You can use the 28-nm devices (Arria® V, Cyclone® V, and Stratix® V device families) to implement fractional phase-locked loop (PLL) reconfiguration and dynamic phase shift for fractional PLLs with the Altera PLL and Altera PLL Reconfig IP cores in the Intel® Quartus® Prime software.

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 (fPLL) 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 fPLL components to update the clock frequency, fPLL bandwidth, and phase shift in real time, without reconfiguring the entire FPGA.

Related Information
- Design Example 1: PLL Reconfiguration with Altera PLL Reconfig IP Core to Reconfigure M, N, and C Counters
- Design Example 2: PLL Reconfiguration with Altera PLL Reconfig IP Core to Perform Dynamic Phase Shift
- Design Example 3: PLL Reconfiguration with Altera PLL Reconfig IP Core using Qsys Design Flow
- Design Example 4: Dynamic Phase Shift with Altera PLL IP Core
- Design Example 5: .mif Streaming Reconfiguration
- PLL Reconfiguration Calculator
- .tcl Script to Stitch Two .mif Files

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

The fPLLs in 28-nm devices support dynamic reconfiguration. While the device is in user mode, you can download a new fPLL configuration in real time without reconfiguring the entire FPGA.
The following fPLL components are reconfigurable in real time using the dynamic reconfiguration IP core:

- Post-scale output counter (C)
- Feedback counter (M)
- Prescale counter (N)
- Charge pump current (I_{CP}), and loop-filter components (R, C)

**Note:** The Intel Quartus Prime software version 12.0 and later support 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 fPLL reconfiguration in real time. fPLL reconfiguration is also beneficial in prototyping environments, allowing you to sweep fPLL 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 Intel Quartus Prime Software

You can use the Altera PLL Reconfig IP core to enable reconfiguration circuitry in the 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.

### Connectivity between Altera PLL and Altera PLL Reconfig IP Cores

**Figure 1:** Connectivity between Altera PLL and Altera PLL Reconfig IP Cores in the Intel Quartus Prime Software
Connecting Altera PLL and Altera PLL Reconfig IP Cores

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


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.

Avalon-MM Signals in Altera PLL Reconfig IP Core

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: Avalon-MM Signals in Altera PLL Reconfig IP Core

<table>
<thead>
<tr>
<th>Port</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mgmt_read_data[31:0]</code></td>
<td>Output</td>
<td>Data read from the Altera PLL Reconfig IP core when you assert the <code>mgmt_read</code> signal.</td>
</tr>
<tr>
<td><code>mgmt_write_data[31:0]</code></td>
<td>Input</td>
<td>Data written to the Altera PLL Reconfig IP core when you assert the <code>mgmt_write</code> signal.</td>
</tr>
<tr>
<td><code>mgmt_address[5:0]</code></td>
<td>Input</td>
<td>Specifies the address of the memory mapped register for a read or write operation.</td>
</tr>
<tr>
<td><code>mgmt_read</code></td>
<td>Input</td>
<td>Active high signal. Asserted to indicate a read operation. When present, read data is available on the <code>mgmt_read_data</code> bus.</td>
</tr>
<tr>
<td><code>mgmt_write</code></td>
<td>Input</td>
<td>Active high signal. Asserted to indicate a write operation. When present, the <code>mgmt_write_data</code> bus requires write data.</td>
</tr>
<tr>
<td><code>mgmt_reset</code></td>
<td>Input</td>
<td>Active-high signal that resets all PLL settings to their initial .sof file values.</td>
</tr>
<tr>
<td><code>mgmt_waitrequest</code></td>
<td>Output</td>
<td>Active high signal. When the Altera PLL Reconfig IP core asserts this signal, the IP core ignores read or write operations.</td>
</tr>
</tbody>
</table>
# Registers and Counters Settings

