sysCLOCK™ PLL Usage Guide for ispXPGA®, ispGDX2,™ ispXPLD™ and ispMACH™ 5000VG Devices

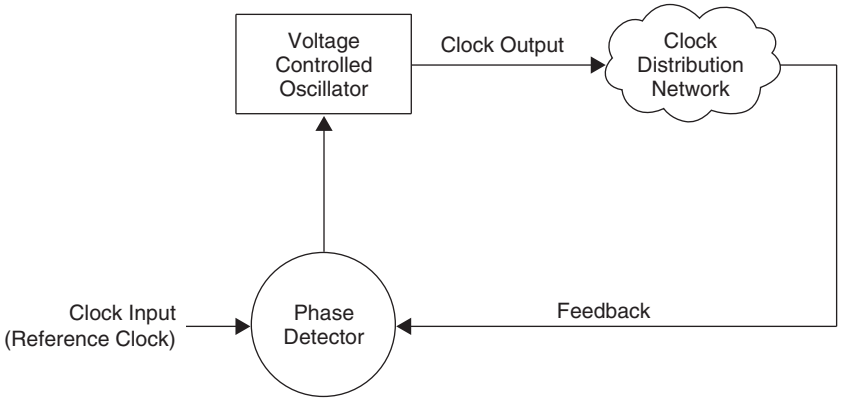
September 2004 Technical Note TN1003

Introduction

As programmable logic devices (PLDs) grow in size and complexity, on-chip clock distribution becomes a major factor in performance. The delay and skew of the clocks significantly affect the performance of the device. Furthermore, distribution of these clock signals to other devices on the board increases the complexity of the design. To compensate for these effects, many of the Lattice devices include phase locked loops (PLLs) referred to as sysCLOCK PLLs (Table 1).

Lattice's sysCLOCK PLLs can be used to align the clock tree, distribute multiple clock frequencies, perform duty cycle correction, and provide phase shift. As with most traditional PLLs, the internal PLLs compare the input clock and output clock and compensate for the offset by adjusting the output frequency and phase through a voltage controlled oscillator (VCO) and phase detector (Figure 1).

Figure 1. Phase-Locked Loop Block Diagram



The sysCLOCK PLL includes dividers on the input, output, and feedback lines, which allow the PLL to synthesize various output clock frequencies. In addition to the dividers, there are delay elements on the input and feedback lines which shift the internal clock to allow the optimization of set-up and clock-to-out times. The PLLs also perform duty cycle correction, which results in a stable 50/50-output duty cycle for various input duty cycles.

These features give designers the flexibility of creating a variety of clock signals within the PLD. This simplifies board design and reduces cost, because designers no longer need external circuitry to create these clocks. The PLL can further simplify the design by using the external feedback pin to align the clock at the board level.

Table 1. Lattice Device Families with sysCLOCK

Device Family	sysCLOCK PLLs	Input Frequency Range (MHz)	Output Frequency Range (MHz)	Delay Step
ispMACH 5000VG	2	5 - 180	5 - 180	500 ps
ispXPLD 5000MX	2	10 - 320	10 - 320	250 ps
ispXPGA	8	10 - 320	10 - 320	250 ps
ispGDX2	Up to 4	10 - 320	10 - 320	333 ps

Note: Refer to the device data sheets for additional specifications.

PLLs vs. DLLs

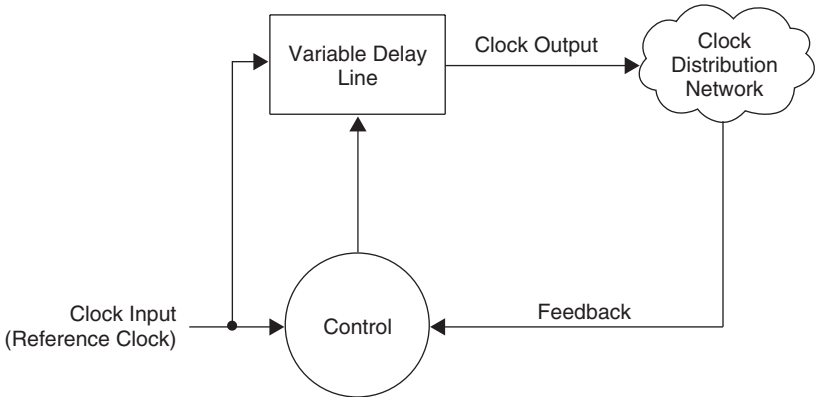
Today, two forms of clock synthesis and control (PLLs and DLLs) are embedded in CPLDs and FPGAs. Phase locked loops (PLLs) are the oldest and most widely used form. Delay locked loops (DLLs) are newer, but use the same basic concepts. Each has its own advantages and disadvantages.

PLLs are based on analog circuitry containing a voltage-controlled oscillator and a phase detector (Figure 1). The phase detector references the input clock and feedback signal to determine the relationship between the two and instructs the oscillator to either speed up or slow down the clock signal in order to make the input and feedback match.

DLLs are based on digital circuitry containing a delay element and phase comparator (Figure 2). The input clock and a signal from the phase comparator are both fed to the delay element, which introduces delay between the input clock and the feedback until they are in phase. The phase comparator uses the feedback signal from the DLL output and the clock input to determine the relationship and send a control signal to the delay element.

While DLLs tend to migrate between process technologies more easily due to their digital nature, they are somewhat limited in their functionality. PLLs, however, can provide much more flexible clock multiplication, division, control, and delay functionality.

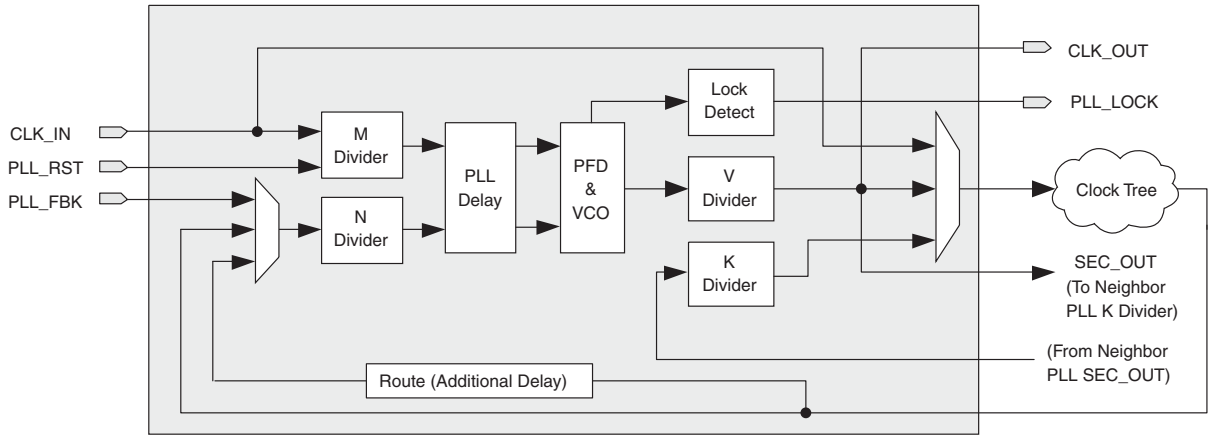
Figure 2. Delay Locked Loop Block Diagram



sysCLOCK PLL

The sysCLOCK PLL receives its clock inputs from the global clock pins of the device and provides outputs to the global clock nets. Additionally, each PLL has a set of *PLL_RST*, *PLL_FBK*, and *PLL_LOCK* signals used for external control. Figure 3 shows the sysCLOCK PLL block diagram.

