

Introduction

Designing for Altera® Programmable Logic Devices (PLDs) is very similar, both in concept and in practice, to designing for Xilinx PLDs. In most cases, you can simply import your register transfer level (RTL) into Altera's Quartus® II software and begin compiling your design to the target device. This document will demonstrate the similar flows between the Altera Quartus II software and the Xilinx ISE software.

For designs, which the designer has included Xilinx CORE generator modules or instantiated primitives, the bulk of this document guides the designer in design conversion considerations.

Who Should Read This Document

The first and third sections of this application note are designed for engineers who are familiar with the Xilinx ISE software and are using Altera's Quartus II software. This first section describes the possible design flows available with the Altera Quartus II software and demonstrates how similar they are to the Xilinx ISE flows. The third section shows you how to convert your ISE constraints into Quartus II constraints.



For more information on setting up your design in the Quartus II software, refer to the *Altera Quick Start Guide For Quartus II Software*.

The second section of this application note is designed for engineers whose design code contains Xilinx CORE generator modules or instantiated primitives. The second section provides comprehensive information on how to migrate a design targeted at a Xilinx device to one that is compatible with an Altera device. If your design contains pure behavioral coding, you can skip the second section entirely.



This application note assumes you are familiar with the Virtex and Spartan device families and features. Familiarity with VHDL, Verilog HDL, and third-party synthesis tools is also assumed. This application note is based on the latest information available for the Quartus II software version 5.0 and Xilinx ISE 7.1i software version.

Table of Contents

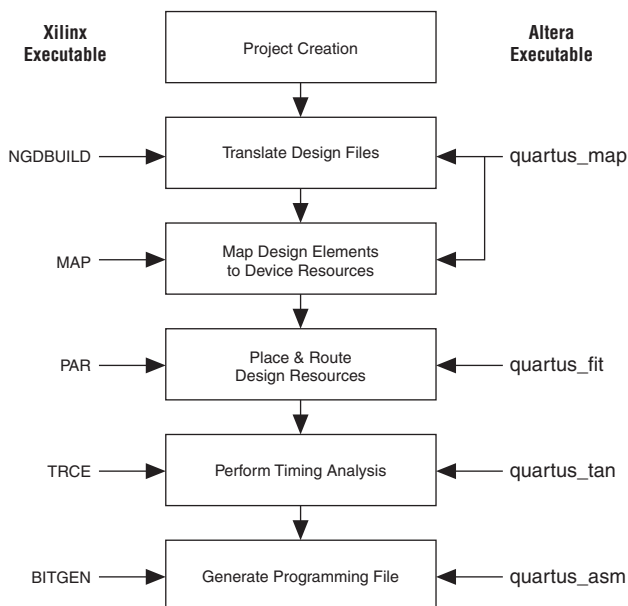
| Section 1 | Page |
|---|------|
| Introduction | 1 |
| Basic FPGA Design Flow Using Command Line Scripting. | 2 |
| Basic FPGA Design Flow Using Tools with GUIs. | 8 |
| Additional Quartus II Features | 23 |
| Scripting with Tcl and Synopsys Design Constraints (SDC) in the Quartus II Software | 25 |
| System Design with SOPC Builder | 31 |
| Hardware Verification with SignalTap II. | 31 |
| Summary of Altera GUI Equivalents for Xilinx ISE Features | 31 |
| <hr/> | |
| Section 2 | |
| Xilinx-to-Altera Design Conversion | 32 |
| Converting Xilinx Primitives for Use In Altera Devices | 33 |
| RAM Architecture Functional Specifications | 37 |
| DCM and DLL Conversion | 51 |
| Double-Data Rate (DDR) I/O Conversion. | 73 |
| <hr/> | |
| Section 3 | |
| Constraints | 80 |
| <hr/> | |
| Summary | |
| References | 83 |

The Quartus II Approach to FPGA Design

The Quartus II software allows you to perform design implementation either by using command-line executables and scripting, or by using the Quartus II graphical user interface (GUI).

Basic FPGA Design Flow Using Command Line Scripting

The ability to automate the FPGA design process saves time and increases productivity. Both Xilinx's ISE software and the Quartus II software provide the tools necessary to automate your FPGA design flow. [Figure 1](#) shows the similarity between a typical command line implementation flow using either Xilinx's ISE or Altera's Quartus II software.

Figure 1. Typical Implementation Flow

Refer to the *Command-Line Scripting* chapter in volume 2 of the *Quartus II Handbook* for information on the Quartus II command-line executable flow.

For ISE software users who are familiar with the command-line implementation flow that compiles a design and generates a programming file for FPGA design files, a similar flow exists within the Quartus II software known as the compilation flow. The compilation flow is the sequence and method by which the Quartus II software translates your design files, maps the translated design to device specific elements, places and routes the design in the device, and generates a programming file. These functions are performed by the Quartus II Integrated Synthesis (QIS), Fitter, Assembler, and Timing Analyzer. Below is a description and comparison of the two software flows using command line executables.

Design File Translation (ngdbuild versus quartus_map)

The ISE software provides the NGDBuild executable that is used to translate your design files into a generic netlist consisting of device specific primitives for the Xilinx implementation flow. This generic netlist will then be used by the subsequent executables in the implementation flow. Similarly, the Quartus II software provides the `quartus_map`

executable that will create a project database that integrates all the design files in your project and performs an analysis and synthesis, if required, on your design files.

The following is an example usage of the `quartus_map` executable:

```
quartus_map filtref --source=filtref.bdf
  --family=stratixii --ver=<revision name>
```

The above command creates a new Quartus II project called **filtref** with the **filtref.bdf** Block Design File (.bdf) as the top-level entity, including the revision name. It targets the Stratix® II device family, and performs logic synthesis and technology mapping on the design files.

Mapping Design Elements to Device Resources (map versus quartus_map)

The ISE software provides the MAP executable which is used to map the logical design elements created by NGDBuild to actual device resources such as memory blocks and I/Os. In the Quartus II command-line flow, `quartus_map` performs both the design translation and mapping of design elements to device resources.

Place and Route Design Resources (par versus quartus_fit)

In place of the PAR executable provided by the ISE software to place and route all device resources into your selected FPGA device, the Quartus II software provides the `quartus_fit` command-line executable. Use `quartus_fit` to place and route all device resources into your selected FPGA device. The following is an example of `quartus_fit` usage:

```
quartus_fit filtref --part=EP2S15F484C3 --fmax=80MHz --tsu=8ns
```

This command performs fitting on the **filtref** project. A Stratix II EP2S15F484C3 device is specified, and the fitter attempts to meet a global f_{MAX} requirement of 80 MHz and a global t_{SU} requirement of 8 ns.

Timing Analysis (trce versus quartus_tan)

In place of the TRCE executable provided in the ISE software for performing a static timing analysis on your design, the Quartus II software provides the `quartus_tan` executable. The following is an example of `quartus_tan` usage:

```
quartus_tan filtref
```

The `quartus_tan filtref` command performs timing analysis on the **filtref** project to determine whether the design meets the timing requirements that were specified using the `quartus_fit` command earlier in the process.

Programming File Generation (bitgen versus quartus_asm)

The ISE software provides the BITGEN executable to generate FPGA programming files. Similarly, the Quartus II software provides the `quartus_asm` executable to generate programming files for FPGA configuration. The following is an example of `quartus_asm` usage:

```
quartus_asm filtref
```

The `quartus_asm filtref` command creates programming files for the `filtref` project.

Table 1 provides a summary and a description of the various executables available in the ISE software and the Quartus II software.

| <i>Table 1. Implementation Flow Summary</i> | | |
|---|--------------------------|---|
| Xilinx Executable | Altera Executable | Description |
| EDIF2NGD, XNF2NGD, NGDBuild, MAP | <code>quartus_map</code> | Translates project design files, e.g. RTL or EDA netlist, and map design elements to device resources |
| PAR | <code>quartus_fit</code> | Places and routes the device resources into the FPGA |
| TRCE | <code>quartus_tan</code> | Performs a static timing analysis on the design |
| BITGEN | <code>quartus_asm</code> | Generates programming file from post-placed-and-route design |
| NGDANNO | <code>quartus_cdb</code> | Back-annotates design for either post or pre place-and-route design |
| NGD2EDIF, NGD2VER, NGD2VHDL | <code>quartus_eda</code> | Generates output netlist files for use with other EDA tools |



For command line help on any of the Quartus II executables described above, type `<command-line executable> --help` at the command prompt. A GUI-enabled help browser is also available that covers all of the Quartus II command-line executables. Start this browser by typing `quartus_sh --qhelp` at the command prompt.

The following is a complete Quartus II compilation flow of the multiple executable example given in the previous sections:

```
quartus_map filtref --source=filtref.bdf --family=stratixii
quartus_fit filtref --part=EP2S15F484C3 --fmax=80MHz --tsu=8ns
quartus_tan filtref
quartus_asm filtref
```

All four executables can be run in sequence either through a script file or Makefile to generate a programming file for your FPGA design.



Refer to the *Command-Line Scripting* chapter in volume 2 of the *Quartus II Handbook* for information on command-line scripting of Quartus II command-line executables.

Scripting with Quartus II Command-Line Executables

The Quartus II command-line executables reduce the amount of system memory required during each step in the design flow. Because it targets only one step in the design flow, each executable is relatively compact, both in terms of file size and the amount of memory used when running. This is particularly beneficial in design environments with heavily-used computer networks or legacy workstations with low amounts of RAM.

Command-line executables allow for easy integration with scripted design flows. You can easily create scripts in any language with a series of command-line executable commands. These scripts can be batch-processed, allowing for integration with distributed computing in server farms. The Quartus II command-line executables can also be integrated in Makefile-based design flows. All these features enhance the ease of integration between the Quartus II software and other EDA synthesis, simulation, and verification software.

Similar to ISE's XFLOW implementation command, the Quartus II shell (`quartus_sh`) contains a `--flow` option, which can be used to open a project and perform a compilation or other related flow with one command. For example, executing

```
quartus_sh --flow compile <project name> -c <revision name>
```

performs a complete compilation, including analysis and synthesis, fitting, timing analysis, and programming file generation, based on constraints and settings contained in the project's revision name.

The Quartus II software provides command-line executables for each stage in the design flow shown in [Figure 1 on page 3](#). Additional command-line executables are provided for specific tasks. [Table 2](#) lists each Quartus II command-line executable and provides a brief description of its function.

| Executable | Description |
|-------------------|---|
| quartus_map | The Quartus II Integrated Synthesis engine builds a single project database that integrates all the design files in a design entity or project hierarchy, performs logic synthesis to minimize the logic of the design, and performs analysis and synthesis on the logic in the design. |
| quartus_fit | The Quartus II Fitter fits the logic of a design into a device. The Fitter selects appropriate interconnection paths, pin assignments, and logic cell assignments. |
| quartus_tan | The Quartus II Timing Analyzer computes delays for the given design and device and annotates them on the netlist for subsequent use by the Simulator. Then, the Timing Analyzer performs timing analysis, allowing you to analyze the performance of all logic in your design. |
| quartus_sim | The Quartus II Simulator performs one of two types of simulation: functional simulation or timing simulation. The Quartus II Simulator is a powerful tool for testing and debugging the logical operation and internal timing of the design entities in your project. |
| quartus_asm | The Quartus II Assembler converts the Fitter's device, logic cell, and pin assignments into a programming image for the device, in the form of one or more Programmer Object Files (.pof), SRAM Object Files (.sof), Hexadecimal (Intel-Format) Output Files (.hexout), Tabular Text Files (.ttf), and Raw Binary Files (.rbf). |
| quartus_pgm | The Quartus II Programmer programs the Altera provided devices. The programmer will use one of the valid supported file format: SOF, POF, jam, and JAM Byte-Code File (.jbc). |
| quartus_swb | The Quartus II Software Builder creates object code from source files to run on either the ARM-based Excalibur™ devices or the Nios®II and Nios embedded processor. |
| quartus_drc | The Quartus II Design Assistant checks the reliability of a design based on a set of design rules. The Design Assistant is especially useful for checking the reliability of a design before converting the design for HardCopy® devices. |
| quartus_cdb | The Quartus II Compiler Database Interface generates and accesses information on atoms. It can be used to back-annotate designs. |
| quartus_eda | The Quartus II Netlist Writer generates output netlist files for use with other EDA tools. |

Table 2. Quartus II Command-Line Executables and Descriptions (Part 2 of 2)

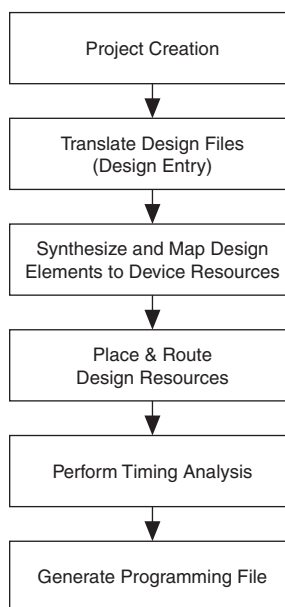
| Executable | Description |
|-------------|---|
| quartus_cpf | The Quartus II Convert Programming Files converts one programming file format to a different possible format. |
| quartus_sh | The Quartus II Shell is a simple Quartus II tool command language (Tcl) interpreter. The Shell may be started with a Tcl script to evaluate, as an interactive Tcl interpreter (shell), or as a quick Tcl command evaluator, evaluating the remaining command-line arguments as one or more Tcl commands. |



Each of the Quartus II executables creates its own report file. For example, the `quartus_map` executable creates a file titled `<project name>.map.rpt`.

Basic FPGA Design Flow Using Tools with GUIs

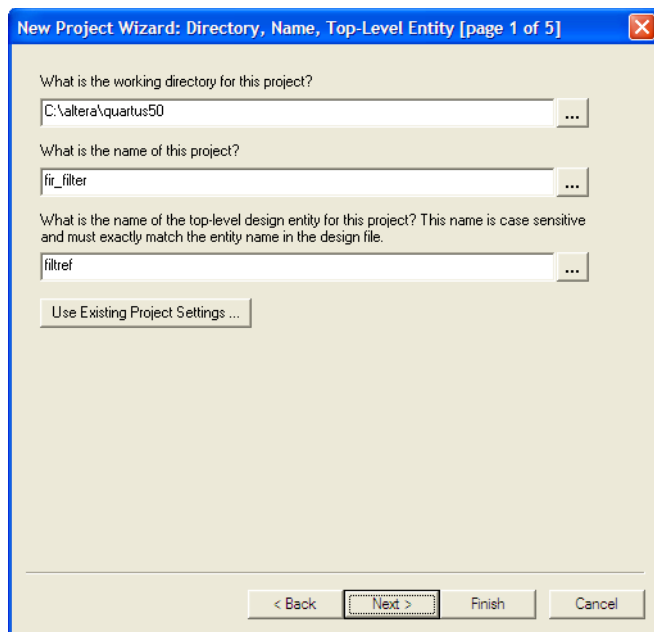
The Quartus II and ISE software Graphical User Interfaces (GUIs) address each of the major FPGA design steps (shown in [Figure 2](#)) in different ways. The following subsections present the Altera equivalents for Xilinx ISE features.

Figure 2. Typical FPGA Design Flow

Project Creation

To begin a design in either the ISE or Quartus II software, you must first create a project. The project specifies the design files and tools that will be used in the project. Similar to the New Project command (File menu) in the ISE software, the Quartus II software uses the New Project Wizard to guide you through specifying a project name and directory, the top-level design entity, any EDA tools you are using, and a target device. To invoke the New Project Wizard, select **New Project Wizard** (File menu). [Figure 3](#) shows the first page of the Quartus II New Project Wizard.

Figure 3. The New Project Wizard Start Screen



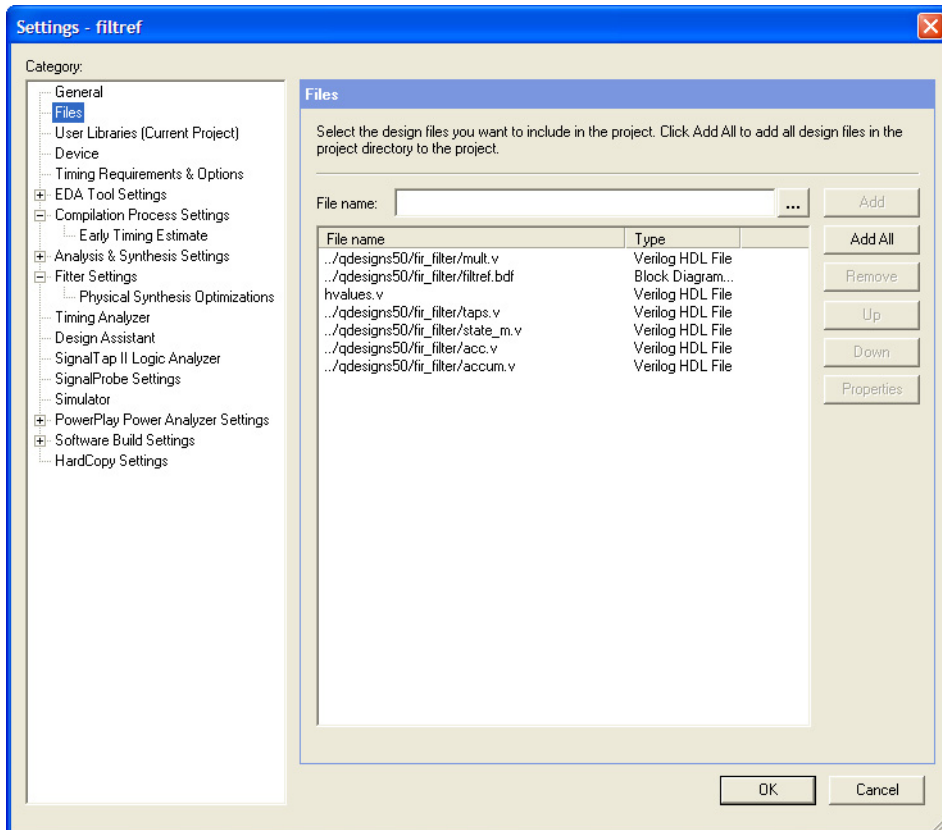
All the settings you make when creating your project with the New Project Wizard can be modified at later stages in the design process.

Design Entry

Both the ISE software and the Quartus II software support hardware description language (HDL), EDA netlist, and schematic design files as design entry methods.

In place of the **Add Source** dialog box of the Xilinx ISE software, the **Files** page of the **Settings** dialog box (Assignments menu) in the Quartus II software allows you to add or remove existing design files from your project (Figure 4).

Figure 4. Files Page of Settings Dialog Box



HDL Design Entry

To create a new HDL design file in the Quartus II software, choose **New** (File menu) and select the type of file to create. To assist you in creating HDL designs, the Quartus II software provides templates for overall AHDL, VHDL and Verilog HDL file structures and constructs, including various logic functions and parameter declarations. Also, the Quartus II Text Editor offers syntax coloring for highlighting HDL reserved words and comments.

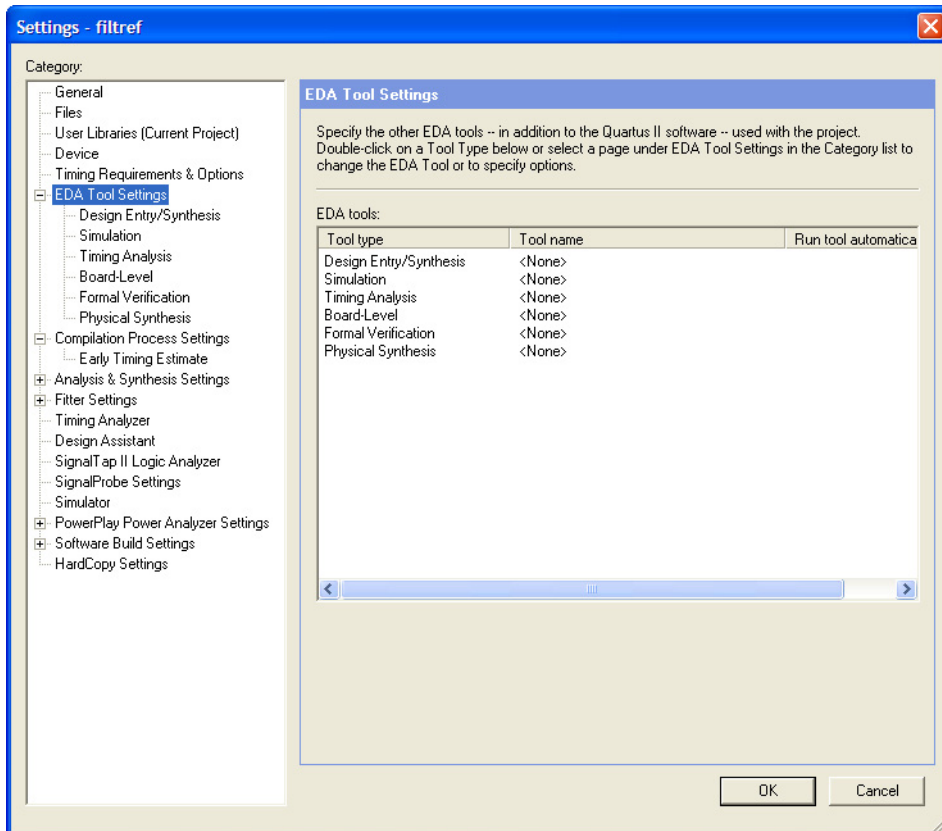


For more information on design guidelines in the Quartus II software, refer to the *Design Recommendations for Altera Devices* and *Recommended HDL Coding Styles* chapters in volume 1 of the *Quartus II Handbook*.