## Fractional PLL Dynamic Reconfiguration Registers and Settings

### Table 2: Fractional PLL Dynamic Reconfiguration Registers and Settings

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Register Size</th>
<th>Address (Binary)</th>
<th>Counter Bit Setting</th>
<th>Read/Write</th>
</tr>
</thead>
</table>
| 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\(_{\text{counter}}[7:0]\) = low\(_{\text{count}}\)  
- N\(_{\text{counter}}[15:8]\) = high\(_{\text{count}}\)  
- Total\(_{\text{div}}\) = high\(_{\text{count}}\) + low\(_{\text{count}}\)  
- N\(_{\text{counter}}[16]\) = bypass enable \(^{(1)}\)  
  - N\(_{\text{counter}}[16]\) = 0, \(f_{\text{REF}} = f_{\text{IN}}/\text{Total\(_{\text{div}}\)}\)  
  - N\(_{\text{counter}}[16]\) = 1, \(f_{\text{REF}} = f_{\text{IN}} (\text{N counter is bypassed})\)  
- N\(_{\text{counter}}[17]\) = odd division \(^{(1)}\)  
  - N\(_{\text{counter}}[17]\) = 0, even division, duty cycle = high\(_{\text{count}}\)/Total\(_{\text{div}}\)  
  - N\(_{\text{counter}}[17]\) = 1, odd division, duty cycle = (high\(_{\text{count}}\) – 0.5)/Total\(_{\text{div}}\) | Read/Write |
| M Counter      | 18            | 000100           | - M\(_{\text{counter}}[7:0]\) = low\(_{\text{count}}\)  
- M\(_{\text{counter}}[15:8]\) = high\(_{\text{count}}\)  
- Total\(_{\text{div}}\) = high\(_{\text{count}}\) + low\(_{\text{count}}\)  
- M\(_{\text{counter}}[16]\) = bypass enable \(^{(1)}\)  
  - M\(_{\text{counter}}[16]\) = 0, \(f_{\text{FB}} = f_{\text{VCO}}/\text{Total\(_{\text{div}}\)}\)  
  - M\(_{\text{counter}}[16]\) = 1, \(f_{\text{FB}} = f_{\text{VCO}} (\text{M counter is bypassed})\)  
- M\(_{\text{counter}}[17]\) = odd division \(^{(1)}\)  
  - M\(_{\text{counter}}[17]\) = 0, even division, duty cycle = high\(_{\text{count}}\)/Total\(_{\text{div}}\)  
  - M\(_{\text{counter}}[17]\) = 1, odd division, duty cycle = (high\(_{\text{count}}\) – 0.5)/Total\(_{\text{div}}\) | Read/Write |

\(^{(1)}\) The bypass enable bit, even division bit, and odd division bit of the \(M, N,\) and \(C\) counters support write operation only.
<table>
<thead>
<tr>
<th>Register Name</th>
<th>Register Size</th>
<th>Address (Binary)</th>
<th>Counter Bit Setting</th>
<th>Read/Write</th>
</tr>
</thead>
</table>
| 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  
• 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  
• 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 |
| 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 the Dynamic Phase Shift Counter and cnt_select (Dynamic_Phase_Shift[20:16]) Bit Setting table.  
• 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      |

(2) For c counter read operation, use the address for the selected counter in the Counter Address and Bit Setting During Read Operation table.
### Fractional PLL Dynamic Reconfiguration Registers and Settings

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Register Size</th>
<th>Address (Binary)</th>
<th>Counter Bit Setting</th>
<th>Read/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>M Counter Fractional Value (K)</td>
<td>32</td>
<td>000111</td>
<td>Fractional part of the M counter (for DSM). The actual fractional values are:</td>
<td>Write</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• $M_{FRAC} = \frac{K[(X-1):0]}{2^X}$ ($X = 8, 16, 24, \text{ or } 32$)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• M counter final value = Total_div for M Counter + $M_{FRAC}$</td>
<td></td>
</tr>
<tr>
<td>Bandwidth Setting</td>
<td>4</td>
<td>001000</td>
<td>For the bandwidth settings, refer to the PLL Reconfiguration Calculator.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Charge Pump Setting</td>
<td>3</td>
<td>001001</td>
<td>For the charge pump settings, refer to the PLL Reconfiguration Calculator.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>VCO Post Divide Counter Setting</td>
<td>1</td>
<td>011100</td>
<td>Enable or disable /2 divider for VCO to keep VCO frequency in operating range.</td>
<td>Read/Write</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 0: VCO DIV = 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1: VCO DIV = 1</td>
<td></td>
</tr>
<tr>
<td>.mif Base Address</td>
<td>9</td>
<td>011111</td>
<td>Base address for the start of a PLL profile in a Memory Initialization File (.mif).</td>
<td>Write</td>
</tr>
</tbody>
</table>

**Related Information**

- **Counter Address and Bit Setting During Read Operation** on page 7
  Provides the address for the selected C counter.
- **Dynamic Phase Shift Counter and cnt_select (Dynamic_Phase_Shift[20:16]) Bit Setting** on page 7
  Provides the cnt_select and the corresponding dynamic phase shift counter.
- **PLL Reconfiguration Calculator**

