

This chapter discusses important FPGA design planning considerations, provides recommendations, and describes various tools available for Altera® FPGAs to improve design productivity.

Introduction



The inherent flexibility of advanced FPGAs means that the pin layout, power consumption, area utilization, and timing performance for each design block are all dependent on the final design implementation. Because of the significant increase in FPGA device densities over the last few years, designs are increasingly complex and might involve multiple designers. System architects must resolve these design issues when integrating design blocks, often leading to problems that affect the overall time to market and thereby increase cost. Many potential problems can be solved earlier in the design cycle by performing good design planning.

This chapter contains the following sections:

- “Creating Design Specifications” on page 1–2
- “Intellectual Property Selection” on page 1–2
- “Device Selection” on page 1–3
- “Planning for Device Programming or Configuration” on page 1–4
- “Early Power Estimation” on page 1–5
- “Early Pin Planning and I/O Analysis” on page 1–6
- “Selecting Third-Party EDA Tool Flows” on page 1–9
- “Planning for On-Chip Debugging Options” on page 1–10
- “Design Practices and HDL Coding Styles” on page 1–12
- “Planning for Hierarchical and Team-Based Design” on page 1–14
- “Fast Synthesis and Early Timing Estimation” on page 1–18

Before reading the design planning guidelines discussed in this chapter, consider your design priorities. You should know what are the most important factors for your design. More device features, density, or performance can increase system cost. Signal integrity and board issues might impact I/O pin locations. Power, timing performance, and area utilization affect each other, and compilation time is affected by optimizations for these factors.

The Quartus® II software optimizes designs for the best average results, but you can change the settings to focus on one aspect of the design results and trade off other aspects. Certain tools or debugging options can lead to restrictions in your design flow. If you know what is important in a particular design, this knowledge helps you choose the tools, features, and methodologies that you should use with the design. This chapter cannot cover every possible consideration for planning a complex FPGA design, but once you understand your design priorities, you can use the design planning considerations described here as a guide to help ensure a productive and successful FPGA design flow.

-  This chapter provides an introduction to various design and planning features in the Quartus II software. For a general overview of the Quartus II design flow and features, refer to the *Introduction to the Quartus II Software* manual. For more details about specific Quartus II features and methodologies, this chapter provides references to other appropriate chapters in the *Quartus II Handbook*.
-  After you have selected a device family, to check if additional guidelines are available, refer to the Design Guideline section of the device on the [Altera Literature and Technical Documentation](#) page.

Creating Design Specifications

Before you create your logic design or complete your system design, create detailed design specifications that define the system, specify the I/O interfaces for the FPGA, identify the different clock domains, and include a block diagram of basic design functions. Taking the time to create these specifications helps improve design efficiency.


Creating a test plan at this phase also helps you to design for testability and manufacturability. For example, if you want to perform any built-in self-test functions to drive interfaces, you can use a UART interface with a Nios® II processor inside the FPGA device. You might require the ability to validate all the design interfaces. For guidelines related to analyzing and debugging the device after it is in the system, refer to “[Planning for On-Chip Debugging Options](#)” on page 1–10.

If your design includes multiple designers, it is also useful to consider a common design directory structure at this point. This eases the design integration stages. For more suggestions on team-based designs, refer to “[Planning for Hierarchical and Team-Based Design](#)” on page 1–14.

Intellectual Property Selection

Altera and its third-party intellectual property (IP) partners offer a large selection of off-the-shelf IP cores optimized for Altera devices. The IP selection often affects system design, especially if the FPGA interfaces with other devices in the system. Consider which I/O interfaces or other blocks in your system design are implemented using IP cores, and plan to incorporate these cores in your FPGA design.


The OpenCore Plus feature, which is available for many IP cores, allows you to program the FPGA to verify your design in the hardware before you purchase the IP license. The evaluation supports an untethered mode, in which the design runs for a limited time, or a tethered mode. The tethered mode requires an Altera serial JTAG cable connected between the JTAG port on your board and a host computer running the Quartus II Programmer for the duration of the hardware evaluation period.

 For descriptions of available IP cores, refer to the [Intellectual Property](#) page on the Altera website.

Device Selection


The first stage in design planning is choosing the best device for your application. The device selection affects the rest of your design cycle, including board specification and layout. Most of this planning is performed outside of the Quartus II software, but this section provides a few suggestions to aid in the planning process.

Choose the device family that best suits your design requirements. Different families offer different trade-offs, including cost, performance, logic and memory density, I/O density, power utilization, and packaging. You should also consider feature requirements, such as I/O standards support, high-speed transceivers, global or regional clock networks, and the number of phase-locked loops (PLLs) available in the device.

 You can review important features of each device family in the [Selector Guides](#) page. Each device family also has a device handbook or set of data sheets that documents the device features in detail.

Determining the required device density can be a challenging part of the design planning process. Devices with more logic resources and higher I/O counts can implement larger and potentially more complex designs, but may have a higher cost. Smaller devices have lower static power utilization. Select a device that meets your design requirements with some safety margin, in case you want to add more logic later in the design cycle to upgrade or expand your design, or reserve logic and memory for on-chip debugging (refer to [“Planning for On-Chip Debugging Options” on page 1–10](#)). Consider requirements for specific types of dedicated logic blocks, such as memory blocks of different sizes, or digital signal processing (DSP) blocks to implement certain arithmetic functions.

Many next-generation designs use a current design as a starting point. If you have other designs that target an Altera device, you can use their resource utilization as an estimate for your new design. Compile existing designs in the Quartus II software with the **Auto device selected by the Fitter** option in the **Settings** dialog box. Review the resource utilization to find out which device density fits the design. Consider that coding style, device architecture, and the optimization options used in the Quartus II software can significantly affect the resource utilization and timing performance of your design.

