RAM-Based Shift Register (ALTSHIFT_TAPS) IP Core

User Guide

Subscribe

This document describes the Altera-provided a RAM-based shift register (ALTSHIFT_TAPS) megafunction IP core. This IP core contains features not found in a conventional shift register. Traditional shift registers implemented with standard flip-flops use many logic cells for large shift registers. The ALTSHIFT_TAPS IP core is implemented in the device memory blocks, saving logic cells and routing resources. In a complicated design such as a digital signal processing (DSP) application that requires local data storage, it is more efficient to implement an ALTSHIFT_TAPS IP core as the shift register.

The ALTSHIFT_TAPS IP core is a parameterized shift register with taps. The taps provide data outputs from the shift register at certain points in the shift register chain. You can add additional logic that uses the output from these taps for further applications. The IP core's output tap feature is useful for applications such as the Linear Feedback Shift Register (LFSR) and Finite Impulse Response (FIR) filters.

Features

The ALTSHIFT_TAPS IP core implements a shift register with taps and offers additional features, which include:

- Selectable RAM block type
- A wide range of widths for the `shiftin` and `shiftout` ports
- Support for output taps at certain points in the shift register chain
- Selectable distance between taps

General Description

Use the IP Catalog (**Tools > IP Catalog**) and parameter editor to easily configure the IP core. The ALTSHIFT_TAPS IP core is implemented in the embedded memory block of all supported device families with simple dual-port RAM. You can select the RAM block type according to the capacity you require. The capacity that is represented by the width and the depth of the memory block depends on the `TAP_DISTANCE`, `NUMBER_OF_TAPS`, and `WIDTH` parameters of the ALTSHIFT_TAPS IP core.

For the features and capacities of the typical memory block, refer to the chapter of your device handbook that contains information about TriMatrix embedded memory blocks.

The ALTSHIFT_TAPS IP core supports single-bit and multiple-bit data shifting at one clock cycle, depending on the width of the `shiftin` and `shiftout` ports. For example, if the `shiftin` and `shiftout` ports are single-bit data, only one bit is shifted per clock cycle. If the `shiftin` and `shiftout` ports are multiple-bit data, such as one-word data (8-bit), the whole word is shifted per clock cycle.

The IP core also supports output taps at certain points in the shift register chain, but the tap points must be evenly spaced. You set the space between taps in the parameter editor.

Figure (a) in Figure 1–1 shows a traditional 12-word-depth shift register. Figure (b) shows how the data in the shift register chain are being tapped at even spaces (1st, 4th, 7th, and 10th) at the output taps of the ALTSHIFT_TAPS IP core.

Figure 1–1. Tapping Data at Certain Points of the Shift Register Chain *(Note 1)*, *(2)*, *(3)*
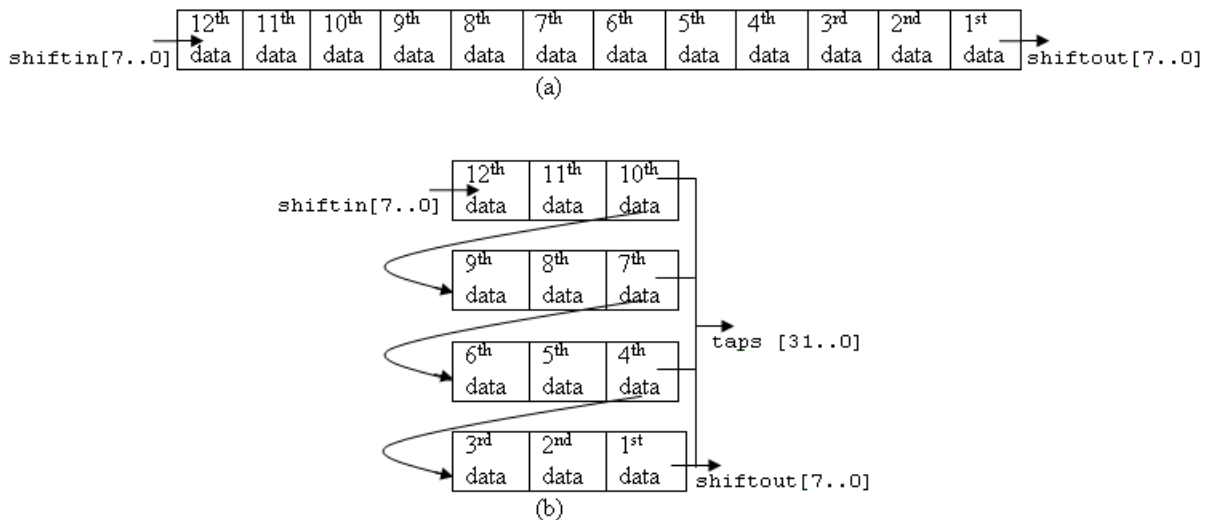
Notes for Figure 1–1

(1) The ALTSHIFT_TAPS IP core depicted here has `TAP_DISTANCE` = 3 and `NUMBER_OF_TAPS` = 4.

(2) The tapped data is output to `taps[31..0]`. Note that `taps[31..0]` is a 32-bit output because it taps four words at one time. The first word from the MSB of the taps (`taps[31..24]`) represents the first data and is followed by the 4th data, 7th data, and 10th data.

(3) The `shiftout[7..0]` word is equivalent to `taps[31..24]`.