---

(3) 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.
Counter Address and Bit Setting During Read Operation

Table 3: Counter Address and Bit Setting During Read Operation

<table>
<thead>
<tr>
<th>Counter Name</th>
<th>Address (Binary)</th>
<th>Counter Bit Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>001010</td>
<td></td>
</tr>
<tr>
<td>C1</td>
<td>001011</td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>001100</td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td>001101</td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td>001110</td>
<td></td>
</tr>
<tr>
<td>C5</td>
<td>001111</td>
<td></td>
</tr>
<tr>
<td>C6</td>
<td>010000</td>
<td></td>
</tr>
<tr>
<td>C7</td>
<td>010001</td>
<td></td>
</tr>
<tr>
<td>C8</td>
<td>010010</td>
<td></td>
</tr>
<tr>
<td>C9</td>
<td>010011</td>
<td></td>
</tr>
<tr>
<td>C10</td>
<td>010100</td>
<td></td>
</tr>
<tr>
<td>C11</td>
<td>010101</td>
<td></td>
</tr>
<tr>
<td>C12</td>
<td>010110</td>
<td></td>
</tr>
<tr>
<td>C13</td>
<td>010111</td>
<td></td>
</tr>
<tr>
<td>C14</td>
<td>011000</td>
<td></td>
</tr>
<tr>
<td>C15</td>
<td>011001</td>
<td></td>
</tr>
<tr>
<td>C16</td>
<td>011010</td>
<td></td>
</tr>
<tr>
<td>C17</td>
<td>011011</td>
<td></td>
</tr>
</tbody>
</table>

- $C_{counter}[7:0] = low\_count$
- $C_{counter}[15:8] = high\_count$
- $Total\_div = high\_count + low\_count$

Dynamic Phase Shift Counter and cnt_select (Dynamic_Phase_Shift[20:16]) Bit Setting

Table 4: Dynamic Phase Shift Counter and cnt_select (Dynamic_Phase_Shift[20:16]) Bit Setting

<table>
<thead>
<tr>
<th>Dynamic Phase Shift Counter</th>
<th>Dynamic_Phase_Shift[20:16] Bit Setting</th>
<th>Device Family</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>5'b000000</td>
<td>Arria V, Cyclone V, and Stratix V</td>
</tr>
<tr>
<td>C1</td>
<td>5'b000001</td>
<td>Arria V, Cyclone V, and Stratix V</td>
</tr>
<tr>
<td>C2</td>
<td>5'b000100</td>
<td>Arria V, Cyclone V, and Stratix V</td>
</tr>
<tr>
<td>C3</td>
<td>5'b000111</td>
<td>Arria V, Cyclone V, and Stratix V</td>
</tr>
<tr>
<td>C4</td>
<td>5'b001000</td>
<td>Arria V, Cyclone V, and Stratix V</td>
</tr>
<tr>
<td>C5</td>
<td>5'b001010</td>
<td>Arria V, Cyclone V, and Stratix V</td>
</tr>
<tr>
<td>C6</td>
<td>5'b001110</td>
<td>Arria V, Cyclone V, and Stratix V</td>
</tr>
<tr>
<td>C7</td>
<td>5'b001111</td>
<td>Arria V, Cyclone V, and Stratix V</td>
</tr>
</tbody>
</table>
### Reconfiguring Fractional PLL Settings with Avalon-MM Interface

You can dynamically reconfigure fPLLs 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 Fractional PLL Dynamic Reconfiguration Registers and Settings, and Dynamic Phase Shift Counter and \( \text{cnt}_\text{select} \) (Dynamic Phase Shift\( \text{[20:16]} \)) Bit Setting tables.

3. **Repeat Step 2 for all the reconfigurable elements (\( N, M, C \) counters, \( M_{\text{FRAC}} \) 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:

   - 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.
   
   - 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.

