Floorplanning Methodology Guide

UG633 (v 12.1) May 3, 2010

Table of Contents

About this Manual

FPGA devices have grown considerably over the years. Engineers are taking advantage of larger FPGAs to implement more complex designs. The implementation tools have improved along with these complexities. On some designs, guidance from the designer can guide the implementation tools to a higher system clock frequency, shorter implementation run times, more consistency in timing, or, in some cases, all of these benefits together.

This guide covers some of the basics of floorplanning and presents two approaches to floorplanning that can help a design meet timing more consistently. This guide focuses on the floorplanning considerations and techniques of the Xilinx® software.

This document covers the following:

- Chapter 1, "Introduction to Floorplanning," which includes floorplanning and timing closure basics, and design considerations and techniques when floorplanning.
- Chapter 2, "The Floorplanning Flows," which covers the two recommended approaches to floorplanning: placement re-use, and hierarchical floorplanning.

Note: It is recommended that readers of this guide be familiar with the PlanAhead™ software to get the most out of this guide. Refer to the PlanAhead tutorials and the *PlanAhead User Guide* (UG632) for more information about the PlanAhead software.

Additional Resources

Perform one of the PlanAhead tutorials to learn about the PlanAhead functionality using a sample design:

http://www.xilinx.com/support/documentation/dt_planahead_planahead12-1_tutorials.htm

Refer to the *PlanAhead User Guide* (UG632) for information about specific PlanAhead functionality and more details on specific commands:

http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_1/PlanAhead_UserGuide.pdf

For general PlanAhead information, video demonstrations, and white papers, go to:

http://www.xilinx.com/planahead

Chapter 1

Introduction to Floorplanning

This chapter contains the following sections:

- Introduction
- Timing Closure Basics
- Floorplanning Basics
- Considerations
- Logic Synthesis Recommendations
- Increasing Consistency
- Using Clock Resources to Guide Floorplanning

Introduction

A good floorplanning methodology can improve performance and help the placed and routed design meet timing. Floorplanning is the process of choosing the best grouping and connectivity of logic in a design, and of manually placing blocks of logic in an FPGA, where the goal is to increase density, routability, or performance. The intent is to reduce route delays for selected logic by suggesting a better placement.

Floorplanning may be considered when a design does not meet timing consistently or when the design has never met timing. Floorplanning can range from the detailed approach, such as placing individual logic elements of a critical path to a specific site on the chip, to the more abstract approach, such as constraining levels of hierarchy to specific regions on the chip.

Some design teams may spend time floorplanning a design before the first iteration through place and route. Others may take a wait-and-see approach, electing to wait until a problem is identified before floorplanning.

The various strategies for floorplanning have benefits and tradeoffs, which are discussed in this guide.

Timing Closure Basics

Most engineers begin to floorplan when a design does not meet the setup timing constraints consistently. Floorplanning is introduced as a means to reduce path delays, leading to timing closure. During the implementation process, the implementation tools compare the delay of the logic and routing against the time allowed by the timing constraint, with some modifications for clock-to-clock skew and clock noise.

The tool reports how much time the paths beat timing constraints (i.e., met timing) or exceeded timing constraints (i.e., failed timing). Figure 1 shows an example timing report.