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 2–1. IP core Installation Path
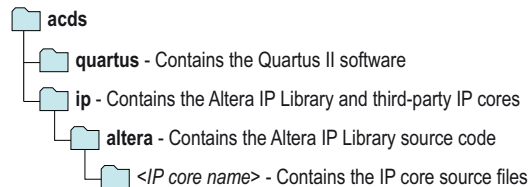
☞ The default installation directory on Windows is **<drive>:\altera\<version number>**; on Linux it is **<home directory>/altera/<version number>**.

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 2–2. Quartus II IP Catalog
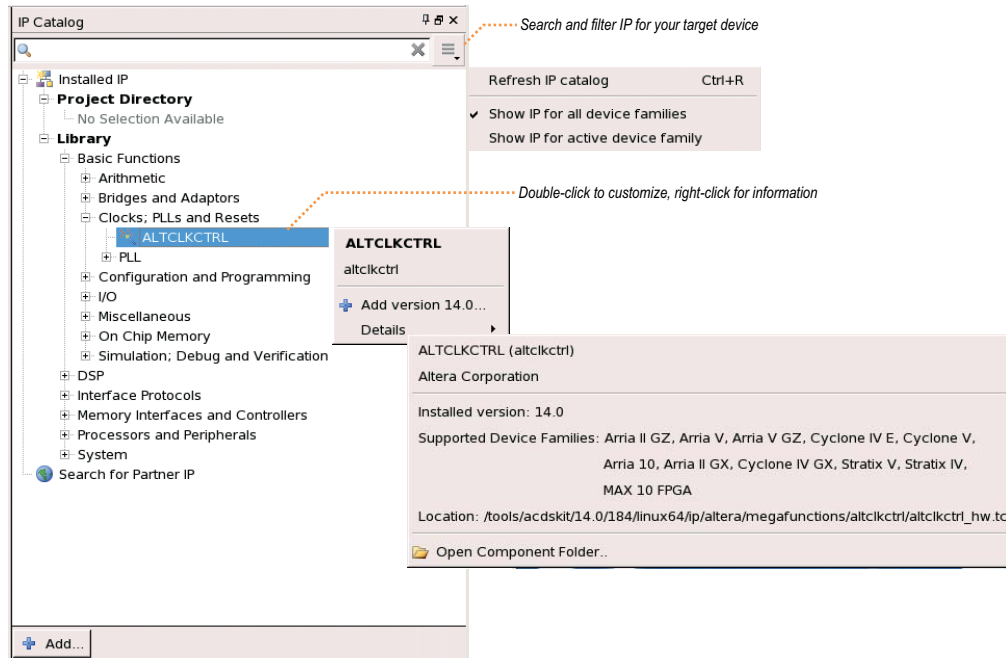
☞ The IP Catalog is also available in Qsys (**View > IP Catalog**). The Qsys IP Catalog includes exclusive system interconnect, video and image processing, and other system-level IP that are not available in the Quartus II IP Catalog.
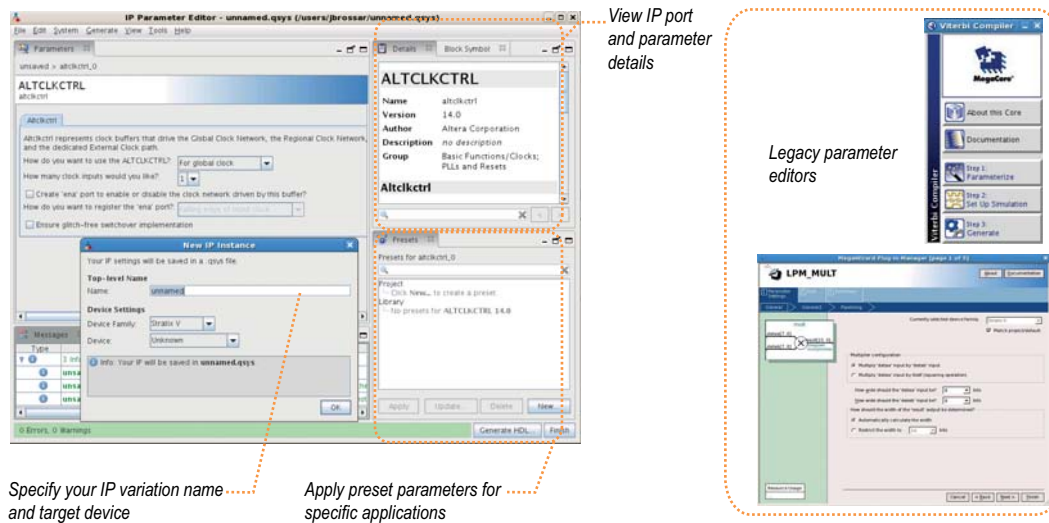
Using the Parameter Editor

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 2–3. IP Parameter Editors

View IP port and parameter details

Legacy parameter editors

Specify your IP variation name and target device

Apply preset parameters for specific applications

Specifying IP Core Parameters and Options

The parameter editor GUI allows you to quickly configure your custom IP variation. Use the following steps to specify IP core options and parameters in the Quartus II software. Refer to Specifying IP Core Parameters and Options (Legacy Parameter Editors) for configuration of IP cores using the legacy parameter editor.

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 for your custom IP variation. The parameter editor saves the IP variation settings in a file named .*<your_ip>***qsys**. Click **OK**.

