Stratix II PLL and Clock WYSIWYG Description

Version 2.5
May 30, 2005

by
Altera Corporation

Author:	Altera Corporation
System:	
Subsystem:	
Keywords:	Stratix II, PLL, ClockLock, Clock Synthesis, Clock Buffer, Clkbuf

Table of Contents:

1 Introduction

This document describes the WYSIWYG primitive for the Stratix II PLL and the WYSIWYG primitive for the Stratix II Clock Buffer.

The Stratix II PLL is similar to the Stratix PLL and is capable of implementing most Stratix features – see later for a description of the differences between the Stratix PLL and Stratix II PLL. See *Stratix_pll_wys_doc.doc* for a description of the Stratix PLL WYSIWYG.

The Stratix II Clock Buffer is new in Stratix II and allows the user to dynamically enable/disable the clock network, and to dynamically switch between multiple sources to drive the clock network.

2 Stratix II PLL

2.1 PLL Primitive

stratixii_pll <*pll_name*>
(
.inclk[1:0](<*input clock 1..0 source*>**),**
.clkswitch (<*signal to initiate clock switch*>**),**
.ena(<*pll enable source*>**),**
.areset(<*signal to initiate PLL resynchronization*>**),**
.pfdena(<*signal to disconnect the PFD, but keep the VCO running***),**
.fbin(<*external feedback source*>**),**
.scanclk(<*real-time programming clock input*>**),**
.scanread(<*real-time programming read control*>**),**
.scanwrite(<*real-time programming write control*>**),**
.scandata(<*real-time programming data*>**),**

.testin[3:0](<*test only input bus*>**),**

.scandataout(<*real-time programming data output*>**),**
.scandone(<*real-time programming reconfig done output*>**),**
.clk[5:0](<*global locked clock 5..0 output*>**),**
.clkbad[1:0](<*clk1, clk0 toggle indicator*>**),**
.activeclock(<*active clock indicator*>**),**
.clkloss(<*clock switchover indicator***>),**
.locked(<*lock status output*>**),**

.sclkout[1:0](<*LVDS* fast *SCLK outputs***>),**
.enable0(<*LVDS enable pulse 0 output***>),**
.enable1(<*LVDS enable pulse 1 output*>**),**

.testupout(<*Test output*>**),**

```
        .testdownout(<Test output>)
);

//
// Generic parameters, used with both standard and advanced parameters
//

defparam <pll_name>.operation_mode = <operation mode>;
defparam <pll_name>.pll_type = <PLL type>
defparam <pll_name>.compensate_clock = <compensate clock>;
defparam <pll_name>.feedback_source=<output clock used for feedback>;

// The following setting can only be set in test mode (used for internal testing only):
defparam <pll_name>.test_input_comp_delay_chain_bits = <input compensation delay chain
bit setting>;
defparam <pll_name>.test_feedback_comp_delay_chain_bits = <feedback compensation
delay chain bit setting>;

// For the following parameter, n can have values {1,0} specifying separate parameters
// for each clk input:
defparam <pll_name>.inclk<n>_input_frequency = <input period of inclk[n]>;

// Switch-over parameters
defparam <pll_name>.switch_over_type = <switchover type selector>;
defparam <pll_name>.switch_over_on_lossclk = <switchover scheme selector>;
defparam <pll_name>.switch_over_on_gated_lock = <switchover scheme selector>;
defparam <pll_name>.enable_switch_over_counter = <switch over counter selector>;
defparam <pll_name>.switch_over_counter = <value for switch_over_counter>;

// Self-reset parameters
defparam <pll_name>.self_reset_on_gated_loss_lock = <self-reset scheme selector>;

// Lock-detection parameters
defparam <pll_name>.gate_lock_signal = <gated lock signal indicator>;
defparam <pll_name>.gate_lock_counter = <value for gate lock_counter>;
defparam <pll_name>.valid_lock_multiplier = <number of half-clock cycles for lock high
multiplier>;
defparam <pll_name>.invalid_lock_multiplier = <number of half-clock cycles for lock low
multiplier >;
defparam <pll_name>.qualify_conf_done = <CONF_DONE qualifier>;


//
// Standard parameters (i.e. non-advanced, external parameters)
//

// For the following clk parameters, n can have values {5,4,3,2,1,0} specifying a separate set
// of parameters for each clk output:
defparam <pll_name>.clk<n>_output_frequency = <output frequency for clk<n>>;
defparam <pll_name>.clk<n>_multiply_by = <multiplication factor for clk<n>>;
defparam <pll_name>.clk<n>_divide_by = <division factor for clk<n>>;
```

defparam <*pll_name*>**.clk<n>_phase_shift** = <*phase shift of clk<n> output*>**;**
defparam <*pll_name*>**.clk<n>_duty_cycle** = <*duty cycle of clk<n>*>**;**
defparam <*pll_name*>**.clk<n>_use_even_counter_mode** = <*even counter mode for clk<n>*>**;**
defparam <*pll_name*>**.clk<n>_use_even_counter_value** = <*even counter value for clk<n>*>**;**

// Bandwidth control
defparam <*pll_name*>**.bandwidth** = <*bandwidth value*>**;**
defparam <*pll_name*>**.bandwidth_type** = <*bandwidth type*>**;**

// Spread Spectrum control
defparam <*pll_name*>**.spread_frequency** = <*Spread Spectrum Modulation Frequency* >**;**
defparam <*pll_name*>**.down_spread** = <*down spread percentage*>*;*

//
// Advanced Parameters (i.e. internal parameters):
//

defparam <*pll_name*>**.m_initial** = <*intial value for M counter*>**;**
defparam <*pll_name*>**.m** = <*modulus for M counter*>**;**
defparam <*pll_name*>**.n** = <*modulus for N counter*>**;**
defparam <*pll_name*>**.vco_post_scale** = <*modulus for VCO post scale counter*>**;**

// Spread Spectrum control
defparam <*pll_name*>**.m2** = <*Spread Spectrum modulus for M counter*>**;**
defparam <*pll_name*>**.n2** = <*Spread Spectrum modulus for N counter*>**;**
defparam <*pll_name*>**.ss** = < *modulus for Spread Spectrum counter*>**;**

// For the following five settings, <counter> can have values {c0, c1, c2, c3, c4, c5}:
defparam <*pll_name*>**.<counter>_high** = <*high period count for <counter>* >**;**
defparam <*pll_name*>**.<counter>_low** = <*low period count for <counter>* >**;**
defparam <*pll_name*>**.<counter>_initial** = <*initial value for <counter>* >**;**
defparam <*pll_name*>**.<counter>_mode** = <*mode for <counter>* >**;**

// For the following setting, <counter> can have values {c1, c2, c3, c4, c5}:
defparam <*pll_name*>**.<counter>_use_casc_in** = <*use cascade input for <counter>*>**;**

// For the following setting, <counter> can have values {c0, c1, c2, c3, c4, c5, m}:
defparam <*pll_name*>**.<counter >_ph** = <*VCO phase tap for <counter>* >

// The following setting can only be set in test mode (used for internal testing only):
defparam <*pll_name*>**.<counter>_test_source** = <*test source for <counter>* >**;**

// For the following setting, <n> can have values {5,4,3,2,1,0}:
defparam <*pll_name*>**.clk<n>_counter** = <*counter for <clkn>* >

// Bandwidth control
defparam <*pll_name*>**.charge_pump_current** = <*charge pump charge current*>
defparam <*pll_name*>**.loop_filter_r** = <*loop filter resistor value*>
defparam <*pll_name*>**.loop_filter_c** = <*loop filter capacitance value*>

```
//
// Simulation only parameters:
//

defparam <pll_name>.vco_min = <VCO min frequency>;
defparam <pll_name>.vco_max = <VCO max frequency>;
defparam <pll_name>.vco_center = <VCO center frequency>;
defparam <pll_name>.pfd_min = <PFD min frequency>;
defparam <pll_name>.pfd_max = <PFD max frequency>;

defparam <pll_name>.lock_high = <number of half-clock cycles for lock high>;
defparam <pll_name>.lock_low = <number of half-clock cycles for lock low>;


//
// LVDS mode specific parameters
//

// For the following parameters, <n> can have values {1,0}:
defparam <pll_name>.enable<n>_counter = <counter for enable<n> output>
defparam <pll_name>.sclkout<n>_phase_shift = <phase shift of sclkout<n> output>;

// VCO frequency control
defparam <pll_name>.vco_multiply_by = <multiplication factor for VCO frequency>;
defparam <pll_name>.vco_divide_by = <division factor for VCO frequency>;
```

2.2 Input Signals

The Enhanced PLL has up to TBD inputs from core and up to TBD inputs from pins.