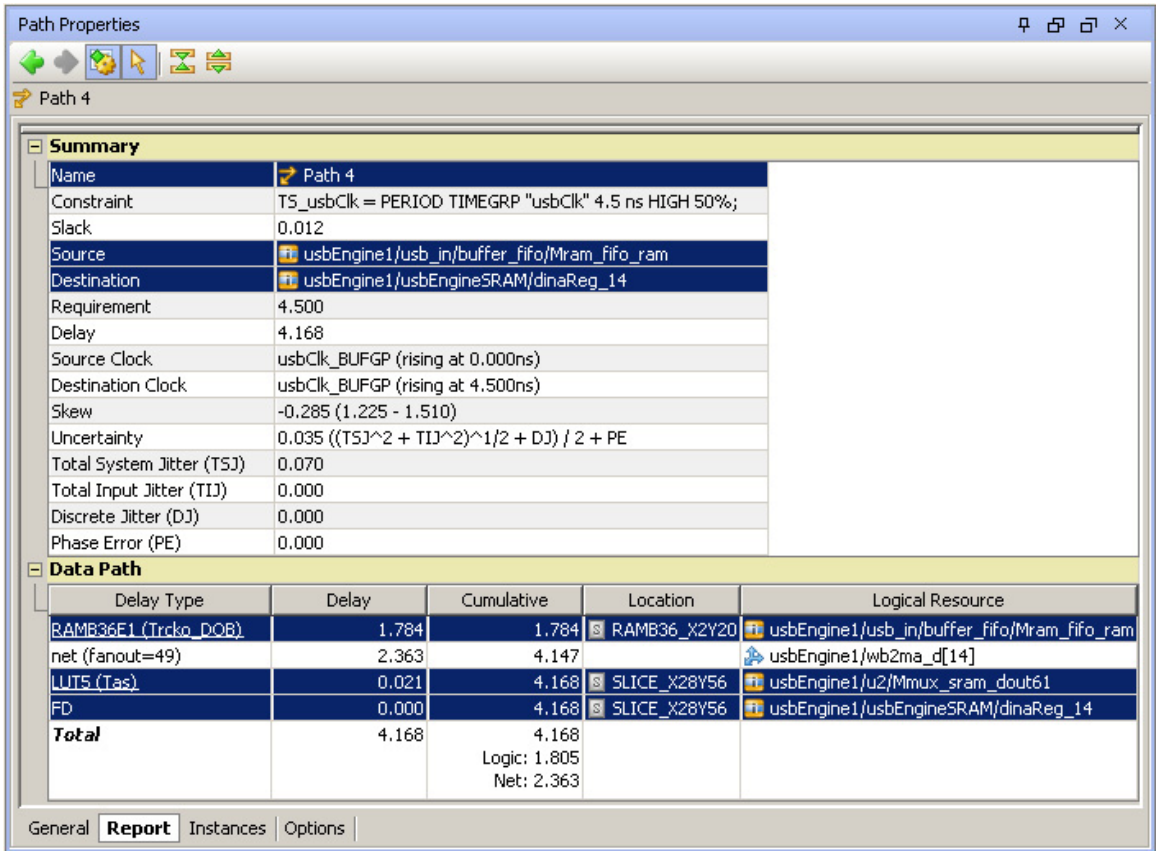

Figure 1: An Example Timing Report

A first step is to ensure the timing constraints are accurate. Determine whether the path is a multi-cycle path or a false path. Sections of some designs are not clocked every clock cycle or the paths may not be reached due to the control structure. The implementation tools cannot make this determination. These paths will be needlessly timed unless timing constraints mark this logic as multi-cycle paths or false paths. Many designs improve timing when the constraints are relaxed to match the design logic. See the *Xilinx Timing Constraints User Guide* (UG612) for a discussion of multi-cycle paths and false paths (http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_1/ug612.pdf).

The allowed time is modified by clock jitter and clock-to-clock skew. If the destination clock rises before the source clock, the allowable time is reduced, effectively tightening the period. If the source clock has jitter, the tools need to modify the allowed time. The timing report shows the modifications. For failing timing paths, make sure the jitter and skew numbers are reasonable.

Once the timing constraints and clocking structures are verified, timing can be met by reducing path delay. The path delay is split between the logic and the routing. The delay contributions from one or both must be reduced. Compare the logic delay against the allowed period. If the logic delay exceeds or is a large percentage of the allowed path delay, the path needs additional gates. Either the RTL needs to be modified or the synthesis engine needs to be set up differently.

If a large percentage of the path delay is from the routing and the logic is spread out over the device, there could be a placement problem. Check to see if high fanout nets, pin placement, or other structures force a spread out of placement. If not, floorplanning can be used either to reduce route delay or to determine how RTL needs to be modified.

Floorplanning Basics

Floorplanning is a technique that can be used to reduce the amount of route delay in a critical path. You can identify logic that is contributing to timing problems and guide the place and route tools to keep the logic close together. The end goal is to improve the timing of the critical paths by reducing the amount of routing delay.

Floorplanning does not change the logic that makes up the critical path. You must guide the synthesis tool to structure the gates to support the floorplan. If most of the delay in the critical path is coming from logic delay, re-synthesizing the design can bring larger gains then floorplanning. Other issues may be discovered during floorplanning that benefit from re-synthesis. One common practice is to replicate registers to stay local to clusters of dispersed loads.

Many designs benefit from floorplanning. Even with a good floorplan there is no guarantee that a design will meet timing. The floorplan does nothing to fix routing. It only provides a placement seed. For those whose design goals value design consistency over absolute performance, incremental design techniques can be used with floorplanning. For more information, see "Floorplanning Partitions" in Chapter 2, "Design Considerations" of the *Hierarchical Design Methodology Guide* (UG748): http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_1/Hierarchical_Design_Methodology_Guide.pdf.

Floorplanning can range from the abstract to the detailed. It is possible for an engineer to hand place every gate for a tight timing critical path, as shown in Figure 2. This approach is only recommended as a last resort. It is time consuming and requires knowledge of the device to get the proper routing. The gate-level placement is also fragile, and if gates or gate names change during synthesis, the placement may no longer be valid.