EDA Netlist Design Entry

The ISE and Quartus II software allow you to compile designs from netlists generated from EDA tools such as Synplify or Precision RTL™. In the Quartus II software, you can specify the EDA tools you are using for synthesis, simulation, timing analysis, board-level signal verification, formal verification, and physical synthesis in the **EDA Tool Settings** page of the **Settings** dialog box (Assignments menu) as well as on the appropriate page of the New Project Wizard (Figure 5). For more information using third-party synthesis tools, see “[Synthesis](#)” on page 12.

Figure 5. The EDA Tool Settings Page of Settings Dialog Box





For more information on using the Synplify and Precision RTL tools in conjunction with the Quartus II software, refer to the *Synplicity Synplify & Synplify Pro Support* and *Mentor Graphics Precision RTL Synthesis Support* chapters in volume 1 of the *Quartus II Handbook*. Refer also the *Synopsys Design Compiler FPGA Support* chapter in volume 1 of the *Quartus II Handbook*.

Schematic Design Entry

In the Quartus II software, you can use Altera-supplied design elements such as Boolean gates and registers, or you can create your own symbols from HDL or EDA netlist design entities. The Quartus II software also includes an extensive library of megafunctions supplied with the software. These are added using representative schematic symbols customized using the MegaWizard® Plug-In Manager.



See “[Quartus II MegaWizard Plug-In Manager](#)” on page 23 for more information.

To create a block design file from a VHDL design file, a Verilog HDL design file, or an EDA Netlist choose **Create/Update > Create Symbol Files for Current File** (File menu).

Synthesis

Similar to the Xilinx Synthesis Technology (XST) in ISE, the Quartus II software includes Quartus II Integrated Synthesis (QIS) which provides full synthesis support for AHDL, VHDL, and Verilog HDL. The integrated synthesis engine is invoked whenever the Quartus II software encounters any files of the three supported HDL language types.

The Quartus II software also supports synthesized design files from third-party synthesis tools including EDIF (.edf) and Verilog Quartus II Mapping (.vqm) netlist files.

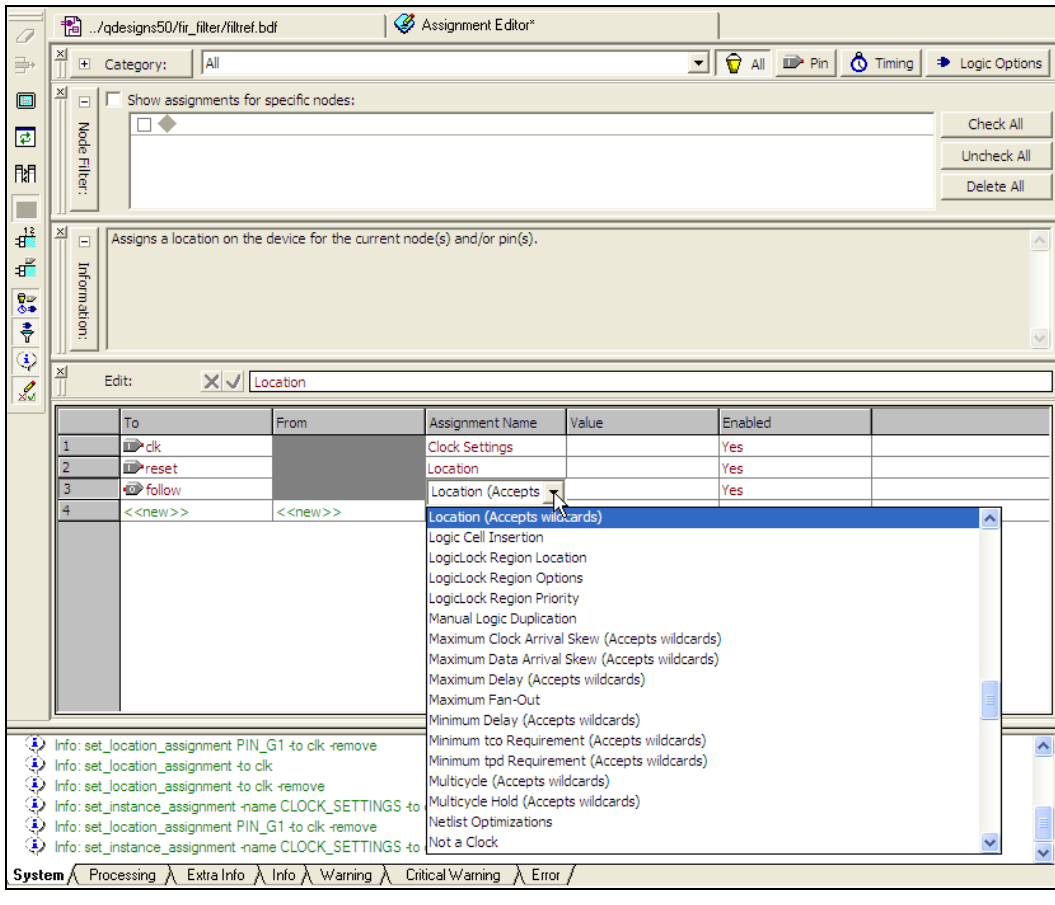
Design Assignments

Specifying device assignments assures that your design takes advantage of specific features of your targeted device architecture and meets performance goals. The ISE software provides two tools—the Constraints Editor and the Pinout and Constraints Editor (PACE)—to create and edit constraints. The Quartus II Assignment Editor conveniently allows you to create and view constraints using a single centralized interface. Additionally, the Quartus II Pin Planner allows you to view, create, and edit pin assignments in a graphical interface.

The Quartus II Assignment Editor

In place of the Constraints Editor and PACE tools in the ISE software, use the Quartus II **Assignment Editor** (Assignment menu) to make timing and placement design constraints for your design. The Quartus II software dynamically validates the assignments whenever changes are made with the Assignment Editor, issuing errors or warnings for invalid assignments. Adding or changing assignments is acknowledged with messages reported in the **System** tab of the Quartus II message utility window (Figure 6).

Figure 6. The Assignment Editor



The Quartus II Pin Planner

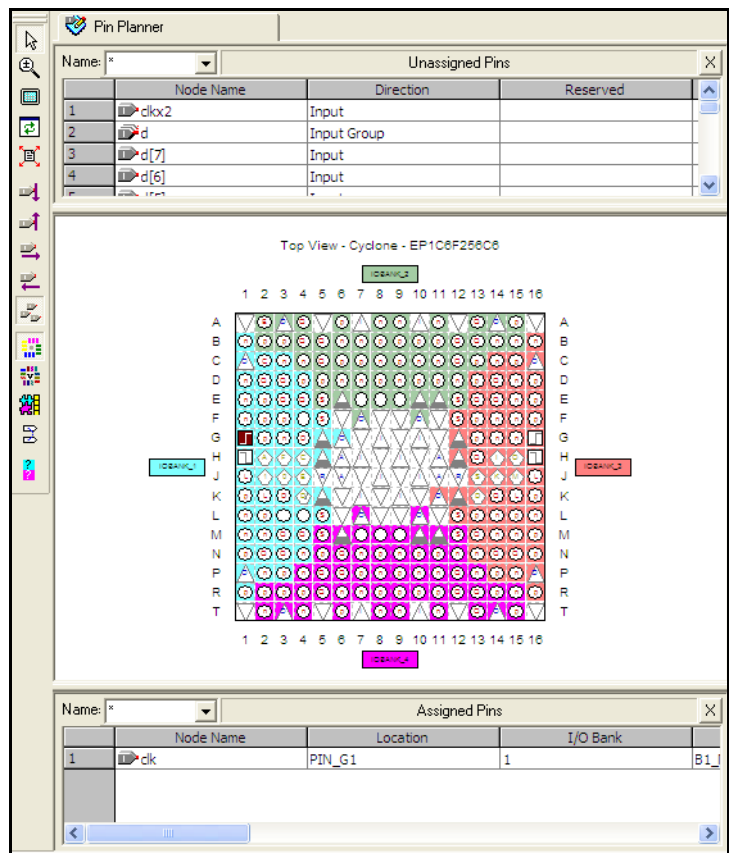
Like the PACE tool in the ISE software, the Quartus II Pin Planner provides a graphical Package view allowing you to make pin location assignments using a device Package view instead of pin numbers. With the Pin Planner, you can identify I/O banks, VREF groups, and differential pin pairings to help you through the I/O planning process.

To use the Pin Planner, choose **Pin Planner** (Assignments menu). [Figure 7](#) shows the Pin Planner.



For more information on using the Pin Planner, refer to the *I/O Planning* chapter in volume 2 of the *Quartus II Handbook*.

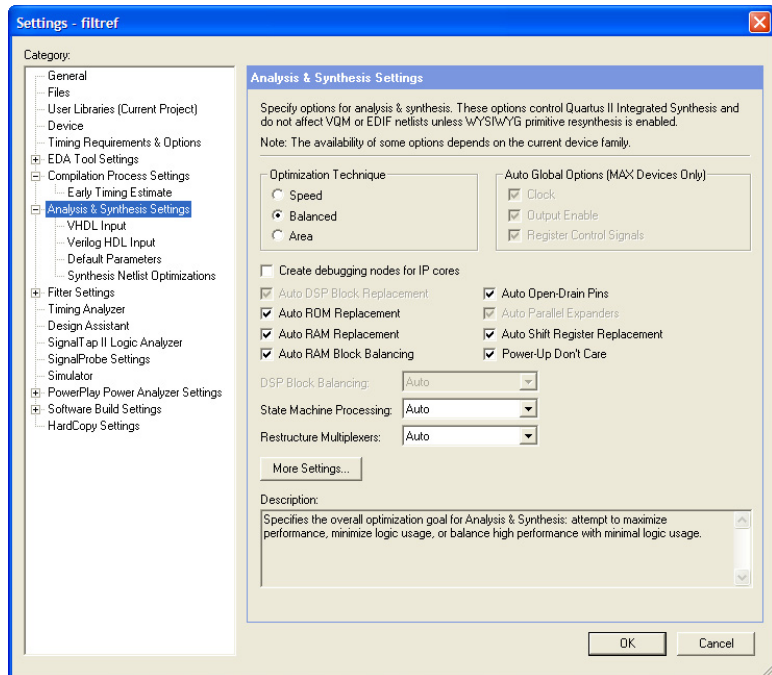
Figure 7. The Pin Planner



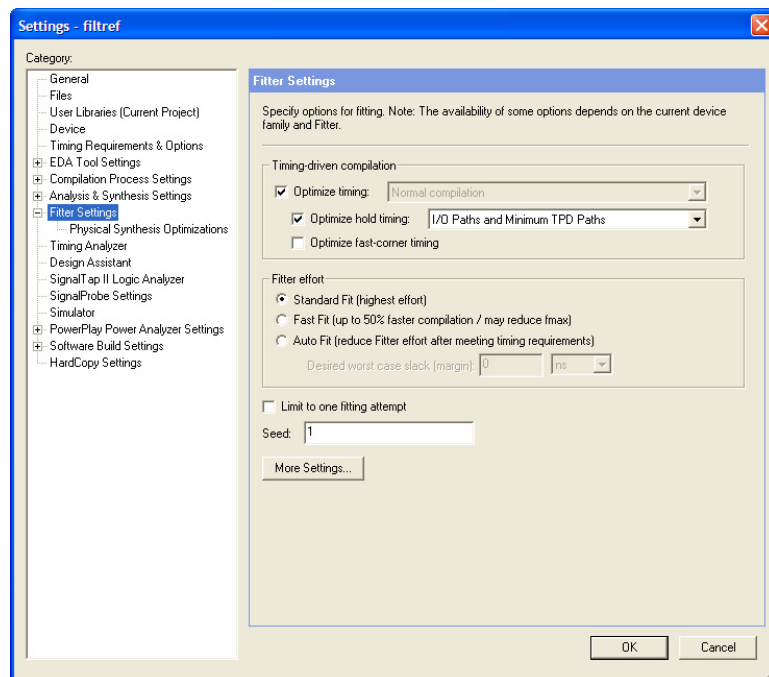
Analysis & Synthesis and Fitter Settings


The **Analysis & Synthesis Settings** and the **Fitter Settings** dialog boxes allow you to easily set project-wide Quartus II compiler settings. The **Analysis & Synthesis Settings** dialog box (see [Figure 8](#)) allows you to set options that affect the analysis and synthesis stage of the compilation flow. These options include the Optimization Technique, State Machine Processing, Restructure Multiplexers, and others.

Figure 8. Analysis & Synthesis Settings Dialog Box



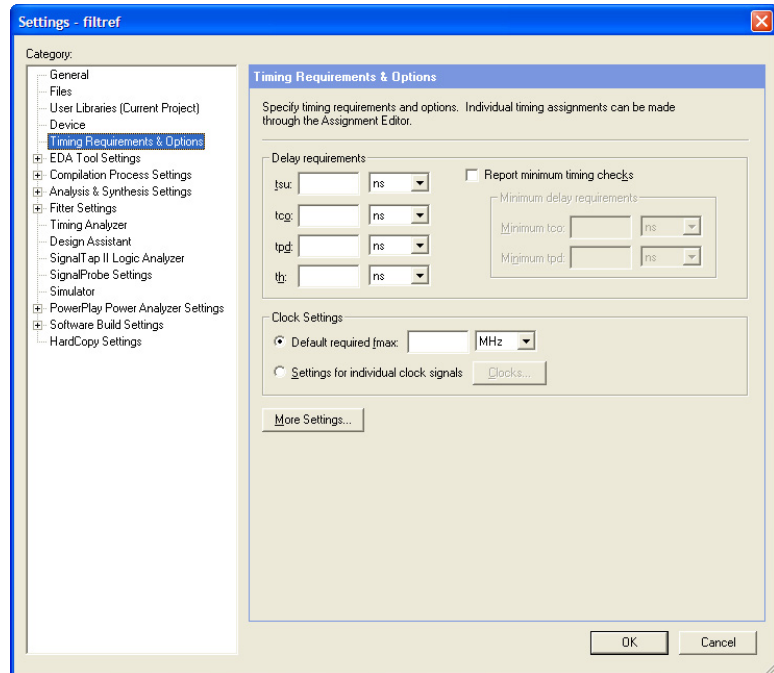
The **Fitter Settings** dialog box (see [Figure 9](#)) allows you to set options that affect the Fitter stage of the compilation flow. These options include Fitter Effort, Fitting Attempts, Seed value, and so forth.

Figure 9. Fitter Settings Dialog Box

 You can also use the **Device** page of the **Settings** dialog box (Assignments menu) to select or change devices for your project.

Timing Settings

The **Timing Requirements & Options** page of the **Settings** dialog box (Assignments menu) in the Quartus II software allows you to easily set project-wide timing requirements for your design. You can specify requirements for overall circuit frequency (f_{MAX}), project-wide setup time (t_{SU}), hold time (t_H), clock-to-output time (t_{CO}), and pin-to-pin time (t_{PD}). You can also specify clock relationships and enter settings to control timing analysis.

Figure 10. Timing Requirements Dialog Box

For more information on the Quartus II Timing Analyzer, refer to the *Quartus II Timing Analysis* chapter in volume 3 of the *Quartus II Handbook*.

You can use the Assignment Editor to make individual timing assignments or to assign clock settings to a clock signal.

Design Implementation

The ISE software follows an implementation flow that compiles a design and generates a programming file for your FPGA design files. A similar flow exists within the Quartus II software known as the compilation flow. The compilation flow is the sequence and method by which the Quartus II software translates your design files, maps the translated design to device specific elements, places-and-routes the design in the device, and generates a programming file. These functions are performed by the QIS, Fitter, Assembler, and Timing Analyzer.

You can start the compilation flow at any point in the design process, whether or not you have completed making your project settings and constraints. In the Quartus II software, choose **Start Compilation** (Processing menu) to start the compilation process.

In the initial compilation phase, the QIS creates a database from your design files containing all necessary design information. A design rule check is performed on all design files in the project, ensuring that no boundary connectivity errors or syntax errors exist. This database is available for use for all subsequent steps in the compilation flow.

The QIS optimizes your design for the targeted Altera FPGA and maps the design to the device. Mapping converts your design files into architecture-specific atoms that target device resources such as logic elements (LEs) and RAM blocks.

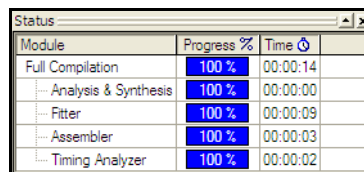
The Fitter places and routes the atoms created by the QIS in the selected device. The Fitter performs additional optimization to improve your design's timing and resource usage based on timing constraints.

When the optimized fit is achieved, the Assembler generates the programming file for your design. The programming file contains all placement and routing information for your design and is used to program the target Altera device.

The Timing Analyzer performs a static timing analysis on every path in your design. This analysis allows you to identify critical paths and timing errors necessary to meet the design timing budget or achieve timing closure.

The **Status utility** window shows progress of the current compilation (Figure 11). The results of a compilation may be viewed in the **Compilation Report** window (Processing menu). The report window opens automatically when you compile a design, and shows the design hierarchy, a compilation summary, statistics on the performance of the design, and a link to the floorplan view.

Figure 11. The Status Utility Window

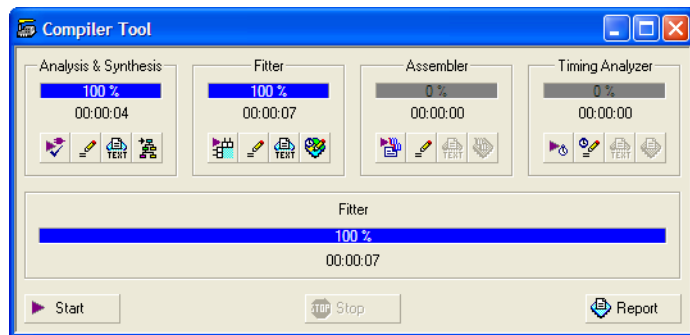


| Module | Progress % | Time |
|--------------------------|------------|----------|
| Full Compilation | 100 % | 00:00:14 |
| ... Analysis & Synthesis | 100 % | 00:00:00 |
| ... Fitter | 100 % | 00:00:09 |
| ... Assembler | 100 % | 00:00:03 |
| ... Timing Analyzer | 100 % | 00:00:02 |

Use the **Timing Closure Floorplan** (Assignments menu) to view compiler partitioning, fitting and timing results or to assign physical device resources.

Each of the phases in the compilation flow can also be started independently of the others, similar to the command-line executable flow, within the Quartus II GUI using the **Compiler Tool** (Tools menu) as shown in [Figure 12](#).

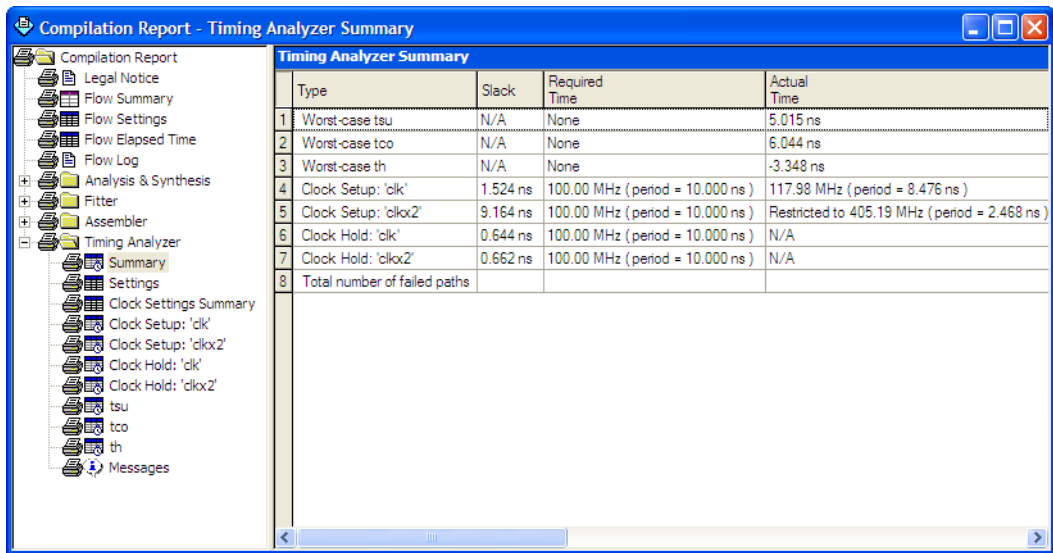
Figure 12. Compiler Tool Window



Timing Analysis

Roughly equivalent to the Post-Place and Route Static Timing Report generated by the Xilinx ISE software, the Quartus II Timing Analyzer analyzes and reports the performance of all logic in your design, allowing you to determine all of the critical paths that limit your design's performance. [Figure 13](#) shows the timing analysis report.