 To obtain resource utilization estimates for certain configurations of Altera’s IP designs, refer to the user guides for Altera megafunctions and IP MegaCores on [Literature: IP and Megafunctions](#) section of the Altera website.

Device Migration Planning

Determine whether you want the option of migrating your design to another device density to allow flexibility when your design nears completion, or whether you want to migrate to a HardCopy® ASIC when your design reaches volume production. In some cases, designers may target a smaller (and less expensive) device and then move to a larger device if necessary to meet their design requirements. Other designers may prototype their design in a larger device to reduce optimization time and achieve timing closure more quickly, and then migrate to a smaller device after prototyping. Similarly, many designers compile and optimize their design for an FPGA device and then migrate to a HardCopy ASIC when the design is complete and ready for higher-volume production. If you want the flexibility to migrate your design, you should specify these migration options in the Quartus II software at the beginning of your design cycle.

To specify the target migration devices, perform the following:

1. In the Assignments menu, select **Settings**. The **Settings** dialog box appears.
2. On the **Device** page, click on the **Migration Devices** button. The **Migration Devices** dialog box appears.
3. In the **Migration Devices** dialog box, select your target device in the **Compatible migration devices** section.

Selecting a migration device has an impact on pin placement because some pins may serve different functions in different device densities or package sizes. If you are making pin assignments in the Quartus II software, the Pin Migration View in the Pin Planner highlights pins that change function between your migration devices. (For more information, refer to “[Early Pin Planning and I/O Analysis](#)” on page 1–6.) Selecting a companion device might restrict logic utilization to ensure that your design is compatible with a selected HardCopy device. Adding migration or companion devices later in the design cycle is possible, but requires extra effort to check pin assignments, and might require design changes to fit into the new target device. Altera recommends that you consider these issues early in the design cycle than at the end, when the design is near completion and ready for migration.

In addition, if you are using a HardCopy ASIC, review HardCopy guidelines early in the design cycle for any Quartus II settings that should be used or other restrictions you should consider. You must use complete timing constraints if you want to migrate to a HardCopy device because of the rigorous verification requirements for ASICs.



For more information about timing requirements and analysis for HardCopy designs, refer to the *HardCopy Series Handbook*, and the *Quartus II Support for HardCopy Series Devices* chapter in volume 1 of the *Quartus II Handbook*.

Planning for Device Programming or Configuration

Another important part of the device planning is determining how you want to program or configure the device in your system. Choosing your programming or configuration method early allows system and board designers to determine what companion devices, if any, are required for your system. Your board layout also depends on the type of programming or configuration method you plan to use for programmable devices. Many programming options require a JTAG interface to connect to the devices, so you might have to set up a JTAG chain on the board. In

In addition, the Quartus II software uses the settings for the configuration scheme, configuration device, and configuration device voltage to enable the appropriate dual purpose pins as regular I/O pins after configuration is complete. The Quartus II software performs voltage compatibility checks of those pins during I/O assignment analysis and compilation of your design. Click the **Configuration** tab of the **Device and Pin Options** dialog box and select your configuration scheme.



The device family handbooks describe the configuration options available for a given device family. For more details about configuration options, refer to the *Configuration Handbook*. For information about programming CPLD devices, refer to your device data sheet or handbook.

Early Power Estimation

You can use the Quartus II power estimation and analysis tools to provide information to PCB board and system designers. You can perform early power estimation before you create any source code, or when you have a preliminary version of the design source code, and then perform the most accurate analysis with the PowerPlay Power Analyzer when the design is complete.

You must accurately estimate device power consumption to develop an appropriate power budget and to design the power supplies, voltage regulators, heat sink, and cooling system. Power estimation and analysis helps you satisfy two important planning requirements:

- **Thermal planning**—You must ensure that the cooling solution is sufficient to dissipate the heat generated by the device. The computed junction temperature must fall within normal device specifications.
- **Power supply planning**—You must ensure that the power supplies provide adequate current to support device operation.


Power consumption in FPGA devices is dependent on the logic design. This dependence can make power estimation challenging during the early board specification and layout stages. Altera's PowerPlay Early Power Estimator (EPE) spreadsheet allows you to estimate power utilization before the design is complete. To use the EPE, you must provide information about the device resources that are used in the design, as well as the operating frequency, toggle rates, and environmental considerations.

If you have an existing design or a partially-completed design, the Quartus II software power estimator file can provide input to the EPE spreadsheet to specify information about your current design (refer to "[Creating Powerplay EPE Spreadsheets](#)").

The PowerPlay EPE spreadsheets for each supported device family are available on the [PowerPlay Early Power Estimator and Power Analyzer](#) page.

Estimating power consumption early in the design cycle allows planning of power budgets and avoids unexpected results for designers developing the PCB.

When the design is complete, perform a complete power analysis to check the power consumption more accurately. The PowerPlay Power Analyzer tool in the Quartus II software provides an accurate estimation of power, ensuring that thermal and supply budgets are not violated. For the most accurate power estimation, use gate-level simulation results from a Verilog Value Change Dump File (.vcd) with the PowerPlay Power Analyzer.

 For more information about power estimation and analysis, refer to the *PowerPlay Power Analysis* chapter in volume 3 of the *Quartus II Handbook*.

Creating Powerplay EPE Spreadsheets

You can use PowerPlay EPE spreadsheets to perform a preliminary thermal analysis and power consumption estimate for your design. You can enter the data manually, or you can use the tools in the Quartus II software to assist you in generating the device resources usage information for your design.

If you manually enter data into the EPE spreadsheet, you can enter the device resources, operating frequency, toggle rates, and other parameters for your design. If you do not have an existing design, you can estimate the number of device resources used in your design and enter them manually.

If you have an existing design or a partially completed design, you can use the Quartus II software to generate the PowerPlay EPE file to assist you in completing the PowerPlay EPE spreadsheet.

