Implementing PLL Reconfiguration in Stratix & Stratix GX Devices

December 2005, ver. 2.0

Application Note 282

Introduction

Phase-locked loops (PLLs) use several divide counters and delay elements to perform frequency synthesis and phase shifts. In Stratix™ and Stratix GX enhanced PLLs, these counters and delay elements are configurable in real-time. Designers can vary output clock frequency and time delay in real time without reconfiguring the entire FPGA.

The PLL reconfiguration feature is useful in applications that might operate at different frequencies. It is also useful in prototyping environments where you can easily sweep PLL output frequencies and adjust clock delay on the fly. For instance, a system generating test patterns might generate and transmit patterns at either 50 or 100 MHz, depending on the unit under test. Real-time reconfiguration of PLL components allows you to switch between two such output frequencies within 20 µs (excluding the lock time for the PLL). You can also use this feature to adjust clock-to-out (t_{CO}) delays in real time by changing the output clock delay. This approach eliminates the need to regenerate a programming file with the new PLL settings.

This document discusses the implementation of real-time PLL reconfiguration in Stratix and Stratix GX enhanced PLLs, and explains how to select the PLL parameters. This application note also explains how to implement the PLL reconfiguration feature in the Quartus® II software.

PLL Reconfiguration Hardware Implementation

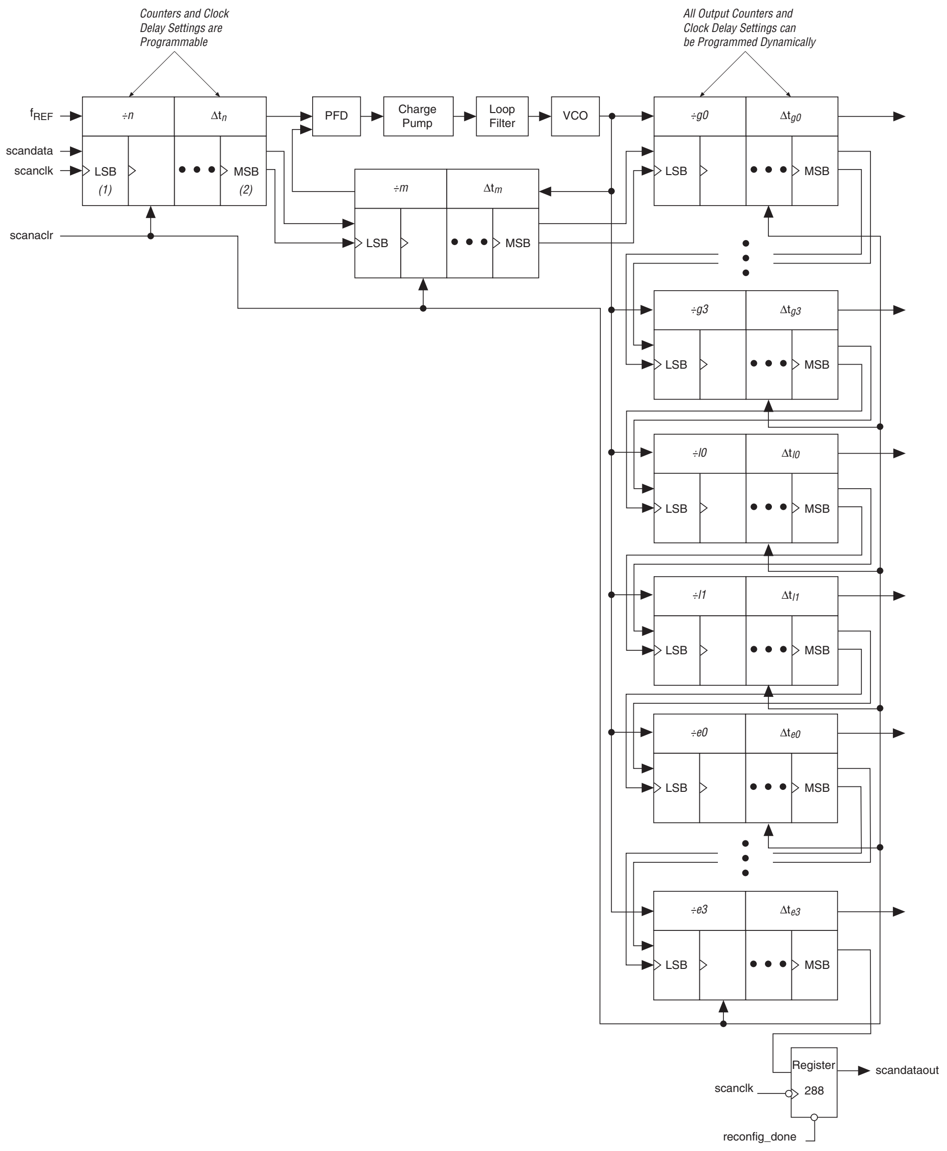
Enhanced PLLs in Stratix devices support real-time PLL reconfiguration. The following PLL components are configurable in real time:

- Pre-scale counter and delay element (n, Δt_n)
- Feedback counter and delay element (m, Δt_m)
- Post-scale output counters and delay elements (g, Δt_g, l, Δt_l, e, Δt_e)

You cannot dynamically adjust the charge pump current, loop filter components, and phase shifts using voltage-controlled oscillator (VCO) taps.

Figure 1 shows the reconfiguration circuit block diagram.

Figure 1. PLL Reconfiguration Circuit Block Diagram



Notes to Figure 1:

(1) LSB: least significant bit.

(2) MSB: most significant bit.

PLL counters and delay elements can be dynamically adjusted by shifting their new settings into a serial shift register chain or shift register, as shown in Figure 1. Serial data is input to the shift register via the `scandata` port and shift registers are clocked by `scanclk`. After the last bit of data is clocked, the PLL configuration bits are synchronously updated with the data in the scan registers. While the counter and delay element settings are updated synchronously, these settings are updated one counter at a time based on the individual counter clock frequency. The maximum input shift clock rate for `scanclk` equals 22 MHz. You can also asynchronously clear the registers in the shift register using `scanaclr`. The minimum pulse width for this clear signal is 5 ns. All of these signals can be driven by the FPGA logic array or I/O pins (see Table 1).

Table 1. Real-Time PLL Reconfiguration Ports

PLL Port Name	Description	Source	Destination
scandata	Serial input data stream	Logic array or I/O pins	PLL reconfiguration circuit
scanclk	Serial clock input signal	Logic array or I/O pins	PLL reconfiguration circuit
scanaclr	Active high, asynchronous clear signal for serial shift register chain	Logic array or I/O pins	PLL reconfiguration circuit
scandataout	Output of the last shift register in the shift register	PLL reconfiguration circuit	Logic array or I/O pins

All PLL counters have 20 configuration bits and all delay elements have 4 configuration bits. The counters are either pre-scale and feedback, or post-scale.

Pre-Scale & Feedback Counters (*n*, *m*)

The pre-scale counter, *n*, and feedback counter, *m*, can implement spread spectrum by switching between two different divide settings. These counters range from 1 to 511. Hence, the nominal count value and the spread spectrum count value need 9 configuration bits each, for a total of 18 configuration bits. Two additional configuration bits are used to bypass the nominal and spread counters (i.e., divide by 1). When using spread spectrum, you must set these two bypass bits to the same value for proper operation. This setting brings the total number of counter configuration bits to 20. When spread spectrum is not used, the device will use the nominal count value and ignore the spread spectrum count value and bypass bit.

Post-Scale Counters (*g*, *l*, *e*)

The *g*, *l*, and *e* post-scale counters do not implement spread spectrum, but implement programmable duty cycle. Each counter has a 9-bit high time setting and a 9-bit low time setting. The duty cycle is the ratio of output high or low time to the total cycle time, which is the sum of the two. Additionally, these counters have two control bits (`RBYPASS` and `RSELODD`) bringing the total number of configuration bits to 20.