Figure 13. An Example Timing Analysis Report



| Timing Analyzer Summary | | | | |
|-------------------------|------------------------------|----------|---------------------------------|--|
| | Type | Slack | Required Time | Actual Time |
| 1 | Worst-case tsu | N/A | None | 5.015 ns |
| 2 | Worst-case tco | N/A | None | 6.044 ns |
| 3 | Worst-case th | N/A | None | -3.348 ns |
| 4 | Clock Setup: 'clk' | 1.524 ns | 100.00 MHz (period = 10.000 ns) | 117.98 MHz (period = 8.476 ns) |
| 5 | Clock Setup: 'clkx2' | 9.164 ns | 100.00 MHz (period = 10.000 ns) | Restricted to 405.19 MHz (period = 2.468 ns) |
| 6 | Clock Hold: 'clk' | 0.644 ns | 100.00 MHz (period = 10.000 ns) | N/A |
| 7 | Clock Hold: 'clkx2' | 0.662 ns | 100.00 MHz (period = 10.000 ns) | N/A |
| 8 | Total number of failed paths | | | |



For more information on the differences between timing analysis methodologies, refer to the white paper *Performing Equivalent Timing Analysis Between the Altera Quartus II Software and Xilinx ISE*.

Design Optimization

The Quartus II Fitter is guided by your design's timing requirements, and will attempt to satisfy all the timing constraints specified. Ensure that your timing settings accurately reflect the timing requirements of your design.

Choose **Settings** (Assignments menu) and select **Fitter Settings** in the **Category** list. Specify the type of timing-driven compilation for the Fitter to perform by turning on or off one or both of the following options:

- **Optimize timing** directs the Fitter to optimize routing within a device to meet timing requirements.
- **Optimize hold timing** directs the Fitter to optimize minimum delay timing constraints.
- **Optimize fast-corner timing** directs the Fitter to optimize routing under the fast-corner (fastest manufactured device, operating in low-temperature and high-voltage conditions) condition. By default, the

Fitter will optimize under slow-corner (slowest manufactured device for a given speed grade, operating in high-temperature and low-voltage conditions) conditions.



For more strategies for optimizing your design, refer to *Section III: Area Optimization & Timing Closure* in volume 2 of the *Quartus II Handbook*.

Design Partitioning

LogicLock™ block-based design is a methodology available in the Altera Quartus II software. The LogicLock methodology allows you to optimize each design entity/module independent of other entities/modules. Productivity is increased because each design module needs to be optimized only once. During integration and system-level verification, the performance of each logic module is preserved.

To create a LogicLock region in the Timing Closure Floorplan, perform the following steps:

1. Choose **Settings** (Assignments menu), then select **Device** in the **Category** list.
2. Specify a target device, then choose the **Timing Closure Floorplan** (Assignments menu).
3. Click the **Create New LogicLock Region** button on the toolbar, then click and drag the pointer until the region is the desired size.



You must select a target device (step 1) or the **Create New LogicLock Region** button will be grayed out.

4. Choose **Properties** (right-button pop-up menu) on the LogicLock region to view or edit the region's properties.



For more information on using LogicLock methodology, refer to the *LogicLock Design Methodology* chapter in volume 2 of the *Quartus II Handbook*.

Simulation

Just as with the Xilinx ISE software, the Quartus II software supports integration with many third-party EDA simulation tools, including those from Mentor Graphics®, Cadence, and Synopsys. To perform functional/behavioral simulation on designs containing LPMs or MegaWizard-generated functions, use the Altera functional simulation models installed with the Quartus II software.

The LPM simulation model files are:

- **220model.v** for Verilog HDL
- **220pack.vhd** and **220model.vhd** for VHDL

The Altera megafunction simulation model files are:

- **altera_mf.v** for Verilog HDL
- **altera_mf.vhd** and **altera_mf_components.vhd** for VHDL

To perform gate-level timing simulation on a design, the Quartus II software generates output netlist files containing information on how the design was placed into device-specific architectural blocks. The Quartus II software provides this information in the form of a **.vo** file for Verilog HDL and a **.vho** file for VHDL output files. The accompanying timing information is stored in a SDF file that annotates the delay for the elements found in the **.vo** or **.vho** output netlist.

The Quartus II Simulator enables testing and debugging the logical operation and internal timing of the design.

The **Simulator** page in the **Settings** dialog box (Assignments menu) helps you create and save Simulator settings by specifying the time period covered by the simulation and the source of the vector stimuli. You can also turn on options for reporting the simulation coverage and setup and hold time violations.

Create a Vector Waveform File (**.vwf**) in the Waveform Editor by choosing **New** (File menu) and clicking the **Other Files** tab. Select **Vector Waveform File** and click **OK**. The VWF should contain the vector inputs for simulation and the names of the outputs to be simulated.

The **Automatically add pins to simulation output waveforms** option on the **Simulator** section of the **Settings** dialog box (Assignments menu) directs the Simulator to automatically add waveforms for all the output pins in the project to the simulation outputs. This eliminates the need to manually enter the names of the output nodes you want to monitor.

Choose **Start Simulation** (Processing menu) to run the simulation. View the simulation results by choosing **Simulation Report** (Processing menu).

Device Programming

The Quartus II Programmer allows you to use files generated in the compilation flow to program or configure all Altera programmable logic devices and supported configuration devices.

Open the Programmer by choosing **Programmer** (Tools menu).



For more information on using the **Programmer**, refer to the *Assembler* section of the Online help in the Quartus II software.

Additional Quartus II Features

In addition to providing the standard set of tools required in any FPGA design flow, the Quartus II software provides additional features and tools to assist you with achieving your desired design requirements.

Quartus II MegaWizard Plug-In Manager

In place of the CoreGen and the Architecture Wizard available in the Xilinx ISE software, the Altera MegaWizard Plug-In Manager helps you create highly customized megafunctions that are optimized for the device targeted in your design. These customizations draw on Altera-provided megafunctions, including library-of-parameterized-modules (LPM) functions, ranging from simple Boolean gates to complex memory structures. The MegaWizard Plug-In Manager categorizes all supported modules into folders titled: arithmetic, gates, I/O, memory compiler, and storage.

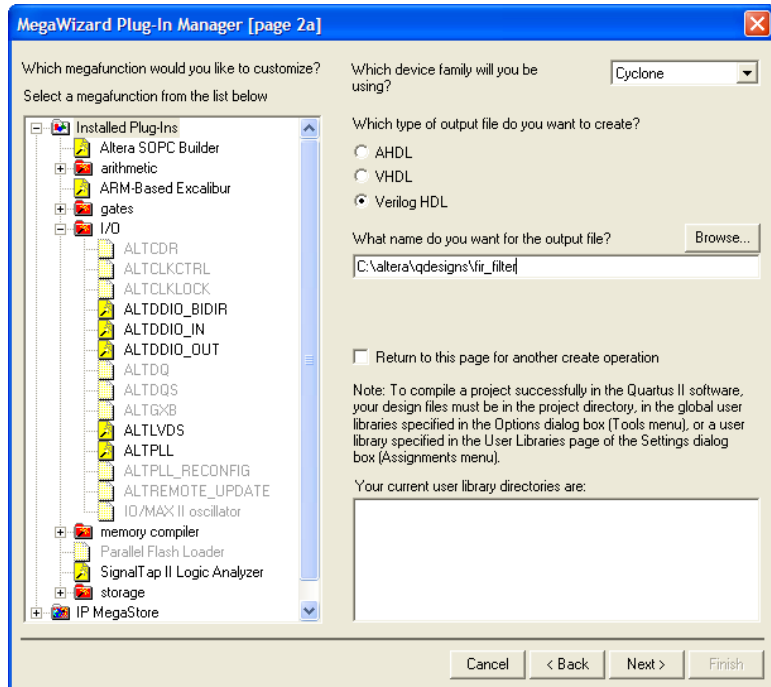
You can access the MegaWizard Plug-In Manager as a stand-alone tool or as an integrated tool in your Quartus II project. [Table 3](#) provides a brief list of supported megafunctions.

Table 3. Supported Megafunctions in the MegaWizard Plug-In Manager

| Folder | Megafunction |
|-----------------|---|
| Arithmetic | altaccumulate, altfp_mult, altmemmult, altmult_accum (MAC), altmult_add, altsqrt, lpm_add_sub, parallel_add |
| Gates | lpm_add, lpm_bustri, lpm_clshift |
| I/O | altdio_bidir, altdio_in, altll_reconfig, altpll, altlvds |
| Memory Compiler | FIFO, RAM : 1-Port, RAM : 3-Port |
| Storage | altshift_taps, altsyncram, lpm_shiftreg |

Use the MegaWizard Plug-In Manager to generate Altera equivalents for Xilinx primitives and CoreGen and Architecture Wizard modules.

[Figure 14](#) shows page 2a of the MegaWizard Plug-In Manager.

Figure 14. MegaWizard Plug-In Manager

The MegaWizard Plug-In Manager automatically generates a Component Declaration file (with the extension **.cmp**) that can be used in VHDL Design Files (**.vhd**) and an AHDL Include File (**.inc**) that can be used in Text Design Files (**.tdf**) and Verilog Design Files (**.v**). The MegaWizard Plug-In Manager also creates a sample instantiation template with the extension **_inst.tdf** for AHDL designs, **_inst.vhd** for VHDL designs, and **_inst.v** for Verilog HDL designs. A sample declaration file with a **_bb.v** extension is also created for Verilog HDL designs. The sample files contain module and port declarations for the custom megafunction variation. A Block symbol file (**.bsf**) will also be created which is a symbol that represents the logic in a schematic file.

Quartus II Incremental Compilation

In place of the Incremental Design feature in ISE, Quartus II Incremental Compilation allows you to organize your design in logical and physical partitions for synthesis and fitting. Design iteration time can be dramatically reduced by focusing new compilations on a particular design partition and merging the results with previous compilation

results from other partitions. Incremental compilation facilitates block-based design, and allows you to preserve performance for unchanged blocks of your design. You can also target optimization techniques, such as physical synthesis, to specific design partitions while leaving other blocks untouched.



For more information, refer to the *Quartus II Incremental Compilation* chapter in volume 1 of the *Quartus II Handbook*.

Scripting with Tcl and Synopsys Design Constraints (SDC) in the Quartus II Software

The Quartus II GUI provides an easy way to access all features and commands offered by the software. However, as designs grow in resource utilization and complexity, the need to automate common tasks and to streamline the entire FPGA design flow becomes a requirement. The Quartus II software provides support for Tcl and SDC to help facilitate project assignments, compilation, and constraints. The following provides a brief description of the Quartus II software support of Tcl and SDC.



For more information, refer to the *Tcl Scripting* chapter in volume 2 of the *Quartus II Handbook*.

The Quartus II software contains Tcl application program interface (API) functions that you can use to automate a variety of common tasks, such as making assignments, compiling designs, analyzing timing, and controlling simulation. The Quartus II software offers the following ways to run your Tcl scripts:

- Interactively from the shell
- Using scripts in batch from the shell
- As a Batch file from the DOS or UNIX prompt
- Directly from the command line

Running Tcl Scripts Interactively from the Shell

Using the `-s` or `--shell` switch option starts an interactive Tcl shell session, replacing the normal command line prompt with `tcl>`:

```
c:\>quartus_sh -s
Info: *****
Info: The Quartus II Shell supports all Tcl commands in addition
to
Info: Quartus II Tcl commands
Info: All unrecognized commands are assumed to be external
Info: and are run using Tcl's "exec" command
Info: To exit, type "exit"
```

```
Info: For a list of all Quartus II Tcl commands, type "help"
Info: *****
tcl>
```

Everything typed in the Tcl shell will be directly interpreted by the Quartus II Tcl interpreter.



The Tcl shell includes a history list of previous commands entered, but it does not allow commands to span more than one line.

Using Scripts in Batch from a Shell

Once you create a Tcl script file (**.tcl**), you can run it by typing the following command in a Tcl shell:

```
source <script_name>.tcl
```

This will run the Tcl script, a previously written set of Tcl commands to help configure a project or project assignments created in Tcl.

Running Scripts from the DOS or UNIX Prompt

The following command will run the Quartus II Tcl shell and use the Tcl file specified by the **-t** option as the input Tcl script:

```
quartus_sh -t <script_name>.tcl
```

The Quartus II Tcl interpreter will read in process and execute the Tcl commands in the Tcl script file and then exit back to the command-line prompt. Most of the examples in this application note are written to be run as a batch file.

Running Scripts Directly from the Command-line

The last way of accessing Tcl is using the **--tcl_eval** option. This directly evaluates the rest of the command line arguments as one or more Tcl commands. If there are two or more Tcl commands, you have to separate them with semicolons. The Quartus II Tcl interpreter is used to interpret these Tcl commands. For example, typing the following command:

```
quartus_sh --tcl_eval puts Hello; puts World
```

will cause the following output:

```
Hello
World
```

The Tcl evaluate option allows external scripting programs (such as `make`, `perl`, and `sh`) to access information from Quartus II software. One such application may be to obtain device family information for a targeted part.

The `--tcl_eval` option is also very useful to get Tcl help information directly from the command-line prompt.

Using the Tcl Console in the Quartus II GUI

You can execute Tcl commands directly in the Quartus II Tcl Console window. To open the Tcl Console window, choose **Auxiliary Windows > Tcl Console** (View menu). The Tcl Console is usually located on the bottom-right corner of the Quartus II GUI.

The following example Tcl script performs these tasks:

- Opens the `fir_filter` project if it exists. If the project doesn't exist, the script creates the project
- Sets the project to target a Stratix II EP2S15F672C3 device
- Assigns the `clk` pin to the physical pin F18
- performs compilation

```
# This Tcl file works with quartus_sh.exe
# This Tcl file will compile the Quartus II tutorial
# fir_filter design

# set the project_name to fir_filter
# set compiler setting to filtref

set project_name fir_filter
set csf_name filtref

# Create a new project and open it
# Project_name is project name
# Require package ::quartus::project

if {[project_exists $project_name]} {
    project_new -cmp $csf_name $project_name;
} else {
    project_open -cmp $csf_name $project_name;
}

#----- Make device assignments -----#

set_global_assignment -name FAMILY "Stratix II"
set_global_assignment -name DEVICE EP2S15F672C3

#----- Make instance assignments -----#

# assign pin clk to pin location F18
```

```

set_location_assignment -to clk Pin_F18

#----- project compilation -----#

# The project is compiled here

package require ::quartus::flow
execute_flow -compile

project_close

```

Using Synopsys Design Constraints with the Quartus II Software

To ease the integration with EDA synthesis tools, the Quartus II software supports the SDC functions. [Table 4](#) provides a brief list of supported SDC functions.

Table 4. Quartus II Supported SDC Functions (Part 1 of 2)

| Command | Description |
|------------------------|--|
| create_clock | Creates a base clock with the given name and waveform, and applies the clock to the specified clock pin list. |
| set_clock_latency | Inserts a source latency into an existing base clock. |
| set_false_path | Specifies that the timing paths that start from a designated start node and end at a designated destination node are false paths. |
| set_input_delay | Specifies the external input delay of a set of input or bidirectional pins with respect to the designated clock. |
| remove_clock | Removes all the clocks that are used in the current design if the <code>-all</code> option is specified. |
| create_generated_clock | Creates a derived, or generated clock from the given clock source. A generated clock can be derived only from a base clock. The generated clock is always assumed to be propagated. |
| get_clocks | Returns the list of clock pins as specified in the <code><clock_pin_list></code> . The input list is returned as the output. When <code><no port list></code> is specified, the command returns nothing. |
| remove_input_delay | Removes the specified input delay assignments from the current design. |
| remove_output_delay | Removes the specified output delay assignments from the current design. |
| reset_path | Removes the specified timing path assignments from the current design. If neither the <code>-setup</code> or <code>-hold</code> option is specified, then both setup and hold paths are removed. |
| set_false_path | Specifies that the timing paths that start from the designated <code><from_pin_list></code> and end in the designated <code><to_pin_list></code> are false paths. |

Table 4. Quartus II Supported SDC Functions (Part 2 of 2)

| Command | Description |
|----------------------|--|
| set_input_delay | Specifies the external input delay of a set of input or bidir pins with respect to the designated clock. The delay applies to both the positive and negative edges of the clock. The specification is internally translated into the equivalent Quartus II software t_{SU} requirements. |
| set_max_delay | Specifies the maximum delay for the timing paths that start from the designated <i><from_pin_list></i> and end in the designated <i><to_pin_list></i> . |
| set_min_delay | Specifies the minimum delay for the timing paths that start from the designated <i><from_pin_list></i> and end in the designated <i><to_pin_list></i> . |
| set_multicycle_path | Specifies that the given timing paths have multicycle setup or hold delays with the number of cycles specified by the <i><path_multiplier></i> . The meaning of multicycle hold differs between the Quartus II timing analysis and the Synopsys PrimeTime software. Refer to the online Help for each software package for more information. |
| set_output_delay | Specifies the external output delay of a set of output or bidir pins with respect to the designated clock. The delay applies to both the positive and negative edges of the clock. The specification is internally translated into the equivalent Quartus II software t_{CO} requirements. |
| set_propagated_clock | Specifies that a given clock is propagated using the actual clock network delays. This command is included for compatibility with the Quartus II software SDC commands. The Quartus II software ignores the command because it supports only propagated clocks. |
| get_ports | Returns the list of ports as specified in the <i><port list></i> . |



Refer to the Quartus II on-line help for a complete list of supported Tcl and SDC functions.

Cross-probing in the Quartus II Software

Cross-probing is the ability to select design elements from one tool and locate them in another tool. All features and tools within the Quartus II software are highly integrated, resulting in a design environment that provides seamless cross-probing abilities. Table 5 shows the cross-probing support provided by the Quartus II software.

| From | To | | | | | | | |
|----------------------------------|------------------|------------|------------|-------------------|-------------|--------------|-----------------------|-------------|
| | Schematic Editor | HDL Editor | RTL Viewer | Assignment Editor | Pin Planner | Floorplanner | Technology Map Viewer | Chip Editor |
| Project Navigator | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Message Window | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Schematic Editor | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| HDL Editor | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| RTL Viewer | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ |
| Assignment Editor | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ |
| Pin Planner | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ |
| Waveform Editor | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Floorplanner | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |
| Technology Map Viewer | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ |
| Timing Report | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Chip Editor | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| Signal Tap II | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| PowerPlay Power Analyzer Reports | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

For example, with the cross-probing ability in the Quartus II software you can locate design elements from the RTL Viewer to the Assignment Editor. This eliminates the searching time for node names and pin names when applying design constraints in the Assignment Editor.



Refer to the *Quartus II Handbook* for more information on the features and tools listed in [Table 5](#).

System Design with SOPC Builder

The SOPC Builder feature included and integrated in the Quartus II software enables the use of processors (such as Altera Nios and Nios II embedded processors), interfaces to off-chip processors, standard peripherals, IP cores, on-chip memory, interfaces to off-chip memory, and user-defined logic into a custom system module.

SOPC Builder generates a single system module that instantiates these components, and automatically generates the necessary interconnect logic to bind them together.



For more information on system design with the SOPC Builder, refer to *Volume 4: SOPC Builder* of the *Quartus II Handbook*.

Hardware Verification with SignalTap II

The SignalTap® II Logic Analyzer is a multiple-input, digital acquisition instrument that captures and stores signal activity from any internal device node(s).