To generate the power estimation file, you must first compile your design in the Quartus II software. After compilation is complete, on the Project menu, click **Generate PowerPlay Early Power Estimator File**. The PowerPlay EPE file is a Comma-Separated Value File (.csv) named <project>_early_power.csv. If your design targets a Cyclone, Stratix, or Stratix GX device, the PowerPlay EPE file is in a Tab-Separated Value File (.txt) named <project>_early_power.txt.

The PowerPlay EPE spreadsheet includes the Import Data macro that parses the information in the power estimation file and transfers it into the spreadsheet. If you do not want to use the macro, you can manually transfer the data into the EPE spreadsheet. For example, after importing the PowerPlay EPE file information into the PowerPlay EPE spreadsheet, you can add additional devices resource information at any time. If the existing Quartus II project represents only a portion of your full design, you can manually enter the additional device resources used in the final design.

Early Pin Planning and I/O Analysis

In many design environments, FPGA designers want to plan top-level FPGA I/O pins early to help board designers to start developing the PCB design and layout. The FPGA device's I/O capabilities and board layout guidelines influence pin locations and other types of assignments. If the board design team specifies an FPGA pin-out, it is crucial that the pin locations are verified in the FPGA placement and routing software to avoid board design changes.

In the past, designers and system architects could not check the validity of FPGA pin assignments until the design was completed. You can now create a preliminary pin-out for an Altera FPGA with the Quartus II Pin Planner before the source code is developed, based on standard I/O interfaces (such as memory and bus interfaces) and any other I/O-related assignments defined by system requirements. For more information, refer to “[Creating a Top-Level Design File for I/O Analysis](#)” on page 1-8. The Quartus II I/O Assignment Analysis checks that the pin locations and assignments are supported in the target FPGA architecture. You can use **I/O Assignment Analysis** to validate I/O-related assignments that you create or modify throughout the design process. When you compile your design in the Quartus II software, the I/O Assignment Analysis in the Fitter validates that the assignments meet all the device requirements and generates messages if there are any problems.

The Pin Planner enables easy I/O pin assignment planning, assignment, and validation. You can use the View menu in the Pin Planner to create pin location and other assignments using a device package view instead of pin numbers.

With the Pin Planner, you can identify I/O banks, voltage reference (VREF) groups, and differential pin pairings to help you through the I/O planning process. If migration devices are selected (including HardCopy devices) as described in “[Device Migration Planning](#)” on page 1-4, the **Pin Migration View** highlights pins that have changed functions in the migration device when compared to the currently selected device. Selecting pins in the Device Migration view cross-probes to the rest of the Pin Planner, so you can use device migration information when planning your pin assignments. You can also configure board trace models of selected pins for use in “board-aware” signal integrity reports generated with the **Enable Advanced I/O Timing** option. This option ensures you get very accurate I/O timing analysis. You have the option to use a Microsoft Excel spreadsheet to start the I/O planning process if you normally use a spreadsheet in your design flow, and you can export a **.csv** containing your I/O assignments for spreadsheet use when all pins are assigned.

When planning is complete, the pin location information can be passed to PCB designers. The Pin Planner is tightly integrated with certain PCB design EDA tools, and can read pin location changes from these tools to check the suggested changes. Your pin assignments must match between the Quartus II software and your schematic and board layout tools to ensure the design works correctly on the board on which it is placed, especially if changes to the pin-out must be made. The system architect can use the Quartus II software to pass pin information to team members designing individual logic blocks, for better timing closure when they compile their design. When the design is complete, the Quartus II Fitter reports are used for the final sign-off of pin assignments. After compilation, the Quartus II software generates the Pin-Out File (**.pin**). You can use this file to verify that each pin is correctly connected in board schematics.

Starting FPGA pin planning early—before the HDL design is complete—improves the confidence in early board layouts, reduces the chance of error, and improves the design’s overall time to market.



For more information about I/O assignment and analysis, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*. For more information about passing I/O information between the Quartus II software and third-party EDA tools, refer to the *Mentor Graphics PCB Design Tools Support* and *Cadence PCB Design Tools Support* chapters in the *I/O and PCB Tools* section in volume 2 of the *Quartus II Handbook*.

Creating a Top-Level Design File for I/O Analysis

Early in the design process, before the source code is created, the system architect has information about the standard I/O interfaces (such as memory and bus interfaces), the IP cores that are used in the design, and any other I/O-related assignments defined by system requirements. You can use this information with the **Create/Import Megafunction** feature in the Pin Planner to specify details about the design I/O interfaces. Specifying these details allows you to create a top-level design file that includes all your I/O information, so you can analyze the I/O assignments in the Quartus II software.

The Pin Planner interfaces with the MegaWizard™ Plug-In Manager, and allows you to create or import custom megafunctions and IP cores that use I/O interfaces. You can configure how they are connected to each other by specifying matching node names for selected ports in the **Set Up Top-Level Design File** dialog box. Create any other I/O-related assignments for these interfaces or other design I/O pins in the Pin Planner.

When you have entered as much I/O-related information as possible, generate a top-level design file using the **Create Top-Level Design File** command. The Pin Planner creates virtual pin assignments for internal nodes, so internal nodes are not assigned to device pins during compilation. After analysis and synthesis of the newly generated top-level wrapper file, use the generated netlist to perform I/O Analysis with the **Start I/O Assignment Analysis** command.

You can use the I/O analysis results to change pin assignments or IP parameters, and repeat the checking process until the I/O interface meets your design requirements and passes the pin checks in the Quartus II software. When this initial pin planning is complete, you can create a Quartus II Revision based on the Quartus II-generated netlist. You then have a choice on how to proceed: you can use the generated netlist to develop the top-level file for the actual design, or disregard the generated netlist and use the generated Quartus II Settings File (.qsf) with the actual design.