Figure 3. sysCLOCK PLL Block Diagram



Input Clock (M) Divider

The Input Clock (M) Divider is used to control the input clock frequency into the PLL block. It can be set to an integer value of 1 to 32. 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 device data sheet. Therefore, the value of the Input Divider significantly affects the input clock frequency range.

Feedback Loop (N) Divider

The Feedback Loop (N) Divider is used to divide the feedback signal. 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. Like the input divider, the feedback loop divider can be set to an integer value of 1 to 32. The input and output of the Feedback Divider must be within the input and output frequency ranges specified in the device data sheet. Therefore, the value of the Feedback Divider significantly affects the input and output clock frequency range.

Variable Delay Block

The PLL can insert or remove delay from *CLK_OUT* by inserting delay on the input or feedback lines. This effectively gives the PLL phase shift capabilities. See the device data sheet for the delay range and delay increments.

VCO and Phase Detector Block

The operation of the VCO and Phase Detector block is the same as the basic PLL block described previously. It synchronizes the input clock and feedback signals and performs duty cycle correction. When there is an input clock (*CLK_IN*) signal the VCO and Phase Detector will oscillate to produce the correct output clock. When there is no initial signal on *CLK_IN*, the VCO and Phase Detector will not start oscillating. When the *CLK_IN* signal is held high, held low, disconnected or falls below the minimum input frequency the VCO and Phase Detector will stop oscillating after a finite amount of time, which reduces power consumption. The input and output signals of the PLL block must be within the frequency ranges specified in the device data sheet.

Post-Scalar (V) Divider

The Post-scalar (V) Divider compensates for the frequency range of the voltage-controlled oscillator output (FVCO). This allows the input and output clocks to run at lower frequencies without compromising the stability of the PLL. It can be set to values of 1, 2, 4, 8, 16 or 32.

Secondary Clock (K) Divider

The Secondary Clock (K) Divider feeds the global clock net. It divides the *CLK_OUT* signal of the PLL by the value of the divider. It can be set to values of 2, 4, 8, 16 or 32. By design, a setting of 1 is not available as this would result in an unnecessary duplication of the clocks.

CLK_IN Input

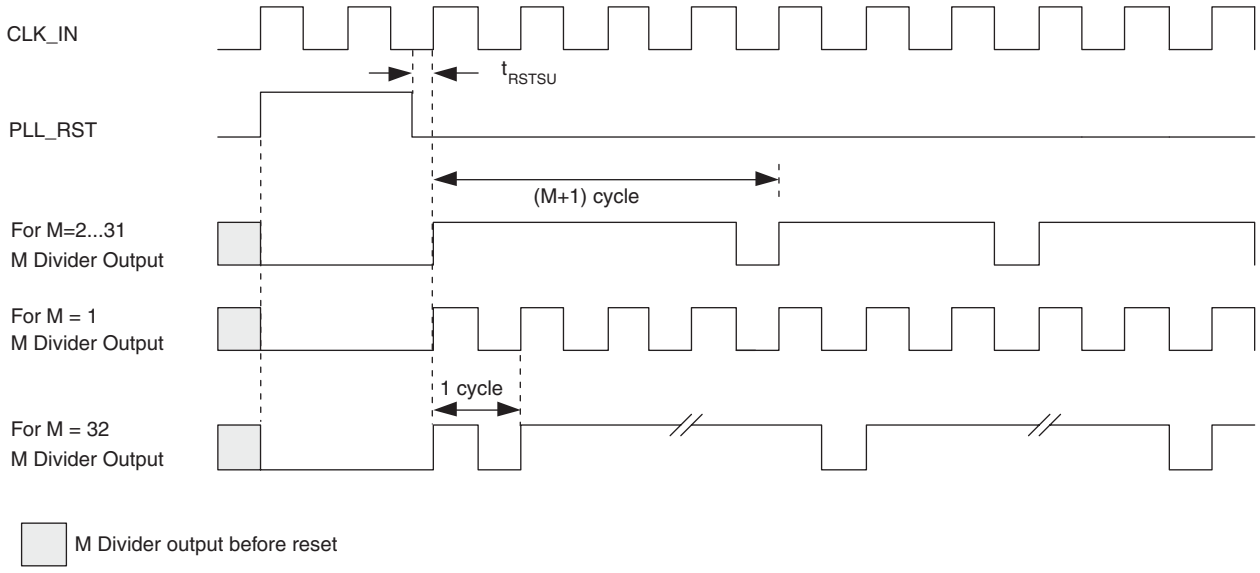
The global clock pins provide the *CLK_IN* signal for the sysCLOCK PLLs. The *CLK_IN* signal is the reference clock for the PLL. *CLK_IN* must conform to the specifications in the data sheet in order for the PLL to operate correctly.

PLL_RST Input

The *PLL_RST* signal resets the timing of *CLK_IN* (M) Divider Output referenced to *CLK_IN* as shown in Figure 4. The *PLL_RST* signal is not required, and if not used will be set to logic 0. The *PLL_RST* pin is a dual-purpose pin that can be configured as *PLL_RST* or a regular I/O signal. This signal is used to reference the starting point of the PLL (the first edge of the input clock). The PLL_RST signal is active high.

The *PLL_RST* signal must be asserted for the minimum reset pulse width and de-asserted within the reset recovery time before the clock, as defined in the device data sheet. The *PLL_RST* pin is most commonly used when multiple sysCLOCK PLLs are dividing the same input clock and a reset signal is needed to synchronize the PLLs (Figure 4).

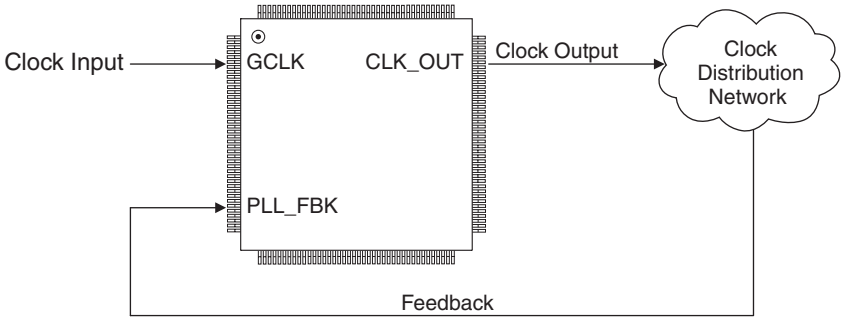
Figure 4. PLL_RST Timing Diagram (Example M=4)



PLL_FBK Input

The feedback signal to the PLL, which is fed through the feedback divider can be derived from the global clock net or the *PLL_FBK* pin. Feedback must be supplied in order for the PLL to synchronize the input and output clocks. The *PLL_FBK* pin is a dual-purpose pin that can be configured as either *PLL_FBK* or a regular I/O signal. The external feedback allows the designer to compensate for board-level clock alignment. Figure 5 is an example of a possible configuration of the sysCLOCK PLL using the external *PLL_FBK* pin.