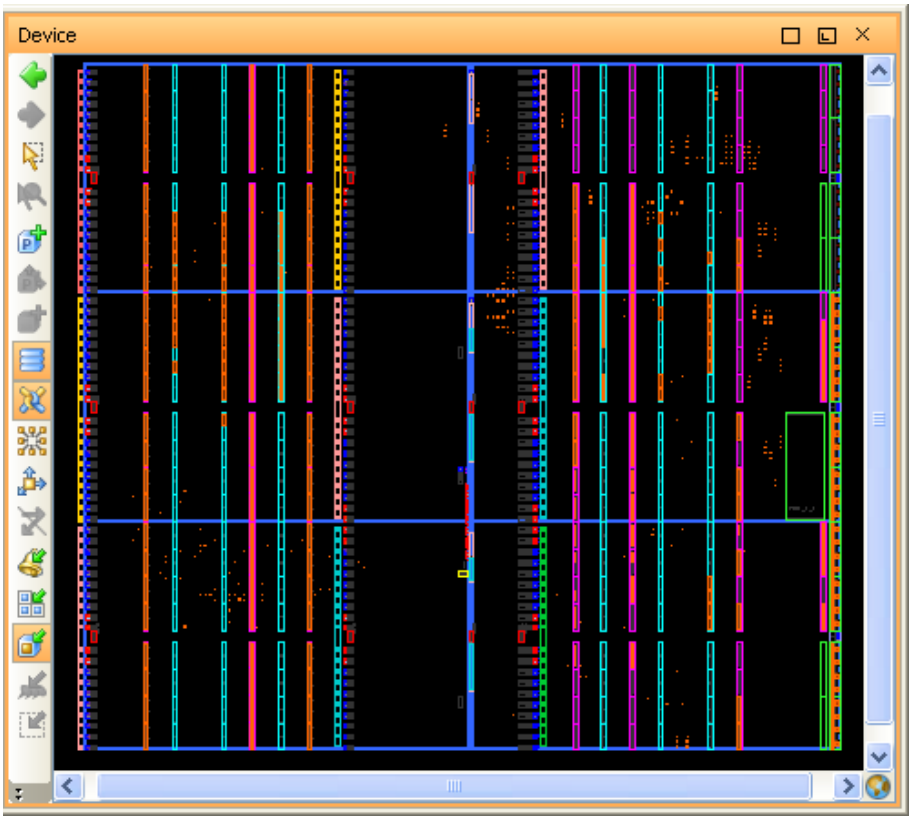

Figure 2: Floorplanned Logic by Hand

Hierarchical floorplanning is recommended in place of gate level floorplanning. Hierarchical floorplanning enables you to place one or more levels of hierarchy on a small region of the chip, as shown in Figure 3. It enables you to provide quick guidance to the placer. The placer relies on a detailed knowledge of the device and timing arcs to generate a fine grain placement. The resulting floorplan is typically resistant to design changes. The hierarchy contains all of the gates. As long as the hierarchy names do not change, gate changes do not render the floorplan invalid.

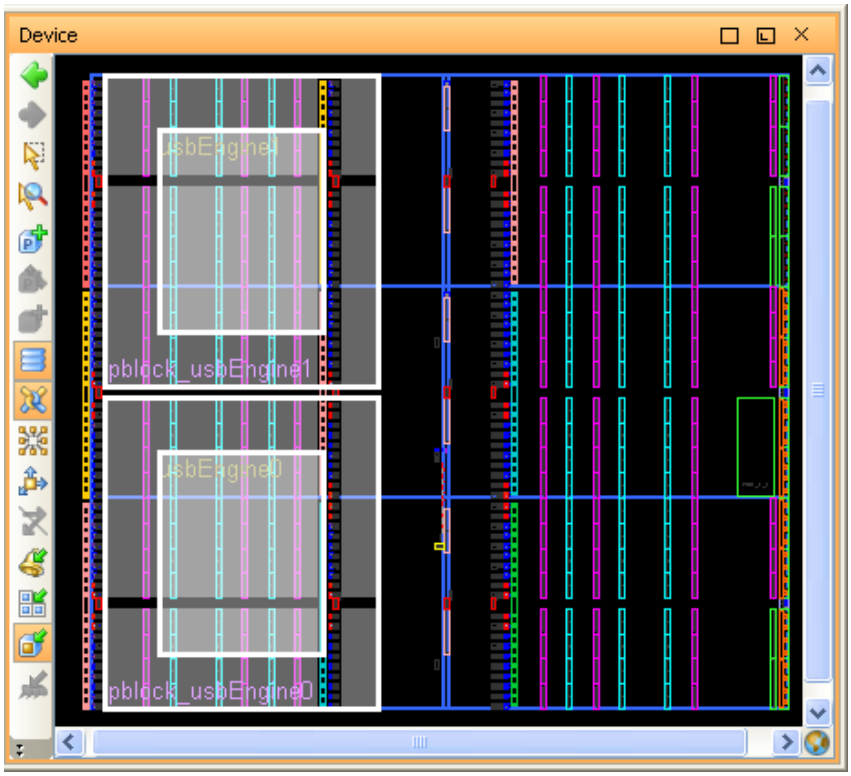

Figure 3: A Floorplanned Hierarchy

Occasionally, you may need to generate a high-level floorplan for a design as the RTL is being architected and as the pinout is being done. The high-level floorplan enables you to visualize data flow across the device. This must be done on the hierarchical blocks when the synthesized netlist does not exist. This exercise can be used to generate better RTL and a better pinout. Do not use this floorplan for place and route. It is recommended that you:

- Synthesize the design.
- Run implementation first with only pinout constraints.
- If the design fails timing, use a high-level floorplan in conjunction with the information from place and route to generate a new floorplan that is likely to improve timing.