<table>
<thead>
<tr>
<th>Dynamic Phase Shift Counter</th>
<th>Dynamic Phase Shift[20:16] Bit Setting</th>
<th>Device Family</th>
</tr>
</thead>
<tbody>
<tr>
<td>C8</td>
<td>5'b01000</td>
<td>Arria V, Cyclone V, and Stratix V</td>
</tr>
<tr>
<td>C9</td>
<td>5'b01001</td>
<td>Arria V and Stratix V</td>
</tr>
<tr>
<td>C10</td>
<td>5'b01010</td>
<td>Arria V and Stratix V</td>
</tr>
<tr>
<td>C11</td>
<td>5'b01011</td>
<td>Arria V and Stratix V</td>
</tr>
<tr>
<td>C12</td>
<td>5'b01100</td>
<td>Arria V and Stratix V</td>
</tr>
<tr>
<td>C13</td>
<td>5'b01101</td>
<td>Arria V and Stratix V</td>
</tr>
<tr>
<td>C14</td>
<td>5'b01110</td>
<td>Arria V and Stratix V</td>
</tr>
<tr>
<td>C15</td>
<td>5'b01111</td>
<td>Arria V and Stratix V</td>
</tr>
<tr>
<td>C16</td>
<td>5'b10000</td>
<td>Arria V and Stratix V</td>
</tr>
<tr>
<td>C17</td>
<td>5'b10001</td>
<td>Arria V and Stratix V</td>
</tr>
<tr>
<td>All C counters</td>
<td>5'b11111</td>
<td>Arria V, Cyclone V, and Stratix V</td>
</tr>
<tr>
<td>M counter</td>
<td>5'b10010</td>
<td>Arria V, Cyclone V, and Stratix V</td>
</tr>
</tbody>
</table>
Related Information

- Fractional PLL Dynamic Reconfiguration Registers and Settings on page 4
- Dynamic Phase Shift Counter and cnt_select (Dynamic_Phase_Shift[20:16]) Bit Setting on page 7

Waveform Example for Dynamic Reconfiguration with Avalon-MM Interface

Figure 2: Waveform Example for Performing Dynamic Reconfiguration in Reconfiguring $M_{\text{FRAC}}$, $M$, $N$, and $C$ Counters

The operation of the waveform example is as follows:

1. Avalon-MM writes to the mode register (address = 0x00) to set the Altera PLL Reconfig IP core to operate in polling mode.
2. Avalon-MM writes to the $M$ counter register (address = 0x04) to reconfigure the $M$ counter to 36.
3. Avalon-MM writes to the $M$ counter Fractional Value ($K$) register (address = 0x07) to reconfigure $M_{\text{FRAC}}$ to 0.2666667 (decimal value).
4. Avalon-MM writes to the $N$ counter register (address = 0x03) to reconfigure the $N$ counter to 4.
5. Avalon-MM writes to the $C$ counter register (address=0x05) to reconfigure the $C_0$ counter to 6 ($\text{high\_count}=3, \text{low\_count}=3, \text{even division}$).
6. Avalon-MM writes to the $C$ counter register (address=0x05) to reconfigure the $C_1$ counter to 8 ($\text{high\_count}=4, \text{low\_count}=4, \text{even division}$).
7. Avalon-MM writes to the bandwidth setting register (address=0x08) to reconfigure the bandwidth setting to medium bandwidth.
8. Avalon-MM writes to the charge pump setting register (address=0x09) to reconfigure the charge pump setting to medium bandwidth.
9. Avalon-MM writes to the start register (address = 0x02) to start the reconfiguration.
10. Avalon-MM reads from the status register (address = 0x01) until a value of 1 is read from the status register, indicating a successful reconfiguration.

.mif Streaming Reconfiguration

.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 streaming, follow these steps:

1. Write the base address in the ROM where the .mif file 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 as follows:
1. 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 file 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.

**Stitching Two .mif Files Generated by the Intel Quartus Prime Software**

To stitch two .mif files generated by the Intel Quartus Prime software with a .tcl script, follow these steps:

1. Source the `merge_mif.tcl` in `.tcl` console.
2. Type the following command to stitch two .mif files (for example A.mif and B.mif) into `output.mif`.
   \[
   \text{Command:} \ \text{stitch A.mif B.mif [output.mif]}
   \]
   If you do not specify the optional parameter, `[output.mif]`, the .tcl script defaults to `merged.mif`. In the `output.mif`, the content of B.mif comes after A.mif.
.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 Altera PLL Reconfig IP 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.

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** signal is asserted until the operation is complete.

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

The .mif file 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 file specifying the setting to be reconfigured to the new value. Addresses are stored as operation codes (opcodes) in the lower six bits of each entry word.