Simultaneous Switching Noise Analysis

Simultaneous switching noise (SSN) is defined as a noise voltage inducted onto a victim I/O pin of a device due to the switching behavior of other aggressor I/O pins in the device. SSN noise often leads to the degradation of signal integrity by causing signal distortion, thereby reducing the noise margin of a system. It is best approach to address SSN with estimation early in your system design, to reduce the chance of any later board design changes. When the design is complete, tape out your PCB with complete SSN analysis of your FPGA in the Quartus II software.

Altera provides tools for SSN analysis and estimation, including SSN characterization reports, an Early SSN Estimator (ESE) tool, and the SSN Analyzer in the Quartus II software.

The ESE tool is a good starting point to estimate SSN in your FPGA design, and it is available for various device families.



For more information and device support for the ESE spreadsheet tool, refer to [Altera's Signal Integrity Center](#) on the Altera website. For more information about the SSN Analyzer, refer to the *Simultaneous Switching Noise (SSN) Analysis and Optimizations* chapter in volume 2 of the *Quartus II Handbook*.

Selecting Third-Party EDA Tool Flows

Your complete FPGA design flow may include third-party EDA tools in addition to the Quartus II software. Determine which tools you want to use with the Quartus II software to ensure that they are supported and set up correctly, and that you are aware of any useful features or undesired limitations.

Synthesis Tools

The Quartus II software includes advanced and easy-to-use integrated synthesis that supports Verilog HDL and VHDL, as well as the Altera hardware description language (AHDL) and schematic design entry. You can also use supported standard third-party EDA synthesis tools to synthesize your Verilog HDL or VHDL design, and then compile the resulting output netlist file in the Quartus II software. Different synthesis tools may give different results for each design. To assess the best-performing tool for your application, you can experiment by synthesizing typical designs for your specific application and coding style. Perform placement and routing in the Quartus II software to get accurate timing analysis and logic utilization results.

Because tool vendors frequently add new features, fix tool issues, and enhance performance for Altera devices, Altera recommends using the most recent version of third-party synthesis tools. The *Quartus II Software Release Notes* lists the version of each synthesis tool that is officially supported by that version of the Quartus II software.

To use the correct Library Mapping File (.lmf) for your synthesis netlist, specify your synthesis tool in the New Project Wizard or the **EDA Tools Settings** page of the **Settings** dialog box.

Your synthesis tool may offer the capability to create a Quartus II project and pass constraints, such as the EDA tool setting, device selection, and timing requirements that you specified in your synthesis project. You can use this capability to save time when setting up your Quartus II project for placement and routing.

If you want to take advantage of an incremental compilation methodology, you should partition your design for synthesis and generate multiple output netlist files. For more information, refer to “*Incremental Compilation with Design Partitions*” on page 1-15.




For more information about synthesis tool flows, refer to the appropriate chapter in the *Synthesis* section in volume 1 of the *Quartus II Handbook*.

Simulation Tools

Altera provides the ModelSim Starter Edition with the Quartus II software. You can also purchase the ModelSim-Altera Edition to support large designs and achieve faster simulation performance. The Quartus II software can generate both functional and timing netlist files for ModelSim and other third-party simulators.


Use the simulator version that is supported with your Quartus II version for best results. You should also use the model libraries provided with your Quartus II software version. Libraries can change between versions, which might cause a mismatch with your simulation netlist. The *Quartus II Software Release Notes* list the version of each simulation tool that is officially supported with that particular version of the Quartus II software.

Specify your simulation tool in the **EDA Tools Settings** page of the **Settings** dialog box to generate the appropriate output simulation netlist.

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

Formal Verification Tools

The Quartus II software supports some formal verification flows. Consider whether your desired formal verification flow impacts the design and compilation stages of your design.

 For more information about formal verification flows and supported tools, refer to the appropriate chapter in the *Formal Verification* section in volume 3 of the *Quartus II Handbook*.

Using a formal verification flow can impact performance results because it requires that certain logic optimizations be turned off, such as register retiming, and forces hierarchy blocks to be preserved, which can restrict optimization. Formal verification treats memory blocks as black boxes. Therefore, it is best to keep memory in a separate hierarchy block so other logic does not get incorporated into the black box for verification. There are other restrictions that may also limit your design, so consult the documentation for details. If formal verification is important to your design, it is easier to plan for limitations and restrictions in the beginning than to make changes later in the design flow.

Specify your formal verification tool in the **EDA Tools Settings** page of the **Settings** dialog box to generate the appropriate output netlist.

Planning for On-Chip Debugging Options

Altera's in-system debugging tools offer different advantages and trade-offs, so a particular debugging tool may work better for different systems and designers. It is beneficial to evaluate on-chip debugging options early in your design process, to ensure that your system board, Quartus II project, and design are all set up to support the appropriate options. Planning can reduce time spent during debugging and eliminates having to make changes later to accommodate your preferred debugging methodologies.

The Quartus II portfolio of verification tools includes the following in-system debugging features:

- **SignalProbe incremental routing**—Quickly routing internal signals to I/O pins without affecting the design. Starting with a fully routed design, you can select and route signals for debugging to either previously reserved or currently unused I/O pins.
- **SignalTap® II Embedded Logic Analyzer**—Probes the state of the internal signals in the design without the use of external equipment or extra I/O pins, while the design is running at full speed in an FPGA device. Defining custom trigger-condition logic provides greater accuracy and improves the ability to isolate problems. The SignalTap II Embedded Logic Analyzer does not require external probes or changes to the design files to capture the state of the internal nodes or

I/O pins in the design; all captured signal data is conveniently stored in device memory until you are ready to read and analyze the data. The SignalTap II Embedded Logic Analyzer works best for synchronous interfaces. For debugging asynchronous interfaces, consider using SignalProbe or an external logic analyzer to view the signals most accurately.

