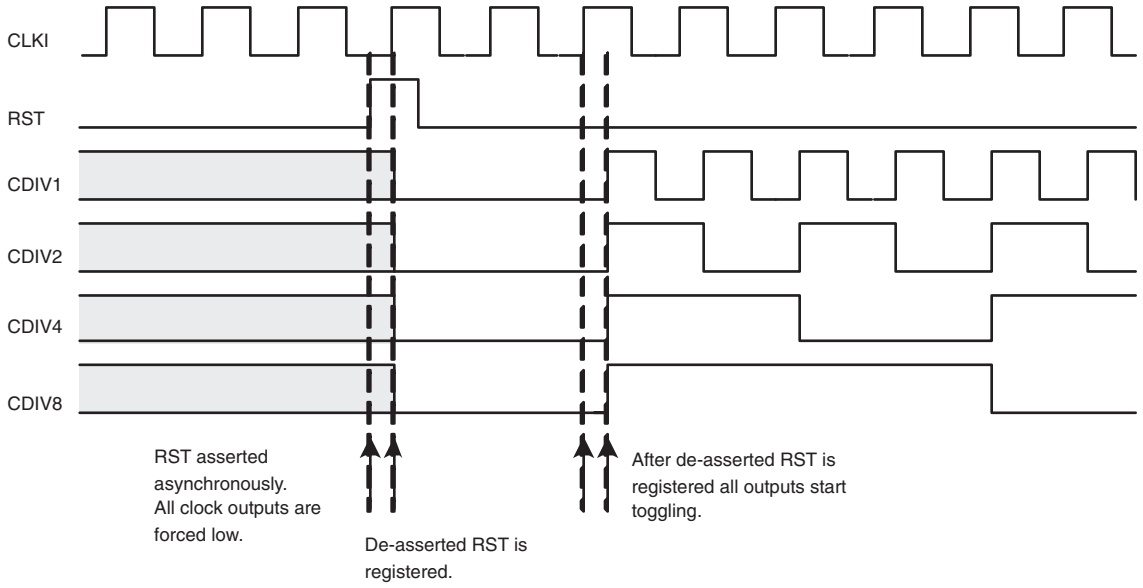
Figure 10-26. CLKDIV Application Example



Reset Behavior

Figure 10-27 illustrates the asynchronous RST behavior.

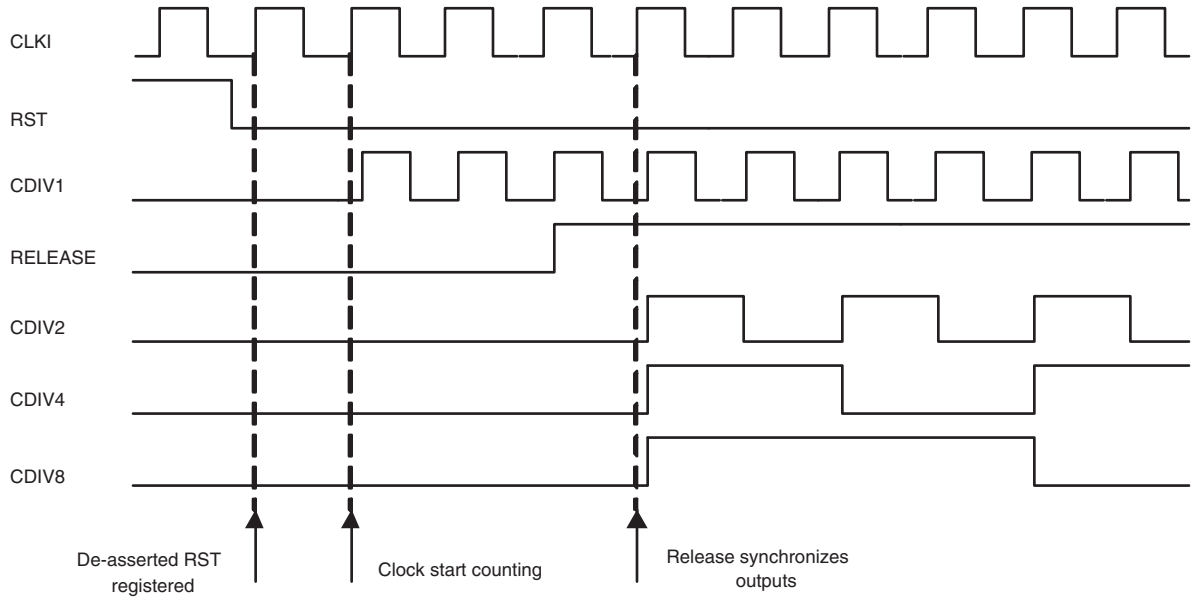
Figure 10-27. CLKDIV Reset Behavior



Release Behavior

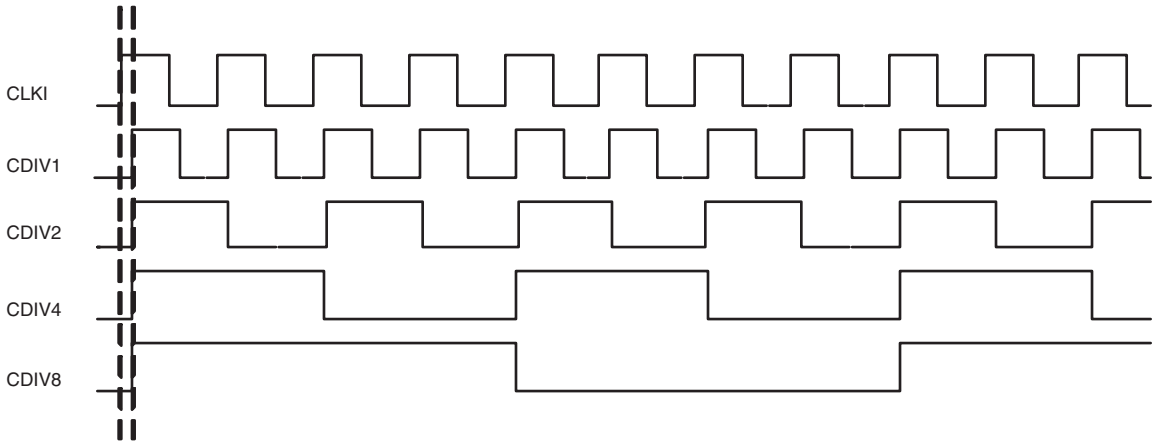
The port, “Release” is used to synchronize the all outputs after RST is de-asserted. Figure 10-28 illustrates the release behavior.