For more information on SignalTap II Logic Analysis, see the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

Summary of Altera GUI Equivalents for Xilinx ISE Features

[Table 6](#) lists the stages of the design flow, the Xilinx software features used in each stage, and their equivalents in the Quartus II software.

| Feature | Xilinx ISE | Quartus II |
|-------------------------------|-----------------------------|---|
| Project Creation | New Project | New Project Wizard |
| Design constraint assignments | Constraints Editor and PACE | Quartus II Assignment Editor, I/O Analyzer, and Pin Planner |

Table 6. GUI Equivalents in the Quartus II Software for ISE Features (Part 2 of 2)

| Feature | Xilinx ISE | Quartus II |
|---|--|--|
| Design Entry | HDL Editor | HDL Editor |
| | Schematic Entry | Schematic Entry |
| | CoreGen and Architecture Wizard | MegaWizard Plug-In Manager |
| Synthesis | Xilinx Synthesis Technology (XST) or Third-party EDA Synthesis | Quartus II Integrated Synthesis (QIS) or Third-Party EDA Synthesis |
| Implementation Flow consists of design translation, mapping, and place-and-route of the design into the FPGA to meet requirements set by user | Design Implementation: Translate, Map, and Place-and-Route | Design Compilation: QIS and Fitter |
| Power Analysis | XPower | PowerPlay Power Analyzer |
| Static timing analysis on post-fitted design | Xilinx Timing Analyzer and Trace | Quartus II Timing Analyzer |
| Functional and Timing Simulation | 3rd Party Simulation Tools | 3rd Party Simulation Tools or Native Quartus II Simulator |
| Generation of device programming file | Bitgen | Assembler |
| Hardware Verification | ChipScope Pro | SignalTap II |
| Viewing and editing design placement | Floorplanner or FPGA Editor | Timing Closure Floorplan, Chip Editor |
| Customization and generation of IP cores through the GUI | CoreGen System and Architecture Wizard | MegaWizard Plug-In Manager |
| Compilation and assignment process for power users | | Tcl Scripting |
| Technique used to design, optimize, and lock down nodes one at a time | Modular Design Flow | LogicLock, Netlist Optimization options |

Xilinx-to-Altera Design Conversion

Successfully converting a Xilinx-targeted design for use in an Altera device is a three-step process.

1. Replace Xilinx-specific primitives with Altera primitives, megafunctions, or constraints.
2. Replace Xilinx CoreGen or Architecture Wizard modules with Altera megafunctions generated with the Quartus II MegaWizard Plug-In Manger.
3. Set timing and device constraints using the Quartus II software corresponding to those found in the Xilinx design you are converting.

Converting Xilinx Primitives for Use In Altera Devices

Primitives are the basic building blocks of a Xilinx design. They perform various dedicated functions within the device, such as shift registers, and implement specific I/O standards for the Xilinx device I/O pins. Xilinx primitives have fixed ports and cannot be customized.

Primitives can be easily identified because their names are standardized. [Table 7](#) lists commonly used Xilinx primitives and describes an equivalent Altera design element.

| Xilinx Primitive | Description | Altera Equivalent | Conversion Type |
|---------------------------------|--|--|--|
| BUF, 4, 8, 16 | General Purpose Buffer | WIRE assignment | HDL |
| BUFG | Global Clock Buffer | GLOBAL | Altera Primitive or Global Signal Assignment Editor setting |
| FD | D Flip Flop | DFF or DFFE | Altera Primitive |
| IBUF, 4, 8, 16 | Single and Multiple Input Buffers | WIRE assignment | HDL |
| IBUFG_<selectable I/O standard> | Input Global Buffer with selectable interface | WIRE and I/O Assignment with the Assignment Editor | HDL and I/O Standard Assignment Editor setting |
| IBUF_<selectable I/O standard> | Input Buffer with selectable I/O interface | WIRE and I/O Assignment with the Assignment Editor | HDL and I/O Standard Assignment Editor setting |
| IOBUF_<selectable I/O standard> | Bidirectional buffer with selectable I/O interface | WIRE and I/O Assignment with the Assignment Editor | HDL and I/O Standard Assignment Editor setting |
| OBUF, 4, 8, 16 | Single and Multiple Output Buffers | WIRE assignment | HDL |
| OBUF_<selectable I/O standard> | Output Buffer with selectable I/O interface | WIRE and I/O Assignment with the Assignment Editor | HDL and I/O Standard Assignment Editor setting |
| OBUFG_<selectable I/O standard> | Output Global Buffer with selectable I/O interface | WIRE and I/O Assignment with the Assignment Editor | HDL and I/O Standard Assignment Editor setting |
| SRL16 | 16-Bit Shift Register Look-Up-Table (LUT) | LPM_SHIFTREG or ALTSHIFT_TAPS | Altera MegaWizard |



The following is a sample of I/O standards that are supported by primitives with the `<selectable I/O standard>` parameter listed in [Table 5](#): LVTTTL (default), AGP, CTT, GTL, HSTL_I, LVCMOS2, LVCMOS18, LVDS, LVPECL, PCI33_3, PCI33_5, PCI66_3, PCIX, PCIX66_3, SSTL18_I, SSTL2_I, SSTL3_I, and SSTL3_II.

The following methods can be used to replace Xilinx primitives with Altera equivalents.

- Replace the primitive with an equivalent Altera primitive
- Replace the primitive with a user assignment in the Quartus II Assignment Editor, such as an I/O standard assignment
- Replace the primitive with an equivalent Altera function generated using the MegaWizard Plug-In Manager

Input, output, or bidirectional buffers are automatically inserted by the Quartus II Compiler. As a result, remove input, output, and bidirectional buffers used in a Xilinx design. A simple wire assignment is shown in the Verilog HDL code samples below. Before wire conversion:

```
module top (a, b, c, clk);

input  a, b, clk;
output c;

reg    c;
wire  clk_out;

//global buffer instantiation
BUFG inst1 (.I (clk), .O (clk_out));

always @ (posedge clk_out)
begin
    c<=a & b;
end

endmodule
```

After wire conversion:

```
module top (a, b, c, clk);

input  a, b, clk;
output c;

reg    c;
wire  clk_out;

//no need for buffer
```

```
//BUFG inst1 (.I (clk), .O (clk_out));

//simple wire assignment
assign clk_out = clk;

always @ (posedge clk_out)
begin
    c<=a & b;
end
endmodule
```

As an alternative to creating a wire to replace the buffers, you can also delete them. The Quartus II software automatically inserts the appropriate buffers in the design. Deleting these buffers requires that you replace the output of the buffer with the input into the buffer in your HDL. The following is an example of this process (in both VHDL and Verilog HDL):

```
module top (a, b, c, clk);
input      a, b, clk;
output     c;
reg        c;

//no need for wire
//wire clk_out;
//no need for buffer
//BUFG inst1 (.I (clk), .O (clk_out));
//replaced clk_out port with clk port

always @ (posedge clk)
begin
    c<=a & b;
end
endmodule
```

The following shows deletion of an input buffer in VHDL:

```
ENTITY top IS
    PORT(
        a, b : IN std_ulogic;
        clk : IN std_ulogic;
        c : OUT std_ulogic
    );
END top;
ARCHITECTURE behave OF top IS
--no need for clk_out signal
--signal clk_out : std_ulogic;
--no need of BUFG component
--component BUFG
--port (O : out STD_ULOGIC;
--I : in STD_ULOGIC);
```

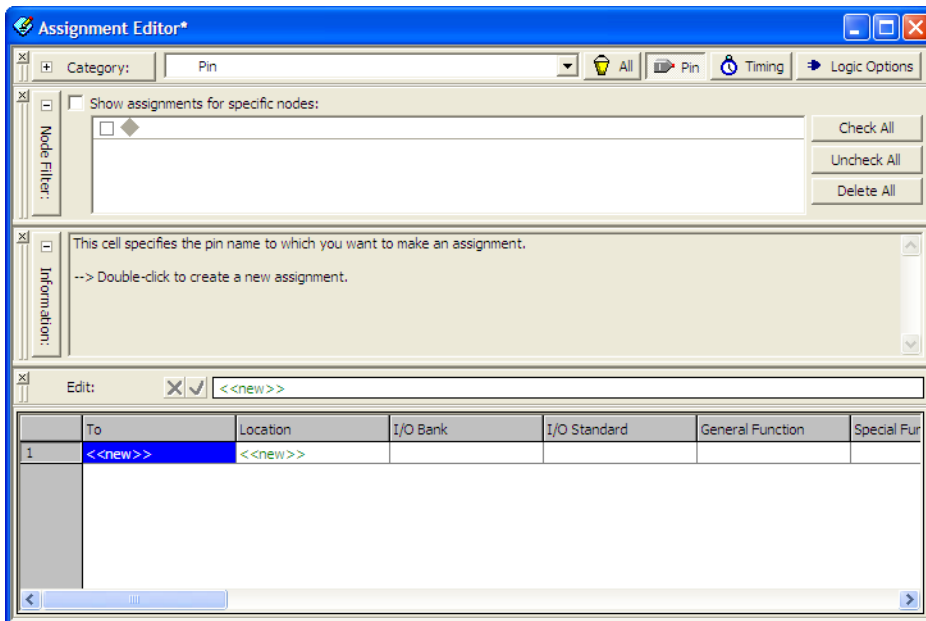
```

--end component;
BEGIN
--no need for port map
--inst1 : BUFG port map (O => clk_out, I => clk);
--replaced clk_out port with clk port
  PROCESS(clk)
  BEGIN
    IF (clk'event and clk = '1') then
      c <= a AND b;
    END IF;
  END PROCESS;
END behave;

```

Xilinx I/O standard primitives are converted into Quartus II assignments using the Assignment Editor. I/O standard assignments are not required to be declared within your HDL when designing with the Quartus II software. Instead you use the Assignment Editor to make I/O standard assignments. [Figure 15](#) shows an I/O standard assignment with the Assignment Editor.

Figure 15. I/O Standard Assignment with the Assignment Editor



RAM Architecture Functional Specifications

Stratix and Stratix GX devices feature the TriMatrix™ memory structure, composed of three sizes of embedded RAM blocks. TriMatrix memory includes 512-bit M512 blocks, 4-Kbit M4K blocks, and 512-Kbit M-RAM blocks, each of which can be configured to support a wide range of features.


 Cyclone™ devices contain M4K blocks only.

Table 8 summarizes the features supported by the three sizes of the TriMatrix memory in Stratix, Stratix GX and Cyclone devices.

| Feature | M512 Block | M4K Block | M-RAM Block |
|--|------------|-----------|-------------|
| Total RAM bits (including parity bits) | 576 | 4608 | 589,824 |
| Parity bits | ✓ | ✓ | ✓ |
| Byte Enable | | ✓ | ✓ |
| Single-port memory | ✓ | ✓ | ✓ |
| Simple dual-port memory | ✓ | ✓ | ✓ |
| True dual-port memory | | ✓ | ✓ |
| Embedded shift register | ✓ | ✓ | |
| Simple dual-port mixed width support | ✓ | ✓ | ✓ |
| True dual-port mixed width support | | ✓ | ✓ |
| Memory initialization (.mif and .hex) | ✓ | ✓ | |
| Mixed-clock mode | ✓ | ✓ | ✓ |

Table 8. Summary of RAM Architectural Features in Stratix, Stratix GX, and Cyclone Devices *Note (1) (Part 2 of 2)*

| Feature | M512 Block | M4K Block | M-RAM Block |
|------------------------------|---|---|---|
| Power-up condition | Outputs cleared | Outputs cleared | Outputs unknown |
| Register clears | Input and output registers (2) | Input and output registers (3) | Output registers |
| Same-port read-during-write | New data available at positive clock edge | New data available at positive clock edge | New data available at positive clock edge |
| Mixed-port read-during-write | Outputs set to unknown or old data | Outputs set to unknown or old data | Unknown output |

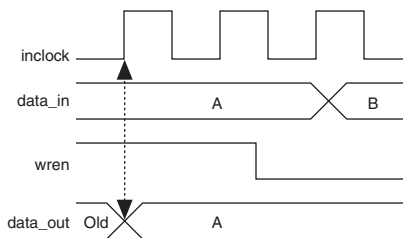
Notes to Table 8:

- (1) Cyclone devices do not contain M512 and M-RAM blocks.
- (2) The `rden` register on the M512 memory block does not have a clear port.
- (3) On the M4K block, asserting the clear port of the `rden` and `byte enable` registers drives the output of these registers high.

Differences exist in the RAM structure supported by Altera devices and Xilinx devices. These differences are detailed in the following sections.

Read-during-Write Operation at the Same Address

For RAM in Altera devices, there are two types of read-during-write operations: same-port and mixed-port. Figure 16 shows the output of the RAM during a write operation.

Figure 16. Same-Port Read-during-Write Functionality

In mixed-port read-during write operations, the same memory location is written to on one port and simultaneously read from a different port using the same clock for both.

The `READ_DURING_WRITE_MODE_MIXED_PORTS` parameter for M512 and M4K memory blocks determines whether to output the old data at the address or a “don’t care” value. Setting this parameter to `OLD_DATA` outputs the old data at that address. Setting this parameter to `DONT_CARE` outputs an unknown value. See Figures 17 and 18 for sample functional waveforms showing this operation. These figures assume that the outputs are not registered.

Figure 17. Mixed-Port Read-during-Write: OLD_DATA

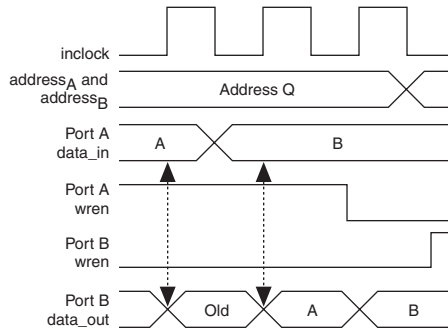
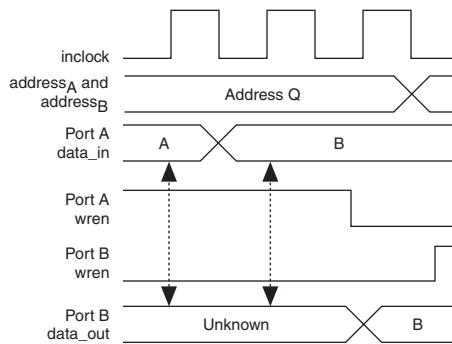


Figure 18. Mixed-Port Read-during-Write: DON'T_CARE



Designs containing Xilinx RAM read-during-write modes not supported by Altera RAM need to be modified.

Table 9 lists the output of the RAM during read-during-write operation at the same address for Altera RAM and Xilinx RAM (Virtex-II and Virtex-II PRO only).

| Feature | Virtex-II and Virtex-II PRO RAM | Altera M512 Block (1) | Altera M4K Block (1) (2) | Altera M-RAM Block (1) |
|------------------------------|---|--|--|--|
| Same-port read-during-write | The following modes are supported 1.READ_FIRST 2.WRITE_FIRST 3.NO_CHANGE | New data available at positive edge of the clock | New data available at positive edge of the clock | New data available at positive edge of the clock |
| Mixed-port read-during-write | The following modes are supported 2.WRITE_FIRST 3.NO_CHANGE | Outputs set to unknown or old data | Outputs set to unknown or old data | Unknown output |

Notes to Table 9:

- (1) Included in Stratix and Stratix GX devices.
 (2) Included in Cyclone devices.

Byte Enable

In M4K and M-RAM blocks, byte enables can mask the input data so that only specific bytes of data are written. The unwritten bytes retain the previous written value. Memory in Xilinx devices does not support this feature.



For more information on the byte enable feature, see the *TriMatrix Embedded Memory Blocks in Stratix & Stratix GX Devices* chapter in volume 2 of the *Stratix Device Handbook*.

SRVAL Constraint

In Virtex-II and Virtex-II Pro, the SRVAL constraint initializes the output of the memory to a user-defined value when the SSR signal is asserted. Memory in Altera devices does not support this feature.

Memory Port Configurations

Memory in Altera and Xilinx devices can be configured as single-port, simple dual-port, and true dual-port RAM:

- Single-port: Single-port mode supports non-simultaneous reads and writes.

- Simple dual-port: Simple dual-port mode supports a simultaneous read and write.
- True dual-port: True dual-port mode supports any combination of two-port operations: two reads, two writes, or one read and one write at two different clock frequencies.

The table below compares the different port configurations supported by Altera and Xilinx RAM.

| Feature | M512 Block (1) | M4K Block (1) (2) | M-RAM Block (1) | Xilinx Distributed RAM | Xilinx Block RAM |
|------------------|-------------------|----------------------|--------------------|------------------------------|---------------------|
| Single-Port | ✓ | ✓ | ✓ | ✓ | ✓ |
| Simple Dual-Port | ✓ | ✓ | ✓ | ✓ | ✓ |
| True Dual-Port | | ✓ | ✓ | | ✓ |

Notes to Table 10:

- (1) Included in Stratix and Stratix GX devices.
 (2) Included in Cyclone devices.

RAM Stitching

Stratix TriMatrix memory structures are optimized to implement different types of memory functions, providing a complete coverage of RAM applications. For example, the small M512 blocks are used for small first-in first-out (FIFO) functions. The M4K blocks are ideal for applications requiring medium-sized memory, such as asynchronous transfer mode (ATM) cell processing.



When creating memory modules with the MegaWizard Plug-In Manager, the stitching of RAM is not required. The MegaWizard Plug-In Manager will combine the appropriate TriMatrix memory blocks together to create the required memory width and depth.

The memory implemented in Xilinx distributed RAM modules can be combined together in the M512 or M4K blocks in Altera devices for better performance and more efficient use of logic resources. Similarly, smaller RAM modules created using Xilinx 18Kb Block RAMs can be transferred to Altera M4K or M512 blocks to use RAM resources most efficiently. See [“RAM Stitching Example” on page 48](#) for examples of RAM stitching.



The `altsyncram` megafunction can be configured to automatically assign the most appropriate RAM module for the requested amount of memory. For more information, see [“Creating Altera RAM Using the MegaWizard Plug-In Manager”](#) on page 47.

Converting Asynchronous RAM to Synchronous RAM

This section describes how to convert Xilinx asynchronous distributed RAM to Altera synchronous RAM.

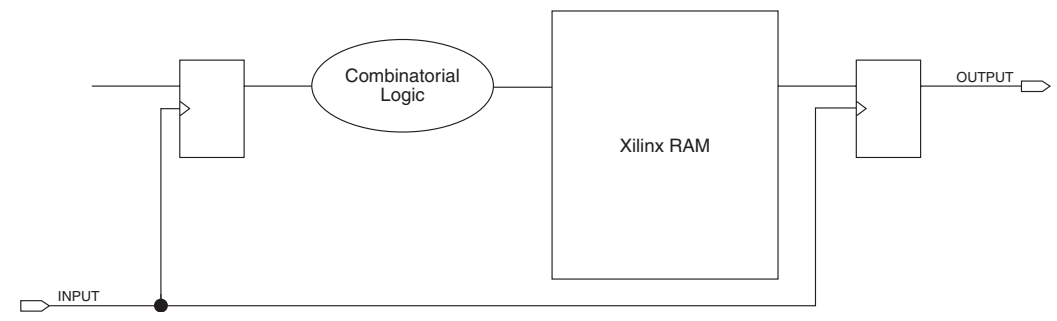
Xilinx distributed RAM supports synchronous write and asynchronous read operations. Xilinx Block RAM supports synchronous read and write operations. Altera RAM supports synchronous write and read for M512, M4K and M-RAM blocks. In addition, Altera RAM has an optional output register.

Where RAM has input and/or output registers in slices with no combinatorial logic between them, the design needs to be modified so that the Altera RAM absorbs the slice input and output registers. The converted design will have the same latency as the original design.

Where distributed RAM has no input register, the only option is to directly replace Xilinx asynchronous distributed RAM with Altera synchronous RAM. Latency of the converted design will change from 0 to 1. To maintain the functionality of the original design, modification may be required.

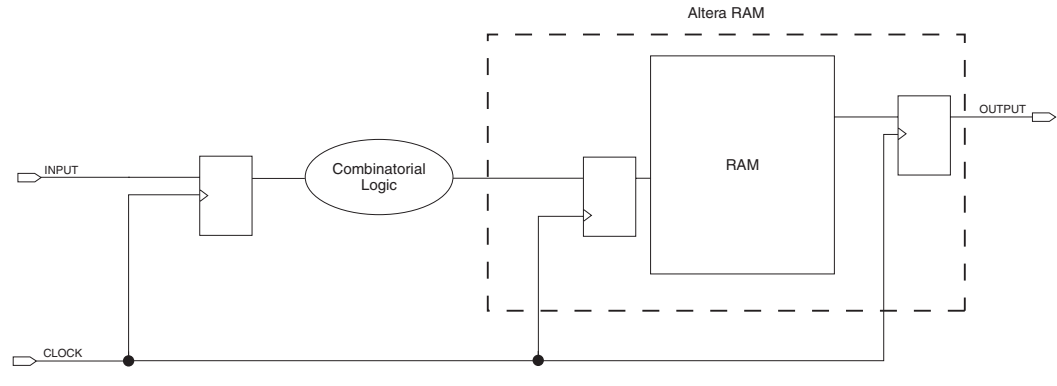
[Figure 19](#) illustrates a situation when combinatorial logic exists between the input register and the RAM. In this example, the data will be valid on the output two clock cycles after the address is valid at the input.

