# Phase-Locked Loop Reconfiguration (ALTPLL_RECONFIG) IP Core

UG-032405-2017.06.16 User Guide

This user guide describes the features and behavior of the ALTPLL_RECONFIG megafunction IP core that you can configure through the parameter editor in the Quartus® II software.

This user guide assumes that you are familiar with IP cores and how to create them. If you are unfamiliar with Altera IP cores or the parameter editor, refer to *Introduction to Altera IP Cores*.

Phase-locked loops (PLLs) use divide counters and voltage-controlled oscillator (VCO) phase taps to perform frequency synthesis and phase shifts. In enhanced and fast PLLs, you can reconfigure the counter settings as well as phase shift the PLL output clock in real time. You can also change the charge-pump and loop-filter components, which dynamically affect the PLL bandwidth. The ALTPLL_RECONFIG IP core implements reconfiguration logic to facilitate dynamic real-time reconfiguration of PLLs in Altera devices. You can use the IP core to update the output clock frequency, PLL bandwidth, and phase shifts in real time, without reconfiguring the entire FPGA.

## Features

The ALTPLL_RECONFIG IP core offers the following additional features to the ALTPLL IP core:

- Reconfiguration of pre-scale counter (*N*) parameters.
- Reconfiguration of feedback counter (*M*) parameters.
- Reconfiguration of post-scale output counter (*C*) parameters.
- Reconfiguration of delay element or phase shift of each counter. For Stratix® IV, Cyclone® IV, Cyclone 10 LP, and Arria® II GX devices, use the ALTPLL IP core to access this feature.
- Dynamic adjustment of the charge-pump current and loop-filter components to facilitate dynamic reconfiguration of the PLL bandwidth. This feature is available only in Stratix IV devices.
- Reconfiguration from multiple configuration files using external read-only memory (ROM) in user mode. This feature is available only in Stratix IV, Cyclone IV, Cyclone 10 LP, and Arria II GX devices. The ALTPLL_RECONFIG supports reconfiguration from Memory Initialization File (**.mif**) and Hexadecimal File (**.hex**).

For more details about these features, refer to the *Clock Networks and PLLs* chapter of the respective device handbook.

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

# Common Applications

Use the ALTPLL_RECONFIG IP core in designs that must support dynamic changes in the frequency and phase shift of clocks and other frequency signals. The IP core is also useful in prototyping environments because it allows you to sweep PLL output frequencies and dynamically adjust the output clock phase. For example, a system generating test patterns is required to generate and transmit patterns at 50 or 100 MHz, depending on the device under test. Reconfiguring the PLL components in real-time allows you to switch between two such output frequencies within a few microseconds. You can also adjust the clock-to-output (tCO) delays in real-time by changing the output clock phase shift. This approach eliminates the need to regenerate a configuration file with the new PLL settings.

Reconfigurable PLLs are very useful in DDR 2 and DDR 3 interfaces to implement the dynamic data path (via the ALTMEMPHY IP core). The PLL is needed to drive the DLL used in the dynamic external memory interface operation. This operation requires dynamic phase-shifting.

For more information about dynamic phase-shifting in DDR 2 and DDR 3 interfaces, refer to the *ALTMEMPHY IP Core User Guide*.

In addition, you can dynamically configure Stratix IV, Cyclone IV, Cyclone 10 LP, and Arria II GX PLLs by using multiple configuration files stored on the external ROM.

# Resource Utilization and Performance

For details about the resource usage and performance of the ALTPLL_RECONFIG IP core in various devices, refer to the compilation reports in the Quartus II software.

To view the compilation reports for the ALTPLL_RECONFIG IP core in the Quartus II software, follow these steps:

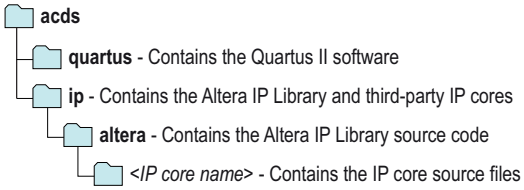
1. On the Processing menu, click **Start Compilation** to run a full compilation.
2. After compiling the design, on the Processing menu, click **Compilation Report**.
3. In the Table of Contents browser, expand the **Fitter** folder by clicking the "+" icon.
4. Under **Fitter**, expand **Resource section**, and select **Resource Usage Summary** to view the resource usage information.
5. Under **Fitter**, expand **Resource section**, and select **Resource Utilization by Entity** to view the resource utilization information.

# Installing and Licensing IP Cores

The Altera IP Library provides many useful IP core functions for production use without purchasing an additional license. You can evaluate any Altera IP core in simulation and compilation in the Quartus II software using the OpenCore evaluation feature.

Some Altera IP cores, such as MegaCore® functions, require that you purchase a separate license for production use. You can use the OpenCore Plus feature to evaluate IP that requires purchase of an additional license until you are satisfied with the functionality and performance. After you purchase a license, visit the Self Service Licensing Center to obtain a license number for any Altera product. For additional information, refer to *Altera Software Installation and Licensing*.

**Figure 1. IP core Installation Path**




The default installation directory on Windows is **<*drive*>:\altera\<*version number*>**; on Linux it is **<*home directory*>/altera/<*version number*>**.

# IP Catalog and Parameter Editor

The Quartus II IP Catalog (**Tools > IP Catalog**) and parameter editor help you easily customize and integrate IP cores into your project. You can use the IP Catalog and parameter editor to select, customize, and generate files representing your custom IP variation.

The IP Catalog (**Tools > IP Catalog**) and parameter editor replace the MegaWizard™ Plug-In Manager for IP selection and parameterization, beginning in Quartus II software version 14.0. Use the IP Catalog and parameter editor to locate and paramaterize Altera IP cores.

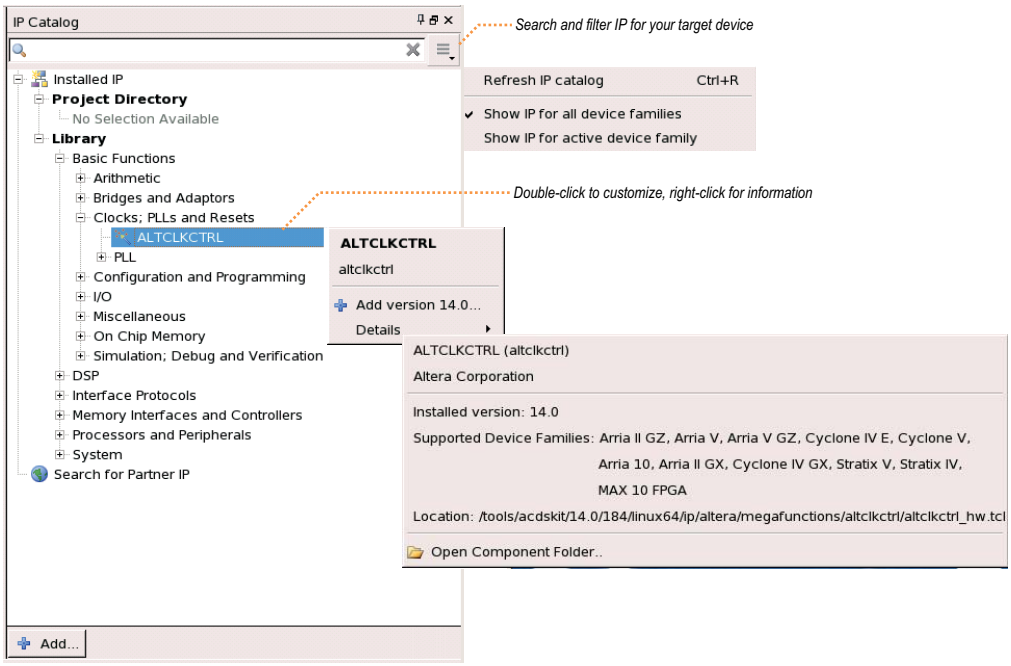
The IP Catalog lists IP cores available for your design. Double-click any IP core to launch the parameter editor and generate files representing your IP variation. The parameter editor prompts you to specify an IP variation name, optional ports, and output file generation options. The parameter editor generates a top level Qsys system file (**.qsys**) or Quartus II IP file (**.qip**) representing the IP core in your project. You can also parameterize an IP variation without an open project.

Use the following features to help you quickly locate and select an IP core:

- Filter IP Catalog to **Show IP for active device family** or **Show IP for all device families**.
- Search to locate any full or partial IP core name in IP Catalog. Click **Search for Partner IP**, to access partner IP information on the Altera website.

- Right-click an IP core name in IP Catalog to display details about supported devices, installation location, and links to documentation.

**Figure 2. Quartus II IP Catalog**




The IP Catalog is also available in Qsys (**View > IP Catalog**). The Qsys IP Catalog includes exclusive system interconnect, video and image processing, and other system-level IP that are not available in the Quartus II IP Catalog.

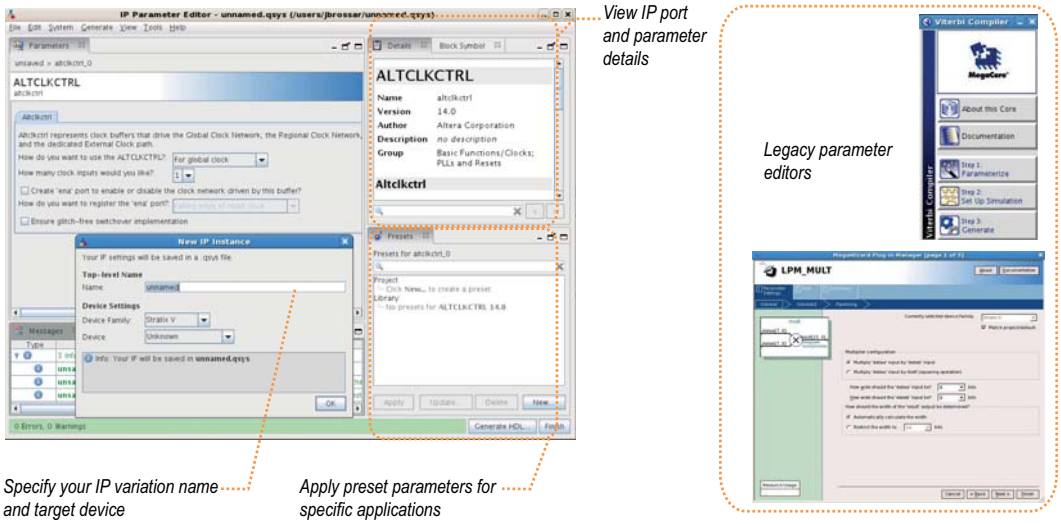
# Using the Parameter Editor

The parameter editor helps you to configure your IP variation ports, parameters, architecture features, and output file generation options:

- Use preset settings in the parameter editor (where provided) to instantly apply preset parameter values for specific applications.
- View port and parameter descriptions and links to detailed documentation.

- Generate testbench systems or example designs (where provided).

**Figure 3. IP Parameter Editors**

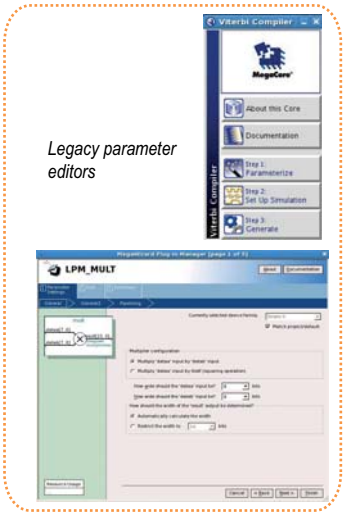



# Specifying IP Core Parameters and Options (Legacy Parameter Editors)

Some IP cores use a legacy version of the parameter editor for configuration and generation. Use the following steps to configure and generate an IP variation using a legacy parameter editor.

The legacy parameter editor generates a different output file structure than the latest parameter editor. Refer to *Specifying IP Core Parameters and Options* for configuration of IP cores using the latest parameter editor.

**Figure 4. Legacy Parameter Editors**



1. In the IP Catalog (**Tools > IP Catalog)**, locate and double-click the name of the IP core to customize. The parameter editor appears.
2. Specify a top-level name and output HDL file type for your IP variation. This name identifies the IP core variation files in your project. Click **OK**.
3. Specify the parameters and options for your IP variation in the parameter editor. Refer to your IP core user guide for information about specific IP core parameters.
4. Click **Finish** or **Generate** (depending on the parameter editor version). The parameter editor generates the files for your IP variation according to your specifications. Click **Exit** if prompted when generation is complete. The parameter editor adds the top-level **.qip** file to the current project automatically.

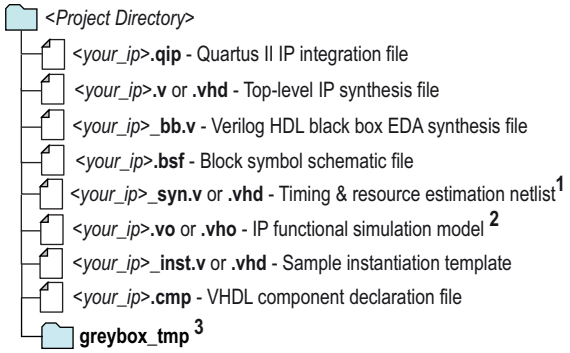
To manually add an IP variation generated with legacy parameter editor to a project, click **Project > Add/Remove Files in Project** and add the IP variation **.qip** file.

For information about using the latest parameter editor, refer to "Specifying IP Core Parameters and Options" in the *Introduction to Altera IP Cores*.

# Files Generated for Altera IP Cores (Legacy Parameter Editor)

The Quartus II software generates the following output file structure for Altera IP cores that use the legacy parameter editor:

**Figure 5. IP Core Generated Files (Legacy Parameter Editor)**




Notes:
1. If supported and enabled for your IP variation
2. If functional simulation models are generated
3. Ignore this directory

# Modifying an IP Variation

You can easily modify the parameters of any Altera IP core variation in the parameter editor to match your design requirements. Use any of the following methods to modify an IP variation in the parameter editor.

**Table 1. Modifying an IP Variation**

| Menu Command | Action |
|---|---|
| **File > Open** | Select the top-levelHDL(**.v**, or **.vhd**) IP variation file to launch the parameter editor and modify the IP variation. Regenerate the IP variation to implement your changes. |
| **View > Utility Windows > Project Navigator > IP Components** | Double-click the IP variation to launch the parameter editor and modify the IP variation. Regenerate the IP variation to implement your changes. |
| **Project > Upgrade IP Components** | Select the IP variation and click **Upgrade in Editor** to launch the parameter editor and modify the IP variation. Regenerate the IP variation to implement your changes. |

# Upgrading IP Cores

IP core variants generated with a previous version of the Quartus II software may require upgrading before use in the current version of the Quartus II software. Click **Project > Upgrade IP Components** to identify and upgrade IP core variants.

The **Upgrade IP Components** dialog box provides instructions when IP upgrade is required, optional, or unsupported for specific IP cores in your design. You must upgrade IP cores that require it before you can compile the IP variation in the current version of the Quartus II software. Many Altera IP cores support automatic upgrade.

The upgrade process renames and preserves the existing variation file (**.v**, **.sv**, or **.vhd**) as <*my_ip*>_ **BAK.v**, **.sv**, **.vhd** in the project directory.

**Table 2. IP Core Upgrade Status**

| IP Core Status | Corrective Action |
|---|---|
| Required Upgrade IP Components | You must upgrade the IP variation before compiling in the current version of the Quartus II software. |
| Optional Upgrade IP Components | Upgrade is optional for this IP variation in the current version of the Quartus II software. You can upgrade this IP variation to take advantage of the latest development of this IP core. Alternatively you can retain previous IP core characteristics by declining to upgrade. |
| Upgrade Unsupported | Upgrade of the IP variation is not supported in the current version of the Quartus II software due to IP core end of life or incompatibility with the current version of the Quartus II software. You are prompted to replace the obsolete IP core with a current equivalent IP core from the IP Catalog. |

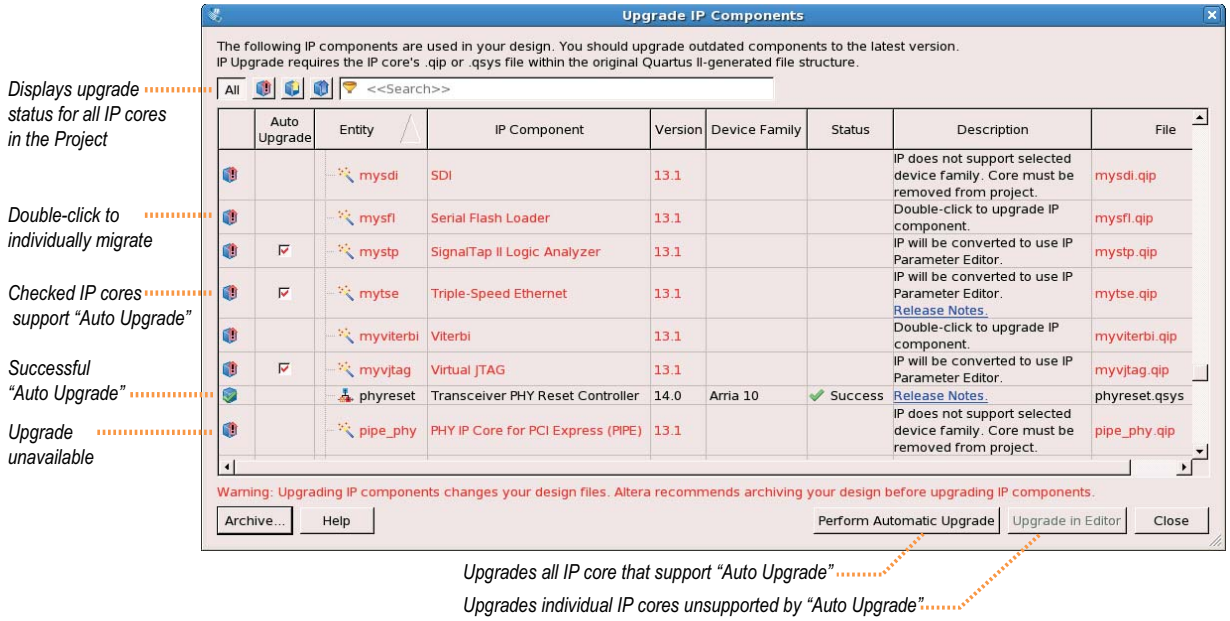
### Before you begin

- Archive the Quartus II project containing outdated IP cores in the original version of the Quartus II software: Click **Project > Archive Project** to save the project in your previous version of the Quartus II software. This archive preserves your original design source and project files.