Considerations

Floorplanning is often an iterative process. The first pass at a floorplan may address the issues in a section of the design only to show that a second section is failing. Floorplanning can hurt timing as well as improve it, especially when it is not clear what needs to be floorplanned and where the design needs to be placed. Multiple trials and notes about the design can help guide you to a working floorplan.

Floorplanning Logic with Critical Timing

A good starting point when floorplanning for the first time is to floorplan only the logic that the implementation tools consider timing critical. Generally start with the lower-level hierarchies that the Place and Route (PAR) tool finds to be timing critical. It can be tempting to floorplan the entire chip based on the data flow diagrams. This almost always hurts timing. Most FPGA designs, as presented to PAR in the post-synthesis netlist form, are designed to support full design floorplanning. However, floorplanning the entire design is not recommended.

Working with Hierarchical Netlists

The structure of the RTL can help or hinder floorplanning for timing closure. You can floorplan the hierarchy that is coded into the RTL as presented by the synthesis tool. The synthesis tool should be set up to generate a hierarchical netlist. It is much easier to work with a hierarchical netlist than one without a hierarchy. Timing can be met more easily if the hierarchy is constructed with knowledge of how the design will be spread out on the chip.

If two similar memory interfaces have to be on opposite sides of the chip, you can give each one their own copy of high fanout control signals in the RTL source. The synthesis tools often do not replicate signals optimally. When synthesis replicates a high fanout driving a flip flop, such as a reset flop, synthesis may make two copies with lower loading that both have to span the chip. Instead, the RTL designer can replicate the register by hand to create two copies with lower fanout, where one drives the loads on one side of the chip and the other drives the loads on the opposite side of the chip.

Logic Synthesis Recommendations

The following are suggestions on a logic synthesis methodology:

- To the extent possible, structure the RTL logic so that critical timing paths are confined to individual modules. Critical paths that span large numbers of hierarchical modules can be difficult to floorplan.
- Register the outputs of all the modules to help limit the number of modules involved in a critical path.
- Replicate the drivers of nets that will be separated on the die. Synthesis may need an attribute to preserve logically equivalent logic.
- Long paths in single large hierarchical block can make floorplanning a difficult task. Consider dividing large hierarchical blocks in the RTL. It is easier to work with smaller hierarchical blocks.
- Intermingled critical paths can be difficult to floorplan. Consider dividing large critical blocks into smaller and easier to isolate blocks.

- If the design is expected to change often, consider an incremental approach to synthesis. In an incremental approach, individual blocks can be synthesized separately or the synthesis attributes (**SYN_HIER=HARD**) can be used to preserve the hierarchy. Hierarchy preservation helps an incremental flow but may hurt performance since global optimizations across hierarchy are disabled. This tradeoff needs to be considered before you embark on an incremental RTL synthesis methodology.
- Constrain the synthesis engine to rebuild or otherwise preserve the hierarchy in the synthesized netlist. Flattened netlists may be optimal from a synthesis perspective, but they make it very difficult to reliably floorplan and constrain placement. Consider using the synthesis option to rebuild the hierarchy. For XST, use **-netlist_hierarchy = rebuilt**. If using the PlanAhead™ tool for synthesis, the PlanAhead defaults strategy includes this option.

Increasing Consistency

Floorplanning can help with design consistency and quality of results (QOR). It is possible to create a floorplan that takes a design from failing timing to meeting timing. Many hierarchical floorplans will work across multiple netlist revisions as bug fixes are incorporated from simulation and board testing. However, blocks that meet timing on one pass may fail timing on another pass. Placement is only a guide to place and route. Routing is not locked down. If achieving design consistency is more important than achieving the highest performance, consider the tradeoffs of incorporating incremental synthesis and implementation. These flows can limit the scope of gate-level netlist changes and preserve placement and routing between different runs. These techniques gain consistency at the cost of some QOR. The decision to use either of these flows should be made at the start of a design cycle and not once the design is well underway. For more information, refer to Chapter 2, "Design Considerations" of the *Hierarchical Design Methodology Guide* (UG748):
http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_1/Hierarchical_Design_Methodology_Guide.pdf.

Using Clock Resources to Guide Floorplanning

Different FPGA device families have different restrictions on the placement of logic for a design with a high percentage of clock resources of the device. Consider the clock rules of the device when placing the logic. The PlanAhead tool can help constrain certain clocks to certain regions on the chip. It is possible to graphically display the various clock regions or clock quadrants within the chip. The Clock Region Properties or Pblock Properties Statistics will show what clock nets and clock regions are in all Pblocks, which are defined by AREA_GROUP constraints. The schematic view can show what logic and hierarchy is attached to each clock net.