When you set the `RBYPASS` bit to 1, it bypasses the counter, resulting in a divide by 1. When you set this bit to 0, the high and low time counters are added to compute the effect division of the VCO output frequency. For example, if the post-scale divide factor is 10, the high and low count values could be set to 5 and 5, respectively, to achieve a 50-50% duty cycle. On the other hand, a 4 and 6 setting for the high and low count values would produce an output clock with a 40-60% duty cycle.

The `RSELODD` bit indicates an odd divide factor for the VCO output frequency along with a 50% duty cycle. For example, if the post-scale divide factor was 3, you can set the high and low time count values to 2 and 1, respectively, to achieve this division. This would also create a 33 (low time) to 67% (high time) duty cycle. However, if the design requires a 50-50% duty cycle, you can set the `RSELODD` control bit to 1 to achieve this duty cycle in spite of an odd division factor. The PLL implements this duty cycle by transitioning the output clock from high to low on a falling edge of the VCO output clock.

Delay Elements

All configurable delay elements are identical and their settings are shown in Table 2. The delay elements can add delay in steps of 250 ps and a maximum delay of 3 ns. All counters inside the Stratix enhanced PLL have an associated delay element. For instance, the Δt_m delay element is associated with the feedback counter and can be used to advance all PLL clock outputs (in effect adding negative delay).

Table 2. Delay Element Settings

Delay Element Bit Settings				Delay (ns) (1)
3	2	1	0	
0	0	0	0	0.00
0	0	0	1	0.25
0	0	1	0	0.50
0	0	1	1	0.75
0	1	0	0	1.00
0	1	0	1	1.25
0	1	1	0	1.50
0	1	1	1	1.75
1	0	0	0	2.00
1	0	0	1	2.25
1	0	1	0	2.50
1	0	1	1	2.75
1	1	(2)	(2)	3.00

Notes to Table 2:

(1) Actual delays will vary depending on the speed grade of the device.

(2) The setting for this entry is don’t care.

Altera® recommends using phase shift instead of time delay unless you plan on reconfiguring the time delay using the PLL reconfiguration feature.

Scan Chain Order

The length of the shift register is different for different PLLs. Enhanced PLLs 5 and 6 have 12 counters and delay elements, and a 289-bit shift register. The last register in the shift register holds the transfer enable bit. Figure 2 shows the shift register order of PLL components in these PLLs. Enhanced PLLs 11 and 12 have only eight counters and delay elements, and a 193-bit shift register. Figure 3 shows the shift register order of components in these PLLs. Scan registers for the counter settings are marked *n*, *m*, *g*0, and *g*1, whereas scan registers for the delay elements are marked Δt_m, Δt_n, Δt_{g0}, and Δt_{g1}. For all registers in the chain, the MSB is shifted in first and the LSB last.

Figure 2. Shift Register Order for Enhanced PLLs 5 & 6

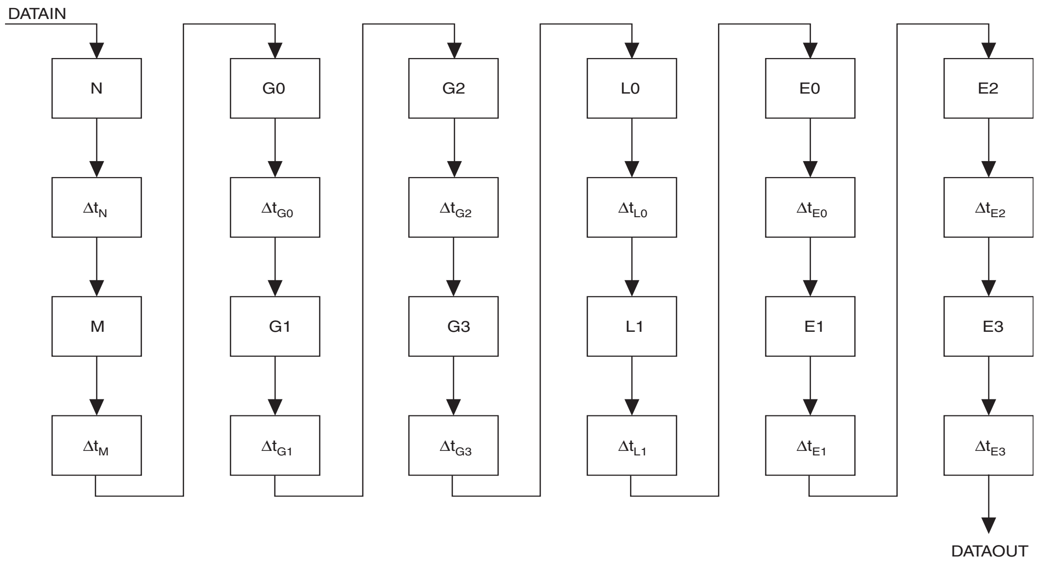
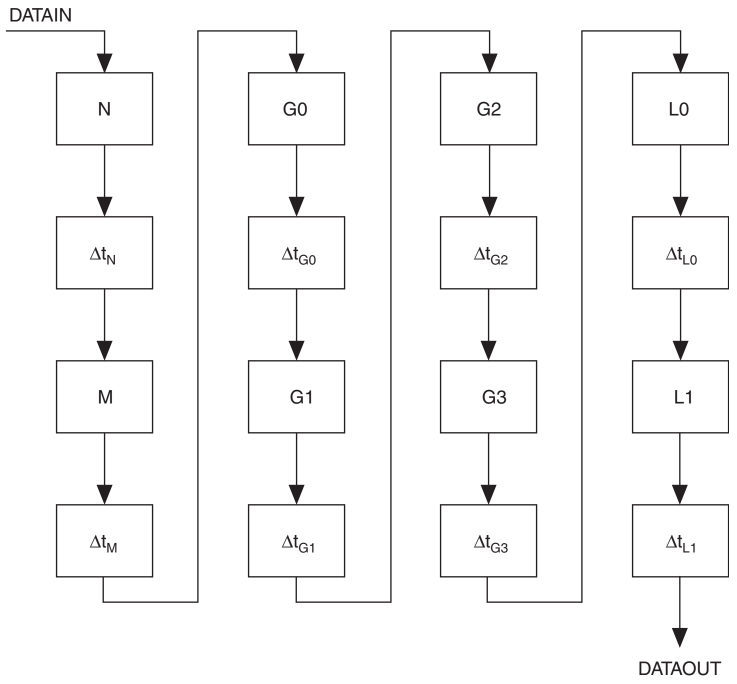


Figure 3. Shift Register Order for Enhanced PLLs 11 & 12



Bypassing PLL Counters

Bypassing a PLL counter results in a multiply (*m* counter) or a divide (*n*, *g*, *l*, or *e* counters) factor of 1.

The only way to bypass an *m* or *n* counter is to set the bypass bit for that counter to `1` and set the counter's LSB to `0`. For the *m* and *n* counters only, setting the bypass bit high and then setting the count value to `1` (or any number that ends with `1`) disables the counter, rendering the PLL inoperable. Table 3 shows the bit settings for the *m* and *n* counters.

Table 3. m & n Counter Settings

Bits [9..0] Settings *(2)*	Description
`1xxxxxxxx0`	PLL counter bypassed
`1xxxxxxxx1`	PLL counter disabled
`0xxxxxxxxx`	PLL counter not bypassed or disabled *(1)*
`0000000000`	PLL counter set to a count value of 511

Note to Table 3:
(1) The PLL counter is not bypassed or disabled because the bypass bit `[9]` is set to 0.
(2) The LSB for the counter is on the right hand side.

☞ The Quartus II software sets bits `[9..0]` for the m and n spread counter setting to `0000000000` if the spread spectrum feature is not used. If this feature is used, bits `[8..0]` show the corresponding spread count value. The spread count bypass setting, bit `[9]`, is always set to 0.