- Restore the archived project in the latest version of the Quartus II software: Click **Project > Restore Archived Project**. Click **OK** if prompted to change to a supported device or overwrite the project database. File paths in the archive must be relative to the project directory. File paths in the archive must reference the IP variation **.v** or **.vhd** file or **.qsys** file (not the **.qip** file).

1. In the latest version of the Quartus II software, open the Quartus II project containing an outdated IP core variation. The **Upgrade IP Components** dialog automatically displays the status of IP cores in your project, along with instructions for upgrading each core. Click **Project > Upgrade IP Components** to access this dialog box manually.
2. To simultaneously upgrade all IP cores that support automatic upgrade, click **Perform Automatic Upgrade**. The **Status** and **Version** columns update when upgrade is complete. Example designs provided with any Altera IP core regenerate automatically whenever you upgrade the IP core.

**Figure 6. Upgrading IP Cores**




## Upgrading IP Cores at the Command Line

You can upgrade IP cores that support auto upgrade at the command line. IP cores that do not support automatic upgrade do not support command line upgrade.

- To upgrade a single IP core that supports auto-upgrade, type the following command:
  `quartus_sh -ip_upgrade -variation_files` *<my_ip_filepath/my_ip>.<hdl>* *<qii_project>*
  Example: `quartus_sh -ip_upgrade -variation_files mega/pll25.v hps_testx`

- To simultaneously upgrade multiple IP cores that support auto-upgrade, type the following command:
  `quartus_sh -ip_upgrade -variation_files "<my_ip_filepath/my_ip1>.<hdl>;<my_ip_filepath/my_ip2>.<hdl>" <qii_project>`
  Example: `quartus_sh -ip_upgrade -variation_files "mega/pll_tx2.v;mega/pll3.v" hps_testx`

IP cores older than Quartus II software version 12.0 do not support upgrade. Altera verifies that the current version of the Quartus II software compiles the previous version of each IP core. The *MegaCore IP Library Release Notes* reports any verification exceptions for MegaCore IP. The *Quartus II Software and Device Support Release Notes* reports any verification exceptions for other IP cores. Altera does not verify compilation for IP cores older than the previous two releases.

# Migrating IP Cores to a Different Device

IP migration allows you to target the latest device families with IP originally generated for a different device. Some Altera IP cores require individual migration to upgrade. The **Upgrade IP Components** dialog box prompts you to double-click IP cores that require individual migration.

1. To display IP cores requiring migration, click **Project > Upgrade IP Components**. The **Description** field prompts you to double-click IP cores that require individual migration.
2. Double-click the IP core name, and then click **OK** after reading the information panel. The parameter editor appears showing the original IP core parameters.
3. For the **Currently selected device family**, turn off **Match project/default**, and then select the new target device family.
4. Click **Finish**, and then click **Finish** again to migrate the IP variation using best-effort mapping to new parameters and settings. Click **OK** if you are prompted that the IP core is unsupported for the current device. A new parameter editor opens displaying best-effort mapped parameters.
5. Click **Generate HDL**, and then confirm the Synthesis and Simulation file options. Verilog is the parameter editor default HDL for synthesis files. If your original IP core was generated for VHDL, select **VHDL** to retain the original output HDL format.
6. To regenerate the new IP variation for the new target device, click **Generate**. When generation is complete, click **Close**.
7. Click **Finish** to complete migration of the IP core. Click **OK** if you are prompted to overwrite IP core files. The **Device Family** column displays the migrated device support. The migration process replaces <*my_ip*>**.qip** with the <*my_ip*>**.qsys** top-level IP file in your project.

   If migration does not replace <*my_ip*>**.qip** with <*my_ip*>.**qsys**, click **Project > Add/Remove Files in Project** to replace the file in your project.

8. Review the latest parameters in the parameter editor or generated HDL for correctness. IP migration may change ports, parameters, or functionality of the IP core. During migration, the IP core's HDL generates into a library that is different from the original output location of the IP core. Update any assignments that reference outdated locations. If your upgraded IP core is represented by a symbol in a supporting Block Design File schematic, replace the symbol with the newly generated <*my_ip*>**.bsf** after migration.

The migration process may change the IP variation interface, parameters, and functionality. This may require you to change your design or to re-parameterize your variant after the **Upgrade IP Components** dialog box indicates that migration is complete. The **Description** field identifies IP cores that require design or parameter changes.

For more information about specific IP cores, refer to IP user guide documentation and the *Altera IP Release Notes*.

# Parameter Settings

Altera recommends that you configure the IP core using the parameter editor. This section describes the parameters in the ALTPLL_RECONFIG parameter editor.

Expert users may choose to instantiate and configure the IP core using the clear box generator.

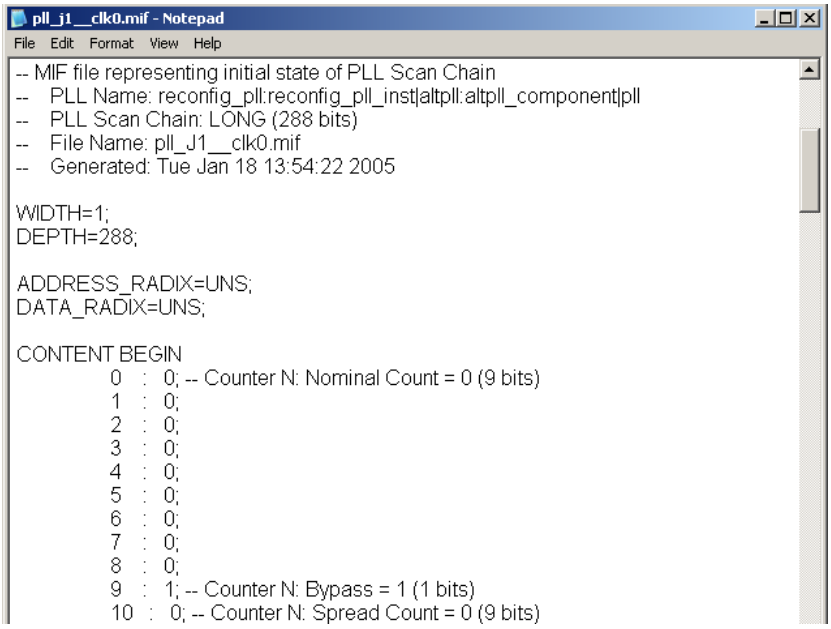
Table 3 lists the parameter settings for the ALTPLL_RECONFIG IP core.

**Table 3. ALTPLL_RECONFIG Parameter Settings**

| Page | Options | Description |
|---|---|---|
| Parameter Settings | Currently Selected Device Family | Specifies the chosen device family. |
| | Which scan chain type will you be using? | Scan chain is serial shift register chain that is used to store settings. It acts like a cache. When you assert the reconfig signal, the PLL is reconfigured with the values in the cache. The type of scan chain must follow the type of PLL to be reconfigured.<br>For Stratix IV, —Specifies the scan chain type as either **Top/Bottom** or **Left/Right**. For Cyclone IV, Cyclone 10 LP, and Arria II GX devices—The scan chain type has a default value of **Left/Right**. |
| | Do you want to specify the initial value of the scan chain? | Specifies the initial value of the scan chain. Select **No, leave it blank** to not specify a file or select **Yes, use this file for the content data** to browse for a **.hex** or **.mif** file.<br>You can also choose to initialize from ROM by turning on **Do not use pre-initialized RAM - initialize from ROM instead**.<br>For Cyclone IV and Cyclone 10 LP devices—The option to initialize from a ROM is not available. However, you can choose to add ports to write to the scan chain from an external ROM during runtime by turning on **Add ports to write to the scan chain from external ROM during run time**. |
| | Add ports to write to the scan chain from external ROM during run time | This option is only available for Stratix IV, Cyclone IV, Cyclone 10 LP, and Arria II GX devices. This option takes advantage of cycling multiple configuration files, which are stored in external ROMs during user mode. This capability is demonstrated in the functional description section, "Functional Description—Implementing Multiple Reconfiguration Using an External ROM" on page 14. |
| EDA | | Specifies the libraries needed for functional simulation. |
| | Generate netlist | Specifies whether to turn on the option to generate synthesis area and timing estimation netlist. |
| Summary | | Specifies the types of files to be generated. A gray checkmark indicates a file that is automatically generated; a red checkmark indicates an optional file.<br>Choose from the following types of files:<br>■ AHDL Include file (*<function name>*.**inc**)<br>■ VHDL component declaration file (*<function name>*.**cmp**)<br>■ Quartus II symbol file (*<function name>*.**bsf**)<br>■ Instantiation template file (*<function name>*_**inst.v** or *<function name>*_**inst.vhd**<br>■ Verilog HDL block box file (*<function name>*_**bb.v**)<br>If **Generate netlist** option is turned on, the file for that netlist is also available (*<function name>*_**syn.v)**. |

You can open a **.mif** in a text editor to make use of the comments embedded within the file. These comments show you the scan chain values and positions based on your design parameterization (see Figure 7). If you open a **.mif** in the Quartus II software, you can regenerate the **.mif** in the ALTPLL parameter editor to restore the comments.

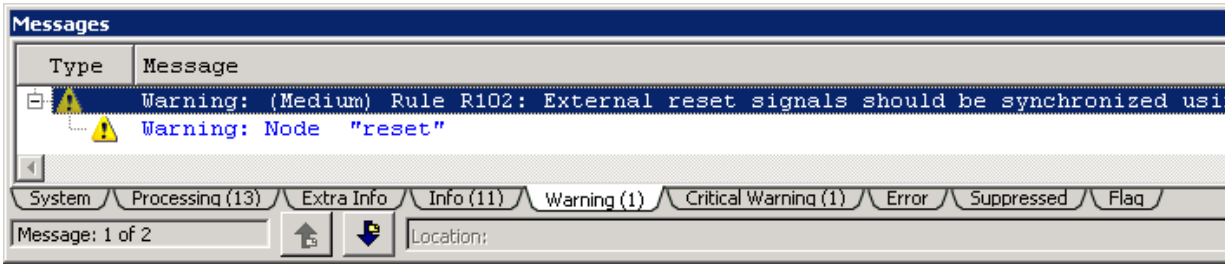
**Figure 7. MIF file**




# Checking Design Violations With the Design Assistant

The Design Assistant is a design rule checking tool that allows you to check for design issues early in the design flow. When you run the Design Assistant in the Quartus II software for the ALTPLL_RECONFIG IP core, you might receive the warning message shown in Figure 8.

**Figure 8. Warning Message in Design Assistant**




This message appears because there is a combinational logic in the IP core that connects the synchronous signal to the asynchronous external reset signal. To fix the issue, you must synchronize the external reset signal outside the IP core.

To synchronize the external reset signal, use the sample Verilog HDL code shown in Example 1. In the example, the input of `sync_reset_dffe1` is connected to the external reset pin, and the output of `sync_reset_dffe2` is connected to the `reset` input port of the ALTPLL_RECONFIG IP core.

**Example 1. Code to Synchronize External Reset Signal**

```
module synch_reg (reset, reconfig_clk, sync_reset_dffe2);
   input reset, reconfig_clk;
   output sync_reset_dffe2;
   reg sync_reset_dffe1, sync_reset_dffe2;
   always @(posedge reconfig_clk)
   begin
      sync_reset_dffe1 = reset;
   end
   always @(posedge reconfig_clk)
   begin
       sync_reset_dffe2 = sync_reset_dffe1;
   end
endmodule
```

# Simulation

You can perform functional and gate-level timing simulations of the IP core.

For more information, refer to the appropriate chapter in the *Simulation* section in volume 3 of the *Quartus II Handbook*.

If phase-shifting occurs after a PLL reconfiguration, use gate-level timing simulation instead of functional simulation to verify the correct counter settings and phase shifts. For non-zero PLL phase shifts, the frequency of the output clocks after a reconfiguration is correct, but the phase may be incorrect. If the phase shift is significant, use gate-level timing simulation to verify the timing behavior.

# Functional Description—Implementing Multiple Reconfiguration Using an External ROM

The ALTPLL_RECONFIG IP core allows you to reconfigure the PLL using an external ROM with multiple configuration files. With this feature, you can perform the following:

- Specify an external ROM and feed its content to the ALTPLL_RECONFIG IP core.
- Use the IP core with multiple PLL configuration settings that are stored in configuration files during user mode.
- Use the IP core with applications that require flexible dynamic-shifting of PLL settings during user mode.
- Reconfigure the initial PLL settings from a source other than an embedded random-access memory (RAM), such as an off-chip flash device, which is useful in HardCopy-type applications.

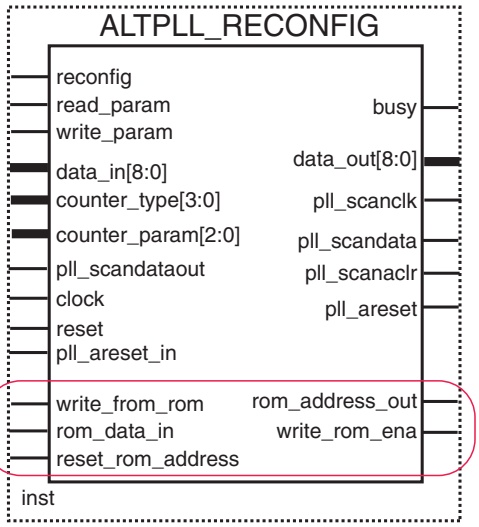
  This feature is available for Stratix IV, Cyclone IV, Cyclone 10 LP, and Arria II GX devices only.

To support reconfiguration from multiple configuration files, the ALTPLL_RECONFIG IP core has three input ports and two output ports:

- The `write_from_rom` input port signals the ALTPLL_RECONFIG IP core instantiation to write to the scan cache from the ROM.
- The `rom_data_in` input port holds data from the ROM.
- The `reset_rom_address` input port lets you restart the read process from the ROM. The data arrives serially from the ROM, starting from bit 0.
- The `rom_address_out` output bus holds the current address of the ROM data to be written to the scan cache.
- The `write_rom_ena` output port enables the intended ROM to be read out.

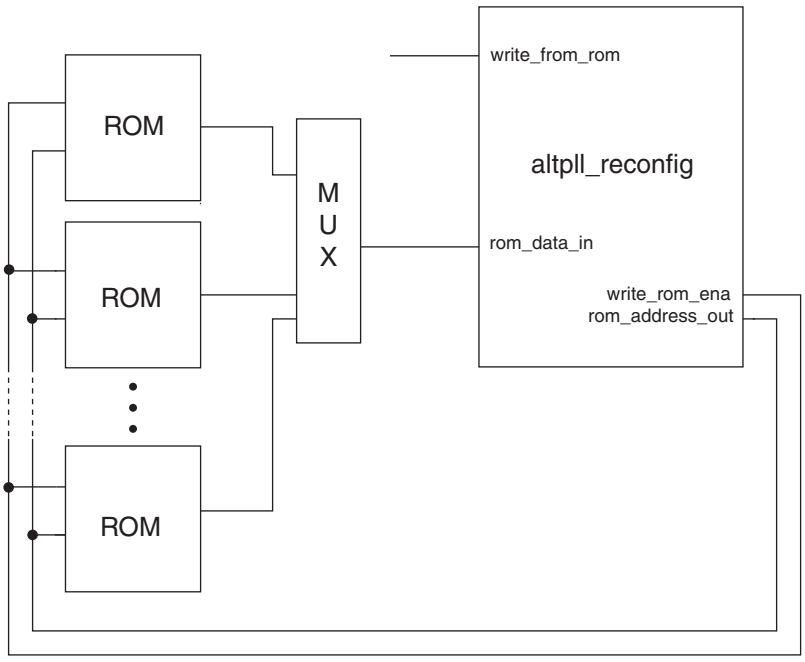
The input and output ports to support reconfiguration using multiple configuration files are shown in Figure 9, circled in red.

**Figure 9. Ports to Support Reconfiguration Using Multiple Configuration Files**




The reconfiguration feature using multiple configuration files allows you to feed data from multiple ROMs to a multiplexer that feeds the `rom_data_in` port. Figure 10 shows a sample design. In this scheme, the `write_rom_ena` signal feeds back to the ROM as the enable signal, which allows the ROM to be read out. The `rom_address_out` bus provides the intended ROM address, which determines the exact ROM data.

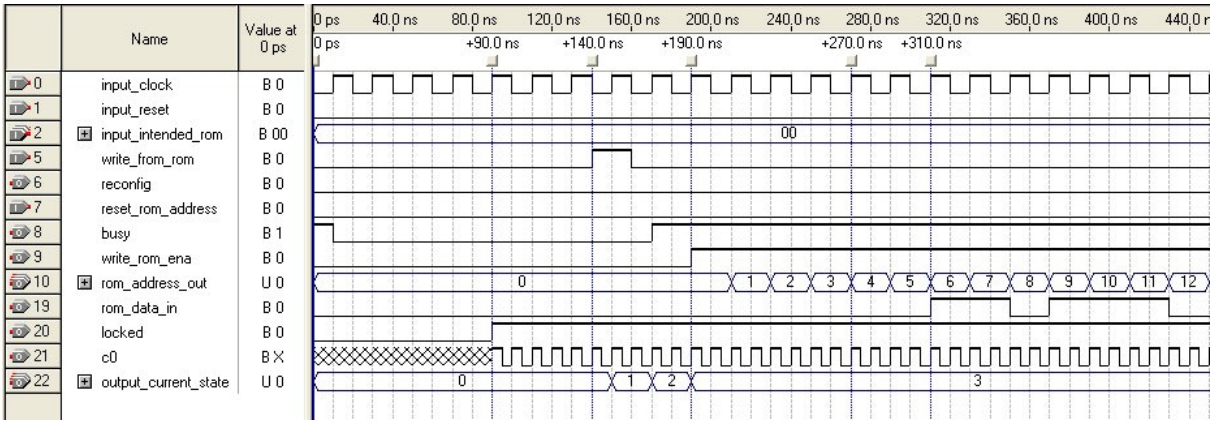
**Figure 10. Typical Scheme for Reconfiguring PLLs from External ROMs**



To copy the data from a ROM to the ALTPLL_RECONFIG IP core scan cache (a memory location that stores the PLL reconfiguration settings), you must hold the `write_from_rom` signal high for 1 clock cycle. The IP core asserts the busy signal on the first rising edge of the clock after the `write_from_rom` signal goes high. The busy signal remains asserted until all the bits are written into the scan cache.

