Implementing Fractional PLL Reconfiguration with Altera PLL and Altera PLL Reconfig Megafunctions

AN-661-3.1 Application Note

This application note describes the flow for implementing fractional phase-locked loop (PLL) reconfiguration and dynamic phase shifting for fractional PLLs in 28-nm devices (Arria® V, Cyclone® V, and Stratix® V device families) with the Altera PLL and Altera PLL Reconfig megafunction IP cores in the Quartus® II software.

This application note discusses the following topics:

Functional Description

Fractional PLLs use divide counters and different voltage-controlled oscillator (VCO) taps to perform frequency synthesis and phase shifts. For example, you can reconfigure the counter settings and dynamically phase-shift the fractional PLL output clock in the PLLs of 28-nm devices. You can also change the charge pump and loop filter components, which dynamically affect the fractional PLL bandwidth. You can use these fractional PLL components to update the clock frequency, fractional PLL bandwidth, and phase shift in real time, without reconfiguring the entire FPGA.

Fractional PLL Reconfiguration in 28-nm Devices

The fractional PLLs in 28-nm devices also support integer PLL. Fractional PLLs provide robust clock management and synthesis for device clock management, external system clock management, and high-speed I/O interfaces.

The fractional PLLs in 28-nm devices support dynamic reconfiguration. While the device is in user mode, you can download a new fractional PLL configuration in real time without reconfiguring the entire FPGA.

The following fractional PLL components are reconfigurable in real time using the dynamic reconfiguration IP:

- Postscale output counter (C)
- Feedback counter (*M*)
- Prescale counter (*N*)

101 Innovation Drive
San Jose, CA 95134
www.altera.com

- Charge-pump current (I_{CP}), and loop-filter components (R, C)

  The Quartus II software version 12.0 supports I_{CP}, R and C reconfiguration.

- Dynamic phase shifting of each counter
- Fractional division (M_{FRAC}) for Delta Sigma Modulator (DSM)

Applications that operate at multiple frequencies can benefit from fractional PLL reconfiguration in real time. Fractional PLL reconfiguration is also beneficial in prototyping environments, allowing you to sweep fractional PLL output frequencies and adjusting the clock output phase at any stage of your design. You can also use this feature to adjust clock-to-out (t_{CO}) delays in real time by changing the output clock phase shift.

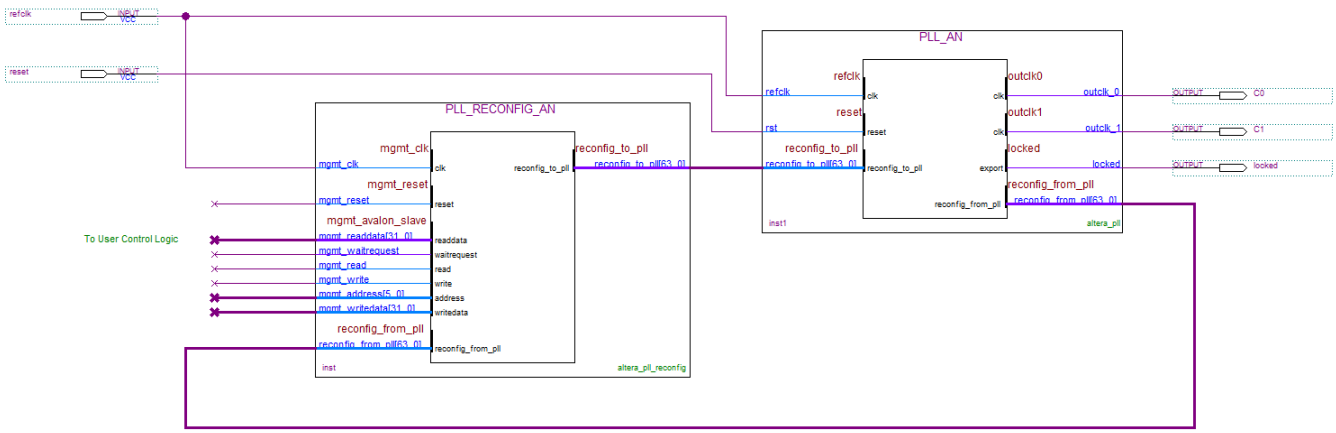
Implementing Fractional PLL Reconfiguration on 28-nm Devices Using the Quartus II Software

You can use the Altera PLL Reconfig IP to enable reconfiguration circuitry in the Altera Phase-Locked Loop (Altera PLL) IP core instantiation in your design. The Altera PLL Reconfig IP core simplifies the fractional PLLs reconfiguration process.

The Altera PLL Reconfig IP core interacts with a user control logic and a bus that connects directly to the Altera PLL instance using the Avalon Memory-Mapped (Avalon-MM) interface.

Figure 1 shows the connection between the Altera PLL and the Altera PLL Reconfig IP cores.

Figure 1. Altera PLL Reconfig and Altera PLL IP cores in the Quartus II Software



To connect the Altera PLL and Altera PLL Reconfig instances in your design, follow these steps:

1. Connect the reconfig_to_pll[63:0] bus on the Altera PLL Reconfig instance to the reconfig_to_pll[63:0] bus on the Altera PLL instance.
2. Connect the reconfig_from_pll[63:0] bus on the Altera PLL instance to the reconfig_from_pll[63:0] bus on the Altera PLL Reconfig instance.
3. Connect the mgmt_clk signal to a clock source. The mgmt_clk signal can be a free running clock, eliminating the need to control the start and stop of the mgmt_clk signal.
4. To perform Avalon read or write operations, connect the mgmt_reset, mgmt_read_data[31:0], mgmt_write, mgmt_address[5:0], mgmt_write_data[31:0] buses, and the mgmt_wait_request and mgmt_read signals to user control logic.

Control Interface

The control interface for the Altera PLL Reconfig IP core is an Avalon-MM slave interface, which the master user logic controls. External user logic uses these Avalon ports to reconfigure the fractional PLL settings directly.

Table 1 lists the Avalon-MM signals in the Altera PLL Reconfig IP core. Logic are sampled with rising edge of the clock. The first rising edge after deassertion of the waitrequest signal will sample a read/write command.

Table 1. Avalon-MM Signals in the Altera PLL Reconfig IP Core

Port	Direction	Description
mgmt_read_data[31:0]	Output	Data read from the Altera PLL Reconfig IP core when you assert the mgmt_read signal.
mgmt_write_data[31:0]	Input	Data written to the Altera PLL Reconfig IP core when you assert the mgmt_write signal.
mgmt_address[5:0]	Input	Specifies the address of the memory mapped register for a read or write operation.
mgmt_read	Input	Active high signal. Asserted to indicate a read operation. When present, read data is available on the mgmt_read_data bus.
mgmt_write	Input	Active high signal. Asserted to indicate a write operation. When present, the mgmt_write_data bus requires write data.
mgmt_reset	Input	Active-high signal that resets all PLL counters to their initial values. When this signal is driven high, the PLL resets its counters, clears the PLL outputs, and loses lock. After this signal goes low again, the lock process begins and the PLL resynchronizes to its input reference clock.
mgmt_waitrequest	Output	Active high signal. When the ALTERA_PLL_RECONFIG IP core asserts this signal it will ignore read or write operations.

Table 2 lists the fractional PLL dynamic reconfiguration registers and settings.