Figure 10-28. CLKDIV Release Behavior



CLKDIV Inputs-to-Outputs Delay Matching

Figure 10-29. CLKDIV Inputs-to-Outputs Delay Matching



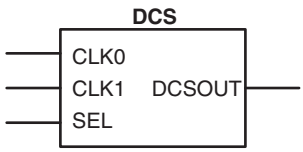
DCS (Dynamic Clock Select)

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

As can be seen in Figure 10-30, the DCS inputs are driven by all the same signals that drive non-DCS clocks, except that the PLL and DLL inputs are somewhat restricted. The select input is driven by a signal from the general routing fabric. The outputs of the DCS then reach primary clock distribution via the feedlines. Figure 10-30 shows the block diagram of the DCS.

When CLK6 or CLK7 is used as a primary clock and there is only one clock input to the DCS, the DCS is assigned as a buffer mode by the software, but will still inject some delay into the net. In order to be glitchless, the DCS must have both clock inputs switching. If one of the inputs is not switching, the DCS will not be able to switch.

Figure 10-30. DCS Library Symbol



DCS Library Definition

Table 10-13 defines the I/O ports of the DCS block. There are eight modes to select from. Table 10-14 describes how each mode is configured.

Table 10-13. DCS I/O Definition

I/O	Name	Description
Input	SEL	Input Clock Select
	CLK0	Clock input 0
	CLK1	Clock Input 1
Output	DCSOUT	Clock Output

Table 10-14. DCS Modes of Operation

Attribute Name	Description	Output		Value
		SEL=0	SEL=1	
DCS MODE	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 a unique operation. The clock output timing is determined by input clocks and the edge of the SEL signal. Figure 10-31 describes the timing of each mode.

Figure 10-31. Timing Diagrams by DCS MODE

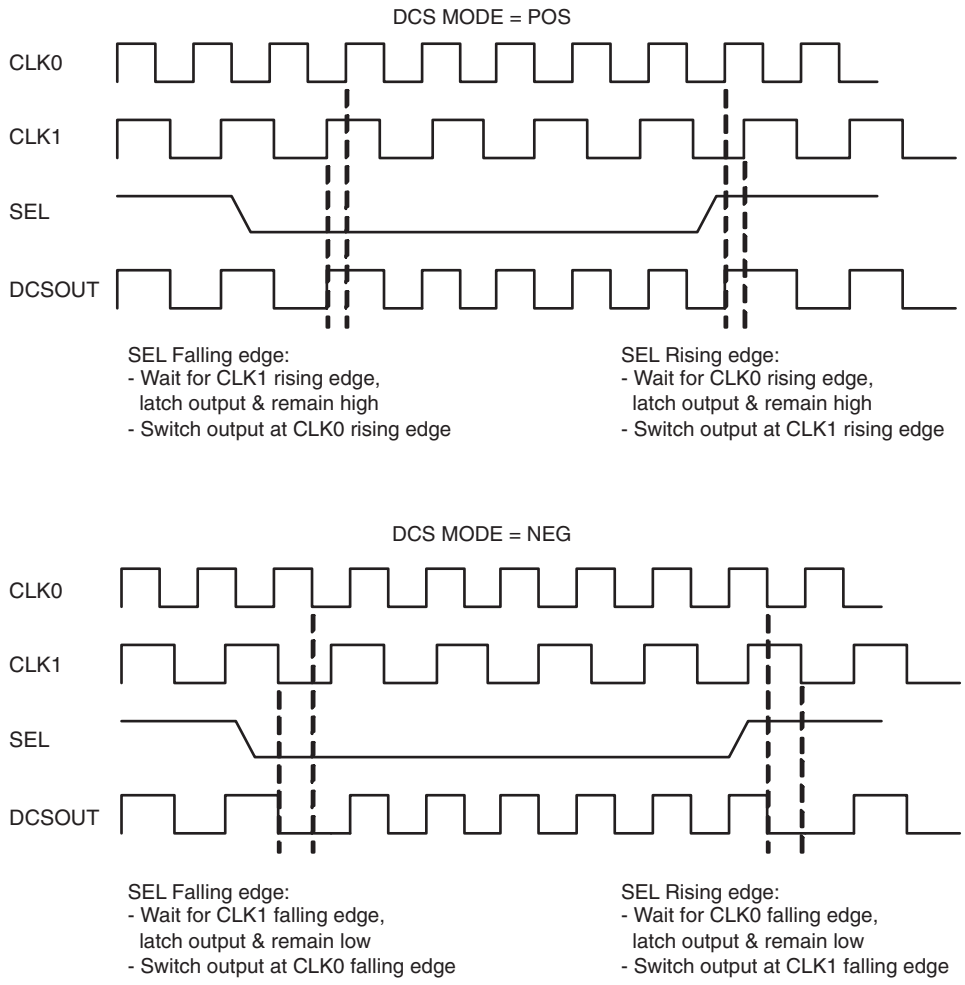
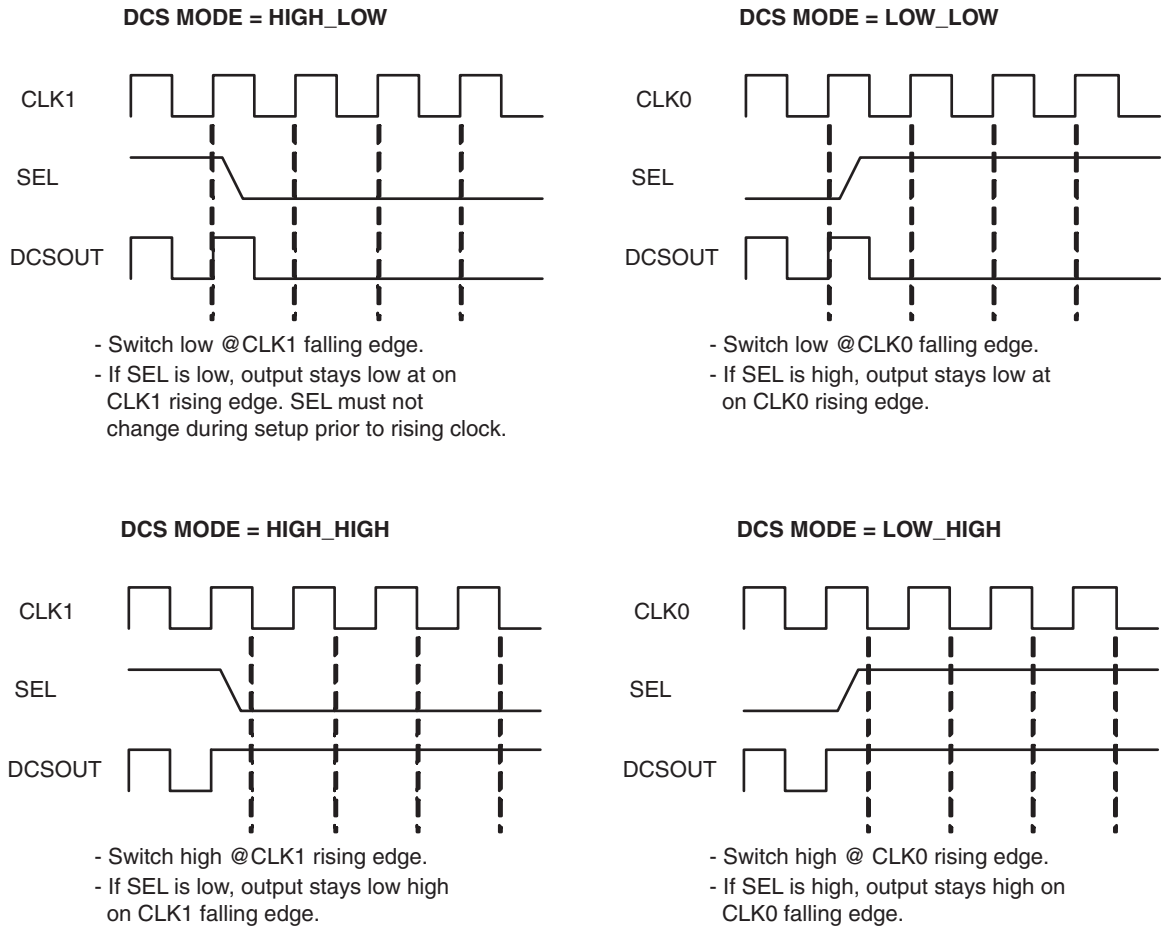