To set m or n to 1, the Quartus II software always selects the nominal count setting (`000000000`) and sets the bypass bit to 1. (The nominal count value above is shown with the LSB on the right side.)

To set any of the 10 output counters (*g*, *l*, or *e*) to bypass, set the bypass bit to `1`. The values on the other bits will be ignored.

Implementing Reconfigurable PLLs in the Quartus II Software

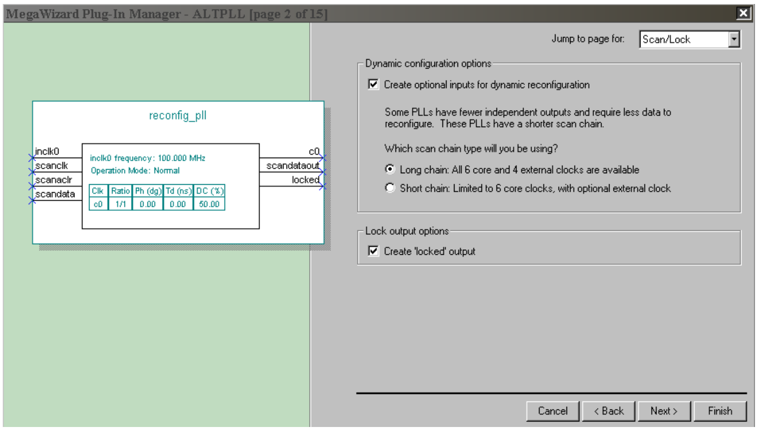
You can specify the PLL input/output frequencies and phase/time shifts in the Quartus II `altpll` MegaWizard® Plug-In Manager. Based on these parameters, the Quartus II software picks internal settings for the PLL. The SRAM Object File (**.sof**) stores these internal settings, and the PLL uses these settings after configuration and power-up. Use the `altpll_reconfig` megafunction in the Quartus II software to implement PLL reconfiguration.

altpll Megafunction

You can use the altpll MegaWizard Plug-In Manager to enable the PLL reconfiguration circuitry, as shown in Figure 4. On page 2 of the altpll MegaWizard Plug-In do the following:

- Select **Long chain: All 6 core and 4 external clocks are available** when using enhanced PLLs 5 and 6, since these PLLs have the four external clock output counters.
- Select **Short chain: Limited to 6 core clocks, with optional external clock** when using PLLs 11 and 12, since these PLLs do not have four external clock outputs. Enabling the reconfiguration circuitry adds the scanclk, scandata, and scanaclr and scandataout ports to the altpll megafunction.

Figure 4. Enabling PLL Reconfiguration in the Quartus II MegaWizard Plug-In Manager



inclk0 frequency: 100.000 MHz
Operation Mode: Normal

Clk	Ratio	Ph (dg)	Td (ns)	DC (%)
c0	1/1	0.00	0.00	50.00

Jump to page for: Scan/Lock

Dynamic configuration options

[✓] Create optional inputs for dynamic reconfiguration

Some PLLs have fewer independent outputs and require less data to reconfigure. These PLLs have a shorter scan chain.

Which scan chain type will you be using?

[✓ Long chain: All 6 core and 4 external clocks are available]
Short chain: Limited to 6 core clocks, with optional external clock

Lock output options

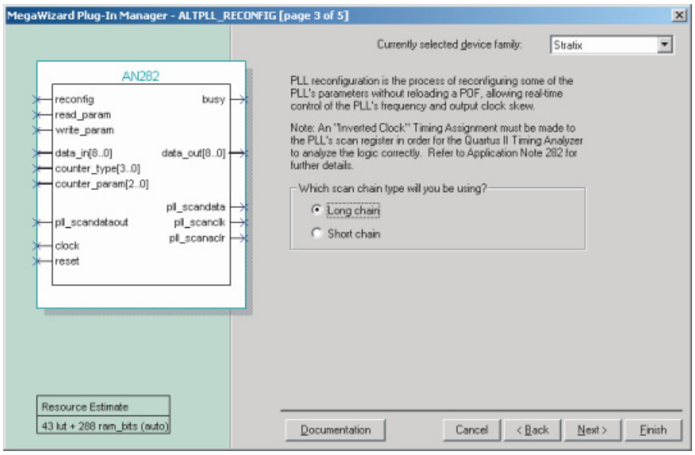
[✓] Create 'locked' output

altpll_reconfig Megafunction

The altpll_reconfig megafunction provides an easy and efficient way to reconfigure the Stratix or Stratix GX enhanced PLLs on the fly. You can use the altpll_reconfig MegaWizard Plug-In Manager to reconfigure the Stratix or Stratix GX enhanced PLLs.

Figures 5 and 6 show the MegaWizard interface for the altpll_reconfig megafunction. You can access this megafunction in the Quartus II software through **libraries\megafunctions\IO\altpll_reconfig**.

Figure 5. MegaWizard Plug-In Manager Interface 1


MegaWizard Plug-In Manager - ALTPLL_RECONFIG [page 3 of 5]

Currently selected device family: Stratix

PLL reconfiguration is the process of reconfiguring some of the PLL's parameters without reloading a POF, allowing real-time control of the PLL's frequency and output clock skew.

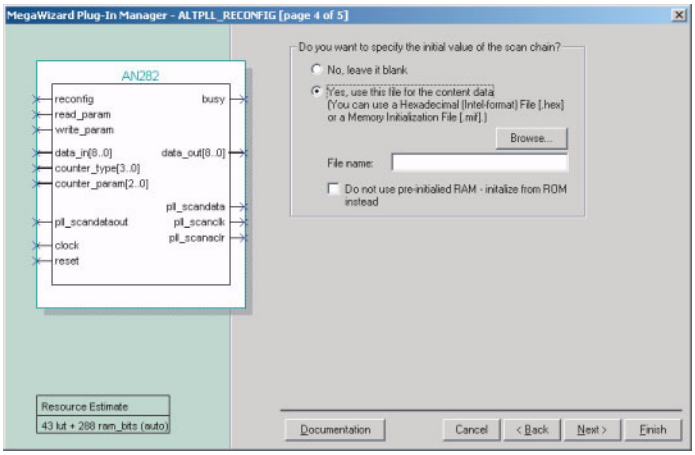
Note: An "Inverted Clock" Timing Assignment must be made to the PLL's scan register in order for the Quartus II Timing Analyzer to analyze the logic correctly. Refer to Application Note 282 for further details.

Which scan chain type will you be using?

[✓ Long chain] Short chain

Resource Estimate: 43 lut + 288 ram_bits (auto)

Figure 6. MegaWizard Plug-In Manager Interface 2


MegaWizard Plug-In Manager - ALTPLL_RECONFIG [page 4 of 5]

Do you want to specify the initial value of the scan chain?

No, leave it blank [✓ Yes, use this file for the content data]
(You can use a Hexadecimal (Intel-format) File [.hex] or a Memory Initialization File [.mif].)

File name:

☐ Do not use pre-initialied RAM - initalize from ROM instead

Resource Estimate: 43 lut + 288 ram_bits (auto)

You must specify the type of the scan chain: either a long chain (289 bits for PLLs 5 and 6) or a short chain (193 bits for PLLs 11 and 12). You can also specify a Memory Initialization File (**.mif**) commonly called MIF, or a hexadecimal (**.hex**) file with the new PLL settings and transfer it into the PLL scan registers. Alternatively, you can choose to leave the initial value of the scan chain as blank, which means that the device powers up with all PLL counters programmed with zero's.

☞ Hardcopy devices do not support pre-initialized RAM; you can choose to initialize from ROM instead. The **Do not use pre-initialized RAM - initialize from ROM instead** option on page 2 of the `altpll_reconfig` MegaWizard Plug-In Manager is intended specifically for Hardcopy devices. When the Hardcopy device powers up, the contents of the ROM are copied to a RAM which is then used to load the PLL counters.