On the second rising edge of the clock after the `write_from_rom` signal goes low again, the intended ROM address for the write operation appears on the `rom_address_out` port. The data of the ROM specified by the intended address on `rom_address_out` is fed to the `rom_data_in` input port of the ALTPLL_RECONFIG IP core instantiation. The `write_rom_ena` signal is also asserted on the second rising edge of the clock after the `write_from_rom` signal goes low again (refer to Figure 11).

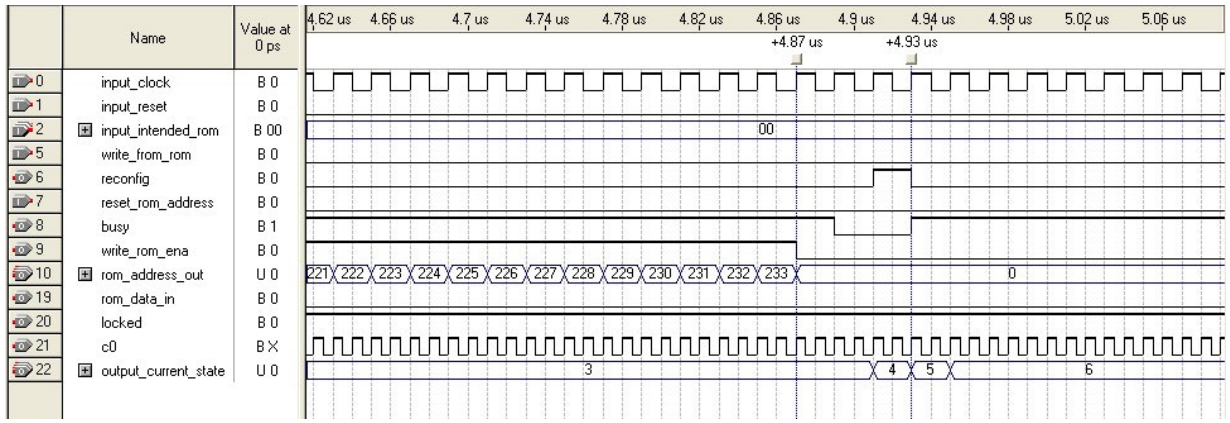
**Figure 11. Beginning Write to the Scan Cache of the ALTPLL_RECONFIG from the ROM**




The writing-to-scan cache process continues until the address reaches the specific size of the scan cache . This process is completed when the `busy` signal is deasserted. This means that the scan cache of the ALTPLL_RECONFIG IP core is written with the intended reconfiguration settings from the ROM.

After this, the `reconfig` signal can be asserted for 1 clock cycle to reconfigure the PLL to the intended settings that have been written to the scan cache of the ALTPLL_RECONFIG IP core (refer to Figure 12).

**Figure 12. Completing Write to the Scan Cache of the ALTPLL_RECONFIG from the ROM (1)**

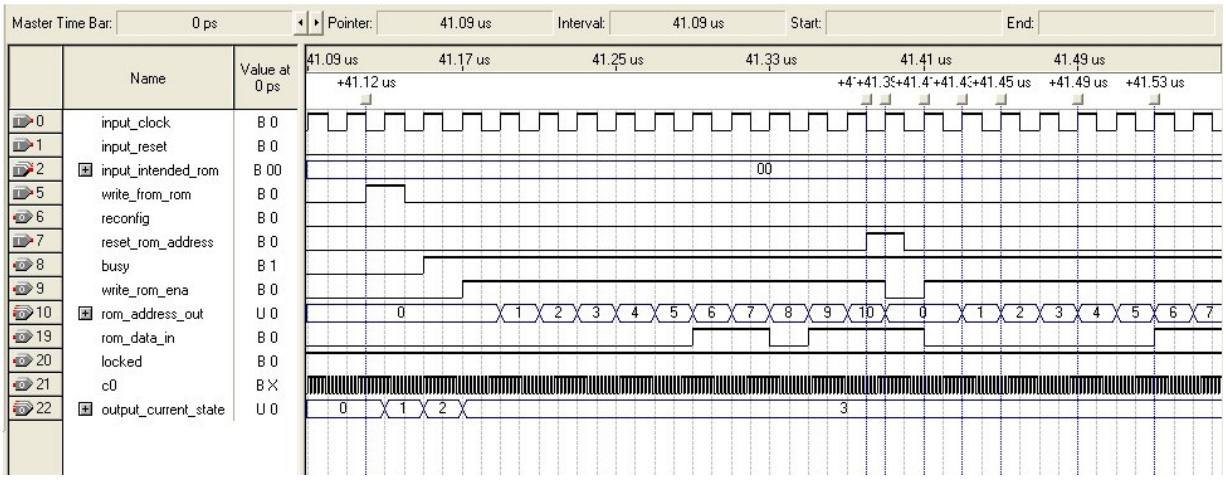



**Note to Figure 12:**

(1) This figure also shows the beginning of the reconfiguration process.

If you assert the `reset_rom_address` signal, the `write_rom_ena` signal is deasserted for 1 clock cycle and the `rom_address_out` signal resets. When the `write_rom_ena` gets asserted, the write process then restarts from address `0` (refer to Figure 13).

**Figure 13. Asserting the reset_rom_address Signal**



# Design Example

You can download design examples for this IP core from the following locations:

- On the Documentation: Quartus II Development Software page, in the Using IP cores section under I/O
- On the Documentation: User Guides webpage, with this user guide

The designs are simulated using the ModelSim®-Altera software to generate a waveform display of the device behavior. For more information about the ModelSim-Altera software, refer to the ModelSim-Altera Software Support page on the Altera website. The support page includes links to such topics as installation, usage, and troubleshooting.

## Frequency Division

This design example uses the ALTPLL_RECONFIG IP core to change the clock frequency of an enhanced PLL. This example demonstrates how to reconfigure the `c0` counter using the ALTPLL_RECONFIG IP core to vary the frequency of this counter by changing the *c* value. Figure 14 shows the formula for changing the *c* value for different PLL output frequencies.

**Figure 14. Frequency Division Formula**

Divide-by value = c = (Fin * m)/(Fout * n)

Where:
c value = High time count = Low time count
Fin = Input frequency
m = *m* modulus value
n = *n* modulus value
Fout = Required output frequency

This example reconfigures the output frequency of `c0` from 100 to 50 MHz by changing the divide-by value from 3 to 6.

### Generating the ALTPLL and ALTPLL_RECONFIG IP Cores

To generate the ALTPLL and ALTPLL_RECONFIG IP cores, follow these steps:

1. Open the **altpll_reconfig_DesignExample_ex1.zip** file and extract **pll_recon_ex1_1.1.qar**.
2. In the Quartus II software, open the **pll_recon_ex1_1.1.qar** file and restore the archive file into your working directory.
3. From the IP Catalog (**Tools > IP Catalog**), locate and double-click the ALTPLL IP core. The parameter editor appears.
4. Specify the following parameters for your IP variation:

**Table 4. Configuration Settings for the ALTPLL IP Core (Part 1 of 2)**

| Settings | Value |
|---|---|
| Which type of output file do you want to create? | **VHDL** |
| What name do you want for the output file? | **reconfig_pll** |

**Table 4. Configuration Settings for the ALTPLL IP Core (Part 2 of 2)**

| Settings | Value |
|---|---|
| Return to this page for another create operation | Turned on |
| Currently selected device family | **Stratix IV** |
| Match project/default | Turned on |
| Which device speed grade will you be using? | **Any** |
| What is the frequency of inclk0 input | **100 MHz** |
| Which PLL type will you be using? | **Enhanced PLL** |
| How will the PLL outputs be generated? | Select **Use the feedback path inside the PLL**. Select **In normal mode** |
| Which output clock will be compensated for? | **c0** |
| Create optional inputs for dynamic reconfiguration | Turned on |
| Long chain: All 6 core and 4 external clocks are available | Selected |
| Create an 'pllena' input to selectively enable the PLL | Turned off |
| Create an 'areset' input to asynchronously reset the PLL | Turned on |
| Create an 'pfdena' input to selectively enable the phase/frequency detector | Turned off |
| Create 'locked' output | Turned on |
| Create output file(s) using 'Advanced' PLL parameters | Turned off |
| Use this clock | Turned on |
| Enter output clock frequency | **100 MHz** |
| Clock phase shift | **0** degrees |
| Clock duty cycle (%) | **50** |
| Create a clock enable input | Turned off |
| Generate netlist | Turned off |
| Variation file | Turned on |
| PinPlanner ports PPF file | Turned on |
| AHDL Include file | Turned on |
| VHDL component declaration file | Turned on |
| Quartus II symbol file | Turned on |
| Instantiation template file | Turned on |

5. Click **Finish**. The `reconfig_pll` module is built.

6. Use the IP Catalog and parameter editor to specify the following parameters for the ALTPLL_RECONFIG IP variation:

**Table 5. Configuration Settings for ALTPLL_RECONFIG**

| Settings | Value |
|---|---|
| Which type of output file do you want to create? | **VHDL** |
| What name do you want for the output file? | **pll_reconfig** |
| Return to this page for another create operation | Turned off |
| Currently selected device family | **Stratix IV** |
| Match project/default | Turned on |
| Which scan chain type will you be using | **Long chain** |
| Do you want to specify initial value of the scan chain? | Select **Yes, use this file for the content data** |
| File name | **pll_j1__clk0.mif** |
| Do not use pre initialized RAM - initialize from ROM instead | Turned off |
| Generate netlist | Turned off |
| Variation file | Turned on |
| AHDL Include file | Turned on |
| VHDL component declaration file | Turned on |
| Quartus II symbol file | Turned on |
| Instantiation template file | Turned on |

7. Click **Finish**. The `pll_reconfig` module is built.

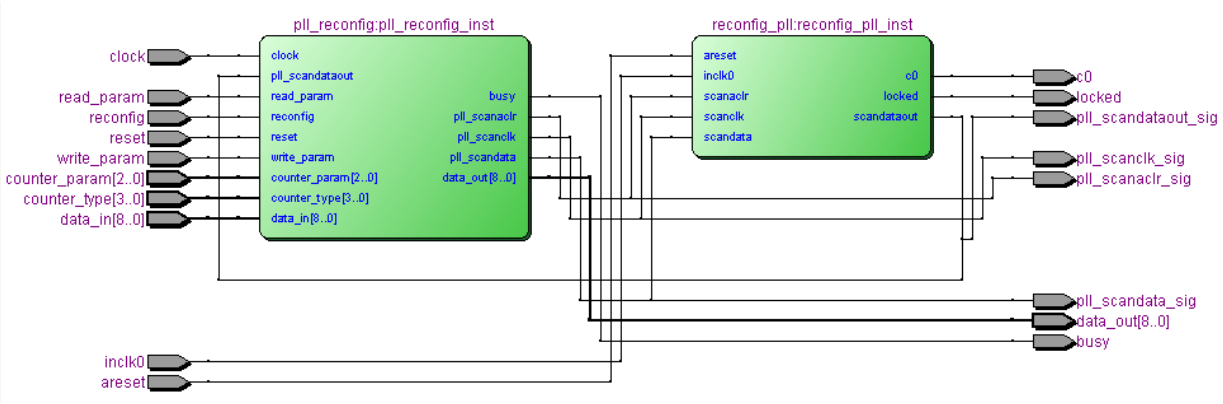
## Compiling ALTPLL and ALTPLL_RECONFIG

To compile the design in the Quartus II software, follow these steps:

1. Click **Processing > Start Compilation**.
2. When the **Full Compilation was successful** message box appears, click **OK**.

You have now created and compiled the complete design file, which can be viewed in the RTL Viewer (Figure 15). To display the RTL Viewer, in the Tools menu, select **Netlist Viewers**, and click on **RTL Viewer**.

**Figure 15. RTL Viewer — Complete Design File**




## Simulating the Design Example

To simulate the design example using the ModelSim-Altera software, follow these steps:

1. Unzip the **altpll_reconfig_ex1_msim.zip** file to any working directory on your PC.
2. Browse to the folder in which you unzipped the files.
3. Open **remote_update_ex2.do** in a text editor.
4. In line 1 of the **altpll_reconfig_ex1_msim.do** file, ensure that the directory path of the library files is correct. For example, C:/Modeltech_ae/altera/verilog/stratix.
5. On the File menu, click **Save**.
6. Launch the ModelSim-Altera software.
7. On the File menu, click **Change Directory**.
8. Select the folder in which you unzipped the files.
9. Click **OK**.
10. On the Tools menu, click **Execute Macro**.
11. Select the **altpll_reconfig_ex1_msim.do** file and click **Open**. This is a script file for ModelSim-Altera software to automate all the necessary settings for the simulation.
12. Verify the results shown in the Wave window.

You can rearrange, remove, and add signals, and change the radix by modifying the script **altpll_reconfig_ex1_msim.do**.

Figure 16 and Figure 17 show the expected simulation results in the ModelSim-Altera software. Figure 17 shows the change in `c0` frequency starting from 12.75 ms.

**Figure 16. Simulation Results in the ModelSim-Altera Software (8.9 to 9.5 ms)**

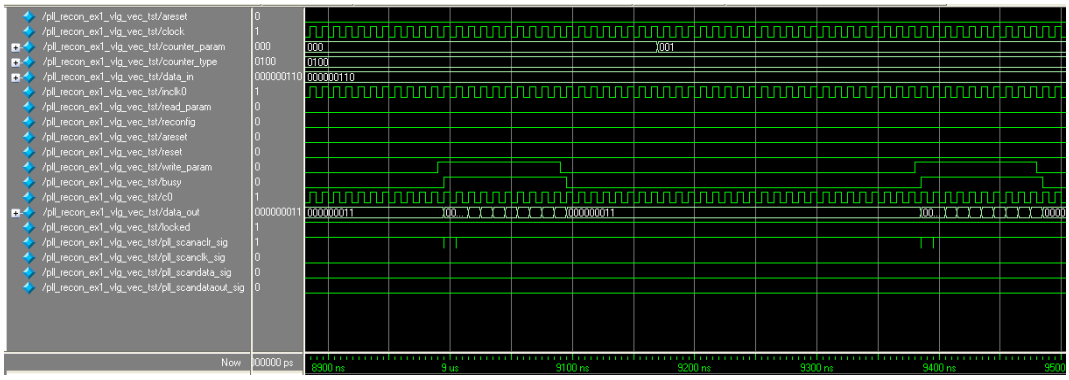
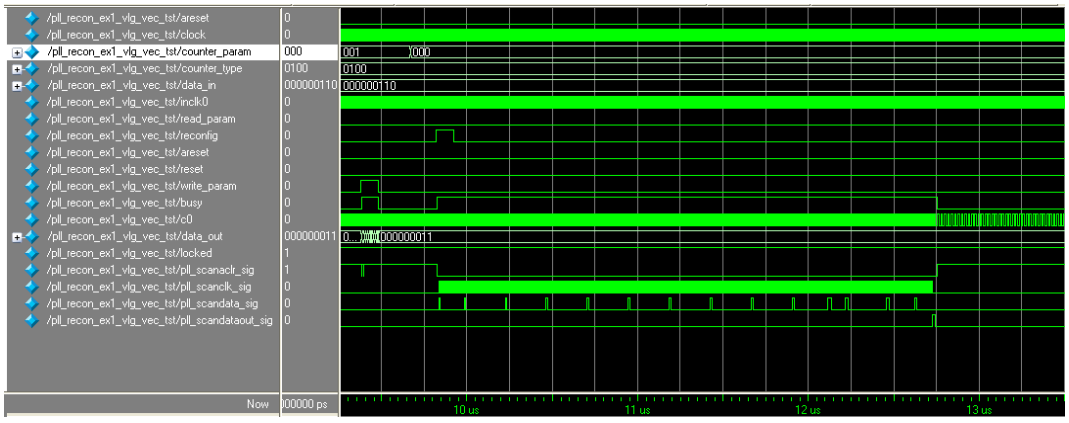



**Figure 17. Simulation Results in the ModelSim-Altera Software (9.5 to 13.5 ms)**



## Pulse Width Variation

This design example uses the ALTPLL_RECONFIG IP core to modify the pulse width of an enhanced PLL. This example demonstrates how to reconfigure the `c1` counter using the ALTPLL_RECONFIG IP core to vary the pulse width of this counter by changing the high-count and low-count values. The formula for changing the duty cycle is shown in Figure 18.

**Figure 18. Changing the Duty Cycle Formula**

Duty cycle =(Ch/Ct) % high time count and (Cl/Ct) % low time count
with RSELODD = 0

Where:
Ch = High time count
Cl = Low time count
Ct = Total time

When you set RSELODD = 1, you subtract 0.5 cycles from the high time and you add 0.5 cycles to the low time.

For example, if:
Ch = 2 cycles
Cl = 1 cycle
(Note: For odd division factors, the larger number is for the Ch counter; the smaller number is for the Cl counter.)

Setting RSELODD = 1 effectively changes the Ch and Cl to:

High time count = 1.5 cycles
Low time count = 1.5 cycles

Duty cycle = (1.5/3) % high time count and (1.5/3) % low time count

In this example, the pulse width is programmed to change from 50% to 25% , and then to 75% of the duty cycle.