Figure 4: Single .mif File Format for .mif Streaming Reconfiguration

<table>
<thead>
<tr>
<th>31</th>
<th>5</th>
<th>0</th>
<th>MIF_BASE_ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSVD</td>
<td>Opcode = SOM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSVD</td>
<td>Opcode = M_counter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA M_counter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSVD</td>
<td>Opcode = N_counter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA N_counter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSVD</td>
<td>Opcode = Bandwidth</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA Bandwidth</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• • •</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• • •</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSVD</td>
<td>Opcode = EOM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

.mif Streaming Reconfiguration Operation Codes

Table 5: .mif Streaming Reconfiguration Operation Codes

<table>
<thead>
<tr>
<th>Operation Name</th>
<th>Operation Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start of .mif (SOM)</td>
<td>111110</td>
<td>Indicates start of single PLL configuration</td>
</tr>
<tr>
<td>End of .mif (EOM)</td>
<td>111111</td>
<td>Indicates end of single PLL configuration</td>
</tr>
</tbody>
</table>

Each .mif streaming reconfiguration must be indicated with a SOM and EOM opcodes. The remaining opcodes are the addresses for each of the registers in the Altera PLL Reconfig IP core.

You can save multiple PLL configurations in a .mif file if you mark the SOM or EOM opcodes appropriately. 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 change as parameters are passed to the top-level module. For .mif streaming reconfiguration, 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 opcode is reached.

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

You can also construct your own .mif files by stitching together existing .mif files or writing your own commands (following the .mif syntax). This allows what settings are stored in the M20K for future reconfiguration.
Fractional PLL Dynamic Phase Shifting in the Intel Quartus Prime 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:

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

**Note:** In the Intel Quartus Prime 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 in the Intel Quartus Prime software version 12.0 and later.

Performing Dynamic Phase Shifting with 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. You can only assert the phase_en signal after the phase_done signal goes from low to high.

Waveform Example for Dynamic Phase Shift with Altera PLL IP Core

**Figure 5: Waveform Example for Dynamic Phase Shift with Altera PLL IP Core**

![Waveform Example](image)
The operation of the waveform example is as follows:

1. Set the cntsel port to logical counter C1 and the updn port to positive phase shift direction.
2. Assert the phase_en port to begin the first phase shift operation on logical counter C1.
3. Deassert the phase_en port after phase_done goes low.
4. Assert the phase_en port again to begin the second phase shift operation.
5. Deassert the phase_en port after phase_done goes low.

Dynamic Phase Shift Signals in Altera PLL IP Core

Table 6: Dynamic Phase Shift Signals in Altera PLL IP Core

<table>
<thead>
<tr>
<th>Port</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>phase_en</td>
<td>Input</td>
<td>Transition from low to high enables dynamic phase shifting, one phase shift per transition from low to high.</td>
</tr>
<tr>
<td>scanclk</td>
<td>Input</td>
<td>Free running clock from the core in combination with phase_en to enable and disable dynamic phase shifting.</td>
</tr>
<tr>
<td>updn</td>
<td>Input</td>
<td>Selects dynamic phase shift direction; 1 = positive phase shift; 0 = negative phase shift. The PLL registers the signal on the rising edge of scanclk.</td>
</tr>
<tr>
<td>cntsel</td>
<td>Input</td>
<td>Logical Counter Select(^{(4)(5)}). Five bits decoded to select one of the C counters for phase adjustment. The PLL registers the signal on the rising edge of scanclk.</td>
</tr>
<tr>
<td>phase_done</td>
<td>Output</td>
<td>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.</td>
</tr>
</tbody>
</table>

Related Information

Logical Counter Bit Setting on page 14

Logical Counter Bit Setting

Table 7: Logical Counter Bit Setting

<table>
<thead>
<tr>
<th>Logical Counter</th>
<th>cntsel[4:0] Bit Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical counter C0</td>
<td>5'b000000</td>
</tr>
<tr>
<td>Logical counter C1</td>
<td>5'b00001</td>
</tr>
<tr>
<td>Logical counter C2</td>
<td>5'b00010</td>
</tr>
<tr>
<td>Logical counter C3</td>
<td>5'b00011</td>
</tr>
<tr>
<td>Logical counter C4</td>
<td>5'b00100</td>
</tr>
</tbody>
</table>

\(^{(4)}\) For the corresponding address of a selected logical counter, refer to the Logical Counter Bit Setting table.

\(^{(5)}\) For the Intel Quartus Prime software versions prior to 13.1, cntsel refers to physical counter. For the Intel Quartus Prime software version 13.1 and later, cntsel refers to logical counter. Refer to the Logical Counter Bit Setting table for the cntsel bit setting for both physical counter and logical counter.
Performing Dynamic Phase Shifting with Altera PLL Reconfig IP Core

To perform dynamic phase shifting, follow steps 1-4 in “Reconfiguring Fractional PLL Settings with Avalon-MM Interface”, except in step 2, you only need to write to the Dynamic_Phase_Shift register.

