

Introduction

This application note explains what floorplanning is, when it should be used, and how it is done with respect to ORCA FPGA/FPSC designs. This document is divided into four major sections:

- Floorplanning definition, logical and physical
- When to floorplan, general versus specific reasons
- How to floorplan: grouping constraints, examples, and Floorplanner GUI presentation
- Special considerations, large and special groupings

Supported Architectures

Floorplanning can be done on Series 3 and Series 4 ORCA FPGA architectures, including any FPSC device based on these architectures.

Related Documentation

Designers are encouraged to reference the following software documentation for Floorplanner GUI operation, simple examples, and syntax usage.

- The Floorplanner GUI in-software Help manual.
- The *ORCA Foundry User's Guide*:
 - Chapter 3: Optimizing Mapped Designs - Map pgroup/ugroup support
 - Chapter 4: PGROUP physical preference
 - Appendix C: Properties - COMP, LOC, PBBOX, PDELAY, PFULOCK, PGROUP, PIOCOUNT, PIODIRECTION, PIOCLOC, PIOPGROUP, PLOC, PRBBOX, PLOC, PREGION, PRLOC, and UGROUP

Floorplanning Definition

Floorplanning a digital logic design for implementation in a programmable device involves the physical or logical partitioning of the design modules which results in a change in the physical placement or implementation of the modules. In other words, floorplanning is the grouping of design modules in a certain way to improve the performance of a design.

With ORCA FPGAs, floorplanning is an optional methodology to help designers improve the performance and density of a fully, automatically placed and routed design. Floorplanning is particularly useful in structured designs and data path logic. Design floorplanning is very powerful and provides a combination of automation and user control for design reuse and modular, hierarchical, and incremental design flows.

Complex FPGA Design Management

ORCA FPGAs can implement large system designs that contain millions of gates, hundreds of thousands of embedded memory bits, and intellectual property (IP) components. Design teams often work on large designs. The design complexity requires EDA tools to manage and optimize the design. Large design management is difficult, but performance optimization is even more difficult. Optimization requires many design iterations when adding or modifying components. Complex, large system designs require the following:

- The use of modular, hierarchical, or incremental design methods
- Software that makes management and optimization easier
- Use of IP blocks
- Reuse of previously optimized design modules

By controlling the placement of specified logic modules, ORCA design floorplanning methodologies help designers meet the requirements of large system design.

Floorplanning Design Flow

In both traditional and floorplanning FPGA design flows, the designer divides the system into modules. The modules can be individual circuits, parts of circuits, or parts of the design hierarchy. After module design and optimization, the designer integrates the modules into the system. Finally, the designer tests and optimizes the system.

In the traditional flow, the system may not meet performance requirements even if each module meets the requirements before integration. Even when timing requirements have been satisfied, changes to one module can affect the performance of others. Re-optimizing modules to meet system performance results in many design iterations.

ORCA floorplanning methodologies assist in the design, testing, and optimization of each individual module while retaining the optimized characteristics of the individual modules. Module integration into the system requires only system optimization between modules. The ORCA floorplanning methodologies provide additional flexibility by allowing the ORCA Foundry software to automatically place defined modules, or allowing the user to control the placement of specific modules, which provide performance preservation and optimization.

When to Floorplan

Floorplanning methodologies are intended to assist users who require some degree of handcrafting for their designs. The designer must understand both the details of the device architectures and the ways floorplanning can be used to refine a design. Successful floorplanning is very much an iterative process and it can take time to develop a floorplan that outperforms an automatic, software processed design. Because of the nature of floorplanning and its interaction with the automatic MAP and PAR software tools, several prerequisites are necessary in order to floorplan a design successfully.

- Detailed knowledge of the specifics of the target architecture and device
- Detailed knowledge of the specifics of the design being implemented
- A design that lends itself to floorplanning
- A willingness to iterate a floorplan to achieve the desired results
- Realistic performance and density goals

For ORCA FPGAs, the general rule-of-thumb is that floorplanning should be considered when the performance needed cannot be met and routing delays account for over 60% of the critical path delays. That is, interacting components are too far apart in the FPGA array to achieve short routing delays. This has shown to be a problem especially with large designs in high density FPGAs because of the possibilities of long distance routes. As programmable logic design densities continue to escalate beyond 100,000 gates, traditional design flow — design entry to synthesis to place and route — will sometimes not yield predictable, timely, and optimized results.

Note that the guidelines discussed above only apply to designs that have been routed by the software for several routing iterations. The default number of routing iterations via the ORCA Foundry Control Center (OFCC) is 15.

A note on delays: Path delays in programmable devices are made up of two parts: logical delays and routing delays. Logical delays in this context are delays through components, where an FPGA component is a programmable function unit (PFU), a programmable input/output (PIO), or an embedded function (i.e. a block RAM, PLL, or embedded FPSC ASIC). The routing delay is the delay of the interconnect between components. Figures 1 and 2 show delay examples from timing wizard report files (.twr).

Figure 1. Floorplanning May be Able to Help Bring These Registers Closer

Logical Details:	Cell type	Pin type	Cell name (clock net +/-)
Source:	FF	Q	ibuf/reg_init_start (from clk_ib+)
Destination:	FF	Data in	ibuf/sd/reg_new_state (to clk_ib +)
Delay:	8.062ns (18.2% logic, 81.8% route), 2 logic levels.		

Figure 2. Floorplanning is Not Needed Here Because the Routing is Efficient

Logical Details:	Cell type	Pin type	Cell name (clock net +/-)
Source:	FF	Q	mem_if_tx_address_8 (from clk_c +)
Destination:	FF	Data in	mem_if_tx_address_17 (to clk_c +)
Delay:	7.358ns (61.2% logic, 38.8% route), 4 logic levels.		

Floorplan to Improve Design Performance

Properly applied, design floorplanning not only preserves, but improves design performance. Floorplanning methodologies can be used to place modules, entities, or any group of logic to regions in a device's floorplan. Because floorplanning assignments can be hierarchical, designers can have more control over the placement and performance of modules and groups of modules.

In addition to hierarchical blocks, like grouping an entire VHDL entity or Verilog module, designers can floorplan individual nodes (e.g., instantiate a library macro for a function in the critical path and then group the macro). This technique is useful if the critical path spans multiple design blocks.

Note: Although floorplanning can increase performance, it may also degrade performance if it is not applied correctly within software limitations.

Floorplan to Preserve Module Performance

Floorplanning with design constraints maintains design performance by grouping the placement of nodes in a device, i.e., the relative placement of logic within a grouped region remains constant. The ORCA Foundry software then places the grouped region into the top-level design with these constraints. When placing logic in a region, the ORCA Foundry software does not preserve the routing information. This approach provides more flexibility when the software imports the region into the top-level design and helps fitting.

Floorplan for Design Reuse

ORCA floorplanning facilitates design reuse by its ability to reproduce the performance of a module designed in a different project. For frequently used modules, designers can create a library of verified designs that can be incorporated into other larger designs. The library only has to contain the VHDL or Verilog source code along with grouping attributes and some comments detailing useful information to the user, such as performance and size. With a parameterized module, in-code assignments can specify the module's size and grouping assignments.

Targeting the same device used in the original design likely achieves the best results, although other devices in the same family will likely work well. When using a different device in the same family, the exact placement of the region may not be possible. Similar performance, however, may be possible by moving or floating regions. A floating region groups the logic together and guides the ORCA Foundry software toward achieving a placement that meets the performance requirements of the module. A similar approach can also be taken if exact placement of a module is not applicable because of multiple instantiations of a module in a top-level design.

How to Floorplan a Design

Design Performance Enhancement Strategies

ORCA floorplanning methodologies improve the performance of designs that do not necessarily consist of individually optimized modules. The ability of specifying regions to group nodes together and provide relative placement enhances the usability of the ORCA Foundry place-and-route software tools. The design strategies for performance enhancement depend on the structure of a particular circuit. Strategies include:

- Defining regions based on design hierarchy if the hierarchy closely resembles the structure of the circuit. These designs typically consist of tightly integrated modules, where the logic for each module is self-contained and modules communicate through well-defined interfaces.
- Defining regions based on the critical path, if the critical path is long and spans multiple modules. Keeping the nodes in the critical path or the modules containing the critical path together may lead to improved performance.
- Defining regions based on connections by grouping nodes with high fan-outs and high fan-ins together to reduce delays in connections and wiring congestion in the device.

Note that designers may need to change existing design hierarchy and structure to make the design more amiable to floorplanning. This is especially applicable if modular hierarchy and structure was not considered at the beginning of design conception.

With the ORCA floorplanning methodologies, the user can choose to optimize modules either individually or after they have been integrated with the top-level design. The user can exercise varying amounts of control over the placement by using different types of regions. By using bounding boxes and location anchors selectively, the ORCA Foundry software can easily determine the best size and location for a region. Another approach is to optimize the top-level design without first optimizing the individual modules. This approach allows the ORCA Foundry software to place nodes within regions and move regions across the device. The user assigns modules to regions and then compiles the entire design. With this approach the user can place elements from different modules in a region.

Design Floorplanning Methodologies

There are several methods available to aid in the floorplanning of a logical design in an ORCA FPGA. This section illustrates these methods with examples of how to use the ORCA Foundry software tools to achieve performance goals. The three main floorplanning tools available include:

- The **PGROUP Physical Constraint** can be used as an attribute in VHDL and Verilog HDL source code or as a physical Preference in the .prf design constraint file. This can be used directly by the ORCA Foundry Placer software to bound and locate sections of a design for grouping in the FPGA array.
- The **UGROUP Logical Constraint** can be used as an attribute in VHDL and Verilog HDL source code to gather logical sections of a design for grouping in the FPGA array.
- The **Floorplanner Graphical User Interface (GUI)** can be used to interactively specify placement parameters in either the logical or physical domain for some of a design's modules (e.g. logic gates, registers, arithmetic functions) from one graphical user interface.

When to use PGROUP vs. UGROUP

UGROUPing differs from PGROUPing as follows:

- A PGROUP logical identifier, in EDIF, is prepended with text that describes the identifier's hierarchy.
- A UGROUP logical identifier, in EDIF, is not changed by prepending the hierarchy on the block instance identifier.

In other words, PGROUPing enforces strict hierarchical control while UGROUPing allows for a grouping of blocks in different hierarchies or a grouping of blocks with no hierarchy at all.

Also note that the PGROUP attribute can be placed on multiple instantiations of modules (e.g. VHDL generate statements) and each instantiation will have its own PGROUP. Using a UGROUP will not work in this case.

In Figures 3 and 4, the arrows represent control and data paths where there is interaction between different levels of hierarchy. The thick lined arrow represents the critical path where the design is failing to make performance.

Figure 3. PGROUP Same Hierarchy Example, PGROUP CONTROLLER

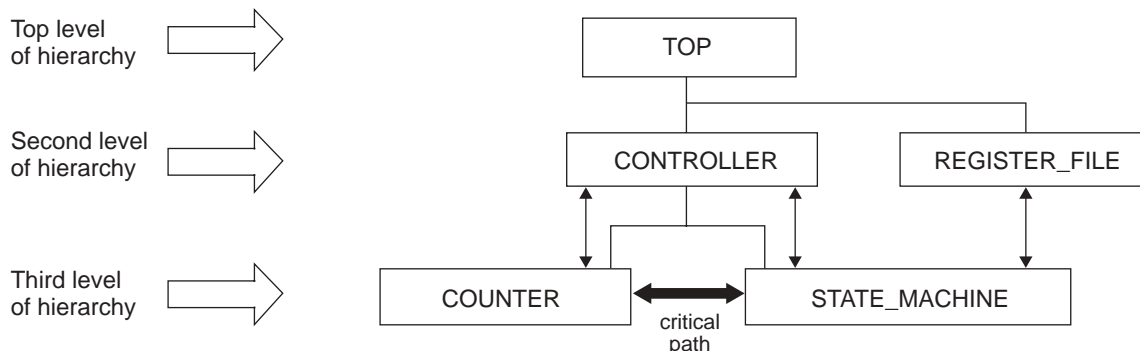


Figure 3 illustrates a design hierarchy where the failing paths are the connections between COUNTER and STATE_MACHINE design blocks. The easiest implementation for this example is to PGROUP the CONTROLLER, which is the module in which the COUNTER and STATE_MACHINE are instantiated within.

For example, if the following Synplify attribute is in the Verilog HDL file:

```
module CONTROLLER (<port_list>)/* synthesis pgroup="CONTROL_GROUP" */;
```

then the COUNTER and STATE_MACHINE will be grouped in the FPGA inside a boundary box. Now assume that the COUNTER is mapped into PFU_0 and PFU_1 and the STATE_MACHINE is mapped into PFU_2. The resulting preference generated by map in the .prf file will be:

```
PGROUP "TOP/CONTROLLER/CONTROL_GROUP"
COMP "PFU_0"
COMP "PFU_1"
COMP "PFU_2" ;
```

Notice the TOP/ hierarchy is prepended to the CONTROLLER PGROUP identifier.

Figure 4. UGROUP Different Hierarchy Example, UGROUP REGISTER_FILE and STATE MACHINE

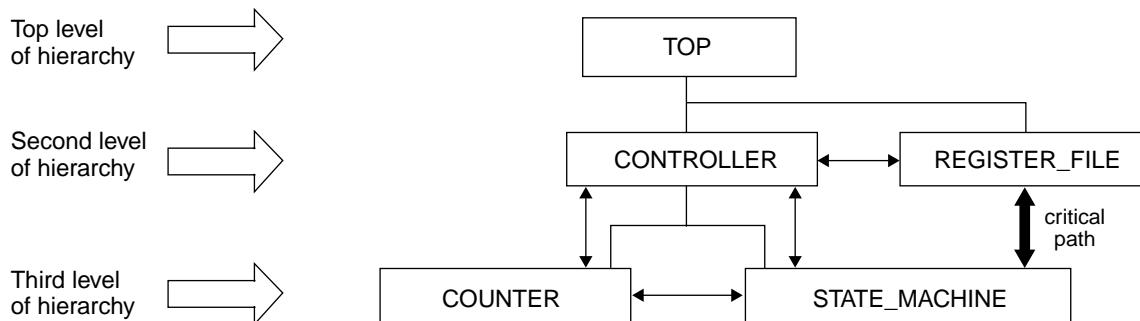


Figure 4 shows an example design hierarchy where the failing paths are the connections between REGISTER_FILE and STATE_MACHINE modules. The simplest thing to do here is to UGROUP the REGISTER_FILE and STATE_MACHINE together.

For example, if the following Synplify attributes are in the Verilog HDL file:

```
module REGISTER_FILE (<port_list>) /*synthesis ugroup="CRITICAL_GROUP" */;
```

and

```
module STATE_MACHINE (<port_list>) /*synthesis ugroup="CRITICAL_GROUP" */;
```

then the REGISTER_FILE and STATE_MACHINE will be grouped in the FPGA inside a default boundary box. Now assume that the REGISTER_FILE is mapped into PFU_4 and PFU_5 and the STATE_MACHINE is mapped into PFU_3. The resulting preference generated by map in the .prf file will be:

```
PGROUP "CRITICAL_GROUP"
  COMP "PFU_3"
  COMP "PFU_4"
  COMP "PFU_5" ;
```

Notice the TOP/ hierarchy is not appended to the PGROUP identifier CRITICAL_GROUP. Also notice that UGROUP attributes result in PGROUP preferences. There is no UGROUP preference.

If PGROUP attributes instead of UGROUP attributes had been used for Figure 4:

```
module REGISTER_FILE (<port_list>) /*synthesis pgroup="CRITICAL_GROUP" */;
```

and

```
module STATE_MACHINE (<port_list>) /*synthesis pgroup="CRITICAL_GROUP" */;
```

then the resulting preference generated by map in the .prf file would be:

```
PGROUP "TOP/CONTROLLER/STATE_MACHINE/CRITICAL_GROUP"
  COMP "PFU_3"

PGROUP "TOP/REGISTER_FILE/CRITICAL_GROUP"
  COMP "PFU_4"
  COMP "PFU_5" ;
```

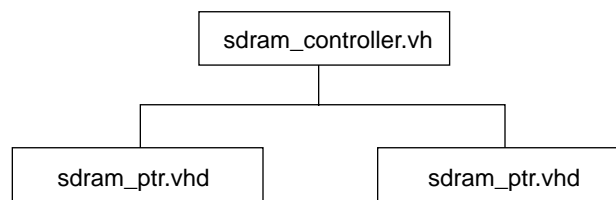
So, with PGROUP attributes, the STATE_MACHINE module would be grouped together in one bounding box and REGISTER_FILE module would be grouped together separately in another bounding box and the critical path shown in Figure 4 will not be optimized.

These examples do not utilize all the possible tools available for floorplanning. Please refer to chapter 4 of the *ORCA Foundry User's Guide*, PGROUP section for many small syntax examples.

PGROUP Example: SDRAM Controller State Machine

This example is a detailed, "real world" design with multiple state machines. The design serves as an SDRAM controller consisting of the three VHDL files in the /fp/sm/source directory included with the application note example package. Figure 5 shows the hierarchy of the design.

Figure 5. Original SDRAM Design Hierarchy



Without any floorplanning, the sdram_controller design was ran through the Synplicity Synplify 7.0.2 synthesizer tool to get an EDIF file. Then this EDIF file was ran through the ORCA Foundry 2001 map, place, route and trace tools, with default settings.

The preference file used (`/fp/sm/orca/sm.prf`) consists of just a simple block and frequency preference:

```
BLOCK RESETPATHS ;
FREQUENCY NET "sclk_c" 133 MHz ;
```

The results are as follows:

Table 1. Original SDRAM Design Performance

Target Device	Target Frequency (MHz)	Area (PFUs)	Frequency (MHz)
OR4E02-2 BM680	133MHz	175	113

Analyzing the performance of the `sdram_controller` design in the TRACE generated timing report file (`/fp/sm/orca/sm.twr`) shows that:

- Up to 64% of critical path delay is incurred by the routing delay:

```
(36.1% logic, 63.9% route), 3 logic levels
```

- 8 of the top 10 worst paths are sourced by `rd_eop` ports:

```
Logical Details: Cell type Cell name
Source:          Port      rd_eop_<#>
```

- 7 of the top 10 worst path destinations are state or state condition registers:

```
Logical Details: Cell type Cell name
Destination:    FF          term_active
Destination:    FF          term_active
Destination:    FF          state_4
Destination:    FF          state_8
Destination:    FF          state_5
Destination:    FF          state_4
Destination:    FF          terminate
```

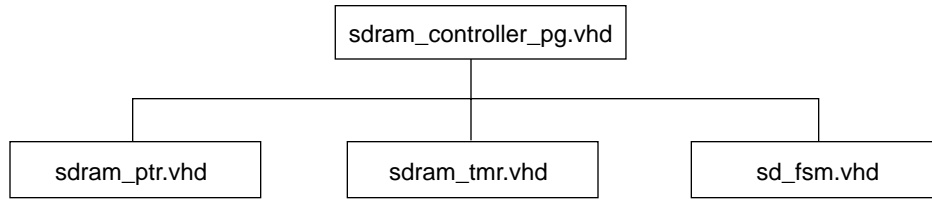
Looking at the SDRAM VHDL source file, we can see that there is minimal control on hierarchy, with 27 processes in the top level file, `sdram_controller.vhd`. Since PGROUP attributes cannot be directly attached to VHDL processes, there is a need to add hierarchy to the design so that floorplanning can be accomplished.

From the above observations, we can see that some of the design needs to be extracted from the large, top-level `sdram_controller.vhd` file, including:

- the main SDRAM controller interface state machine (`sd_fsm:process`)
- the new state generation state machine (`new_st:process`)
- the initial refresh process (`iniref:process`)

These processes are moved to a new file, `sd_fsm.vhd`. The `sd_fsm.vhd` file consists of the three removed processes and the new `sdram_controller_pg.vhd` is the new top-level file with a declaration and an instantiation of the `sd_fsm` entity. Figure 6 shows the new hierarchy of the design.

Figure 6. Modified SDRAM Design Hierarchy



Now that the processes are in their own hierarchy, they can be grouped together with the following attribute syntax:

```

attribute pgroup : string;
attribute pgroup of structure : architecture is "sd_fsm_pgroup";
  
```

These attributes are in the `sd_fsm.vhd` file in the architecture declaration, before the begin statement.

With this floorplanning, the newdesign was run through the Synplicity® Synplify® 7.0.2 synthesizer tool to produce an EDIF file. The pgroup attribute is passed into the EDIF as a property on the `sd_fsm` cell:

```

(cell sd_fsm ...
...
... (property pgroup (string "sd_fsm_pgroup")))
  
```

After mapping the design with the default settings of ORCA Foundry Map, a new preference file is created:

```

SCHEMATIC START ;
# map: version ORCA Foundry 2001 Production
PGROUP "fsm/sd_fsm_pgroup"
  COMP "PFU_119"
  COMP "PFU_126"
  COMP "PFU_181"
  COMP "PFU_182"
  COMP "PFU_183"
  COMP "PFU_184"
  COMP "PFU_185"
  COMP "PFU_205";
SCHEMATIC END ;
BLOCK RESETPATHS ;
FREQUENCY NET "sclk_c" 133.000000 MHz ;
  
```

The Map software automatically inserted the PGROUP preference in the newly created .prf file `\sm\orca\sdr_controller_4\Map_Rev_1\sm.prf`. The eight PFUs listed are the physical FPGA components the processes in `sd_fsm.vhd` were mapped to. The PGROUP preference directs the ORCA Foundry Placer program to place these eight PFUs in a square bounding box within the FPGA.

After running the netlist through the place, route and trace tools (default settings), the improved results are as follows:

Table 2. Improved SDRAM Design Performance

Target Device	Target Frequency (MHz)	Area (PFUs)	Frequency (MHz)
OR4E02-2 BM680	133MHz	177	133

The PGROUP floorplanning improved the placement of the design, thereby increasing its performance. With just a hierarchical change to the design and a single, strategic PGROUP attribute, we achieve a $(133 - 113) / 113 = 15\%$ frequency improvement. The area increase is small (2 PFUs) for this example. Area will generally increase when

hierarchy and floorplanning are applied. This is because hierarchy and grouping boundaries keep modules apart; without floorplanning, PFU logic resources can be shared across modules.

Floorplanner GUI Usage

Generally, the PGROUPs and UGROUPs are preferable to the Floorplanner GUI since they are easier to implement. For example, it is easier to type in a PGROUP attribute in the HDL code than to load the GUI with large netlists and find the desired block and perform add the PGROUP via mouse clicks. More importantly, the GUI does not allow the retention of floorplanning the way PGROUPing and UGROUPing does. Since the GUI does not back-annotate the grouping attributes into the HDL, the GUI operations have to be redone every time there is a new design iteration.

The Floorplanner GUI can be useful for

- Viewing elements in a graphical environment to see a design's logical hierarchy.
- Viewing existing PGROUPs and UGROUPs.
- Resizing regions and boundary boxes (BBOXes).
- Graphically placing regions, PGROUPs, and UGROUPs and then running map, place, route, and trace to see the effects. This is usually an iterative process before finding an optimal solution.

If the Floorplanner GUI is used, it is strongly recommended that after finding the optimal grouping with the GUI, grouping attributes (PGROUP and/or UGROUPS) should be inserted into the HDL source code to preserve module performance over design revisions.

Note that all references to the Floorplanner GUI tool in this document are specific to the ORCA Foundry 2001 Production software suite. Later versions of the ORCA Foundry software suite may be dramatically different in appearance and functionality. Please refer to the software documentation for Floorplanner GUI operation.

Special Floorplanning Considerations

Embedded Block RAM Placement

Block RAM placement can be done with simple LOCATE preferences. It is not always necessary to locate block RAMs. Do not use the PGROUPs, UGROUPs, or the Floorplanner GUI to group Block RAMs. Legal location site names and considerations are given in Appendix A.

PIO Grouping

There is a complete set of physical constraints on PGROUPing Programmable I/O (PIO) components. Please refer to chapter 4 of the *ORCA Foundry User's Guide*, Defining PIO Component Groups, for keyword explanations and syntax examples.

Large Module Grouping

It is strongly recommended that larger PGROUPs/UGROUPs (with many PFUs) be anchored and bounded by LOCATE and BBOX keywords. From the STATE_MACHINE example, we see that without anchoring the groups, the performance worsened compared to no floorplanning at all. The advanced example in the appendix shows very detailed anchoring of groups.

The BBOX should be strategically shaped and sized according to the module to be placed inside the BBOX. If the BBOX shape and size is not specified, the default BBOX size will be a square that is as small as possible. This is not the optimal BBOX for typical modules. The designer should shape the design with the datapath in mind and size the BBOX to be larger than needed so that the ORCA Foundry placer program can have more flexibility in placing PFUs inside the BBOX.

Carry Chains and Bus Grouping

Carry chains (used by ripple arithmetic functions like adders, counters, and multipliers) and logic modules connected by busses can easily be floorplanned inappropriately by a designer that is not aware of the internal routing

resources available to optimize these carry chain and bus routes. As we saw from the multiplier example, certain groupings can reduce the performance of a design compared to no floorplanning at all. Great care should be used when floorplanning designs that use carry chains or busses so that these routes fall in optimal locations for optimal performance.

Broken Carry Chain Example

A 9-bit adder that is PGROUPed with no relative placement on the adder. PFUs may give worse performance because the adder carry-chain is broken.

SLICs in Groups

Supplemental Logic and Interconnect Cells (SLICs) are automatically removed from PGROUPs and UGROUPs by the ORCA Foundry software if they are not relatively placed. This is because SLICs are used by the tools for interconnects that are not foreseeable by designers. If SLIC placement has to be controlled for a design, the designer will need to instantiate and locate the SLICs in their preference or HDL files. It is recommended to allow the ORCA Foundry software to automatically place SLICs.

Summary

This application note defined floorplanning, discussed when it should be used, and detailed how floorplanning is done with respect to ORCA FPGA/FPSC designs. Examples were used to illustrate and compare the different tools available to the designer for floorplanning.

Important items discussed:

- Floorplanning can improve timing for targeted critical paths.
- Improper floorplanning can make timing worse.
- Correct floorplanning can increase PFU count slightly.
- Completed floorplanning should be annotated into the HDL.
- Floorplanning enhances the reusability of designs by keeping placement directives in the HDL.
- Use PGROUPs for physical grouping in the same hierarchy.
- Use UGROUPs for logical grouping across hierarchies.

With ORCA FPGAs, floorplanning is an *optional* methodology to help designers improve performance and density of a fully, automatically placed and routed design. Floorplanning is particularly useful on structured designs and data path logic.

ORCA design floorplanning is very powerful and provides a combination of automation and user control for design reuse and modular, hierarchical, and incremental design flows. It advances the design of large systems on FPGAs. It provides the designer with capabilities in the management and optimization of large systems. Its multifaceted capabilities result in shortened design cycles and faster time to market.

Appendix A: Block RAM Site Naming and Location Conventions

In the ORCA Series 4 FPGA, block RAM sites are organized along the top and the bottom of the device. The number of sites is dependent on the size of the FPGA being used. Table 3 provides the number of block RAM sites for the FPGAs in the Series 4 family.

Table 3. Number of EBR per FPGA

Device	EBRs (Top/Bottom)	EBR kbits
OR4E2	8 (4/4)	74
OR4E4	12 (6/6)	111
OR4E6	16 (8/8)	147

Block RAM Sites

The EBR site names on the top edge of the ORCA Series 4 FPGA/FPSC device are as follows (left to right):

```
RAM1024_0-RAM512_1-RAM1024_2-RAM512_3- . . .
```

The EBR site names on the bottom edge of the ORCA Series 4 FPGA/FPSC device are as follows (left to right):

```
RAM1024_32-RAM512_33-RAM1024_34-RAM512_35- . . .
```

For example: an OR4E2 device will have eight sites available for embedded block RAM placement. Those site names and locations are as follows:

Top row (left to right):

```
RAM1024_0-RAM512_1-RAM1024_2-RAM512_3
```

Bottom row (left to right):

```
RAM1024_32-RAM512_33-RAM1024_34-RAM1024_35
```

Placement Rules

- A 512x18 can be located to any site (RAM1024_x or RAM512_x)
- A 1024x18 can be located in only the RAM1024_x sites. The 1024x18 block will, however, consume two sites: RAM1024_(x) and the adjacent site to the right, RAM512_(x + 1).

Legal and Illegal LOCATE Preference Examples

Quad-Port 1024x18 blocks:

- LOCATE COMP "U1_BR1024x18" SITE "RAM1024_0"; //legal placement
- LOCATE COMP "U1_BR1024x18" SITE "RAM512_1"; //illegal placement

Quad-Port 512x18 blocks:

- LOCATE COMP "U1_BR512x18" SITE "RAM1024_32"; //legal placement
- LOCATE COMP "U2_BR512x18" SITE "RAM512_33"; //legal placement

Two Quad-Port blocks (1024x18 and 512x18):

- LOCATE COMP "U1_BR1024x18" SITE "RAM1024_0"; //legal placement (2 sites)
- LOCATE COMP "U2_BR512x18" SITE "RAM512_1"; //illegal placement

The next available site for COMP U2_BR512 x 18 would be RAM1024_2:

- LOCATE COMP "U2_BR512x18" SITE "RAM1024_2"; //legal placement

Note: Locating RAM blocks is not required. The software tool will automatically place the blocks if they are not located.

Technical Support Assistance

Hotline: 1-800-LATTICE (Domestic)
1-408-826-6002 (International)
e-mail: techsupport@latticesemi.com