Table 2. Fractional PLL Dynamic Reconfiguration and Status Register (Part 1 of 2)

Register Name	Register Size	Address (Binary)	Counter Bit Settings	Read/Write
Mode Register	1	000000	■ Write 0 for waitrequest mode ■ Write 1 for polling mode	Read/Write
Status Register	1	000001	■ 0 = busy ■ 1 = ready	Read
Start Register	1	000010	Write either 0 or 1 to start fractional PLL reconfiguration or dynamic phase shift	Write
N Counter	18	000011	■ N_counter[7:0] = low_count ■ N_counter[15:8] = high_count ■ Total_div = high_count + low_count ■ N_counter[16] = bypass enable (3) ■ N_counter[16] = 0, $f_{REF} = f_{IN}$/Total_div ■ N_counter[16] = 1, $f_{REF} = f_{IN}$ (*N* counter is bypassed) ■ N_counter[17] = odd division (3) ■ N_counter[17] = 0, even division, duty cycle = high_count/Total_div ■ N_counter[17] = 1, odd division, duty cycle = (high_count-0.5)/Total_div	Read/Write
M Counter	18	000100	■ M_counter[7:0] = low_count ■ M_counter[15:8] = high_count ■ Total_div = high_count + low_count ■ M_counter[16] = bypass enable (3) ■ M_counter[16] = 0, $f_{FB} = f_{VCO}$/Total_div ■ M_counter[16] = 1, $f_{FB} = f_{VCO}$ (*M* counter is bypassed) ■ M_counter[17] = odd division (3) ■ M_counter[17] = 0, even division, duty cycle = high_count/Total_div ■ M_counter[17] = 1, odd division, duty cycle = (high_count-0.5)/Total_div	Read/Write
C Counter	23	000101	■ C_counter[7:0] = low_count ■ C_counter[15:8] = high_count ■ Total_div = high_count + low_count ■ C_counter[16] = bypass enable (3) ■ C_counter[16] = 0, $f_{OUT} = f_{VCO}$/Total_div ■ C_counter[16] = 1, $f_{OUT} = f_{VCO}$ (*C* counter is bypassed) ■ C_counter[17] = odd division (3) ■ C_counter[17] = 0, even division, duty cycle = high_count/Total_div ■ C_counter[17] = 1, odd division, duty cycle = (high_count-0.5)/Total_div ■ C_counter[22:18] is a five bit binary number ranging from 00000 to 10001 (0 -17) to select which C counter to change. For example, if you want to change C2, set C_counter[22:18] to 00010.	Read/Write (2)

Table 2. Fractional PLL Dynamic Reconfiguration and Status Register (Part 2 of 2)

Register Name	Register Size	Address (Binary)	Counter Bit Settings	Read/Write
Dynamic_Phase_Shift	22	000110	■ `Dynamic_Phase_Shift[15:0]` = number of shifts ■ Number of shifts = the number of times you want to shift the output clock. Every time you perform a shift, the actual amount of shift is 1/8 of the VCO period. For example, if the VCO is running at 1.6 GHz, each phase shift equals to 78.125 ps. ■ `Dynamic_Phase_Shift[20:16]` = `cnt_select`. ■ `cnt_select` is a five bit value that specifies which counter output is shifted. For more information about `cnt_select` mapping, refer to Table 4. ■ `Dynamic_Phase_Shift[21]` = `up_dn` ■ `up_dn` = The direction of the shift. ■ `up_dn` = `1` (positive phase shift). ■ `up_dn` = `0` (negative phase shift).	Write
M Counter Fractional Value (K)	32	000111	Fractional part of the *M* counter (for DSM). The actual fractional values are: ■ M_{FRAC} = K[X:0]/2^X (X=8, 16, 24 or 32) *(1)* ■ M counter final value = `Total_div` for M Counter + M_{FRAC}	Write
Bandwidth Setting	4	001000	For the bandwidth settings, refer to the PLL Reconfiguration Calculator. You can download the PLL Reconfiguration Calculator from the Documentation: Application Notes page on the Altera website.	Read/Write
Charge Pump Setting	3	001001	For the charge pump settings, refer to the PLL Reconfiguration Calculator. You can download the PLL Reconfiguration Calculator from the Documentation: Application Notes page on the Altera website.	Read/Write
VCO DIV Setting	1	011100	Enable or disable /2 divider for VCO to keep VCO frequency in operating range. ■ 0: VCO DIV = 1 ■ 1: VCO DIV = 2	Read/Write
MIF Base Address	9	011111	Base address for the start of a PLL profile in a **.mif**.	Write

Notes to Table 2:

(1) K counter reconfiguration is effective only when you configure the PLL in fractional mode prior to reconfiguration. For optimum performance, set the M_{FRAC} value between 0.05 and 0.95. X = fractional carry bit, determined in the Altera PLL parameter editor. The default value for X is 24 and cannot be reconfigured during PLL reconfiguration.

(2) For C counter read operation, use the address for the selected counter in Table 3.

(3) The bypass enable bit, even division bit, and odd division bit of the M, N, C counters support write operation only.

Table 3 lists the corresponding address for selected counter during read operation.

Table 3. Corresponding Address for Selected Counter During Read Operation

Address (Binary)	Selected Counter	Counter Bit Setting
001010	Counter C0	■ C_counter[7:0] = low_count ■ C_counter[15:8] = high_count ■ Total_div=high_count+low_count
001011	Counter C1	
001100	Counter C2	
001101	Counter C3	
001110	Counter C4	
001111	Counter C5	
010000	Counter C6	
010001	Counter C7	
010010	Counter C8	
010011	Counter C9	
010100	Counter C10	
010101	Counter C11	
010110	Counter C12	
010111	Counter C13	
011000	Counter C14	
011001	Counter C15	
011010	Counter C16	
011011	Counter C17	

Table 4 lists the mapping of the cnt_select (Dynamic_Phase_Shift[20:16]) and the counter for dynamic phase shift.

Table 4. Mapping for cnt_select (Dynamic_Phase_Shift[20:16]) (Part 1 of 2)

Device Family	Dynamic_Phase_Shift[20:16]	Dynamic Phase Shift Selected Counter
Arria V, Cyclone V, and Stratix V	5'b00000	Counter C0
Arria V, Cyclone V, and Stratix V	5'b00001	Counter C1
Arria V, Cyclone V, and Stratix V	5'b00010	Counter C2
Arria V, Cyclone V, and Stratix V	5'b00011	Counter C3
Arria V, Cyclone V, and Stratix V	5'b00100	Counter C4
Arria V, Cyclone V, and Stratix V	5'b00101	Counter C5
Arria V, Cyclone V, and Stratix V	5'b00110	Counter C6
Arria V, Cyclone V, and Stratix V	5'b00111	Counter C7
Arria V, Cyclone V, and Stratix V	5'b01000	Counter C8
Arria V and Stratix V	5'b01001	Counter C9
Arria V and Stratix V	5'b01010	Counter C10
Arria V and Stratix V	5'b01011	Counter C11
Arria V and Stratix V	5'b01100	Counter C12
Arria V and Stratix V	5'b01101	Counter C13

Table 4. Mapping for cnt_select (Dynamic_Phase_Shift[20:16]) (Part 2 of 2)

Device Family	Dynamic_Phase_Shift[20:16]	Dynamic Phase Shift Selected Counter
Arria V and Stratix V	5'b01110	Counter C14
Arria V and Stratix V	5'b01111	Counter C15
Arria V and Stratix V	5'b10000	Counter C16
Arria V and Stratix V	5'b10001	Counter C17
Arria V, Cyclone V, and Stratix V	5'b11111	All C Counters
Arria V, Cyclone V, and Stratix V	5'b10010	M Counter