- **Logic Analyzer Interface (LAI)**—Enables you to connect and transmit internal FPGA signals to an external logic analyzer for analysis. You can use this feature to connect a large set of internal device signals to a small number of output pins for debugging purposes, and allows you to take advantage of advanced features in your external logic analyzer or mixed signal oscilloscope.
- **In-System Memory Content Editor**—Provides read and write access to in-system FPGA memories and constants through the JTAG interface, making it easy to test changes to memory contents and constant values in the FPGA while the device is functioning in a system.
- **In-System Sources and Probes**—Sets up customized register chains to drive or sample the instrumented nodes in your logic design, providing an easy way to input simple virtual stimuli and capture the current value of instrumented nodes. You can force trigger conditions set up using the SignalTap II Logic Analyzer, create simple test vectors to exercise your design without the use of external test equipment, and dynamically control run-time control signals with the JTAG chain.
- **Virtual JTAG Megafunction**—Enables you to build your own system-level debugging infrastructure, including both processor-based debugging solutions and debugging tools in software for system-level debugging. The SLD_VIRTUAL_JTAG megafunction can be instantiated directly in your HDL code to provide one or more transparent communication channels to access parts of your FPGA design using the JTAG interface of the device.



For more information about debugging tools, refer to the appropriate “Referenced Documents” on page 1–19. For an overview of debugging options that can help you decide which option to use, refer to the Introduction section in *Section V. In-System Design Debugging* in volume 3 of the *Quartus II Handbook*.

If you intend to use any of these features, you may have to plan for the features when developing your system board, Quartus II project, and design. The following paragraphs describe various factors to consider during your design planning stages.

The SignalTap II Embedded Logic Analyzer, Logic Analyzer Interface, In-System Memory Content Editor, In-System Sources and Probes, and Virtual JTAG megafunction require JTAG connections to perform in-system debugging. Plan your system and board with JTAG ports that are available for debugging.

The JTAG debugging features also require a small amount of additional logic resources to implement the JTAG hub logic. If you set up the appropriate feature early in your design cycle, you can include these device resources in your early resource estimations to ensure you do not overfill the device with logic.

The SignalTap II Embedded Logic Analyzer uses device memory to capture data during system operation. To ensure that you have enough memory resources to take advantage of this debugging technique, consider reserving device memory to be used during debugging.

To use incremental debugging with the SignalTap II Embedded Logic Analyzer, the **Full incremental compilation** option must be turned on. This option is on by default for projects created in the Quartus II software version 6.1 or later, but is not turned on automatically for existing projects. If incremental compilation is not enabled, you must recompile the entire design when you want to add debugging functions, or when you make certain changes to SignalTap II settings. Using incremental compilation with the SignalTap II Embedded Logic Analyzer greatly reduces the compilation time required for debugging.

The SignalProbe and the Logic Analyzer Interface require I/O pins for debugging. Consider reserving I/O pins for debugging so that you do not have to change the design or board to accommodate debugging signals later. Keep in mind that the Logic Analyzer Interface can multiplex signals with design I/O pins if required. Ensure that your board supports some kind of debugging mode, where debugging signals do not affect system operation.

If you want to use the Virtual JTAG megafunction for custom debugging applications, you must instantiate and incorporate it as part of the design process.

The In-System Sources and Probes feature requires that you instantiate a megafunction in your HDL code. In addition, you have the option to instantiate the SignalTap II Embedded Logic Analyzer as a megafunction, so you can manually connect it to nodes in your design and ensure that the tapped node names do not change during synthesis. You can add the debugging block as a separate design partition for incremental compilation to minimize recompilation times.

To use the In-System Memory Content Editor for RAM or ROM blocks or the LPM_CONSTANT megafunction, turn on the **Allow In-System Memory Content Editor** to capture and independently update content of the system clock option when you create the memory block in the MegaWizard Plug-In Manager.

Design Practices and HDL Coding Styles

In the development of complex FPGA designs, design practices and coding styles have an enormous impact on your device's timing performance, logic utilization, and system reliability. Follow Altera's recommendations to achieve the best synthesis and fitting results.

Design Recommendations

You can use synchronous design practices to consistently meet your design goals. Problems with other design techniques include reliance on propagation delays in a device, incomplete timing analysis, and possible glitches. In a synchronous design, a clock signal triggers all events. As long as all the registers' timing requirements are met, a synchronous design behaves in a predictable and reliable manner for all process, voltage, and temperature (PVT) conditions. You can easily target synchronous designs to different device families or speed grades.

Pay particular attention to clock signals, because they have a large effect on your design's timing accuracy, performance, and reliability. Problems with clock signals can cause functional and timing problems in your design. You can use dedicated clock pins and clock routing for best results, and if PLLs are available in your target device, use the PLLs for clock inversion, multiplication, and division. For clock multiplexing and gating, use the dedicated clock control block or PLL clock switchover feature instead of combinational logic if these features are available in your device. If you must use internally-generated clock signals, register the output of any combinational logic used as a clock signal to reduce glitches.


The Design Assistant in the Quartus II software is a design-rule checking tool that enables you to check for design issues early in the design flow. The Design Assistant checks your design for adherence to Altera-recommended design guidelines or design rules. To run the Design Assistant, on the Processing menu, point to **Start** and click **Start Design Assistant**. To set the Design Assistant to run automatically during compilation, turn on **Run Design Assistant during compilation** in the **Settings** dialog box. You can also use third-party "lint" tools to check your coding style.

You should also understand the target architecture of your device to target your design to take advantage of those features. For example, the control signals should use the dedicated control signals in the device architecture, so in some cases you might be required to limit the number of different control signals used in your design to achieve the best results.