Figure 5. PLL_FBK Configuration



CLK_OUT Output

The sysCLOCK PLL primary clock output, *CLK_OUT*, drives its associated clock net. Depending on the device, *CLK_OUT* can also drive either routing or a dedicated dual-purpose pin. For those devices using dual-purpose pins, they are configurable as either CLK_OUT pins or regular I/O pins, and the *CLK_OUT* signal is directly connected to its associated CLK_OUT pin.

SEC_OUT Output

The secondary clock output, SEC_OUT, drives its neighbor, PLL clock net. The neighbor PLL is defined in the data sheet.

When SEC_OUT is used, the neighbor PLL is not available as a PLL.

PLL_LOCK Output

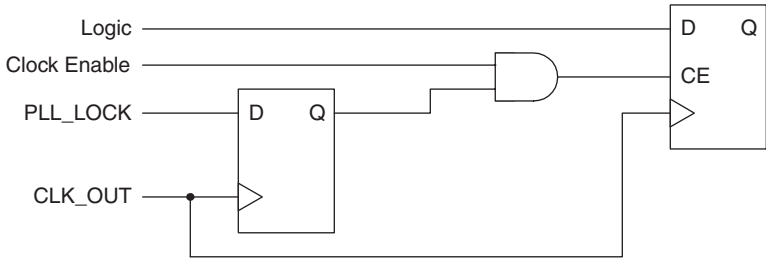
The *PLL_LOCK* output provides information about the status of the PLL. After the device is powered up and the input clock is valid the PLL will achieve lock within the specified lock time (t_{LOCK}). Once lock is achieved the PLL lock signal will be asserted. If during operation the input clock or feedback signals to the PLL become invalid the PLL will lose lock. It takes time from the occurrence of the invalid signals until the *PLL_LOCK* signal is de-asserted. Refer to the appendices for device-specific information.

In ispGDX2 devices, *PLL_LOCK* is routed to a dedicated pin.

Design Tips

1. Care must be taken not to violate the input jitter specification.
2. The input clock frequency must not exceed the specification detailed in the device data sheet. The divider settings will effect the input and output frequency range of the PLL.
3. For the software to generate the best possible results, an input clock frequency should always be specified.
4. The divider settings cannot produce frequencies outside the range specified in the device data sheet.
5. The lowest common denominator should be used for multiply and divide values to maximize the input frequency range.
6. The external *PLL_FBK* signal should be generated from the *CLK_OUT* signal.
7. If the *PLL_LOCK* signal is used as a clock enable, it should be synchronized before being used by the registered logic. The synchronization ensures that setup and hold times are not violated when *PLL_LOCK* is asserted. Figure 6 shows a common example.
8. When the PLL is not used, the V_{CCP} and GNDP pins should be electrically connected to V_{CC} and GND, respectively.
9. When the global clock pins are not used, they should be treated as no connects.
10. When using the external *CLK_OUT* pin to output the PLL primary output clock, the number of I/Os switching in the same bank as the *CLK_OUT* pin significantly affects the amount of jitter on this pin. Care should be taken to reduce the number of switching I/Os in the bank to reduce the jitter on the *CLK_OUT* pin.

Figure 6. PLL_LOCK Common Usage Example



PLL Attributes

The PLL utilizes several attributes that allow the configuration of the PLL through source constraints. The following section details these attributes and their usage. Appendix A lists the attributes and the macros that utilize them.

IN_FREQ

The input frequency can be any value within the specified frequency range based on the divider settings. If the divider settings are invalid, the software will generate and error. To determine if your divider settings are valid, use the equations in Appendix B.

MULT, DIV, POST and SECDIV

The M, N, V and K dividers correspond directly to the DIV, MULT, POST, and SECDIV values respectively. The user is not allowed to input an invalid combination; determined by the input frequency, the dividers, and the PLL specifications.

PLL_DLY

The "PLL_DLY" attribute is used to pass the Delay factor associated with the Output Clock of the PLL. This allows the user to advance or retard the Output Clock by the value passed multiplied by PLL Delay Increment specified in the data sheet as t_{PLL_DELAY}.

CLK_OUT_TO_PIN

The "CLK_OUT_TO_PIN" attribute is used to configure the CLK_OUT pin as the PLL Output Clock or as a regular I/O pin. This attribute allows the designer the ability to route the global clock net associated with the PLL to the CLK_OUT pin for observation or distribution on the board.

WAKE_ON_LOCK

The WAKE_ON_LOCK cell determines if the device will wait for the PLL to lock before beginning the wake-up process. If the attribute is set to "ON", the device will not wake up until the PLL_LOCK signal for the given PLL is active. If is set to "OFF", the device will wake up regardless of the state of the PLL_LOCK signal.

PLL_FBK_ATTRIBUTE

This attribute is designed for ispXPGA only. Default is CLKTREE (even when user do not add this attribute in HDL).

The ROUTE attribute allows users to add additional delay to the PLL_FBK.

Software Usage

The PLL is not an inherent part of most digital design tools. With the addition of PLLs to PLDs, there must be provisions made to fully utilize the PLL. These provisions include VHDL, Verilog and ABEL components, and user constraints in the place and route tools. The following sections describe how to utilize the PLL in the Lattice design tools.

Macro Definitions

The Lattice libraries contain components to allow designers to utilize the PLL. Each component has several attributes associated with it for configuring the PLL. Appendix A lists these attributes, their meaning and the symbols associated with the attribute.

Figure 7 shows the library symbol for a simple PLL (SPLL). This component is used to produce a zero delay/skew clock using the PLL. It is the PLL with all the dividers set to one, the feedback generated internally, and the reset disabled.

Figure 7. Simple PLL Component

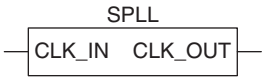


Figure 8 shows the library symbol for the Standard PLL (STDPLL). This macro has the PLL_LOCK output and attributes for setting the multiply, divide, and post-scalar divider factors for the PLL and the programmable delay.

Figure 8. Standard PLL Component

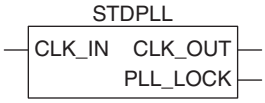
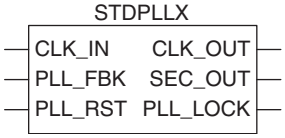


Figure 9 shows the library symbol Extended PLL (STDPLLX). This symbol gives full access to all features of the PLL including external reset and feedback.