Reconfiguring Fractional PLL Settings

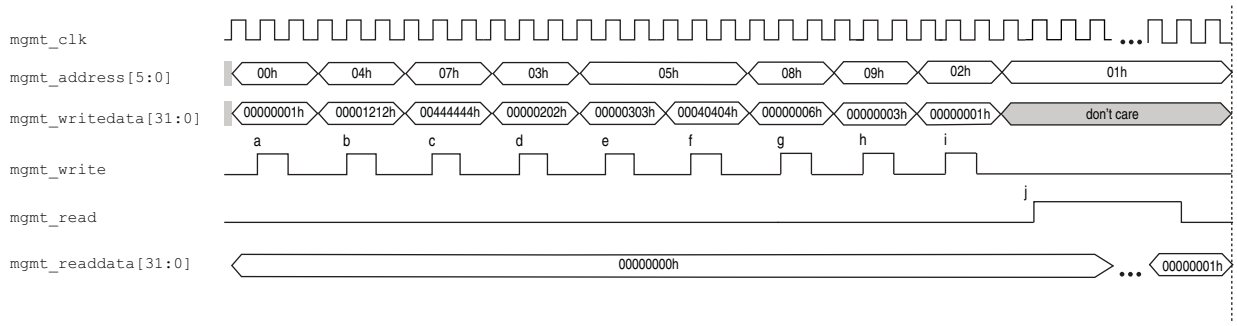
You can dynamically reconfigure fractional PLLs with the Avalon-MM interface. To perform dynamic reconfiguration, follow these steps:

1. Through an Avalon write operation, write to the mode register a value of `0` or `1` at the startup of the Altera PLL Reconfig IP core. The mode register determines whether the Altera PLL Reconfig IP core operates in waitrequest or polling mode.
2. Specify the element and its new value through an Avalon write operation. For more information about the address for each reconfigurable element, refer to Table 2 on page 4 and Table 4 on page 6.
3. Repeat Step 2 for all the reconfigurable elements (*N*, *M*, C counters, M_{FRAC} value and others) that you want to change.
4. Through an Avalon write operation, write either `0` or `1` to the start register. Writing to the start register triggers the dynamic reconfiguration, the dynamic phase shift, or both:
   a. If you set the mode register to `0` (waitrequest mode) in Step 1, the Altera PLL Reconfig IP core asserts the `mgmt_waitrequest` signal until after the reconfiguration. You can only perform another Avalon read or write operation after the Altera PLL Reconfig IP core deasserts the `mgmt_waitrequest` signal, or
   b. If you set the mode register to `1` (polling mode) in Step 1, the Altera PLL Reconfig IP core writes `0` (busy) to the status register. You can poll bit 0 of the status register periodically by performing Avalon read operations to ensure that the reconfiguration is complete. The Altera PLL Reconfig IP core ignores any new reconfiguration instructions (Avalon write operations) until a value of `1` has been read from the status register.

Lock the fractional PLL to the reference clock before you perform the dynamic reconfiguration or the dynamic phase shifting.

Figure 2 shows a waveform example for performing dynamic reconfiguration in reconfiguring *M*, M_{FRAC}, *N* and C counters.

Figure 2. Waveform Example for Performing Dynamic Reconfiguration in Reconfiguring M, MFRAC, N and C counters



The following list describes the operation in Figure 2:

a. Avalon-MM writes to the mode register (address = 0x00) to set the Altera PLL Reconfig IP core to operate in polling mode.

b. Avalon-MM writes to the *M* counter register (address = 0x04) to reconfigure the *M* counter to 36.

c. Avalon-MM writes to the *M* counter Fractional Value (K) register (address = 0x07) to reconfigure M_{FRAC} to 0.2665 (decimal value).

d. Avalon-MM writes to the *N* counter register (address = 0x03) to reconfigure the *N* counter to 4.

e. Avalon-MM writes to the C counter register (address=0x05) to reconfigure the C0 counter to 8 (`high_count` =3, `low_count`=3, even division).

f. Avalon-MM writes to the C counter register (address=0x05) to reconfigure the C1 counter to 8 (`high_count` =4, `low_count`=4, even division).

g. Avalon-MM writes to the bandwidth setting register (address=0x08) to reconfigure the bandwidth setting to medium bandwidth.

h. Avalon-MM writes to the charge pump setting register (address=0x09) to reconfigure the charge pump setting to medium bandwidth.

i. Avalon-MM writes to the start register (address = 0x02) to start the reconfiguration.

j. Avalon-MM reads from the status register (address = 0x01) until a value of `1` is read from the status register, indicating a successful reconfiguration.

Fractional PLL Dynamic Phase Shifting in the Quartus II Software

The dynamic phase shifting feature allows the output phases of individual fractional PLL outputs to be dynamically adjusted relative to each other and to the reference clock. The smallest incremental step equals to 1/8th of the VCO period. The output clocks are active during this dynamic phase-shift process.

You can use the following methods to perform dynamic phase shifting:

- the Altera PLL Reconfig IP core
- the dynamic phase-shifting circuitry using the Altera PLL IP core directly.

In the Quartus II software version 11.1 sp2, you can only perform dynamic phase shifting using the Altera PLL Reconfig IP core. However, you can perform dynamic phase shifting directly using the Altera PLL IP core beginning from the Quartus II software version 12.0 onwards.

Performing Dynamic Phase Shifting with the Altera PLL IP Core

You can directly perform dynamic phase shift by enabling the dynamic phase shift ports.

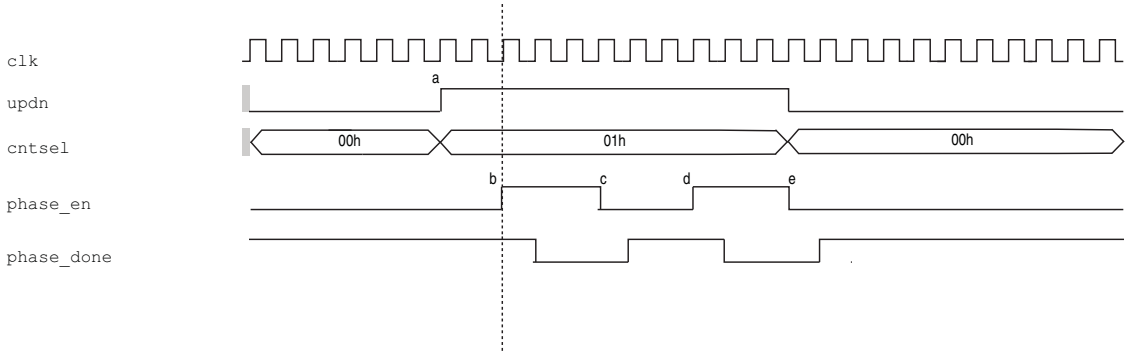
To perform one dynamic phase shift, follow these steps:

1. Set the `updn` and `cntsel` ports.
2. Assert the `phase_en` port for at least two `scanclk` cycles. Each `phase_en` pulse enables one phase shift.
3. Deassert the `phase_en` port after `phase_done` goes low.

The `updn`, `cntsel`, and `phase_en` ports are synchronous to `scanclk`. The `phase_done` signal going low is synchronous to the `scanclk` signal, but it is asynchronous to the `scanclk` signal when going high. Depending on the VCO and `scanclk` frequencies, the low time of the `phase_done` port may be greater than or less than one `scanclk` cycle. Each `phase_en` pulse enables one phase shift. If you want to perform multiple phase shifts, you must assert the `phase_en` signal multiple times. The `phase_en` signal must only be asserted after the `phase_done` signal goes from low to high.

Figure 3 shows the waveform for dynamic phase shift.