Figure 19. Original Xilinx Design With Combinatorial Logic Between the Registers and the RAM



The first option for solving this problem uses straight replacement of Xilinx memory with Altera memory as indicated in Figure 20 below. When an address is valid on the input, the data will be valid on the output three clock cycles later. The latency of the design is increased by one clock cycle. To maintain the functionality of the original design, the converted design might need modifications.

Figure 20. Replacement of Xilinx RAM with Altera RAM



The second option for solving this problem replaces Xilinx memory with Altera memory, inverting the clock (Figure 21). This option does not add any additional cycles of latency. When address is valid on the input, the data will be valid on the output two clocks later. However, meeting timing requirements can be a challenge with this approach. The input signal has only one half of a clock cycle to propagate from register 1 to register 2. The following timing equation has to be satisfied for the design to function without errors:

$$T_{CO}(\text{Register 1}) + T(\text{Combinatorial Logic}) + T_{SU}(\text{Register 2}) < \frac{T}{2}$$

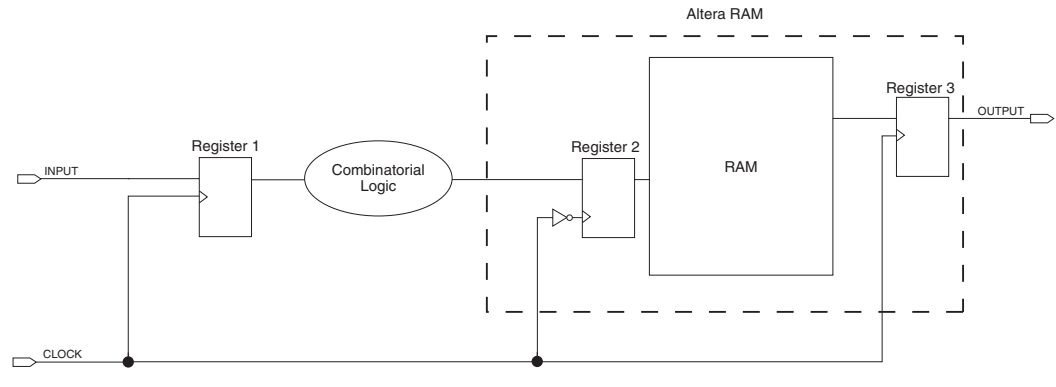
where:

T_{CO} (Register 1) = the clock to output time of register 1

T (Combinatorial logic) = the time taken for the data to propagate through the combinatorial logic

T_{SU} (Register 2) = the setup time of register 2

T = the time period of the input clock

Figure 21. Substitution of Xilinx Memory with Altera Memory, Inverting the Clocks

After you convert an asynchronous RAM to a synchronous RAM, check latency of the converted design. Further changes may be needed to maintain functionality of the original design. Make sure the timing parameters of the converted RAM are within timing budget limitations. Perform timing/functional simulation to ensure the converted design functions as expected.

Port Mapping Between Altera RAM and Xilinx RAM

Table 11 lists the `altsyncram` megafunction ports (when used in true dual-port mode) and the corresponding Xilinx Block RAM primitive ports.

Table 11. Port Mapping from the Xilinx Block RAM Primitive to the Altera `altsyncram` Megafunction in True Dual Port Mode (Part 1 of 2)

| Feature | Altera <code>altsyncram</code> Megafunction | Xilinx Block RAM |
|-----------------------------------|---|------------------|
| Address Port A | <code>address_a</code> | ADDRA |
| Data Port A | <code>data_a</code> | DIA |
| Write Enable Port A | <code>wren_a</code> | WEA |
| Enable Port A | <code>enable_a</code> | ENA |
| Synchronous Initialization Port A | See Table 13 | SINITA |
| Clock Port A | <code>clock_a</code> | CLKA |
| Output Port A | <code>q_a</code> | DOA |
| Address Port B | <code>address_b</code> | ADDRB |
| Data Port B | <code>data_b</code> | DIB |

Table 11. Port Mapping from the Xilinx Block RAM Primitive to the Altera altsyncram Megafunction in True Dual Port Mode (Part 2 of 2)

| Feature | Altera altsyncram Megafunction | Xilinx Block RAM |
|-----------------------------------|--------------------------------|------------------|
| Write Enable Port B | wren_b | WEB |
| Enable Port B | enable_b | ENB |
| Synchronous Initialization Port B | See Table 13 | SINITB |
| Clock Port B | clock_b | CLKB |
| Output Port B | q_b | DOB |

[Table 12](#) lists the altsyncram megafunction ports (when used in single-port and simple dual-port modes) and the corresponding Xilinx distributed RAM primitive ports.

Table 12. Port Mapping from Xilinx Distributed RAM Primitive to the Altera altsyncram Megafunction in Single Port and Simple Dual Port Modes

| Feature | Altera altsyncram Megafunction | Xilinx Distributed RAM |
|-------------------------|--------------------------------|------------------------|
| Data Port | data | D |
| Write Address Port | wraddress | A |
| Read Address Port | rdaddress | DPRA |
| Read Enable Port | rden | - |
| Write Clock Port | wrclock | WCLK |
| Read Clock Port | rdclock | - |
| Write Enable Port | wren | WE |
| Read Clock Enable Port | rdclocken | - |
| Write Clock Enable Port | wrclocken | - |
| Clear for Read Port | rd_clr | - |
| Clear for Write Port | wr_clr | - |
| Output | q | DPO |

Table 13 lists the `altsyncram` megafunction ports and the corresponding Xilinx Block RAM CoreGen module ports.

| Table 13. Port Mapping from the Xilinx Block RAM CoreGen Module to the Altera <code>altsyncram</code> Megafunction in True Dual Port Mode | | |
|--|--|-------------------------|
| Feature | Altera <code>altsyncram</code> Megafunction | Xilinx Block RAM |
| Address Port A | <code>address_a</code> | ADDRA |
| Data Port A | <code>data_a</code> | DINA |
| Write Enable Port A | <code>wren_a</code> | WEA |
| Enable Port A | <code>enable_a</code> | ENA |
| Synchronous Initialization Port A | See Table 11 and Table 12 | SINITA |
| Clock Port A | <code>clock_a</code> | CLKA |
| Output Port A | <code>q_a</code> | DOUTA |
| Address Port B | <code>address_b</code> | ADDRB |
| Data Port B | <code>data_b</code> | DINB |
| Write Enable Port B | <code>wren_b</code> | WEB |
| Enable Port B | <code>enable_b</code> | ENB |
| Synchronous Initialization Port B | See Table 11 and Table 12 | SINITB |
| Clock Port B | <code>clock_b</code> | CLKB |
| Output Port B | <code>q_b</code> | DOUTB |

Table 14 lists the `altsyncram` megafunction ports (when used in single port and simple dual port modes) and the corresponding Xilinx distributed RAM CoreGen module ports.

| Table 14. Port Mapping from the Xilinx Block RAM CoreGen Module to the Altera <code>altsyncram</code> Megafunction in Single Port and Simple Dual Port Modes (Part 1 of 2) | | |
|---|--|-------------------------------|
| Feature | Altera <code>altsyncram</code> Megafunction | Xilinx Distributed RAM |
| Write Address | <code>wraddress</code> | A |
| Input Clock | <code>inclock</code> | CLK |
| Data Input | <code>data</code> | D |
| Write Enable | <code>wren</code> | WE |
| Input Clock Enable | <code>inclocken</code> | I_CE |

Table 14. Port Mapping from the Xilinx Block RAM CoreGen Module to the Altera altsyncram Megafunction in Single Port an Simple Dual Port Modes (Part 2 of 2)

| Feature | Altera altsyncram Megafunction | Xilinx Distributed RAM |
|---------------------------|--------------------------------|------------------------|
| Read Address | rdaddress | DPRA |
| Output Clock Enable | outclocken | QDPO_CE |
| Non-registered Output | q | DPO |
| Registered Output | q | QDPO |
| Output Asynchronous Reset | out_aclr (1) | QDPO_RST |
| Output Synchronous Reset | out_aclr (1) | QDPO_SRST |
| Output Clock | outclock | QDPO_CLK |

Note to Table 14:

- (1) altsyncram does not support synchronous and asynchronous reset. The port can be connected to the “out_aclr” port.

Creating Altera RAM Using the MegaWizard Plug-In Manager

The MegaWizard Plug-In Manager is a GUI that enables users to quickly and easily specify parameters for Altera-specific functions. Using the MegaWizard Plug-In Manager, the Memory Compiler can be used to instantiate single- and multi-port RAM in Stratix, Stratix GX, and Cyclone devices. The following options allow you to customize the megafunctions contained in the Memory Compiler to meet requirements.

- The altsyncram function can be used in one of the following ways:
 - With one read port (ROM mode)
 - With one read/write port (Single-port mode)
 - With one read and one write port (Simple dual-port mode)
 - With two read/write ports (True dual-port mode)
- You can specify the memory size in terms of number of bits or number of words
- You can select the width and depth of RAM
- You can select different widths for different ports
- There are three options available to select the clocking scheme for the RAM:

- Single clock
- Dual clock-use separate 'read' and 'write' clocks
- Dual clock-use separate 'input' and 'output' clocks

- Option to turn the output register on or off

- Option to create asynchronous clear and enable signals

- Four options to select the TriMatrix memory block type to be used to implement the memory function. Choosing Auto allows Quartus II software to choose the TriMatrix block type during compilation. Alternately, choosing M512, M4K, or M-RAM options specify the TriMatrix block to be used.

- The initial memory content can be specified by using a Memory Initialization File (.mif) or Hexadecimal (Intel-Format) file (.hex)



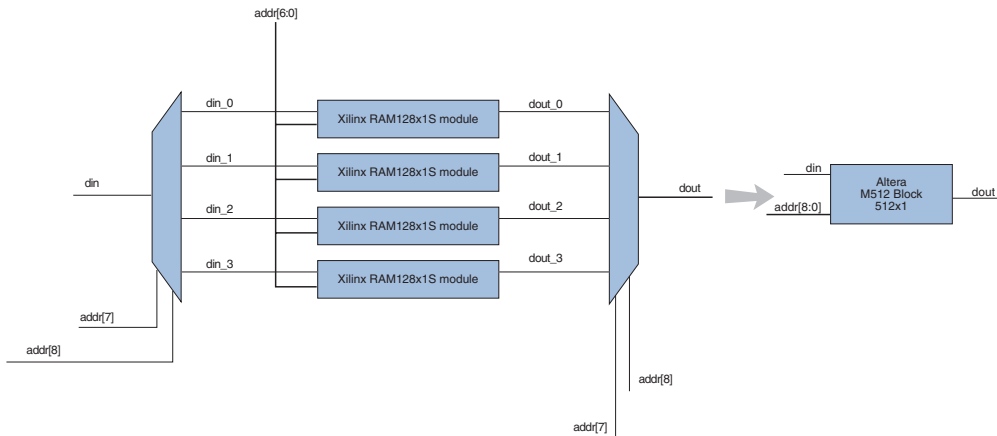
For more information on the `altsyncram` megafunction, refer to *AN 207: TriMatrix Memory Selection using the Quartus II Software*.

Examples

This section contains examples on RAM stitching and Verilog code before and after RAM conversion.

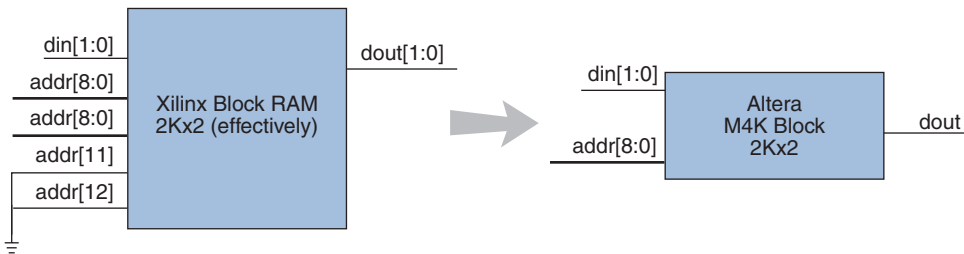
RAM Stitching Example

In this example, four Xilinx 128 × 1S distributed RAM modules are stitched together to form one 512 × 1 RAM block (Figure 22). The extra logic and routed resources required to stitch up the RAMs can be saved by implementing a 512 × 1 sized RAM in an Altera M512 Block.

Figure 22. Four Xilinx RAM128x1 RAM Modules Converted to One Altera M512 Block

RAM Limiting Example

In [Figure 23](#), the 18Kb Xilinx block RAM is being used as a 4K RAM block by grounding two address bits. RAM resources are not being used efficiently since 8K RAM bits cannot be accessed. The same design can be implemented in one Altera M4K block.

Figure 23. Xilinx 16K Block RAM Converted to Altera M4K RAM

Example Verilog Code Containing Xilinx RAM Modules Before and After Conversion

In this section sample Verilog code targeting a Xilinx Virtex-II device is converted to target Altera Stratix, Stratix GX, or Cyclone devices. Below is the original Verilog code targeting a Virtex-II device. A 512×36 RAM is instantiated by the RAMB16_S36_S36 Xilinx primitive.

The following is Verilog code targeting a Xilinx Virtex-II device:

```

module 512x36dualport (CLKA, ENA, SSRA, WEA, ADDRA, DIA, DOA,
CLKB, ENB, SSRB, WEB, DDRB, DIB, DOB, DOPA, DOPB, DIPB, DIPB);

input          CLKA;    // Port A clock
input          ENA;    // Port A select
input          SSRA;   // Port A reset, active low
input          WEA;    // Port A direction control, 1=write
input [ 8:0]   ADDRA;  // Port A address
input [31:0]   DIA;    // Port A input data
input [ 3:0]   DIPB;   // Port A input data parity
output [31:0]  DOA;    // Port A output data
output [ 3:0]  DOPA;   // Port A output data parity
input          CLKB;   // Port B clock
input          ENB;   // Port B select
input          SSRB;  // Port B reset, active low
input          WEB;   // Port B direction control, 1=write
input [ 8:0]   DDRB;  // Port B address
input [31:0]   DIB;   // Port B input data
input [ 3:0]   DIPB;  // Port B input data parity
output [31:0]  DOB;   // Port B output data
output [ 3:0]  DOPB;  // Port B output data parity

RAMB16_S36_S36 ram( .ADDRA(ADDRA),
                    .CLKA(CLKA),
                    .DIA(DIA),
                    .DIPA(DIPA),
                    .WEA(WEA),
                    .ENA(ENA),
                    .SSRA(SSRA),
                    .DOA(DOA),
                    .DOPA(DOPA),
                    .ADDRB(ADDRB),
                    .CLKB(CLKB),
                    .DIB(DIB),
                    .DIPB(DIPB),
                    .WEB(WEB),
                    .ENB(ENB),
                    .SSRB(SSRB),
                    .DOB(DOB),
                    .DOPB(DOPB));

endmodule;

```

To target the sample Verilog code to Altera devices, only the RAMB16_S36_S36 module was altered. The following changes were made to the original Verilog code:

- The data and parity inputs in Xilinx RAM have different input ports. In Altera RAM, parity and data inputs have the same ports. To account for this difference, the following modification was made to the code.

```

assign data_in_a = {DIPA, DIA}
assign data_in_b = {DIPB, DIB}

```

- The data and parity outputs from the Xilinx RAM have different output ports (DOA, DOPA). For Altera RAM, the most significant bits of the q_a/b ports correspond to the parity bits. To account for this difference, the following modification was made to the code.

```
assign DOA    = data_out_a[31:0];
assign DOPA   = data_out_a[35:32];
assign DOB    = data_out_b[31:0];
assign DOPB   = data_out_a[35:32];
```

- The ramb16_s36_s36_altera module was created from the altsyncram megafunction using the MegaWizard Plug-In Manager. The MegaWizard Plug-In Manager generates a **RAM16_s36_s36_altera.vhd** file that should be included with the 512x36dualport module during compilation.

This portion of the code replaces the RAMB16_S36_S36 module in the original code. Everything else remains the same.

```
wire [36:0] data_in_a, data_in_b, data_out_a, data_out_b;
assign data_in_a = {DIPA, DIA};
assign data_in_b = {DIPB, DIB};
```

```
ramb16_s36_s36_alteraramb16_s36_s36_altera_inst (
  data_a ( data_in_a ),
  wren_a ( WEA ),
  address_a ( ADDRA ),
  data_b ( data_in_b ),
  address_b ( ADDRb ),
  wren_b ( WEB ),
  clock_a ( CLKA ),
  enable_a ( ENA ),
  clock_b ( CLKb ),
  enable_b ( ENB ),
  q_a ( data_out_a ),
  q_b ( data_out_b ) );
```

```
assign DOA = data_out_a[31:0];
assign DOPA = data_out_a[35:32];
assign DOB = data_out_b[31:0];
assign DOPB = data_out_b[35:32];
```

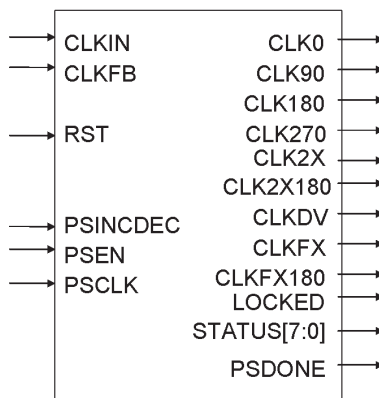
DCM and DLL Conversion

To increase device and board-level performance, some Altera and Xilinx FPGA device families offer support for phase-locked loops (PLLs) and digital clock managers (DLL / DCM) respectively. These specialized blocks allow you to minimize clock skew and clock delay and provide support for clock synthesis.

Architectural Description

In place of the DLLs / DCMs provided for clock skew, frequency synthesis, and phase shifting in the Xilinx Spartan-IIIE, Virtex-II and Virtex-II Pro families (see [Figure 24](#)), Altera Stratix devices contain two types of PLLs: enhanced PLLs and fast PLLs. The enhanced PLLs provide you with complete control over clocks and system timing. The fast PLLs provide general-purpose clocking with multiplication and phase shifting as well as high-speed output for high-speed differential I/O support.

Figure 24. Virtex-II DCM



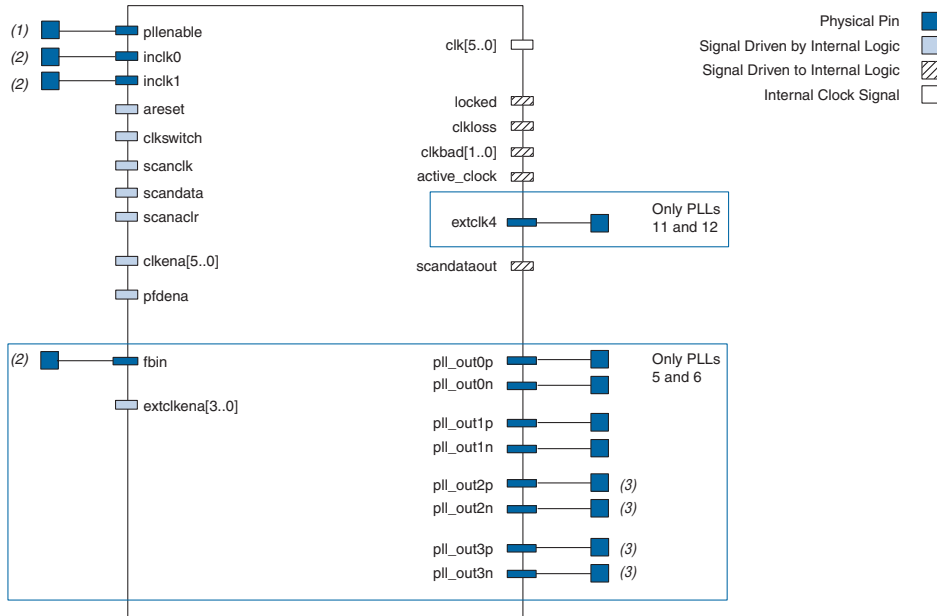
Enhanced PLLs

The enhanced PLLs available in Stratix devices are highly versatile and robust PLLs that support the following features:

- Clock multiplication and division
- Clock switchover
- Phase and delay shifting
- Clock feedback
- PLL reconfiguration
- Programmable bandwidth
- External clock outputs
- Spread-spectrum clocking
- Lock detect and programmable gated lock
- Programmable duty cycle
- Advanced clear and enable control

This section will briefly cover clock multiplication and division, clock switchover, phase and delay shifting, and clock feedback. [Figure 25](#) shows a block diagram of the signals used by an enhanced PLL.

Figure 25. Enhanced PLL Signals



For more detailed information on Stratix enhanced PLLs, refer to the *General-Purpose PLLs in Stratix & Stratix GX Devices* chapter in volume 2 of the *Stratix Device Handbook*.

Clock Multiplication and Division

Enhanced PLLs perform clock multiplication and division using $m / (n \times \text{post-scale counter})$ as scaling factors for the output of the PLL. The input clock is divided by a pre-scaler divider, n , and is then multiplied by the m feedback factor. This ability allows you to customize each output of the enhanced PLL to the requirements of your design.