3. Specify the parameters and options for your IP variation in the parameter editor, including one or more of the following. Refer to your IP core user guide for information about specific IP core parameters.

    ■ Optionally select preset parameter values if provided for your IP core. Presets specify initial parameter values for specific applications.

    ■ Specify parameters defining the IP core functionality, port configurations, and device-specific features.

    ■ Specify options for processing the IP core files in other EDA tools.

4. Click **Generate HDL**, the **Generation** dialog box appears.

5. Specify output file generation options, and then click **Generate**. The IP variation files generate according to your specifications.

6. To generate a simulation testbench, click **Generate > Generate Testbench System**.

7. To generate an HDL instantiation template that you can copy and paste into your text editor, click **Generate > HDL Example**.

8. Click **Finish**. The parameter editor adds the top-level **.qsys** file to the current project automatically. If you are prompted to manually add the **.qsys** file to the project, click **Project > Add/Remove Files in Project** to add the file.

9. After generating and instantiating your IP variation, make appropriate pin assignments to connect ports.
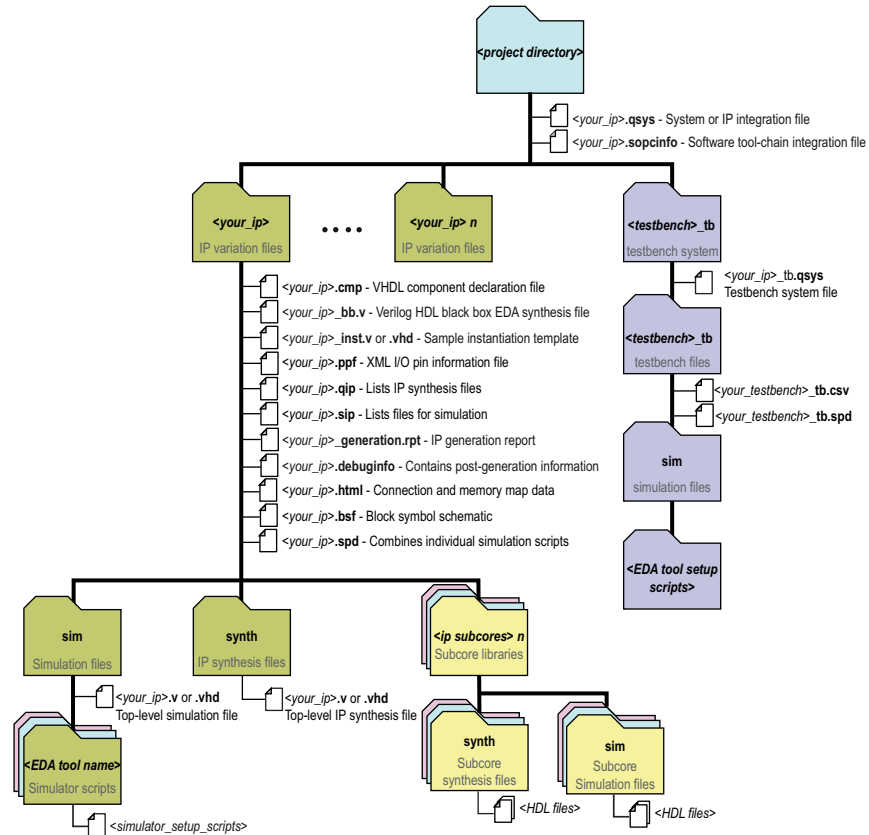
For information about using a legacy parameter editor, refer to "Specifying IP Core Parameters and Options (Legacy Parameter Editors)" in the *Introduction to Altera IP Cores*.

Files Generated for Altera IP Cores

The Quartus II software version 14.0 Arria 10 Edition and later generates the following output file structure for Altera IP cores.

Figure 2–4. IP Core Generated Files

Table 2–1. IP Core Generated Files (version 14.0a10)

| File Type	Description
<my_ip>.**qsys**	The Qsys system or top-level IP variation file. *<my_ip>* is the name that you give your IP variation.
<system>.**sopcinfo**	Describes the connections and IP component parameterizations in your Qsys system. You can parse its contents to get requirements when you develop software drivers for IP components. Downstream tools such as the Nios II tool chain use this file. The **.sopcinfo** file and the **system.h** file generated for the Nios II tool chain include address map information for each slave relative to each master that accesses the slave. Different masters may have a different address map to access a particular slave component.
<my_ip>.**cmp**	The VHDL Component Declaration (**.cmp**) file is a text file that contains local generic and port definitions that you can use in VHDL design files.
<my_ip>.**html**	A report that contains connection information, a memory map showing the address of each slave with respect to each master to which it is connected, and parameter assignments.