**Caution:** If you assert the areset signal to fPLL after the dynamic phase shift, you lose all successful phase adjustment with dynamic phase shift in user mode.

Related Information
Reconfiguring Fractional PLL Settings with Avalon-MM Interface on page 8

Waveform Example for Dynamic Phase Shift with Altera PLL Reconfig IP Core

Figure 6: Waveform Example for Dynamic Phase Shift with Altera PLL Reconfig IP Core
The operation of the waveform example is as follows:

1. Avalon writes to the mode register (address=0x00) to set the Altera PLL Reconfig IP core to operate in polling mode.
2. Avalon writes to the dynamic phase shift register (address=0x06) to perform dynamic phase shift on \( C_1 \) counter for four steps forward.
3. Avalon writes to the start register (address=0x02) to start dynamic phase shifting.
4. 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.

**Design Considerations**

You must consider the following information when using fPLL reconfiguration:

- Changing prescale and feedback counter settings (\( M, N, M_{FRAC} \)), charge pump/loop filter settings affects the fractional PLL VCO frequency, which may 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 post-scale 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.
- Altera recommends resynchronizing the fractional PLL using the \( areset \) signal if the phase relationship between output clocks is important. Always assert the \( areset \) signal after each \( mgmt\_reset \) operation or after each fPLL reconfiguration process, to reinitiate the fPLL locking process.
- Fractional PLL reconfiguration interface supports a free running \( mgmt\_clk \) signal, eliminating the need to precisely control the start and stop of \( mgmt\_clk \) signal.
- Changing the \( M \) or \( N \) counter values affect all the output clock frequencies.
- \( C \) counters can also be reconfigured individually.
- You 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 is 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 the software requirements and the use of the design examples.

Software Requirement

You must install the following software in your PC:

- The Intel Quartus Prime software version 11.1 SP2 or later
- MegaCore IP Library version 11.1 or later (installed with the Intel Quartus Prime software)
- The Nios II Embedded Design Suite (EDS) version 11.1 SP2 or later

Note:
- Ensure that you extract the Intel Quartus Prime Archive File (.qar) of the design example.
- This application note assumes that you install the software in the default locations.

Design Example 1: PLL Reconfiguration with Altera PLL Reconfig IP Core to Reconfigure M, N, and C Counters

The design example uses a 5SGXEA7 device. This design example consists of the Altera PLL and Altera PLL Reconfig IP cores. The fPLL 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 fPLL 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 fPLL 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 SignalTap™ II File (.stp) and download the SRAM Object File (.sof).
6. Provide a low pulse on the `reset_SM` input pin to start the reconfiguration.

The expected $C_0$ output frequency is 151.11 MHz and the expected $C_1$ output frequency is 113.33 MHz.

**Design Example 2: PLL Reconfiguration with Altera PLL Reconfig IP Core to Perform Dynamic Phase Shift**

This design example is similar to “Design Example 1”, except that this design example demonstrates the dynamic phase shift feature of the fPLL 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 $C_1$ 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` file and download the `.sof` file.
6. Provide a low pulse on the `reset_SM` input pin to start the reconfiguration.

**Related Information**

[Design Example 1: PLL Reconfiguration with Altera PLL Reconfig IP Core to Reconfigure M, N, and C Counters](#) on page 17

**Design Example 3: PLL Reconfiguration with Altera PLL Reconfig IP Core using Qsys Design Flow**

The design example for fPLL reconfiguration uses the Qsys design flow, targeting the 5SGXEA7 device. The fPLL synthesizes four output clock of 106 MHz with 0 ps, 168 ps, 336 ps, and 505 ps on $C_0$, $C_1$, $C_5$, and $C_{10}$ output respectively. The input frequency of the fPLL is 50 MHz.

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

1. Download and restore the `pll_reconfig_qsys.qar` file.
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 .stp file and download the .sof file.
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 Executable and Linking Format File (.elf) to run the example test.

Qsys System and Components

Figure 7: Qsys System and Components for Design Example 3

The C code program in the Nios II processor controls the fPLL reconfiguration IP core. This program consists of a simple loop that receives and executes command from the JTAG UART.