.inclk[1..0](<*input clock source*>**),** are the clock inputs that are driving this PLL. By default, the PLL will use inclk[0]. The PLL can automatically switch from inclk[0] to inclk[1] if automatic switchover is enabled (i.e. switch_over_on_lossclk is on, or switch_over_on_gated_lock is on, and switch_over_type is auto), and the user can toggle between the inputs using the **clkswitch** port. Alternatively, the user can do manual switchover by dynamically selecting which clock input to use (either inclk[0] or inclk[1]) using the **clkswitch** signal. The inclk[0] signal must be connected, while the others should only be connected if switching is desired.

.clkswitch(<*signal to initiate clock switch*>**),** is an input from the core that allows toggling between inclk[0] and inclk[1] when performing automatic switchover, or allows manually selecting the input to use when performing manual switchover. For automatic switchover, on a rising edge the input switches from the currently active clock to the other clock (i.e. it toggles between the two clocks). For manual switchover, this signal controls a MUX that selects exactly which clock should be used – a value of '0' selects inclk[0] and a value of '1' selects inclk[1].

.fbin(<*external feedback source*>**),** is the external feedback pin for the PLL. It can only be used in the External Feedback mode. It must come from one of the output clocks of the PLL that is driving a pin, and the **feedback_source** parameter must be set to the corresponding output clock.

Connecting this input will restrict the PLL to be placed only as an Enhanced PLL – see later for details.

.ena(<*pll enable source*>**),** is the PLL enable pin. When the pin is high, the PLL drives out. When it is low, the PLL does not drive out a signal, and the PLL will go out of lock. It essentially acts as a combined enable/reset. When the pin is reasserted, the PLL will then have to re-lock. There is only one enable pin for all PLLs on the part. Therefore, each PLL must have the **ena** signal connected to the same pin, or disconnected. It is legal to have some PLLs using the pin and some not connecting it.

.areset(<*signal to initiate PLL resynchronization*>**),** is an input from core which initiates resynchronization of the PLL. If asserted (value of high), this resets all counters to their initial values, but most likely does not reset the gate lock counter. Users are advised to reset the PLL after they reprogram it dynamically in case they want the relative phase between the outputs to be guaranteed. This signal is optional.

.pfdena(<*signal to disconnect the PFD, but keep the VCO running)***),** is a signal which disables the PFD so the PLL will keep running regardless of the input clock. A high value enables the PFD (i.e. normal operation), and a low value disables the PFD (i.e. PFD keeps running regardless of clock input). The output clock frequency will not change for some time, so the user can use this feature to perform shutdown or cleanup functions when a reliable input clock is no longer available. This signal is not required and defaults to high (i.e. enabled).

.scanclk(<*real-time programming clock input*>**),** is the clock signal for the serial scan chain loading interface. This signal is not required.

.scanread(<*real-time programming read control*>**),** is the read control signal that determines whether the real-time programming scan chain should read in the input from the scandata port. This signal is not required.

.scanwrite(<*real-time programming write control*>**),** is the write control signal that determines whether the real-time programming scan chain should be written into the PLL. This signal is not required.

.scandata(<*real-time programming data*>**),** is the data for the serial scan chain loading interface. This signal is not required.

2.3 Output Signals

The PLL can have up to 6 generic clock outputs, and TBD outputs to core.

.clk[5:0](<*global clock 5..0 output*>**),** are the clock outputs of the PLL that can feed the core, the clock output pins, or the clock networks. *Using more than 4 clock outputs will restrict the PLL to be placed only as an Enhanced PLL – see later for details.*

.clkbad[1:0](<*clk1, clk0 toggle indicator*>**),** are signals which go high when the respective input clock stops to toggle (i.e. the clock signal is lost). *Using this output will restrict the PLL to be placed only as an Enhanced PLL – see later for details.*

.active_clock(<*active clock indicator*>**),** is a signal which indicates which input clock is being used currently. Low means inclk[0] and high means inclk[1]. *Using this output will restrict the PLL to be placed only as an Enhanced PLL – see later for details.*

.clkloss(<*clock switchover indicator*>**),** is a signal indicating that the clock switchover circuit has initiated a switchover. This could happen either if the primary reference clock went bad or if the user initiated the switch via the **clkswitch** signal. *Using this output will restrict the PLL to be placed only as an Enhanced PLL – see later for details.*

.locked(<*lock status output*>**),** gives the status of the PLL. When the PLL is locked, this signal is high. When the PLL falls out of lock, this signal is low. *This signal is always active high (unlike Stratix where this was active-low for Fast PLLs).*

.scandataout(<*real-time programming data output*>**),** is the output data for the serial scan chain interface. The output of the last register in the scan-chain is output on this port.

.scandone(<*real-time programming reconfig done output*>**),** is the output signal that can be used to determine when reconfiguration is complete. The signal goes high when the scan-chain write is initiated, and goes low when the PLL completes reconfiguration.

2.4 LVDS Only Ports

.enable0(<*LVDS enable pulse 0 output*>**), .enable1(**<*LVDS enable pulse 1 output*>**),** is the enable pulse output when the PLL is in LVDS mode. It is used in conjunction with the sclkout[0] or sclkout[1] output port, which provides the serial clock corresponding to the enable pulse. This signal feeds a DPA/SERDES block. This port can only be connected if the PLL is in LVDS mode.

.sclkout[1:0](<*LVDS fast SCLK output*>**),** is the fast serial clock taken from the VCO tap selection point for the specifiy counter. It is used in conjunction with the enable0 or enable1 output port, which provides the corresponding load enable pulse. It uses the same counter as specified for the corresponding enable0 or enable1 output port. This signal must feed a DPA/SERDES block. This port can only be connected if the PLL is in LVDS mode.

2.5 Test Ports

The following ports are test ports that can only be used in test mode (requiring a special ini), and are to be used mainly by PE for test purposes (i.e. these parameters should NOT be exposed to the user). No simulation support will be provided for these ports.

.testin(<*Test only input*>**)** corresponds to the EXTCLKEN[3:0] (i.e. TESTSIG[3:0]) on the Fast PLL (i.e. LVDS PLL) for use as test-only inputs. These ports should not be connected for Enhanced PLLs (i.e. GPP PLL).

.testupout(<*Test output*>**)** corresponds to the UP signal from the PLL to the core.

.testdownout(<*Test output*>**)** corresponds to the DOWN signal from the PLL to the core.

In test mode, the following test only parameters are also enabled:

1. test_input_comp_delay
2. test_feedback_comp_delay
3. <counter>_test_source

Refer to details later in the document for more details on these parameters.