Table 2–1. IP Core Generated Files (version 14.0a10)

File Type	Description
*<my_ip>*_**generation .rpt**	IP or Qsys generation log file. A summary of the messages during IP generation.
<my_ip>.**debuginfo**	Contains post-generation information. Used to pass System Console and Bus Analyzer Toolkit information about the Qsys interconnect. The Bus Analysis Toolkit uses this file to identify debug components in the Qsys interconnect.
<my_ip>.**qip**	Contains all the required information about the IP component to integrate and compile the IP component in the Quartus II software.
<my_ip>.**csv**	Contains information about the upgrade status of the IP component.
<my_ip>.**bsf**	A Block Symbol File (**.bsf**) representation of the IP variation for use in Quartus II Block Diagram Files (**.bdf**).
<my_ip>.**spd**	Required input file for ip-make-simscript to generate simulation scripts for supported simulators. The **.spd** file contains a list of files generated for simulation, along with information about memories that you can initialize.
<my_ip>.**ppf**	The Pin Planner File (**.ppf**) stores the port and node assignments for IP components created for use with the Pin Planner.
*<my_ip>*_**bb.v**	You can use the Verilog black-box (_**bb.v**) file as an empty module declaration for use as a black box.
<my_ip>.**sip**	Contains information required for NativeLink simulation of IP components. You must add the .sip file to your Quartus II project.
*<my_ip>*_**inst.v** or **_inst.vhd**	HDL example instantiation template. You can copy and paste the contents of this file into your HDL file to instantiate the IP variation.
<my_ip>.**regmap**	If IP contains register information, **.regmap** file generates. The **.regmap** file describes the register map information of master and slave interfaces. This file complements the **.sopcinfo** file by providing more detailed register information about the system. This enables register display views and user customizable statistics in the System Console.
<my_ip>.**svd**	Allows HPS System Debug tools to view the register maps of peripherals connected to HPS within a Qsys system. During synthesis, the **.svd** files for slave interfaces visible to System Console masters are stored in the **.sof** file in the debug section. System Console reads this section, which Qsys can query for register map information. For system slaves, Qsys can access the registers by name.
<my_ip>.**v** or *<my_ip>*.**vhd**	HDL files that instantiate each submodule or child IP core for synthesis or simulation.
mentor/	Contains a ModelSim® script **msim_setup.tcl** to set up and run a simulation.
aldec/	Contains a Riviera-PRO script rivierapro_setup.tcl to setup and run a simulation.
/synopsys/vcs **/synopsys/vcsmx**	Contains a shell script **vcs_setup.sh** to set up and run a VCS® simulation. Contains a shell script **vcsmx_setup.sh** and **synopsys_ sim.setup** file to set up and run a VCS MX® simulation.
/cadence	Contains a shell script ncsim_setup.sh and other setup files to set up and run an NCSIM simulation.
/submodules	Contains submodule HDL files and /sim and /synth directories.

Specifying IP Core Parameters and Options (Legacy Parameter Editors)

The Quartus II software version 14.0 and previous uses a legacy version of the parameter editor for IP core 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 in the Quartus II software version 14.0a10 and later.

Figure 2–5. Legacy Parameter Editors
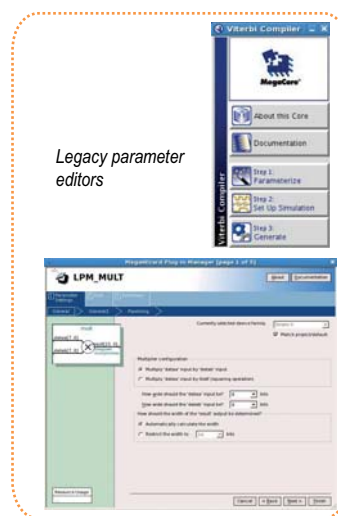
1. In the IP Catalog (**Tools > IP Catalog)**, locate and double-click the name of the IP core to customize. The parameter editor appears.

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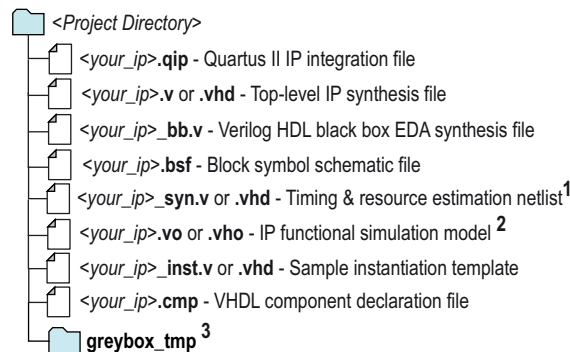
☞ To manually add an IP variation generated with legacy parameter editor to a project, click **Project > Add/Remove Files in Project** and add the IP variation **.qip** file.

👣 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 version 14.0 and previous parameter editor generates the following output file structure for Altera IP cores:

Figure 2–6. IP Core Generated Files (Legacy Parameter Editor)

```
📁 <Project Directory>
   📄 <your_ip>.qip - Quartus II IP integration file
   📄 <your_ip>.v or .vhd - Top-level IP synthesis file
   📄 <your_ip>_bb.v - Verilog HDL black box EDA synthesis file
   📄 <your_ip>.bsf - Block symbol schematic file
   📄 <your_ip>_syn.v or .vhd - Timing & resource estimation netlist[1]
   📄 <your_ip>.vo or .vho - IP functional simulation model [2]
   📄 <your_ip>_inst.v or .vhd - Sample instantiation template
   📄 <your_ip>.cmp - VHDL component declaration file
   📁 greybox_tmp [3]
```

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 2–2. 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 Outdated 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 2–3. 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 2–7. Upgrading IP Cores

Upgrades all IP core that support "Auto Upgrade"

Upgrades individual IP cores unsupported by "Auto Upgrade"

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*.