### Generating ALTPLL and ALTPLL_RECONFIG

To generate the ALTPLL and ALTPLL_RECONFIG IP cores, perform the following steps:

1. Open **ALTPLL_RECONFIG_DesignExample_ex2.zip** and extract **pll_recon_ex2_1.1.qar**.
2. In the Quartus II software, open **pll_recon_ex2_1.1.qar** and restore the archive file into your working directory.
3. From the IP Catalog (**Tools > IP Catalog**), locate and double-click the ALTPLL IP core. The parameter editor appears.
4. Specify the following parameters for your IP variation:

**Table 6. Configuration Settings for ALTPLL (Part 1 of 2)**

| Settings | Value |
|---|---|
| Which type of output file do you want to create? | **VHDL** |
| What name do you want for the output file? | **reconfig_pll** |
| Return to this page for another create operation | Turned on |
| Currently selected device family | **Stratix IV** |

**Table 6. Configuration Settings for ALTPLL (Part 2 of 2)**

| Settings | Value |
|---|---|
| Match project/default | Turned on |
| Which device speed grade will you be using? | **Any** |
| What is the frequency of inclk0 input | **20 MHz** |
| Which PLL type will you be using? | **Enhanced PLL** |
| How will the PLL outputs be generated? | Select **Use the feedback path inside the PLL**.<br>Select **In normal mode** |
| Which output clock will be compensated for? | **c1** |
| Create optional inputs for dynamic reconfiguration | Turned on |
| Long chain: All 6 core and 4 external clocks are available | Selected |
| Create an 'pllena' input to selectively enable the PLL | Turned off |
| Create an 'areset' input to asynchronously reset the PLL | Turned on |
| Create an 'pfdena' input to selectively enable the phase/frequency detector | Turned off |
| Create 'locked' output | Turned on |
| Create output file(s) using 'Advanced' PLL parameters | Turned off |
| Use this clock | Turned on |
| Enter output clock frequency | **15 MHz** |
| Clock phase shift | **0 degrees** |
| Clock duty cycle (%) | **50** |
| Create a clock enable input | Turned off |
| Generate netlist | Turned off |
| Variation file | Turned on |
| PinPlanner ports PPF file | Turned on |
| AHDL Include file | Turned on |
| VHDL component declaration file | Turned on |
| Quartus II symbol file | Turned on |
| Instantiation template file | Turned on |

5. Click **Finish**. The `reconfig_pll` module is built.

6. Use the IP Catalog and parameter editor to specify the following parameters for the ALTPLL_RECONFIG IP variation:

**Table 7. Configuration Settings for ALTPLL_RECONFIG**

| Settings | Value |
|---|---|
| Which type of output file do you want to create? | **VHDL** |
| What name do you want for the output file? | **pll_reconfig** |
| Return to this page for another create operation | Turned off |
| Currently selected device family | **Stratix IV** |
| Match project/default | Turned on |
| Which scan chain type will you be using | **Long chain** |
| Do you want to specify initial value of the scan chain? | Select **Yes, use this file for the content data** |
| File name | **pll_j1__pll.mif** |
| Do not use pre initialized RAM - initialize from ROM instead | Turned off |
| Generate netlist | Turned off |
| Variation file | Turned on |
| AHDL Include file | Turned on |
| VHDL component declaration file | Turned on |
| Quartus II symbol file | Turned on |
| Instantiation template file | Turned on |

7. Click **Finish**. The `pll_reconfig` module is built.

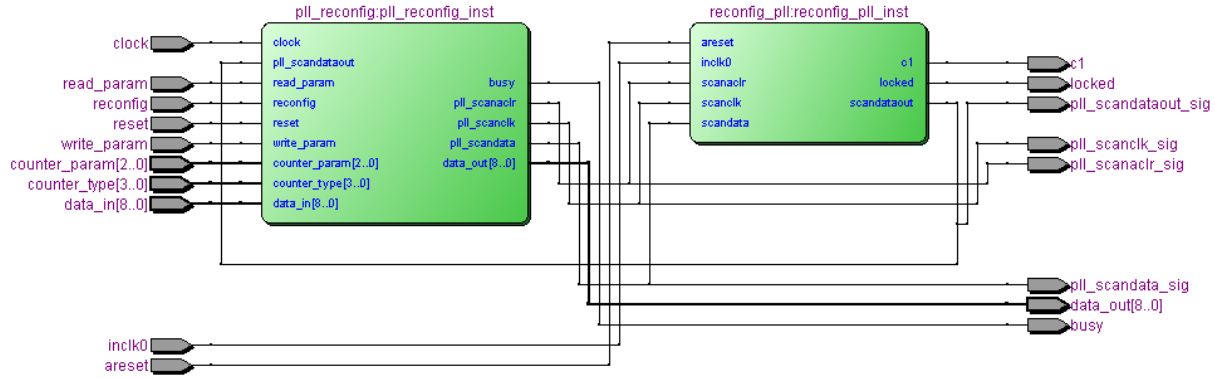
## Compiling ALTPLL and ALTPLL_RECONFIG

To compile the design in the Quartus II software, follow these steps:

1. To compile the design, on the Processing menu, click **Start Compilation**.
2. When the **Full Compilation was successful** message box appears, click **OK**.

You have now created and compiled the complete design file, which can be viewed in the RTL Viewer (Figure 19). To display the RTL Viewer, in the Tools menu, select **Netlist Viewers**, and click on **RTL Viewer**.

**Figure 19. RTL Viewer — Complete Design File**



## Simulating the Design Example

To simulate the design example using the ModelSim-Altera software, follow these steps:

1. Unzip the **altpll_reconfig_ex2_msim.zip** file to any working directory on your PC.
2. Browse to the folder in which you unzipped the files.
3. Open the **remote_update_ex2.do** file in a text editor.
4. In line 1 of the **altpll_reconfig_ex2_msim.do** file, make sure the directory path of the library files is correct. For example, C:/Modeltech_ae/altera/verilog/stratix.
5. On the File menu, click **Save**.
6. Launch the ModelSim-Altera software.
7. On the File menu, click **Change Directory**.
8. Select the folder in which you unzipped the files.
9. Click **OK**.
10. On the Tools menu, click **Execute Macro**.
11. Select the **altpll_reconfig_ex2_msim.do** file and click **Open**. This is a script file for ModelSim-Altera software to automate all of the necessary settings for the simulation.
12. Verify the results shown in the Wave window.

You can rearrange, remove, and add signals. and change the radix by modifying the script **altpll_reconfig_ex2_msim.do**.

Figure 20 through Figure 25 show the expected simulation results in the ModelSim-Altera software. The duty cycle changes from a ratio of 50:50 to 25:75 and finally to 75:25.

**Figure 20. Changing Parameters (2.11 to 7.23 ms)**

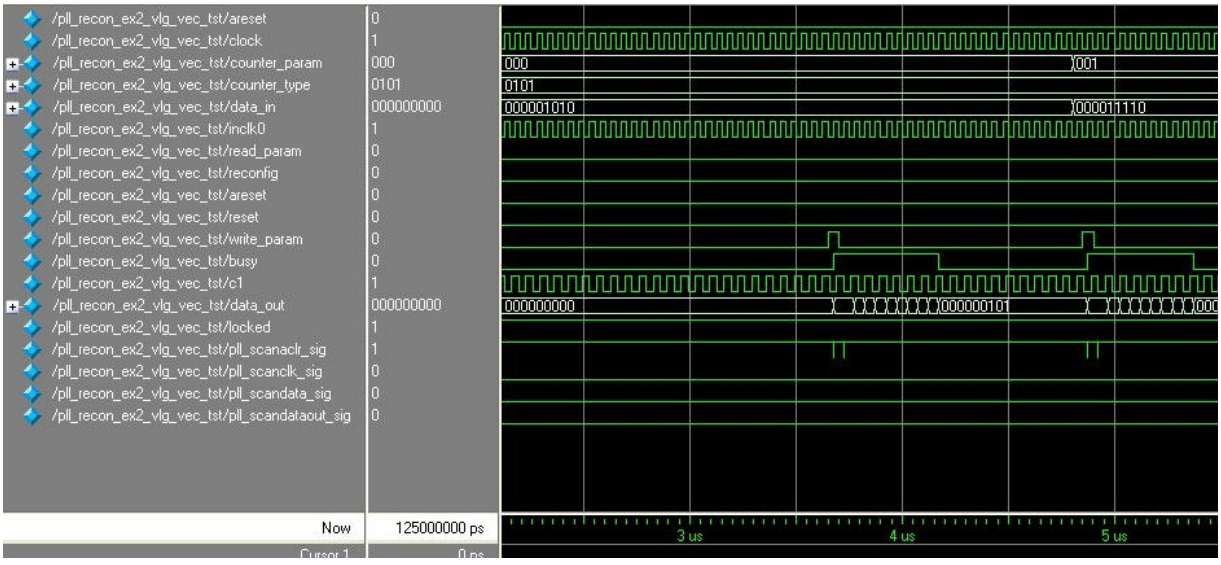


**Figure 21. Reconfiguration (6.32 to 26.8 ms)**

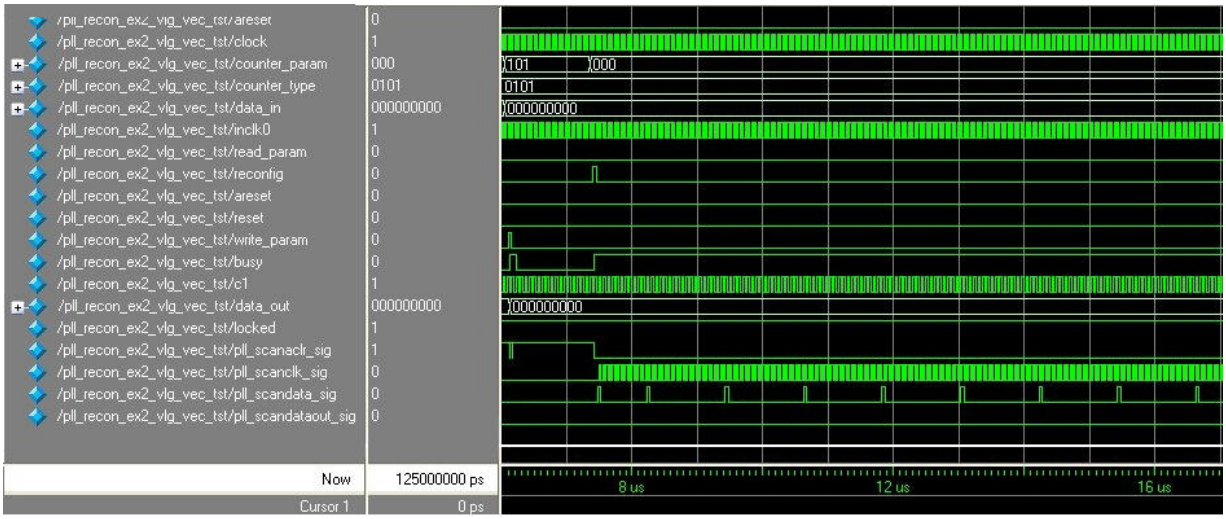



**Figure 22. Pulse Width Changes From 50:50 Ratio to 25:75 Ratio (20 to 26 ms)**

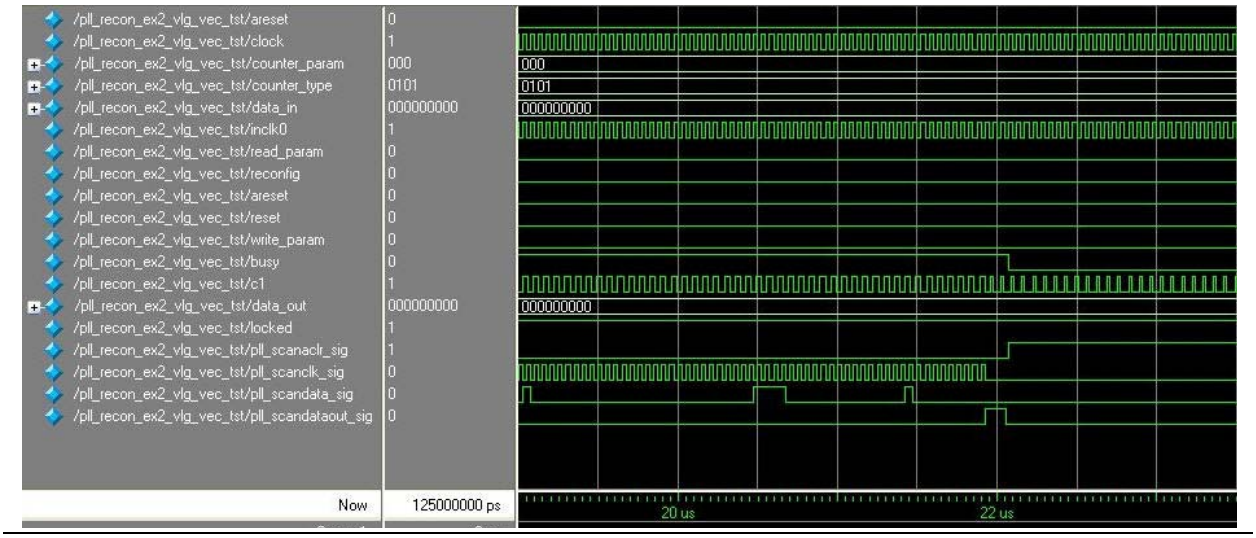


**Figure 23. Changing Parameters (91.79 to 96.91 ms)**

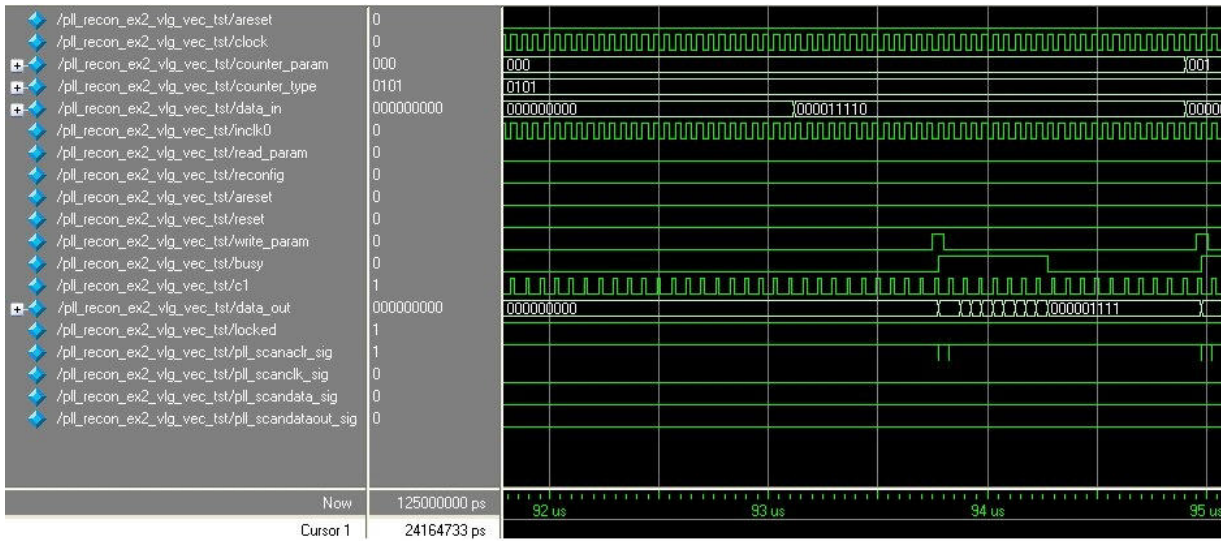



**Figure 24. Reconfiguration (96.92 to 117.4 ms)**

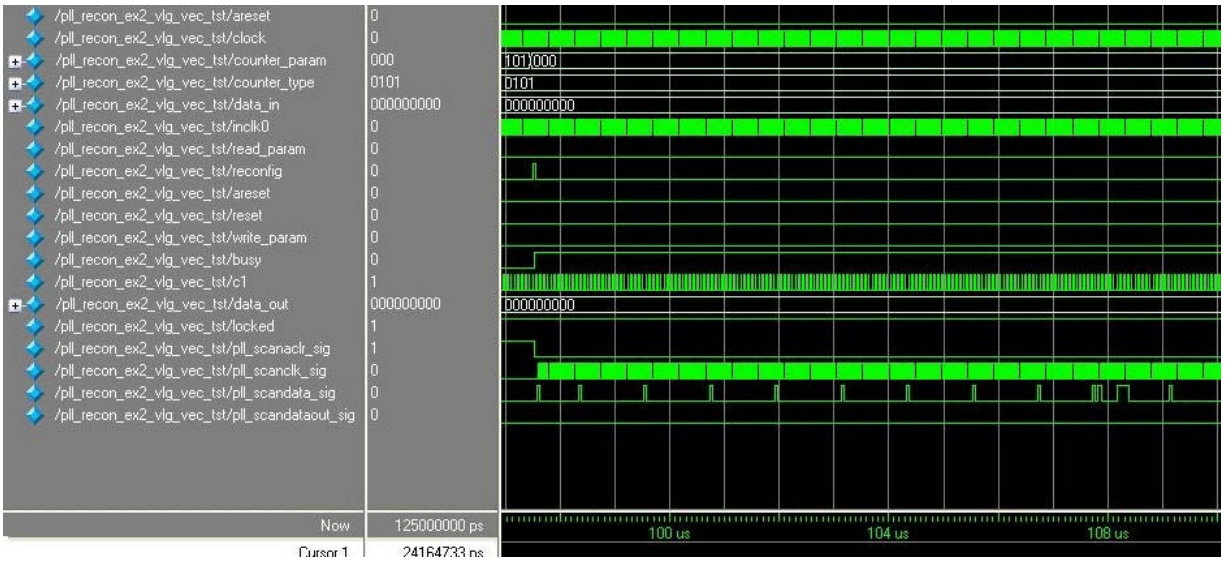
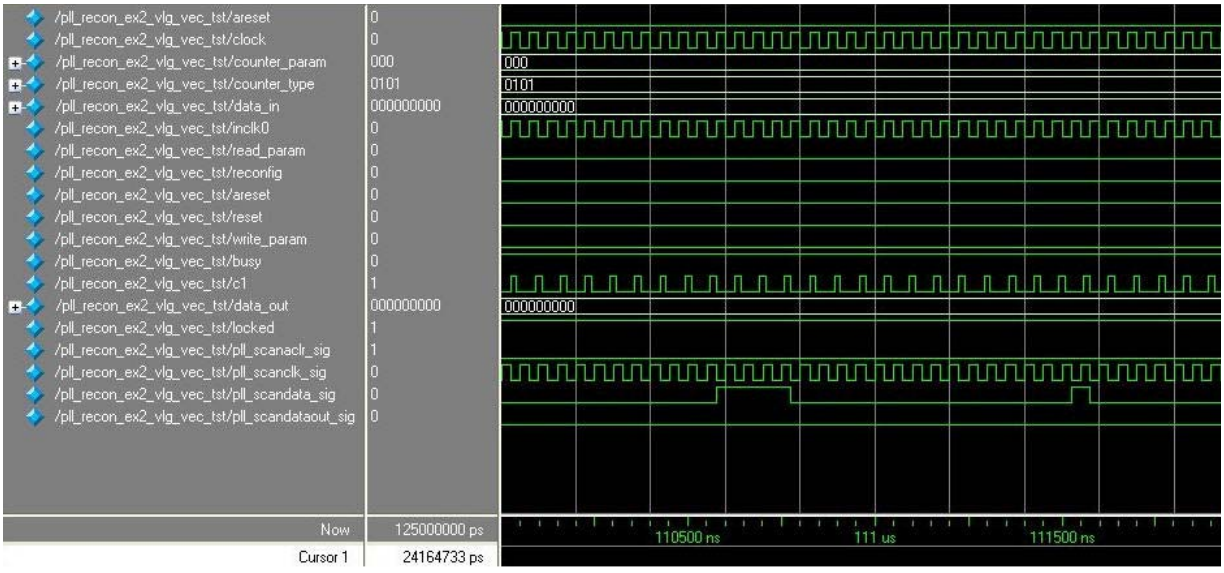