Chapter 2

The Floorplanning Flows

This chapter contains the following sections:

- Recommended Flows
- Re-use Flow
- Hierarchical Floorplanning Flow
- Using Floorplanning for Timing Closure: An Example
- Summary

Recommended Flows

The two commonly used floorplanning flows discussed in this guide are:

- Re-use flow
- Hierarchical floorplanning flow

Re-use Flow

The re-use flow can quickly help a design that meets timing only some of the time. The idea is to re-use some of the block RAM and DSP48 placement from a successful implementation run to seed a later run.

The re-use flow has the following advantages:

- Is quick to apply.
- Can cut down implementation run times.
- Can improve consistency of meeting timing.
- Does not require much knowledge of the device to get the design placed.

The re-use flow has the following disadvantages:

- Will not work if the design does not meet timing some of the time.
- Limits design change.
- May not consistently meet timing.

Hierarchical Floorplanning Flow

The hierarchical floorplanning flow is more powerful than the re-use flow. A good hierarchical floorplan can close timing on a design that has never met timing. The analysis done to create a good floorplan can suggest design and logic changes to meet timing more easily and consistently.

The hierarchical floorplanning flow has the following advantages:

- Is resistant to design change.
- Can close timing.
- Can bring consistency.

The flow requires engineering time, and may require iterations.

In both flows, it is possible to significantly impact timing. If floorplanned logic is slower now, remove the floorplanned logic and try something else. If logic that is not floorplanned is slower try floorplanning it as well.

Re-use Flow

One of the sources of timing variability is the macro placement, such as block RAM and DSP48. Placed macros can act as a seed to the LUT/FF placement. By looking at the macro placement from an implementation run that meets timing and re-using the macro placement, it is possible to reduce some of the variability from one implementation run to the next. The idea is to let the implementation tools find a placement that meets timing and re-use some of it for later turns. This approach can be used when:

- The design meets timing some of the time.
- The names and structures for the macros do not change.

The placement of the larger macros can help suggest a placement for the other gates. Timing can be more stable and, in some cases, implementation run times will decrease.

Start with a PAR run that routes and meets timing. Look for a Timing Score of 0, and 0 unroutes in the Project Summary (timing report), as shown in the following figure. If you have multiple runs that meet timing, start with the one that has the shortest implementation run time. You can use an implementation run in scripts, in Project Navigator, or in the PlanAhead™ tool. You will want to load the design that meets timing into PlanAhead to constrain placement.

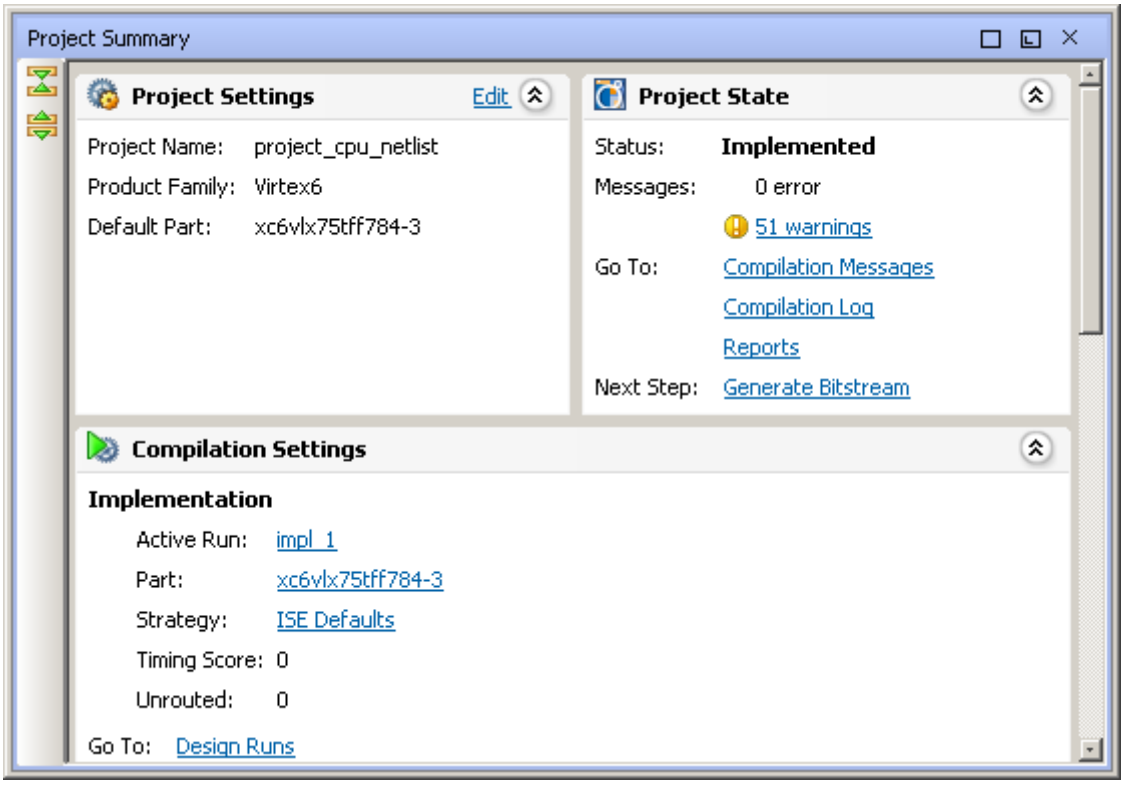

Figure 4: Project Summary

To see where the implementation tools placed the gates, use one of:

- Run the **Analyze Timing / Floorplan Design** process in Project Navigator.
- Click the **Implement Design** button in the Flow Navigator in PlanAhead to open the implemented design.
- If implementation was run in stand-alone scripts, create a new PlanAhead project and select **Import ISE Place and Route results** in the New Project wizard.

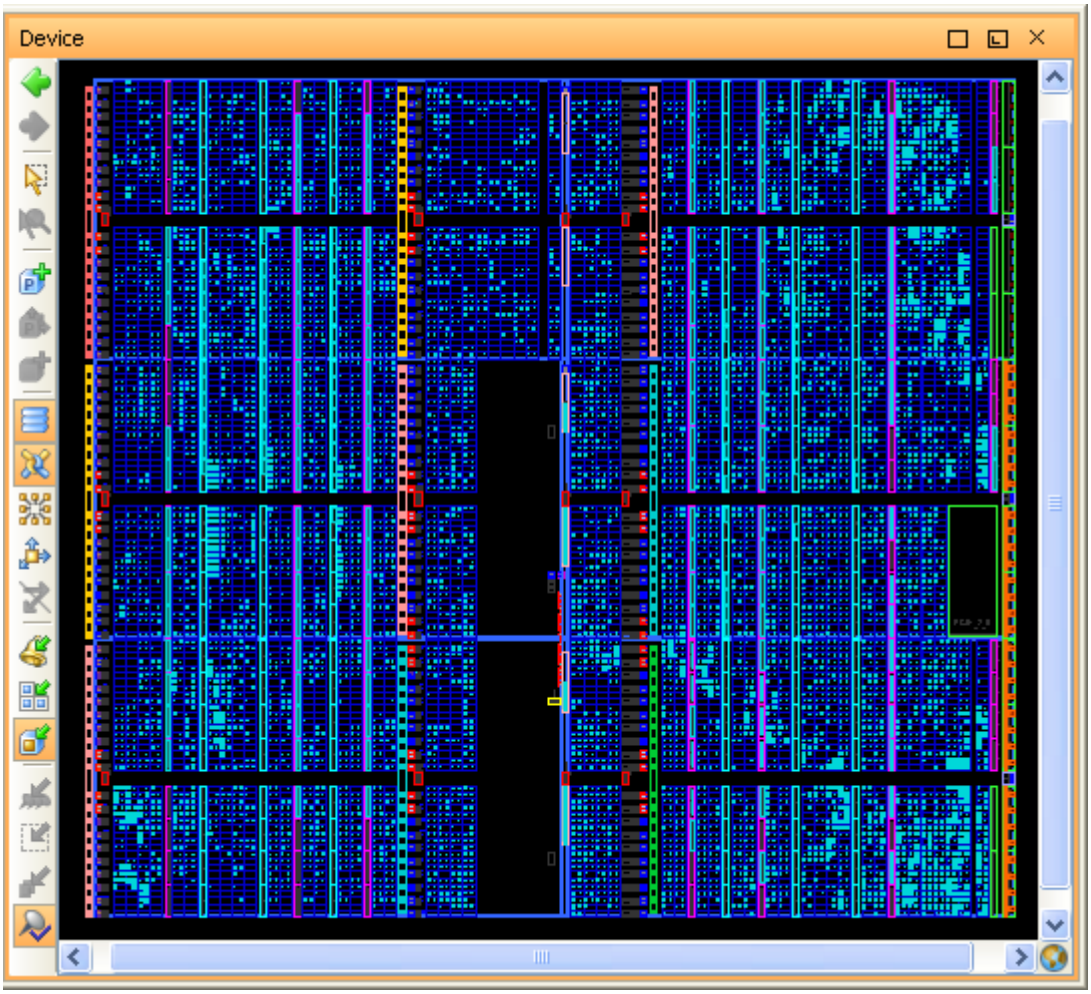

Figure 5: Viewing Implementation Placement

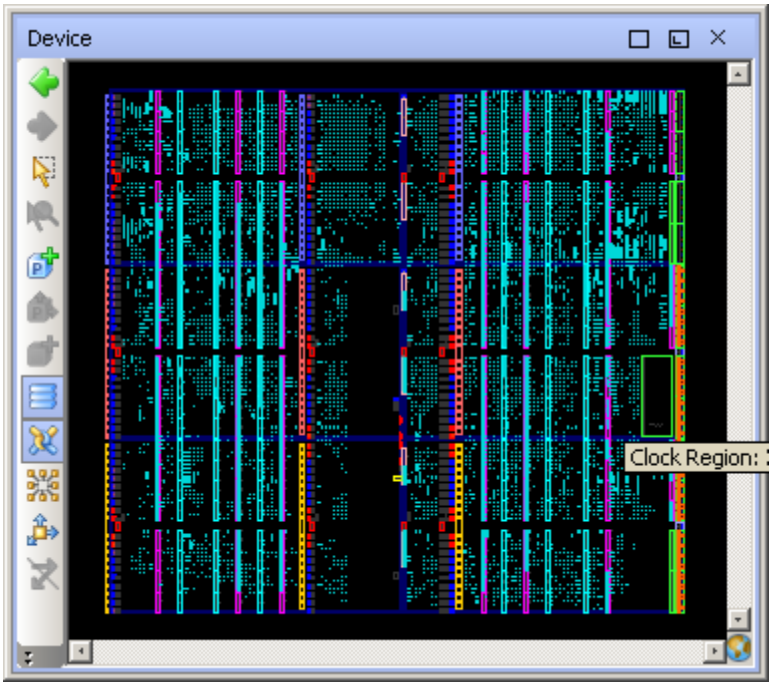

Figure 6: Viewing Implementation Placement

When the design meets timing, it is also possible to re-use the placement. Do not fix everything in place since the design is likely to change. On most designs the block RAM and DSP48 primitives have a relatively stable set of primitives and names. Re-using the placement of just the block RAM and DSP48 can help keep timing as other gates change. In the PlanAhead tool, it is easy to find all Block Memory (RAMB and FIFO primitives), as well as Block Arithmetic (MULT and DSP primitives, depending on architecture). It might be easier to visualize placements using the **Highlight** or **Mark** commands.

Select **Edit > Find**, and use the Find dialog box to search for these primitives.

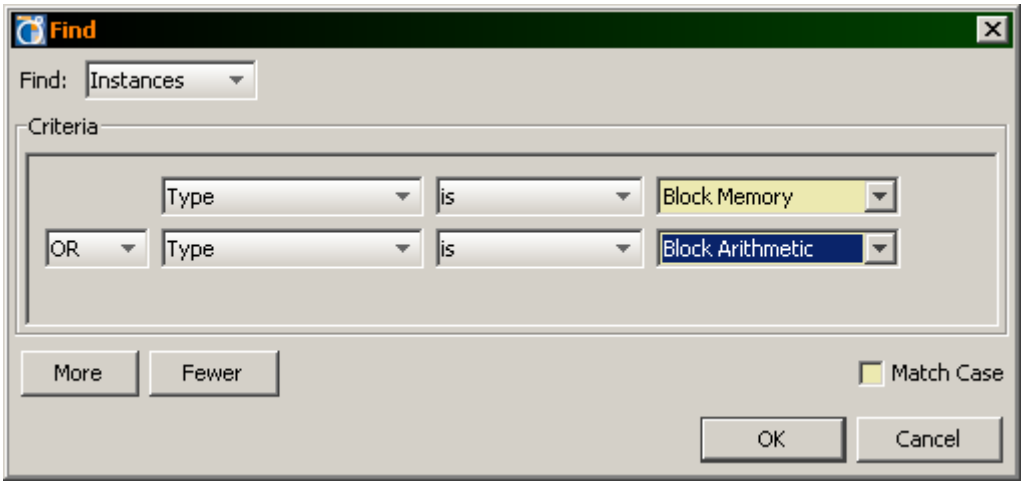

Figure 7: Searching for the Memory and Arithmetic Blocks

The search compiles a list of all matching objects. All placements for the implementation run are loaded. The macro placement that is needed for a seed needs to be isolated from the other placement.

The PlanAhead software has two types of placement:

- Fixed - Placement from a UCF, hand created by the user, or designated by the user in the PlanAhead tool is defined as *fixed*. This type of placement can be reused.
- Unfixed - Placement that is back-annotated from the implementation tools is defined as *unfixed*. This type of placement should not be reused.

To fix the logic:

1. Select all placements you wish to fix (as from the Find Results, as shown in Figure 8).
2. Right-click and select **Fix Instances**.