The following table shows the parameter settings available for this IP core:

Table 3–1. Shift Register (RAM-based) Parameters

Configuration Setting	Description
How wide should the 'shiftin' input and the 'shiftout' output buses be?	Specify the width of the data input and output buses. This value is represented by the term w in the Shift Register Memory Configuration.
How many taps would you like?	Specify the number of taps. This value is represented by the term n in the Shift Register Memory Configuration.
Create groups for each tap output	Turn on this option to create separate groups for output data tapped from the register chain. *(3)*
How wide should the distance between taps be?	Specify the distance between taps. This value is represented by the term m in the Shift Register Memory Configuration. *(4)*
Create a clock enable port	Turn on this option to create an enable signal for register ports. The register ports are always enabled if this option is not turned on. *(5)*
Create an asynchronous clear port	Turn on this option to create an asynchronous clear signal. When asserted, the outputs of the shift register are immediately cleared.
What should the RAM block type be?	Choose the type of memory block that supports the feature, memory configuration, and capacity for your application. *(6)*

Notes for Table 3–1

(1) The widths of the shiftin input bus and shiftout output bus are identical, and they are not registered. However, the output data can be considered synchronous with the clock because the internal read address to the memory block is synchronous to the clock.

(2) The width of the output taps is the multiplication of w (width of input data) and n (number of taps). Also, the word from the MSB of the output taps is equivalent to the shiftout output bus.

(3) The combination of these groups represent the taps[wn-1:0] bus.

(4) The distance between taps, m, must be at least 3.

(5) The registered port is referred to as the internal register at the memory address ports. The shiftin and shiftout ports are not registered.

(6) For information about the chosen memory block type, refer to the TriMatrix Embedded Memory Block chapter of your target device handbook. You can also choose AUTO if you are not particular about the RAM block type used. With the AUTO option, the memory block type is determined by the Quartus II software synthesizer or Fitter at compile time. To determine the type of memory block used, check the Quartus II Fitter Report.

This chapter describes the prototypes, declarations, ports, and parameters of the ALTSHIFT_TAPS IP core. You can use the ports and parameters to customize the ALTSHIFT_TAPS IP core according to your application.

Verilog HDL Prototype for the ALTSHIFT_TAPS

You can locate the following Verilog HDL prototype in the Verilog Design File (**.v**) **altera_mf.v** in the *<Quartus II installation directory>***eda****synthesis** directory.

```
module     altshift_taps
#(         parameter          intended_device_family = "unused",
           parameter          number of taps = 1,
           parameter          power_up_state = "CLEARED",
           parameter          taps_distance = 1,
           parameter          width = 1,
           parameter          lpm_type = "altshift_taps",
           parameter          lpm_hint = "unused")
(          input wire         aclr,
           input wire         clken,
           input wire         clock,
           input wire         [width-1:0]    shiftin,
           output wire        [width-1:0]    shiftout,
           output wire        [width*number_of_taps-1:0]    taps)/*synthesis syn_black_box=1 */;
endmodule \\altshift_taps
```

VHDL Component Declaration for the ALTSHIFT_TAPS

You can locate the following VHDL Design File (**.vhd**) **altera_mf.vhd** in the *<Quartus II installation directory>***libraries****bhdl****altera_mf** directory.

```
component altshift_taps
       generic (
               intended_device_family :    string := "unused";
               number_of_taps :       natural;
               power_up_state :       string := "CLEARED";
               tap_distance   :       natural;
               width  :        natural;
               lpm_hint       :       string := "UNUSED";
               lpm_type       :       string := "altshift_taps"
       );
       port(
               aclr   :        in std_logic := '0';
               clken  :        in std_logic := '1';
               clock  :        in std_logic;
               shiftin:        in std_logic_vector(width-1 downto 0);
               shiftout       :        out std_logic_vector(width-1 downto 0);
               taps   :        out std_logic_vector(width*number_of_taps-1 downto 0)
       );
end component;
```

VHDL Library-Use Declaration

The VHDL LIBRARY-USE declaration is not required if you use the VHDL component declaration.

```
LIBRARY alterea_mf;
USE altera_mf.altera_mf_components.all;
```

Ports and Parameters for the ALTSHIFT_TAPS

Figure 4–1 below shows the ports and parameters for the ALTSHIFT_TAPS IP core.

The parameter details are only relevant when implementing the IP core directly in HDL.

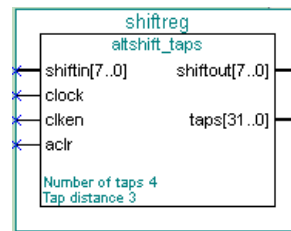**Figure 4–1. Shift Register (RAM-based) Ports and Parameters**

Table 4–1 shows the input ports of the ALTSHIFT_TAPS IP core.

Table 4–1. Shift Register (RAM-based) Input Ports

Name	Required	Description
shiftin[]	Yes	Data input to the shifter. Input port WIDTH bits wide.
clock	Yes	Positive-edge triggered clock.
clken	No	Clock enable for the clock port. clken defaults to V_{CC}.
aclr	No	Asynchronously clears the contents of the shift register chain. The shiftout outputs are cleared immediately upon the assertion of the aclr signal.

Table 4–2 shows the output ports of the ALTSHIFT_TAPS IP core.

Table 4–2. Shift Register (RAM-based) Output Ports

Name	Required	Description
shiftout[]	Yes	Output from the end of the shift register. Output port WIDTH bits wide.
taps[]	Yes	Output from the regularly spaced taps along the shift register. Output port WIDTH * NUMBER_OF_TAPS wide. This port is an aggregate of all the regularly spaced taps (each WIDTH bits) along the shift register.