**Figure 25. Pulse Width Changes To 75:25 Ratio (110 to 113 ms)**



## PLL Reconfiguration with Multiple .mif Files

This design example uses the ALTPLL_RECONFIG IP core to reconfigure the output of the `c0` counter based on the PLL settings specified in multiple **.mif** files from external ROMs in Stratix III devices. The **.mif** files specify PLL settings that reconfigure the output of the `c0` counter from 100 to 200 MHz, 300 MHz, 400 MHz, and 500 MHz, then back to 200 MHz.

### Generating ALTPLL and ALTPLL_RECONFIG

To generate the ALTPLL and ALTPLL_RECONFIG IP cores, follow these steps:

1. Open the **ALTPLL_RECONFIG_DesignExample_ex3.zip** file to any directory on your PC.
2. Open the **ALTPLL_RECONFIG_rom.qar** project file.
3. From the IP Catalog (**Tools > IP Catalog**), locate and double-click the ALTPLL IP core. The parameter editor appears.

4. Specify the following parameters for your IP variation:

**Table 8. Configuration Settings for ALTPLL (Part 1 of 2)**

| Settings | Value |
|---|---|
| Which type of output file do you want to create? | **Verilog** |
| What name do you want for the output file? | **the_pll.v** |
| Return to this page for another create operation | Turned on |
| Currently selected device family | **Stratix IV** |
| Match project/default | Turned on |
| Which device speed grade will you be using? | **Any** |
| What is the frequency of inclk0 input | **50 MHz** |
| Which PLL type will you be using? | **Top_Bottom PLL** |
| How will the PLL outputs be generated? | Select **Use the feedback path inside the PLL**. Select **In normal mode** |
| Which output clock will be compensated for? | **c0** |
| Create an 'pllena' input to selectively enable the PLL | Disabled |
| Create an 'areset' input to asynchronously reset the PLL | Turned on |
| Create an 'pfdena' input to selectively enable the phase/frequency detector | Turned off |
| Create 'locked' output | Turned on |
| Enable self reset on loss lock | Turned off |
| Create output file(s) using 'Advanced' PLL parameters | Turned off |
| Create optional inputs for dynamic reconfiguration | Turned on |
| Initial Configuration File (filename) | **the_pll_initial.mif**—taking an `inclock` of 50 MHz and generating `c0` of 100 MHz<br>Ensure that this option shows the correct path of the .mif file before compiling the design to avoid scan chain mismatch warnings. |
| Additional Configuration File (filename) | The files are already generated. They are:<br>■ **the_pll_200_mhz.mif**—taking an `inclock` of 50 MHz and generating `c0` of 200 MHz<br>■ **the_pll_300_mhz.mif**—taking an `inclock` of 50 MHz and generating c0 of 300 MHz<br>■ **the_pll_400_mhz.mif**—taking an `inclock` of 50 MHz and generating `c0` of 400 MHz<br>■ **the_pll_500_mhz.mif**—taking an `inclock` of 50 MHz and generating `c0` of 500 MHz |

**Table 8. Configuration Settings for ALTPLL (Part 2 of 2)**

| Settings | Value |
|---|---|
| Create optional inputs for dynamic phase reconfiguration | Turned off |
| Use this clock | Turned on |
| Enter output clock frequency | **100 MHz** |
| Clock phase shift | **0 degrees** |
| Clock duty cycle (%) | **50** |
| Generate netlist | Turned off |
| Variation file | Turned on |
| PinPlanner ports PPF file | Turned on |
| AHDL Include file | Turned on |
| VHDL component declaration file | Turned on |
| Quartus II symbol file | Turned on |
| Instantiation template file | Turned on |
| Verilog HDL block box file | Turned on |
| Reconfiguration File for altpll_reconfig | Turned on |

The ALTPLL IP core allows you to generate additional configuration files without going through a compilation stage. It allows you to generate as many unique configuration files as you need without the difficulty of multiple compilation flows. All you need to do is to set the intended PLL settings, enter the file name, and click **Generate A Configuration File**. Use this capability with the PLL reconfiguration of multiple **.mif** files via external ROMs in the ALTPLL_RECONFIG IP core.

5. Click **Finish**. The **the_pll.v** module is built.
6. Use the IP Catalog and parameter editor to specify the following parameters for the ALTPLL_RECONFIG IP variation:

**Table 9. Configuration Settings for ALTPLL_RECONFIG (Part 1 of 2)**

| Settings | Value |
|---|---|
| Which type of output file do you want to create? | **Verilog** |
| What name do you want for the output file? | **pll_reconfig_circuit.v** |
| Return to this page for another create operation | Turned off |
| Currently selected device family | **Stratix IV** |
| Match project/default | Turned on |
| Which scan chain type will you be using | **Top/Bottom** |
| Do you want to specify initial value of the scan chain? | Select **Yes, use this file for the content data** |

**Table 9. Configuration Settings for ALTPLL_RECONFIG (Part 2 of 2)**

| Settings | Value |
|---|---|
| File name | **the_pll_initial_mif.mif**<br>Ensure that this option shows the correct path of the **.mif** file before compiling the design to avoid scan chain mismatch warnings. |
| Do not use pre initialized RAM - initialize from ROM instead | Turned off |
| Add ports to write to the scan chain from external ROM during run time | Turned on |
| Generate netlist | Turned off |
| Variation file | Turned on |
| AHDL Include file | Turned on |
| VHDL component declaration file | Turned on |
| Quartus II symbol file | Turned on |
| Instantiation template file | Turned on |
| Verilog HDL block-box file | Turned on |

7. Click **Finish**. The `pll_reconfig_circuit` module is built.

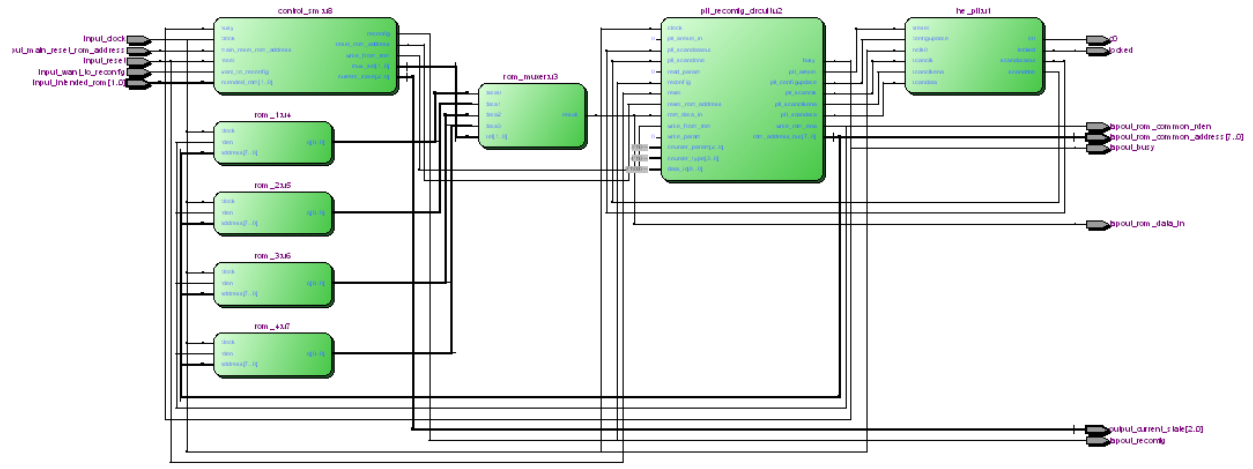
## Compiling ALTPLL and ALTPLL_RECONFIG

To compile the design in the Quartus II software, follow these steps:

1. To compile the design, on the Processing menu, click **Start Compilation**.
2. When the **Full Compilation was successful** message box appears, click **OK**.

You have now created and compiled the complete design file, which can be viewed in the RTL Viewer (Figure 26). To display the RTL Viewer, in the Tools menu, select **Netlist Viewers**, and click on **RTL Viewer**.

**Figure 26. Top-Level Design Implementation Using the RTL Viewer**



This design consists of eight modules, which are:

1. **the_pll:u1**—This represents the PLL (Top and Bottom PLL) that is to be reconfigured. The settings are as follows:
   - `inclk` = 50 MHz
   - `c0` = 100 MHz
2. **pll_reconfig_circuit:u2**—This represents the PLL reconfiguration circuitry used by the PLL to reconfigure its settings during user mode. In addition, this circuitry has a scan-chain cache, which contains the intended PLL settings to be reconfigured. It also has additional ports to take advantage of cycling multiple **.mif** files for reconfiguration from external ROMs. This design example demonstrates the capability of these ports. The settings are represented by the **the_pll_initial.mif** file. The settings are as follows:
   - `inclk` = 50 MHz
   - `c0` = 100 MHz
3. **rom_muxer:u3**—This represents a 4-to-1 multiplexer used to multiplex serial data coming from four ROMs to the `rom_data_in` port of the pll_reconfig_circuit module. The multiplexer is used because the `rom_data_in` port is 1 bit in size; however, it is controlled by a 2-bit selector, hence its ability to multiplex four signals.
4. **rom_1:u4**—This represents the external ROM, which contains the intended reconfiguration settings of the PLL. It has a 1-bit output port (`q`) because of the serial nature of writing the intended PLL settings to the scan-chain cache of the **pll_reconfig_circuit** module. It has a capacity of 256 words of 1-bit size. The ROM uses 256 words because that is the closest approximate size of the scan-chain file for this type of PLL, which is 234 bits. For this ROM, it is represented by the **the_pll_200_mhz.mif** file, which is 234 bits. The settings are as follows:
   - `inclk` = 50 MHz
   - `c0` = 200 MHz
5. **rom_2:u5**—This represents the external ROM, which contains the intended reconfiguration settings of the PLL. It has a 1-bit output port (`q`) because of the serial nature of writing the intended PLL settings to the scan-chain cache of the **pll_reconfig_circuit** module. It has a capacity of 256 words of 1-bit size. The ROM uses 256 words because that is the closest approximate size of the scan-chain file for this type of PLL, which is 234 bits. For this ROM, it is represented by the **the_pll_300_mhz.mif** file, which is 234 bits. The settings are as follows:
   - `inclk` = 50 MHz
   - `c0` = 300 MHz

6. **rom_3:u6**—This represents the external ROM, which contains the intended reconfiguration settings of the PLL. It has a 1-bit output port (`q`) because of the serial nature of writing the intended PLL settings to the scan-chain cache of the **pll_reconfig_circuit** module. It has a capacity of 256 words of 1-bit size. The ROM uses 256 words because that is the closest approximate size of the scan-chain file for this type of PLL, which is 234 bits. For this ROM, it is represented by the .mif file **the_pll_400_mhz.mif**, which is 234 bits. The settings are as follows:
   - `inclk` = 50 MHz
   - `c0` = 400 MHz
7. **rom_4:u7**—This represents the external ROM, which contains the intended reconfiguration settings of the PLL. It is has a 1-bit output port (`q`) because of the serial nature of writing the intended PLL settings to the scan-chain cache of the **pll_reconfig_circuit** module. It has a capacity of 256 words of 1-bit size. The ROM uses 256 words because that is the closest approximate size of the scan-chain file for this type of PLL, which is 234 bits. For this ROM, it is represented by the **.mif** file **the_pll_500_mhz.mif**, which is 234 bits. The settings are as follows:
   - `inclk` = 50 MHz
   - `c0` = 500 MHz
8. **control_sm:u8**—This represents the state machine that controls the three main processes involved in the PLL reconfiguration with multiple **.mif** files via external ROMs. The state machine selects the ROM to be reconfigured, initiates the writing of the ROM content to the scan-chain cache, and initiates the reconfiguration of the PLL using the written content in the scan-chain cache to the PLL. You can modify this simple state machine to suit your design needs.

Figure 27 shows the state diagram for the state machine.

**Figure 27. Control_sm Module State Diagram**

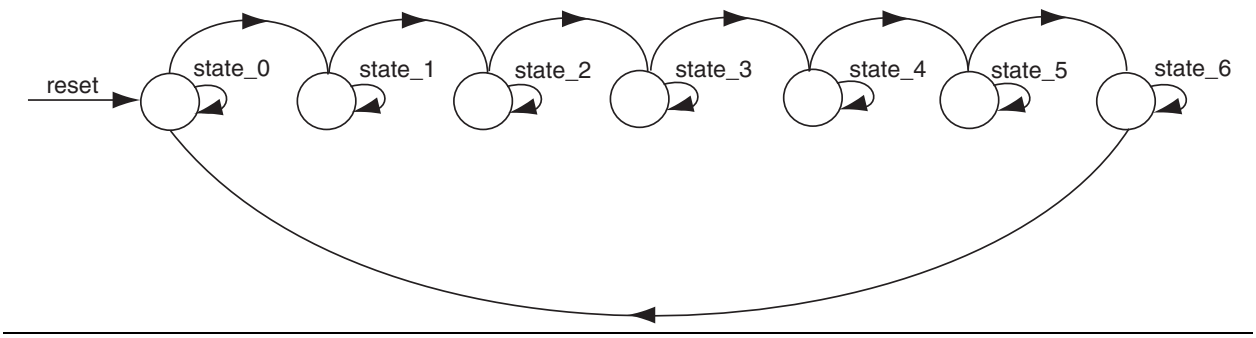


Figure 28 shows the corresponding condition for the state transition.

**Figure 28. Control_sm Module State Transition Conditions**

| | Source State | Destination State | Condition |
|---|---|---|---|
| 1 | state_0 | state_0 | (!want_to_reconfig) |
| 2 | state_0 | state_1 | (want_to_reconfig) |
| 3 | state_1 | state_2 | |
| 4 | state_2 | state_2 | (!busy) |
| 5 | state_2 | state_3 | (busy) |
| 6 | state_3 | state_3 | (busy) |
| 7 | state_3 | state_4 | (!busy) |
| 8 | state_4 | state_5 | |
| 9 | state_5 | state_5 | (!busy) |
| 10 | state_5 | state_6 | (busy) |
| 11 | state_6 | state_0 | (!busy) |
| 12 | state_6 | state_6 | (busy) |

Figure 29 shows how the whole state machine module (`control_sm`) is being implemented in the RTL Viewer.

**Figure 29. control_sm Design Implementation via the RTL Viewer**

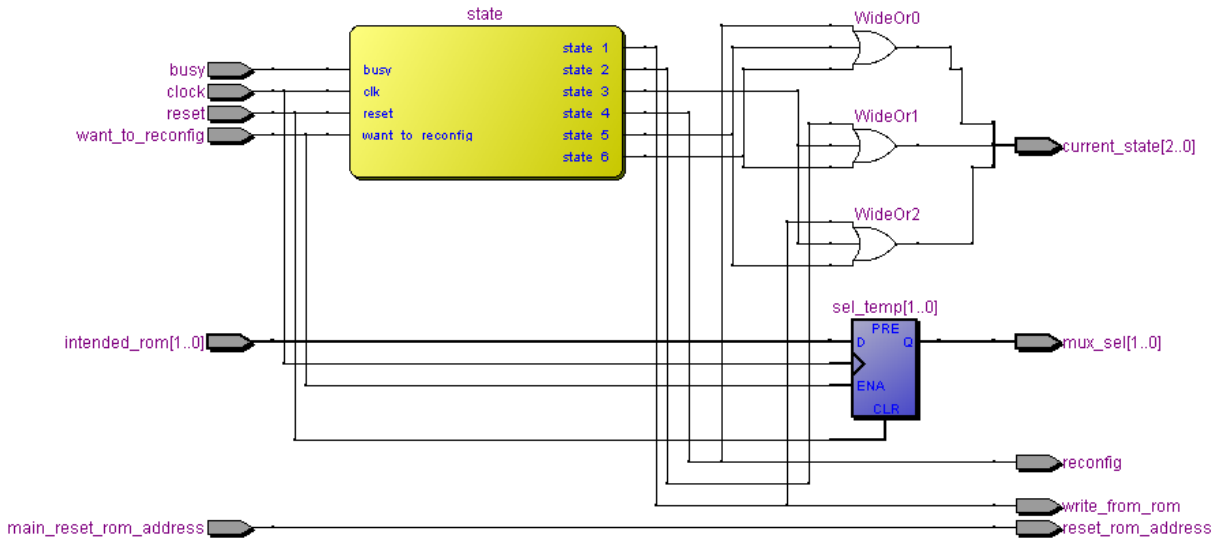



Figure 27 through Figure 29 show how the state machine's control paths and data paths are implemented. The next section describes the state machine behavior in detail.

