


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

Because of the significant increase in FPGA device densities, designs are complex and can sometimes involve multiple designers. System architects must resolve design issues when integrating design blocks, often leading to problems that affect the overall time to market and thereby increasing cost. You can solve potential problems early in the design cycle by following the design planning considerations provided in this chapter.


This chapter contains the following sections:

- “Creating Design Specifications” on page 1–2
- “Selecting Intellectual Property” on page 1–2
- “Using Qsys and Standard Interfaces in System Design” on page 1–3
- “Selecting a Device” on page 1–3
- “Planning for Device Programming or Configuration” on page 1–5
- “Estimating Power” on page 1–5
- “Early Pin Planning and I/O Analysis” on page 1–6
- “Selecting Third-Party EDA Tools” on page 1–9
- “Planning for On-Chip Debugging Tools” on page 1–10
- “Design Practices and HDL Coding Styles” on page 1–11
- “Planning for Hierarchical and Team-Based Design” on page 1–13
- “Fast Synthesis and Early Timing Estimation” on page 1–15

 This chapter provides only an introduction to various design planning features in the Quartus® II software. For more information about specific Quartus II features and methodologies, this chapter provides references to other appropriate chapters in the *Quartus II Handbook*.

Before reading the design planning guidelines discussed in this chapter, consider your design priorities. More device features, density, or performance requirements can increase system cost. Signal integrity and board issues can impact I/O pin locations. Power, timing performance, and area utilization all affect each other, and compilation time is affected when optimizing these priorities.

The Quartus II software optimizes designs for the best results, but you can change the settings to better optimize of one aspect of your design. Certain tools or debugging options can lead to restrictions in your design flow. Your design priorities help you choose the tools, features, and methodologies to use for the design.

 After you select a device family, to check if additional guidelines are available, refer to the design guidelines section of the appropriate device handbook.

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.

In addition, creating a test plan helps you to design for verification and manufacturability. For example, you might need to validate interfaces incorporated in the design. 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. For guidelines related to analyzing and debugging the device after it is in the system, refer to [“Planning for On-Chip Debugging Tools” on page 1–10](#).


If more than one designer works on your design, you should consider a common design directory structure or source control system to make design integration easier. For more suggestions on team-based designs, refer to [“Planning for Hierarchical and Team-Based Design” on page 1–13](#). Consider whether you want to standardize on an interface protocol for each design block. To improve reusability and ease of integration, refer to [“Using Qsys and Standard Interfaces in System Design”](#).

Selecting Intellectual Property

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 you select 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 the following modes:

- Untethered—the design runs for a limited time.
- Tethered—the design 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 of the Altera website.

Using Qsys and Standard Interfaces in System Design

You can use the Quartus II Qsys system integration tool to create your design with fast and easy system-level integration. With Qsys, you can specify system components in a GUI and generate the required interconnect logic automatically, along with adapters for clock crossing and width differences. Because system design tools change the design entry methodology, you should plan to start developing your design within the tool. Ensure all design blocks use appropriate standard interfaces from the beginning of the design cycle so that you do not need to make changes later.

Qsys components use Avalon® standard interfaces for the physical connection of components, and you can connect any logical device (either on-chip or off-chip) that has an Avalon interface. The Avalon Memory-Mapped interface allows a component to use an address mapped read or write protocol that enables flexible topologies for connecting master components to any slave components. The Avalon Streaming interface enables point-to-point connections between streaming components that send and receive data using a high-speed, unidirectional system interconnect between source and sink ports.



In addition to enabling the use of a system integration tool such as Qsys, using standard interfaces ensures compatibility between design blocks from different design teams or vendors. Standard interfaces simplify the interface logic to each design block and enable individual team members to test their individual design blocks against the specification for the interface protocol to ease system integration.

-  For more information about using Qsys to improve your productivity, refer to the *System Design with Qsys* section in volume 1 of the *Quartus II Handbook*.
-  Qsys replaces the SOPC Builder system integration tool for new designs. For more information about SOPC Builder, refer to the *SOPC Builder User Guide*.

Selecting a Device


The device you choose affects board specification and layout. This section provides guidelines in the device selection process.

Choose the device family that best suits your design requirements. Families differ in 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 use the [Altera Product Selector](#) available on the Altera website to help you choose your device. You can also review important features of each device family in the [Selector Guides](#) page of the Altera website. Each device family also has a device handbook, including a data sheet, which documents device features in detail. You can also see a summary of the resources for each device in the **Device** dialog box in the Quartus II software.
-  For a list of device selection guides, refer to *Devices and Adapters* in Quartus II Help.