Table 4–3 shows the ALTSHIFT_TAPS IP core parameters.

Table 4–3. Shift Register (RAM-based) Parameters

Name	Type	Required	Description	
NUMBER_OF_TAPS	Integer	Yes	Specifies the number of regularly spaced taps along the shift register.	
TAP_DISTANCE	Integer	Yes	Specifies the distance between the regularly spaced taps in clock cycles. This number translates to the number of RAM words that will be used. TAP_DISTANCE must be at least 3.	
WIDTH	Integer	Yes	Specifies the width of the input pattern.	
POWER_UP_STATE	String	No	Specifies the shift register contents at power-up. Values are CLEARED and DONT_CARE. If omitted, the default is CLEARED.	
			Value	**Description**
			CLEARED	Zero content. For Stratix and Stratix II device families, you must use M512 or M4K RAM blocks.
			DONT_CARE	Unknown contents. M-RAM blocks can be used with this setting.

Design Example: Shift Register with Taps

The objective of this design example is to implement and instantiate an ALTSHIFT_TAPS IP core using the IP Catalog and parameter editor. This example uses a shift register with a data width, *w*, of 8 bits, a taps distance, *m*, of 3, and the number of taps, *n*, equal to 4. It also demonstrates how you can tap the data at specific points from the shift register chain.

Design Files

The example design files are available in the User Guides section on the Literature page of the Altera® website (www.altera.com).

Configuration Settings

Refer to Chapter 2, Using Altera IP Cores to define a shift register function with the following parameters.

Table 5–1. Shift Register (RAM-Based) Configuration Settings

Configuration Setting	Value
How wide should the '`shiftin`' input and the '`shiftout`' output buses be?	8 bits
How many taps would you like?	4
Create groups for each tap output	Selected
How wide should the distance between taps be?	3
Create a clock enable port	Selected
Create an asynchronous clear port	Selected
What should the RAM block type be?	Auto

Functional Simulation in the ModelSim-Altera Simulator

Simulate the design in the ModelSim®-Altera software to generate a waveform display of the device behavior.

You should be familiar with the ModelSim-Altera software before trying the design example. If you are unfamiliar with the ModelSim-Altera software, refer to the support page for software products on the Altera website (www.altera.com). On the support page, there are links to such topics as installation, usage, and troubleshooting.

Set up and simulate the design in the ModelSim-Altera software by performing the following steps.

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

2. Start the ModelSim-Altera software.

3. On the File menu, click **Change Directory**.

4. Select the folder in which you unzipped the files.

5. Click **OK**.

6. On the Tools menu, click **Execute Macro**.

7. Select the **DE_ALTSHIFT_TAPS.do** file and click **Open**. The **DE_ALTSHIFT_TAPS.do** file is a script file for the ModelSim-Altera software to automate all the necessary settings for the simulation.

View the simulation results in the Wave window. Figure 5–1 shows the expected simulation results in the ModelSim-Altera software.

Figure 5–1. Simulation Waveform for Shift Register with Taps Design Example
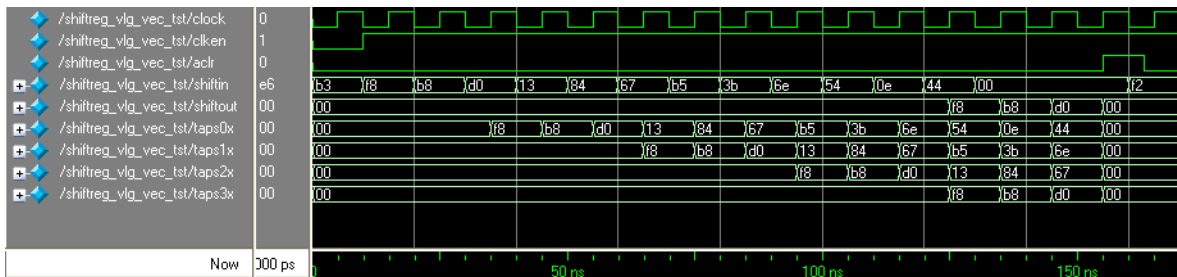
Understanding the Simulation Results

In this example, you configured the shift register to have the following properties:

■ 8-bit data width

■ Distance between taps (taps length) equals to 3

■ Number of taps equals to 4

■ Created groups for each tap output

■ Created a clock-enable signal and an asynchronous-clear signal

This example shows how you can tap the 1st-4th-7th-10th data words simultaneously (followed by the 2nd-5th-8th-11th and 3rd-6th-9th-12th) when all 12 words of data are shifted into the shift register.

Figure 5–2 shows the shift register chain that is analogous to the configuration you set in the ALTSHIFT_TAPS IP core in this example.

Figure 5–2. Shift Register Chain Analogy to Configured ALTSHIFT_TAPS
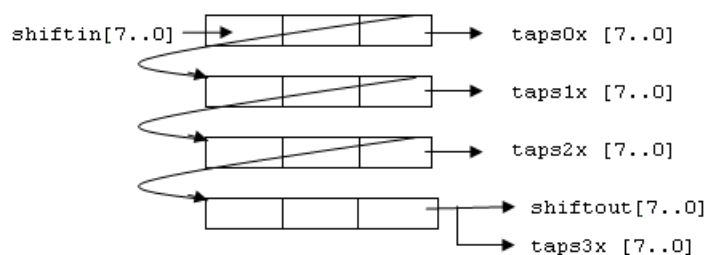
The next section uses this shift register chain to explain the shifting operation and the output operation of the ALTSHIFT_TAPS IP core.