Figure 10-32. Timing Diagrams by DCS MODE (Cont.)



DCS Usage with 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 DCSinst0 : label is “POS”;

begin
```

```
DCSInst0: DCS
-- synthesis translate_off

        GENERIC MAP (
        DCSMODE  => “POS”
);
-- synthesis translate_on

        PORT MAP (
                SEL         =>  clksel,
                CLK0        =>  dcsclk0,
                CLK1        =>  sysclk1,
                DCSOUT      =>  dcsclk
                );
```

DCS Usage with Verilog - Example

```
module dcs(clk0,clk1,sel,dcsout);

input clk0, clk1, sel;
output dcsout;

DCS DCSInst0 (.SEL(sel),.CLK0(clk0),.CLK1(clk1),.DCSOUT(dcsout));
defparam DCSInst0.DCSMODE = "POS";

endmodule
```

Oscillator (OSCF)

There is a dedicated oscillator in the LatticeECP3 device whose output is made available for users.

The oscillator frequency output is routed through a divider which is used as an input clock to the clock tree. The available outputs of the divider are shown in Table 10-15. The oscillator frequency output can be further divided by internal logic (user logic) for lower frequencies, if desired. The oscillator is powered down when not in use.

The output of this oscillator is not a precision clock. It is intended as an extra clock that does not require accurate clocking.

Library Element: OSCF

Table 10-15. OSCE Port Definition

I/O	Name	Description
Output	OSC	Oscillator Clock Output

Table 10-16. OSCE Attribute Definition

User Attribute	Attribute Name	Value (MHz)	Default Value
Nominal Frequency	NOM_FREQ	2.5, 4.3, 5.4, 6.9, 8.1, 9.2, 10, 13, 15, 20, 26, 30, 34, 41, 45, 51, 55, 60, 130	2.5

OSC Library Symbol (OSCF)

Figure 10-33. OSC Symbol

OSC Usage with VHDL - Example

```
COMPONENT OSCF

      PORT (OSC:OUT   std_logic);

END COMPONENT;
begin
OSCInst0: OSCF
      PORT MAP ( OSC=>  osc_int);
```

OSC Usage with Verilog - Example

```
module OSC_TOP(OSC_CLK);

output OSC_CLK;

OSCF OSCinst0 (.OSC(OSC_CLK));

defparam OSCinst0.NOM_FREQ = "5.4" ;

endmodule
```

Setting Clock Preferences

Designers can use clock preferences to implement clocks to the desired performance. Preferences can be set in the ispLEVER Design Planner Spreadsheet View (or Diamond Spreadsheet View) or in preference files. Frequently used preferences are described in Appendix C.

Power Supplies

Each PLL has its own power supply pin, VCCPLL.

PLL/DLL Names and Preferred Pads

Table 10-17 lists the names and preferred pads for all PLLs and DLLs for each device size in the LatticeECP3 family. To obtain the pin/ball associated with each pad, refer to the pinout table for the target device in the LatticeECP3 Family Data Sheet. Here is an example of a PLL locate preference:

```
LOCATE COMP “PLL_0/PLLInst_0” SITE “PLL_R35C70”;
```

Table 10-17. PLL/DLL Names and Preferred Pads

		Pad IN_A	Pad IN_B	Pad FB_A	Pad FB_B
LatticeECP3-17					
LUM0_GDLLT	DLL_R26C15	PL20A	PL20B	PL21A	PL21B
RUM0_GDLLT	DLL_R26C42	PR20A	PR20B	PR21A	PR21B
LUM0_GPLLT	PLL_R26C5	PL26E_C	PL26E_D	PL26E_A	PL26E_B
RUM0_GPLLT	PLL_R26C52	PR26E_C	PR26E_D	—	—

Table 10-17. PLL/DLL Names and Preferred Pads (Continued)

		Pad IN_A	Pad IN_B	Pad FB_A	Pad FB_B
LatticeECP3-35					
LUM0_GDLLT	DLL_R35C15	PL29A	PL29B	PL30A	PL30B
RUM0_GDLLT	DLL_R35C60	PR29A	PR29B	PR30A	PR30B
LUM0_GPLLT	PLL_R53C5	PL35E_C	PL35E_D	PL35E_A	PL35E_B
LLM1_GPLLT	PLL_R35C5	PL53E_C	PL53E_D	PL53E_A	PL53E_B
RLM1_GPLLT	PLL_R53C70	PR53E_C	PR53E_D	PR53E_A	PR53E_B
RUM0_GPLLT	PLL_R35C70	PR35E_C	PR35E_D	PR35E_A	PR35E_B
LatticeECP3-70/95					
LUM0_GDLLT	DLL_R43C15	PL37A	PL37B	PL38A	PL38B
RUM0_GDLLT	DLL_R43C132	PR37A	PR37B	PR38A	PR38B
LUM2_GPLLT	PLL_R25C5	PL25E_C	PL25E_D	PL25E_A	PL25E_B
LUM0_GPLLT	PLL_R43C5	PL43E_C	PL43E_D	PL43E_A	PL43E_B
LLM1_GPLLT	PLL_R61C5	PL61E_C	PL61E_D	PL61E_A	PL61E_B
LLM2_GPLLT	PLL_R70C5	PL70E_C	PL70E_D	PL70E_A	PL70E_B
LLM3_GPLLT	PLL_R79C5	PL79E_C	PL79E_D	PL79E_A	PL79E_B
RLM3_GPLLT	PLL_R79C142	PR79E_C	PR79E_D	PR79E_A	PR79E_B
RLM2_GPLLT	PLL_R70C142	PR70E_C	PR70E_D	PR70E_A	PR70E_B
RLM1_GPLLT	PLL_R61C142	PR61E_C	PR61E_D	PR61E_A	PR61E_B
RUM0_GPLLT	PLL_R43C142	PR43E_C	PR43E_D	PR43E_A	PR43E_B
RUM2_GPLLT	PLL_R25C142	PR25E_C	PR25E_D	PR25E_A	PR25E_B
LatticeECP3-150					
LUM0_GDLLT	DLL_R61C15	PL55A	PL55B	PL56A	PL56B
RUM0_GDLLT	DLL_R61C168	PR55A	PR55B	PR56A	PR56B
LUM2_GPLLT	PLL_R43C5	PL43E_C	PL43E_D	PL43E_A	PL43E_B
LUM0_GPLLT	PLL_R61C5	PL61E_C	PL61E_D	PL61E_A	PL61E_B
LLM1_GPLLT	PLL_R79C5	PL79E_C	PL79E_D	PL79E_A	PL79E_B
LLM3_GPLLT	PLL_R97C5	PL97E_C	PL97E_D	PL97E_A	PL97E_B
LLM4_GPLLT	PLL_R106C5	PL106E_C	PL106E_D	PL106E_A	PL106E_B
RLM4_GPLLT	PLL_R106C178	PR106E_C	PR106E_D	PR106E_A	PR106E_B
RLM3_GPLLT	PLL_R97C178	PR97E_C	PR97E_D	PR97E_A	PR97E_B
RLM1_GPLLT	PLL_R79C178	PR79E_C	PR79E_D	PR79E_A	PR79E_B
RUM0_GPLLT	PLL_R61C178	PR61E_C	PR61E_D	PR61E_A	PR61E_B
RUM2_GPLLT	PLL_R43C178	PR43E_C	PR43E_D	PR43E_A	PR43E_B

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 2009	01.0	Initial release.
July 2009	01.1	Updated Secondary Clocks text section.
September 2009	01.2	Updated LatticeECP3 Primary Clock Muxes figure.
November 2009	01.3	Updated Edge Clocks text section.
February 2010	01.4	Reconciled LOCK description among MachXO, LatticeXP2, LatticeECP2/M and LatticeECP3.