 For more information about design recommendations and using the Design Assistant, refer to the *Design Recommendations for Altera Devices and the Quartus II Design Assistant* chapter in volume 1 of the *Quartus II Handbook*. You can also refer to industry papers for more information about multiple clock design. For a good analysis, refer to *Synthesis and Scripting Techniques for Designing Multi-Asynchronous Clock Designs* under **Papers** (www.sunburst-design.com).

Recommended HDL Coding Styles

HDL coding styles can have a significant effect on the quality of results for programmable logic designs. You can use Altera's recommended coding styles to achieve optimal synthesis results. If you are designing memory and DSP functions, you should understand your device's target architecture so you can take advantage of the dedicated logic block sizes and configurations. Follow the coding guidelines for inferring megafunctions and targeting dedicated device hardware, such as memory and DSP blocks.


 For specific HDL coding examples and recommendations, refer to the *Recommended HDL Coding Styles* chapter in volume 1 of the *Quartus II Handbook*. For any additional tool-specific guidelines, refer to your synthesis tool's documentation. In the Quartus II software, you can use the HDL examples in the Language Templates available from the right-click menu in the text editor.

Managing Metastability

Metastability problems can occur in digital design when a signal is transferred between circuitry in unrelated or asynchronous clock domains, because the designer cannot guarantee that the signal meets the setup and hold time requirements during the signal transfer. Designers commonly use a synchronization chain to minimize the occurrence of metastable events.


You can use the Quartus II software to analyze the average mean time between failures (MTBF) due to metastability when a design synchronizes asynchronous signals, and optimize the design to improve the metastability MTBF. The MTBF due to metastability is an estimate of the average time between instances when metastability could cause a design failure. A high MTBF (such as hundreds or thousands of years between metastability failures) indicates a more robust design. Determine an acceptable target MTBF given the context of your entire system and the fact that MTBF calculations are statistical estimates.

The Quartus II software can help you determine whether you have enough synchronization registers in your design to produce a high enough MTBF at your clock and data frequencies.

 For information about the industry-leading metastability analysis, reporting, and optimization features in the Quartus II software, refer to the *Managing Metastability with the Quartus II Software* chapter in volume 1 of the *Quartus II Handbook*.

Planning for Hierarchical and Team-Based Design

If you want to create a hierarchical design that can take advantage of the compilation-time savings and performance preservation of the Quartus II software incremental compilation, plan for an incremental compilation flow from the beginning of your design cycle. The following subsections describe the flat compilation flow, in which the design hierarchy is flattened without design partitions, and then the incremental compilation flows that use design partitions. Incremental compilation flows offer several advantages but require more design planning to ensure good quality of results. The last subsections discuss factors to consider when planning an incremental compilation flow: planning design partitions and creating a design floorplan.

 For information about using the incremental compilation flows in the Quartus II software, refer to the *Quartus II Incremental Compilation for Hierarchical and Team-Based Design* chapter in volume 1 of the *Quartus II Handbook*.

Flat Compilation Flow with No Design Partitions

In this compilation flow in the Quartus II software, the entire design is compiled together in a “flat” netlist. This flow is used if you do not create any design partitions. Your source code can have hierarchy, but the design is flattened during compilation and all the design source code is synthesized and fit in the target device whenever the design is recompiled after any change in the design. By processing the entire design, the software performs all available logic and placement optimizations on the entire design to improve area and performance. You can use debugging tools in an incremental design flow, such as the SignalTap II Logic Analyzer, but you do not specify any design partitions to preserve design hierarchy during compilation.

The flat compilation flow is easy to use; you do not have to plan any design partitions. However, because the entire design is recompiled whenever there are any changes to the design, compilation times can be relatively long for large devices. In addition, you may find that the results for one part of the design change when you change a different part of your design.

Incremental Compilation with Design Partitions

In an incremental compilation flow, the system architect splits a large design into partitions which can be designed separately. Team members can work on partitions independently, which can simplify the design process and reduce compilation time.

When hierarchical design partitions are well chosen and placed in the device floorplan, you can speed up your design compilation time while maintaining or even improving the quality of results.

You may want to use incremental compilation later in the design cycle when you are not interested in improving the majority of the design any further, and want to make changes to, or optimize, one specific block. In this case, you may want to preserve the performance of modules that are unmodified and reduce compilation time on subsequent iterations.

Incremental compilation may also be useful for both reducing compilation time and achieving timing closure. For example, you may want to specify which partitions should be preserved in subsequent incremental compilations and then recompile the other partitions with advanced optimizations turned on.

If a part of your design is not yet complete, you can create an empty partition for the incomplete part of the design while compiling the completed partitions. Then, save the results for the complete partitions while you work on the new part of the design.

Alternately, different designers or IP providers may be working on different blocks of the design using a team-based methodology, and you may want to combine these blocks in a bottom-up compilation flow.

If you are planning your design code and hierarchy, ensure that each design entity is created in a separate file so the entities remain independent when you make source code changes in the file. If you use a third-party synthesis tool, create separate Verilog Quartus Mapping (VQM) or EDIF netlists for each design partition in your synthesis tool. You may have to create separate projects within your synthesis tool, so the tool synthesizes each partition separately and generates separate output netlist files. Refer to your synthesis tool documentation for information about support for Quartus II incremental compilation. The netlists are then considered the source files for incremental compilation.

Single-Project Versus Multiple-Project Incremental Flows

The Quartus II incremental compilation feature supports various design methodologies.

The easiest compilation methodology is having one designer or project lead who compiles the entire design in the software. Different designers or IP providers can design and verify different parts of the design, and the project lead can add design entities to the project as they are completed. You can also target optimizations on one part of the design while designating the rest of the design as “empty.” Regardless of the source for all the design logic, the project lead compiles and optimizes the top-level project as a whole.