Carefully study the device density requirements for your design. Devices with more logic resources and higher I/O counts can implement larger and more complex designs, but at a higher cost. Smaller devices use lower static power. Select a device larger than what is required for your design, in case you want to add more logic later in the design cycle to upgrade or expand your design, and reserve logic and memory for on-chip debugging (refer to [“Planning for On-Chip Debugging Tools” 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.

If you have older designs that target an Altera device, you can use their resources as an estimate for your 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 learn which device density fits the design. Consider coding style, device architecture, and the optimization options used in the Quartus II software, which can significantly affect the resource utilization and timing performance of your design.

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


Device Migration Planning

Determine whether you want to migrate 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, you may target a smaller (and less expensive) device and then move to a larger device if necessary to meet your 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.

-  For more information about specifying the target migration devices, refer to [Specifying Devices for Device Migration](#) in Quartus II Help.


Selecting a migration device impacts pin placement because some pins may serve different functions in different device densities or package sizes. If you make 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. Consider these issues early in the design cycle rather than at the end, when the design is near completion and ready for migration.

Additionally, if you plan to migrate your design to a HardCopy ASIC, review HardCopy guidelines early in the design cycle for any Quartus II settings that you should use or other restrictions you should consider. You must use complete timing constraints if you want to migrate to a HardCopy ASIC because of the rigorous verification requirements for ASIC devices.

-  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

Planning how to program or configure the device in your system allows system and board designers to determine what companion devices, if any, your system requires. 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. Additionally, 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 you complete configuration. The Quartus II software performs voltage compatibility checks of those pins during I/O assignment analysis and compilation of your design. You can use the **Configuration** tab of the **Device and Pin Options** dialog box to 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.

Estimating Power

You can use the Quartus II power estimation and analysis tools to provide information to PCB board and system designers. Power consumption in FPGA devices depends on the logic design, which can make planning difficult. You can estimate power 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 you complete the design.

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**—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**—ensure that the power supplies provide adequate current to support device operation.

The PowerPlay Early Power Estimator (EPE) spreadsheet allows you to estimate power utilization for your design.

You can manually enter data into the EPE spreadsheet, or use the Quartus II software to generate device resource information for your design.


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

If you have an existing design or a partially completed design, you can use the Quartus II software to generate the PowerPlay Early Power Estimator File (.txt, .csv) to assist you in completing the PowerPlay EPE spreadsheet.


- ② For more information about generating the PowerPlay EPE File, refer to *Performing an Early Power Estimate Using the PowerPlay Early Power Estimator* in Quartus II Help.

The PowerPlay EPE spreadsheet includes the Import Data macro that parses the information in the PowerPlay EPE File and transfers the information 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 device resource information. If the existing Quartus II project represents only a portion of your full design, manually enter the additional device resources you use in the final design.

Estimating power consumption early in the design cycle allows planning of power budgets and avoids unexpected results when designing the PCB.

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

When you complete the design, 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 limitations are met.

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

Early Pin Planning and I/O Analysis

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

You can create a preliminary pin-out for an Altera FPGA with the Quartus II Pin Planner before you develop the source code, based on standard I/O interfaces (such as memory and bus interfaces) and any other I/O-related assignments defined by system requirements. The Quartus II I/O Assignment Analysis checks that the pin locations and assignments are supported in the target FPGA architecture. You can

then 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, I/O Assignment Analysis runs automatically in the Fitter to validate that the assignments meet all the device requirements and generates error messages.

This section describes pin planning and I/O analysis features for different stages of the design flow.

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 that 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 to connect the functions and cores 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, as described in this section. 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.

- ④ For more information about setting up the nodes in your design, refer to [Set Up Top-Level Design File Window \(Edit Menu\)](#) in Quartus II Help.

You can use the I/O analysis results to change pin assignments or IP parameters even before you create the design, 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 you complete initial pin planning, you can create a revision based on the Quartus II-generated netlist. You can then use the generated netlist to develop the top-level design file for the design, or disregard the generated netlist and use the generated Quartus II Settings File (.qsf) with the design.

During this initial pin planning, after you have generated a top-level design file, or when you have developed your design source code, you can assign pin locations and assignments with the Pin Planner.


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 with 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 the pins that have changed functions in the migration device when compared to the currently

selected device. Selecting the pins in the Device Migration view cross-probes to the rest of the Pin Planner, so that 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 that you get very accurate I/O timing analysis. You can 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 Comma-Separated Value File (.csv) containing your I/O assignments for spreadsheet use when you assign all pins.