Figure 9. Extended PLL Component



PLL Usage in Module/IP Manager and HDL

Including sysCLOCK PLLs in a Design

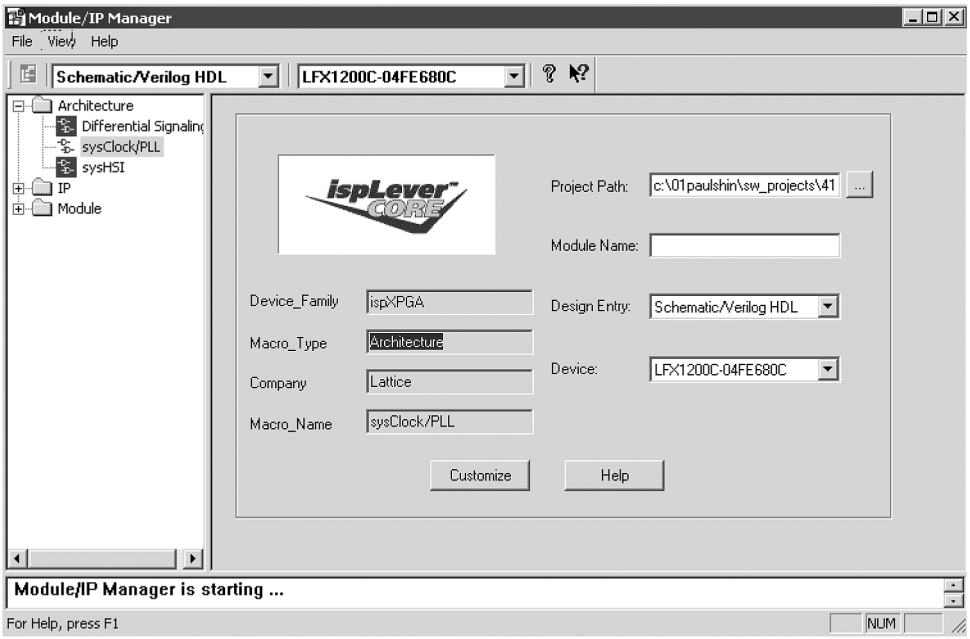
The sysCLOCK PLL capability can be accessed either through the Module/IP Manager or directly instantiated in a design's source code. The following sections describe both classes of usage.

Module/IP Manager Usage

The sysCLOCK PLL in ispXPGA, ispGDX2 and ispXPLD use identical architecture and is fully supported in Module/IP Manager in the Lattice ispLEVER® software. The Module/IP Manager allows the user to define the desired PLL using a simple, easy-to-use GUI. Following definition, a VHDL or Verilog module that instantiates the desired PLL is created. This module can be included directly in the user's design.

Figure 10 shows the main window when PLL is selected. The only entry required in this window is the module name. After entering a module name, click "Customize" to open the "Customizing" window, as shown in Figure 11.

Figure 10. Module/ IP Manager Main Window



In the Main window, when sysClock/PLL is selected, the user only needs to enter the Module Name. Other entries are already set when the project is created. The user can enter different Design Entry, Device and Project Path parameters if desired.

Normal Mode Window

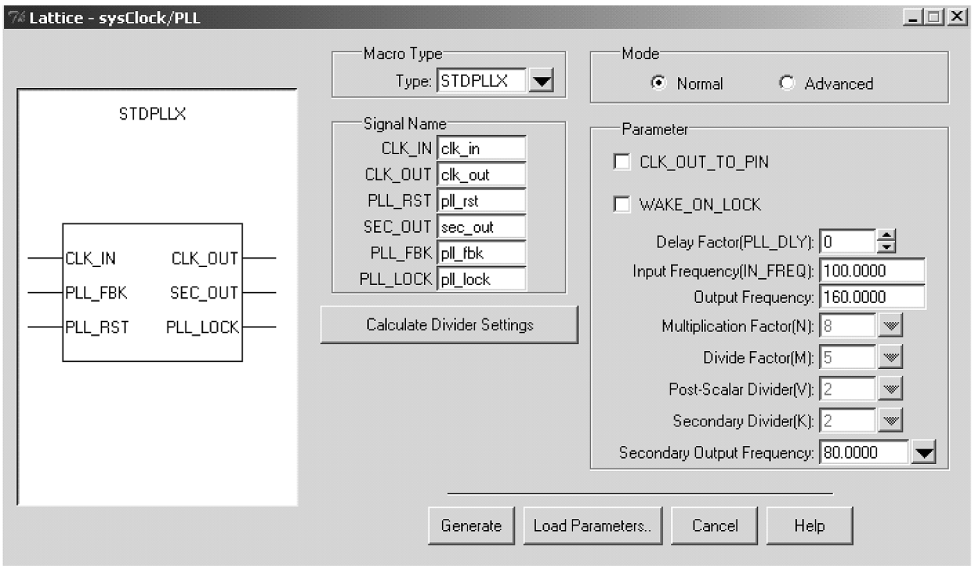
The next window starts with the default mode called 'Normal Mode'. In this mode, the user sets the input and output frequency and the GUI will calculate the divider settings.

This Window provides the ability to define the following:

- Macro type
- Signal name
- Mode of configuration
- CLK_OUT_TO_PIN option
- WAKE_ON_LOCK option
- PLL_FBK_ATTRIBUTE option for XPGA
- Input frequency and output frequency

- Calculate the divider settings for the user

Figure 11. Normal Mode Window



Advanced Mode Window

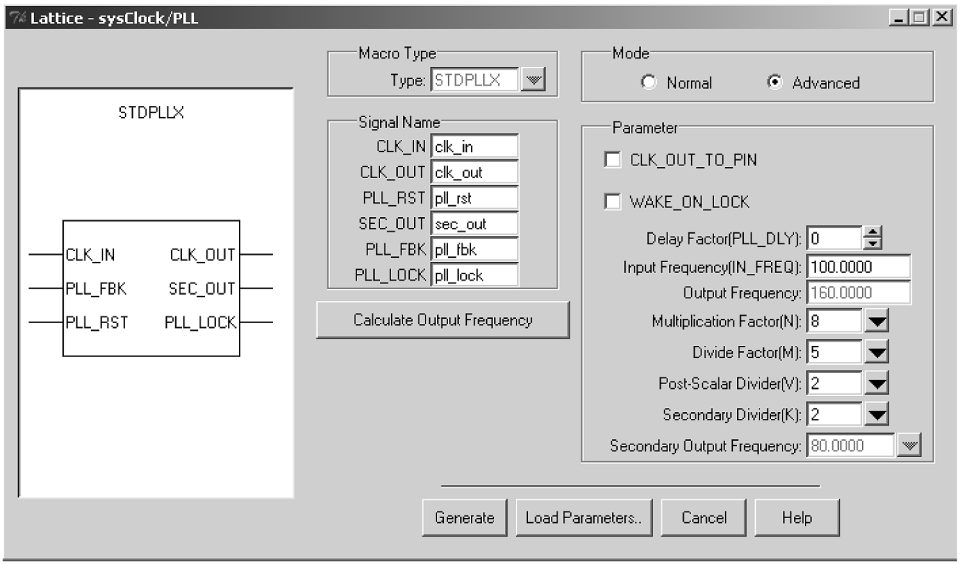
The window will change slightly when 'Advanced Mode' is selected. In this mode, the user sets input and divider settings. The GUI will calculate output frequency automatically for the user.