## Simulating the Design Example

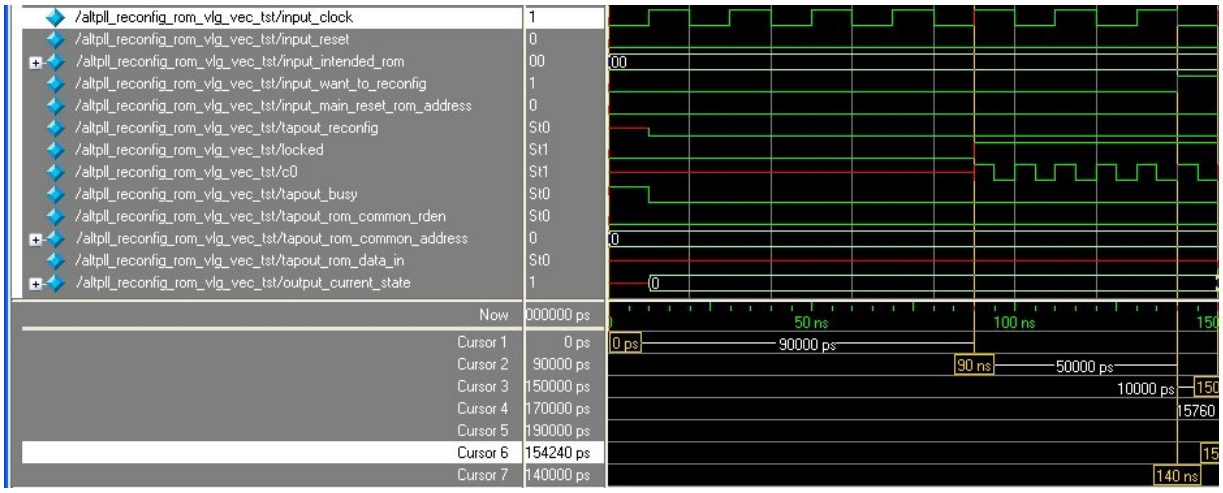
To simulate the design example using the ModelSim-Altera software, follow these steps:

1. Unzip the **altpll_reconfig_ex3_msim.zip** file to any directory on your PC.

2. Browse to the folder in which you unzipped the files.
3. Open **remote_update_ex2.do** file in a text editor.
4. In line 1 of the **altpll_reconfig_ex3_msim.do**, ensure that the directory path of the library files is correct. For example, C:/Modeltech_ae/altera/verilog/stratix
5. On the File menu, click **Save**.
6. Launch the ModelSim-Altera software.
7. On the File menu, click **Change Directory**.
8. Select the folder in which you unzipped the files.
9. Click **OK**.
10. On the Tools menu, click **Execute Macro**.
11. Select the **altpll_reconfig_ex3_msim.do** file and click **Open**. This is a script file for the ModelSim-Altera software to automate all of the necessary settings for the simulation.
12. Verify the results shown in the Wave window.

Figure 30 shows the simulation results when writing from ROM 1 to scan cache of the ALTPLL_RECONFIG IP core for the duration of 0 to 250 ns.

**Figure 30. Initial Writing from ROM 1 to the Scan Cache of ALTPLL_RECONFIG (0 to 250 ns)**




The simulation begins when the PLL gets locked (refer to Figure 30); the `locked` signal is asserted at 90 ns. The PLL output `c0` produces a 100 MHz clock. The original settings of the PLL have an input clock of 50 MHz and generates an output clock of 100 MHz.

The `output_current_state` signal is `0`, which shows the current state of the state machine that controls the PLL reconfiguration process from the external ROMs.

When the state machine is at `0` (indicated by the output_current_state signal), it is waiting for the assertion of the `input_want_to reconfig` signal together with the value of the `input_intended_rom [1:0]` signal, which is `00`. The state machine remains at this state until the above conditions are satisfied.

At 140 ns, the input_want_to_reconfig signal is asserted for 1 clock cycle and the input_intended_rom [1:0] signal is set to 00. The input_want_to_reconfig signal controls the write_from_rom signal of the ALTPLL_RECONFIG instantiation. This begins the process of writing the contents of the intended ROM to the scan cache of the ALTPLL_RECONFIG IP core. The input_intended_rom [1:0] signal is used to control the selector (sel [1:0] signal) of the multiplexer instantiation, which multiplexes the intended ROM contents (in this case, ROM 1) to the rom_data_in signal of the ALTPLL_RECONFIG instantiation.

At 150 ns, the state machine is at 1 (indicated by the output_current_state signal). This signifies that the input_want_to_reconfig signal has been asserted together with the value of the input_intended_rom [1:0] signal, which is 00. This causes the write_from_rom signal of the ALTPLL_RECONFIG instantiation to be asserted. This also causes the selector (sel [1:0] signal) of the multiplexer instantiation to multiplex the intended ROM contents (in this case, ROM 1) to the rom_data_in signal of the ALTPLL_RECONFIG instantiation.

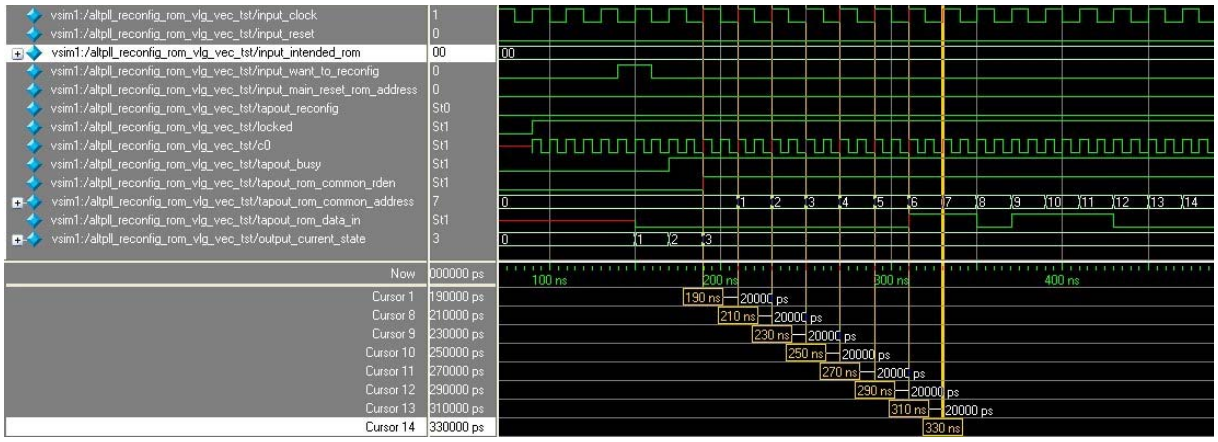
At 170 ns, the state machine is at 2 (indicated by the output_current_state signal). At this state, the state machine is waiting for the assertion of the tapout_busy signal, which signifies the busy signal of the ALTPLL_RECONFIG instantiation. The state machine tracks the busy signal because whatever operation the ALTPLL_RECONFIG is in (for example, read_param, write_param, reconfig, or write_from_rom) when asserted for 1 clock cycle, the busy signal is asserted for a particular duration. This indicates that the particular operation is being processed by the ALTPLL_RECONFIG instantiation. In this state, the tapout_busy signal has been asserted, signifying that the ALTPLL_RECONFIG instantiation has begun processing the write_from_rom operation.

At 190 ns, the state machine is at 3 (indicated by the output_current_state signal). This signifies that the tapout_busy signal has been asserted. At this point the state machine waits until the tapout_busy signal gets deasserted, to signify that the process of writing from the ROM to the scan cache of the ALTPLL_RECONFIG instantiation has been completed. Observe that the tapout_rom_common_rden signal has been asserted. This is the probed-out signal of the write_rom_ena signal, which is part of the ALTPLL_RECONFIG instantiation. This signal functions as the enable signal to the ROMs used in this design. Observe that the tapout_rom_common_address [7:0] signal begins changing value. This is the probed-out signal of the rom_address_out [7:0], which is part of the ALTPLL_RECONFIG instantiation. This signal controls which address of the ROM should be read out to the multiplexer instantiation. When the tapout_rom_common_rden signal is asserted together with the value of 0 for the tapout_rom_common_address [7:0] signal, it reads out the data from address 0 of the ROM 1 to the q port of the ROM, which is connected to the data_0 signal of the multiplexer. Then the data is multiplexed according to the selector of the multiplexer, and sent out to the rom_data_in port of the ALTPLL_RECONFIG instantiation. This port is probed out and is observed by the tapout_rom_data_in port. Therefore, the data from the intended ROM can be observed in simulation.

The tapout_rom_common_rden signal is asserted 1 clock cycle later, after the tapout_busy signal is asserted.

Figure 31 shows the simulation results when writing from ROM 1 to the scan cache of the ALTPLL_RECONFIG IP core for the duration of 60 to 580 ns.

**Figure 31. Initial Writing from ROM 1 to the Scan Cache of ALTPLL_RECONFIG (60 to 580 ns)**

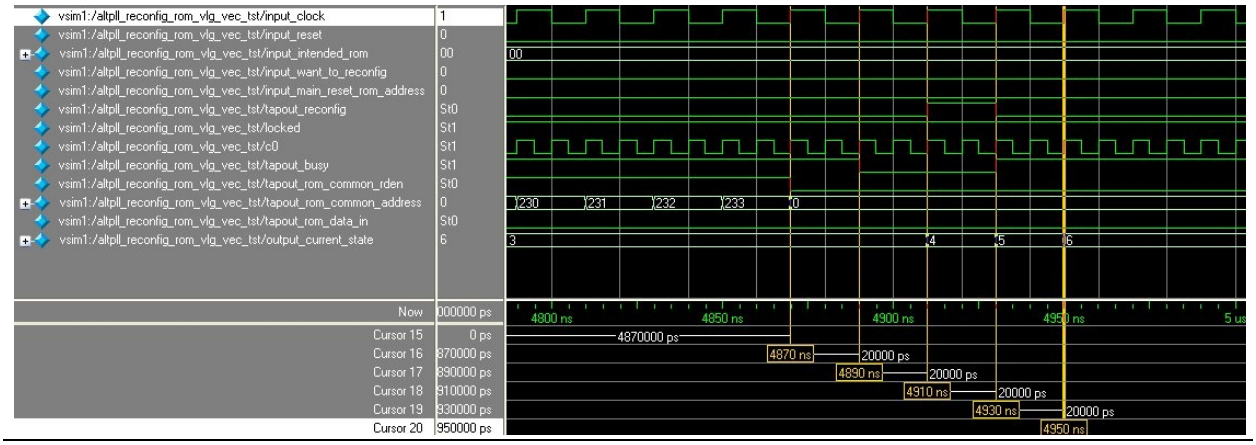



When data from the ROM is written to the scan cache of the ALTPLL_RECONFIG, it is done serially, meaning a 1-by-1 bit per cycle, from address `0` to address 233. You can observe this via the `tapout_rom_data_in` signal. The valid data read out from the ROM is delayed by 2 clock cycles.

When the simulation is at 270 ns, the `tapout_rom_common_rden` signal is asserted with the `tapout_rom_common_address [7:0]` signal with a value of 4. The data read out has a value of 1 in the **.mif** file in Figure 31. However, this value only appears when the `tapout_rom_common_rden` signal is asserted via the `tapout_rom_common_address [7:0]` signal with a value of 6 at 310 ns, implying a 2-clock-cycle delay.

This writing process continues until it has reached address `233`. Figure 32 shows the simulation results of the final process in writing the contents of ROM 1 to the scan cache of the ALTPLL_RECONFIG IP core.

**Figure 32. Ending Process of Writing the Contents of ROM 1 to the Scan Cache of ALTPLL_RECONFIG (4700 to 5040 ns)** *(1)*




**Note to Figure 32:**

(1) This figure also shows the initialization of the reconfiguration process.

At 4850 ns, the tapout_rom_common_rden signal is asserted with the tapout_rom_common_address [7:0] signal with a value of 233. This is the last address read out from ROM 1.

The data is available only 2 clock cycles later on the tapout_rom_data_in signal. The tapout_rom_common_rden signal and the tapout_busy signal are still asserted. The output_current_state signal still has a value of 3. It will remain in this state until the tapout_busy signal is deasserted.

At 4870 ns, the tapout_rom_common_rden signal is deasserted with the tapout_rom_common_address [7:0] signal with a value of 0. This stops the ROM 1 from reading out data; therefore, the address becomes 0. The tapout_rom_data_in signal still generates valid output data and the tapout_busy signal is asserted. The output_current_state signal still has a value of 3 .

At 4890 ns, the tapout_rom_common_rden signal remains deasserted with the tapout_rom_common_address [7:0] signal with a value of 0. Observe that the tapout_rom_data_in signal generates the last valid output data from ROM 1 (from address 233). The tapout_busy signal is deasserted and the output_current_state signal still has a value of 3.

At 4910 ns, the tapout_rom_common_rden signal remains deasserted with the tapout_rom_common_address [7:0] signal with a value of 0.

The tapout_rom_data_in signal no longer generates valid output data. The tapout_busy signal remains deasserted. The output_current_state signal changes value from 3 to 4. In this state, the state machine is initiating the reconfiguration process.

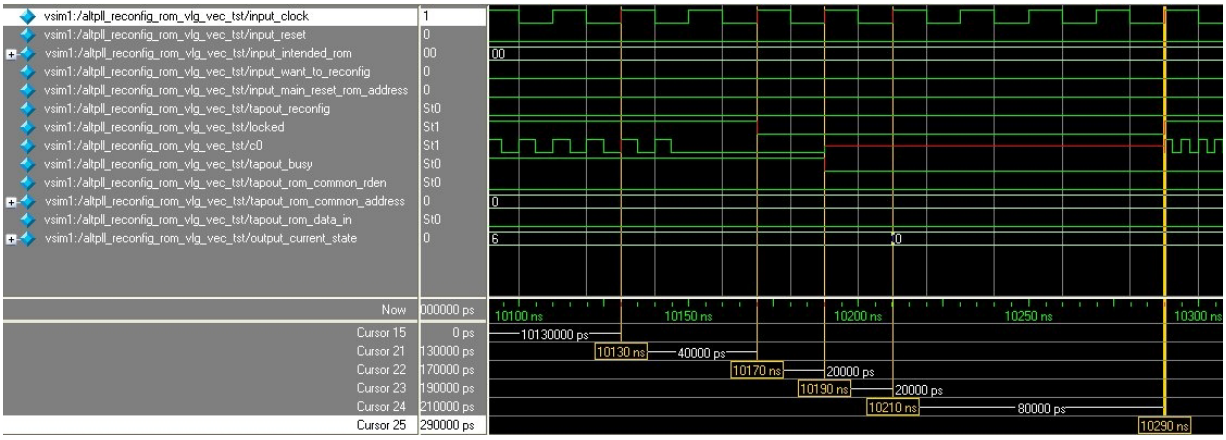
You can observe this state by the assertion of the tapout_reconfig signal for 1 clock cycle. This signal controls the reconfig signal of the ALTPLL_RECONFIG instantiation. When this signal is asserted for 1 clock cycle, the ALTPLL_RECONFIG instantiation begins the PLL reconfiguration process by using the settings written from ROM 1 to the scan cache of the ALTPLL_RECONFIG instantiation.

At 4930 ns, the tapout_reconfig signal is deasserted and the tapout_busy signal is asserted. This indicates that the state machine tracks the busy signal. The state machine tracks the busy signal because whatever operation the ALTPLL_RECONFIG is in (for example read_param, write_param, reconfig, write_from_rom) when asserted for 1 clock cycle, the busy signal is asserted for a particular duration. This indicates that the particular operation is being processed by the ALTPLL_RECONFIG instantiation. The output_current_state signal changes value from 4 to 5. It remains in this state until the tapout_busy signal is asserted.

At 4950 ns, the tapout_busy signal remains asserted. The output_current_state signal changes value from 5 to 6. It remains in this state until the tapout_busy signal is deasserted.

Figure 33 shows the final part of the reconfiguration process.

**Figure 33. PLL Reconfiguration (10,000 to 10,400 ns)** *(1)*




**Note to Figure 33:**

(1) From `c0` = 100 MHz to `c0` = 200 MHz.

At 10,130 ns, the `tapout_busy` signal remains asserted. The `output_current_state` signal remains at a value of `6`. Therefore, the state machine is still waiting for the `tapout_busy` signal to be deasserted. The `c0` signal is still at 100 MHz and the `locked` signal remains asserted, which means the PLL is still locked to a 100-MHz signal.

At 10,170 ns, the `tapout_busy` signal remains asserted. The `output_current_state` signal remains at a value of `6`. Therefore, the state machine is still waiting for the `tapout_busy` signal to be deasserted. The `locked` signal is deasserted, which means the PLL has lock loss, and the `c0` signal is at a `0` value and not producing a clock pulse.