2.6 Parameters

2.6.1 Overview

There are two independent ways of defining the initial condition of the PLL - using standard WYSIWYG parameters and using advanced WYSIWYG parameters. Those two ways are mutually exclusive and the advanced WYSIWYG parameters override the standard WYSIWYG parameters. However, the generic parameters (non-clock or loop-specific parameters) can be specified in either case. See section 2.6.6 for a detailed description of all WYSIWYG parameters, including their dependencies and priorities. In addition the PLLs are re-programmable during operation via the scan-chain interface (see section 2.6.5).

2.6.2 Clock settings

The Stratix II PLL will not support PLL synthesis from clock settings.

2.6.3 WYSIWYG parameters

Input frequencies for the input clocks and division, multiplication, phase shift per output clock are needed. If the output frequency is entered, then the division and multiplication should not be entered.

2.6.4 Advanced WYSIWYG parameters

The user enters directly the internal parameters of the PLL as described in the data book.

2.6.5 Real-time programming

For real-time programming the user needs to send the values of the internal PLL parameters to the PLL via the scan-chain interface – a serial data interface. The values of all of the output counter moduli and the M and N loop counter moduli as well as the phase tap associated with those counters can be changed. Additionally, the Charge Pump and Loop Filter parameters can be changed. The initial values of the counters cannot be changed and neither can the lock window settings, the compensation method or the lock counter parameters.

2.6.6 Description of all WYSIWYG parameters

<*pll_name*> is the unique identifier for this PLL. *This field is required*

2.6.6.1 Generic Parameters

These parameters can be used with both the standard and advanced parameters.

<*operation mode*> is one of {*normal, source_synchronous, zero_delay_buffer, external_feedback, no_compensation*}. In *normal* mode, the PLL will compensate for the entire delay of the clock path from the input pin to the destination clock network used by the clock

output specified in parameter **compensate_clock**. Therefore, if the PLL feeds an external clock output pin, a phase error will be observed on the pin due to the delay introduced by it. If multiple destination registers exist, then the compensation will be set for the worst-case clock path delay. In *source_synchronous* mode, the PLL will match the delay of the clock path to the delay of the data path, such that the data and clock will have the same alignment at the register as they did at the chip inputs. Only input IO registers fed by the PLL will be compensated for - registers in the core will be ignored. If multiple input IO registers exist in the compensated clock fanout, then the average data path delay will be match with the PLL's clock path delay. In *zero_delay_buffer* mode, the PLL must feed an external clock output pin, and will compensate for the delay introduced by that pin. The signal observed on the pin will be synchronized to the input clock. If the PLL also drives the internal clock network, there will be a phase difference on that network. When in *external_feedback* mode, the **fbin** port must be connected to an input pin, and there must be a board-level connection between this input pin and an external clock output pin, specified with the parameter **feedback_source**. Those two pins must have the same IO standard. The **fbin** port is aligned with the input clock. In *no_compensation* mode the PLL will not be set to align any clock to the input, but that should lead to better jitter performance. This field is not required and defaults to *normal*. *Using zero_delay_buffer or external_feedback modes will restrict this PLL to be placed only as an Enhanced PLL.*

<*PLL type*>, is the type of the PLL, must be one of *Enhanced, Fast, AUTO.* If not specified, the compiler will select the type of PLL automatically. The Fast type PLL is the same as the LVDS PLL, however the behavior is slightly different when the Fast PLL is used in LVDS mode (have LVDS only outputs and parameters). *Setting the type to anything other than AUTO will restrict the PLL to be placed only as the specified type.*

<*compensate clock*> is one of clk{0,1,2,3,4,5} or GCLK, LCLK, LVDSCLK, (for *NORMAL* mode) and specifies the output clock which should be compensated for. This clock could be made to have no offset with respect to the reference clock, and this relationship will be preserved closely even upon temperature and frequency changes. If not set, this parameter will default to clk0.

<*output clock used for feedback*> is one of clk{0..5}. This field is required when in feedback_mode and specifies which clock output will have a board level connection to **fbin**. *Specifying this parameter will restrict the PLL to be placed only as an Enhanced PLL – see later for details.*

<*input period for inclk{0,1}* > specifies the frequency of that input clock as period in picoseconds. The input frequency for inclk[0] must be specified. If the frequency for the other clock is not specified, Quartus will assume that it is the same as inclk[0]. If a different frequency is specified for the other clock, then Quartus will try to compute parameters, so that all of them are in lock range. In case this is not possible, this will cause a no fit with an error message suggesting use of dynamic reprogramming. Note that although the parameter is named inclk<n>_frequency, the value passed in is actually the clock period.

Test only generic parameters:

<*input compensation delay chain bit setting*>, <*feedback compensation delay chain bit setting*> can only be used in test mode (i.e. requiring a special ini), and specifies the bit setting of the input compensation delay chain, or the feedback compensation delay chain for the PLL. By default, the delay chain is set automatically by Quartus, but can be overridden using this parameter. This

parameter is NOT specified as a delay value, but instead is specified as the corresponding bit value (i.e. CRAM bit setting represented as a decimal number) for the particular delay as defined in the NPP (CRAMs CR_CLKCOMPIN and CR_FBCOMPIN).

Switch-over Parameters:

<*switchover type selector*>, is ***auto*** or ***manual***, and specifies what type of switchover will be used. This parameter is ignored if there is only one input clock connected. A value of *auto* will implement automatic switchover and the clkswitch signal will behave as a toggle between the two clocks. A value of *manual* will implement manual switchover and the clkswitch signal will behave as a MUX select signal and will directly specify which input signal is currently active. *Specifying a value of auto will restrict the PLL to be placed only as an Enhanced PLL – see later for details.*

<*switchover scheme selector*>, *is* ***on*** *or* ***off***, selecting if this particular condition should initiate a clock switchover. Defaults to off. *Specifying this parameter will restrict the PLL to be placed only as an Enhanced PLL – see later for details.*

<*switchover counter selector*>, *is* ***on*** *or* ***off***, selecting if the switchover should be asynchronous or synchronous. Defaults to on, which selects switchover at the falling edge of the Nth clock edge, where N is the value of the **switch_over_counter** parameter. *Specifying this parameter will restrict the PLL to be placed only as an Enhanced PLL – see later for details.*

<*value for switchover_counter*> is the value for the 5-bit counter (1-32) that specifies how many clock cycles after a switchover condition is detected the actual switchover is implemented. This field is required if any switchover scheme is used. *Specifying this parameter will restrict the PLL to be placed only as an Enhanced PLL – see later for details.*

Self-reset Parameters:

<*self-reset scheme selector*>, *is* ***on*** *or* ***off***, selecting if this particular condition should initiate a self-reset. Defaults to off.

Lock-detection Parameters:

<*gated lock signal indicator*>**,** indicates if the locked signal should be internally gated with a 20-bit programmable counter so it does not oscillate during initial power-up. Valid values are **yes** and **no**. Once the counter finishes counting however, this signal tracks the output of the lock detect circuit precisely. The counter is enabled when the PLL is enabled (i.e. **ena** signal is asserted), and asserting the **areset** pin will reset the counter.

<*value for gated lock counter*> is the value for the 20-bit counter (0-1048575) that gates the locked output before sending it to the **locked** signal. This field is required if the **gate_lock_signal** parameter is set to on.

<*number of half-clock cycles for lock high multiplier*> specifies the scaling factor (in half-clock cycles) for the number of cycles that the output clocks must be locked for before the lock pin goes high. Valid settings are {*1,5*}. The number of half-clock cycles for locked high will then be {1 |

5} *N, where N is the modulus of the N counter. It will be usually calculated automatically by the compiler, based on the other requested parameters. *This parameter is required if the **.locked** port is connected. It is only used by the compiler, and is ignored in the 3rd party simulation models.*

<*number of half-clock cycles for lock low multiplier*> specifies the scaling factor (in half-clock cycles) for the number of cycles that the output clocks must be locked for before the lock pin goes low. Valid settings are {*1,5*}. The number of half-clock cycles for locked low will then be {1 | 5} *N, where N is the modulus of the N counter. It will be usually calculated automatically by the compiler, based on the other requested parameters. *This parameter is required if the **.locked** port is connected. It is only used by the compiler, and is ignored in the 3rd party simulation models.*

<*CONF_DONE qualifier*>, is one of ON or OFF. If this parameter is set to ON, the PLL will hold the CONF_DONE signal of the chip low until the PLL locks. This means the chip will not begin operating until the PLL has locked. Defaults to OFF.