The Customizing Window provides the ability to define the following:

- Macro type
- Signal name
- Mode of configuration
- CLK_OUT_TO_PIN option
- WAKE_ON_LOCK option
- PLL_FBK_ATTRIBUTE option for XPGA
- Input frequency and divider settings
- Calculate output frequency for the user

Clicking 'Generate' creates a VHDL (module name.vhd and module name_sim.vhd) or Verilog (module name.v, module name.lpc, module name_sim.v and module nameheader.v) file in the working directory that instantiates the core. The load parameters button can be used to reload configurations from previously created parameter files (*.lpc files).

Figure 12. Advanced Mode Window



Direct Instantiation Into Source Code

If desired, the Module/IP Manager can be bypassed and the sysCLOCK PLL can be instantiated directly in the source code. Appendix A provides examples of source code generated by the Module/IP Manager. These examples can be used as templates for directly instantiating the sysCLOCK PLL in the source code.

PLL Usage in the ispLEVER Constraint Editor

The ispLEVER Constraint Editor includes a PLL Attribute Sheet. This sheet gives the user the ability to view the settings of the sysCLOCK PLL instantiation. Table 2 is an illustration of the PLL Attribute Sheet.

Table 2. PLL Attributes Sheet

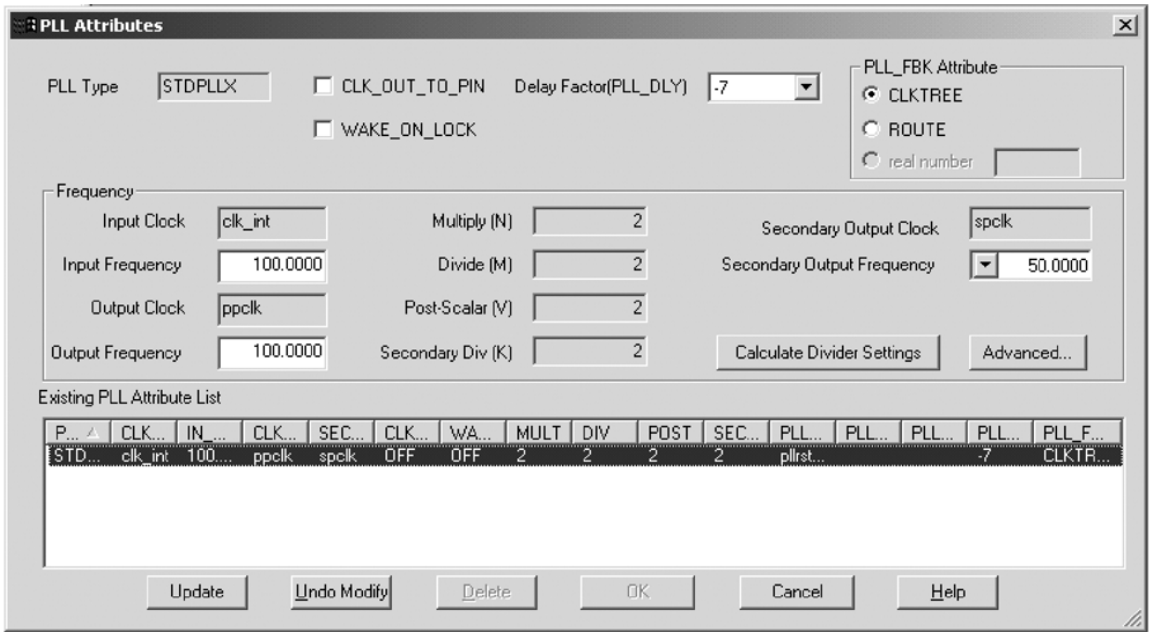
PLL Type	Input Clock	Input Frequency	Output Clock	Secondary Clock	CLK_OUT_TO_PIN	WAKE_ON_LOCK	Multiply	Divide	SecDiv	PLL_RST	PLL_FBK	PLL_LOCK	PLL_DLY	PLL_FBK_Attribute
STDPLLX	CLK_INT	100.0000	PPCLK	SPCLK	OFF	OFF	2	2	2	PLLRST			-7	CLKTREE

Settings can be fine-tuned without changing the design source by using the PLL Attributes Settings window.

PLL Attributes Window

The ispLEVER Constraint Editor also includes a PLL Attributes Window. This window includes two sections (Frequency and Existing PLL Attributes). At the top of the window there is a grayed text box displaying the PLL type, a check box for setting the CLK_OUT_TO_PIN attribute, and a dial box for setting the PLL_DLY attribute. Figure 13 illustrates the PLL Attributes Window.

Figure 13. PLL Attributes Window

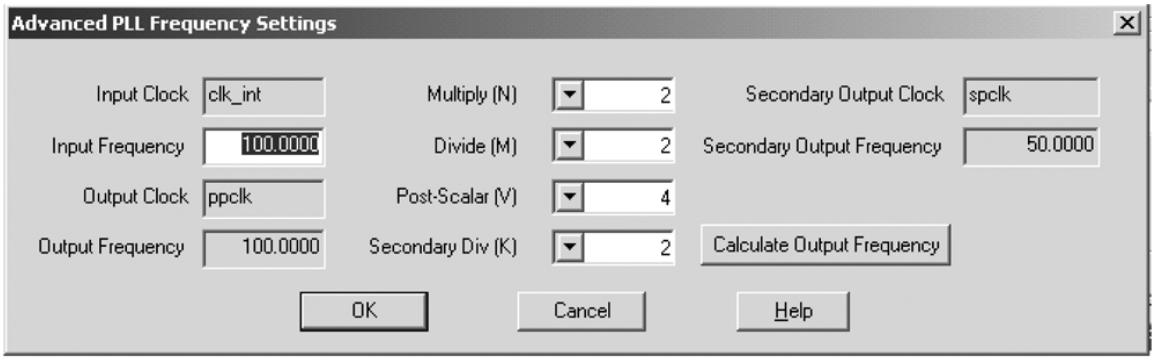


The “Frequency” section has text boxes to modify the input frequency and output frequency, grayed text boxes displaying the input clock, output clock, divider settings, and secondary output clock. There is a grayed drop-down box that allows the user to select a Secondary Output Frequency. One of the 5 possible frequencies, given the current output frequency (access to the K-Divider without actually setting the K-Divider). When a Secondary Output Frequency is selected from the drop-down box, the K-Divider setting will change to the setting that corresponds to that frequency. There are two buttons labeled “Calculate Divider Settings” and “Advanced”. The “Calculate Divider Settings” button updates the M, N, and V divider settings and Secondary Output Frequency when the input or output frequencies are modified. If an output frequency is entered that can not be achieved, a warning message displays the closest obtainable output frequency. The “Advanced…” button opens the “Advanced PLL Frequency Settings” window.

The “Existing PLL Attributes List” section has a window displaying the current PLL attributes from the PLL Attributes Sheet. When a PLL is selected in this window and the “Modify” button is selected, that PLL becomes available for modification and all of its settings are displayed.

The “Advanced PLL Frequency Settings” window has the same appearance as the “Frequency” section of the PLL Attributes window. However, the Output Frequency is grayed out and the Divider settings are displayed in the middle of the window. This window allows the user to update the divider settings and calculate the resulting output frequency. Figure 14 illustrates the Advanced PLL Frequency Settings Window.