Figure 3. Dynamic Phase Shift Waveform



The following list describes the operation in Figure 3:

a. Set the cntsel port to logical counter C1 and the updn port to positive phase shift direction.

b. Assert the phase_en port to begin the first phase shift operation on logical counter C1.

c. Deassert the phase_en port after phase_done goes low.

d. Assert the phase_en port again to begin the second phase shift operation.

e. Deassert the phase_en port after phase_down goes low.

Table 5. Dynamic Phase Shift Signals in the Altera PLL IP Core

Port	Direction	Description
phase_en	Input	Transition from low to high enables dynamic phase shifting, one phase shift per transition from low to high.
scanclk	Input	Free running clock from the core in combination with phase_en to enable and disable dynamic phase shifting.
updn	Input	Selects dynamic phase shift direction; 1 = positive phase shift; 0 = negative phase shift. The PLL registers the signal on the rising edge of scanclk.
cntsel	Input	Logical Counter Select (1). Five bits decoded to select one of the C counters for phase adjustment. The PLL registers the signal on the rising edge of scanclk.
phase_done	Output	When asserted, this port informs the core-logic that the phase adjustment is complete and the PLL is ready to act on a possible next adjustment pulse. Asserts based on internal PLL timing. Deasserts on the rising edge of scanclk.

Note to Table 5:

(1) For the corresponding address of a selected logical counter, refer to Table 6.

Table 6. Logical Counter Location in PLL (Part 1 of 2)

cntsel[4:0]	Selected Logical Counter
5'b00000	Logical counter C0
5b'00001	Logical counter C1
5'b00010	Logical counter C2
5'b00011	Logical counter C3
5'b00100	Logical counter C4
5'b00101	Logical counter C5
5'b00110	Logical counter C6
5'b00111	Logical counter C7
5'b01000	Logical counter C8
5'b01001	Logical counter C9
5'b01010	Logical counter C10
5'b01011	Logical counter C11

Table 6. Logical Counter Location in PLL (Part 2 of 2)

cntsel[4:0]	Selected Logical Counter
5'b01100	Logical counter C12
5'b01101	Logical counter C13
5'b01110	Logical counter C14
5'b01111	Logical counter C15
5'b10000	Logical counter C16
5'b10001	Logical counter C17

Performing Dynamic Phase Shifting with the Altera PLL Reconfig IP Core

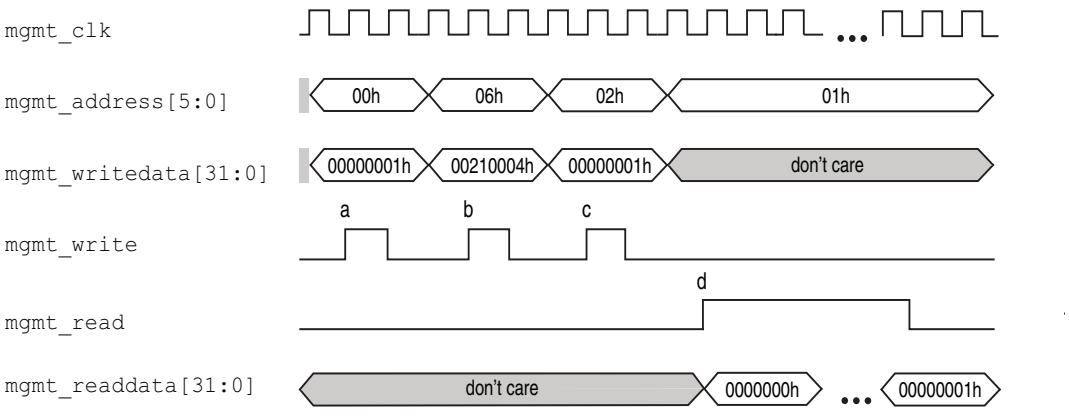
To perform dynamic phase shifting, follow Steps 1-4 in "Reconfiguring Fractional PLL Settings", except in Step 2, you only need to write to the `Dynamic_Phase_Shift` register.

CAUTION

If you assert the `areset` signal to fractional PLL after the dynamic phase shift, you lose all successful phase adjustment with dynamic phase shift in user mode.

Figure 4 shows a waveform example for performing dynamic phase shifting.

Figure 4. Waveform Example for Performing Dynamic Phase Shifting



The following list describes the operations in Figure 4:

a. Avalon writes to the mode register (address=0x00) to set the Altera PLL Reconfig IP core to operate in polling mode.

b. Avalon writes to the dynamic phase shift register (address=0x06) to perform dynamic phase shift on C1 counter for four steps forward.

c. Avalon writes to the start register (address=0x02) to start dynamic phase shifting.

d. Avalon reads from the status register (address=0x01) until a value of 1 has been read from the status register, indicating the dynamic phase shifting is complete.

MIF Streaming

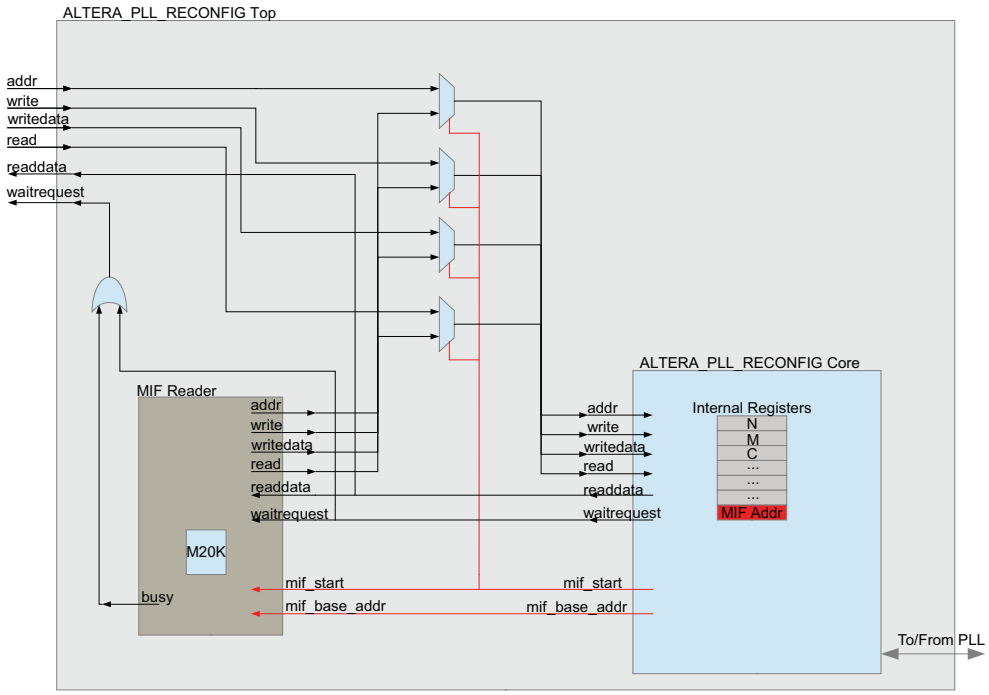
MIF streaming allows you to dynamically reconfigure PLLs through the Altera PLL Reconfig IP core using predefined settings saved in an on-chip RAM.

To start reconfiguration via MIF:

1. Write the base address in the ROM where the **.mif** of the PLL is located. You can have multiple **.mif** files in a ROM. You just need to use a different address location in the ROM for another **.mif** streaming.The following lists the details:
   - Writedata = base address in ROM.
   - Write address = MIF address (011111).
2. Write to the START register to begin.
3. The Altera PLL Reconfig IP core then starts reading the **.mif** for new settings and values, changing the PLL accordingly.
4. The Altera PLL Reconfig IP core generates signals when all changes are done and PLL is locked.