Incremental compilation preserves the compilation results and performance of unchanged partitions in your design, greatly reducing design iteration time by focusing new compilations on changed design partitions only. New compilation results are then merged with the previous compilation results from unchanged design partitions. Additionally, you can target optimization techniques, such as physical synthesis, to specific design partitions while leaving other partitions untouched. You can also use this flow with empty partitions if parts of your design are incomplete or missing.

If individual designers or IP providers want to complete the optimization of their design in separate projects, they can integrate each lower-level project into one top-level project.

Incremental compilation provides export and import features to enable this type of design methodology. Designers of lower-level blocks can export the optimized netlist for their design, along with a set of assignments, such as LogicLock™ regions. The system architect then imports each design block as a design partition in a top-level project.

With imported partitions, it is very important that the system architect provide guidance to designers of lower-level blocks to ensure that each partition uses the appropriate device resources. Because the designs are developed independently, each lower-level designer has no information about the overall design or how their partition connects with other partitions. This lack of information can lead to problems during system integration. The top-level project information, including pin locations, physical constraints, and timing requirements, is communicated to the designers of lower-level partitions before they start their design.

The system architect can plan design partitions at the top level and use Quartus II incremental compilation to communicate information to lower-level designers through automatically-generated scripts. The **Generate bottom-up design partition scripts** option automates the process of transferring top-level project information to lower-level modules. The software provides a project manager interface for managing project information in the top-level design.

The scripts can create Quartus II projects for all the lower-level design blocks and pass all the relevant project assignments. Using these scripts makes it easier for designers of lower-level modules to implement the instructions from the project lead, and avoid conflicts between projects when importing and incorporating the projects into the top-level design. You can use this methodology to help reduce the need to further optimize the designs after integration and improve overall designer productivity and team collaboration.

You can combine compilation flows to take advantage of a single Quartus II project for part of your design, while importing parts of the design that are developed independently.

The single-project flow is generally simpler to perform. For example, having to export and import lower-level designs is eliminated, and having a single project provides the design software with information about the entire design, so it can perform global placement optimizations when no part of the design is locked down to a specific location.


Planning Design Partitions

Partitioning a design for an FPGA requires planning to ensure optimal results when the partitions are integrated, and ensure that each partition is placed well relative to other partitions in the device. Following Altera's recommendations for creating design partitions improves the overall quality of results. For example, registering partition I/O boundaries keeps critical timing paths inside one partition that can be optimized independently. When the design partitions are specified, you can use the Incremental Compilation Advisor to ensure that partitions meet Altera's recommendations.

Determining a timing budget before designers develop their individual blocks reduces the chance of timing problems during system integration. If you optimize lower-level partitions separately, any unregistered paths that cross between partitions are not optimized as an entire path. To ensure that the software correctly optimizes the input and output logic in each partition, you can perform some manual timing budgeting. For each unregistered timing path that crosses between partitions, Altera recommends creating timing assignments on the corresponding I/O path in each partition to constrain both ends of the path to the budgeted timing delay. Assigning a timing budget for each part of the connection ensures that the software optimizes paths appropriately so they meet the top-level design requirements.

You can also plan and balance your resource utilization. If you are performing incremental compilation, the software synthesizes each partition separately, with no data about the resources used in other partitions. Therefore, device resources can be overused in the individual partitions during synthesis, and the design may not fit in the target device when the partitions are merged.

In a design flow in which designers optimize their lower-level designs and export them to a top-level design, the software also places and routes each partition separately. In some cases, partitions can use conflicting resources when combined at the top level. Balancing resource utilization between the design partitions avoids any problems with conflicting resources when all the partitions are integrated.


 For guidelines on creating design partitions and organizing your source code, refer to the *Best Practices for Incremental Compilation Partitions and Floorplan* chapter in volume 1 of the *Quartus II Handbook*.

Creating a Design Floorplan

To take full advantage of incremental compilation, creating a design floorplan prevents conflicts between design partitions, and ensures that each partition is placed well relative to other partitions. Creating location assignments for each partition ensures that no conflicts occur for locations between different partitions. In addition, a design floorplan helps to avoid a situation in which the Fitter is directed to place or replace a portion of the design in an area of the device in which most resources are claimed. Without floorplan assignments, this situation can lead to increased compilation time and reduced quality of results.

You can use the Quartus II Chip Planner to create a design floorplan using LogicLock region assignments for each design partition. With a basic design framework for the top-level design, these floorplan editors allow you to view connections between regions, estimate physical timing delays on the chip, and move regions around the device floorplan. When you have compiled the full design, you can also view logic placement and locate areas of routing congestion to improve the floorplan assignments.

Good partition and floorplan design helps lower-level designs meet top-level design requirements when integrated with the rest of the design, reducing the time spent integrating and verifying the timing of the top-level design.

 For information about creating placement assignments in the design floorplan, refer to the *Analyzing and Optimizing the Design Floorplan* chapter in volume 2 of the *Quartus II Handbook*. For guidelines on creating a design floorplan for incremental compilation, refer to the *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* chapter in volume 1 of the *Quartus II Handbook*.

Fast Synthesis and Early Timing Estimation

It is more cost-effective to find design issues early in the design cycle than to find problems in the final timing closure stages. When the first version of the design source code is complete, you may want to perform a quick compilation to create a kind of silicon virtual prototype (SVP) that you can use to perform timing analysis.

If you synthesize with the Quartus II software, you can choose to perform a **Fast** synthesis, which reduces the compilation time but may give reduced quality of results. On the Assignments menu, click **Settings**. On the **Analysis & Synthesis Settings** tab, click **More Settings** and set the **Synthesis Effort**.