Figure 14. Advanced PLL Frequency Settings Window



The Constraint Editor determines the input and output frequencies based on the selected divider settings. If the input frequency given in the design source is out of range, the Constraint Editor will flag the problem and report the possible input frequency range based on the divider settings.

Furthermore, the *CLK_OUT_TO_PIN* attribute can be set from within the Constraint Editor. This allows the designer to route the PLL output to the *CLK_OUT* pin for evaluation without changing the design source.

Input Frequency

The input frequency can be any value within the specified frequency range based on the divider settings. If the divider settings are invalid, the Constraint Editor will generate and error. To determine if your divider settings are valid, use the equations in Appendix B.

Divider Configuration

The M, N, V and K dividers correspond directly to the DIV, MULT, POST, and SECDIV values in the Constraint Editor, respectively. The user is not allowed to input an invalid combination; determined by the input frequency, the dividers, and the PLL specifications.

PLL_RST

The PLL_RST cell automatically displays the pin or node name from the source file that is connected to the reset line. If there is no reset defined, the PLL_RST cell will be empty and the PLL will only reset on power-up.

PLL_FBK

The PLL_FBK cell automatically displays the pin or node name from the source file that is connected to the feedback line. If there is no feedback defined, the PLL_FBK cell will be empty and the PLL will use the internal feedback in the device.

PLL_LOCK

The PLL_LOCK cell automatically displays the pin or node name from the source file that is connected to the lock line. If a lock signal is not defined in the source file, the PLL_LOCK cell will be empty and the lock signal will not be available.

CLK_OUT_TO_PIN

The CLK_OUT_TO_PIN cell displays the state of the CLK_OUT signal being routed to the dedicated CLK_OUT pin. If it is routed to the pin, the cell will display “ON”. If it is not routed out to the pin, the cell will display “OFF”. By default, the CLK_OUT_TO_PIN attribute is “OFF”. However, if the design source declares the CLK_OUT signal as an output, the CLK_OUT_TO_PIN attribute will be ignored and the CLK_OUT signal will be routed to the CLK_OUT pin.

WAKE_ON_LOCK

The WAKE_ON_LOCK cell determines if the device will wait for the PLL to lock before beginning the wake-up process. If the cell displays “ON”, the device will not wake up until the PLL_LOCK signal for the given PLL is active. If the cell displays “OFF”, the device will wake up regardless of the state of the PLL_LOCK signal.

PLL_DLY

The PLL_DLY value defaults to an empty cell, resulting in zero delay inserted. The cell sets the number of delay increment steps. The delay increment value is specified in the device data sheet as $t_{PLL\_DELAY.}$ Modifying the value in the PLL delay cell advances or delays the CLK_OUT signal by the set value multiplied by the delay increment. Negative values specify advancement, and positive values add delay.

Timing Analysis and Simulation with PLLs

The use of the sysCLOCK PLL feature in Lattice devices significantly affects the timing of the device. The following cases outline the timing analysis and simulation implications of many common uses of the sysCLOCK PLLs. In all cases, the divider settings and delay settings of the PLL are included in the simulation of the device. The simulation does not compensate for external delays and dividers in the feedback loop. Furthermore, the *PLL_LOCK* signal is not simulated according to the t_{LOCK} specification. The *PLL_LOCK* signal will appear active shortly after the simulation begins, but will remain active throughout the simulation.

Case 1. Internal Clock Net Internal Feedback

When the registers of the design are driven by the output clock of the PLL (*CLK_OUT*) and the *PLL_FBK* signal is generated internally, the Lattice design tools automatically adjust the delay associated with the clock net and the resulting simulation mimics the device behavior.

Case 2. External Clock Internal Feedback

When the PLL drives a clock signal off chip but derives its feedback internally, the timing of the clock output signal (*CLK_OUT*) at the device pin relative to the input clock (GCLK) at the device pin is defined by the $t_{CLK\_OUT\_DLY}$ specification in the data sheet. The Lattice design tools automatically compensate for this delay and the simulation of the output clock will reflect the correct timing.

Case 3. External Clock External Feedback

When the PLL uses external feedback and the PLL drives the clock signal off chip, the input clock to external feedback delta (t_ϕ) specification defines the delay between the input clock and the feedback. This delay is not reflected in the timing simulation. The timing tool always assumes local feedback and it simulates a clock delay of $t_{CLK\_OUT\_DLY}$. To compensate for any delay in the feedback, the input clock must be advanced by the same amount as the delay in the feedback plus the inherent delay of the input clock and feedback pins (t_ϕ) (Equation 6).

$$t_{ADV\_INPUT} = t_{FBK\_DLY} + t_\phi + t_{CLK\_OUT\_DLY} \quad (6)$$

Where t_{ADV_INPUT} is the amount to advance the input clock and t_{FBK_DLY} is the amount of delay in the feedback line. The $t_{CLK\_OUT\_DLY}$ parameter should only be used when the feedback is generated by the CLK_OUT pin.

Case 4. Internal Clock Net External Feedback

When the PLL provides the clock for internal registers and uses external feedback, the Lattice design tools do not adjust the simulation models to account for the delay in the feedback. To compensate for any delay in the feedback, the input clock must be advanced by the same amount as the delay in the feedback (See Equation 6).

Case 5. Secondary Clock Timing Internal Feedback

When the PLL drives internal resisters via the secondary clock divider and uses internal feedback, the Lattice design tools adjust the delay associated with the clock net through the use of the internal adder $t_{PLL\_SEC\_DELAY.}$

Thus the design tools provide the correct timing simulations for the registers connected to the corresponding clock net.

Case 6. Secondary Clock Timing External Feedback

When the PLL drives internal resisters via the secondary clock divider and uses external feedback, the Lattice design tools do not adjust the simulation models to account the delay in the feedback. To compensate for any delay in the feedback, the input clock must be advanced by the same amount as the delay in the feedback (See Equation 6).

Appendix A. PLL Attributes

Attribute Name	Value	Default	Description	Components Applied		
				SPLL	STDPLL	STDPLLX
IN_FREQ	Real	None	Sets input clock frequency[1]	X	X	X
MULT	Integer	2	N divider setting: 1 to 32		X	X
DIV	Integer	2	M divider setting: 1 to 32		X	X
POST	Integer	1	V divider setting: 1,2,4,8,16, 32		X	X
SECDIV	Integer	None	K divider setting: 2,4,8,16,32			X
PLL_DLY	Integer	0	Delay Factor: -7,-6,..0..6,7		X	X
CLK_OUT_TO_PIN	ON, OFF	OFF	Sets PLL output clock to CLK_OUT pin	X	X	X
WAKE_ON_LOCK	ON, OFF	OFF	Determines if the device will wait for the PLL to lock before beginning the wake-up process	X	X	X
PLL_FBK_ATTRIBUTE[2]	CLKTEE, ROUTE	CLKTREE	Add additional delay to the feedback			X

1. Down to 4-bit resolution after decimal point in MHz.
2. For ispXPGA only.

M, N and V Setting Limitations

All combinations of M, N and V values are allowed as long as the frequency is within the specified range. **Exception**: the combination of M=1, N=1 and V=1 is not a valid combination for ispXPGA, ispGDX2 and ispXPLD. The V divider must be set to produce the highest possible f_{VDIVIN} for optimum PLL performance.