MIF streaming is a subsystem generated in the Altera PLL Reconfig IP core. If you set the **Enable MIF Streaming** option to 0, MIF streaming is not generated and the top level module's ports map directly to the core. If you set the **Enable MIF Streaming** option to 1, the MIF reader is instantiated in the top-level of the Altera PLL Reconfig module as shown in Figure 5:

Figure 5. Altera PLL Reconfig with MIF Streaming Enabled



When you write to the MIF Base Address register and starts the MIF streaming operation, the Altera PLL Reconfig IP core signals the MIF reader to begin the operation. Waitrequest is asserted until the operation is complete.

MIF streaming switches the Altera PLL Reconfig core to waitrequest mode and asserts this signal until the core completed the operation. At this point, MIF streaming restores the previous mode (waitrequest or polling), unless it is explicitly changed by the **.mif**.

MIF File Format

The **.mif** stores the commands that you want to send to the Altera PLL Reconfig IP core. MIF streaming then simulates the commands the user logic would otherwise send to the Altera PLL Reconfig IP core. These commands are address/data pairs in the **.mif** specifying which setting should be reconfigured to the new value. Addresses are stored as opcodes in the lower 6 bits of each entry word.

Table 7. Single MIF file for PLL Reconfiguration

31 5	0	MIF_BASE_ADDR
RSVD	Opcode=SOM	
RSVD	Opcode=M_counter	
DATA M_counter		
RSVD	Opcode=N_counter	
DATA N_counter		
	Opcode=Bandwith	
DATA Bandwidth		
...		
...		
RSVD	Opcode=EOM	

Each **.mif** must be indicated with a Start of MIF (SOM) and End of MIF (EOM):

Table 8.

Opcode Name	Opcode ID	Description
Start of MIF (SOM)	111110	Indicate start of single PLL configuration
End of MIF (EOM)	111111	Indicate end of single PLL configuration

The remaining opcodes are the addresses for each of the registers in the Altera PLL Reconfig IP core.

Multiple PLL configurations can be saved in a **.mif** if appropriately marked by SOM or EOM. The MIF reader reads the settings from an M20K RAM block, which has default address width = 9 bits, data width = 32 bits (total words = 512). These sizes can be changed as parameters are passed to the top-level module; however, data width must be 32 bits to match the Altera PLL Reconfig IP core.

The M20K is initialized by MIF_FILE_NAME, also a top-level parameter. On the start of MIF streaming operation, the MIF reader checks the MIF base address in the M20K RAM block for a SOM opcode. The MIF reader continues to read the file until EOM is reached.

You can generate a **.mif** independently with a PLL MIF configuration file generated by the Altera PLL parameter editor, which stores the entire PLL profile.

You can also construct you own **.mif** files by stitching together existing **.mif** or writing your own commands (following the **.mif** syntax). This allows what settings are stored in the M20K for future reconfiguration.

Design Considerations

You must consider the following information when using fractional PLL reconfiguration:

- Changing prescale and feedback counter settings (*M*, *N*, M_{FRAC}), charge pump/loop filter settings affects the fractional PLL VCO frequency, which might require the fractional PLL to relock to the reference clock.
- Changing the *M* counter phase shift setting changes the phase relationship of the output clocks with respect to the fractional PLL reference clock, which also requires the fractional PLL to relock. Although the exact effect of changing prescale and feedback counter settings (*M*, *N*) depends on the changes to the settings, any changes require relocking.
- Adding phase shift using the *M* counter phase shift setting pulls in all the fractional PLL clock outputs with respect to the reference clock. This effectively adds a negative phase shift because the *M* counter is in the feedback path.
- When making changes to the loop elements (*M*, *N*, M_{FRAC}, M counter phase, I_{cp}, R, C), Altera recommends disabling fractional PLL outputs to the logic array using the `clkena` signals available on the ALTCLKCTRL IP core. This recommendation eliminates the possibility of an over frequency condition affecting system logic while fractional PLL is regaining lock.
- Changing the K counter values is only effective if the PLL is in fractional mode before reconfiguration.
- Changing postscale counters (C) and phase do not affect the fractional PLL lock status or VCO frequency. The resolution of phase shift is a function of VCO frequency, with the smallest incremental step equal to 1/8th of the VCO period.
- Because the phase relationship between output clocks is important, Altera recommends resynchronizing the fractional PLL using the `areset` signal. Using the `areset` signal resets all internal fractional PLL counters and reinitiates the locking process.
- Fractional PLL reconfiguration interface supports a free running `mgmt_clk`, eliminating the need to precisely control the start and stop of `mgmt_clk`.
- Changing the *M* or *N* counter values affect all the output clock frequencies.
- C counters can also be reconfigured individually.
- Can perform phase shifting, even if C counter is set to 1 and bypass is enabled.
- When the PLL has two clocks with 0 degree initial phase shift between the clocks, the Fitter synthesizes away the second clock automatically. To prevent the clocks from merging, Altera recommends manually performing location constraint for each of the PLL output counters which share the same frequency and phase shift.
- Readback counter operation needs at least three `scanclk` cycle latency.

- In waitrequest mode, the `mgmt_waitrequest` signal deasserts when PLL reconfiguration is complete. If the PLL loses lock after reconfiguration is complete, assert the `mgmt_waitrequest` signal again until the PLL locks. There may be a brief period after PLL reconfiguration is complete, but before the PLL has lost lock, when the `mgmt_waitrequest` signal will be de-asserted. Altera recommends allowing sufficient time for the PLL to lock after PLL reconfiguration is complete before performing a new Avalon read or write operation.
- In polling mode, the status register changes from 0 (busy) to 1 (ready) when PLL reconfiguration is complete. If the PLL loses lock after reconfiguration is complete, the status register is at 0 (busy) until the PLL locks. There may be a brief period after PLL reconfiguration is complete, but before the PLL has lost lock, when the status register is at 1 (ready). Altera recommends allowing sufficient time for the PLL to lock after PLL reconfiguration is complete before performing a new Avalon read or write operation.

Using the Design Examples

The following sections describe how to set up and use the design examples:

- "Installing the Design Examples"
- "Software Requirement"
- "Design Examples"

Installing the Design Examples

You can download the design examples in this application note from the Documentation: Application Notes page on the Altera website. For more information about using the design examples, refer to "Design Examples" on page 16.

Software Requirement

You must install the following software in your PC:

- The Quartus II Software, beginning from version 11.1 SP2
- MegaCore IP Library, beginning from version 11.1 (installed with the Quartus II software)
- The Nios® II Embedded Design Suite (EDS), beginning from version 11.1 SP2

Ensure that you extract the Quartus Archived File (**.qar)** of the design example.

This application note assumes that you install the software in the default locations.

Design Examples

The following sections describe the design examples for implementing fractional PLL dynamic reconfiguration and phase shifting using Altera PLL and Altera PLL Reconfig IP cores.

Design Example 1

The design example uses a 5SGXEA7 device. This design example consists of the Altera PLL and Altera PLL Reconfig IP cores. The fractional PLL synthesizes two output clocks of 233.34 MHz, with 0 ps and 107 ps phase shift on C0 and C1 output respectively. The input reference clock to the fractional PLL is 100 MHz.