April 2010	01.5	Updated the Figure for Time Reference Delay Circuitry Example
		Updated example for DLLDELB Usage with TRDLLB - Verilog
		Added new section - PLL/DLL Names and Preferred Pads
June 2010	01.6	Updated for Lattice Diamond design software support.
January 2011	01.7	Added General Routing for Clocks text section.
February 2011	01.8	Added Fine Delay Ports text section.
June 2011	01.9	Output Clock (CLKOP) Divider text section – VCO frequency changed from 435MHz - 870 MHZ to 500MHz - 1000MHz.
January 2012	02.0	Updated CLKI Input text section.
		Updated CLKFB Input text section.
		Updated DCS (Dynamic Clock Select) text section.
February 2012	02.1	Updated document with new corporate logo.
April 2012	02.2	Updated CLKDIV Usage with Verilog - Example.
		Updated CLKDIV Example Circuits text section.
		Updated DCS Usage with Verilog - Example.
May 2012	02.3	Updated CLKOK2 signal to clarify timing relationship to CLKOP.
September 2012	02.4	Update to fix Table 10-2, add primary clock preference examples, and clarify secondary clock regions and how to create preferences for them.

Appendix A. Primary Clock Sources and Distribution

Figure 10-34. LatticeECP3 Primary Clock Sources and Distribution

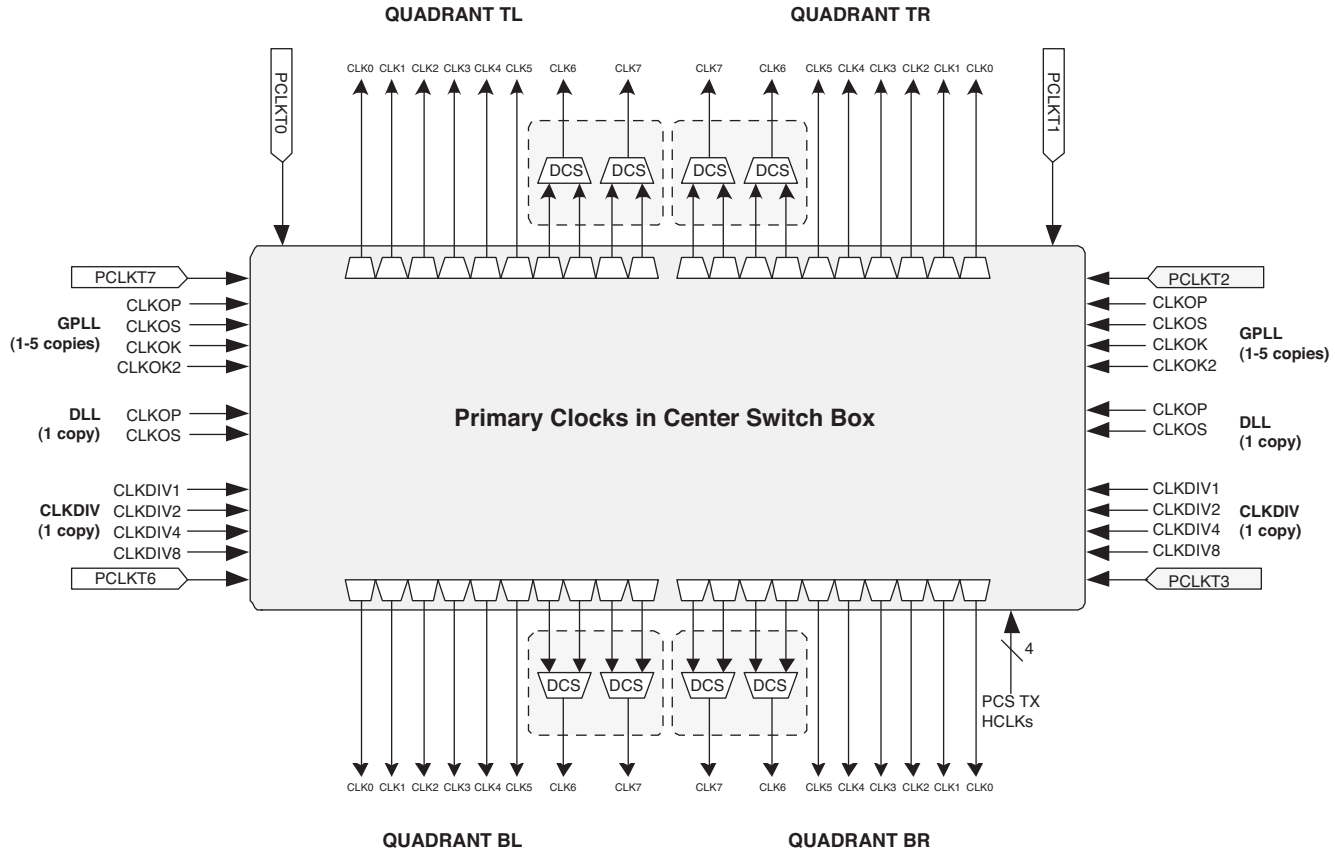
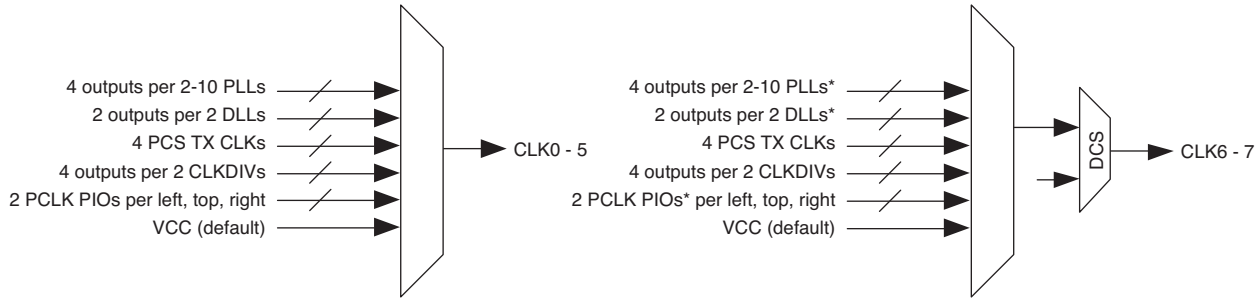


Figure 10-35. LatticeECP3 Primary Clock Muxes (Simplified)

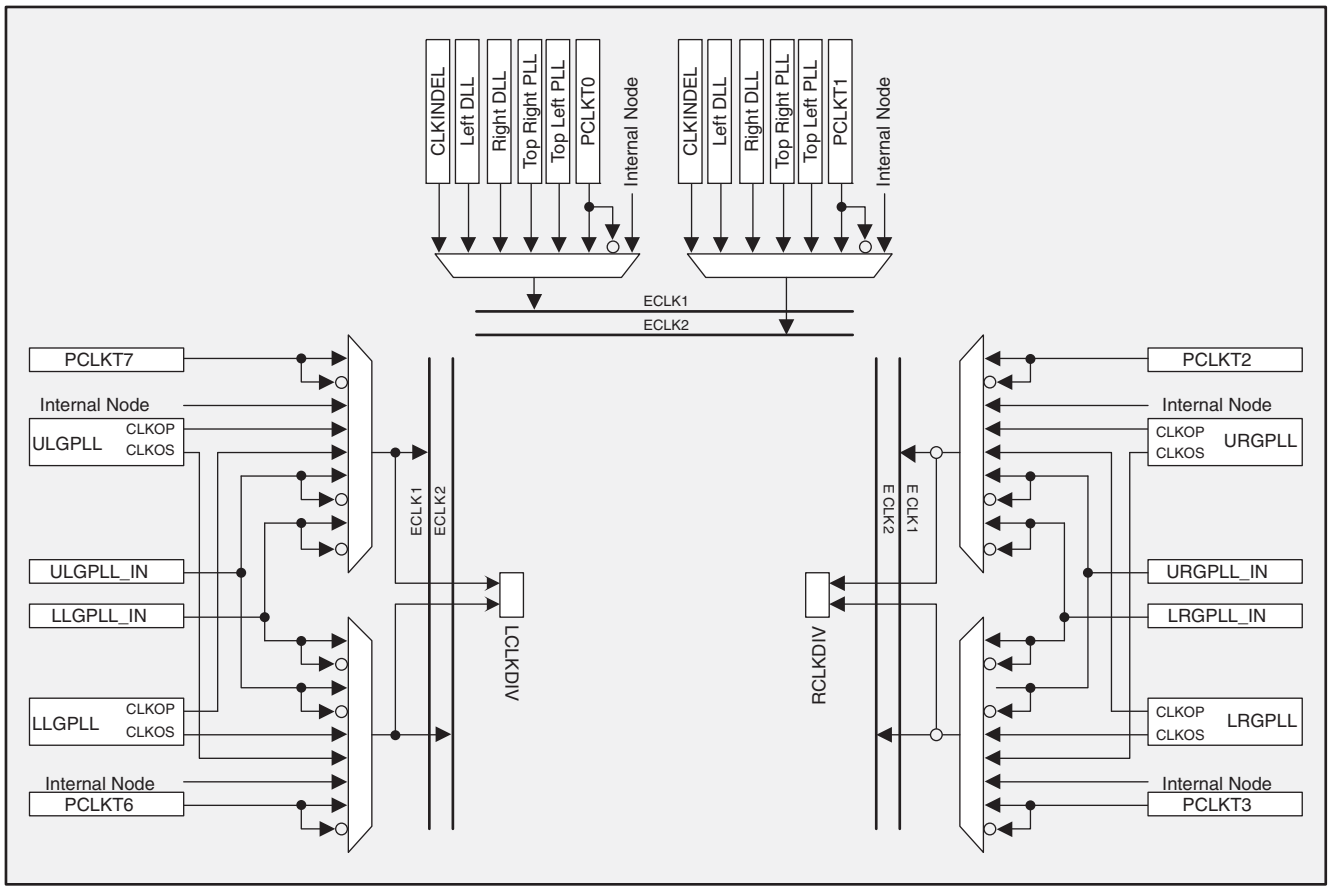


*For CLK6 and CLK7 muxes, the PLL, DLL and PCLK PIOs, inputs are not fully populated.
Note: Use EPIC to see exact routing

Appendix B. PLL, CLKIDV and ECLK Locations and Connectivity

Figure 10-36 shows the locations, site names and connectivity of the PLLs, CLKDIVs and ECLKs

Figure 10-36. PLL, CLKIDV and ECLK Locations and Connectivity



Appendix C. Lattice Diamond Usage Overview

This appendix discusses the use of Lattice Diamond design software for projects that include the LatticeECP2M SERDES/PCS module.

For general information about the use of Lattice Diamond, refer to the Lattice Diamond Tutorial.

If you have been using ispLEVER software for your FPGA design projects, Lattice Diamond may look like a big change. But if you look closer, you will find many similarities because Lattice Diamond is based on the same toolset and work flow as ispLEVER. The changes are intended to provide a simpler, more integrated, and more enhanced user interface.