Appendix B. PLL Frequency Limit Equations

The divider values are specified as M, N, V, and K, which correspond to the DIV, MULT, POST, and SECDIV settings, respectively.

These values for f_{IN}, f_{OUT}, and f_{VDIVIN} are the absolute frequency ranges for the sysCLOCK PLL. The values for f_{INMIN}, f_{INMAX}, f_{OUTMIN}, and f_{OUTMAX} are the calculated frequency ranges based on the divider settings. These calculated frequency ranges become the limits for the specific divider settings used in the design. An error will be generated if f_{IN} or f_{OUT} violate these calculated frequency ranges.

Equations for Generating Input and Output Frequency Ranges

ispMACH 5000VG

	Min. (MHz)	Max. (MHz)
f_{IN}	5	180
f_{OUT}	5	180
$f_{VDIVIDIN}$	60	200

$f_{INMIN} = (f_{VDIVIDINMIN} /(V*N))*M$, if below 5*M round up to 5*M

$f_{INMAX} = (f_{VDIVIDINMAX} /(V*N))*M$, if above 180 round down to 180

$f_{OUTMIN} = f_{INMIN}*(N/M)$, if below 5*N round up to 5*N

$f_{OUTMAX} = f_{INMAX}*(N/M)$, if above 180 round down to 180

$f_{VDIVIN} = f_{OUT} * V$

If $f_{INMIN} > f_{INMAX}$, the divider settings are invalid. If f_{INMIN} is above 180MHz or f_{INMAX} is below 5MHz, the divider values are invalid.

ispXPGA, ispGDX2, ispXPLD

	Min. (MHz)	Max. (MHz)
f_{IN}	10	320
f_{OUT}	10	320
$f_{VDIVIDIN}$	100	400

$f_{INMIN} = (f_{VDIVIDINMIN} /(V*N))*M$, if below 10*M round up to 10*M

$f_{INMAX} = (f_{VDIVIDINMAX} /(V*N))*M$, if above 320 round down to 320

$f_{OUTMIN} = f_{INMIN}*(N/M)$, if below 10*N round up to 10*N

$f_{OUTMAX} = f_{INMAX}*(N/M)$, if above 320 round down to 320

$f_{VDIVIN} = f_{OUT} * V$

If $f_{INMIN} > f_{INMAX}$, the divider settings are invalid. If f_{INMIN} is above 320MHz or f_{INMAX} is below 10MHz, the divider values are invalid.

Appendix C. PLL_LOCK Behavior

The PLL_LOCK signal in the ispMACH™ 5000VG does not indicate an "out-of-lock" condition immediately after the PLL looses lock given certain conditions. This also implies that the PLL_LOCK signal does not indicate when a clock cycle is missing from the input clock under these conditions. Table 3 describes the behavior of the PLL_LOCK signal in the ispMACH 5000VG devices.

Table 3. ispMACH 5000VG PLL_LOCK Behavior

CLK_IN	PLL_RST	Previous State of PLL_LOCK	Next State of PLL_LOCK
Normal Operation	0	0	1 after t_{LOCK}
Normal Operation	0	1	1
Normal Operation	1	0	0
Normal Operation	1	1	0 after 5µs
Stuck High	0	0	0
Stuck High	0	1	0 after 5µs
Stuck High	1	0	0
Stuck High	1	1	0 after 5µs
Stuck Low	0	0	0
Stuck Low	0	1	1
Stuck Low	1	0	0
Stuck Low	1	1	0 after 5µs

Appendix D. Source Code Examples Generated by Module/IP Manager

STDPLLX Module (Verilog)

Header File for Verilog

```
module STDPLLX(CLK_IN, PLL_FBK, PLL_RST, PLL_LOCK, SEC_OUT, CLK_OUT);

parameter in_freq = "1";
parameter mult = "1";
parameter div = "1";
parameter post = "1";
parameter pll_dly = "1";
parameter secdiv = "1";
parameter clk_out_to_pin = "ON";
parameter wake_on_lock = "OFF";

input CLK_IN;
input PLL_FBK;
input PLL_RST;
output CLK_OUT;
output PLL_LOCK;
output SEC_OUT;

endmodule

//This design can be synthesized by Synplify and LeonardoSpectrum.
//It contains attributes for both synthesis tools.

module xt(clk_in, pll_fbk, pll_rst, clk_out, sec_out, pll_lock);


input clk_in;
input pll_fbk;
input pll_rst;
output clk_out;
output pll_lock;
output sec_out;


defparam    I1.in_freq = "100.0000",
                  I1.mult = "8",
                  I1.div = "5",
                  I1.post = "2",
                  I1.pll_dly = "0",
                  I1.secdiv = "2",
                  I1.clk_out_to_pin = "OFF",
                  I1.wake_on_lock = "OFF";


STDPLLX I1 (.CLK_IN(clk_in),  .PLL_FBK(pll_fbk), .PLL_RST(pll_rst),
.CLK_OUT(clk_out), .PLL_LOCK(pll_lock), .SEC_OUT(sec_out));
```

```
// exemplar attribute I1 in_freq 100.0000
// exemplar attribute I1 mult 8
// exemplar attribute I1 div 5
// exemplar attribute I1 post 2
// exemplar attribute I1 pll_dly 0
// exemplar attribute I1 secdiv 2
// exemplar attribute I1 clk_out_to_pin OFF
// exemplar attribute I1 wake_on_lock OFF


endmodule
```

STDPLLX Module (VHDL)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
library lattice;
use lattice.components.all;

entity txx is
port        (clk_in : in std_logic;
                  pll_fbk           : in std_logic;
                  pll_rst           : in std_logic;
                  clk_out           : out std_logic;
                  sec_out           : out std_logic;
                  pll_lock          : out std_logic);
end txx;

architecture behave of txx is
component STDPLLX

generic(    in_freq                 :     string;
                  clk_out_to_pin    :     string;
                  wake_on_lock      :     string;
                  mult              :     string;
                  div               :     string;
                  post              :     string;
                  pll_dly           :     string;
                  secdiv            :     string);

port(             CLK_IN:           in    std_logic;
                  PLL_FBK:          in    std_logic;
                  PLL_RST:          in    std_logic;
                  CLK_OUT:          out   std_logic;
                  PLL_LOCK:         out   std_logic;
                  SEC_OUT:          out   std_logic);

end component;
```

```
attribute in_freq                   : string;
attribute mult                      : string;
attribute div                       : string;
attribute post                      : string;
attribute pll_dly                   : string;
attribute secdiv                    : string;
attribute clk_out_to_pin            : string;
attribute wake_on_lock              : string;
attribute in_freq of I1             : label is "100.0000";
attribute mult of I1                : label is "8";
attribute div of I1                 : label is "5";
attribute post of I1                : label is "2";
attribute pll_dly of I1             : label is "0";
attribute secdiv of I1              : label is "2";
attribute clk_out_to_pin of I1      : label is "OFF";
attribute wake_on_lock of I1        : label is "OFF";

begin
I1: STDPLLX
generic map(
                  in_freq        => "100.0000",
                  mult           => "8",
                  div            => "5",
                  post           => "2",
                  pll_dly        => "0",
                  secdiv         => "2",
                  clk_out_to_pin => "OFF",
                  wake_on_lock   => "OFF")
port map(         CLK_IN         => clk_in,
                  PLL_FBK        => pll_fbk,
                  PLL_RST        => pll_rst,
                  CLK_OUT        => clk_out,
                  PLL_LOCK       => pll_lock,
                  SEC_OUT        => sec_out);

end behave;
```

ABEL

Library Instantiation

```
library 'lattice';
```

Simple PLL Declaration

```
LAT_SPLL(clk_in,in_freq,clock_out_to_pin,wake_on_lock);
```

Standard PLL Declaration