The Altera PLL Reconfig IP core connects to a state machine to perform the required Avalon write and read operations. A low pulse on the `reset_SM` pin starts the Avalon write and read sequence. After reconfiguration, the fractional PLL operates with the following configuration:

- *M* counter = 36
- M_{FRAC} = 0.2665
- *N* counter = 4
- C0 = 6 (`high_count` = 3, `low_count` = 3, even division)
- C1 = 8 (`high_count` = 4, `low_count` = 4, even division)
- Bandwidth setting = 0110 (for medium bandwidth)
- Charge pump setting = 010 (for medium bandwidth)

To run the test with the design example, perform these steps:

1. Download and restore the **PLL_Reconfig_MNC.qar** file.
2. Regenerate the Altera PLL and Altera PLL Reconfig instances in the design.
3. Change the pin assignment and I/O standard of the design example to match your hardware.
4. Recompile your design. Ensure that your design does not contain any timing violation after recompilation.
5. Open the **.stp** and download the **.sof**.
6. Provide a low pulse on the `reset_SM` input pin to start the reconfiguration. The expected C0 output frequency is 151.11 MHz and the expected C1 output frequency is 113.33 MHz.

Design Example 2

This design example is similar to "Design Example 1", except that this design example demonstrates the dynamic phase shift feature of the fractional PLL with the Altera PLL Reconfig IP core. A low pulse on the `reset_SM` pin starts the Avalon write and read sequence to dynamically phase shift the PLL output. After completing the dynamic phase shifting successfully, the C1 output is phase-shifted for four forward steps.

To run the test with the design example, perform these steps:

1. Download and restore the **PLL_Reconfig_DPS.qar** file.
2. Regenerate the Altera PLL and Altera PLL Reconfig instances in the design.
3. Change the pin assignment and I/O standard of the design example to match your hardware.
4. Recompile your design and ensure your design does not contain any timing violation after recompilation.
5. Open the **.stp** and download the **.sof**.
6. Provide a low pulse on the `reset_SM` input pin to start the reconfiguration.

Design Example 3

The design example for fractional PLL reconfiguration uses the Qsys design flow, targeting the 5SGXEA7 device. The fractional PLL synthesizes four output clock of 106 MHz with 0 ps, 168 ps, 336 ps, and 505 ps on C0, C1, C5, and C10 output respectively. The input frequency of the fractional PLL is 50 MHz. Figure 6 shows the Qsys system and the components for this design example. The C code program in the Nios II processor controls the fractional PLL reconfiguration IP. This program consists of a simple loop that receives and executes command from the JTAG UART.

Table 9 lists the main menu commands from the JTAG UART that the Nios II processor C code program receives and executes.

Table 9. Main Menu Commands

Command	Description
Switch A	This command reconfigures the *M* counter to 26 (`high_count` = 13, `low_count` = 13).
Switch B	This command reconfigures the *N* counter to 4 (`high_count` = 2, `low_count` = 2).
Switch C	This command reconfigures the C0 to 16 with 62.5% duty cycle (`high_count` = 10, `low_count` = 6).
Switch D	This command reconfigures the C1 to 20 with 30% duty cycle (`high_count` = 6, `low_count` = 14).
Switch E	This command reconfigures the K counter to M_{FRAC} = 0.375.
Switch F	This command performs dynamic phase shift on C0 for three steps forward.
Switch G	This command performs dynamic phase shift on C1 for seven steps backward.
Switch H	This command reconfigures the M and M_{FRAC} counters to 68.75, *N* counters to 4 and C0 to 8 (with 37.5% duty cycle).
Switch I	This command enters submenu to readback register bit. For the commands in the submenu, refer to Table 10.

Table 10 lists the submenu commands.

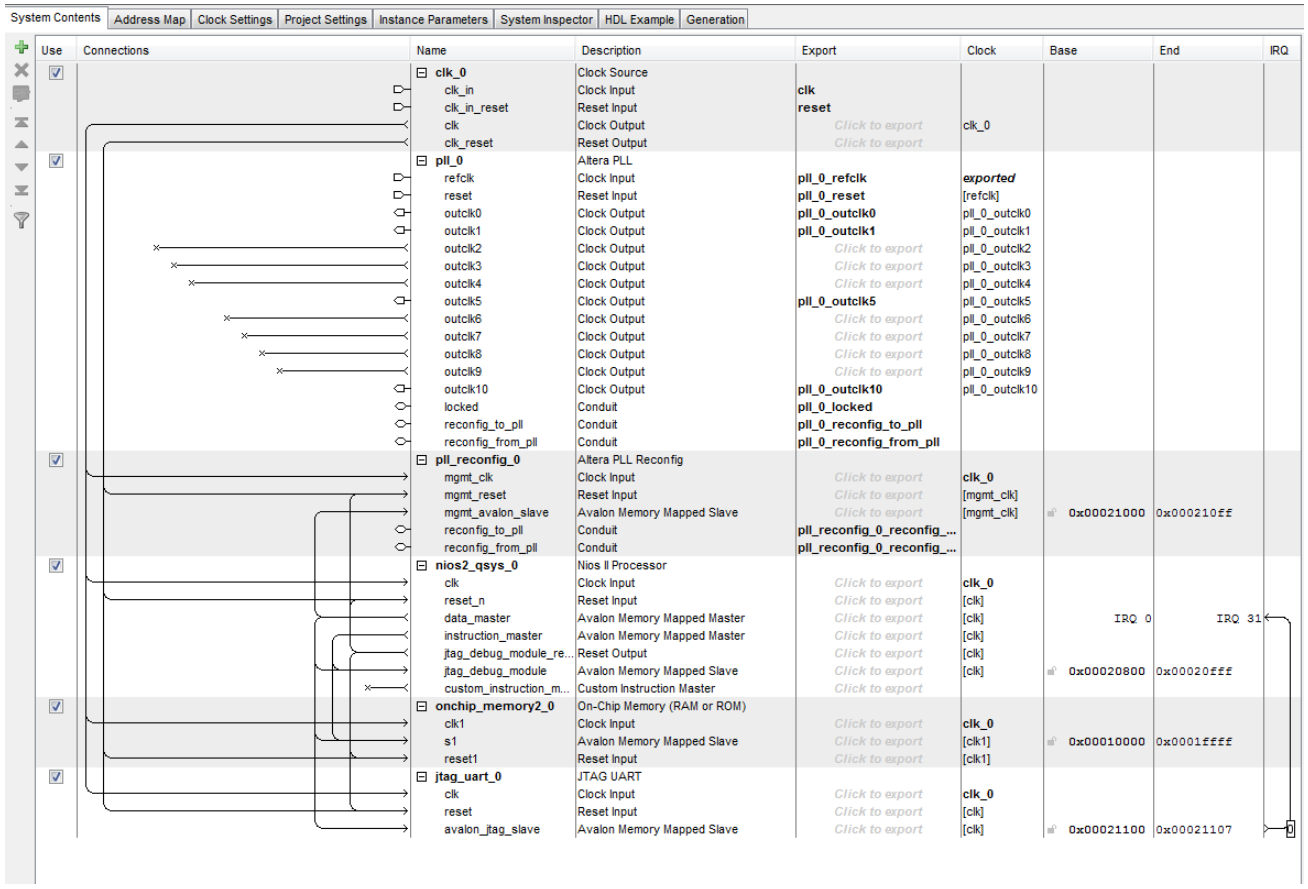
Table 10. Submenu Commands

Command	Description
Switch A	This command readback value in N counter register.
Switch B	This command readback value in M counter register.
Switch C	This command readback value in C0 counter register.
Switch D	This command readback value in C1 counter register.
Switch E	This command readback value in bandwidth setting register.
Switch F	This command readback value in charge pump setting register.
Switch R	This command returns to main menu. For the commands in the submenu, refer to Table 9.