2.6.6.2 Standard Parameters

These parameters specify the counter and loop settings using high-level concepts, as opposed to the lower-level settings specified by the advanced parameters. These are also referred to as "external" parameters. These parameters cannot be used if using advanced parameters.

<*output frequency for clk{0,1,2,3,4,5}* > specifies the desired frequency for the given clock output. If this parameter is specified, the compiler determines the appropriate input clock division and multiplication to achieve this frequency. This parameter is not required, and is ignored if the corresponding clk[] is not used. This parameter cannot be used if the multiplication or division factors are also specified. This parameter defaults to 0, which indicates that it is not used.

<*multiplication factor for clk{0,1,2,3,4,5}* > specifies the integer multiplication factor for **.clk{0,1,2,3,4,5}** with respect to the input clock frequency. The value must be > 0. If **.clk{0,1,2,3,4,5}** is not used, this parameter is ignored. This parameter cannot be used if the output frequency for the clock is also specified.

<*division factor for clk{0,1,2,3,4,5}* > specifies the integer division factor for **.clk{0,1,2,3,4,5}** with respect to the input clock frequency. The value must be > 0. If **.clk{0,1,2,3,4,5}** is not used, this parameter is ignored. This parameter cannot be used if the output frequency for the clock is also specified.

<*phase shift for <counter>*>. Specifies the delay of the output clock relative to the input clock, expressed as a time unit or a phase with respect to the input clock. If no units are specified, they are assumed to be picoseconds. Phase shifts of k/8, (k=0..7) times the VCO period (0, 45, 90, 135, 180, 225, 270 or 315 degrees) will be implemented precisely. This translates to resolution of 45/post-divider's value, for a phase with respect to the output clock, so it is always better than 45. For other phase shifts, the compiler will choose the closest achievable value. The allowable range for the phase shift is between 0ps and the input clock period. A positive phase shift represents a forward shift in time; that is, the output of the PLL will lag the input clock. This field is optional and defaults to 0.

<*duty cycle for clk{0,1,2,3,4,5}* specifies the duty cycle for this output clock by giving the percentage of high time, e.g. 60. Default value is 50.

<*even counter mode for clk{0,1,2,3,4,5}* specifies whether the given clock output should be forced to be implemented using even counter mode (instead of odd mode). This parameter is currently hidden to the user (i.e. undocumented). Default value is 'off'.

<*even counter value for clk{0,1,2,3,4,5}* specifies whether the given clock output should be forced to be implemented using even counter values (instead of odd values). This parameter is currently hidden to the user (i.e. undocumented). Default value is 'off'.

Bandwidth Control:

<*bandwidth type*>, is the desired bandwidth type for this PLL. Valid values are ***Low, High, Medium, Auto, and "Custom".*** In the case when **Custom** is specified then the **bandwidth** parameter must be given specifying the desired value in frequency units. Allowable range is pending characterization.

<*bandwidth value*>, is the desired bandwidth for the PLL. The units are in MHz. If this is not specified, then the bandwidth will be chosen by the compiler as required to satisfy the other PLL settings**.**

Spread Spectrum Control:

<*Spread Spectrum Modulation Frequency* >**;** is the desired modulation frequency for spread spectrum in time units. *Specifying this parameter will restrict the PLL to be placed only as an Enhanced PLL – see later for details.*

<*down spread percentage*>, is the desired down spectrum percentage*;* Valid range is 0.4-0.6% and is pending Si characterization. *Specifying this parameter will restrict the PLL to be placed only as an Enhanced PLL – see later for details.*

2.6.6.3 Advanced Parameters

These parameters provide direct access to the numerous internal PLL parameters. If any of these parameters are specified they must all be used, and the standard parameters will be ignored. Error checking will consist only of verifying that the settings are legal.

<*initial value for M counter*>*;* Valid range is 1-256
<*modulus for M counter*>*;* Valid range is 1-256
<*modulus for N counter*>**;** Valid range is 1-256

<*modulus for VCO post scale counter*>**;** Valid range is 1-2. It specifies whether VCO post divider should be activated in order to implement lower VCO frequencies than are normally legal without the divider. *Specifying this parameter to a value other than 1 will restrict the PLL to be placed only as a Fast PLL – see later for details.*

Spread Spectrum Control:
<*Spread Spectrum modulus for M counter*>**;** Valid range is 1-256
<*Spread Spectrum modulus for N counter*>**;** Valid range is 1-256
<*modulus for Spread Spectrum counter*>**;** Valid range is 1-32768 (TBD)

The counter parameters below are ignored or illegal if the mode of the counter is bypass.

For the following settings, <counter> could be c0, c1, c2, c3, c4, or c5
<*mode for <counter>*> is one of *bypass, odd, even* specifying odd or even division or no division at all.
<*high period count for <counter> counter* >**;** Valid range is 1-256
<*low period count for <counter> counter* >**;** Valid range is 1-256
<*initial value for <counter> counter* >**;** Valid range is 1-256

Test only advanced parameter:
<*test source for <counter>* > can only be used in test mode (i.e. requiring a special ini), and specifies the counter's source. A legal value is one of {0, 1, 2, 3, 4} and represents a source of TCLK0 to TCLK4, respectively. However, not all values are legal for all counters – the NPP specifies which values are legal, and what each of these paths represent. Additionally, simulation will NOT be supported for the TCLK4 path (i.e. spread spectrum counter), but will be supported for the other paths. If this parameter is not specified, then by default the counter's source is either the VCO or the previous counter if the use of cascaded counters is enabled.

For the following setting, <counter> could be c1, c2, c3, c4, or c5
<*use cascade input for <counter> counter*>**;** Specifies source driving the counter - valid values are *on* or *off*, where *on* indicates the cascade input (i.e. output from the previous counter) drives this counter, and *off* indicates the VCO drives this counter. The default is *off*. *Specifying this parameter will restrict the PLL to be placed only as an Enhanced PLL – see later for details.*

For the following setting, <counter> could be c0, c1, c2, c3, c4, or c5 *and* m.
<*phase tap for <counter> counter;* Valid values are integers 0,1,2,3,4,5,6,7

The following select which outputs use which counter for the clock signals and the clock enable signals.
<*counter for clock<n>*>**; n=0,1,2,3,4,5;** Valid settings are c0, c1, c2, c3, c4, c5.

Bandwidth Control (Loop filter parameters):

<*charge pump charge current***>** is the value of the charge pump current. If given, it should be between 2 and 205 uA (TBD). Only units of uA are supported and if no units are given then they are assumed uA.

<*loop filter resistor value***>** is the value of the loop resistor. If given, it should be between 1 and 20 kOhm (TBD), but not all values in the range are achievable. Only units of kOhm are supported and if no units are given, the units are assumed to be kOhm.

<*loop filter capacitance value***>** is the value of the loop capacitor. If given, it should be in the range of 5-20 pF (TBD), but not all values in the range are achievable. Only units of pF are supported and if no units are used, pF are assumed.

2.6.6.4 Simulation Parameters

The following parameters are used only by the simulation models and will be ignored by the compiler.