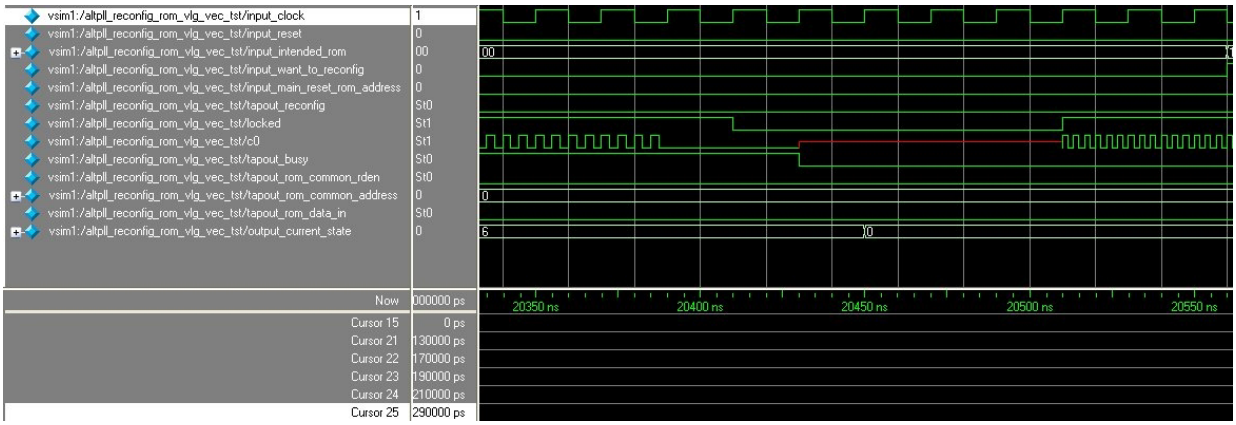
At 10,190 ns, the `tapout_busy` signal is deasserted. This means the PLL reconfiguration process is complete. The `output_current_state` signal remains at a value of `6`. The `locked` signal remains deasserted. The `c0` signal is an unknown value and still not producing a clock pulse.

At 10,210 ns, the `tapout_busy` signal remains deasserted. The `output_current_state` signal changes to a value of `0`. This is the original state, in which the state machine waits for the next reconfiguration from an external ROM. The `locked` signal remains deasserted. The `c0` signal is an unknown value and still does not produce a clock pulse.

At 10,290 ns, the `tapout_busy` signal remains deasserted. The `output_current_state` signal remains at a value of `0`. The `locked` signal is asserted. The `c0` signal now produces a 200-MHz clock signal, which is the intended setting from ROM 1.

Figure 34 through Figure 36 show the PLL reconfiguration process from the remaining ROMs (ROM 2, ROM 3, and ROM 4, respectively).

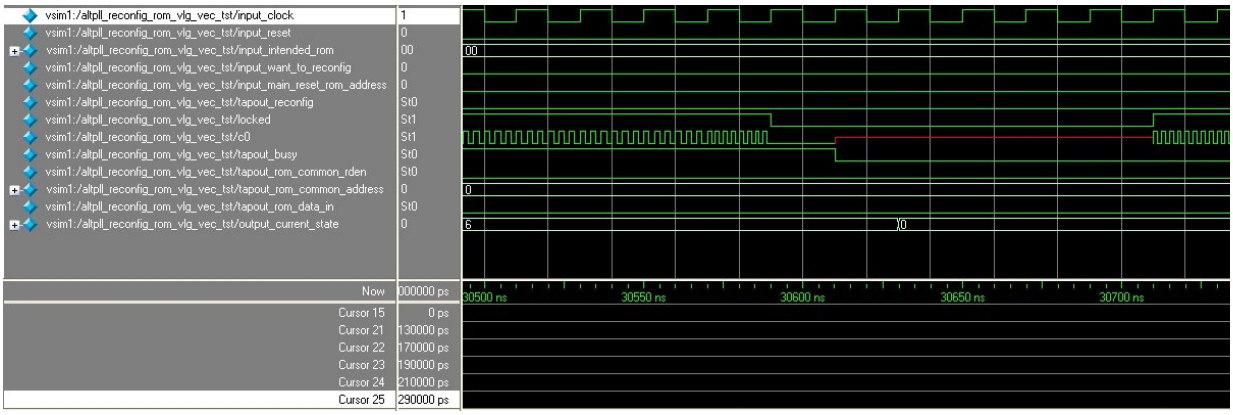
**Figure 34. PLL Reconfiguration from ROM 2 (20,330 to 20,600 ns)** *(1)*




**Note to Figure 34:**

(1) From c0 = 200 MHz to c0 = 300 MHz.

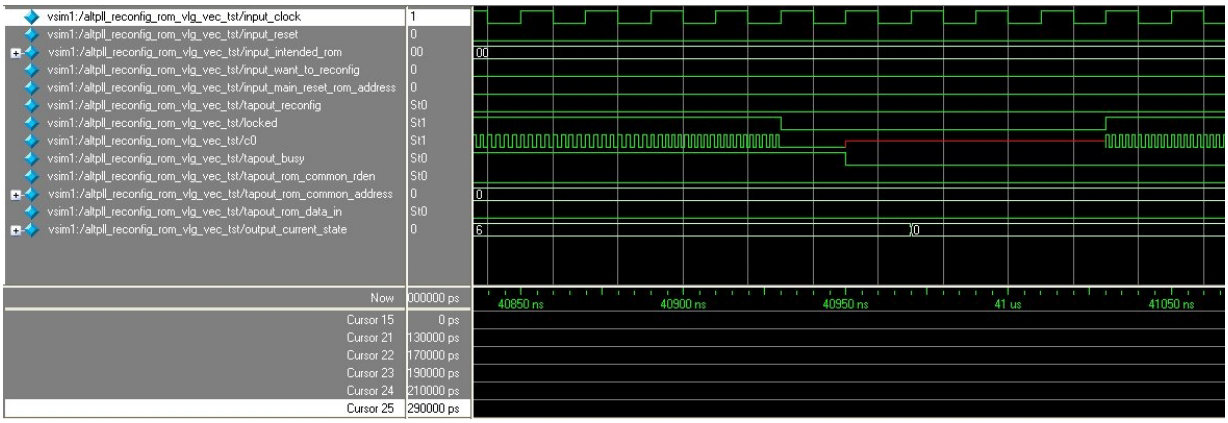
**Figure 35. PLL Reconfiguration from ROM 3 (30,480 to 30,750 ns)** *(1)*




**Note to Figure 35:**

(1) From c0 = 300 MHz to c0 = 400 MHz.

**Figure 36. PLL Reconfiguration from ROM 4 (40,800 to 41,090 ns)** *(1)*




**Note to Figure 36:**

(1) From c0 = 400 MHz to c0 = 500 MHz.

The next part of the simulation demonstrates the PLL reconfiguration from ROM 1 again, but highlights the ROM address resetting capabilities during writing from an external ROM to the scan cache of the ALTPLL_RECONFIG instantiation. The c0 signal is 500 MHz and is reconfigured to 100 MHz. Figure 37 shows the process of initiating writing the contents of ROM 1 to the ALTPLL_RECONFIG instantiation.

**Figure 37. Initial Writing from ROM 1 to the Scan Cache of ALTPLL_RECONFIG (41,070 to 41,330 ns)**

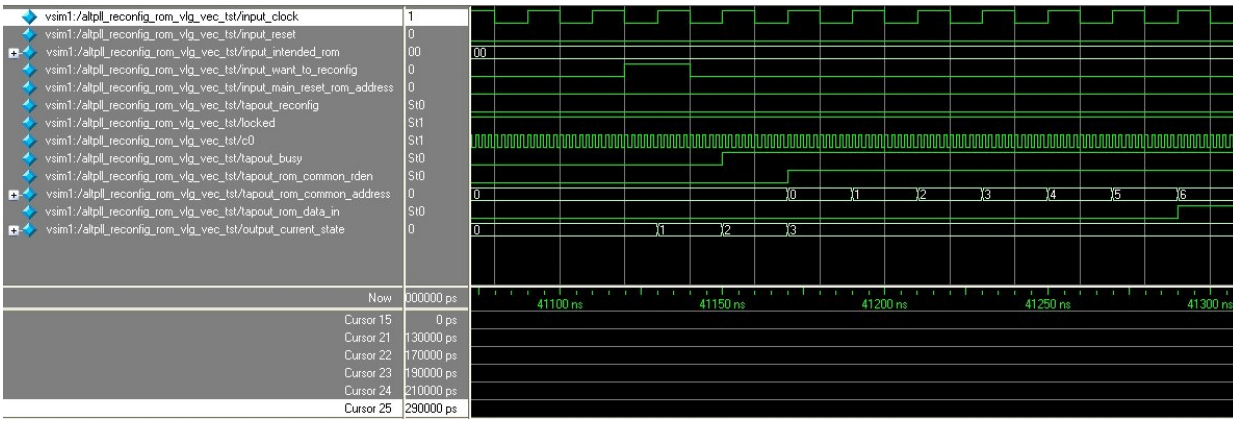
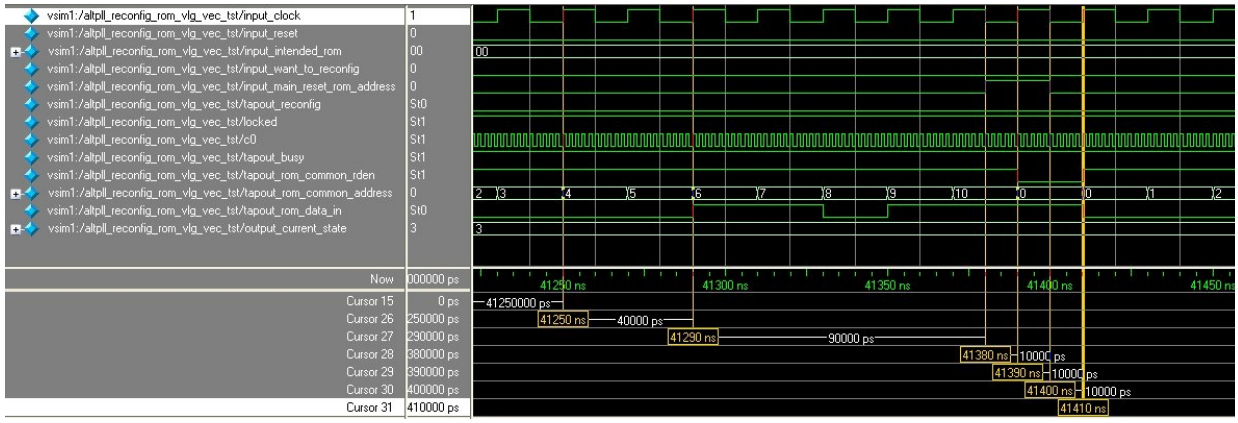


Figure 38 shows how the reset_rom_address capability is used.

**Figure 38. Resetting the Address when Writing from ROM 1 to the Scan Cache of ALTPLL_RECONFIG (41,200 to 41,450 ns)**




At 41,250 ns, the tapout_rom_common_rden signal is asserted with the tapout_rom_common_address [7:0] signal with a value of 4. Observe that the tapout_busy signal is asserted. The output_current_state signal is 3. The normal writing process from ROM to scan cache of the ALTPLL_RECONFIG instantiation continues.

At 41,290 ns, the tapout_rom_common_rden signal is asserted with the tapout_rom_common_address [7:0] signal with a value of 6. The tapout_rom_data_in signal contains the valid data from address 4 because of the 2-clock-cycle delay. The tapout_busy signal is asserted. The output_current_state signal is 3.

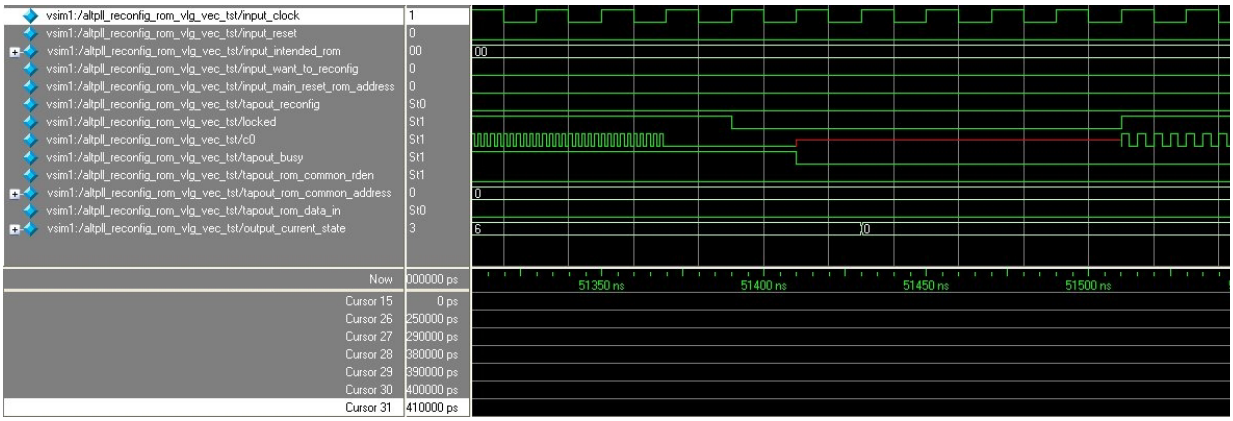
At 41,380 ns, the input_main_reset_rom_address is asserted for 1 clock cycle. This signal controls the reset_rom_address of the ALTPLL_RECONFIG instantiation. It resets the address counter in the ALTPLL_RECONFIG instantiation to 0.

At 41,390 ns, the tapout_rom_common_rden signal is deasserted for 1 clock cycle. The tapout_rom_common_address [7:0] signal is changed to a value of 0. The tapout_rom_data_in signal still generates the previous 2-clock-cycle output data (address 9), which is of value 1. The tapout_busy signal is still asserted. The output_current_state signal is 3.

At 41,410 ns, the tapout_rom_common_rden signal is reasserted. The tapout_rom_common_address [7:0] signal is changed to a value of 0. This restarts the writing of the contents of ROM 1 to the ALTPLL_RECONFIG scan cache from address 0. The tapout_busy signal is still asserted. The output_current_state signal is 3.

The PLL reconfiguration process continues as normal until the c0 signal is 200 MHz, as shown in Figure 39.

**Figure 39. PLL Reconfiguration From ROM1 (40,800 to 41,090 ns)** *(1)*




**Note to Figure 39:**

(1) From c0 = 500 MHz to c0 = 200 MHz.

You can modify the design to suit your requirements when attempting to reconfigure the PLL from multiple **.mif** files via external ROMs.

# Specifications

This section describes the prototypes, component declarations, ports, and parameters of the ALTPLL_RECONFIG IP core. These ports and parameters are available to customize the ALTPLL_RECONFIG IP core according to your application.

## Verilog HDL Prototype

The following Verilog HDL prototype is located in the Verilog Design File (**.v**) **altera_mf.v** in the <*Quartus II installation directory*>**\eda\synthesis** directory.

```
modulealtpll_reconfig
#( parameterintended_device_family = "unused",
   parameterinit_from_rom = "NO",
   parameterpll_type = "UNUSED",
   parameterscan_chain = "UNUSED",
   parameterscan_init_file = "UNUSED",
   parameteruse_scanclk_sync_register = "NO",
   parameterlpm_type = "altpll_reconfig",
   parameterlpm_hint = "unused")
(  outputwirebusy,
   inputwireclock,
   inputwire[2:0]counter_param,
   inputwire[3:0]counter_type,
   inputwire[8:0]data_in,
```

```
    outputwire[8:0]data_out,
    outputwirepll_areset,
    inputwirepll_areset_in,
    outputwirepll_configupdate,
    outputwirepll_scanaclr,
    outputwirepll_scanclk,
    outputwirepll_scanclkena,
    outputwirepll_scandata,
    inputwirepll_scandataout,
    inputwirepll_scandone,
    outputwirepll_scanread,
    outputwirepll_scanwrite,
    inputwireread_param,
    inputwirereconfig,
    inputwirereset,
    inputwirereset_rom_address,
    outputwire[7:0]rom_address_out,
    inputwirerom_data_in,
    inputwirewrite_from_rom,
    inputwirewrite_param,
    outputwirewrite_rom_ena)/* synthesis syn_black_box=1 */;

endmodule //altpll_reconfig
```

## VHDL Component Declaration

The following VHDL component declaration is located in the VHDL Design File (**.vhd**) **altera_mf_components.vhd** in the <*Quartus II installation directory*>**\libraries\vhdl\altera_mf** directory.

```
component altpll_reconfig
    generic (
        intended_device_family:string := "unused";
        init_from_rom:string := "NO";
        pll_type:string := "UNUSED";
        scan_chain:string := "UNUSED";
        scan_init_file:string := "UNUSED";
        use_scanclk_sync_register:string := "NO";
        lpm_hint:string := "UNUSED";
        lpm_type:string := "altpll_reconfig"
    );
    port(
        busy:  out std_logic;
```

```
        clock: in std_logic;
        counter_param:in std_logic_vector(2 downto 0) := (others =>
'0');
        counter_type:in std_logic_vector(3 downto 0) := (others =>
'0');
        data_in:in std_logic_vector(8 downto 0) := (others => '0');
        data_out:out std_logic_vector(8 downto 0);
        pll_areset:out std_logic;
        pll_areset_in:in std_logic := '0';
        pll_configupdate:out std_logic;
        pll_scanaclr:out std_logic;
        pll_scanclk:out std_logic;
        pll_scanclkena:out std_logic;
        pll_scandata:out std_logic;
        pll_scandataout:in std_logic := '0';
        pll_scandone:in std_logic := '0';
        pll_scanread:out std_logic;
        pll_scanwrite:out std_logic;
        read_param:in std_logic := '0';
        reconfig:in std_logic := '0';
        reset: in std_logic;
        reset_rom_address:in std_logic := '0';
        rom_address_out:out std_logic_vector(7 downto 0);
        rom_data_in:in std_logic := '0';
        write_from_rom:in std_logic := '0';
        write_param:in std_logic := '0';
        write_rom_ena:out std_logic
    );
end component;
```

## Ports and Parameters

This section describes the ports and parameters of the ALTPLL_RECONFIG IP core.

Table 10 lists the ALTPLL_RECONFIG IP core input ports.