Virtex-II DCM limits you to 3 customizable output ports; CLKDV, CLKFX, and CLKFX180 for clock multiplication and division. The output port CLKDV supports clock division in the form of $(1/n) \times \text{CLK0}$. CLKFX

provides clock multiplication in the form of $(\text{CLKFX_MULTIPLY_value} / \text{CLKFX_DIVIDE_value}) \times \text{CLK0}$. CLKFX180 is limited to the same frequency multiplication as CLKFX except with a 180° shift.

Clock Switchover

Stratix-enhanced PLLs offer a flexible clock switchover capability unavailable in the Virtex-II Pro DCM. This configurable capability in Stratix devices provides an effective means of developing high-reliability systems which must contain multiple clocks to provide redundancy. Clock-sense circuitry automatically switches from the PLL reference primary clock to the secondary clock when the primary clock signal is not present.

Phase and Delay Shifting

Phase shifting is implemented in the Stratix enhanced PLLs by specifying a phase shift (in degrees or time units) for each PLL clock output port or for all outputs together. In addition to the phase-shift feature, the fine tune clock delay shift feature provides advanced time delay shift control on each of the four PLL outputs. Each PLL output shifts in 250-ps increments for a range of +/-3.0 ns between any two outputs using discrete delay elements.

Phase shifting is also supported by Virtex-II Pro DCM defined attributes associated with the DCM instantiation.

Clock Feedback

Stratix enhanced PLLs support the following clock feedback modes:

- Zero delay buffer: The external clock output pin is phase-aligned with the clock input pin for zero delay.
- External feedback: The external feedback input pin, FBIN, is phase-aligned with the clock input, CLK, pin. Aligning these clocks allows the designer to remove clock delay and skew between devices.
- Normal mode: If an internal clock is used in this mode, it is phase-aligned to the input clock pin. The external clock output pin will have a phase delay relative to the clock input pin if connected in this mode.

Fast PLLs

To complement the enhanced PLLs, Stratix devices also provide fast PLLs with high-speed differential I/O interface ability and general-purpose features. The fast PLLs support the following features:

- Clock multiplication and division
- External clock inputs
- External clock outputs
- Phase shifting
- Control signals

This section will briefly cover clock multiplication and division and external clock inputs.



For more detailed information on Stratix enhanced PLLs, refer to the *General-Purpose PLLs in Stratix & Stratix GX Devices* chapter in volume 2 of the *Stratix Device Handbook*.

Clock Multiplication and Division

Fast PLLs provide clock synthesis for PLL output ports using m / (post scaler) scaling factors. The input clock is multiplied by the m feedback factor. Each output port has a unique post scale counter to divide down the high-frequency V_{CO} .

In the case of a high-speed differential interface, you can set the output counter to 1 to allow the high-speed V_{CO} frequency to drive the dedicated serializer/deserializer (SERDES) circuitry.

External Clock Inputs

Each fast PLL supports single-ended or differential inputs for source-synchronous transmitters or for general-purpose use. Source-synchronous receivers support differential clock inputs.

Converting DLLs / DCMs

You can easily convert DCMs that target a Xilinx device into PLLs in an Altera device using the MegaWizard Plug-In Manager. Unlike the Virtex-II DCM, which requires specific input buffers to feed into the source clock port of the DCM, e.g., `IBUF`, `IBUFG`, or `BUFGMUX`, PLLs in Altera devices do not require input buffers.

When converting DCMs, you can use the `altpll` megafunction. This megafunction will allow you to create and customize your PLLs targeting Stratix, Stratix GX, or Cyclone device families.

Table 15 summarizes the port mapping between the DCM and the `altpll` megafunction.

| Xilinx Port | Altera Port | Comment |
|--|------------------------------------|--|
| CLKIN | inclk0 | |
| CLKFB | fbin | |
| RST | areset | |
| PSINCDEC | Not Supported | |
| PSEN | Not Supported | |
| PSCLK | Not Supported | |
| PSDONE | Not Supported | |
| CLK0, CLK90, CLK180, CLK270, CLK2X, CLK2X180, CLKDV, CLKFX, CLKFX180 | Supported by all outputs of ALTPLL | Set the output of the PLL to correspond to that of the output of the DCM |
| LOCKED | locked | |
| STATUS[7:0] | Not Supported | |

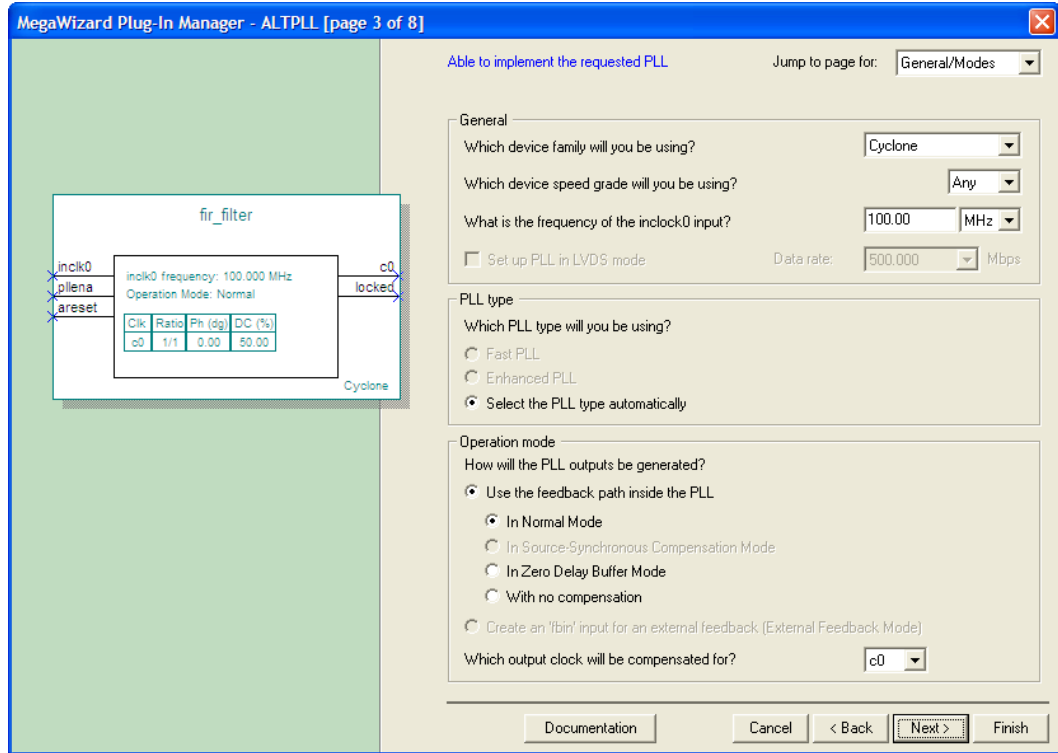
As shown in Table 15, all outputs of `altpll` can be configured to support any of the clock synthesis ports of the DCM. This allows you to combine multiple DCMs into one instance of the `altpll` megafunction. For example, to implement a multiplication factor of 2/3 and 6/5, you would require two `CLKFX` ports in a Xilinx device. However, one `altpll` instance can achieve the same functionality by applying the multiplication factor of 2/3 to clock `c0` and applying 6/5 to clock `c1` or vice versa.

Implementing Altera PLLs Using the MegaWizard Plug-In Manager

The `altpll` megafunction allows you to configure either the enhanced or fast PLL in Stratix or Stratix GX devices. This megafunction will also allow you to configure PLLs in Cyclone devices. This section is a brief description of `altpll` megafunction creation using the MegaWizard.

The third page of the `altpll` megafunction in the MegaWizard Plug-In Manager allows you to customize the general settings of the PLL, such as the PLL type to be used (enhanced or fast), clock frequency into the PLL, and the creation of optional ports. Figure 26 shows the first page of the `altpll` megafunction.

Figure 26. Page 3 of the altpll Megafunction in the MegaWizard Plug-In Manager



Page 4 allow you to set parameters for the scan and lock, bandwidth and spread spectrum, and clock switchover capabilities of the PLL.

Pages 5 through 7 allow you to customize the outputs of the PLL feeding into the core of the device such as multiplication factor, division factor, and phase shifts.

Page 8 allows you to check or uncheck files that are created by the MegaWizard Plug-In Manager.



For more information on the altpll Megafunction, refer to the *altpll Megafunction User Guide*.

Multiplier Conversion

The basic building blocks of all Digital Signal Processing (DSP) applications are high-performance multiply-adders and multiply-accumulators. To address this requirement in FPGA devices,

Altera's Stratix devices offer dedicated DSP blocks, combining five arithmetic operations—multiplication, addition, subtraction, accumulation, and summation—into a single block. Xilinx's Virtex II Pro devices are limited to offering only embedded multipliers to perform multiplications only.

Architectural Description

You can configure a single Stratix DSP block to perform any of the following functions:

- Simple multiply
- Multiply accumulate
- Multiply add

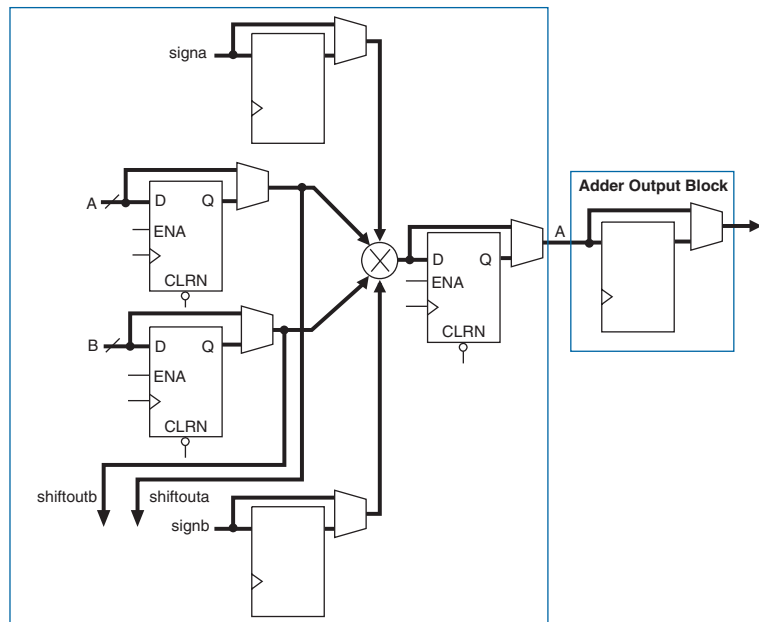
The dedicated multipliers in Virtex-II are known as Embedded Multipliers. These multipliers provide a dedicated 18×18 multiplication function in either combinatorial or pipelined form.

Simple Multiplier Mode

In simple multiplier mode, the Stratix DSP block performs individual multiplication operations. This mode allows you to configure a single DSP block to perform one of the following operations:

- Eight 9×9 bit multiplications
- Four 18×18 bit multiplications
- One 36×36 bit multiplication

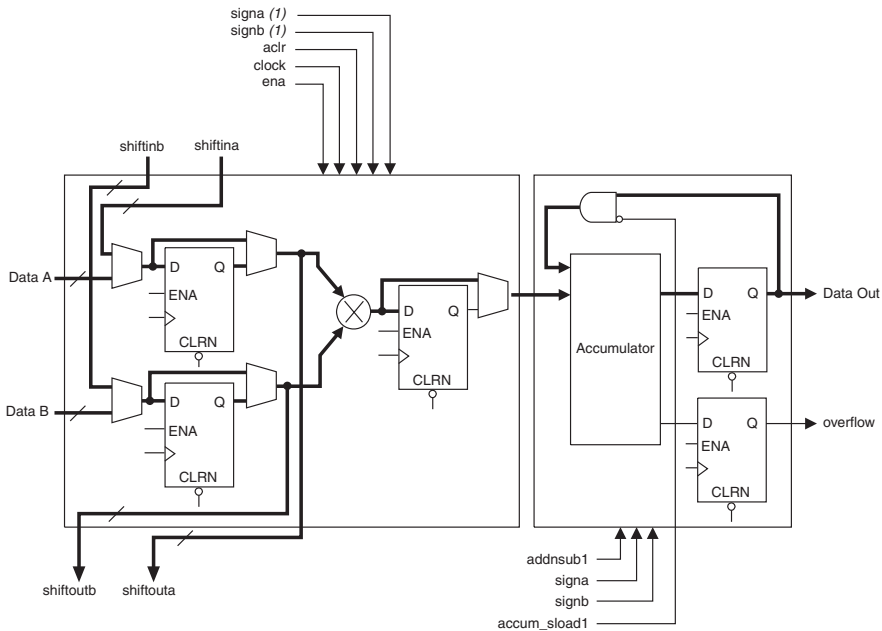
The multiplier operands can accept signed integers, unsigned integers, or a combination as determined by the `signa` and `signb` signals. [Figure 27](#) shows the simple multiplier mode.

Figure 27. Stratix DSP Block in Simple Multiplier Mode


You can implement various independent multipliers or a single large multiplier using a single DSP block. This preserves valuable logic resources when creating simple multipliers.

Multiply Accumulate Mode

In multiply accumulate mode, the output of the multiplier stage feeds the adder/output block, which is configured as an accumulator or subtractor. You can implement up to two independent 18-bit multiply accumulators in one DSP block. [Figure 28](#) shows multiply accumulate mode.

Figure 28. Stratix DSP Block in Multiply Accumulate Mode

Virtex-II Embedded multipliers require additional logic resources to carry out the same multiply accumulator mode.

 In multiply accumulator mode, Stratix DSP blocks are also capable of implementing adder and accumulator functionality previously implemented in the slices of Xilinx devices.

Multiply Add Modes

There are two multiply add modes: two-multiplier adder and four-multiplier adder.

Two-Multiplier Adder

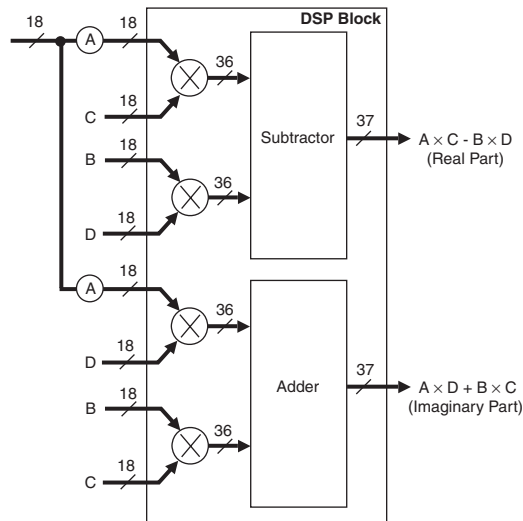
In this mode, the DSP block outputs two sums or differences for multipliers up to 18 bits, or four sums or differences for 9-bit or smaller multipliers. A single DSP block can implement one 18×18 -bit complex multiplier or two 9×9 -bit complex multipliers.

A complex multiplication can be written as:

$$(a + jb) \times (c + jd) = (a \times c - b \times d) + j \times (a \times d + b \times c)$$

In this mode, a single DSP block calculates the real part ($a \times c - b \times d$) using one adder/subtractor/accumulator and the imaginary part ($a \times d + b \times c$) using another adder/subtractor/accumulator for data up to 18 bits. [Figure 29](#) shows an 18-bit complex multiplication.

Figure 29. Complex Multiplier Implemented Using Two-Multiplier Adder Mode



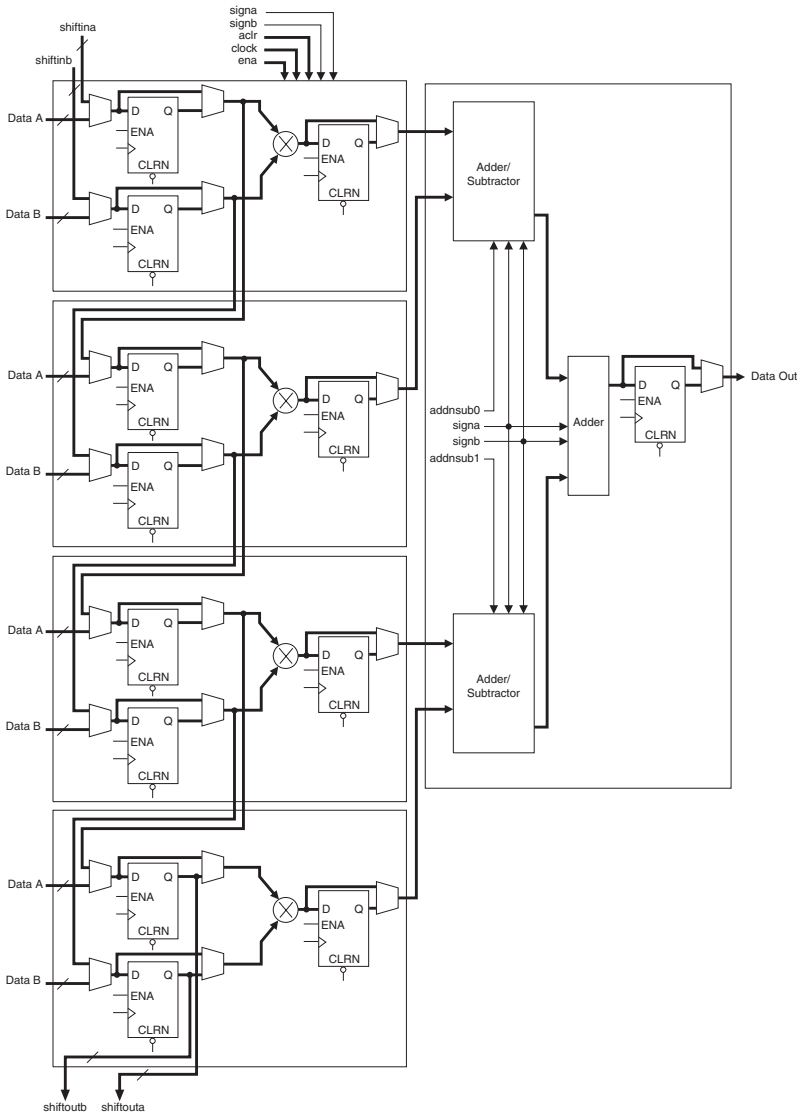
The multiply add modes of Stratix DSP blocks are also capable of implementing adder and accumulator functionality previously implemented in the slices of Xilinx devices.

Four-Multiplier Adder Mode

In the four-multiplier adder mode, which you can use for 1-dimensional and 2-dimensional filtering applications, the DSP block adds the results of two adder/subtractor/accumulators in a final stage. [Figure 30](#) shows four-multiplier adder mode.

Virtex-II embedded multipliers require additional logic resources to carry out the same multiplier adder modes.

Figure 30. Stratix DSP Block in Four-Multiplier Adder Mode



For more detailed information on Stratix's DSP blocks, refer to the *Using the DSP Blocks in Stratix & Stratix GX Devices* chapter in volume 2 of the *Stratix Device Handbook*.

Table 16 summarizes the resource comparison of implementing multipliers with Stratix's DSP blocks to multipliers in Virtex-II Pro Embedded Multipliers.

| Multiplier Size | Stratix DSP Block Resources | Stratix Logic Element Resources | Virtex-II 18 × 18 Multiplier Block Resources | Virtex II Logic Element Resources (3) |
|--------------------------------|-----------------------------|---------------------------------|--|---------------------------------------|
| Signed 9x9 | 1/8 (1) | 0 [0] (2) | 1 | 0 [36] (2) |
| Signed 18x18 | 1/4 (1) | 0 [0] (2) | 1 | 0 [72] (2) |
| Signed 36x36 | 1 | 0 [0] (2) | 4 | 326 [397] (2) |
| 18 x 18 Multiply Accumulate | 1/2 | 0 [0] (2) | 1 | 49 [134] (2) |
| 18 x 18 Complex Multiplication | 1 | 0 [0] (2) | 4 | 76 [153] (2) |

Notes to Table 16:

- (1) One DSP block can be configured as 8 independent 9 × 9 multipliers, 4 independent 18x18 multipliers, or 1 36 × 36 multiplier.
- (2) The numeric value in brackets [] indicates the number of LEs needed when both input and output signals are registered.
- (3) This is the number of LE-equivalent elements.



For more information on logic comparison, refer to the white paper *An Analytical Review of FPGA Logic Efficiency in Stratix, Virtex-II & Virtex-II Pro*.

Converting Multipliers

You can easily convert CoreGen Multipliers that target a Xilinx device into multipliers for an Altera device using the Quartus II MegaWizard Plug-In Manager. Similar to CoreGen multipliers, the Quartus II MegaWizard Plug In Manager multipliers can use either logic elements or the dedicated multiplier blocks in the device.