Use the following steps to implement the `altpll_reconfig` megafunction.

1. Create an `altpll` megafunction with your choice of multiplication/division factors and specify real-time reconfiguration as being used.

2. Create an `altpll_reconfig` megafunction and specify the shift register size (long chain or short chain) to match the `altpll` megafunction created in the previous step.

3. Compile (or synthesize) the design in the Quartus II software to generate a MIF representing the initial or default state of the PLL shift register.

4. Change the MIF to account for the counters/delay settings that will be changed during PLL reconfiguration, or use the MIF as is to represent the initial state of the PLL.

5. On page 1 of the `altpll_reconfig` MegaWizard Plug-In Manager, click the **Browse** button and select the modified MIF. You can also edit the `altpll_reconfig` function instance and specify the `init_scan_file` parameter to the MIF modified in the previous step.

The Quartus II software follows the same guidelines for bypassing an *m* and *n* counter as shown in "Bypassing PLL Counters" on page 7 (i.e., to bypass an *m* and *n* counter, the 9 nominal count bits are set to 0 and the bypass bit is set to 1). Use caution when modifying the MIF for reconfiguring the counter settings. An invalid setting (bypass bit = 1 and an LSB of the *m* and *n* counter = 1) could cause the PLL counter to be disabled after reconfiguration.

The `altpll_reconfig` megafunction has a state machine that will take care of all the control operations for successfully reconfiguring the PLL. These operations include switching off `scanclk` on the 289th falling edge, clocking in `scandata` off the falling edge of the `scanclk` signal, providing an initial clear on the `scanaclr` port, and clearing the scan registers between successive reconfigurations. The `altpll_reconfig` megafunction makes the process of reconfiguring the PLL much easier.

MIF Generation

The compiler generates a MIF representing the initial state of the scan registers for any PLL configured to enable reconfiguration. This MIF can then initialize the scan-chain cache (i.e., `init_mif_file` parameter) of the `altpll_reconfig` megafunction.

The MIF is generated only if the PLL is being reconfigured, and will be determined by whether the `scanclk` port is connected or not. The name of the file generated is derived from the PLL instance name and can be found in the compilation report under the PLL Summary section.

☞ The MIF file generated by the Quartus II software is intended for post-layout simulation only. Using this MIF file to perform reconfiguration simulation using the behavioral model of the PLL could result in erroneous behavior as the counter settings determined by the `altpll` behavioral model may not match those determined by Quartus after a successful compilation.

Ports & Parameters

The parameters, input ports, and output ports for the `altpll_reconfig` megafunction are described in the following sections. Table 4 shows port parameters. Table 5 shows the input ports. Table 6 lists the `counter_type[3..0]` port specifications and Table 7 lists the `counter_param[2..0]` specifications. Table 8 shows the output ports.

Table 4. Parameters

Parameter Name	Description
scan_chain	Defines the length of the shift register that the megafunction controls. You can set this parameter to be either `long` or `short`. PLLs with this parameter set to `long` have four additional external counters and 289 bits of configuration. PLLs with this parameter set to `short` have 193 bits of configuration.
scan_init_file	Name of **.mif** or **.hex** file to be used as the initial value of the shift register cache. This file can either be 193 bits or 289 bits depending on the `scan_chain` length of the PLL. The format of the data bits must follow the format of the shift register as specified in Table 9. If not specified, then the cache could be initialized to an unknown state.

Table 5. Input Ports (Part 1 of 2)

Port Name	Required	Description
clock	Yes	Clock input used to load individual parameters, as well as to drive the PLL during reconfiguration. This port must be connected to a valid clock.
data_in[]	No	Nine-bit bus through which the parameter data can be provided when writing parameters. Some parameters do not use all nine bits. In this case, only the number of bits necessary, starting from bit `[0]`, are used (e.g., delay element values use bits `[3..0]`, bits `[8..4]` are ignored). This bus defaults to 0 if left unconnected.
counter_type[]	No	Four-bit bus that selects which counter type should be updated. Table 6 lists the mapping to determine which counter is specified for each `counter_type` value. The second two bits indicate the counter number for the *g*, *l*, and *e* counters. Additionally, the external `[e3..e0]` counters are only valid for the long shift register PLLs (PLLs 5 and 6).
counter_param[]	No	Three-bit bus selects which parameter for the given counter type should be updated. Table 7 lists the mapping to each parameter type and the corresponding parameter bit-width is defined as follows: The first two bits determine the width (i.e., `00x` = 9, `01x` = 4, `10x` = 1). See *Chapter 1 in the Stratix Device Handbook, Volume 2* for more information on the bit settings for each parameter.
read_param	No	Signal indicating that the parameter specified with `counter_type[]` and `counter_param[]` should be read from the cache and fed to `dataout[]`. The number of bits read and set on `data_out[]` is dependant on the parameter type as indicated above. This signal is sampled at the rising clock edge, at which point the parameter value begins to be read from the cache. Additionally, this signal should only be asserted for one clock cycle to prevent the parameter from being reread on any subsequent clock cycle. The busy signal will be activated as soon as `read_param` is read as asserted. While the parameter is being read, the busy signal remains asserted. Once the busy signal is de-activated, `dataout[]` will be valid and the next parameter can begin to be loaded. While busy, `dataout[]` is not valid.
write_param	No	Signal indicating that the parameter specified with `counter_type[]` and `counter_param[]` should be written into the cache with the value given in `data_in[]`. The number of bits read from `data_in[]` depends on the parameter type as indicated above. This signal is sampled at the rising clock edge, at which point the parameter value begins to be written into the cache. Additionally, this signal should only be asserted for one clock cycle to prevent the parameter from being re-written on any subsequent clock cycle. The busy signal is activated as soon as `write_param` is read and asserted. While the parameter is written, the busy signal remains asserted and the input to `datain[]` is ignored. Once the busy signal is de-activated, the device begins to write the next parameter.

Table 5. Input Ports (Part 2 of 2)

Port Name	Required	Description
reconfig	Yes	Signal indicating that the PLL should be reconfigured with the PLL settings as specified in the current cache. The device samples this signal at the rising clock edge, at which point the cached settings begin to be loaded into the PLL. Additionally, you should only assert this signal for one clock cycle to prevent the PLL from being reloaded after reconfiguration is complete. The `busy` signal is activated as soon as the `reconfig` signal is read. While the PLL is being reconfigured, the `busy` signal remains asserted. Once the `busy` signal is de-activated, you can modify the parameters as before. Additionally, during and after reconfiguration, the shift register data cache remains unchanged. This allows multiple reconfigurations while only modifying one parameter each time since all other parameter values are set with their previous value.
reset	Yes	Asynchronous reset input used to initialize the machine such that it is in a valid state. The machine must be reset before first use otherwise the state is not guaranteed to be valid. This port must be connected.
pll_scandataout	No	Input signal to be driven by the `scandataout` port on the `altpll` megafunction. The `scandataout` signal goes high when all the `scandata` bits have been shifted into the scan chain and hence this signal is used inside the `altpll_reconfig` megafunction to hold the busy signal high while the `scandata` bits are shifting through the scan chain.

Table 6. counter_type[3..0] Settings

Counter Selection	counter_type[3..0]	
	Binary	Hexadecimal
n	`00 00`	`0`
m	`00 01`	`1`
(illegal value)	`00 10`	`2`
(illegal value)	`00 11`	`3`
*g*0	`01 00`	`4`
*g*1	`01 01`	`5`
*g*2	`01 10`	`6`
*g*3	`01 11`	`7`
*l*0	`10 00`	`8`
*l*1	`10 01`	`9`
(illegal value)	`10 10`	`A`
(illegal value)	`10 11`	`B`
*e*0	`11 00`	`C`
*e*1	`11 01`	`D`
*e*2	`11 10`	`E`
*e*3	`11 11`	`F`

Table 7. counter_param[2..0] Settings			
Counter Selection	**counter_param[2..0]**		**Width**
	Binary	**Hexadecimal**	
Nominal count (for *m* and *n*)	`000`	`0`	9
Spread count (for *m* and *n*)	`001`	`1`	9
High cycles count (for *g*, *l*, and *e*)	`000`	`0`	9
low cycles count (for *g*, *l*, and *e*)	`001`	`1`	9
delay element setting	`010`	`2`	4
(illegal value)	`011`	`3`	
counter bypass bit	`100`	`4`	1
counter odd division bit (must be 0 for *m* and *n* when not using spread-spectrum)	`101`	`5`	1
Spread count bypass bit (only valid when using spread-spectrum)	`101`	`5`	1
(illegal value)	`110`	`6`	
(illegal value)	`111`	`7`	

Table 8. Output Ports

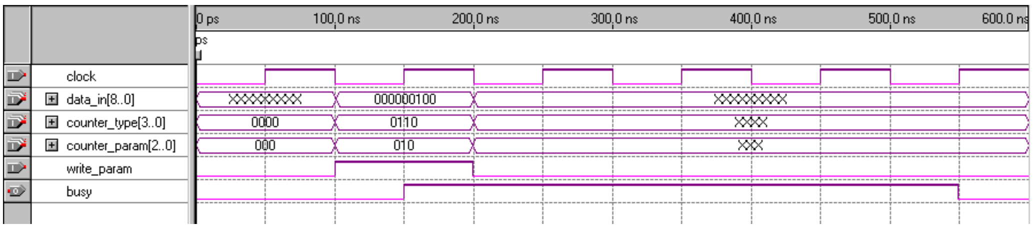
Port Name	Required (Y/N)	Description
data_out[]	No	Nine-bit bus through which the parameter data can be read back by the user. The parameter value is requested using the `counter_type[]` and `counter_param[]` values, and by asserting the `read_param` signal, at which point the value of the parameter will be loaded from the cache and driven on this bus. The data will be valid once the `busy` signal is de-asserted.
busy	No	This signal goes high when either `read_param`, `write_param`, or `reconfig` is asserted, and remains high until the specific operation is complete. While this signal is asserted, the machine will ignore its inputs and cannot be altered until it de-asserts this signal. This implies that changes (read/write operations) can only be made while the `busy` signal is de-asserted.
pll_scanclk	Yes	Signal to drive `scanclk` port on PLL to be reconfigured.
pll_scanaclr	Yes	Signal to drive `scanaclr` port on PLL to be reconfigured.
pll_scandata	Yes	Signal to drive `scandata` port on PLL to be reconfigured.

Reconfiguring the g2 Counter Delay Element Using altpll_reconfig

Set the delay setting parameter of 1.0 ns for counter *g*2 in the scan-chain cache by performing the following steps (see Figure 7):

- Set `data_in[]` to `0100` to specify a delay of 1.0 ns.
- Set `counter_type[]` to `0110` to select counter g2.
- Set `counter_param[]` is set to `010` to select the delay setting parameter.
- Assert `write_param` for one clock cycle so that at t = 150 ns, the data begins to load into the shift register cache.
- The `busy` signal is asserted by the machine at the start of data loading at t = 150 ns until t = 550 ns, at which point the data loading has successfully completed.

Figure 7. Set Delay Setting Parameter Off 1.0 ns for Counter g2

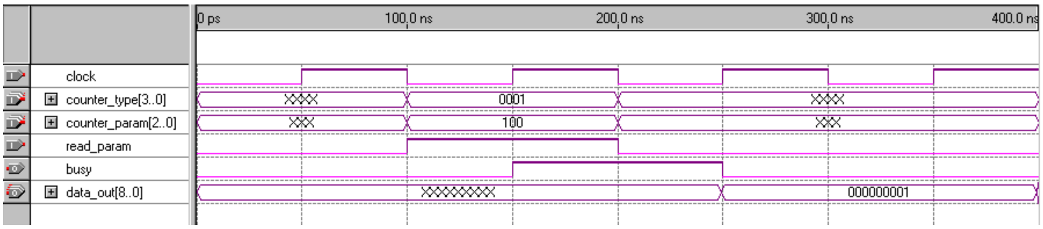


Read Out Value of the Counter Bypass Parameter for Counter m

Figure 8 illustrates how to read out the counter bypass parameter for counter *m*. To read out the counter bypass parameter for counter *m*, perform the following steps:

- Set `counter_type[]` to `0001` to select counter *m*
- Set `counter_param[]` to `100` to select the counter bypass parameter
- Assert `read_param` for one clock cycle so that at t = 150 ns, the data begins to load into the `data_out[]` registers.
- `The busy` signal is asserted by the machine at the start of data loading at t = 150 ns until t = 250 ns, at which point the data loading is complete. Now, `data_out[]` is valid and contains the value of the counter bypass bit in `data_out[0]` (in this example, the value is `1`).

Figure 8. Read Value of Counter Bypass Parameter for Counter M Waveform

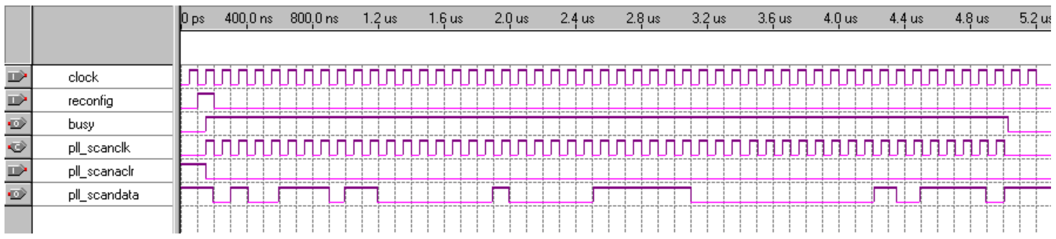


Reconfigure PLL with Current Parameters in Shift Register Cache

Figure 9 illustrates how to reconfigure the PLL with the data in the shift register cache. To reconfigure the PLL with the data in the shift register cache, perform the following steps:

- Assert `reconfig` for one clock cycle so that at t = 150 ns, the data in the shift register cache begins to load into the PLL through the PLL scan ports.
- `The busy` signal is asserted by the machine at the start of data loading at t = 0.150 µs until t = 5.000 µs, at which point the PLL has finished reconfiguration.
- `pll_scanclk`, `aclr`, and `data` signals drive the PLL to reconfigure it with the parameters as given in the shift register cache. The `pll_scandata` waveform shown in Figure 9 is only an example and not an indication of actual data (a real waveform has at least 193 bits of data).

Figure 9. Reconfigure PLL with Current Parameters in Shift Register Cache Waveform



This example does not use the `reset` and `pll_scandata_out` ports.

PLL Configuration Scan Register Bit Map

Advanced PLL users can also select the counter and delay element settings manually based on information detailed in the hardware implementation section. After determining the individual configuration bit settings for the different counters and delay elements, arrange the bits as shown in Table 9. This table provides a bit map for the shift registers. Bit `[0]` is the last bit to be shifted into the shift register. Bit `[288]` is the first bit shifted in for enhanced PLLs 5 and 6, while bit `[192]` is the first bit for PLLs 11 and12.

Table 9. PLL Configuration Shift Register Bit Map (Part 1 of 3) *Notes (1), (2), (3)*

PLL Shift Register Bit Map									PLL Parameter	Size (Bits)
LSB								MSB		
0	1	2	3	4	5	6	7	8	*n* counter nominal count	9
								9	*n* counter bypass bit	1
10	11	12	13	14	15	16	17	18	*n* counter spread count	9
								19	*n* counter spread spectrum bypass bit	1
					20	21	22	23	*n* delay element setting	4
24	25	26	27	28	29	30	31	32	*m* counter nominal count	9
								33	*m* counter bypass bit	1
34	35	36	37	38	39	40	41	42	*m* counter spread count	9
								43	*m* counter spread spectrum bypass bit	1
					44	45	46	47	*m* delay element setting	4
48	49	50	51	52	53	54	55	56	*g*0 counter high cycles count	9
								57	*g*0 counter bypass bit	1
58	59	60	61	62	63	64	65	66	*g*0 counter low cycles count	9
								67	*g*0 counter odd division bit	1

Table 9. PLL Configuration Shift Register Bit Map (Part 2 of 3) Notes (1), (2), (3)

PLL Shift Register Bit Map									PLL Parameter	Size (Bits)
LSB								MSB		
					68	69	70	71	*g*0 delay element setting	4
72	73	74	75	76	77	78	79	80	*g*1 counter high cycles count	9
								81	*g*1 counter bypass bit	1
82	83	84	85	86	87	88	89	90	*g*1 counter low cycles count	9
								91	*g*1 counter odd division bit	1
					92	93	94	95	*g*1 delay element setting	4
96	97	98	99	100	101	102	103	104	*g*2 counter high cycles count	9
								105	g2 counter bypass bit	1
106	107	108	109	110	111	112	113	114	*g*2 counter low cycles count	9
								115	*g*2 counter odd division bit	1
					116	117	118	119	*g*2 delay element setting	4
120	121	122	123	124	125	126	127	128	*g*3 counter high cycles count	9
								129	*g*3 counter bypass bit	1
130	131	132	133	134	135	136	137	138	*g*3 counter low cycles count	9
								139	*g*3 counter odd division bit	1
					140	141	142	143	*g*3 delay element setting	4
144	145	146	147	148	149	150	151	152	*l*0 counter high cycles count	9
								153	*l*0 counter bypass bit	1
154	155	156	157	158	159	160	161	162	*l*0 counter low cycles count	9
								163	*l*0 counter odd division bit	1
					164	165	166	167	*l*0 delay element setting	4
168	169	170	171	172	173	174	175	176	*l*1 counter high cycles count	9
								177	*l*1 counter bypass bit	1
178	179	180	181	182	183	184	185	186	*l*1 counter low cycles count	9
								187	*l*1 counter odd division bit	1
					188	189	190	191	*l*1 delay element setting	4
192	193	194	195	196	197	198	199	200	*e*0 counter high cycles count *(4)*	9
								201	*e*0 counter bypass bit *(4)*	1
202	203	204	205	206	207	208	209	210	*e*0 counter low cycles count *(4)*	9
								211	*e*0 counter odd division bit *(4)*	1
					212	213	214	215	*e*0 delay element setting *(4)*	4
216	217	218	219	220	221	222	223	224	*e*1 counter high cycles count *(4)*	9
								225	*e*1 counter bypass bit *(4)*	1
226	227	228	229	230	231	232	233	234	*e*1 counter low cycles count *(4)*	9
								235	*e*1 counter odd division bit *(4)*	1
					236	237	238	239	*e*1 delay element setting *(4)*	4
240	241	242	243	244	245	246	247	248	*e*2 counter high cycles count *(4)*	9

Table 9. PLL Configuration Shift Register Bit Map (Part 3 of 3) *Notes (1), (2), (3)*

PLL Shift Register Bit Map									PLL Parameter	Size (Bits)
LSB								MSB		
								249	e2 counter bypass bit *(4)*	1
250	251	252	253	254	255	256	257	258	e2 counter low cycles count *(4)*	9
								259	e2 counter odd division bit *(4)*	1
					260	261	262	263	e2 delay element setting *(4)*	4
264	265	266	267	268	269	270	271	272	e3 counter high cycles count *(4)*	9
								273	e3 counter bypass bit *(4)*	1
274	275	276	277	278	279	280	281	282	e3 counter low cycles count *(4)*	9
								283	e3 counter odd division bit *(4)*	1
					284	285	286	287	e3 delay element setting *(4)*	4
								288	Transfer enable register *(4), (5)*	1
									Shift register length *(4)*	289

Notes to Table 9:

(1) The first bit shifted into the shift register is the transfer enable bit. You should set this bit to logic high.
(2) For all registers in the chain, the MSB is shifted in first and LSB last.
(3) For enhanced PLLs 11 and 12, the shift register ends with the L1 delay element setting and total shift register length is 193 (including a one-bit transfer enable register bit).
(4) These values are for enhanced general-purpose PLLs with only eight external outputs (PLLs 5 and 6).
(5) This is bit `[192]` for PLLs 11 and 12.

Design Considerations

You must consider the following aspects of the PLL reconfiguration feature.

Counters & Delay Elements

Changing pre-scale and feedback counter settings (m, n) will affect the PLL VCO frequency, which may require the PLL to relock to the reference clock. Changing the delay element settings (Δt_m, Δt_n) will change the phase relationship of the output clocks with respect to the reference clock, which also requires the PLL to relock. Although the exact effect depends on how drastic the change is, any change typically requires resynchronization.

Adding the Δt_n delay element would delay all PLL clock outputs with respect to the reference clock. You can use this delay element to delay all outputs by up to +3.0 ns. Post-scale delay elements can add an additional +3.0 ns of delay for a total of +6.0 ns. However, any two PLL outputs can only differ by a maximum of 3.0 ns.

Adding the Δt_m delay element would pull in all the PLL clock outputs with respect to reference clock, effectively adding a negative delay (because the Δt_m delay element is in the feedback path). You can use this delay element to advance the clock by +3.0 ns, or in other words delay the output clock by –3.0 ns.

When making changes to the loop elements (m, n, Δt_m, Δt_n), you should disable the PLL outputs to the FPGA logic array using the `CLKENA` signals, eliminating an over-frequency condition affecting system logic.

Changes to post-scale counters or delay elements will not affect PLL lock or VCO frequency. But large changes to delay element settings (greater than 250-ps increments) could lead to glitches on the output clock. You should either use the `CLKENA` signals or change the delay element settings in small increments.

When phase relationship between output clocks is a factor, Altera recommends that you resynchronize the PLL using the `ARESET` signal. This will reset all internal PLL counters and reinitiate the locking process. After the PLL relocks, all the output clocks will have the correct phase relationship. During PLL reconfiguration and/or reset you can disable clock outputs from the PLL to avoid any change of state in the system.

Shift Register Control Operations

The reconfiguration circuit works the same way for both the long chain and the short chain.

Figure 10 on page 23 shows a block diagram of the shift registers used for PLL reconfiguration. Because you can directly control the clock, you must make sure not to clock the chain when you do not intend to shift data. This is important when the device first powers up because there is data in the chain from the initial programming. If the chain is clocked before it is cleared, the device could clock a "`1`" into the last bit in the chain, and whenever a "`1`" is clocked into the last bit in the chain (transfer enable bit), the PLL will start reconfiguring itself with whatever data is in the chain. Initially, make sure `scanclk` is held either high or low until the chain is cleared. If `scanaclr` is held high from the very beginning, `scanclk` can toggle without causing any harm. Also, after the chain is cleared, `scanclk` can toggle while `scandata` is low without changing the state of the chain. Figure 11 on page 24 shows a typical reconfiguration waveform.

After scandata is shifted into register 287, (or register 191 for short chain PLLs) on the rising edge of `scanclk`, the clock must go back to a low state to clock the last negative edge flip-flop (register 288 or 192).

After the clock goes low, there must be a quiet period to allow the bits to be synchronously loaded from the scan registers into all the counters and delay elements in the PLL. During t_{QUIET}, `scanclk` and `scanaclr` must remain low.

☞ Altera recommends using the `altpll_reconfig` megafunction because it takes care of these operations that are required to successfully reconfigure the PLL.

Because a routing delay of `scanclk` from the logic array to PLL can be greater than the routing delay of `scandata` from the logic array to the PLL, you must protect your design against a positive hold time. Clocking `scandata` off the falling edge of `scanclk` will protect against a positive hold time by giving a half cycle setup time and a half cycle hold time. Figure 11 shows this clocking scheme for `scandata`.

Another routing concern is the delay on `scanaclr`. The routing delay of `scanaclr` can be longer or shorter than that of `scanclk`. So if `scanaclr` goes high too close to a rising edge of `scanclk`, it is uncertain which one will get to the chain first. For example, if the `scanaclr` delay is 1-ns longer than the `scanclk` delay and if `scanclk` rises 0.5 ns after `scanaclr`, then `scanaclr` will cancel the rising edge of `scanclk`. However, if the `scanclk` delay is 1-ns larger than the `scanaclr` delay and if `scanclk` rises 0.5 ns before `scanaclr`, then the chain will see the unintended rising edge of `scanclk`. To prevent this, the rising edge of `scanclk` should not occur within 5 ns of the rising edge of `scanaclr`.

Use the following steps to reconfigure a PLL:

6. Disable all PLL outputs using the `CLKEN` signals.

7. Assert `scanaclr` once for 5 ns before starting `scanclk`.

8. Scan in new counter and delay element settings through the `scandata` port.

9. Make sure `scanclk` is turned off after the 289th falling edge, and wait until the end of the quiet period.

10. Assert `scanaclr` again for 5 ns to clear all the scan registers prior to another reconfiguration.

11. Reset the PLL using `ARESET` to maintain phase relationships between output clocks.

12. Re-enable PLL outputs after detecting a valid lock.

If you cannot disable the PLL outputs, make gradual and incremental changes to internal components. For example, if the reference clock input is 100 MHz and the *n* and *m* counters are set to 5 and 25, respectively, the VCO will be running at 500 MHz. If you change the VCO frequency to 600 MHz, you can set the *n* and *m* counters to 5 and 30, respectively. You should gradually change the feedback counter (*m*) from 25 to 30 in increments of 1. This reduces the risk of an over frequency condition (where the output frequency is higher than required) and avoids a loss of lock condition.

If the PLL is losing `lock` during or after PLL reconfiguration, the *m* and *n* counter settings may have changed during the reconfiguration process. If you change the *m* and *n* counter/delay element settings, the PLL could lose lock. For example, if the input clock frequency is 350 MHz and the output clock frequency is 350 MHz, the Quartus II software could set $m = 2$ and $n = 2$, a 350-MHz VCO frequency, and $k = 1$ to get the above frequency combination.

During reconfiguration, if you set your scan bits to get an $m = n = 4$, you still keep the same input/output frequency combinations. However, this setting requires that the *m* and *n* counter values change, causing the PLL to lose `lock`.

For more information on *m* and *n* values to ensure that these settings are not altered by mistake during PLL reconfiguration, see the Quartus II software compilation report.

Reconfiguration Chain Description

The first rising edge on `scanclk` registers the first bit, which is the transfer enable bit into the scan register `0`; the 288th rising edge of `scanclk` registers the transfer enable bit into scan register 287; and on the falling edge of the 288th clock, this bit is registered into scan register 288. Once the transfer enable bit is registered, a parallel set of registers are loaded synchronously with the new counter settings based on each counter’s output clock. After the new settings are loaded into the PLL configuration bits, the `reconfig_done` signal goes high. This signal is `AND`ed with the `scanaclr` signal to reset the transfer enable bit (register 288) after a successful reconfiguration. In the Quartus II software version 2.2 and higher, the `altpll` megafunction automatically enables the `scandataout` port when PLL reconfiguration is enabled. The `scandataout` signal goes high and then low, signaling that the PLL has been reconfigured with the new settings.

The `scandataout` signal stays high for about 3 `scanclk` cycles. When the `reconfig_done` signal goes high (signalling that all the counters have been updated with new settings), register 288 is cleared and the scandataout signal is then de-asserted.

`The reconfig_done` signal is internal to the PLL reconfiguration circuit and is not available for users.

Please refer to the *Stratix FPGA Errata Sheet* for more information on the `scandataout` signal.

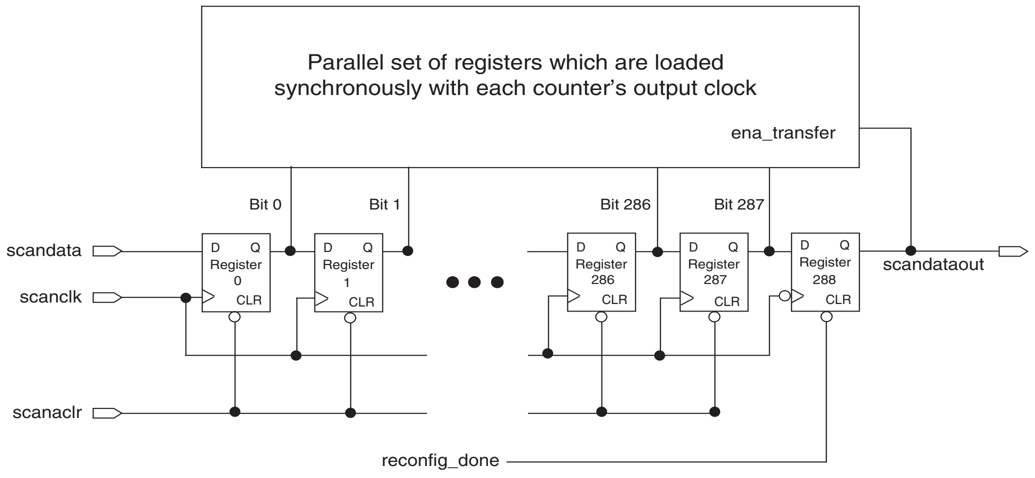
Additionally, to check to see if the correct bits were scanned into the chain during PLL reconfiguration, you can follow the procedure outlined below:

1. Reset the PLL using the `ARESET` port.

2. Enable `scanclk` and read out the contents of the shift register one bit at a time through the `scandataout` port. This port is enabled automatically when you select the PLL reconfiguration feature in the Quartus II software version 2.2.

Therefore, you can check for potential errors or bit ordering mistakes in your scan data bit stream going into the `scandata` port of the PLL.

Figure 10. Reconfiguration Chain Block Diagram *Note (1)*



Note to Figure 10:

(1) This figure shows the long chain for enhanced PLLs (5, 6). For short chain PLLs (11, 12), replace 286, 287, and 288 with 190, 191, and 192.

Figure 11. EPLL Reconfiguration Waveform

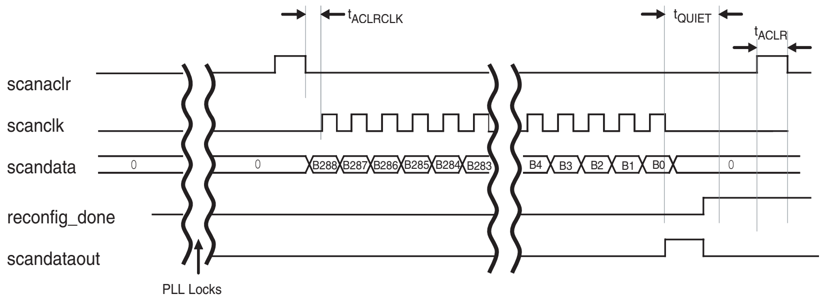


Table 10 shows PLL parameters.

Table 10. PLL Parameters

Parameter	Value
f_{MAX}	The `scanclk` frequency can be a maximum of 22 MHz
t_{ACLR}	Altera recommends a minimum of 5-ns high pulse width to reset the scan registers.
$t_{ACLRCLK}$	Time from the falling edge of `scanaclr` to the rising edge of `scanclk`. Altera recommends a minimum of 5 ns.
t_{QUIET}	During t_{QUIET}, `scanclk` and `scanaclr` should not toggle. They should both be held low as the PLL configuration bits are updated with the new counter settings. Altera recommends a minimum of 50 ns, or twice the clock period of the slowest PLL counter, whichever is larger.