To run the test with the design example, perform these steps (enter submenu):

1. Download and restore the **PLL_Reconfig_Qsys.qar**.
2. Change the pin assignment and I/O standard of the design example to match your hardware.
3. Regenerate the Qsys system in the design example.
4. Recompile your design and ensure your design does not contain any timing violation after recompilation.
5. Open the SignalTap File (**.stp**) and download the **.sof**.
6. Launch the Nios II Software Build Tools (SBT) for Eclipse to set up the Nios II project and compile the test program.
7. Download the **.elf** to run the example test.

Figure 6. Qsys System and Components for the Design Example



Design Example 4

CAUTION This design example is only supported by the Quartus II software version 13.1 onwards due to IP upgrade from physical counter to logical counter.

This design example uses a 5SGXEA7 device. This design example consists of the Altera PLL IP core. The fractional PLL synthesizes two output clocks of 233.34 MHz, with 0 ps and 107 ps phase shift on C0 and C1 output, respectively. The input reference clock to the fractional PLL is 100 MHz.

The Altera PLL IP core connects to a state machine to perform direct dynamic shift operation. A low pulse on the `rest_sm` pin starts the direct dynamic phase shift sequence.

To run the test with the design example, follow these steps:

1. Download and restore **PLL_DynamicPhaseShift.qar**.
2. Regenerate the Altera PLL instances in the design.
3. Change the pin assignment and I/O standard of the design example to match your design.
4. Recompile the design and ensure that the design does not contain any violation after compilation.
5. Open the **.stp** and download the **.sof.**
6. Provide a low pulse on the `reset_sm` input pin to start the dynamic phase shift.

For the walkthrough on creating your design and running the tests, refer the "Tutorial Walkthrough".

Tutorial Walkthrough

This tutorial assumes that you are familiar with the Quartus II software and the Qsys system integration tool.

Creating a New Quartus II Project

In Quartus II software, create a new Quartus II project that targets a Stratix V device.

For more information about creating a new Quartus II project, refer to *Managing Files in a Project* in Quartus II Help.

Ensure that your project path does not include any spaces or extended character

Creating the Qsys System

To create a Qsys system, follow these steps:

1. On the Tool menu, click **Qsys**.
2. To add the Qsys components that your design requires, perform these steps:
   a. To create an Altera PLL instance, perform these steps:
      - Expand **PLL**, select **Altera PLL** and click **Add**.
      - Set **Reference Clock Frequency** to 50.0 MHz.
      - Select the number of output clock and set its output frequency and phase relationship.
      - In the **Settings** tab, turn on the **Enable dynamic reconfiguration of PLL** option.
      - Click **Finish.**
   b. To create an Altera PLL Reconfig instance, perform these steps:
      - Expand **PLL**, select **Altera PLL Reconfig** and click **Add**.
      - Click **Finish**.
   c. Under Component Library, expand **Memories and Memory Controllers** and expand **On-Chip**. Select **On-Chip Memory (RAM or ROM)**, and then click **Add**.
      - For Total Memory size, type `65536` bytes.
      - Click **Finish.**
   d. Under Component Library, expand **Embedded Processors**, select **Nios II Processor** and click **Add**.
      - Select **Nios II/s.**
      - Set **Reset Vector Offset** to **0x00** and **Exception Vector Offset** to **0x20.**
      - Click **Finish.**
   e. Expand **Interface Protocols** and expand **Serial**. Select **JTAG UART** and click **Add**.
      - Under Write FIFO and Read FIFO, for the Buffer depth (bytes) select 64, and for IRQ threshold type `8`.
      - For Prepare interactive windows, select **INTERACTIVE_INPUT_OUTPUT** to open the interactive display window during simulation.
      - Click **Finish.**
   f. In the **Project Setting** tab, for Clock Crossing Adapter Type, select **Auto**.
3. In the **System Contents** tab, make the appropriate bus connection and IRQ lines as shown in Figure 6 on page 19.

If there are warnings about overlapping addresses, on the System menu, click **Assign Base Addresses.** If there are warnings about overlapping IRQ, on the System menu, click **Assign Interrupt numbers**.

4. After setting up the connections, right-click the Nios II Processor and select **Edit** to open the Nios II Processor parameter editor. In the **Core Nios II** tab, for Reset vector memory and Exception vector memory, select **onchip_memory2_0.s1**, and click **Finish**.
5. On the File menu, click **Save**.
6. In the **Generation** tab, under Synthesis, turn on **Create HDL design files for synthesis** and **Create block symbol file (.bsf).**
7. Click **Generate**.

Creating the Top-Level Design File

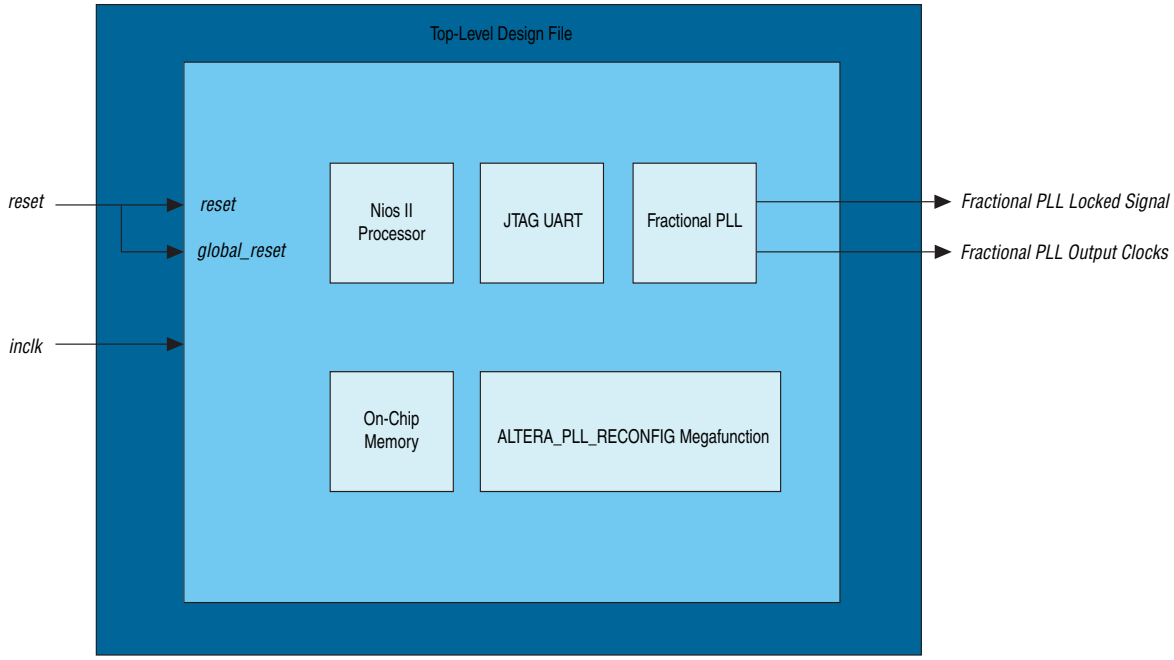
You can consider the Qsys system as a component in your design. The Qsys system can be the only component or one of many components. Hence, when your Qsys system is complete, you must add the system to your top-level design.

The top-level design can be in your preferred HDL language or a **.bdf** schematic design.

In this walkthrough, the top-level design is a simple wrapper file around the Qsys system with no additional components. The top-level design defines only the pin naming convention and port connection.

Figure 7 shows the Qsys top-level block diagram.

Figure 7. Qsys Top-Level Block Diagram



To create a top-level design for your Qsys system with a **.bdf** schematic, follow these steps:

1. In the Quartus II software, on the File menu, click **New**.
2. Select the **Block Diagram/Schematic file** and click **OK**. A blank **.bdf**, **Block1.bdf**, opens.
3. On the File menu, click **Save as**. In the **Save As** dialog box, click **Save**.