When converting CoreGen multipliers, you can use either the `lpm_mult` or `altmult_add` megafunction. With both of these megafunctions, you can create multipliers that use logic elements or dedicated multipliers using Stratix or Stratix GX DSP blocks.

Use the following guidelines with the `lpm_mult` megafunction when replacing CoreGen multipliers.

- Port B can either be a constant or dynamic value
- Both input ports must be of the same sign

If your design does not meet these requirements, you can use the `altmult_add` megafunction to replace CoreGen multipliers.

The required options for the `lpm_mult` megafunction to operate in a similar manner as the CoreGen multiplier function are:

- Disable the 'sum' input port option
- Specify the sign of the multiplier
- If a constant value is used for port B, specify this value
- Specify a pipeline of 2 to register both inputs and outputs. Specify a pipeline of 1 to register only inputs. This value should match the output latency for the CoreGen multiplier.

Figure 31 shows the `lpm_mult` megafunction.

Figure 31. `lpm_mult` Megafunction

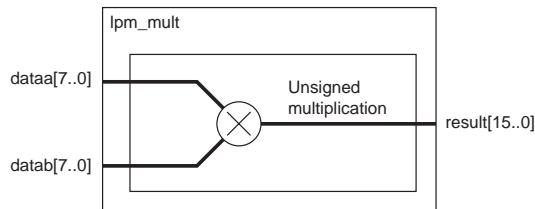


Table 17 summarizes the port mapping between the CoreGen multiplier and `lpm_mult`.

Table 17. CoreGen Multiplier and `lpm_mult` Port Mapping Comparison (Part 1 of 2)

| Xilinx Port | Altera Port | Comment |
|-------------|-------------|-------------------------|
| A | dataa | Data Input Port |
| B | datab | Data Input Port |
| CE | clken | Clock Enable |
| CLK | clock | Clock Port |
| ACLR | aclr | Asynchronous Clear Port |

Table 17. CoreGen Multiplier and lpm_mult Port Mapping Comparison (Part 2 of 2)

| Xilinx Port | Altera Port | Comment |
|-------------|---------------|--|
| Q | result | with registered outputs set |
| O | result | Without registered outputs set |
| A_SIGNED | N/A | lpm_mult allows both ports to be either signed or unsigned |
| LOADB | Not Supported | |
| SWAPB | Not Supported | |
| SCLR | Not Supported | |
| LOAD_DONE | Not Supported | |
| RDY | Not Supported | Hand Shaking Signal |
| RFD | Not Supported | Hand Shaking Signal |
| ND | Not Supported | Hand Shaking Signal |

You can also use the `altmult_add` megafunction (Figure 32) to replace the CoreGen multiplier function if your design does not meet the requirements for the `lpm_mult` megafunction. You can register inputs and outputs with the `altmult_add` megafunction; the sign of port A can be dynamic, and the signs of the input ports can be different.

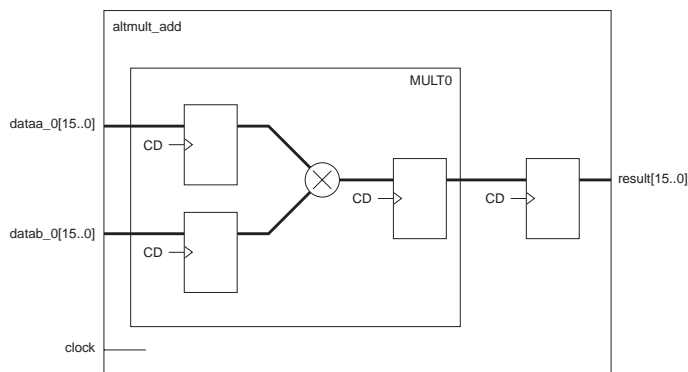
Figure 32. altmult_add Megafunction

Table 18 summarizes the port mapping between CoreGen's multiplier and the altmult_add megafunction.

Table 18. CoreGen multiplier and altmult_add Port Mapping Comparison

| Xilinx Port | Altera Port | Comment |
|-------------|---------------|---------------------------------|
| A | dataa | Data Input Port |
| B | datab | Data Input Port |
| CE | ena0 | Clock Enable |
| CLK | clock0 | |
| ACLR | aclr3 | |
| Q | result | With registered outputs set |
| O | result | Without registered outputs set |
| A_SIGNED | signa | No Registering of port required |
| LOADB | Not Supported | |
| SWAPB | Not Supported | |
| SCLR | Not Supported | |
| LOAD_DONE | Not Supported | |
| RDY | Not Supported | Hand Shaking Signal |
| RFD | Not Supported | Hand Shaking Signal |
| ND | Not Supported | Hand Shaking Signal |

Conversion of Virtex-II Pro's MULT18 × 18 can also be converted using either the `lpm_mult` or `altmult_add` megafunction.

Implementing Altera DSP Blocks Using the MegaWizard Plug-In Manager

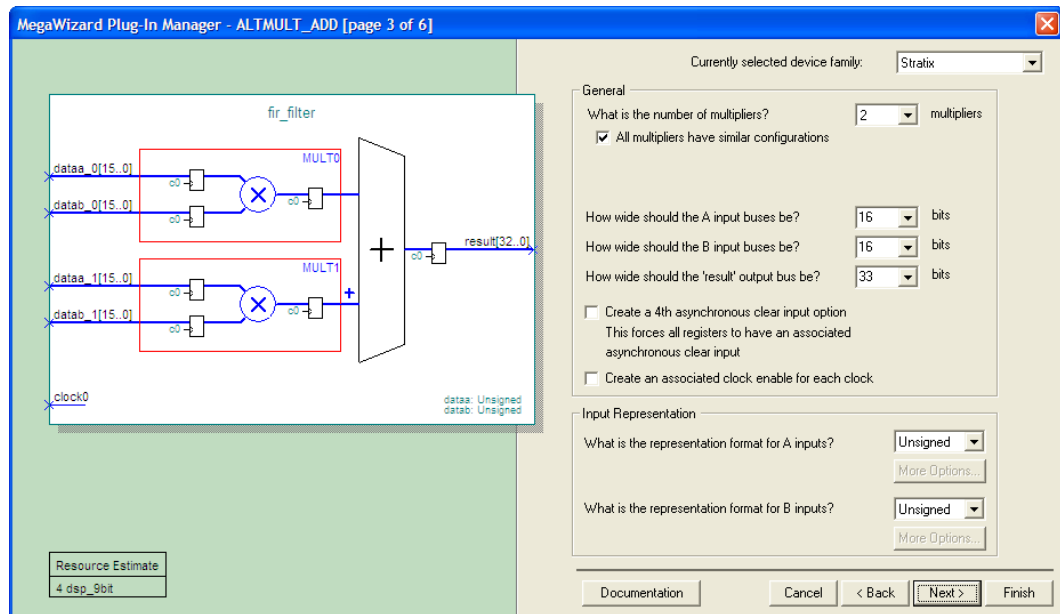
The MegaWizard Plug-In Manager provides you with three megafunctions to ease the integration and allow you to take full advantage of the DSP blocks in Stratix devices in your design. These megafunctions are `lpm_mult`, `altmult_accum`, and `altmult_add`.

The `lpm_mult` megafunction creates simple multipliers. The `altmult_accum` megafunction creates a single multiplier feeding an accumulator. The `altmult_add` megafunction creates one or more multipliers feeding a parallel adder.

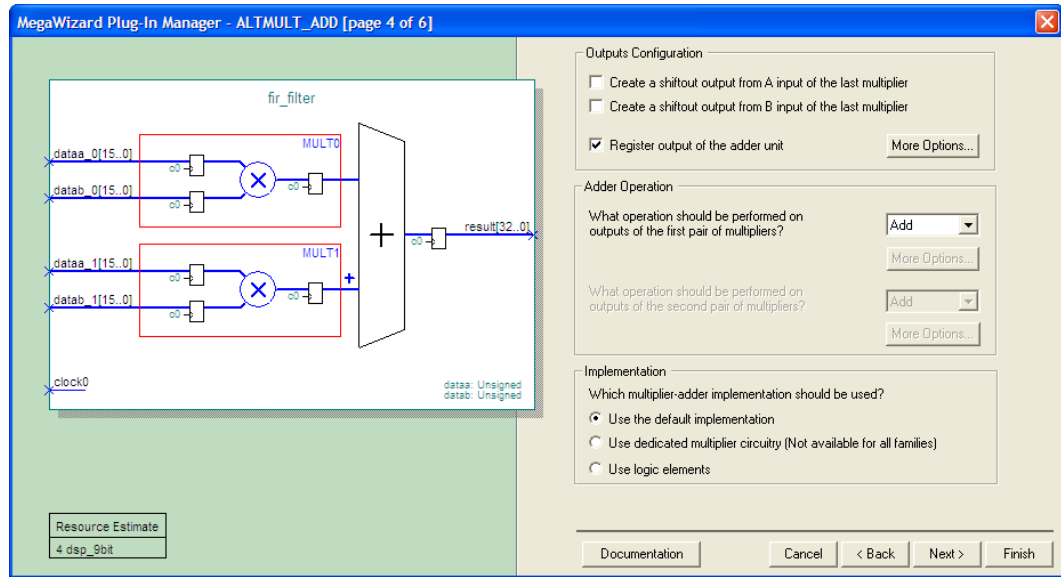
This section shows how to create an `altmult_add` instance using the MegaWizard Plug-In Manager.

The `altmult_add` megafunction configures the DSP block into multiply add mode. The third page of the `altmult_add` megafunction in the MegaWizard Plug-In Manager configures the bus width and sign representation of the operands (see [Figure 33](#)).

Figure 33. Page 3 of `altmult_add` MegaWizard Plug-In Manager

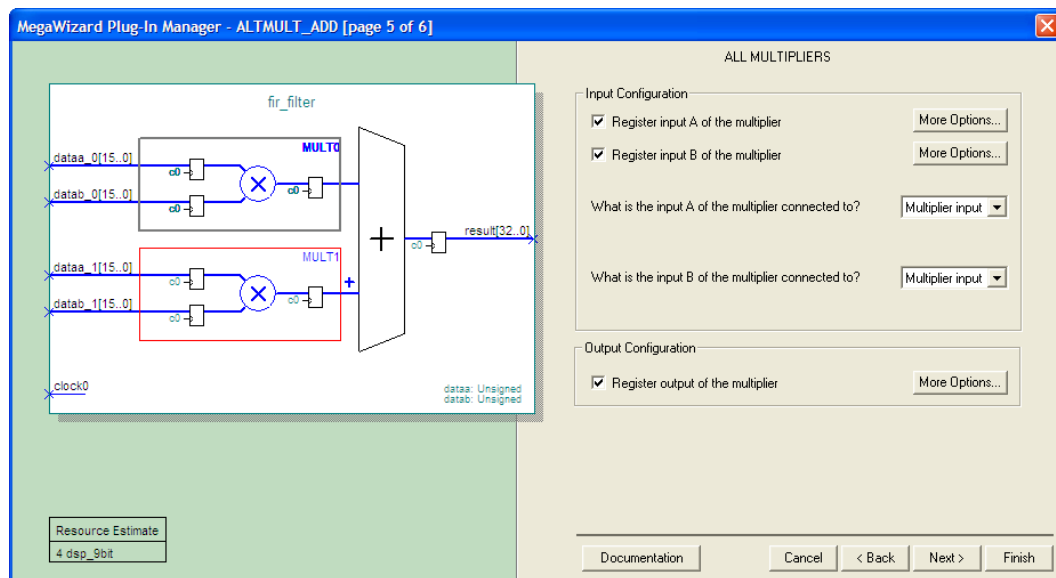


Page 4 of the `altmult_add` megafunction (Figure 34) creates optional ports, types of adder operation, and selects the type of device implementation to be used.

Figure 34. Page 4 of the `altmult_add` in the MegaWizard Plug-In Manager

Page 5 (Figure 35) allows you to further customize the `altmult_add` megafunction by providing the ability to register the input and output ports.

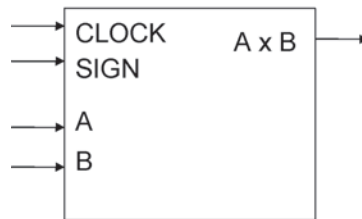
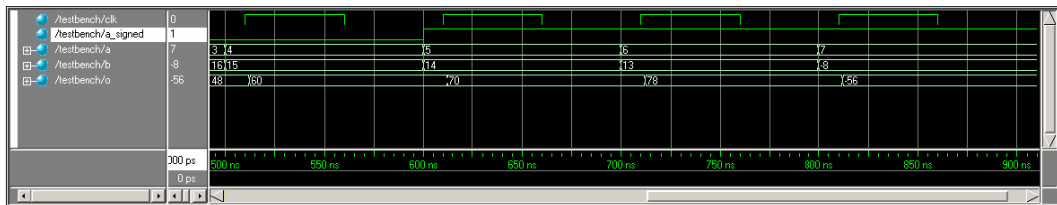
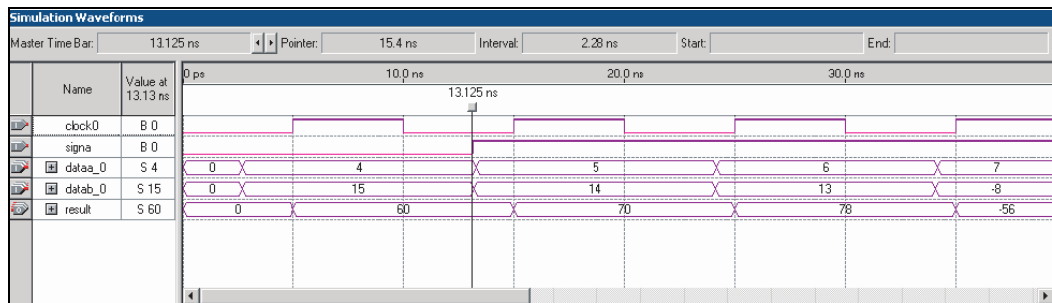
Figure 35. Page 5 of the `altmult_add` in the MegaWizard Plug-In Manager



Third-party synthesis tools can infer Stratix DSP blocks in either VHDL or Verilog HDL. For more information on using the Synplify and Precision RTL tools in conjunction with the Quartus II software, refer to the *Synplicity Synplify & Synplify Pro Support* and *Mentor Graphics Precision RTL Synthesis Support* chapters in volume 1 of the *Quartus II Handbook*.

Examples

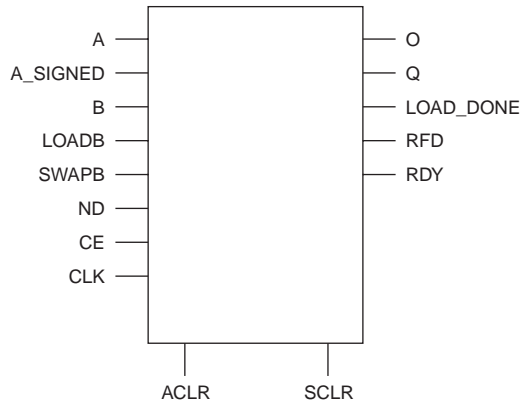
Figure 36 shows a generic multiplier block that has been created using the MegaWizard Plug-In Manager's `lpm_mult` and CoreGen's multiplier module. Figure 37 shows a functional simulation (using ModelSim®) of CoreGen's 16×16 multiplier implemented in a Virtex-II Embedded Multiplier. Figure 38 shows the converted 16×16 multiplier using the `lpm_mult` implemented in a Stratix DSP block simulated in the Quartus II Simulator.

Figure 36. Generic Multiplier Block**Figure 37. CoreGen 16 x 16 Embedded Multiplier with ModelSim XE****Figure 38. Simulation of a 16 x 16 Multiplier in a Stratix DSP Block Using Native Quartus II Simulation**

The conversion of CoreGen multipliers with the `altmult_add` is similar to that of `lpm_mult`.

You can also use the `altmult_add` megafunction, shown in [Figure 39](#), to replace the CoreGen multiplier function if your design does not meet the requirements for the `lpm_mult` megafunction. You can register inputs and outputs with the `altmult_add` megafunction, the sign of port A can be dynamic, and the sign of the input ports can be different.

Figure 39. `altmult_add` Megafunction



[Table 19](#) summarizes the port mapping between CoreGen's multiplier and `altmult_add` megafunction.

Table 19. CoreGen Multiplier and `altmult_add` Port Mapping Summary (Part 1 of 2)

| Xilinx Port | Altera Port | Comment |
|-------------|-------------|---------------------------------|
| A | dataa | Data Input Port |
| B | datab | Data Input Port |
| CE | ena0 | Clock Enable |
| CLK | clock0 | |
| ACLR | aclr3 | |
| Q | result | With registered outputs set |
| O | result | Without registered outputs set |
| A_SIGNED | signa | No Registering of port required |
| LOADB | N/A | |
| SWAPB | N/A | |
| SCLR | N/A | |
| LOAD_DONE | N/A | |

Table 19. CoreGen Multiplier and altmult_add Port Mapping Summary (Part 2 of 2)

| Xilinx Port | Altera Port | Comment |
|-------------|-------------|---------------------|
| RDY | N/A | Hand Shaking Signal |
| RFD | N/A | Hand Shaking Signal |
| ND | N/A | Hand Shaking Signal |

Double-Data Rate (DDR) I/O Conversion

This section contains information on the conversion of the dedicated DDR I/O function and architecture differences between Xilinx devices (Virtex-II, Virtex-II PRO and Spartan-3) and the Altera Stratix and Stratix GX product families.

Architectural Description

Stratix and Stratix GX devices have a dedicated DDR circuit in each of their I/O elements that streams serial data on both the rising and falling edges of the clock, effectively doubling the data rate. Xilinx Virtex-II, Virtex-II PRO and Spartan-3 also have a dedicated DDR function in their I/O blocks that handles timing somewhat differently.



For more information, refer to the *Altera Double Data Rate Megafunctions User Guide*.

Figure 40 shows a functional block diagram of the Stratix and Stratix GX input DDR path.

Figure 40. Stratix and Stratix GX Input DDR Path

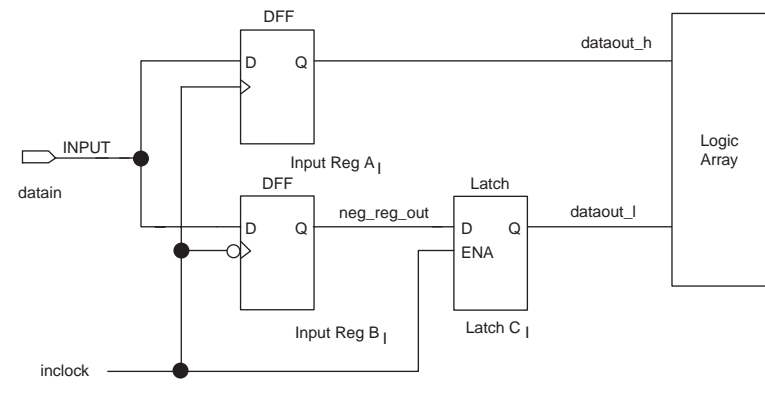


Figure 41 shows a functional block diagram of the Stratix and Stratix GX output DDR I/O path configuration.