<*number of half-clock cycles for lock high*> specifies the number of half-clock cycles that the output clocks must be locked for before the lock pin goes high. *This parameter is only used for the 3rd party simulation models. It is ignored during compilation.*

<*number of half-clock cycles for lock low*> specifies the number of half-clock cycles that the output clocks must be out of lock for before the lock pin goes low. *This parameter is only used for the 3rd party simulation models. It is ignored during compilation.*

<VCO min frequency>;<VCO max frequency>;<VCO center frequency>;
<PFD min frequency>;<PFD max frequency>; These parameters specify the limits of the PFD and VCO hardware implementation. They are used by third party simulation models to verify that the input frequency is in the lock range.

For a description of the simulation default values, refer to a later section.

2.6.6.5 LVDS Mode Specific Parameters

These parameters are only valid when the PLL type is LVDS.

<*counter for enable*<*n*> *output*>**; n=0,1;** Valid settings are c0, c1. The counter used must be the same as the counter used for the corresponding clock output port (i.e. enable0 is same as clk[0], enable1 is same as clk[1]).

<*phase shift of sclkout*<*n*> *output*>**; n=0,1;** Specifies in picoseconds the phase shift of the given sclkout output. The sclkout[] output can only use the VCO phase taps to implement phase, so the maximum legal phase value is 7/8th of one VCO period. The VCO phase tap is shared with the corresponding clk[] output, and so both must have the same "fine grain" phase (i.e. phase amount that is less than one VCO period). In LVDS mode, this parameter defaults to a phase of 0.

<*multiplication factor for VCO frequency*>**;** <*division factor for VCO frequency*>**;** Specifies the multiplication and division factors to achieve the desired VCO frequency. Since the frequency of the sclkout outputs are the same as the VCO frequency, these parameters allow setting the frequency of sclkout outputs. By default, this parameter defaults to 0, which means it will be ignored by the compiler, and instead the compiler will try and achieve the best VCO frequency for the given set of parameters. In LVDS mode, this parameter needs to be specified in order to guarantee the desired sclkout frequency is implemented.

2.6.7 Simulation Default Parameter Values

This section describes the simulation default values for the various parameters. These defaults are for simulation purposes only and are not guaranteed to compile correctly as not all parameters are valid in all modes of operation.

```
operation_mode                  = "normal";
pll_type                        = "auto";
compensate_clock                = "clk0";
feedback_source                 = "clk0";
```

```
test_input_comp_delay_chain_bits      = 0;
test_feedback_comp_delay_chain_bits   = 0;

inclk0_input_frequency                = 10000;
inclk1_input_frequency                = 10000;

switch_over_type                      = "manual";
switch_over_on_lossclk                = "off";
switch_over_on_gated_lock             = "off";
enable_switch_over_counter            = "off";
switch_over_counter                   = 0;

self_reset_on_gated_loss_lock         = “off”

gate_lock_signal                      = "no";
gate_lock_counter                     = 1;

valid_lock_multiplier                 = 1;
invalid_lock_multiplier               = 5;

qualify_conf_done                     = "off";

clk0_output_frequency                 = 0;
clk0_multiply_by                      = 1;
clk0_divide_by                        = 1;
clk0_phase_shift                      = 0;
clk0_duty_cycle                       = 50;
clk0_use_even_counter_mode            = “off”;
clk0_use_even_counter_value           = “off”;

clk1_output_frequency                 = 0;
clk1_multiply_by                      = 1;
clk1_divide_by                        = 1;
clk1_phase_shift                      = 0;
clk1_duty_cycle                       = 50;
clk1_use_even_counter_mode            = “off”;
clk1_use_even_counter_value           = “off”;

clk2_output_frequency                 = 0;
clk2_multiply_by                      = 1;
clk2_divide_by                        = 1;
clk2_phase_shift                      = 0;
clk2_duty_cycle                       = 50;
clk2_use_even_counter_mode            = “off”;
clk2_use_even_counter_value           = “off”;

clk3_output_frequency                 = 0;
clk3_multiply_by                      = 1;
clk3_divide_by                        = 1;
clk3_phase_shift                      = 0;
```

```
clk3_duty_cycle                 = 50;
clk3_use_even_counter_mode      = “off”;
clk3_use_even_counter_value     = “off”;

clk4_output_frequency           = 0;
clk4_multiply_by                = 1;
clk4_divide_by                  = 1;
clk4_phase_shift                = 0;
clk4_duty_cycle                 = 50;
clk4_use_even_counter_mode      = “off”;
clk4_use_even_counter_value     = “off”;

clk5_output_frequency           = 0;
clk5_multiply_by                = 1;
clk5_divide_by                  = 1;
clk5_phase_shift                = 0;
clk5_duty_cycle                 = 50;
clk5_use_even_counter_mode      = “off”;
clk5_use_even_counter_value     = “off”;

bandwidth                       = 0;
bandwidth_type                  = "auto";
spread_frequency                = 0;
down_spread                     = "0.0";

common_rx_tx                    = "on";
rx_outclock_resource            = "auto";

m                               = 1;
m_initial                       = 1;
m_ph                            = 0;
n                               = 1;
m2                              = 1;
n2                              = 1;
ss                              = 0;
vco_post_scale                  = 1;

c0_high                         = 1;
c0_low                          = 1;
c0_initial                      = 1;
c0_mode                         = "bypass";
c0_ph                           = 0;

c1_high                         = 1;
c1_low                          = 1;
c1_initial                      = 1;
c1_mode                         = "bypass";
c1_ph                           = 0;

c2_high                         = 1;
c2_low                          = 1;
```

```
c2_initial                      = 1;
c2_mode                         = "bypass";
c2_ph                           = 0;

c3_high                         = 1;
c3_low                          = 1;
c3_initial                      = 1;
c3_mode                         = "bypass";
c3_ph                           = 0;

c4_high                         = 1;
c4_low                          = 1;
c4_initial                      = 1;
c4_mode                         = "bypass";
c4_ph                           = 0;

c5_high                         = 1;
c5_low                          = 1;
c5_initial                      = 1;
c5_mode                         = "bypass";
c5_ph                           = 0;

clk0_counter                    = "c0";
clk1_counter                    = "c1";
clk2_counter                    = "c2";
clk3_counter                    = "c3";
clk4_counter                    = "c4";
clk5_counter                    = "c5";

c1_use_casc_in                  = "off";
c2_use_casc_in                  = "off";
c3_use_casc_in                  = "off";
c4_use_casc_in                  = "off";
c5_use_casc_in                  = "off";

m_test_source                   = 5;
c0_test_source                  = 5;
c1_test_source                  = 5;
c2_test_source                  = 5;
c3_test_source                  = 5;
c4_test_source                  = 5;
c5_test_source                  = 5;

enable0_counter                 = "c0";
enable1_counter                 = "c1";
sclkout0_phase_shift            = 0;
sclkout1_phase_shift            = 0;

vco_multiply_by                 = 0;
vco_divide_by                   = 0;
vco_post_scale                  = 1;
```

charge_pump_current = 0;
loop_filter_r = "1.0";
loop_filter_c = 1;

pfd_min = 0;
pfd_max = 0;
vco_min = 0;
vco_max = 0;
vco_center = 0;

pll_compensation_delay = 0;

2.7 Input Cell Polarities and Default Values

All signals are active high. All core inputs have programmable inverts, while the clock inputs and the feedback input do not have programmable inverts.

Port	Polarity	Programmable Invert
inclk[1:0]	Active High	No
clkswitch	Active High	Yes
ena	Active High	Yes
areset	Active High	Yes
pfdena	Active High	Yes
fbin	Active High	No
scanclk	Active High	Yes
scanread	Active High	Yes
scanwrite	Active High	Yes
scandata	Active High	Yes

2.8 Legality checks

If you use even one advanced mode parameter then you need as much of them as needed to fully specify the PLL and you cannot use the standard parameters. For example, if you use **m** then you have to use **n** as well, and you cannot use **divide_by**.