```
LAT_STDPLL(clk_in,in_freq,clk_out_to_pin,wake_on_lock,mult,div,post,pll_dly);
```

Extended PLL Declaration

```
LAT_STDPLLX(clk_in,in_freq,clk_out_to_pin,wake_on_lock,secdiv,mult,div,post,pll_dly);
```

Simple PLL Instantiation

```
pll_name SPLL(clk_in,clk_out);
```

Standard PLL Instantiation

```
pll_name STDPLL(clk_in,pll_lock,clk_out);
```

Extended PLL Instantiation

```
pll_name STDPLLX(clk_in,pll_fbk,pll_rst,pll_lock,clk_out,sec_out);
```

Appendix E. A Complete Project Example with Test Bench for ModelSim in VHDL

Top Module

```
--************************************
--* VHDL source constraint example
--* Extended PLL configuration
--* Lattice Semiconductor Corporation
--************************************
-- The following steps are required to use PLL functions in VHDL.
-- Step 1. Lattice library declaration
-- Step 2. PLL component declaration with generics (for simulation and Synplify –
-- synthesis)
-- Step 3. Parameter passing through attributes for the fitter (required by Exemplar)
-- Step 4. PLL hardcore instantiation
-- Step 5. Use of PLL outputs


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

--Step 1: Library declaration
library lattice;
use lattice.components.all;

entity extendedpll is
port (clk   : in std_logic;
      rst   : in std_logic;
      clken : in std_logic;
      pllfbk: in std_logic;  -- PLL extended feed back input
      pllrst: in std_logic;  -- PLL extended reset input
      qa    : out std_logic_vector(3 downto 0);
      qb    : out std_logic_vector(3 downto 0);
      qc    : out std_logic_vector(3 downto 0);
      qd    : out std_logic_vector(3 downto 0));
end extendedpll;

architecture behave of extendedpll is

--Step 2: PLL component declaration.
--            STDPLLX is a hard-coded PLL component.
component stdpllx
          generic( in_freq : string;
                   mult    : string;
                   div     : string;
                   post    : string;
                   pll_dly : string;
                   secdiv  : string);
          port  ( clk_in   : in  std_logic;
                  pll_fbk  : in std_logic;
                  PLL_RST  : in std_logic;
                  pll_lock : out std_logic;
```

```
                sec_out  : out std_logic;
                clk_out  : out std_logic);
end component;


-- Step 3: PLL parameter declaration
--             In STDPLLX, the following parameters are used.

attribute in_freq                     : string;
attribute mult                        : string;
attribute div                         : string;
attribute post                        : string;
attribute pll_dly                     : string;
attribute secdiv                      : string;
attribute clk_out_to_pin              : string;
attribute pll_fbk_attribute           : string;
attribute in_freq of i1               : label is "100.0000";
attribute mult    of i1               : label is "8";
attribute div     of i1               : label is "5";
attribute post    of i1               : label is "2";
attribute pll_dly of i1               : label is "3";
attribute secdiv  of i1               : label is "2";
attribute clk_out_to_pin  of i1       : label is "OFF";
--attribute pll_fbk_attribute of i1    : label is "CLKTREE";
attribute pll_fbk_attribute of i1     : label is "ROUTE";

signal counta                 : std_logic_vector(3 downto 0);
signal countb                 : std_logic_vector(3 downto 0);
signal ppclk                  : std_logic;-- primary PLL clock out
signal lock                   : std_logic;-- PLL lock out
signal spclk                  : std_logic;-- secondary PLL clock out
signal scken                  : std_logic;

signal dummy                  : std_logic;

begin

-- Step 4: PLL instantiation
I1: STDPLLX
generic map  (    in_freq  => "100.0000",
                  mult     => "8",
                  div      => "5",
                  post     => "2",
                  pll_dly  => "3",
                  secdiv   => "2")
port map     (    clk_in   => clk,
                  pll_fbk  => dummy,   -- PLL extended feedback
--                pll_fbk  => pllfbk,  -- PLL extended feedback
                  pll_rst  => pllrst,  -- PLL reset
                  pll_lock => lock,
                  clk_out  => ppclk,
                  sec_out  => spclk);
```

```
process(ppclk)
begin
    if (rst = '0') then
        scken   <= '0';
    elsif (ppclk'event and ppclk = '1') then
            scken <= clken and lock;
    end if;
end process;


process(ppclk, scken, rst)
begin
    if (rst = '0') then
            counta <= "0000";
-- Step 5: Use of PLL primary output clock
    elsif (ppclk'event and ppclk = '1') then
            if scken = '1' then-- clock enable
                counta <= counta + "0001" ;
            else
                counta <= counta;
            end if;
    end if;
end process;

process(spclk, rst)
begin
    if (rst = '0') then
            countb <= "0000";
-- Step 5: Use of PLL secondary output clock
    elsif (spclk'event and spclk = '1') then
            countb <= countb + "0001" ;
    end if;
end process;

qa <= counta;
qb <= countb;

end behave;
```

Test Bench File

```
library ieee;
use ieee.std_logic_1164.all;

entity tb_stdpllx is
end tb_stdpllx;

architecture simulate of tb_stdpllx is

component extendedpll
            port (
                  clk     : in std_logic;
                  rst     : in std_logic;
                  clken   : in std_logic;
```

```
                  pllfbk  : in std_logic;
                  pllrst  : in std_logic;
                  qa      : out std_logic_vector(3 downto 0);
                  qb      : out std_logic_vector(3 downto 0));
end component;

signal inclk, rst, pllrst, pllfbk, clken: std_logic := '0';
signal qa_out : std_logic_vector(3 downto 0);
signal qb_out : std_logic_vector(3 downto 0);

begin

UUT : extendedpll port map (clk=>inclk, rst=>rst, clken=>clken, pllfbk =>pllfbk, pll-
rst=>pllrst,qa=>qa_out,qb=>qb_out);

            inclk <= not inclk after 40 ns;

process
begin
        rst <= '0';
        pllrst <= '1';
        clken <= '0';
        wait for 200 ns;
        rst <= '1';
        pllrst <= '0';
        clken <= '1';
        wait for 10000 ns;
end process;

end simulate;

configuration cfg_tb of tb_stdpllx is
            for simulate
            end for;
end cfg_tb;
```

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-408-826-6002 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com