Main Menu Commands in Qsys

Table 8: Main Menu Commands in Qsys

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch A</td>
<td>This command reconfigures the M counter to 26 (high_count = 13, low_count = 13).</td>
</tr>
<tr>
<td>Switch B</td>
<td>This command reconfigures the N counter to 4 (high_count = 2, low_count = 2).</td>
</tr>
<tr>
<td>Switch C</td>
<td>This command reconfigures the C0 to 16 with 62.5% duty cycle (high_count = 10, low_count = 6).</td>
</tr>
</tbody>
</table>
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 N counter and M_FRAC value 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 the Submenu Commands in Qsys table.

Related Information
Submenu Commands in Qsys on page 20

Submenu Commands in Qsys

Table 9: Submenu Commands in Qsys

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch A</td>
<td>This command readback value in N counter register.</td>
</tr>
<tr>
<td>Switch B</td>
<td>This command readback value in M counter register.</td>
</tr>
<tr>
<td>Switch C</td>
<td>This command readback value in C0 counter register.</td>
</tr>
<tr>
<td>Switch D</td>
<td>This command readback value in C1 counter register.</td>
</tr>
<tr>
<td>Switch E</td>
<td>This command readback value in bandwidth setting register.</td>
</tr>
<tr>
<td>Switch F</td>
<td>This command readback value in charge pump setting register.</td>
</tr>
<tr>
<td>Switch R</td>
<td>This command returns to main menu. For the commands in the main menu, refer to Main Menu Commands in Qsys table.</td>
</tr>
</tbody>
</table>

Related Information
Main Menu Commands in Qsys on page 19

Design Example 4: Dynamic Phase Shift with Altera PLL IP Core

Caution: This design example is only supported by the Intel Quartus Prime software version 13.1 and later 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 fPLL synthesizes two output clocks of 233.34 MHz, with 0 ps and 107 ps phase shift on C0 and C1 outputs, respectively. The input reference clock to the fPLL 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` file.
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` file and download the `.sof` file.
6. Provide a low pulse on the `reset_sm` input pin to start the dynamic phase shift.

**Design Example 5: .mif Streaming Reconfiguration**

This design example is similar to “Design Example 1”, except that this design example demonstrates the .mif streaming reconfiguration of the fPLL with the Altera PLL Reconfig IP Core. This design example consists of the Altera PLL and Altera PLL Reconfig IP cores. The fPLL synthesizes two output clocks of 200.0 MHz, with 0 degree and 7.5 degree phase shift on C0 and C1 output respectively. The input reference clock to the fPLL is 100 MHz.

A low pulse on the `reset_SM` pin starts the Avalon write operation to enable the .mif streaming reconfiguration. After completing the .mif streaming reconfiguration, the C0 and C1 output frequencies are changed to 100 MHz and 300 MHz respectively.

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

1. Download and restore the `pll_mifstreaming.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 `stgkp.stp` file and download the `.sof` file.
6. Provide a low pulse on the `reset_SM` input pin to start the reconfiguration.

**Related Information**

- Design Example 1: PLL Reconfiguration with Altera PLL Reconfig IP Core to Reconfigure M, N, and C Counters on page 17

**Tutorial Walkthrough**

This tutorial walkthrough guides you on creating your design and running the tests. This tutorial assumes that you are familiar with the Intel Quartus Prime software and the Qsys system integration tool.

**Creating a New Intel Quartus Prime Project**

In the Intel Quartus Prime software, create a new Intel Quartus Prime project that targets a Stratix V device.

**Note:** Ensure that your project path does not include any spaces or extended character.

**Related Information**

- Managing Files in a Project, Intel Quartus Prime Help
  Provides more information about creating a new Intel Quartus Prime project.
Creating the Qsys System

To create a Qsys system, follow these steps:

1. On the Tool menu, click Qsys.
2. Add the Qsys components that your design requires.
   a. To create an Altera PLL instance, perform these steps:
      1. Expand PLL, select Altera PLL, and click Add.
      2. Set Reference Clock Frequency to 50.0 MHz.
      3. Select the number of output clock and set its output frequency and phase relationship.
      4. In the Settings tab, turn on the Enable dynamic reconfiguration of PLL option.
      5. Click Finish.
   b. To create an Altera PLL Reconfig instance, perform these steps:
      1. Expand PLL, select Altera PLL Reconfig, and click Add.
      2. Click Finish.
   c. Under Component Library > Memories and Memory Controllers > On-Chip, select On-Chip Memory (RAM or ROM), and then click Add.
      1. For Total Memory size, type 65536 bytes.
      2. Click Finish.
   d. Under Component Library, expand Embedded Processors, select Nios II Processor and click Add.
      1. Select Nios II/s.
      2. Set Reset Vector Offset to 0x00 and Exception Vector Offset to 0x20.
      3. Click Finish.
   e. Under Interface Protocols > Serial, select JTAG UART and click Add.
      1. Under Write FIFO and Read FIFO, for the Buffer depth (bytes) select 64, and for IRQ threshold type 8.
      2. For Prepare interactive windows, select INTERACTIVE_INPUT_OUTPUT to open the interactive display window during simulation.
      3. 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 Qsys System and Components for Design Example 3 figure.
   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 8: 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 Intel Quartus Prime 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 Intel Quartus Prime 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.
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 existing pins to the modified pin names as follows:
<table>
<thead>
<tr>
<th>Existing Pin Name</th>
<th>Modified Pin Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk_clk</td>
<td>inclk</td>
</tr>
<tr>
<td>reset_reset_n</td>
<td>reset_n</td>
</tr>
<tr>
<td>pll_0_reset_reset</td>
<td>areset</td>
</tr>
<tr>
<td>pll_0_locked_export</td>
<td>locked</td>
</tr>
<tr>
<td>pll_0_outclk0_clk</td>
<td>co_output</td>
</tr>
<tr>
<td>pll_0_outclk1_clk</td>
<td>c1_output</td>
</tr>
<tr>
<td>pll_0_outclk10_clk</td>
<td>c10_output</td>
</tr>
<tr>
<td>pll_0_outclk5_clk</td>
<td>c5_output</td>
</tr>
</tbody>
</table>

11. Connect the \texttt{pll_refclk_clk} port to the \texttt{inclk} pin.
12. On the File menu, click \texttt{Save}.
13. On the Project menu, click \texttt{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 \texttt{.sdc} file 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 \texttt{Other}, and select your project directory.
3. On the File menu, point to New and click \texttt{Project}.
4. Select \texttt{Nios II Application and BSP from Template} and click \texttt{Next}.
5. In the \texttt{Select Project Template} list, click \texttt{Blank Project}, and then locate the SOPC Information File (\texttt{.sopcinfo}).
6. Type \texttt{pll_reconfig} as project name under the Application project.
7. Click \texttt{Next} and click \texttt{Finish}.
8. In Window Explorer, drag \texttt{PLL_RECONFIG.c} to the \texttt{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, .rdata, .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 SignalTap II Logic Analyzer. The SignalTap II dialog box appears. The 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.

Implementing Fractional PLL Reconfiguration with Altera PLL and Altera PLL Reconfig IP Cores
Verifying Design with the Nios II SBT for Eclipse

To verify your design with the Nios II SBT for Eclipse, 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.

Document Revision History for AN 661: Implementing Fractional PLL Reconfiguration with Altera PLL and Altera PLL Reconfig IP Cores

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 2019.04.03       | • Changed from "VCO DIV Setting" to "VCO Post Divide Counter Setting", and updated the corresponding counter bit setting in the Fractional PLL Dynamic Reconfiguration Registers and Settings table.  
• Changed instances of Quartus II to Quartus Prime. |
| 2018.11.29       | Fixed broken links for design examples and corrected design files filename. |

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| May 2016         | 2016.05.02 | • Updated the description for `mgmt_reset` signal in Avalon-MM Signals in Altera PLL Reconfig IP Core table.  
• Updated M_{FRAC} equation for \( M \) Counter Fractional Value (\( k \)) in Fractional PLL Dynamic Reconfiguration Registers and Settings table.  
• Updated M_{FRAC} and \( C_0 \) counter values in Waveform Example for Dynamic Reconfiguration with Avalon-MM Interface section.  
• Updated design considerations on resynchronizing the fractional PLL using the arest signal. |
| May 2015         | 2015.05.04 | • Restructured the document.  
• Added a note on logical counter and physical counter in the Dynamic Phase Shift Signals in Altera PLL IP Core section.  
• Added instructions to stitch two `.mif` files generated by Quartus II software with a `.tcl` script.  
• Added Design Example 5: `.mif` Streaming Reconfiguration. |
<p>| August 2014      | 3.1     | • Changed ( C_{\text{counter}[18:22]} ) to ( C_{\text{counter}[22:18]} ) in Table 2 on page 4. |</p>
<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 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.                                                                                                                                  |