If scandata, scanread, or scanwrite is used, then scanclk must be used as well.

If using all clock outputs, then two of the outputs should be of limited fanout (as they can only go to half of the chip).

All of the PLLs in a design can use at most one PLL enable pin, but they are not required to use it if another PLL uses it.

There are various legal ranges for all of the parameters. For Enhanced PLLs M, N, C counters are 8 bit counters, so valid ranges are 1-256. For Fast PLL, valid output counter values are 1-16, M can be 1-32, and N can be 1-4.

Hardware allows fast or enhanced type PLLs to be driven from anything, but we don't allow it for jitter considerations. PLLs can only be driven by other PLLs or clock pins.

If a duty cycle of higher than 60/40 is used on the feedback path, then the lock sense circuit will malfunction and software will give warning.

The LVDS PLL is the only one that can compensate for the LVDSCLK network.

2.9 Differences Between PLL Types

Depending on the type of the PLL, its supported features are different. This section describes all these differences, and which features will restrict a PLL from being used generically as any type of PLL.

The following table summarizes the differences between the PLL types from a feature point of view. Those features not listed can be assumed to be supported by all PLL types.

Feature	Enhanced PLL	Fast PLL
Switchover:		
- Automatic	yes	no
- Manual	yes	yes
Output Counters	6	4
Compensation Modes:		
- Normal	yes	Yes
- External	yes	No
- Zero Delay Buffer	yes	No
- No Compensation	yes	Yes
Spread Spectrum	yes	No
LVDS / DPA	no	Yes
VCO Post Divider	no	Yes

The following table summarizes the ports that are different between the different types of PLLs. The ports not listed can be assumed to be supported by all PLL types.

Feature	Port	Enhanced PLL	Fast PLL
Switchover	clkswitch	yes (select or toggle)	yes (select only)
	clkbad[1:0]	yes	no
	activeclock	yes	no
	clkloss	yes	no
External Feedback Compensation	Fbin	yes	no
Output Counters	Clk[5:0]	Has all 6 ports	Only has clk[3:0]

LVDS / DPA	enable0	no	yes
	enable1	no	yes
	sclkout[1:0]	no	yes

The following table summarizes the parameters that are different between the different types of PLLs. The parameters not listed can be assumed to be supported by all PLL types.

Feature	Parameter	Enhanced PLL	Fast PLL
Switchover	switch_over_type	manual and auto	only manual
	switch_over_on_lossclk	yes	no
	switch_over_on_gated_lock	yes	no
	enable_switch_over_counter	yes	no
	switch_over_counter	yes	no
Compensation	feedback_source	yes	no
	compensate_clock	not for LVDS clock	not for extclk
Spread Spectrum	spread_frequency	yes	no
	down_spread	yes	no
	m2	yes	no
	n2	yes	no
	ss	yes	no
Output Counters	clk<n>_*	N = {0..5}	n ={0..3}
	<counter>_*	counter = {c0..c5}	counter = {c0..c3}
LVDS / DPA	enable<n>_counter	no	yes
	sckout<n>_phase_shift	no	yes
	vco_multiply/divide_by	yes (but not needed)	yes
VCO Post Divider	vco_post_scale	no	yes

Additionally, the legal ranges for counters are different between the types of PLLs: Enhanced PLLs have 8-bit output counters and 9-bit M/N counters, while Fast PLLs have 4-bit counters. Also, the charge-pump and loop-filter values are different between both types of PLLs.

2.10 Mapping to NPP Ports

This section describes the mapping to the ports as specified in the Stratix II NPP.

NPP PLL Type	NPP Input Port Name	WYSIWYG Port Name
LVDS PLL	CLKPIN[]	inclk[]
	CLKCOREIN[]	inclk[]
	RESET	ena
	NRESYNC	areset
	SCANDATA	scandata
	SCANCLK	scanclk
	SCANWR	scanwrite
	SCANRD	scanread

	PFDEN	pfdena
	CLKINMXCNTL	clkswitch
	EXTCLKEN[]	testin[]
GPP PLL	NRESYNC	areset
	SDATA	scandata
	SCANCLK	scanclk
	SCANWR	scanwrite
	SCANRD	scanread
	EXTSWTCH/NCLRLOCK	clkswitch
	PFDEN	pfdena
	CLKPIN[]	inclk[]
	CORECLK	inclk[]
	EXTCLKFB	fbin

NPP PLL Type	NPP Output Port Name	WYSIWYG Port Name
LVDS PLL	SCKOUT[]	sclkout[]
	LOADEN[]	enable1, enable0
	DIVCLKOUT[]	clk[]
	LOCKOUT	locked
	SCANOUT	scandataout
	SCANDONE	scandone
	UPTOUT	testupout
	DNTOUT	testdownout
GPP PLL	LOCKOUT	locked
	SDATAOUT	scandataout
	SCANDONE	scandone
	CLK0BAD	clkbad[0]
	CLK1BAD	clkbad[1]
	REFCLKSEL	activeclock
	CLKLOSS	clkloss
	UPTOUT	testupout
	DNTOUT	testdownout
	EXTOUT	clk[]

3 Stratix II Clock Buffer

The Stratix II Clock Buffer is a new WYSIWYG in Stratix II that represents the clock buffers that drive the Global Clock Network, the Local Clock Network, and the dedicated External Clock path. In Stratix, these buffers were not presented to the user since there was no functionality that the use could control. However, in Stratix II, there is a dynamic MUX and a clock enable that can be controlled by the user, thus requiring a new primitive for the user to specify this logic.

The Global Clock network allows a clock signal (or other global signal) to reach all parts of the chip with the same amount of skew. Similarly, the Local Clock network allows a signal to reach one quadrant of the chip (though half the chip can be reached by driving two quadrants). The

External Clock buffer represents the clock path from the outputs of the PLL to the dedicated clock output pins.

3.1 Clock Buffer Primitive

stratixii_clkbuf <*clkbuf_name*>
(
.inclk[3:0](<*input clock 3..0 source*>**),**
.clkselect[1:0](<*input clock select source*>**),**
.ena(<*clock network enable source*>**),**

.outclk (<*clock network output*>**)**
);

defparam <*clkbuf_name*>**.clock_type =** <*type of clock buffer*>**;**

3.2 Input Signals

.inclk[3..0](<*input clock source*>**),** are the clock inputs that drive the clock network. If more than one inclk[] signal is specified, then selection between them is done using the **clkselect[1:0]** signal. The inclk[0] signal must be connected, while the other clock inputs should only be connected if switching is desired. Clock pins, clock outputs from the PLL, as well as core signals can drive the inclk[] input. However, only certain combinations of signal sources are allowed, as described in a later section. If more than inclk[0] is connected, then only the Global Clock network can be driven by this clock buffer (i.e. the Local Clock network cannot be used).

.clkselect[1..0](<*input clock select source*>**),** is an input from the core that allows dynamically selecting which clock source drives the clock network driven by this clock buffer. The signal selects one of the four clock inputs where a value of '00' selects inclk[0], '01' selects inclk[1], '10' selects inclk[2], and '11' selects inclk[3]. This signal should only be connected if dynamically selecting between multiple clock inputs is desired, and defaults to GND if left unconnected. If this signal is connected, then only the Global Clock network can be driven by this clock buffer.

.ena(<*input clock enable source*>**),** is an input from the core that allows enabling or disabling the entire clock network driven by this clock buffer. If left unconnected, this signal defaults to VCC (i.e. enabled). However, the clock network will be disabled for clock buffers that are not used on the chip (i.e. when no signals are going through the clock buffer).

3.3 Output Signals

.outclk(<*clock network output*>**),** is the clock output that will drive the clock network associated with this clock buffer. It can drive either the Local Clock network or Global Clock network as determined by the compiler.

3.4 Parameters

<type of clock buffer> indicates what type of clock buffer should be used. Valid values are {AUTO, GCLK / "Global Clock", LCLK / "Regional Clock", EXTCLK / "External Clock Output", SIDE_CLK / "Dual-Regional Clock"}. A value of AUTO allows the compiler to pick the best clock buffer to use, while other values restrict usage to only the given clock buffer. The SIDE_CLK clock type refers to the use of two LCLK quadrants to reach half the chip. This parameter is not required and defaults to AUTO if not specified.

3.5 Input Cell Polarities and Default Values

All signals are active high. All core inputs have programmable inverts (TBD), while the clock inputs do not have programmable inverts.

Port	Polarity	Programmable Invert
inclk[3:0]	Active High	No
clkselect[1:0]	Active High	Yes
ena	Active High	Yes

3.6 Connectivity Restrictions

This section describes the restrictions associated with which signal sources can drive the inclk[] input.

- The clkselect[] port can only be used when the type is either AUTO or GCLK. Otherwise it is either not used or set to 0.
- The inclk[] ports used must be consistent with the clkselect[] ports used. For example, if the clkselect[] ports are set to 0, then inclk[0] must be used. If clkselect[0] is fed by a anything other than 0 or 1 and clkselect[1] is set to 1, then either inclk[0] or inclk[1] must be used.
- If the clkselect[] ports are set to anything other than 0, then only pins or PLL clock outputs may feed the inclk[] ports. In addition, pins must feed only inclk[0] or inclk[1], while PLL clock outputs must feed only inclk[2] or inclk[3].
- For certain illegal cases caught by the previous rule, such as a pin feeding inclk[3] while a PLL clock output feeds inclk[1], the compiler may choose (but not necessarily do so) to instead invert the clkselect[1] port and exchange the inclk[3] and inclk[1] inputs and give a message to the user.
- If the clock_type is EXTCLK, then only a PLL clock output may feed the inclk[0] port, while the outclk port must feed an I/O.
- If a CLKBUF feeds any inclk[] port of another CLKBUF, then both must be reducible to a single CLKBUF of equivalent functionality.
- If a pin or PLL clock output feeds multiple CLKBUF that have their clkselect[] set to anything other than 0, then it must feed inclk[] port that has the same index on all such CLKBUF. The compiler may choose (but not necessarily do so) to instead invert the appropriate clkselect[] port and exchange the inclk[] port inputs on selected CLKBUF to avoid the error.

The following table summarizes which ports are usable by the different clock buffer types. If a port is used that is not available for a particular clock type, then the clock buffer placement will be restricted to only the clock buffer types that support the port.

Port	GCLK	LCLK	EXTCLK
inclk[3:0]	Has all 4 ports	Only has inclk[0]	Only has inclk[0]
clkselect[1:0]	yes	no	no
ena	yes	yes	yes
outclk	yes	Yes	yes

As can be seen, all clock buffer types support the clock-enable feature, while only the GCLK buffer supports the dynamic MUX feature.

If the clock_type is specified as AUTO, then Quartus will pick the clock type to try and satisfy all these restrictions. The following lists special considerations when automatically selecting the clock type:

1. If the dynamic MUX is used and the clock buffer feeds a pin, then since the EXTCLK buffer cannot be used, a GCLK buffer will be used if possible to get to the clock pin. Similarly, if the clock buffer feeds a clock pin that cannot be reached by the PLL using the EXTCLK buffer (i.e. not a dedicated clock path), a GCLK will be used. In both cases, a warning message will be issued informing the user about this “irregular” path taken to the pin.

3.7 Mapping to NPP Ports

This section describes the mapping to the ports as specified in the Stratix II NPP.

NPP PLL Type	NPP Input Port Name	WYSIWYG Port Name
LVDS PLL	GCLKDRV[]	inclk[]
	LCLKDRV[]	inclk[]
	CLKPIN[]	inclk[]
	DIVCLKOUT[]	inclk[]
	GCLKEN[]	ena
	LCLKEN[]	ena
	CLKOMXCNTL[]	clkselect[]
GPP PLL	GCLKDRV[]	inclk[]
	LCLKDRV[]	inclk[]
	CLKPIN[]	inclk[]
	nCLKPIN[]	inclk[]
	(PLL counter output)	inclk[]
	GCLKEN[]	ena
	LCLKEN[]	ena
	EXTCLKEN[]	ena
	MUXCTRL[]	clkselect[]

4 Differences from Stratix

This section lists the differences between the Stratix PLL and the Stratix II PLL, as well as what features cannot be implemented in the Stratix II PLL, or need to be implemented differently by the user.

4.1 Stratix Features Different in Stratix II

This section lists features that were supported in Stratix but will be different or not supported in Stratix II, and so will require re-implementation of any Stratix design that was using the feature as described. This does not include features that can be supported using a different method (e.g. time_delay can be supported in Stratix II using phase_shift). For more detail on exactly which parameters and ports are affected, see the next section.

4.1.1 PLL reconfiguration

The dynamic PLL reconfiguration scheme has been re-implemented in Stratix II with a different protocol and a different set of parameters that can be updated. As a result, any Stratix designs that used PLL reconfiguration will need to be modified, and Quartus will give an error when mapping a Stratix PLL that used PLL reconfiguration to an Stratix II PLL. However, since a greater level of control is provided now, it should still be possible to re-implement any Stratix-based PLL reconfiguration designs to use Stratix II.

For Stratix, the user could dynamically reconfigure the counter high\low values and the counter delay element values. For Stratix II, delay element reprogramming is replaced with phase reprogramming, and in addition to the counter high\low values, control of the loop filter and charge-pump are possible. Also, since the counter sizes are different and different parameters are being controlled, the size of the scan-chain will be different from Stratix.

The PLL reconfiguration IP (i.e. the altpll_reconfig megafunction) will still maintain the same interface as used in Stratix. However, the parameter codes and parameter widths will be different since these are different in Stratix II.

4.1.2 Advanced parameters

Stratix II has different counter sizes, no delay elements, and a different set of loop-filter and charge-pump parameters compared to Stratix. As a result, it is not possible to support all the advanced parameter values that were legal in Stratix since some of them will not be legal in Stratix II. However, this is not expected to be a problem since most uses of the PLL does not involve specifying the advanced parameters, and so normal users will not be affected. Additionally, the unsupported values are not expected to be usable in a real application even in Stratix.

4.1.3 Total number of clock outputs

In Stratix, some of the enhanced PLLs had a total of 10 clock outputs: 6 outputs that fed the clock networks, and 4 outputs that fed clock pins. In Stratix II, there are a maximum of 6 clock outputs total. However, every output can drive the clock networks and the clock pins, and so is more flexible. This additional flexibility eliminates most of the need for the 10 clock outputs, but still means that it is technically possible for a Stratix design that used more than 6 clock outputs to not fit in a single Stratix II PLL.

There is a possibility that Quartus create two Stratix II PLLs if more than 6 clock outputs were used in a Stratix PLL, however this is currently considered low priority as it is expected to be rare. Additionally, the phase relationship between the clocks will be different, so the resulting circuit could still be different from Stratix.

4.1.4 Maximum multiplication factor

In Stratix, each counter is 9-bits wide, while in Stratix II they are 8-bits wide. This implies that Stratix II PLLs can have a maximum theoretical clock multiplication factor of 256 (i.e. 2^8) versus 512 in Stratix. However, based on the limitation of the VCO frequency range, and when considering real applications, this is expected to be rarely seen by the user.

Although the same restriction would normally apply to the maximum division factor, since output counters can be cascaded to make larger dividers, this is not expected to be an issue.