Design Approaches

There are many ways to set up the shift register for using the PLL reconfiguration feature, including the following:

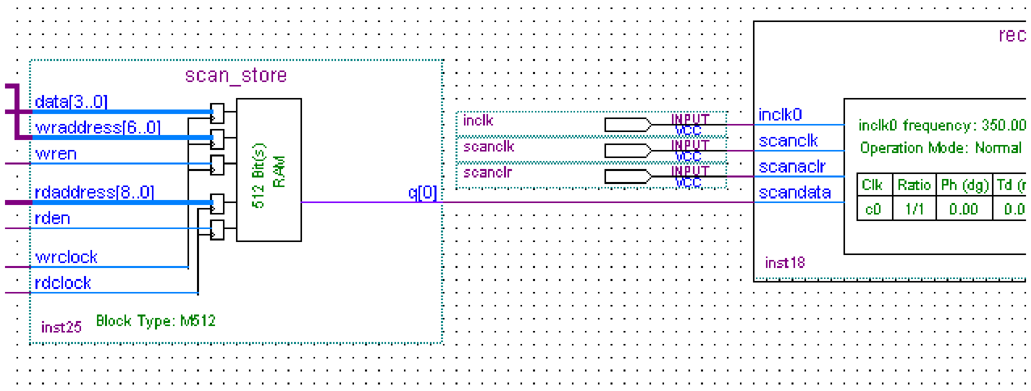
- Scan bits in an M512 block
- Serial shift register chain in logic elements (LEs)
- `altpll_reconfig` megafunction

☞ Altera recommends using the `altpll_reconfig` megafunction to set up the shift register.

Scan Bits in Memory

Figure 12 shows a design where the scan bits are arranged in a M512 block of memory and are read out starting from bit `[288]` which is the transfer enable bit. In Figure 12, the write data port is four bits wide as the delay element bits corresponding to the PLL counters are 4 bits wide. Since the bits read out serially into the PLL's `scandata` port, the read address is nine bits wide to allow the device to read 289 bits out from the M512 memory block. The `scanclk` and `scanaclr` inputs to the PLL can either come from pins or from internal logic. If the inputs come from pins, an intelligent controller (e.g., microprocessor) must take care of operations like turning off `scanclk` after the 289th falling edge of `scanclk.` See "Design Considerations" on page 19.

Figure 12. Scan Data Bits Read Out from an M512 Memory Block

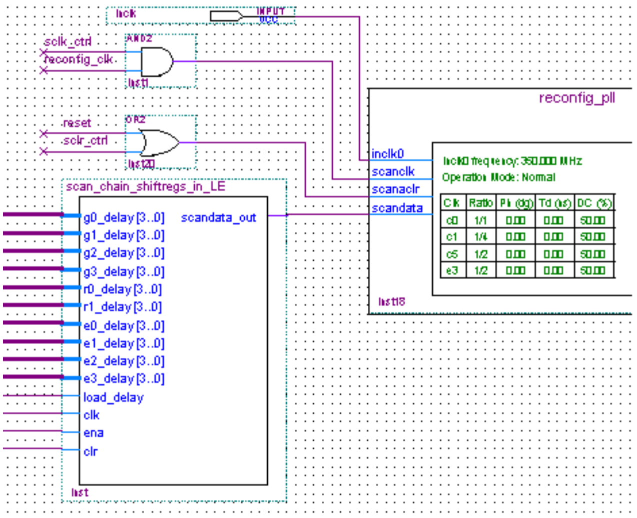


Clk	Ratio	Ph (dg)	Td (
c0	1/1	0.00	0.0

Serial Shift Register Chain in LEs

You can build your own serial shift register chain in LEs emulating the dedicated scan registers in silicon to shift the bits into the `scandata` PLL port. An example design is attached which follows this approach. Figure 13 shows the scan data bits coming from a bank of LEs in the sub design (`scan_chain_shiftregs_in_le`). You can build custom logic to switch off the scan clock after the 289th falling edge or assert `scanaclr` before and after PLL reconfiguration.

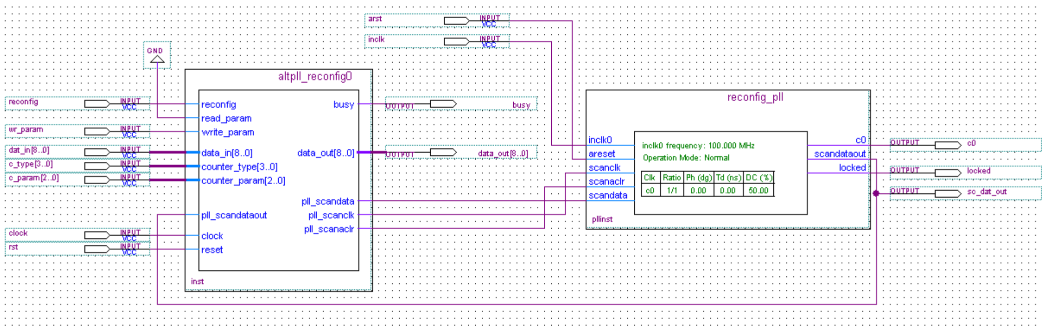
Figure 13. Scan Data Coming from a Shift Register Chain Built Using LEs



altpll_reconfig Megafunction

The `altpll_reconfig` megafunction provides an interface to implement the reconfiguration of the PLL's parameters. You can use this megafunction to configure the PLL timing and change each parameter individually, as well as specify a default configuration using a standard **.mif** or **.hex** file. Figure 14 shows how this megafunction connects to the PLL.

Figure 14. Megafunction Connecting to PLL



For more information, see "Example 3: altpll_reconfig Design".

Conclusion

PLL reconfiguration is a powerful feature that system designers can use to vary the clock output frequency and clock delay on the fly. Important considerations such as PLL loss of lock, glitches, and output phase relationships must be made when selecting the PLL counter and delay element settings. Altera recommends changing PLL parameters gradually or incrementally over single-step drastic changes. PLL reconfiguration time is also typically less that 20 µs, allowing you to rapidly switch operating modes. The flexibility offered by the Stratix PLL make it a superior clock management system.

Design Examples

This section provides examples on implementing reconfigurable PLLs in your design.

Example 1: Shift Register in LEs

Uncompress the design and compile it in the Quartus II software. The design has been setup such that when you toggle (1 -> 0) the `rst` signal, the G0 counter is reconfigured to produce a shift of 250 ps. You can tie the `rst` signal to a push button and push the button multiple times to get incremental 250-ps jumps all the way from 0 to 3000 ps.

Example 2: altpll_reconfig Design with the MIF

Uncompress the design and compile it in the Quartus II software. The MIF is setup such that you reconfigure the *g*0 counter to divide by 12 from an initial setting of divide by 6. This effectively changes the output clock frequency from 100 to 50 MHz. To reconfigure other settings or counters, you will need to first enable the appropriate counter that you wish to reconfigure in the `altpll` MegaWizard Plug-In Manager and then change the corresponding scan bits (as shown in Table 3) in the MIF.

Example 3: altpll_reconfig Design

Uncompress the design and compile it in the Quartus II software. The Vector Waveform File (**.vwf**) is setup such that you reconfigure the *g*0 counter to first divide by 4 to get an output frequency of 150 MHz and then divide by 8 to get a output frequency of 75 MHz, from an initial setting of divide by 6 which resulted in a 100-MHz output clock. To reconfigure other settings or counters, enable the appropriate counter that you wish to reconfigure in the `altpll` MegaWizard Plug-In Manager and then set the corresponding write parameters, as shown in Table 8.

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
www.altera.com
Applications Hotline:
(800) 800-EPLD
Literature Services:
literature@altera.com

I.S. EN ISO 9001