Figure 5–3 shows the first three data words written into the shift register chain, shifted in the register chain, and the first data shown at the taps0x output.

Figure 5–3. First Three Data Written and Shifted in the Shift Register
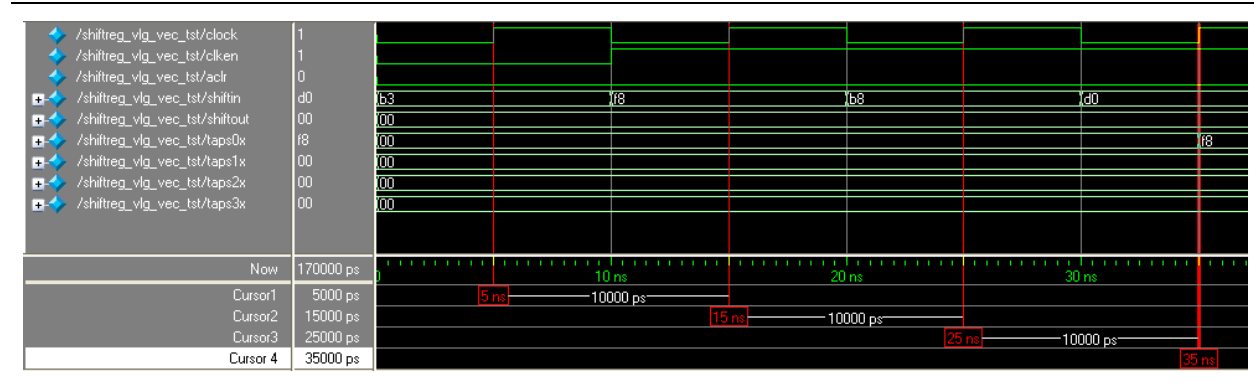
At 5 ns, the clken signal is low and therefore no operation is executed. You can consider 15 ns to be the first rising clock edge, as this is when the operation begins. The first data F8 is shifted into the shift register as shown in Figure 5–4. All outputs show 00 because no data is being shifted to any of the outputs.

Figure 5–4. Content of the Shift Register Chain at 15 ns
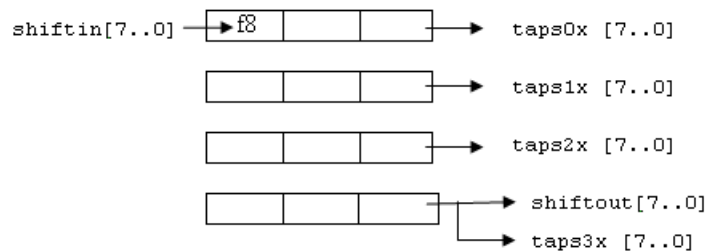
At 25 ns and 35 ns, the second data B8 and the third data D0 are shifted into the shift register, respectively.

☞ The existing data in the shift register chain are shifted right before the shift-in of new data.

Figure 5–5 shows the content in the shift register chain at 35 ns. All of the outputs show 00 except taps0x, which shows the first data, F8.

☞ None of the input and output data ports are registered. Only the address ports of the memory block within the shift register are registered. Therefore, when the data are shifted to any of the output ports, the data are shown immediately at the respective output ports.

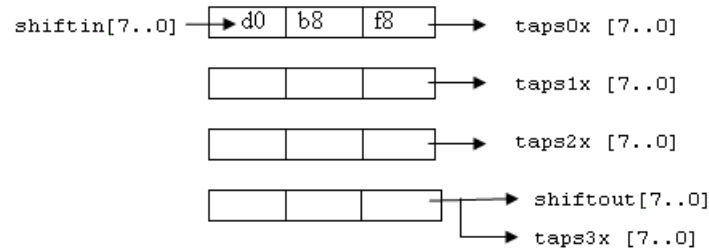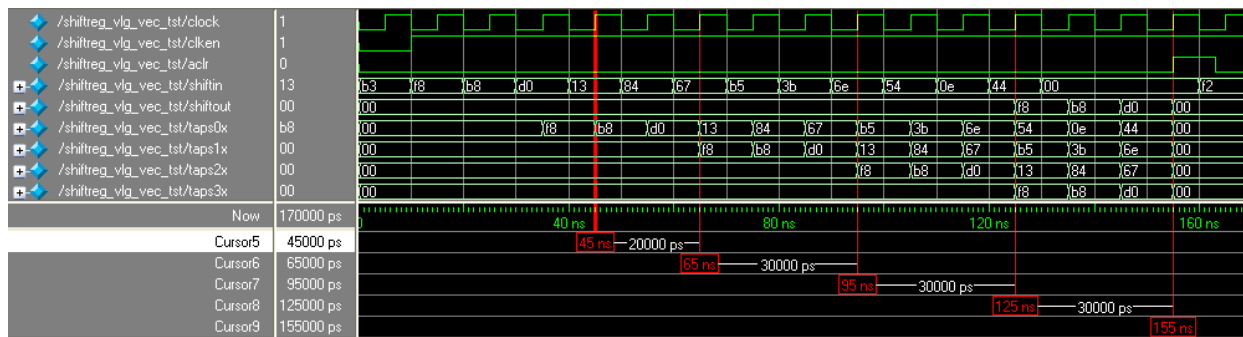**Figure 5–5. Content of the Shift Register Chain at 35 ns**

Figure 5–6 shows the data-shifting and output-tapping from the shift register chain at evenly-spaced intervals.