4.1.5 LVDS mode

In Stratix, one of the clk[] outputs also doubled as the sclkout output used to feed the LVDS receiver or transmitter SERDES blocks. However, for Stratix II, separate ports exist for sclkout[], and therefore the corresponding clk[] outputs do not drive the SERDES blocks and can instead be used to drive the core.

Additionally, in Stratix II actual parameters exist to specify the VCO frequency of the PLL (which is the resulting sclkout frequency). In Stratix, the compiler "hacked" this specification using the clk<n>_multiply_by parameter, which is not needed for Stratix II.

4.1.6 Source Synchronous compensation mode

New to Stratix II is the source_synchronous compensation mode. As described earlier, this is identical to the normal mode except for compensation performed. Stratix only had the normal compensation mode.

4.2 Stratix Parameters and Ports Different in Stratix II

This section lists parameters and ports that were in the Stratix WYSIWYG that are different or missing in the Stratix II WYSIWYG. However, most of these differences can be supported in Stratix II through a different method, unless otherwise indicated. Quartus will automatically map any Stratix WYSIWYGs to Stratix II WYSIWYGs when possible, and give errors when this is not possible.

4.2.1 Ports Different in Stratix II

.clkena[5:0](<*clock enable sources*>**), .extclkena[3:0](**<*extclock enable sources*>**)**, are not part of the Stratix II PLL WYSIWYG. Instead, they can be implemented using the new Stratix II clkbuf WYSIWYG. Quartus will automatically create the clkbuf WYSIWYG and map this port to it for any Stratix WYSIWYGs that use this port.

.extclk[3:0](<*external clock output*>**)**, is not part of the Stratix II PLL WYSIWYG. Instead, every regular **clk[5:0]** output can feed external clock pins, so this port is not needed. Quartus will automatically map any **extclk[3:0]** port on a Stratix WYSIWYG to a **clk[5:0]** port on the Stratix II WYSIWYG. However, since Stratix can have up to 10 different clocks outputs, it is possible that not all **extclk** output ports can be mapped to the 6 output clocks in Stratix II.

.scanaclr(<*scan asynchronous clear*>**)**, is not part of the Stratix II PLL WYSIWYG. Since the PLL reconfiguration protocol is different in Stratix II, this port cannot be mapped to Stratix II if used in a Stratix WYSIWYG. Similarly, the **scandata**, **scanclk**, and **scandataout** ports used on a Stratix WYSIWYG will not be mapped from Stratix to Stratix II. *Instead a user error will be given asking the user to re-implement the PLL reconfiguration portion of their design*

.clk[] in LVDS mode, may be different in the Stratix II PLL WYSIWYG since Stratix re-used one of the clk[] ports as the sclkout output, but Stratix II has an explicit port for the scklout signals.

4.2.2 Parameters Different in Stratix II

.operation_mode = <*operation mode*>, where the mode is *source_synchronous*, is new in Stratix II. This mode is not supported in Stratix.

.scan_chain = <*scan chain type*>, is not supported in Stratix II since there is only one type of Enhanced PLL. This parameter will be ignored if it is specified in a Stratix WYSIWYG since it has no effect on the Stratix II PLL. It was only necessary in Stratix to distinguish between the Enhanced GPP and the non-enhanced GPP, but since Stratix II only has one type of GPP, this parameter is not necessary.

.clk<n>_time_delay = <*time delay of clk*<*n*>>, is not supported in Stratix II since there are no delay elements. Instead, the user should use the **clk<n>_phase_shift** parameter to specify any delay. For Stratix WYSIWYGs that use this parameter, the delay requested will be added to the phase_shift parameter in the Stratix II WYSIWYG.

.extclk<n>_multiply_by, .extclk<n>_divide_by, .extclk<n>_phase_shift, .extclk<n>_time_delay, .extclk<n>_duty_cycle, are not supported in Stratix II since there are no extclk ports. Instead, the regular **clk[5:0]** ports can be driven to external clock pins. If used in a Stratix WYSIWYG, they will be mapped to **clk** parameters corresponding to the **clk** port mapping described earlier.

.primary_clock=<*primary clock*>, is not supported in Stratix II since the clock inputs are fully connected to the PLL clock inputs. As a result, any clock pin can be designated as the primary clock by simply connecting it to the **clk[0]** port. If used in a Stratix WYSIWYG, Quartus will make sure that the primary clock specified is connected to the **clk[0]** port in Stratix II if it is not already.

.feedback_source=<*output clock used for feedback*>, takes different parameter values in Stratix II since there are no **extclk** ports in Stratix II. If used in a Stratix WYSIWYG, it will be mapped to the appropriate **clk** port that the **extclk** port was mapped to.

.skip_vco, is not supported in Stratix II since it is not legal to bypass the VCO in Stratix II. *If used in a Stratix WYSIWYG, this will result in an error informing the user that this parameter cannot be converted to an equivalent Stratix II configuration.* Since this parameter is not available in the megawizard, most users will not be affected by this change. This parameter was initially added to Stratix to implement clock-switchover for Fast-PLLs – however, since Fast-PLLs now support a form of switchover, this is no longer needed.

Advanced Parameters:
The following are advanced parameters that will usually result in compilation errors if used in a Stratix WYSIWYG. However, depending on the difficulty to implement the mapping, they may be mapped to Stratix II.

.<counter>_high, .<counter>_low, .<counter>_initial, .<counter>_mode, .<counter >_ph, have different names for <counter>: instead of l0..l1, g0..g3, valid values will be c0..c5. Additionally, the legal range for the high/low/initial parameters is smaller for Stratix II (8-bits versus 9-bits). If used in a Stratix WYSIWYG, these parameters will be mapped such that g0..g3 maps to c0..c3, and l0..l1 maps to c4..c5 (or another available counter), for values that are legally possible in Stratix II.

.<counter >_time_delay = *<time_delay associated with <counter> counter>***,** is not supported in Stratix II since it does not have delay elements. *If used in a Stratix WYSIWYG, they will result in an error asking the user to instead use the phase-shift parameters.*

.extclk<n>_counter = *<counter for extclk<n>*, is not supported in Stratix II since it does not have a separate **extclk** port. If used in a Stratix WYSIWYG, it will be mapped to the equivalent **clk<n>_counter** parameter, and an available c0..c5 counter will be used.

.clk<n>_counter = *<counter for <clkn>*, has a different set of parameter values than Stratix II since the names for the counters are different. If used in a Stratix WYSIWYG, these parameters will be mapped to available Stratix II counters for values that are legally possible in Stratix II.

.charge_pump_current, .loop_filter_r, .loop_filter_c, have a different set of legal values in Stratix II compared to Stratix. If used in a Stratix WYSIWYG, these parameters will be mapped to the nearest legal value for an Stratix II WYSIWYG.

4.2.3 New Ports in Stratix II

The following is a list of ports that are new to Stratix II and do not exist in the Stratix PLL WYSIWYG:

- The entire **clkbuf** atom is new in Stratix II. It implements the **clkena** and **extclkena** ports that were formally on the Stratix PLL, in addition to new dynamic muxing.
- **scanread** is new in Stratix II
- **scanwrite** is new in Stratix II
- **scandone** is new in Stratix II
- **sclkout[]** is new in Stratix II

4.2.4 New Parameters in Stratix II

The following is a list of parameters that are new to Stratix II and do not exist in the Stratix PLL WYSIWYG:

- All parameters on the **clkbuf** WYSIWYG are new since it did not exist in Stratix
- **switch_over_type** is new in Stratix II
- **switch_over_on_gated_lock** is new in Stratix II
- **qualify_conf_done** is new in Stratix II
- **clk<n>_output_frequency** is new in Stratix II
- **<counter>_use_casc_in** is new in Stratix II
- **sclkout<n>_phase_shift** is new in Stratix II
- **vco_muliply_by, vco_divide_by** are new in Stratix II
- **vco_post_scale** is new in Stratix II