The Quartus II software sets the **.bdf** file name to your project name automatically.

4. Right-click in the **blank.bdf**, point to **Insert** and click **Symbol** to open the **Symbol** dialog box.
5. Expand **Project**, under Libraries select system, click **OK**.
6. Add **system.qip** to the project.
7. Connect the `reconfig_to_pll[63:0]` bus on the Altera PLL Reconfig instance to the `reconfig_to_pll[63:0]` bus on the Altera PLL instance.
8. Connect the `reconfig_from_pll[63:0]` bus on the Altera PLL instance to the `reconfig_from_pll[63:0]` bus on the Altera PLL Reconfig instance.
9. Position the Qsys system component and click **Generate pins for Symbol Ports** to automatically add pins and nets to the schematic symbol.
10. Rename the following pins to the modified pin names. Table 11 lists the existing and modified pin names.

Table 11. Rename Pin Name

Existing Pin Name	Modified Pin Name
clk_clk	inclk
reset_reset_n	reset_n
pll_0_reset_reset	areset
pll_0_locked_export	locked
pll_0_outclk0_clk	co_ouput
pll_0_outclk1_clk	c1_ouptut
pll_0_outclk10_clk	c10_ouptut
pll_0_outclk5_clk	c5_output

11. Connect the `pll_refclk_clk` port to the `inclk` pin.
12. On the File menu, click **Save**.
13. On the Project menu, click **Set as Top-Level Entity**.
14. Assign the I/O standard and pin locations for all pins in your design.
15. Add the timing constraint to the **.sdc** to constrain the input clock of your design.
16. Compile your design.

Incorporating the Nios II SBT for Eclipse

You can add a test code to your project Nios II SBT for Eclipse and use this program to run some simple fractional PLL reconfiguration tests.

Adding Test Code to the Nios II SBT for Eclipse

To add a test code to the Nios II SBT for Eclipse, follow these steps:

1. Launch the Nios II SBT for Eclipse.
2. On the File menu, point to Switch Workspace, click **Other**, and select your project directory.
3. On the File menu, point to New and click **Project.**
4. Select **Nios II Application and BSP from Template** and click **Next**.
5. In the **Select Project Template** list, click **Blank Project**, and then locate the SOPC Information File (**.sopcinfo**).
6. Type `pll_reconfig` as project name under the Application project.
7. Click **Next** and click **Finish.**
8. In Window Explorer, drag **PLL_RECONFIG.c** to the **pll_reconfig** directory.

Setting Up the Nios II Project Settings

To set up the Nios II project settings, follow these steps:

1. In the **Project Explorer** tab, right click <*project_name*> and select **Nios II**, and click **BSP Editor** to launch the Nios II BSP Editor.
2. To optimize the footprint size of the library project, in the **Main** tab, point to **Settings** and turn on **enabled_reduced_device_drivers**.
3. To reduce the memory size allocated for the system library, for **Max file descriptors**, type `4`.
4. In the **Linker Script** tab, select **onchip_memory2_0** and the linker region name for **.bss**, **.heap**, **.rodata**, **.rwdata**, **.stack**, and **.text**.
5. Click **Generate**.

Verifying Design on Hardware

To verify your design on hardware, you must compile your project, download the object file, and then verify your design with the Nios II SBT for Eclipse.

Compiling the Project

To compile your project, follow these steps:

1. Add SignalTap II Logic Analyzer to your design. The SignalTap II Logic Analyzer shows read and write activity in the system.
2. After adding signals to the SignalTap II Logic Analyzer, you can recompile your design. To recompile your design, on the Processing menu, click **Start Compilation**.
3. After compilation, ensure that the TimeQuest timing analysis passes successfully.
4. Connect your hardware to your computer.

Downloading the Object File

To download the object file, perform these steps:

1. On the Tools menu, click **Signal Tap II Logic Analyzer**. The SignalTap II dialog box appears.

   The SRAM Object File (SOF) Manager contains the <*your_project_name*>**.sof**.

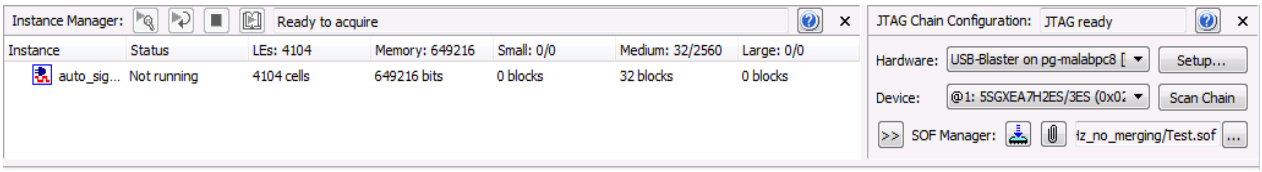
2. Click **…** to open the **Select Program Files** dialog box.
3. Select <*your_project_name*>**.sof**.
4. Click **Open**.
5. To download the file, click **Program Device**.

Verifying Design with the Nios II SBT for Eclipse

Verify your design with the Nios II SBT for Eclipse. To do so, perform these steps:

1. Right-click on <*project*>, point to **Run As**, and click **Nios II Hardware** for the Nios II C/C++ Eclipse to compile the example test program.
2. Type in your selection to test your system.

Figure 8. Install the SRAM Object File in the SignalTap II Dialog Box



Document Revision History

Table 12 lists the revision history for this application note.

Table 12. Document Revision History

Date	Version	Changes
August 2014	3.1	■ Changed `C_counter[18:22]` to `C_counter[22:18]` in Table 2 on page 4.
November 2013	3.0	■ Added a note to Table 2 on page 4 to clarify that the bypass enable bit, even division bit, and odd division bit of the M, N, C counters support write operation only. ■ Updated Table 2 on page 4 to include VCO DIV setting information. ■ Added "MIF Streaming" on page 12. ■ Changed ALTERA_PLL to Altera PLL as per the naming in the GUI. ■ Changed ALTERA_PLL_RECONFIG to Altera PLL Reconfig as per the naming in the GUI. ■ Added a caution note to "Design Example 4" on page 20 to notify users that the design example is only supported by the Quartus II software version 13.1 onwards, due to IP upgrade from physical counter to logical counter. ■ Updated "MIF File Format" on page 13 to inform users that they can also construct their own **.mif** files. ■ Updated Table 2 on page 4 to include MIF Base Address information. ■ Updated Figure 3 on page 9. ■ Updated mgmt_waitrequest information in Table 1 on page 3. ■ Removed Figure 4 on page 11, Figure 5 on page 12, Figure 6 on page 13, Figure 7 on page 14, Figure 8 on page 15, Figure 9 on page 16, and Figure 10 on page 17. ■ Updated "Performing Dynamic Phase Shifting with the Altera PLL IP Core" on page 9 section due to IP upgrade from physical counter to logical counter.
October 2012	2.0	■ Updated "Fractional PLL Reconfiguration in 28-nm Devices" on page 1. ■ Updated Table 1 on page 3, Table 2 on page 4. ■ Added Table 3 on page 6 and Table 10 on page 19. ■ Added Figure 4 on page 11, Figure 5 on page 12, Figure 6 on page 13, Figure 7 on page 14, Figure 8 on page 15, Figure 9 on page 16, Figure 10 on page 17. ■ Added information about performing dynamic phase shifting using the ALTERA_PLL IP core in "Performing Dynamic Phase Shifting with the Altera PLL IP core" on page 9 ■ Added new design example and added "Design Example 4" on page 21 for reference.
May 2012	1.1	Updated "Design Considerations" on page 18.
February 2012	1.0	Initial release.