Regardless of your compilation flow, you can use the an Early Timing Estimate to perform a quick placement and routing, and a timing analysis of your design. On the Processing menu, point to **Start**, and click **Start Early Timing Estimate**. The software chooses a device automatically if required, places any LogicLock regions used to create a floorplan, finds a quick initial placement for all the design logic, and provides a useful estimate of the final design performance. If you have entered timing constraints, timing analysis reports on these constraints.

If you are designing individual blocks separately, you can use these features as you develop the design. Any issues highlighted in the lower-level design blocks are communicated to the system architect. Resolving these issues might require allocating additional device resources to the individual block or changing its timing budget.

If you are a top-level designer, you can also use fast synthesis and early timing estimation to prototype the entire design. Incomplete partitions are marked as empty in an incremental compilation flow, while the rest of the design is compiled to get an early timing estimate and detect any problems with design integration.

A system architect can use early timing estimation along with design partition scripts (as described in [“Planning for Hierarchical and Team-Based Design”](#) on page 1-14) to pass additional constraints to lower-level designers, and provide more information about the other partitions in the design. This information is especially useful to optimize cross-partition paths. Running early timing estimations helps you to find and resolve design problems during the early design stages.

Conclusion

Modern FPGAs support large, complex designs with fast timing performance. By planning several aspects of your design early in the process, you can reduce unnecessary time spent handling issues in later stages of the process. You can use various features of the Quartus II software to quickly plan your design and achieve the best possible results. Following the guidelines presented in this chapter can improve productivity, which reduces the design cost and improves the final product’s time to market.

Referenced Documents

This chapter references the following documents:

- [Analyzing and Optimizing the Design Floorplan](#) chapter in volume 2 of the *Quartus II Handbook*
- [AN 386: Using the MAX II Parallel Flash Loader with the Quartus II Software](#)
- [Best Practices for Incremental Compilation Partitions and Floorplan Assignments](#) chapter in volume 1 of the *Quartus II Handbook*
- [Cadence PCB Design Tools](#) chapter in volume 2 of the *Quartus II Handbook*
- [Configuration Handbook](#)
- [Design Debugging Using the SignalTap II Embedded Logic Analyzer](#) chapter in volume 3 of the *Quartus II Handbook*
- [Design Debugging Using In-System Sources and Probes](#) chapter in volume 3 of the *Quartus II Handbook*
- [Design Recommendations for Altera Devices and the Quartus II Design Assistant](#) chapter in volume 1 of the *Quartus II Handbook*
- [Formal Verification](#) section in volume 3 of the *Quartus II Handbook*
- [I/O Management](#) chapter in volume 2 of the *Quartus II Handbook*
- [In-System Debugging Using External Logic Analyzers](#) chapter in volume 3 of the *Quartus II Handbook*
- [In-System Updating of Memory and Constants](#) chapter in volume 3 of the *Quartus II Handbook*
- [Introduction to the Quartus II Software](#)


- *Mentor Graphics PCB Design Tools Support* chapter in volume 2 of the *Quartus II Handbook*
- *PowerPlay Power Analysis* chapter in volume 3 of the *Quartus II Handbook*
- *Quartus II Incremental Compilation for Hierarchical and Team-Based Design* chapter in volume 1 of the *Quartus II Handbook*
- *Quick Design Debugging Using SignalProbe* chapter in volume 3 of the *Quartus II Handbook*
- *Simulation* section in volume 3 of the *Quartus II Handbook*
- *sld_virtual_jtag Megafunction User Guide*
- *Synthesis* section in volume 1 of the *Quartus II Handbook*

Document Revision History

Table 1-1 shows the revision history for this chapter.

Table 1-1. Document Revision History

Date and Document Version	Changes Made	Summary of Changes
November 2009 v9.1.0	<ul style="list-style-type: none"> ■ Added details to “Creating Design Specifications” on page 1-2 ■ Added details to “Intellectual Property Selection” on page 1-2 ■ Updated information on “Device Selection” on page 1-3 ■ Added reference to “Device Migration Planning” on page 1-4 ■ Removed information from “Planning for Device Programming or Configuration” on page 1-4 ■ Added details to “Early Power Estimation” on page 1-5 ■ Updated information on “Early Pin Planning and I/O Analysis” on page 1-6 ■ Updated information on “Creating a Top-Level Design File for I/O Analysis” on page 1-8 ■ Added new “Simultaneous Switching Noise Analysis” section ■ Updated information on “Synthesis Tools” on page 1-9 ■ Updated information on “Simulation Tools” on page 1-9 ■ Updated information on “Planning for On-Chip Debugging Options” on page 1-10 ■ Added new “Managing Metastability” section ■ Changed heading title “Top-Down Versus Bottom-Up Incremental Flows” to “Single-Project Versus Multiple-Project Incremental Flows” ■ Updated information on “Creating a Design Floorplan” on page 1-18 ■ Removed information from “Fast Synthesis and Early Timing Estimation” on page 1-18 	Updated for the Quartus II 9.1 software release.
March 2009 v.9.0.0	<ul style="list-style-type: none"> ■ No change to content 	Updated for the Quartus II 9.0 software release.
November 2008 v8.1.0	<ul style="list-style-type: none"> ■ Changed to 8-1/2 x 11 page size. No change to content. 	Updated for the Quartus II 8.1 software release.
May 2008 v8.0.0	<ul style="list-style-type: none"> ■ Organization changes ■ Added “Creating Design Specifications” section ■ Added reference to new details in the In-System Design Debugging section of volume 3 ■ Added more details to the “Design Practices and HDL Coding Styles” section ■ Added references to the new Best Practices for Incremental Compilation and Floorplan Assignments chapter ■ Added reference to the Quartus II Language Templates 	Updated for the Quartus II 8.0 software release and related documentation; expanded and improved organization of topic coverage.

 For previous versions of the *Quartus II Handbook*, refer to the [Quartus II Handbook Archive](#).