Converting an ispLEVER Project to Lattice Diamond

Design projects created in ispLEVER can easily be imported into Lattice Diamond. The process is automatic except for the ispLEVER process properties, which are similar to the Diamond strategy settings, and PCS modules. After importing a project, you need to set up a strategy for it and regenerate any PCS modules.

Importing an ispLEVER Design Project

Make a backup copy of the ispLEVER project or make a new copy that will become the Diamond project.

1. In Diamond, choose **File > Open > Import ispLEVER Project**.
2. In the ispLEVER Project dialog box, browse to the project's .syn file and open it.
3. If desired, change the base file name or location for the Diamond project. If you change the location, the new Diamond files will go into the new location, but the original source files will not move or be copied. The Diamond project will reference the source files in the original location.

The project files are converted to Diamond format with the default strategy settings.

Adjusting PCS Modules

PCS modules created with IPexpress have an unusual file structure and need additional adjustment when importing a project from ispLEVER. There are two ways to do this adjustment. The preferred method is to regenerate the module in Diamond. However this may upgrade the module to a more recent version. An upgrade is usually desirable but if, for some reason, you do not want to upgrade the PCS module, you can manually adjust the module by copying its .txt file into the implementation folder. If you use this method, you must remember to copy the .txt file into any future implementation folders.

Regenerate PCS Modules

1. Find the PCS module in the Input Files folder of File List view. The module may be represented by an .lpc, .v, or .vhd file.
2. If the File List view shows the Verilog or VHDL file for the module, and you want to regenerate the module, import the module's .lpc file:
   a. In the File List view, right-click the implementation folder () and choose **Add > Existing File**.
   b. Browse for the module's .lpc file, ***<module_name>*.lpc**, and select it.
   c. Click **Add**. The .lpc file is added to the File List view.
   d. Right-click the module's Verilog or VHDL file and choose **Remove**.
3. In File List, double-click the module's .lpc file. The module's IPexpress dialog box opens.