**Table 10. ALTPLL_RECONFIG Input Ports (Part 1 of 3)**

| Port Name | Required? | Description |
|---|---|---|
| clock | Yes | Clock input for loading individual parameters. This signal also clocks the PLL during reconfiguration.<br>The clock input port must be connected to a valid clock.<br>Refer to the *DC and Switching Characteristics* chapter of the respective device handbooks for the clock $f_{MAX}$. |
| reset | Yes | Asynchronous reset input to the IP core.<br>Altera recommends that you reset this IP core before first use to guarantee that it is in a valid state. However, it does power up in the reset state. This port must be connected. |
| data_in[] | No | Data input that provides parameter value when writing parameters.<br>A 9-bit input port that provides the data to be written to the scan cache during a write operation. The bit width of the counter parameter to be written determines the number of bits of data_in[] that are read into the cache. For example, the low bit count of the C0 counter is 8-bit wide, so data_in[7..0] is read to the correct cache location. The bypass mode for the C0 counter is 1-bit wide, so data_in[0] is read for the value of this parameter. If omitted, the default value is 0. |
| counter_type[] | No | Specifies the counter type.<br>An input port in the form of a 4-bit bus that selects which counter type should be selected for the corresponding operation (read, write, or reconfig). The following table specifies the mapping between the counter_type value and the physical counter to be set. For details, refer to the following tables:<br>■ Table 13 on page 53 counter_type[3..0] settings for Stratix III, Stratix IV, and Cyclone III devices. |
| counter_param[] | No | Specifies the parameter for the value specified in the counter_type port.<br>An input port in the form of a 3-bit bus that selects which parameter for the given counter type should be updated. The mapping to each parameter type and the corresponding parameter bit-width are defined in the following tables:<br>■ Table 14 on page 54 counter_param[2..0] settings for Stratix IV. |

**Table 10. ALTPLL_RECONFIG Input Ports (Part 2 of 3)**

| Port Name | Required? | Description |
|---|---|---|
| read_param | No | Reads the parameter specified with the counter_type and counter_param ports from cache and fed to the data_out[] port.<br>When asserted, the read_param signal indicates that the scan cache should be read and fed to data_out[]. The bit location of the scan cache and the number of bits read and sent to data_out[] depend on the counter_type and counter_param values. The read_param signal is sampled at the rising clock edge. If it is asserted, the parameter value is read from the cache. Assert the read_param signal for 1 clock cycle only to prevent the parameter from being read twice.<br>The busy signal is asserted on the rising clock edge following the assertion of read_param. While the parameter is being read, the busy signal remains asserted. After the busy signal is deasserted, the value on data_out[] is valid and the next parameter can be loaded. While the busy signal is asserted, the value on data_out[] is not valid.<br>When the read_param signal is asserted, the busy signal is only asserted on the following rising edge of the clock and not on the same clock cycle as read_param. |
| write_param | No | Writes the parameter specified with the counter_type and counter_param ports to the cache with the value specified on the data_in[] port.<br>When asserted, the write_param signal indicates that the value on data_in[] should be written to the parameter specified by counter_type[] and counter_param[]. The number of bits read from the data_in[] port depends on the parameter. The write_param signal is sampled at the rising clock edge. If it is asserted, the parameter value is written to the cache. Assert the write_param signal for 1 clock cycle only to prevent the parameter from being written twice.<br>The busy signal is asserted on the rising clock edge following the assertion of write_param. While the parameter is being written, the busy signal remains asserted and input to data_in[] is ignored. After the busy signal is deasserted, the next parameter can be written.<br>When the write_param signal is asserted, the busy signal is only asserted on the following rising edge of the clock and not on the same clock cycle as write_param. |
| reconfig | Yes | Specifies that the phase-locked loop (PLL) should be reconfigured with the PLL settings specified in the current cache.<br>When asserted, the reconfig signal indicates that the PLL should be reconfigured with the values in the cache. The reconfig signal is sampled at the rising clock edge. If it is asserted, the cached settings are loaded in the PLL. Assert the reconfig signal for 1 clock cycle only to prevent reloading the PLL configuration. The busy signal is asserted on the rising clock edge following the assertion of reconfig. While the PLL is being loaded, the busy signal remains asserted. After the busy signal is deasserted, the parameter values can be modified again.<br>During and after reconfiguration, the scan chain data cache remains unchanged. This allows you to easily create a new set of reconfiguration settings that differs from the previous one in only one parameter.<br>If write_param has not been asserted since the previous assertion of reconfig, in Stratix II devices, the pll_scanwrite signal is pulsed during reconfiguration. This feature supports burst-phase stepping. However, in Stratix IV, Cyclone IV, Cyclone 10 LP, and Arria II GX devices, the entire scan chain is shifted in to the PLL again.<br>When the reconfig signal is asserted, the busy signal is only asserted on the following rising edge of the clock and not on the same clock cycle as reconfig. |

**Table 10. ALTPLL_RECONFIG Input Ports (Part 3 of 3)**

| Port Name | Required? | Description |
|---|---|---|
| pll_areset_in | No | Input signal indicating that the PLL should be reset.<br>When asserted, the pll_areset_in signal indicates the PLL IP core should be reset. This port defaults to 0 if left unconnected. When the ALTPLL_RECONFIG IP core is used in a design, you cannot reset the PLL in any other way; you must use this IP core port to manually reset the PLL. |
| pll_scandone | No | Input port for the ALTPLL_RECONFIG IP core that signals update done from the PLL module. This port is driven by the PLL's scandone output signal and determines when the PLL is reconfigured.<br>For Stratix IV devices, pll_scandone is asserted on most operations, and deasserted as soon as it detects the assertion of configupdate. Upon the completion of all PLL reconfiguration updates, scandone is reasserted. |
| pll_scandataout | Yes | Input port driven by the scandataout signal from the ALTPLL IP core. It can be used to read the current configuration of the ALTPLL IP core.<br>This input port holds the ALTPLL IP core scan data output from the dynamically reconfigurable bits. The pll_scandataout port must be connected to the scandataout port of the PLL. The activity on this port can only be observed when the reconfig signal is asserted. |
| rom_data_in | No | Serial data input to the PLL from the ROM.<br>This 1-bit port allows the external ROM to be serially written into the scan cache of the ALTPLL_RECONFIG IP core. The value on this port is valid 2 clock cycles after the write_rom_ena signal is asserted and remains valid until 2 clock cycles after the write_rom_ena signal is deasserted. This port is available for Stratix IV, Cyclone IV, Cyclone 10 LP, and Arria II GX devices only. |
| write_from_rom | No | Specifies that the scan chain must be written from the external ROM.<br>When asserted, this signal tells the ALTPLL IP core to write the scan cache from the external ROM. At the next rising clock edge, the ALTPLL_RECONFIG IP core asserts the write_rom_ena signal and begins placing valid ROM addresses on the rom_address_out port. Assert the write_from_rom signal for 1 clock cycle only. This port is available for Stratix IV, Cyclone IV, Cyclone 10 LP, and Arria II GX devices only. |
| reset_rom_address | No | Resets the ROM address.<br>When asserted, this signal tells the ALTPLL IP core to restart the read from ROM at address 0. Assert the reset_rom_address signal for 1 clock cycle only. This port is available for Stratix IV, Cyclone IV, Cyclone 10 LP, and Arria II GX devices only. |

Table 11 lists the ALTPLL_RECONFIG IP core output ports.

**Table 11. ALTPLL_RECONFIG Output Ports (Part 1 of 2)**

| Port Name | Required? | Description |
|---|---|---|
| data_out[] | No | Data read from the cache when read_param is asserted.<br>9-bit output bus that provides the parameter data to the user. When the read_param signal is asserted, the values on counter_type[] and counter_param[] determine the parameter value that is loaded from cache and driven on the data_out[] bus. When the IP core deasserts the busy signal, the appropriate bits of the bus (for example, [0] or [3..0]) hold a valid value. |
| busy | No | Indicates when the PLL is reading or writing a parameter to the cache, or is configuring the PLL.<br>While the busy signal is asserted, no parameters can be read or written, and no reconfiguration can be initiated. Changes to the IP core can be made only when the busy signal is not asserted. The signal goes high when the read_param, write_param, or reconfig input port is asserted, and remains high until the specified operation is complete. In the case of a reconfiguration operation, the busy signal remains high until the pll_areset signal is asserted and then deasserted. |
| pll_areset | Yes | Drives the areset port on the PLL to be reconfigured.<br>The pll_areset port must be connected to the areset port of the ALTPLL IP core for the reconfiguration to function correctly. This signal is active high. pll_areset is asserted when pll_areset_in is asserted, or, after reconfiguration, at the next rising clock edge after the scandone signal goes high. If you use the ALTPLL_RECONFIG IP core, drive the PLL areset port using the pll_areset output port. |
| pll_configupdate | No | Drives the configupdate port on the PLL to be reconfigured.<br>When asserted, the pll_configupdate port loads selected data to PLL configuration latches. The signal is asserted after the final data bit is sent out. This port is available for Stratix IV, Cyclone IV, Cyclone 10 LP, and Arria II GX devices only. |
| pll_scanclk | Yes | Drives the scanclk port on the PLL to be reconfigured.<br>For information about the maximum scanclk frequency for the various devices, refer to the respective device handbook. |
| pll_scanclkena | No | This acts as a clock enable for the scanclk port on the PLL to be reconfigured.<br>Reconfiguration begins on the first rising edge of pll_scanclk after pll_scanclkena is asserted. On the first falling edge of pll_scanclk, after the pll_scanclkena signal is deasserted, the IP core stops scanning data to the PLL. |
| pll_scanaclr | Yes | Drives the scanaclr port on the PLL to be reconfigured.<br>Not available for Stratix IV, Cyclone IV, Cyclone 10 LP, and Arria II GX devices. |
| pll_scanread | No | Drives the scanread port on the PLL to be reconfigured.<br>Not available for Stratix IV, Cyclone IV, Cyclone 10 LP, and Arria II GX devices. |
| pll_scanwrite | No | Drives the scanwrite port on the PLL to be reconfigured.<br>Not available for Stratix IV, Cyclone IV, Cyclone 10 LP, and Arria II GX devices. |
| pll_scandata | Yes | Drives the scandata port on the PLL to be reconfigured.<br>This output port from the IP core holds the scan data input to the PLL for the dynamically reconfigurable bits. The pll_scandata port sends scandata to the PLL. Any activity on this port can only be observed when the reconfig signal is asserted. |

**Table 11. ALTPLL_RECONFIG Output Ports (Part 2 of 2)**

| Port Name | Required? | Description |
|---|---|---|
| rom_address_out | No | Specifies the address in ROM from which data is written to the scan chain.<br>During the write operation, the address increments from address 0 to the size of the scan cache. The rom_address_out port is valid only while the write_rom_ena port is asserted. Each address is asserted for 1 clock cycle. This port is available for Stratix IV, Cyclone IV, Cyclone 10 LP, and Arria II GX devices only. |
| write_rom_ena | No | Enables the ROM.<br>When the write_rom_ena signal is asserted, the ALTPLL_RECONFIG IP cores generates valid addresses on the rom_address_out port. This port is available for Stratix IV, Cyclone IV, Cyclone 10 LP, and Arria II GX devices only. |

Table 12 lists the ALTPLL_RECONFIG IP core parameters.

**Table 12. ALTPLL_RECONFIG Parameters**

| Parameter | Type | Required? | Description |
|---|---|---|---|
| init_from_rom[] | String | No | Specifies initialization from ROM. Initializes on power-up and not after every reset (to match RAM **.mif** file behavior). The available values are YES and NO. If omitted, the default value is NO. |
| pll_type | String | No | Specifies the type of PLL to instantiate. For Stratix IV, Cyclone IV, Cyclone 10 LP, Arria II GX, devices, values are TOP_BOTTOM and LEFT_RIGHT. |
| scan_init_file | String | No | Specifies the name of the **.mif** or **.hex** used as the initial value of the scan chain cache. Values are *<filename>* and UNUSED. If omitted, the value for every entry in the cache is 0. You must set the input file as shown in the following table:<br>**Device** — **Input File Size**<br>Stratix series devices (except Stratix IV devices) — 192 bits or 288 bits depending on the scan_chain length<br>Stratix IV Top/Bottom PLL devices — 234 bits<br>Stratix IV Right/Left PLL devices — 180 bits<br>Cyclone IV and Cyclone 10 LP devices — 144 bits<br>Arria II GX devices — 180 bits |
| use_scanclk_sync_register | String | No | Specifies whether to use the scanclk port for the synchronization mode for the register. Available for all supported devices except Stratix IV, Cyclone IV, Cyclone 10 LP, and Arria II GX devices. Values are YES and NO. If omitted, the default value is NO. |

Table 13 lists the counter_type settings for Stratix IV, Cyclone IV, Cyclone 10 LP, and Arria II GX devices.

**Table 13. counter_type[3..0] Settings (Part 1 of 2)**

| Counter Selection | Binary | Decimal |
|---|---|---|
| N | 0000 | 0 |
| M | 0001 | 1 |
| CP/LF | 0010 | 2 |

**Table 13. counter_type[3..0] Settings (Part 2 of 2)**

| Counter Selection | Binary | Decimal |
|---|---|---|
| VCO | 0011 | 3 |
| C0 (1) | 0100 | 4 |
| C1 (1) | 0101 | 5 |
| C2 (1) | 0110 | 6 |
| C3 (1) | 0111 | 7 |
| C4 (1) | 1000 | 8 |
| C5 (2) | 1001 | 9 |
| C6 (2) | 1010 | 10 |
| C7 (3) | 1011 | 11 |
| C8 (3) | 1100 | 12 |
| C9 (3) | 1101 | 13 |
| Illegal value | 1110 | 14 |
| Illegal value | 1111 | 15 |

**Notes to Table 13:**

(1) For Stratix IV Top/Bottom PLL, Stratix IV Left/Right PLL, and Cyclone IV and Cyclone 10 LP PLL.

(2) For Stratix IV Top/Bottom PLL, Stratix IV Left/Right PLL, and Arria II GX PLL.

(3) For Stratix IV Top/Bottom PLL only.

Table 14 lists the counter_param settings for Stratix IV, Cyclone IV, Cyclone 10 LP, and Arria II GX devices.

**Table 14. counter_param[2..0] Settings (Part 1 of 2)**

| Counter Type | Counter Param | Binary | Decimal | Width (bits) |
|---|---|---|---|---|
| Regular counters<br>C0-C9: Top-Bottom Stratix IV<br>C0-C6: Left-Right Stratix IV<br>C0-C4: Cyclone IV and Cyclone 10 LP<br>C0-C6: Arria II GX | High count | 000 | 0 | 8 |
| | Low count | 001 | 1 | 8 |
| | Bypass | 100 | 4 | 1 |
| | Mode (odd/even division) | 101 | 5 | 1 |
| CP/LF | Charge pump unused | 101 | 5 | 5 |
| | Charge pump current | 000 | 0 | 3 |
| | Loop filter unused | 100 | 4 | 1 |
| | Loop filter resistor | 001 | 1 | 5 |
| | Loop filter capacitance | 010 | 2 | 2 |
| VCO | VCO Post Scale | 000 | 0 | 1 |

**Table 14. counter_param[2..0] Settings (Part 2 of 2)**

| Counter Type | Counter Param | Binary | Decimal | Width (bits) |
|---|---|---|---|---|
| M/N Counters | High count | 000 | 0 | 8 |
| | Low count | 001 | 1 | 8 |
| | Bypass | 100 | 4 | 1 |
| | Mode (odd/even division) | 101 | 5 | 1 |
| | Nominal count | 111 | 7 | 9 |

**Note to Table 13:**

(1) For even nominal count, the counter bits are automatically set as follows:

- `high_count` = `Nominalcount`/2
- `low_count`= `Nominalcount`/2

For odd nominal count, the counter bits are automatically set as follows:

- `high_count` = (`Nominalcount` + 1)/2
- `low_count` = `Nominalcount` - `high_count`
- odd/even division bit = 1

For nominal count = 1, Bypass bit = 1.

# Document Revision History

Table 15 lists the revision history for this document.

**Table 15. Document Revision History (Part 1 of 2)**

| Date | Version | Changes |
|---|---|---|
| June 2017 | 2017.06.16 | Added support for Cyclone 10 LP devices. |
| August 2014 | 2014.08.18 | ■ Added information about legacy parameter editor GUI and output directories.<br>■ Added information about licensing and upgrading IP cores.<br>■ Added information about editing existing IP variations. |
| July 2014 | 2014.07.07 | ■ Replaced MegaWizard Plug-In Manager information with IP Catalog.<br>■ Added standard information about upgrading IP cores.<br>■ Added standard installation and licensing information.<br>■ Removed outdated device support level information. IP core device support is now available in IP Catalog and parameter editor. |
| February 2012 | 6.0 | ■ Added information about nominal count usage in Table 12.<br>■ Updated the decimal value in Table 12. |
| August 2010 | 5.0 | Updated for the Quartus II software version 10.0. |

**Table 15. Document Revision History (Part 2 of 2)**

| Date | Version | Changes |
|---|---|---|
| July 2008 | 4.0 | ■ Added support for Stratix IV devices.<br>■ Updated screenshots for "MegaWizard Plug-In Manager Page Descriptions" section.<br>■ Updated "Features" section.<br>■ Updated "General Description" section.<br>■ Added "Simulation" section.<br>■ Added "Checking Design Violations With the Design Assistant" section<br>■ Added comment for pll_scanclk port.<br>■ Added support for new port, pllena. |
| November 2007 | 3.2 | ■ Updated for the Quartus II software version 7.2, including: Added new user-mode reconfiguration from external ROM.<br>■ Added three new input ports: rom_data_in, write_from_rom, and reset_rom_address<br>■ Added two new output ports: rom_address_out and write_rom_ena<br>■ Updated design example format<br>■ Updated: Figure 5, Figure 6, Figure 7, Figure 17, Figure 18, Figure 19, Figure 20, Figure 21, Figure 22<br>■ Changed altpll_reconfig to ALTPLL_RECONFIG throughout the document. |
| March 2007 | 3.1 | Added Cyclone III to list of supported devices. |
| December 2006 | 3.0 | Updated for Quartus II software version 6.1. |
| October 2006 | 2.0 | Updated for Quartus II software version 6.0. |
| April 2005 | 1.0 | Initial release. |