Figure 41. Stratix and Stratix GX Output DDR I/O Path Configuration

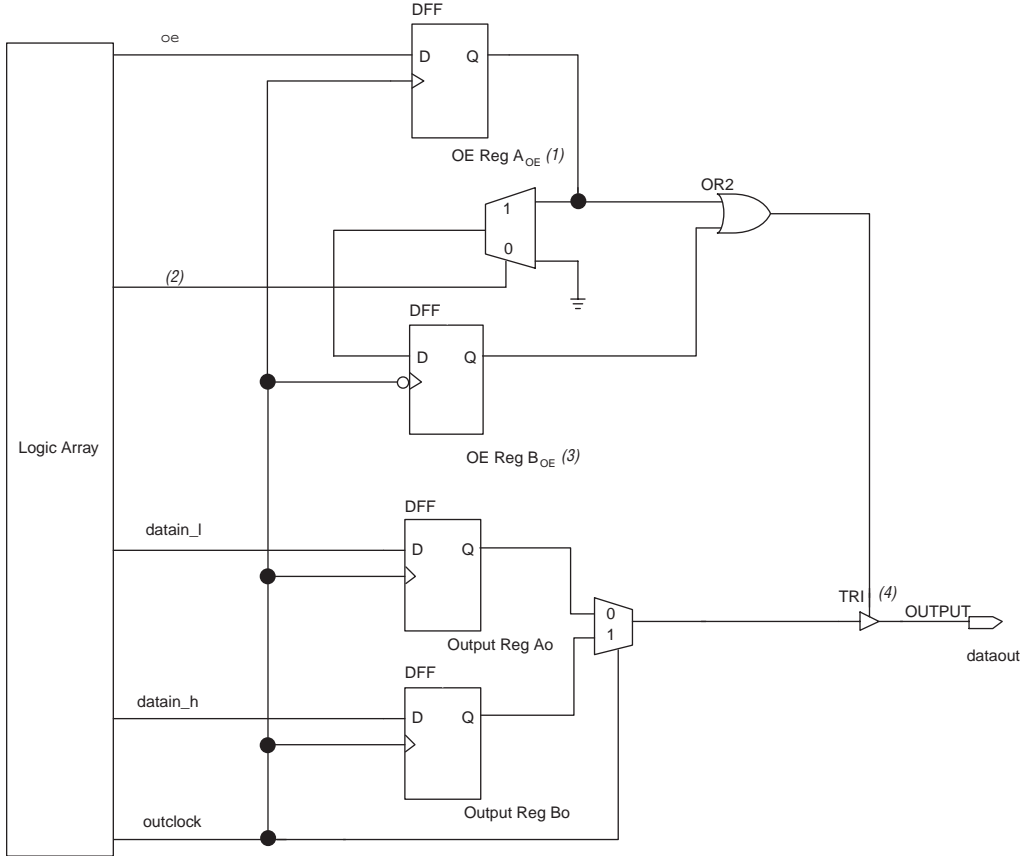
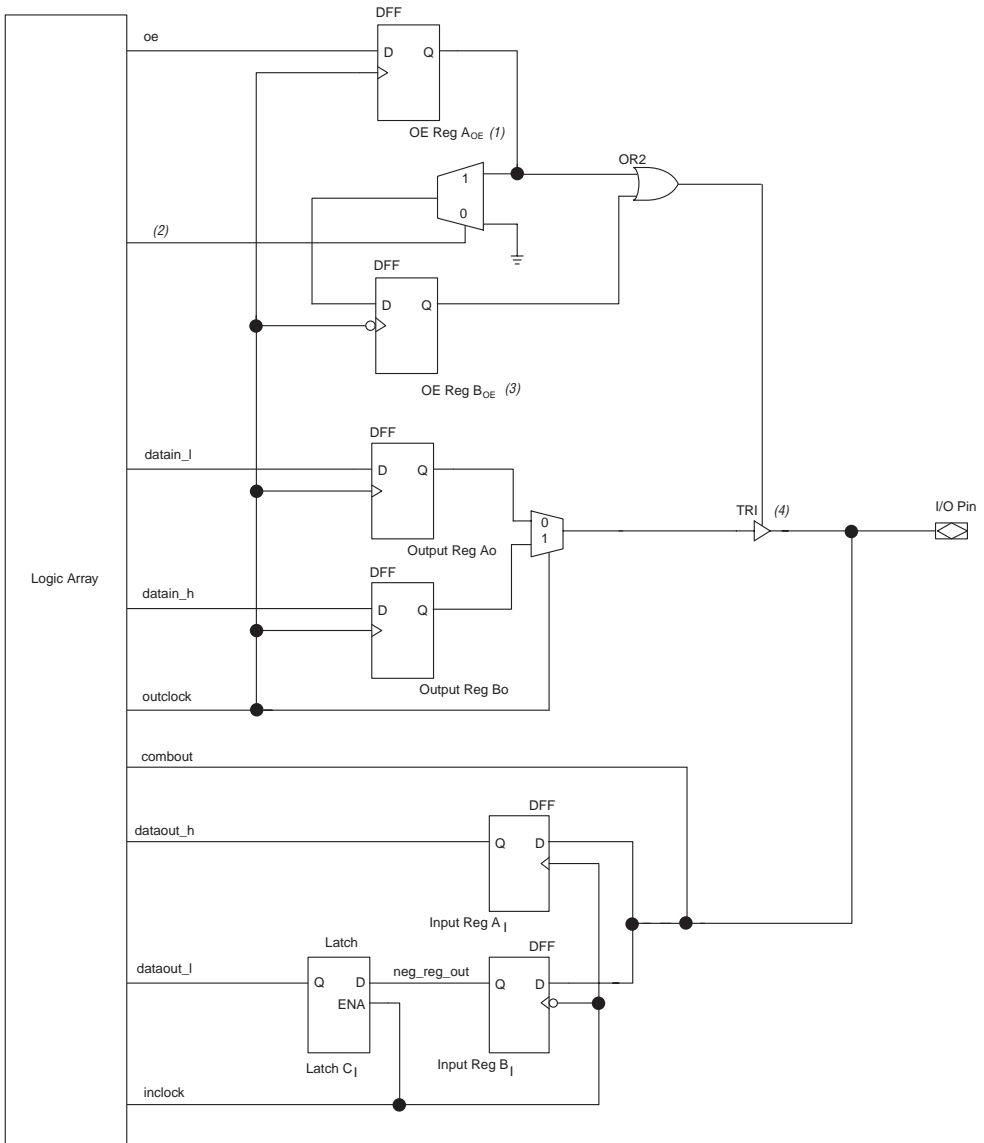


Figure 42 shows a functional block diagram of the Stratix and Stratix GX bidirectional DDR path.

Figure 42. Stratix and Stratix GX Bidirectional DDR Path



The Stratix and Stratix GX DDR input path has an extra latch to delay the low data by half a clock cycle. Consequently, high and low data appear in the LE at the same rising clock edge. The output path has a local clock inverter element on the low data register to optimize for performance and produce a 50% duty cycle.

The differences between Xilinx and Altera DDR I/O configurations are:

- The high and low data in the input path appear at the FPGA LE at the same rising clock edge in Stratix and Stratix GX devices. In Xilinx devices, the high data appears at the rising edge and the low data at the falling edge.
- The DCM is required to clock the DDR registers in Xilinx devices. Xilinx recommends that the 180° phase shift output is used as a second global clock network to feed the low data registers. Stratix and Stratix GX devices do not require the use of a second clock signal since the inversion is done locally.
- Quartus II DDR megafunctions do not have synchronous set/reset. Also, the asynchronous set and reset signals cannot be used at the same time.

Because of these differences, timing discrepancies will occur after device programming when migrating DDR functionality to a design targeting Altera devices. Designers must take this into consideration and adjust the interfacing logic accordingly.

Converting DDR I/O

Xilinx ISE has several DDR primitives for input or output DDR configurations:

- IFDDRCPE: Input DDR with asynchronous clear and preset and clock enable
- IFDDRSE: Input DDR with synchronous reset and set and clock enable
- OFDDRCPE: Output DDR with asynchronous clear and preset and clock enable
- OFDDRSE: Output DDR with synchronous reset and set and clock enable
- OFDDRTCPE: Output DDR with tristate, asynchronous clear and preset and clock enable
- OFDDRTRSE: Output DDR with tristate, synchronous reset and set and clock enable

The Quartus II software includes 3 DDR megafunctions, available from the MegaWizard Plug-In Manager:

- `altdio_in`: for DDR input
- `altdio_out`: for DDR output
- `altdio_bidir`: for DDR bidirectional input/output

DDR Input Conversion

Table 20 lists the mapping of IFDDRCPE and IFDDRRSE Xilinx primitives to the Altera `altdio_in` megafunction.

| <i>Table 20. Mapping of IFDDRCPE and IFDDRRSE Xilinx primitives to the Altera <code>altdio_in</code> Megafunction</i> | | |
|---|--------------------------|---|
| Xilinx Port | Altera Port | Comment |
| PRE / S | <code>aset</code> | <code>altdio_in</code> only supports asynchronous preset |
| D | <code>data_in[]</code> | Can combine multiple signals into a bus |
| CE | <code>inclocken</code> | |
| C0 | <code>inclock</code> | |
| C1 | Not Applicable | The negative clock edge signal is taken care of automatically by <code>altdio_in</code> |
| CLR / R | <code>aclr</code> | <code>altdio_in</code> only supports asynchronous clear |
| Q0 | <code>dataout_h[]</code> | Can combine multiple signals into a bus |
| Q1 | <code>dataout_l[]</code> | Can combine multiple signals into a bus |

Only the C0 port (data high clock signal) of the IFDDRCPE and IFDDRRSE primitives needs to be connected to the `inclock` port of the `altdio_in` megafunction. The C1 port (data low clock signal), which is normally connected to the output of the 180° output of the DCM or the inverted C0 signal, is not used during the mapping. The `altdio_in` megafunction automatically negotiates the relationship between the data high and the data low signals.

Since the appearance of the high and low data between Altera and Xilinx DDR functions are 180° apart, the FPGA logic interfacing with the incoming data may need to be adjusted accordingly.

DDR Output Conversion

Table 21 lists the mapping of IFDDRCPE and IFDDRRSE Xilinx primitives to Altera `altddio_out` megafunction.

| Xilinx Port | Altera Port | Comment |
|-------------|-------------------------|---|
| PRE / S | <code>aset</code> | <code>altddio_out</code> only supports asynchronous preset |
| D0 | <code>datain_h[]</code> | Can combine multiple signals into a bus |
| D1 | <code>datain_l[]</code> | Can combine multiple signals into a bus |
| CE | <code>outclocken</code> | |
| C0 | <code>outclock</code> | |
| C1 | Not Applicable | The negative clock edge signal is taken care of automatically by <code>altddio_out</code> |
| CLR / R | <code>aclr</code> | <code>altddio_out</code> only supports asynchronous clear |
| Q / O | <code>dataout []</code> | Can combine multiple signals into a bus |
| T | <code>oe</code> | |

Only the C0 port (the data high clock signal) of the IFDDRCPE and IFDDRRSE primitives must be connected to the `outclock` port of the `altddio_out` megafunction. The C1 port (data low clock signal), which is normally connected to the output of the 180° output of the DCM or the inverted C0 signal, is not used during the mapping. The `altddio_out` megafunction automatically negotiates of the relationship between the data high and the data low signals.

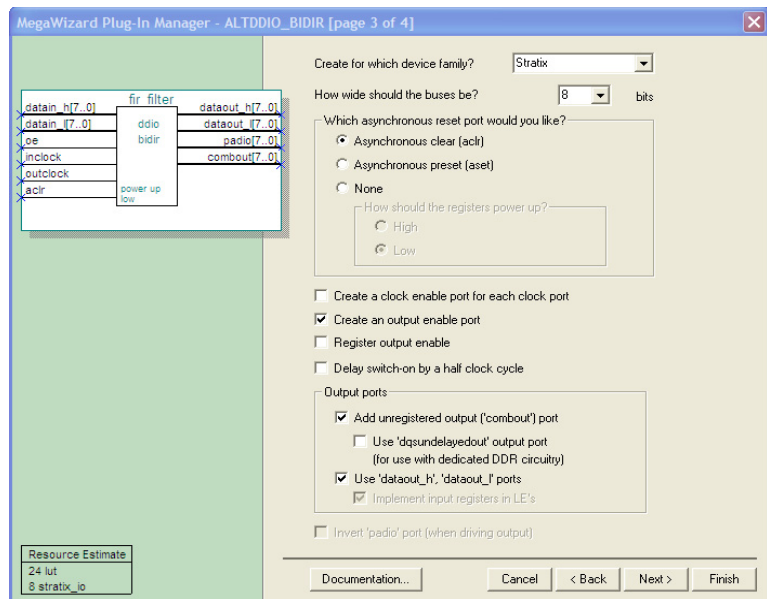
DDR Bidirectional Conversion

The Xilinx ISE software does not have a bidirectional DDR design element. A combination of input and output DDR primitives is used to achieve bidirectional DDR functionality. The combination of input and output DDR primitives located in the same I/O block must be replaced by the `altddio_bidir` primitive in a design targeting an Altera device. See “[DDR Input Conversion](#)” on page 77 and “[DDR Output Conversion](#)” on page 78 for port mapping information.

Implementing DDR I/O Using the Quartus II MegaWizard Plug-In Manager

The 3 DDR I/O megafunctions (`altd dio_in`, `altd dio_out`, and `altd dio_bidir`) included with the Quartus II software are all parameterizable (Figure 43). (The DDR I/O megafunctions are found in the I/O category.) Once a megafunction is selected, the user can parameterize the target architecture, the data bus width, the use of clear and preset signals, and other parameters.

Figure 43. `altd dio_bidir` Megafunction Parameters



Examples

An input DDR conversion in Verilog is shown below.

```

module IFDDRCPE (PRE, D, CE, C0, C1, CLR, Q0, Q1);
input PRE, D, CE, C0, C1, CLR;
output Q0, Q1;
endmodule

module DDR_top (PRE, D, CE, C0, C1, CLR, Q0, Q1);
input PRE, D, CE, C0, C1, CLR;
output Q0, Q1;
IFDDRCPE my_ddr (.PRE(PRE),
                 .D(D),

```

```

        .CE(CE),
        .C0(C0),
        .C1(C1),
        .CLR(CLR),
        .Q0(Q0),
        .Q1(Q1));
endmodule

```

After conversion, the code will look like the following:

```

module my_altdio_in (aset, data_in, inclocken, inclock, aclr,
    datain_h, datain_l);
// This is the module created by the MegaWizard
input aset, data_in, inclocken, inclock, aclr;
output datain_h, datain_l;
endmodule

module DDR_top (PRE, D, CE, C0, C1, CLR, Q0, Q1);
input PRE, D, CE, C0, C1, CLR;
output Q0, Q1;
my_altdio_in my_ddr (.aset(PRE),
    .data_in(D),
    .inclocken(CE),
    .inclock(C0), // C1 is not connected
    .aclr(CLR),
    .datain_h(Q0),
    .datain_l(Q1));
endmodule

```

Constraints

When designing for a Xilinx device, the User Constraint File (**.ucf**) contains the constraints and attributes for the design. This file is similar to the Quartus II Settings File (**.qsf**). The UCF file contains all of the design's constraints and attributes, including timing requirements and location assignments. Since ISE does not report unconstrained paths, you must provide constraints for two purposes: to constrain the net (or instance), and to report the constraint. The Quartus II Timing Analyzer analyzes and reports on all paths in a design, therefore, constraints provided merely to report a constraint are not required. As a result, many constraints in the Xilinx tool are not necessary after converting your design into Altera's Quartus II design environment.

Converting Constraints

Constraints specify what requirements are necessary for the design to function correctly. These constraints may include system performance, I/O timing requirements, or point-to-point timing requirements. The Quartus II Assignment Editor allows you to view, add, and create assignments to nodes and entities, such as location assignments, timing assignments, options for individual nodes only, options for individual entities, parameter, and simulation assignments.

Table 22 provides typical constraints and attributes found in Xilinx UCF and their Altera equivalents.

| Table 22. Typical Constraints and Attributes Found in Xilinx UCF and their Altera Equivalents. (Part 1 of 2) | | |
|---|--|---|
| Xilinx Constraint | Constraint Function | Altera Equivalent |
| DRIVE | This constraint controls the output pin current value | Current Strength Current Strength can be found under the Option field in the Assignment Editor |
| FAST | This constraint turns on Fast Slew Rate Control | Slow Slew Rate Slew Rate can be found under the Option field in the Assignment Editor |
| IOB | This constraint is used to specify whether or not a register should be placed within the IOB of the device | Fast Input Register or Fast Output Register Both constraints can be found under the Option field in the Assignment Editor |
| IOBDELAY | This constraint is used to specify a delay before an input pad feeds the IOB, or an external element, from the IOB. The input pad can either feed the local IOB flip-flop or an external element from the IOB. | Adjust Input Delay to Input Register This constraint can be used to adjust the delay of the input pin to the input register. This option can be turned to either ON or OFF. This constraint can be found under Option in the Assignment Editor |
| IOSTANDARD | This constraint is used to specify the I/O standard for an I/O pin. | I/O standards are specified in the Assignment Editor This constraint can be found under the option "I/O Standard" in the Assignment Editor, then selecting the appropriate I/O standard from the list. |
| KEEP | The KEEP constraint is used to prevent a net from either being absorbed by a block, or synthesized out. | You can insert an LCELL between the two nets in question. Inserting an LCELL between the two will prevent either net from being synthesized out. |

Table 22. Typical Constraints and Attributes Found in Xilinx UCF and their Altera Equivalents. (Part 2 of 2)

| Xilinx Constraint | Constraint Function | Altera Equivalent |
|--------------------------|--|---|
| MAXDELAY | This constraint is used to specify the maximum delay in a net. | Use the Maximum Delay assignment in the Assignment Editor to apply this constraint in the Quartus II software. This assignment will override any clock settings if the assignment is applied a path between two registers. However, an f_{MAX} constraint can be used. If the net is purely combinatorial, a t_{PD} assignment can be made. |
| MAXSKEW | This constraint is used to specify the maximum skew in a net. | Use Maximum Data Arrival Skew or Maximum Clock Arrival, depending on the net, in the Assignment Editor to apply this constraint in the Quartus II software. |
| NODELAY | This constraint is used to reduce setup time at the cost of positive hold time. | Specify a Setup time parameter option, t_{SU} , that is available in the Assignment Editor. Make a t_{SU} assignment using the Assignment Editor |
| OFFSET | This constraint specifies the correlation between a global clock and its associated data in, or data out, pin. This is used to specify setup and Clock to Out constraints on the data registers. | The Assignment Editor can be used to specify the t_{CO} constraint in the Quartus II software. Use the Assignment Editor to specify a t_{SU} constraint. |
| PERIOD | This constraint specifies the timing relationship of a global clock such as an f_{MAX} requirement. | f_{MAX} timing requirements can be specified in the Timing Settings dialog box. Make individual and global clock settings using the Timing Settings dialog box (Project menu). |

DCM and DLL Constraints

Attributes are required to be set when instantiating a either a DCM or DLL in a Xilinx design such as `CLKDV_DIVIDE` and `CLKFX_MULTIPLY`. However, when creating an Altera PLL, these parameters are all specified

using the MegaWizard Plug-In Manager. See “Implementing Altera PLLs Using the MegaWizard Plug-In Manager” on page 56 for more information.

Xilinx-based placement constraints do not carry over to Altera placement constraints. Do not make placement constraints to a design until the conversion process involving the Quartus II software is complete.

Xilinx-based placement constraints include the following:

- LOC
- RLOC
- RLOC_ORIGIN
- RLOC_RANGE
- MAP

Summary

Section one demonstrated that the Quartus II software provides a suite of tools similar to those found in the Xilinx ISE software. In addition, the cross-probing, SOPC Builder, SignalTap Logic Analysis, and Tcl scripting capabilities allow you to increase your productivity when designing for Altera devices. Section two provided guidelines to migrate a design targeted at a Xilinx device to one that is compatible with an Altera device. Section three showed you how to convert your ISE constraints into Quartus II constraints.

References

For more information, refer to the following documents:

- *Using General-Purpose PLLs in Stratix & Stratix GX Devices* chapter in volume 2 of the *Stratix Device Handbook*
- *TriMatrix Embedded Memory Blocks in Stratix & Stratix GX Devices* chapter in volume 2 of the *Stratix Device Handbook*
- *DSP Blocks in Stratix & Stratix GX Devices* chapter in volume 2 of the *Stratix Device Handbook*
- *Switching from Xilinx ISE to the Quartus II Software*—available on www.altera.com/switch
- *Performing Equivalent Timing Analysis Between the Altera Quartus II Software and Xilinx ISE*—white paper
- *Synplicity Synplify & Synplify Pro Support* chapter in volume 1 of the *Quartus II Handbook*
- *Mentor Graphics Precision RTL Synthesis Support* chapter in volume 1 of the *Quartus II Handbook*
- *Synopsys Design Compiler FPGA Support* chapter in volume 1 of the *Quartus II Handbook*
- *Design Recommendations for Altera Devices* chapter in volume 1 of the *Quartus II Handbook*

- *Recommended HDL Coding Styles* chapter in volume 1 of the *Quartus II Handbook*

Revision History

The information contained in the current version *AN 307: Altera Design Flow for Xilinx Users* supersedes information published in previous versions.

| Date & Version | Description of Changes |
|-------------------|--|
| June 2005, v5.0 | <ul style="list-style-type: none"> ● Revised content for software versions ISE 7.1i and Quartus II 5.0 ● Updated terminology ● Added Pin Planner subsection ● Added Quartus II Incremental Compilation |
| Feb 2004, v4.0 | <ul style="list-style-type: none"> ● Revised content for software versions ISE 6.3i and Quartus II 4.2 ● Updated Table 6 for Power ● Updated cross-probing chart |
| Jan 2004, v3.1 | <ul style="list-style-type: none"> ● Updated terminology |
| Oct 2004, v.3.0 | <ul style="list-style-type: none"> ● Revised content for software versions ISE 6.2i and Quartus II 4.1 sp2 ● Added information on cross-probing |
| July 2003, v. 2.0 | <ul style="list-style-type: none"> ● Revised content for software version ISE 5.1i and Quartus II 3.0 ● Added information on the Quartus II modular executables and command-line scripting ● Added information on DDR RAM conversions |
| v1.0 | New document |



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

Copyright © 2005 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



Printed on recycled paper



I.S. EN ISO 9001

