



# 1. Hierarchical Block-Based & Team-Based Design Flows

qii51001-1.0

## Introduction

Today's complex designs require multiple hardware description language (HDL) design files, each of which may undergo significant testing and optimization before being combined into the final top-level design. Many designs require work from more than one member of a design team. In this environment, the traditional flattened netlist approach to design may not be as effective as a hierarchical block-based design methodology.

This chapter discusses the differences between flattened and hierarchical design flows and provides details on the block-based or team-based hierarchical methodology. The chapter highlights the Altera® Quartus® II LogicLock™ design methodology, and discusses considerations for partitioning a design to support this methodology towards achieving optimal results.

## Design Planning: Flattened Design Flows versus Hierarchical Block-Based Design Flows

Most HDL-based designs use either a block-based or a flat design methodology. In flattened designs, you apply a single set of optimizations to the design's top level. Thus, a flattened synthesis flow has one output netlist file for the entire design. However, as designs become more complex and designers work in teams, a block-based hierarchical design flow is often more effective. In this approach, each sub-block has its own output netlist file and you perform optimization on individual sub-blocks. After you optimize all of the sub-blocks, you integrate them into a final design and optimize it at the top level. Synthesizing and optimizing each sub-block separately may provide better quality of results.

Using a block-based design methodology can also reduce the placement and routing changes required with each compilation in the Quartus II software. Using a hierarchical design approach limits the amount of logic impacted by engineering change orders (ECOs) that affect only one part of the design.

When you make small changes to a design, you can use incremental fitting for Cyclone™, Stratix™ II, or Stratix devices by choosing **Start > Start Incremental Fitting** (Processing menu). Incremental fitting can update the design's netlist and placement and routing, while ensuring that the timing characteristics of the design change as little as possible from those of the previous compilation. Also, using incremental fitting

helps to reduce the compilation time necessary to regenerate a netlist (although in rare situations, complexities in incremental fitting may cause longer compilation times).



For more information on incremental fitting and when it can be used, see the Quartus II Help.

When you make changes to a single block in the design, you can use the LogicLock design methodology to preserve your performance results, as discussed in [“Block-Based Design with the Quartus II LogicLock Methodology”](#) on page 1-4.

Table 1-1 describes each design flow and its advantages.

<b>Design Flow</b>	<b>Description</b>	<b>Advantages</b>
Traditional flattened	One output netlist for the entire design	You can perform optimization across design boundaries and hierarchies for the entire design. Simple to manage.
Block-based hierarchical	Separate netlist files for design modules	You compile each module separately. You can apply different optimization techniques to each module. Design modifications do not affect the optimization of other modules if the placement of other modules is locked down in the device. You can use optimized modules in other designs.

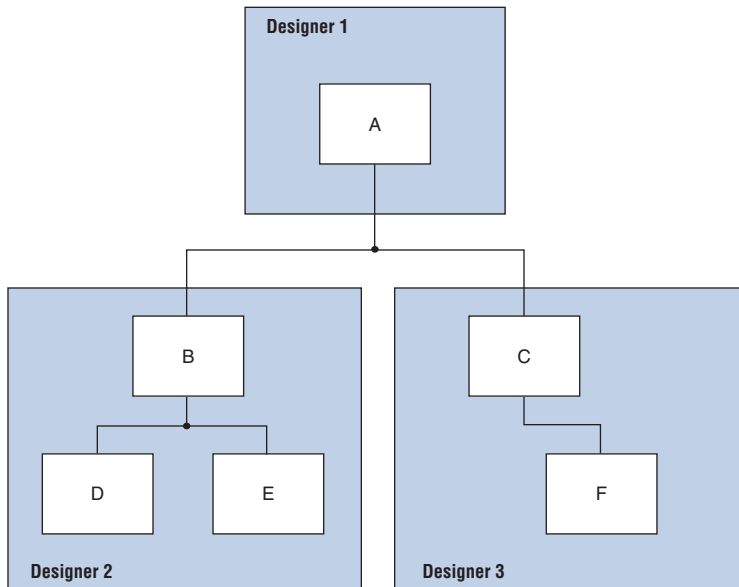
## **Block-Based & Team-Based Designs**

To take advantage of a block-based design flow, you must define different modules as a part of your design hierarchy in different files and instantiate them in a top-level file.

For larger designs, such as those implemented in Stratix II and Stratix devices, a team of designers may work on different modules of a design at the same time.

Figure 1-1 shows an example of a design hierarchy.

Figure 1-1. Quartus II Design Hierarchy



In Figure 1-1, the top-level design A is assigned to one engineer (designer 1), while two engineers work on the lower levels of the design. Designer 2 works on B and its submodules D and E, while designer 3 works on C and its submodule F.

You can treat each module or a group of modules as one block of the design for block-based synthesis. A submodule can be a Verilog HDL module, a VHDL entity, a Verilog Quartus Mapping (**.vqm**) or Electronic Data Interchange Format (**.edf**) netlist file, or any combination of these. During synthesis, you generate a separate VQM or EDF netlist file for each block of submodules. In this case, there is a separate netlist file for each block including modules A, B, and C.

To combine these submodules into a block for synthesis, they must form a single tree in the hierarchical design. For example, you cannot create one netlist file for the two submodules E and C, while A and B are in different netlists, because E and C are in different branches of the design. You can have E and C separate with individual netlists for A, B, C, and E, or have E and C grouped in one netlist for the whole tree under the top-level design A.

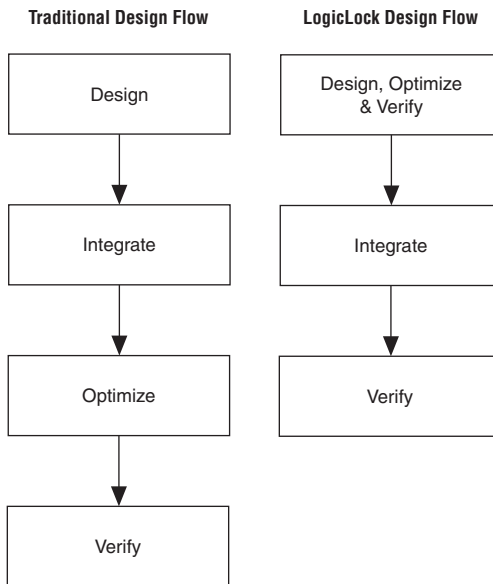
## Block-Based Design with the Quartus II LogicLock Methodology

You can use the LogicLock design methodology in the Quartus II software to perform block-based hierarchical compilation. Using the LogicLock design flow, you can design and optimize each module independently, integrate all optimized modules into a top-level design, and then verify the system. Incorporating each module into the top-level design does not affect the performance of the lower-level modules, as long as each module has registered inputs and outputs.

Having different netlist files for different submodules in the design means that each section is independent of the others. When synthesizing the entire project, only the portions of a design that have been updated must be re-synthesized when you compile the design. You can make changes, optimize, and re-synthesize your section of a design without affecting other sections. Using the LogicLock design methodology, you can place the logic in each netlist file into a fixed or floating region in an Altera device. You can then maintain the placement (and optional routing), thus maintaining the performance of your blocks in the Altera device.

Figure 1–2 compares the traditional design flow with the LogicLock design flow.

**Figure 1–2. Comparison of Traditional Design Flow with Quartus II LogicLock Design Flow**





For more information on using the LogicLock feature in the Quartus II software, see the *LogicLock Design Methodology* chapter in Volume 2 of the *Quartus II Handbook*.

The rest of this chapter assumes that you are familiar with the basic LogicLock features and methodology.

## Preserving Timing Results with the LogicLock Flow

You can use the LogicLock flow to preserve logic placement in an Altera device. Altera recommends using an atom netlist to preserve the node names in sub-blocks of your design. An atom netlist contains design information that fully describes the module's logic in terms of the device architecture. In the atom netlist, the nodes are fixed as Altera primitives and the node names do not change if the atom netlist does not change. If a node name does change, any placement information associated with that node (such as LogicLock assignments made when back-annotating a region) is invalid and is ignored by the Compiler.

If all the netlists are contained in one Quartus II project, use the LogicLock flow to back-annotate the logic in each region. If a design region changes, only the netlist associated with the changed region is affected. When you place-and-route the design with the Quartus II software, you need only re-fit the LogicLock region associated with the changed netlist file.



Altera recommends that you turn on the **Prevent further netlist optimization** option when back-annotating a region with **Synthesis Netlist Optimizations** and/or **Physical Synthesis Optimization** options turned on. This sets the **Netlist Optimizations** option to **Never Allow** for all nodes in the region, avoiding the possibility of a node name change in the top-level design when the region is imported into the top-level design.

You may need to remove previously back-annotated assignments for a modified block because the node names may be different in the newly synthesized version. When you recompile with one new netlist file, the placement and assignments for the unchanged netlist files assigned to different LogicLock regions are not affected. Therefore, you can make changes to a piece of code that exists in an independent block and not interfere with another designer's changes, even if all the blocks are integrated into the same top-level design.

With the LogicLock design methodology, separate pieces of a design can evolve from development to testing without affecting other areas of a design.

## Preserving Routing

LogicLock regions not only allow you to preserve the placement of logic from one compilation to the next, but also allow you to retain the routing inside the LogicLock regions. You can back-annotate and export the routing of a sub-module and then import it into a top-level project. This feature allows you to specify the exact location of the submodule in the device as well as which routing resources the Quartus II PowerFit™ Fitter should use during compilation.

You can only import back-annotated routing if exactly one instance of the imported region exists in the top level of the design. If more than one instance exists, the routing constraint is ignored and the LogicLock region is imported without back-annotation of routing. The routing constraint cannot be applied to multiple instances in different parts of the device because routing channels from one part of the device may not be exactly the same in another area of the device.



For more information on back-annotating routing, see the *LogicLock Design Methodology* chapter in Volume 2 of the *Quartus II Handbook*.

## Design Partitioning & Creating Multiple Netlist Files

When using a block-based design methodology, you typically create separate netlists for separate design modules. Partitioning your design in this way is easiest for team-based methodologies, and supports design reuse. In addition, to take advantage of the LogicLock design methodology when synthesizing a design using the Quartus II software, you must create an atom netlist for each design module before you lock down any nodes into LogicLock regions.

It is important to consider how your design is partitioned into sub-blocks that will be separate netlists in your block-based methodology. Altera recommends using registered boundaries for all modules. This helps ensure that the timing between hierarchical blocks does not become the critical timing path in the design once the blocks are assembled.



See the *Design Recommendations for Altera Devices* chapter in Volume 1 of the *Quartus II Handbook* for more design partitioning guidelines.

Certain third-party synthesis tools allow you to create different netlist files for different sections of a design hierarchy. To ensure the proper functioning of the synthesis tool, you can only create separate netlist files for modules, entities, or existing netlist files. In addition, each module or entity should have its own design file. If two different modules are in the same design file but are defined as being part of different regions, it is difficult to maintain incremental synthesis. This is because both regions would have to be recompiled when you change one of the modules or entities.

Alternately, you can create a black box for each submodule in the file that instantiates it. Creating a black box means that you instantiate the submodule, providing a component declaration in VHDL or a dummy module declaration in Verilog HDL, but you do not provide the actual design or logic that forms that submodule. The submodule has its own netlist file created in a separate synthesis project. Essentially, you instantiate a wrapper for the submodule netlist in any higher module that instantiates it. Some synthesis tools have attributes you can use to tell the synthesis tool that this module is intended to be empty.



See the appropriate chapter in the *Synthesis* section in Volume 1 of the *Quartus II Handbook* for details on your synthesis tool's support for creating multiple netlist files to be used with the LogicLock design methodology, and for more information on creating black boxes.

If you synthesize the design in the Quartus II software to be a VHDL Design File (.vhd), Verilog Design File (.v), Text Design File (.tdf), or a Block Design File (.bdf), you must also create an atom netlist to establish fixed nodes and node names when using the LogicLock design methodology. Turn on the **Save a node-level netlist into a persistent source file (Verilog Quartus Mapping File)** option on the **Compilation Process** page in the **Settings** dialog box (Assignments menu). This option saves your final results as an atom-based netlist in VQM format. By default, the Quartus II software places the VQM in the **atom\_netlists** directory under the current project directory. To create a different VQM with different Quartus II settings, change the file name setting on the **Compilation Process** page in the **Settings** dialog box (Assignments menu).



If you are using an atom netlist from a third-party synthesis tool and the design has black-boxed library of parameterized modules (LPM) functions or Altera megafunctions, you must generate a separate Quartus II VQM for the black-boxed modules.

## Conclusion

Hierarchical design methodologies can improve the efficiency of your design process, providing better design reuse opportunities and less integration problems when working in a team environment. Following the guidelines in this chapter can help you achieve good results with these methodologies.