4. In the bottom of the dialog box, click **Generate**. The Generate Log tab is displayed. Check for errors and close.

In File List, the .lpc file is replaced with an .ipx file. The IPexpress manifest (.ipx) file is new with Diamond. The .ipx file keeps track of the files needed for complex modules.

Using IPexpress with Lattice Diamond

Using IPexpress with Lattice Diamond is essentially same as with ispLEVER.

The configuration GUI tabs are all the same except for the Generation Options tab. Figure 10-37 shows the Generation Options tab window.

Figure 10-37. Generation Options Tab

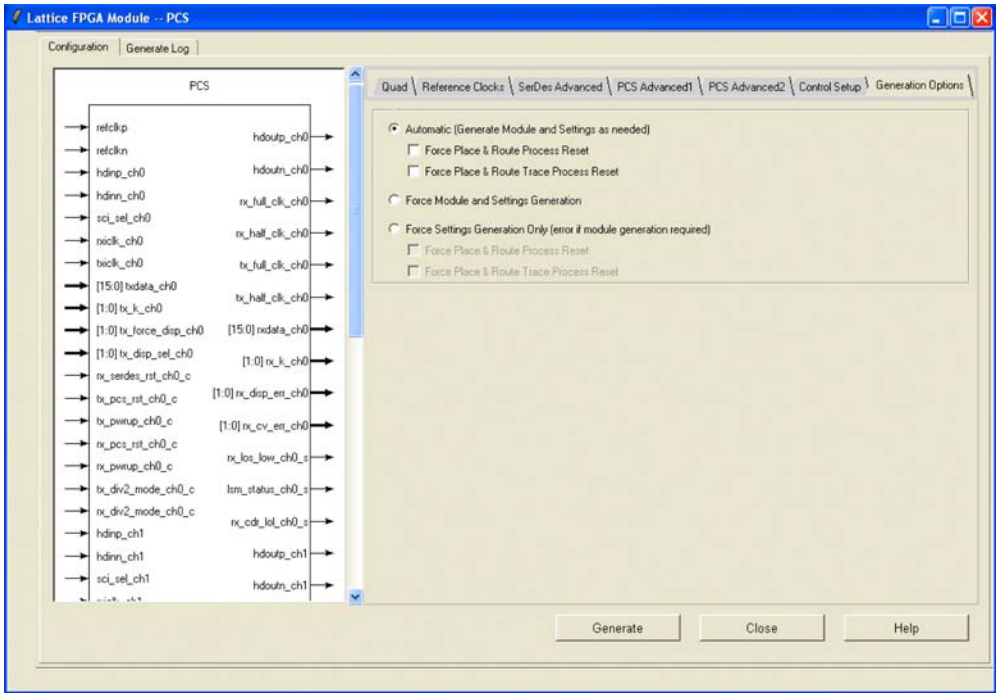


Table 10-18. SERDES_PCS GUI Attributes – Generation Options Tab

GUI Text	Description
Automatic	Automatically generates the HDL and configuration(.txt) files as needed. Some changes do not require regenerating both files.
Force Module and Settings Generation	Generates both the HDL and configuration files.
Force Settings Generation Only	Generates only the attributes file. You get an error message if the HDL file also needs to be generated.
Force Place & Route Process Reset	Resets the Place & Route Design process, forcing it to be run again with the newly generated PCS module.
Force Place & Route Trace Process Reset	Resets the Place & Route Trace process, forcing it to be run again with the newly generated PCS module.

Note:
Automatic is set as the default option. If either Automatic or Force Settings Generation Only and no sub-options (Process Reset Options) are checked and the HDL module is not generated, the reset pointer is set to Bitstream generation automatically.

After the Generation is finished, the reset marks in the process window will be reset accordingly.

Creating a New Simulation Project Using Simulation Wizard

This section describes how to use the Simulation Wizard to create a simulation project (.spf) file so you can import it into a standalone simulator.

1. In Project Navigator, click **Tools > Simulation Wizard**. The Simulation Wizard opens.
2. In the Preparing the Simulator Interface page click **Next**.
3. In the Simulator Project Name page, enter the name of your project in the Project Name text box and browse to the file path location where you want to put your simulation project using the Project Location text box and Browse button.