When you complete your pin planning, you can pass pin location information 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 suggested changes. Your pin assignments must match between the Quartus II software and your schematic and board layout tools to ensure the FPGA works correctly on the board, especially if you must make changes to the pin-out. The system architect uses the Quartus II software to pass pin information to team members designing individual logic blocks, allowing them to achieve better timing closure when they compile their design.

Start FPGA planning before you complete the HDL for your design to improve the confidence in early board layouts, reduce the chance of error, and improve the overall time to market of the design. When you complete the design, use the Fitter reports for the final sign-off of pin assignments. After compilation, the Quartus II software generates the Pin-Out File (.pin), and you can use this file to verify that each pin is correctly connected in board schematics.

 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*.

Simultaneous Switching Noise Analysis

Simultaneous switching noise (SSN) is 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. Altera provides tools for SSN analysis and estimation, including SSN characterization reports, an Early SSN Estimator (ESE) spreadsheet tool, and the SSN Analyzer in the Quartus II software. SSN often leads to the degradation of signal integrity by causing signal distortion, thereby reducing the noise margin of a system. You should address SSN with estimation early in your system design, to minimize later board design changes. When the design is complete, verify your board design by performing a complete SSN analysis of your FPGA in the Quartus II software.


 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 Tools

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 properly, and that you are aware of their capabilities.


Synthesis Tool

The Quartus II software includes integrated synthesis that supports Verilog HDL, VHDL, 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 determine the best 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, you should use the most recent version of third-party synthesis tools. The *Quartus II Software Release Notes* lists the version of each synthesis tool that is supported by a given version of the Quartus II software.

The synthesis tool you choose may allow you 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 save time when setting up your Quartus II project for placement and routing.

To use incremental compilation, 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 1-14.


 For more information about synthesis tool flows, refer to *Volume 1: Design and Synthesis* of the *Quartus II Handbook*.

Simulation Tool

Altera provides the ModelSim®-Altera 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 software 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.

For a list of the version of each simulation tool that is supported with a given version of the Quartus II software, refer to the *Quartus II Software Release Notes*.

 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 Tool

Consider whether the formal verification tool that you want to use is supported by the Quartus II software, and whether the flow impacts the design and compilation stages of your design.

- For more information about formal verification flows and the supported tools, refer to *Volume 3: Verification* of the *Quartus II Handbook*.

Using a formal verification tool can impact performance results because performing formal verification requires turning off certain logic optimizations, such as register retiming, and forces you to preserve hierarchy blocks, which can restrict optimization. Formal verification treats memory blocks as black boxes. Therefore, you should keep memory in a separate hierarchy block so other logic does not get incorporated into the black box for verification. Other restrictions may limit your design, and you should consult *Volume 3: Verification* of the *Quartus II Handbook* for details. If formal verification is important to your design, plan for limitations and restrictions at the beginning of the design cycle rather than make changes later.

Planning for On-Chip Debugging Tools

In-system debugging tools offer different advantages and trade-offs. A particular debugging tool may work better for different systems and designers. You should evaluate on-chip debugging tools early in your design process, to ensure that your system board, Quartus II project, and design can support the appropriate tools. You can reduce debugging time and avoid later changes to accommodate your preferred debugging tools.

- For more information about debugging tools, refer to *Section IV. In-System Debugging* in volume 3 of the *Quartus II Handbook*. For an overview of debugging tools that can help you decide which tools to use, refer to the *System Debugging Tools Overview* chapter in volume 3 of the *Quartus II Handbook*.

If you intend to use any of these tools, you may have to plan for the tools when developing your system board, Quartus II project, and design. Consider the following debugging requirements when you plan your design:

- JTAG connections—required to perform in-system debugging with JTAG tools. Plan your system and board with JTAG ports that are available for debugging.
- Additional logic resources—required to implement JTAG hub logic. If you set up the appropriate tool early in your design cycle, you can include these device resources in your early resource estimations to ensure that you do not overload the device with logic.
- Reserve device memory—required if your tool 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 use during debugging.

- Reserve I/O pins—required if you use the Logic Analyzer Interface (LAI) or SignalProbe tools, which require I/O pins for debugging. If you reserve I/O pins for debugging, you do not have to later change the design or board. The LAI can multiplex signals with design I/O pins if required. Ensure that your board supports a debugging mode, where debugging signals do not affect system operation.
- Instantiate a megafunction in your HDL code—required if your debugging tool uses a Quartus II megafunction.
- Instantiate the SignalTap II Logic Analyzer as a megafunction—required if you want to manually connect the SignalTap II Logic Analyzer to nodes in your design and ensure that the tapped node names do not change during synthesis. You can add the analyzer as a separate design partition for incremental compilation to minimize recompilation times.

Table 1-1 lists which factors are important for each debugging tool.

Table 1-1. Factors to Consider When Using Debugging Tools During Design Planning Stages

Design Planning Factor	SignalTap II Logic Analyzer	System Console	In-System Memory Content Editor	Logic Analyzer Interface (LAI)	SignalProbe	In-System Sources and Probes	Virtual JTAG Megafunction
JTAG connections	✓	✓	✓	✓	—	✓	✓
Additional logic resources	—	✓	—	—	—	—	✓
Reserve device memory	✓	✓	—	—	—	—	—
Reserve I/O pins	—	—	—	✓	✓	—	—
Instantiate a megafunction in your HDL code	—	—	—	—	—	✓	✓

Design Practices and HDL Coding Styles

When you develop complex FPGA designs, design practices and coding styles have an enormous impact on the timing performance, logic utilization, and system reliability of your device.

Design Recommendations

Use synchronous design practices to consistently meet your design goals. Problems with asynchronous 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. When you meet all register timing requirements, 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.

Clock signals have a large effect on the timing accuracy, performance, and reliability of your design. Problems with clock signals can cause functional and timing problems in your design. Use dedicated clock pins and clock routing for best results, and if you have PLLs 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 verify design issues. The Design Assistant checks your design for adherence to Altera-recommended design guidelines. You can also use third-party lint tools to check your coding style.

- For more information about running the Design Assistant, refer to *About the Design Assistant* in Quartus II Help.

Consider the architecture of the device you choose so that you can use specific features in your design. For example, the control signals should use the dedicated control signals in the device architecture. In some cases, you might need 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 *Recommended Design Practices* 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. If you design memory and DSP functions, you should understand the target architecture of your device so you can use 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 additional tool-specific guidelines, refer to the documentation of your synthesis tool.

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. Ensure that your design accounts for synchronization between any asynchronous clock domains. Consider using a synchronizer chain of more than two registers for high-frequency clocks and frequently-toggling data signals to reduce the chance of a metastability failure.


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 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

To create a hierarchical design so that you can use compilation-time savings and performance preservation with the Quartus II software incremental compilation feature, 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 flow that uses design partitions. Incremental compilation flows offer several advantages, but require more design planning to ensure effective results. The last subsections discuss planning an incremental compilation flow, planning design partitions, and optionally creating a design floorplan.

-  For information about using the incremental compilation flow methodology 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 the flat compilation flow with no design partitions in the Quartus II software, the entire design is compiled together in a “flat” netlist. 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 you change the design, compilation times can be slow for large devices. Additionally, you may find that the results for one part of the design change when you change a different part of your design. You can turn on the **Rapid Recompile** option to instruct the software to preserve compatible placement and routing results when the design changes in subsequent compilations. This option can reduce your compilation time in a flat or partitioned design when you make very small changes to the design.

Incremental Compilation with Design Partitions

In an incremental compilation flow, the system architect splits a large design into partitions. When hierarchical design partitions are well chosen and placed in the device floorplan, you can speed up your design compilation time while maintaining the quality of results.

Incremental compilation preserves the compilation results and performance of unchanged partitions in the 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 unchanged. You can also use empty partitions to indicate that parts of your design are incomplete or missing, while you compile the rest of the design.

Third-party IP designers can also export logic blocks to be integrated into the top-level design. Team members can work on partitions independently, which can simplify the design process and reduce compilation time. With exported partitions, the system architect must provide guidance to designers or IP providers to ensure that each partition uses the appropriate device resources. Because the designs may be developed independently, each 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, must be communicated to the designers of lower-level partitions before they start their design.

The system architect plans design partitions at the top level and allows third-party designs to access the top-level project framework. By designing within a copy of the top-level project (or by checking out the project files in a source control environment), the designers of the lower-level block have full information about the entire project, which helps to ensure optimal results.

When you plan your design code and hierarchy, ensure that each design entity is created in a separate file so that 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 or EDIF netlists for each design partition in your synthesis tool. You may have to create separate projects within your synthesis tool, so that the tool synthesizes each partition separately and generates separate output netlist files. The netlists are then considered the source files for incremental compilation.