Figure 5–6. Data-Shifting and Output-Tapping

At 45 ns, the first data `F8` is shifted to the next row of taps and the second data `B8` is shifted to `taps0x`, as shown in Figure 5–7. Other output ports continue to show `00`. Also, at the same rising clock edge, the new data `13` is shifted into the shift register.

**Figure 5–7. Content of the Shift Register Chain at 45 ns**
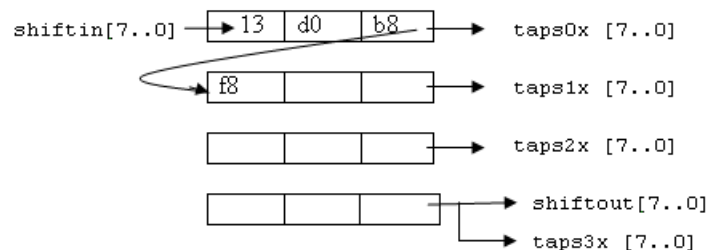
At 65 ns, the first data `F8`, and the fourth data `13` are shifted to `taps1x` and `taps0x`, respectively. At 95 ns, the first data `F8`, the fourth data `13`, and the seventh data `B5` are shifted to `taps2x`, `taps1x`, and `taps0x`, respectively. Finally, at 125 ns, all twelve data words are shifted into the shift register. You can then start to tap the 1st-4th-7th-10th data words simultaneously, from `taps3x`, `taps2x`, `taps1x`, and `taps0x`, respectively.

☞ The `shiftout` output port is equivalent to `taps3x` and both ports generate the same output data.

At the following rising clock edge, you can tap the 2nd-5th-8th-11th data words, followed by the 3rd-6th-9th-12th data words at the next rising edge. Figure 5–8 shows the contents for the shift register chain when all twelve words are being shifted into the shift register.

Figure 5–8. Content of the Shift Register Chain at 125 ns
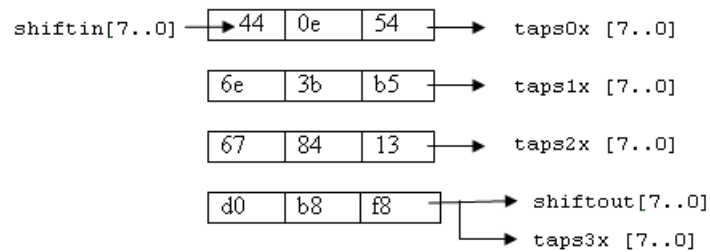
After you have tapped out all the data at 155 ns, you can assert the `aclr` signal to immediately clear all the data at the output ports and the contents of the shift register. You can then start to shift in another twelve words of data.

☞ This design example shows you how the shifting and tapping operation works. It is not meant to show a specific application usage. You can use the tapping feature with additional logic to suit your needs.

This chapter provides additional information about the document and Altera.

Document Revision History

The following table shows the revision history for this document.

Date	Document Version	Changes Made
2014.08.18	2014.08.18	■ Added information about specifying parameters for IP cores targeting Arria 10 devices. ■ Added information about the latest IP output for Quartus II version 14.0a10 targeting Arria 10 devices. ■ Added information about individual migration of IP cores to the latest devices. ■ Added information about legacy parameter editor GUI and output directories. ■ Added information about editing existing IP variations.
2014.06.30	3.0	■ Replaced MegaWizard Plug-In Manager information with IP Catalog. ■ Added standard information about upgrading IP cores. ■ Added standard installation and licensing information. ■ Removed outdated device support level information. IP core device support is now available in IP Catalog and parameter editor. ■ Removed all references to obsolete SOPC Builder tool.
May 2013	2.2	Updated to include Arria V, Cyclone V, and Stratix V devices.
November 2010	2.1	■ Updated ports and parameters ■ Added prototype and component declarations
July 2008	2.0	■ Updated the list of device families supported by this megafunction ■ Created a new design example with explanations showing the features and behaviors of the megafunction ■ Added the description for the new input pin, aclr ■ Reorganized the whole document
March 2007	1.2	Added Cyclone® III support
December 2006	1.1	Added Stratix® III support
September 2006	1.0	Initial release

Typographic Conventions

The following table shows the typographic conventions this document uses.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Indicate document titles. For example, *Stratix IV Design Guidelines*.
italic type	Indicates variables. For example, $n + 1$.
	Variable names are enclosed in angle brackets (< >). For example, *<file name>* and *<project name>***.pof** file.
Initial Capital Letters	Indicate keyboard keys and menu names. For example, the Delete key and the Options menu.
"Subheading Title"	Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, "Typographic Conventions."
Courier type	Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix n denotes an active-low signal. For example, `resetn`.
	Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`.
	Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`).
↵	An angled arrow instructs you to press the Enter key.
1., 2., 3., and a., b., c., and so on	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ■ ■	Bullets indicate a list of items when the sequence of the items is not important.
☞	The hand points to information that requires special attention.
?	A question mark directs you to a software help system with related information.
👣	The feet direct you to another document or website with related information.
⚠ CAUTION	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
⚠ WARNING	A warning calls attention to a condition or possible situation that can cause you injury.
✉	The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents.