

# Creating Hierarchy in VHDL-Based High Density FPGA Design

Carol A. Fields  
Xilinx Inc.—  
2100 Logic Drive,  
San Jose CA 95124

## Abstract

As the density and complexity of FPGA-based designs has increased to 10,000 gates and beyond, the use of high-level design languages (VHDLs) is rapidly supplanting schematic entry as the preferred design entry format. However, to obtain the best results, the hierarchical design techniques already familiar to schematic users can be even more critical in an VHDL-based design. Furthermore, the choice of partition size can be critical to meeting capacity and performance goals, as demonstrated by the implementation of a 15,000 gate design.

## Introduction

In today's highly competitive market, system designers are faced with the conflicting challenges of greater system complexity and the need for short, efficient design cycles. These complexity and time-to-market pressures continue to reshape the art of designing electronic systems, and have led to the emergence and growing acceptance of top-down design methodologies and logic synthesis tools. With top-down design, engineers start work at a higher-level of abstraction than with traditional, gate-level design techniques. The designer manipulates logical or functional abstractions, and uses logic synthesis tools to produce the gate-level implementation. Over the past decade, top-down designs has become increasingly popular for the design of gate array and custom cell devices. As the density and complexity of FPGA-based designs has

increased to 10,000 gates and beyond, users are now employing the same techniques and similar tools for FPGA design. The use of logic synthesis for FPGA design will continue to accelerate as ever-larger FPGA devices are introduced.

The use of high-level hardware design languages, such as VHDL, allows designers to create and manage larger designs. As a result, there may be a temptation to disregard the importance of the hierarchical structure. Current design methodologies tend to target ASIC devices. 10,000 to 20,000 gates ASIC devices can often be compiled disregarding the hierarchical structure. This temptation to disregard the importance of designing the hierarchy should be avoided. Proper use of hierarchy in high density FPGA designing is critical in achieving desired device utilization.

The benefits of hierarchical design basically remain unchanged regardless of the design entry method; the use of hierarchical design techniques adds structure to the design process, ease debugging, allows for 'mixed-mode' design (wherein different design entry methods can be used for different portions of the design), provides a mechanism for dividing the design task among members of the design team, and facilitates the creation of libraries of reusable functions that, in turn, ease evolutionary product development.

Due to the basic functionality and interaction of FPGA synthesis and implementation tools, the actual size of each hierarchical module - in terms of the number of resulting logic blocks used in the target FPGA architecture - can have significant effect on the efficiency

and performance of the resulting circuit. With current tools, synthesizing a design as one large, flattened module can result in designs that are very difficult to route. The placement algorithms in FPGA 'place and route' programs typically are based on a cost function that attempts to minimize the total net length of the resulting block interconnections. As a result, interconnected logic blocks are placed as close together as possible. With very large designs, this strategy can reach the point of diminishing returns, in that the logic condenses into one region of the FPGA device causing local routing congestion. On the other hand, the use of a multiplicity of very small modules can result in wasted logic capacity; since each module is synthesized separately, logic functions are not optimized across module boundaries.

Empirical evidence suggests that, given the current state-of-the-art of FPGA synthesis and implementation tools, partitioning a large design into modules in the 3,000 to 5,000 gate range, and employing floorplanning techniques to govern the relative placement of those modules, results in the most-effective combination of FPGA device capacity and performance. In this manner, the logic is spread more evenly throughout the device, but related logic within a given module is still placed in an optimal topology for meeting performance requirements.

The benefits of such an approach include the following:

- Since each module is reasonably large, gate utilization is not overly diminished by the inability to optimize logic across module boundaries. Of course, careful partitioning of the modules, keeping related logic functions in a common module, further minimizes any deleterious effects caused by not optimizing across module boundaries. For example, if a design contains several small 4-bit incrementors, resource sharing would occur if these incrementors are in the same VHDL 'process'. If they are not in the same process and they were implemented in

gates they could be combined by the optimizer further reducing the gate utilization.

- The design's routability is improved by grouping the modules and specifying their floorplan according to the design's hierarchy and data flow. In this process, the designer has effectively added structure to the design and has passed this information onto the placement and routing tools. Floorplanning these modules into regions of the device evenly spreads-out the logic.
- Routing times are reduced; since modules are constrained into regions of the device, the automatic placement and routing tools has less area to evaluate. The designer has done some of the work by specifying the structure of the design to the placement and routing tools.
- Logic can be added or changed easily, since changes to one module can be made without effecting the placement and routing of other modules by using the re-entrant place and route programs.

Making the design easier to debug. The design's modules are isolated into a region of the device. The contents of the modules and the location are defined by the designer.

### **A Design Example**

The design methodology described in this paper was applied to several difficult to route 5,000 - 20,000-gate designs. The design example used to illustrate this technique is a transceiver receiver circuit for a telecommunication product that was targeted for an XC4025 FPGA device (-5 speed grade, 299-pin PGA package). This design was synthesized with the Synopsys FPGA Compiler using three different design methodologies. First, the design was compiled as one flat module. Second, it was compiled using the design's original hierarchical structure. Third (and recommended), it was compiled in 6 mid-size modules. The Xilinx Floorplanner was used to define the location of the modules. The Xilinx placement and routing tool was used to implement the design into the XC4025pg299-

5 device and to evaluate the utilization and timing results.

This called TOP contained a lower level module called CORE as shown in Figure 1. The "CORE" level of this design consist of two large modules, R0 and X0, and two smaller ones, UP0 and DD0. The two larger modules consist of over 30 sub-hierarchical modules. The sizes of these modules range from 4 to 591 configurable logic blocks (CLBs). Note: The names of the modules have been re-named in order to protect the confidentiality of the design

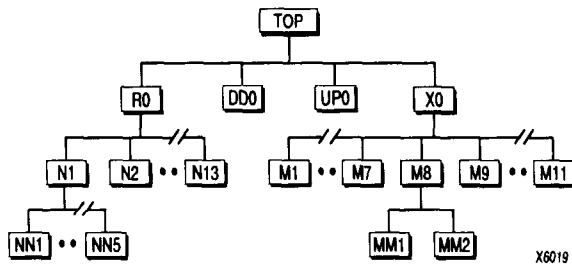


Figure 1 Original Hierarchy of Top Design

### Compiling the Flatten Design

The design was initially compiled as one module using the Synopsys FPGA Compiler's "compile -ungroup\_all" command on the CORE level. This design was unroutable, even though it only utilized 737 out of 1024 (71%) in the 1,024 CLBs available in the XC4025 FPGA. The logic of a large design compiled as one flat module often concentrates into one region of the device, creating a design that is difficult or impossible to route, as illustrated in Figure 2.

Next, the design was compiled using the design's existing hierarchy containing 5 levels of hierarchy and over 30 lower level modules. The logic block utilization increased from 737 CLBs to 1024 CLBS, a 29% increase in gates.

PPR was run on the design with and without floorplanning the X-BLOX RPMs. The design that was not floorplanned had ~8,000 unrouted nets. Once again the logic concentrated into one region of the device,

causing local routing congestion similar to the design shown in Figure 2.

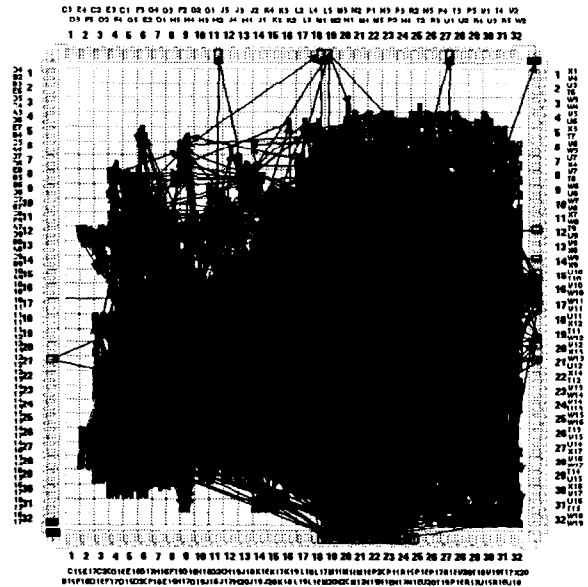


Figure 2 Ratsnets of Design Example Compiled Flat Compiling Using the Existing Hierarchy

Next, the design was compiled using the design's existing hierarchy containing 5 levels of hierarchy and over 30 lower level modules. The logic block utilization increased from 737 CLBs to 1024 CLBS, a 29% increase in gates.

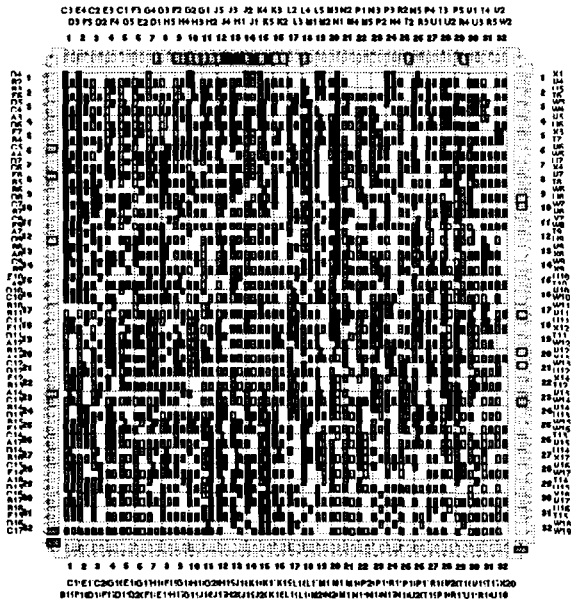
PPR was run on the design with and without floorplanning the X-BLOX RPMs. The design that was not floorplanned had ~8,000 unrouted nets. Once again the logic concentrated into one region of the device, causing local routing congestion similar to the design shown in Figure 2.

In order to route the design using the X-BLOX DesignWare library, the RPMs were floorplanned in a manner that away as to forced the logic to spread out in the device. The placed and routed design is shown in Figure 3. Overconstraining the design or poor floorplanning can make the design unroutable.

This design methodology allows PPR to place the unconstrained cells anywhere in the

device, making critical paths difficult to debug. Any design changes then would require an entirely new placement and routing, and, possibly, further floorplanning. If the design is difficult to route, a design change may cause the design to be unroutable.

An 8 Mhz internal clock speed was required for this design. Using the existing design's hierarchy with X-BLOX, the longest constrained clock-to-setup delay, 143.5 ns, exceeded the requirement of 125 ns. All constrained pad-to-clock met the requirements as shown in Table 1.



**Figure 3** Placement Cells (without Ratsnets) for Top Design Compiled Using the Original Hierarchy with the RPMs Floorplanned

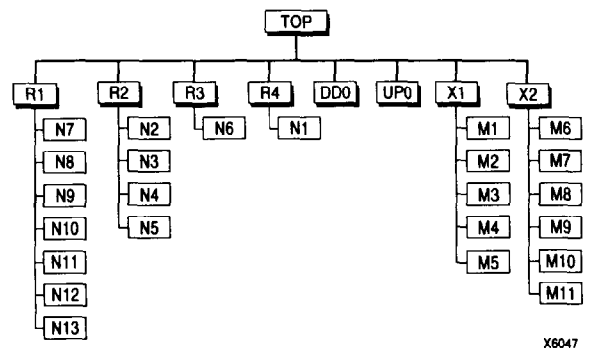
### Re-Grouping the Hierarchy

The recommended methodology for a large VHDL design is to re-group the design's hierarchy into mid-size modules. The original design hierarchy consists of 4 major blocks at the "CORE" level. The estimated CLB utilization was determined using the Synopsys FPGA Compiler's "report\_fpga" command. The block "R0" has an estimate of 591 CLBs, "X0" has 342 CLBs, UP0 has 25

CLBs and DD0 has 4 CLBs. Since the ideal module size is around 100 - 200 CLBs, this design was 're-grouped' to create a better hierarchical structure for the placement and routing tools.

The design's original hierarchy is shown in Figure 1. "R0" was separated into 4 modules and "X0" into 2 modules as shown in Figure 4.

The new grouping of the original modules was based on the sizes of the modules and the modules' interconnections with surrounding modules. An ideal grouping of modules will reduce the gate count and reduce the numbers of nets routed between the top level modules. The ideal module size is between 150 - 250 CLBs. The modules are grouped into new modules as shown in Figure 4.



**Figure 4** New Design Hierarchy

The two smaller modules, UP0 and DD0, were not combined with any other modules since these modules have an equal amount of interconnects with all of the new modules.

The Synopsys "group" command was used to define the new hierarchy. For example the module R0 was regrouped into four smaller module R1 - R4 using the following commands:

```
current_instance = R0
group {N7,N8, N9, N10, N11, N12, N13} -
design_name R1 -cell_name R1
group {N2, N3, N4, N5} -design_name R2 -
cell_name R2
group {N6} -design_name R3 -cell_name R3
group {N1} -design_name R4 -cell_name R4
```

Each group was then compiled individually using the "compile -ungroup\_all" command. A new script file was created that defined the new hierarchical groups, compiled the new groups, and created the XNF file for the CORE level. The lowest level modules were compiled before running this script and saved into a db file (e.g., N1.db). The script for the top level module reads in the top level, reads in the CORE level, assigns the I/Os, and writes out the design to an XNF file, top.sxnf.

A capacity reduction of 9 CLBs (30 packed CLBs) was achieved from compiling and flattening larger groups of logic together. An additional reduction of 67 CLBs (115 packed CLBs) was achieved when the Synopsys DesignWare modules were used in place of the small bit-width RPMs. (The 46 RPMs where 4-6 bits wide).

### Floorplanning the Modules into Regions

Next, the modules were constrained into regions of the device as shown in Figure 5 using the Xilinx Floorplanner as shown in Figure 6. Each region must be large enough to fit the module and provides the placement and routing tools room to route the module. The height of the regions must be tall enough to accommodate the tallest structure in the module. For example, an 8-bit adder would need the region to be at least 5 CLBs tall. The locations of the regions were selected based on data flow.

The two smaller modules UPO and DD0 were not constrained to allow the placement tools to determine the best location.

A constraint file for the placement and routing tools (PPR) was created specifying the regions which each module is constrained by using the Xilinx Floorplanner.

This design was then placed and routed using the constraints file and the timing specification passed from the Synopsys FPGA Compiler.

```
ppr top placer_effort=4 router_effort=3
cstfile=top_des
```

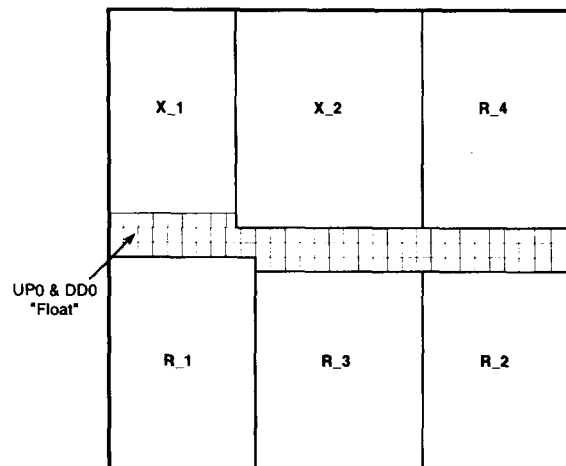


Figure 5 Floorplanning Modules into Areas

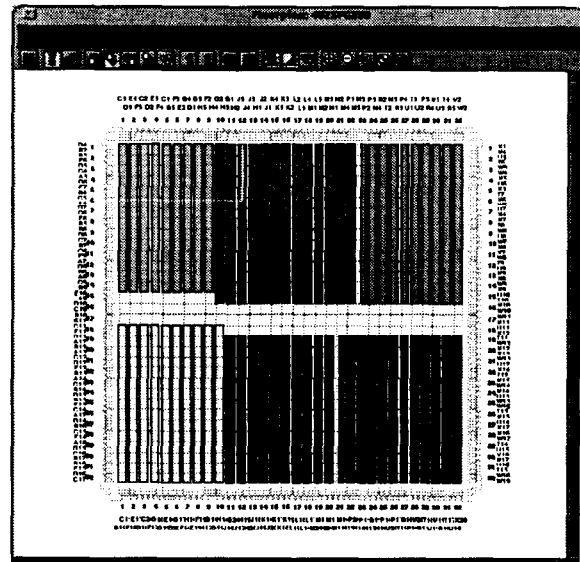
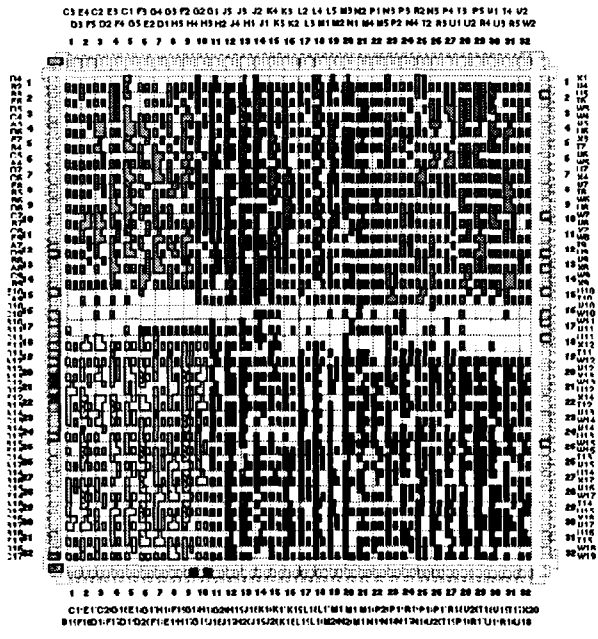


Figure 6 Floorplanning Modules into Areas

The placement of the resulting design is shown in Figure 7. Constraining the CLBs into regions of the device assists PPR in spreading the logic evenly throughout the device.



**Figure 7** Placement and Routing of New Design Hierarchy with Region Constraints

**Debugging a Design**

It is common to connect internal signals to unused I/Os in order to debug a design. If the I/O pins are constrained, design changes are

generally more difficult. The Xilinx Design Editor's (XDE) "Defineprobe" command can be used to specify an unused IOB as a probe point. The "Assignprobe" command is used to route an internal net to a probe point.

Two probe points were defined for both the design with the original hierarchical structure and the design with the re-defined hierarchical structure. In the design with the original hierarchy, the probe points were unroutable. However, in the design with the re-defined hierarchy, the probe points were easy to route, since this design contained unused logic in the center of the device.

**Design Methodology Comparison**

The flattened design used the least amount of resources. However, this design was packed so densely that it was unroutable. In addition, a flat design tends to be difficult to floorplan since the hierarchical structure is lost.

Using the existing design hierarchy required floorplanned to assist PPR in routing the design. This design utilized 100% of a XC4025. The longest constrained clock-to-setup delay was 143.2 ns and the longest constrained pad-to-clock delay was 100.1 ns. A timespec of 100 ns was specified when the design was placed and routed. However, the actual time requirement was 125 ns. 'Place and route' execution time approached 16 hours. Any small changes would require that the design be re-synthesized and re-compiled, resulting in another 16 hours to place and route.

Re-grouping the design's hierarchy did not require that individual cells be Floorplanned, but used the re-defined hierarchy to distribute the logic in the device. The design utilized 93% of a XC4025. The longest constrained clock-to-setup delay was 113.5 ns and the longest constrained pad-to-clock was 112.7 ns. This design did meet the system time requirements. 'Place and route execution time was reduced to 8 hours. Isolating the modules into regions reduces the run time, since less of the device has to be considered. A small design changed made

using this design methodology only requires that the changed module be re-synthesized and re-placed and routed, significantly reducing the placement and routing time. Iterative design changes are easier to make. Logic can be added to unused portions of the device. For example, in Figure 7, additional logic can be added to the center of the device.

### **Efficient VHDL Coding**

Efficient FPGA design starts at the VHDL code, before the design is placed and routed. No amount of re-design of the hierarchy can compensate for a poorly written VHDL code. Like designs entered using schematic capture tools, highly structured, synchronous designs will route easier and perform better than unstructured, asynchronous designs. Devices with limited routing resources require that the designer consider the data flow of the designs during coding. Utilization and system speed can also be improved by using the system features of the FPGA architecture. VHDL tends to abstract the design process, causing designers to lose sight of the implementation of the device. Using the correct construct will create a better implementation. A good understanding of the synthesis tool's capability will further improve the design's utilization and performance.

### **Summary**

Synthesis tools provide VHDL users with the capability to reduce logic when modules are optimized together. Flattening large designs often creates unroutable designs. Retaining an existing design hierarchy with a multiplicity of very small modules can result in wasted logic capacity and can often lead to a design that does not fit in the target device. Re-structuring the design's hierarchy into mid-size modules (i.e., 100 - 200 CLBs) not only reduces the area utilization, but also allows the user to make small changes to the design and to easily locate logic for debugging of critical paths. This also reduces place and route execution times.

This methodology was tested on six additional designs. Half of the designs were

implemented using the XC4013 devices and the other half using XC4025. All six designs exhibited similar results. Each of these designs did not route prior to regrouping the design's hierarchy. In all cases the system speed doubled, design changes were easier to make and execution time was significantly reduced. In one design there was significant utilization improvement to allow the designer to add additional functionality to the device. Additional studies were performed on XC4013 and XC4025 designs which did not require re-structuring of the design's hierarchy. These designs tend to have a significant amount of structure and module boundaries which were registered.

**Table 1. Comparison of Design Methodologies**

<b>Design Methodology XC4025pg299-5 PPR V5.1.0</b>	<b>Packed CLBs</b>	<b>RPMs</b>	<b>Flip- flops</b>	<b>Clock ToSetup Rising Edge</b>	<b>Pad ToSetup</b>	<b>PPR Run Time (CPU Time)</b>	
Flat Design (no X-BLOX)	619 60%	0	958 46%	n/a*	n/a*	n/a*	n/a*
Original Design Hierarchy; no Floorplan- ning	745 72%	50	958 46%	n/a*	n/a*	n/a*	n/a*
Original Design Hierarchy; with Floorplanning	745 72%	50	958 46%	143.2 ns	100.1 ns	Partition Placement Routing Total	01:13:41 02:05:16 12:53:22 16:14:36
Re-Group Design Hierarchy with X-BLOX	715 69%	46	958 46%	106.7 ns	108.8 ns	Partition Placement Routing Total	01:05:42 01:34:49 08:07:08 10:49:37
Re-Group Design Hierarchy without X-BLOX	630 61%	0	958 46%	113.4 ns	107.6 ns	Partition Placement Routing Total	01:02:10 05:39:46 04:29:02 11:12:55

**Reference**

- Xilinx - Xilinx Synopsys Interface Guide (1994)
- Synopsys - VHDL Compiler Reference Manual
- Synopsys - Solv-it
- Xilinx - Floorplanner's User Guide
- Xilinx - High Density Application Note