For more information about support for Quartus II incremental compilation, refer to the *Quartus II Incremental Compilation for Hierarchical and Team-Based Design* chapter of the *Quartus II Handbook*.

Planning Design Partitions and Floorplan Location Assignments

Partitioning a design for an FPGA requires planning to ensure optimal results when you integrate the partitions. Following Altera's recommendations for creating design partitions should improve 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 you specify the design partitions, you can use the Incremental Compilation Advisor to ensure that partitions meet Altera's recommendations.


If you have timing-critical partitions that are changing through the design flow, or partitions exported from another Quartus II project, you can create design floorplan assignments to constrain the placement of the affected partitions. Good partition and floorplan design helps partitions meet top-level design requirements when integrated with the rest of the design, reducing time you spend integrating and verifying the timing of the top-level design.

-  For detailed guidelines about creating design partitions and organizing your source code, as well as information about when and how to create floorplan assignments, refer to the *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* chapter in volume 1 of the *Quartus II Handbook*.
-  For more information about creating floorplan assignments in the Chip Planner, refer to the *Analyzing and Optimizing the Design Floorplan* chapter in volume 2 of the *Quartus II Handbook*.


Fast Synthesis and Early Timing Estimation

You save time when you find design issues early in the design cycle rather than in the final timing closure stages. When the first version of the design source code is complete, you might 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.

-  For more information about Fast synthesis, refer to *Synthesis Effort logic option* in Quartus II Help.

Regardless of your compilation flow, you can run an early timing estimate to perform a quick placement and routing, and a timing analysis of your design. 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.

-  For more information about how to run an early timing estimate, refer to *Running a Timing Analysis* in Quartus II Help.

If you design individual design blocks or partitions separately, you can use the Fast synthesis and early timing estimate 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 partition, or changing the timing budget of the partition.

Expert designers 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.

Conclusion

Modern FPGAs support large, complex designs with fast timing performance. By planning several aspects of your design early, you can reduce time in later stages of the development cycle. Use 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 can reduce cost and development time.

Document Revision History


Table 1-2 shows the revision history for this chapter.

Table 1-2. Document Revision History (Part 1 of 2)

Date	Version	Changes
November 2011	11.0.1	Template update.
May 2011	11.0.0	<ul style="list-style-type: none"> ■ Added link to System Design with Qsys in “Creating Design Specifications” on page 1-2 ■ Updated “Simultaneous Switching Noise Analysis” on page 1-8 ■ Updated “Planning for On-Chip Debugging Tools” on page 1-10 ■ Removed information from “Planning Design Partitions and Floorplan Location Assignments” on page 1-15
December 2010	10.1.0	<ul style="list-style-type: none"> ■ Changed to new document template ■ Updated “System Design and Standard Interfaces” on page 1-3 to include information about the Qsys system integration tool ■ Added link to the Altera Product Selector in “Device Selection” on page 1-3 ■ Converted information into new table (Table 1-1) in “Planning for On-Chip Debugging Options” on page 1-10 ■ Simplified description of incremental compilation usages in “Incremental Compilation with Design Partitions” on page 1-14 ■ Added information about the Rapid Recompile option in “Flat Compilation Flow with No Design Partitions” on page 1-14 ■ Removed details and linked to Quartus II Help in “Fast Synthesis and Early Timing Estimation” on page 1-16

Table 1-2. Document Revision History (Part 2 of 2)

Date	Version	Changes
July 2010	10.0.0	<ul style="list-style-type: none"> ■ Added new section “System Design” on page 1-3 ■ Removed details about debugging tools from “Planning for On-Chip Debugging Options” on page 1-10 and referred to other handbook chapters for more information ■ Updated information on recommended design flows in “Incremental Compilation with Design Partitions” on page 1-14 and removed “Single-Project Versus Multiple-Project Incremental Flows” heading ■ Merged the “Planning Design Partitions” section with the “Creating a Design Floorplan” section. Changed heading title to “Planning Design Partitions and Floorplan Location Assignments” on page 1-15 ■ Removed “Creating a Design Floorplan” section ■ Removed “Referenced Documents” section ■ Minor updates throughout chapter
November 2009	9.1.0	<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
March 2009	9.0.0	<ul style="list-style-type: none"> ■ No change to content
November 2008	8.1.0	<ul style="list-style-type: none"> ■ Changed to 8-1/2 x 11 page size. No change to content.
May 2008	8.0.0	<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

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

 Take an [online survey](#) to provide feedback about this handbook chapter.