   When you designate a project name in this wizard page, a corresponding folder will be created in the file path you choose. Click **Yes** in the popup dialog that asks you if you wish to create a new folder.
4. Click either the Active-HDL® or ModelSim® simulator check box and click **Next**.
5. In the Process Stage page choose which type of Process Stage of simulation project you wish to create Valid types are RTL, Post-Synthesis Gate-Level, Post-Map Gate-Level, and Post-Route Gate-level+Timing. Only those process stages that are available are activated.

   Note that you can make a new selection for the current strategy if you have more than one defined in your project.

   The software supports multiple strategies per project implementation which allow you to experiment with alternative optimization options across a common set of source files. Since each strategy may have been processed to different stages, this dialog allows you to specify which stage you wish to load.
6. In the Add Source page, select from the source files listed in the Source Files list box or use the browse button on the right to choose another desired source file. Note that if you wish to keep the source files in the local simulation project directory you just created, check the **Copy Source to Simulation Directory** option.
7. Click **Next** and a Summary page appears and provides information on the project selections including the simulation libraries. By default, the Run Simulator check box is enabled and will launch the simulation tool you chose earlier in the wizard in the Simulator Project Name page.
8. Click **Finish**.

The Simulation Wizard Project (.spf) file and a simulation script DO file are generated after running the wizard. You can import the DO file into your current project if desired. If you are using Active-HDL, the wizard will generate an .ado file and if you are using ModelSim, it creates and .mdo file.

Note: PCS configuration file, (.txt) must be added in step 6.

Figure 10-38. Diamond Spreadsheet View (see Figure 10-6 for ispLEVER Equivalent)

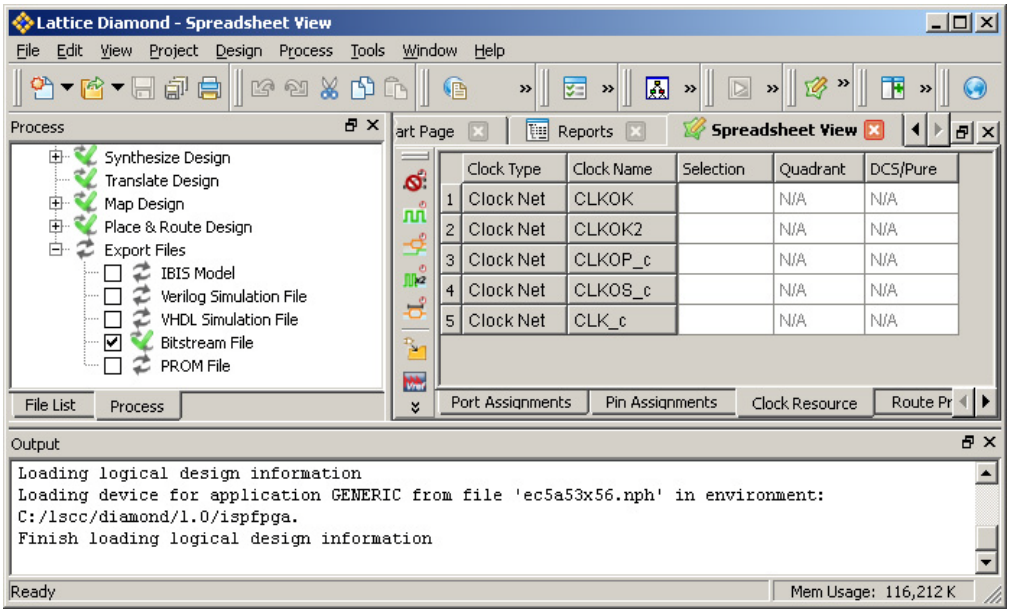


Figure 10-39. Diamond IPexpress Main Window (see Figure 10-14 for ispLEVER Equivalent)

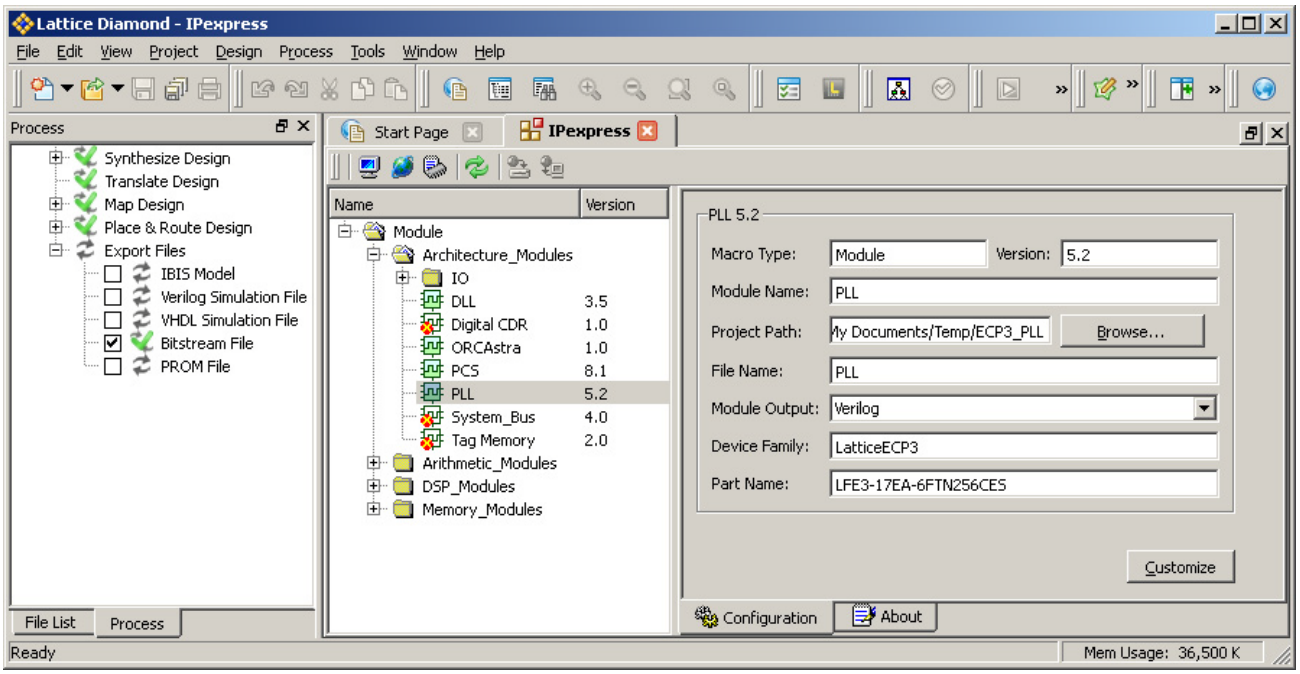


Figure 10-40. Diamond LatticeECP3 PLL Configuration Tab (see Figure 10-15 for ispLEVER Equivalent)

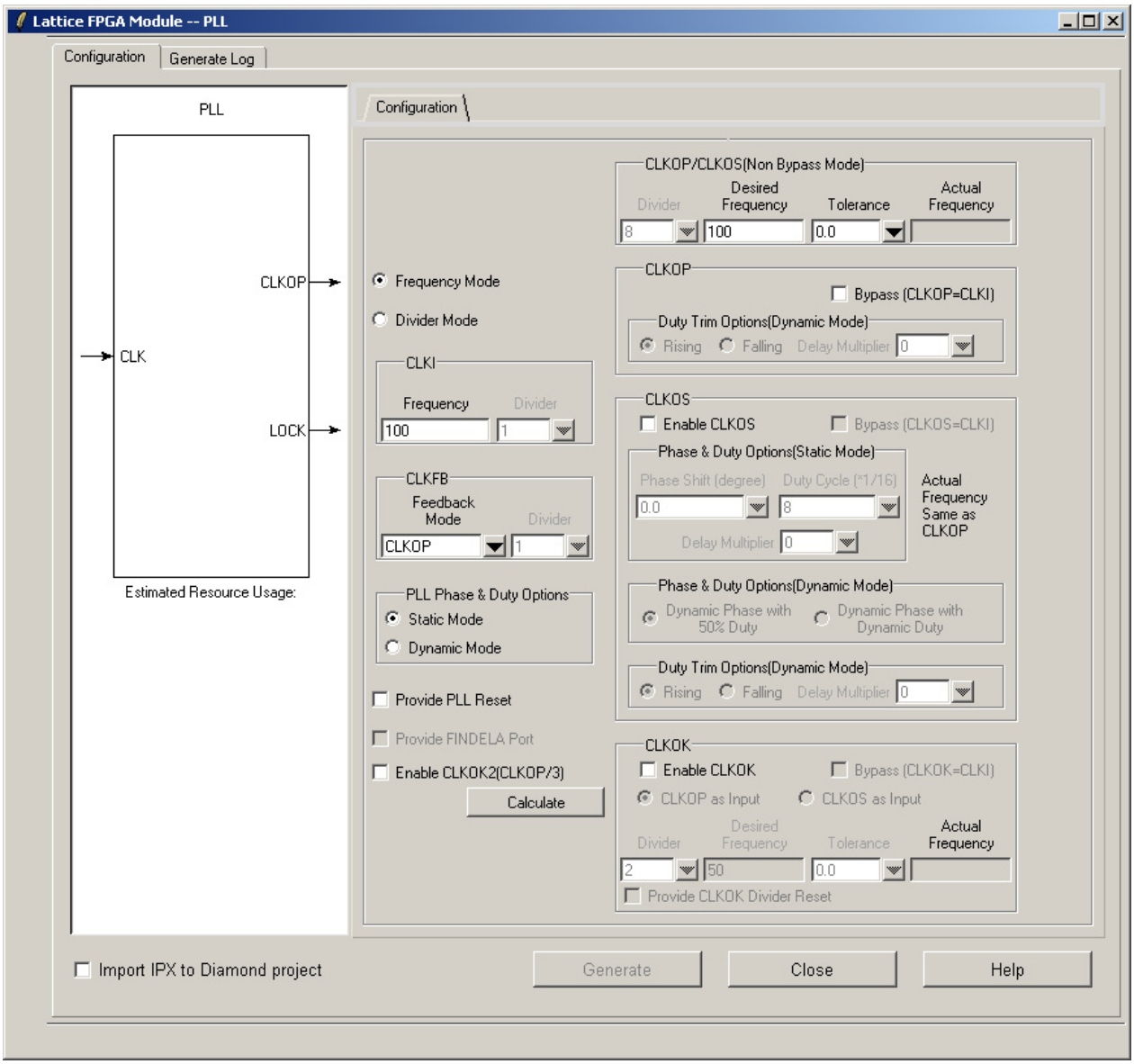


Figure 10-41. Diamond LatticeECP3 IPexpress DLL Configuration Tab (see Figure 10-22 for ispLEVER Equivalent)