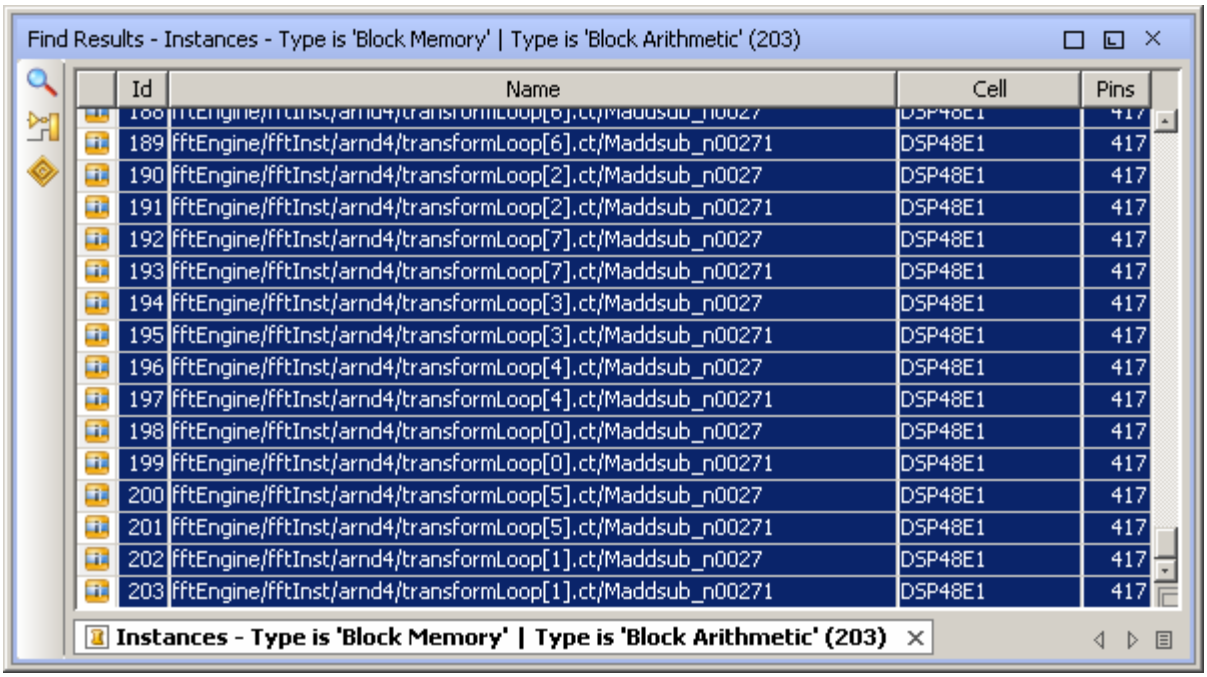

Figure 8 Selecting Logic in Find Results Dialog Box

The color of the placed logic in the Device view will change color to denote the change in how the tool handles the placement. Save and close the project. The UCF will now have multiple gate level constraints of the form:

```
INST "usbEngine0/usb_out/buffer_fifo/Mram_fifo_ram" LOC = RAMB36_X3Y14;
INST "fftEngine/fftInst/arnd2/ct5/Maddsub_n0027" LOC = DSP48_X1Y26;
```

If the names in the gate level netlist change, the placement will need to be re-run as the references defined in the LOC constraints need updating. If there is a change to the macros or the logic around the macros, the placement should be cleared and rerun. Additionally, if the design starts failing timing on a regular basis, you can run PAR without the LOC constraints on the macros. A more advanced user may want to tweak placement of individual block RAM or DSP48. For information on how to analyze and modify placement, see Chapter 10, "Analyzing the Implementation Results" and Chapter 11, "Floorplanning the Design" of the *PlanAhead User Guide* (UG632) http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_1/PlanAhead_UserGuide.pdf.

Hierarchical Floorplanning Flow

Designs that have not met timing require a more involved approach. The hierarchical floorplanning flow is the best approach for closing timing in a design that has not met timing. You can take smaller levels of hierarchy, constrain the hierarchy to a region on the chip, and use that as a guide to implementation. Implementation has comprehensive knowledge of the critical paths and the structure of the chip. Implementation generally does a good job of the fine grain placement. It cannot always find a solution for the coarse placement for a large flat design. You can help implementation by seeding a coarse placement with the hierarchies that contain gates that fail timing after implementation.

It is helpful to have an idea of the final pinout when floorplanning. Blocks that connect to IOs often want to be placed near their IOs. During the floorplanning process it may become obvious that a pinout is pulling timing critical paths apart. If caught early enough it may be possible to change the pinout or logic to improve timing closure.

Using Floorplanning for Timing Closure: An Example

When creating a floorplan, the following questions should be kept in mind:

- What are the timing failures?
- What is the critical hierarchy?
- Are changes to floorplanning or logic alone sufficient to close timing?
- Does anything else need to be floorplanned?
- Can the critical hierarchies be floorplanned?
- What should be placed where?

These questions can be answered by looking at the timing paths, placement and structure of the logic in the paths, and by knowledge of the pinout and the design, as described in the following example of a design walk through.

Looking at Place and Route Results

Only post implementation timing numbers identify what logic is failing timing. After the design has been run through implementation, and the design fails timing, load the results into the PlanAhead tool. The placement and timing results, and gates can all be seen in one place. Selecting multiple critical paths and viewing the placement can offer ideas for troubleshooting, as shown in Figure 9.

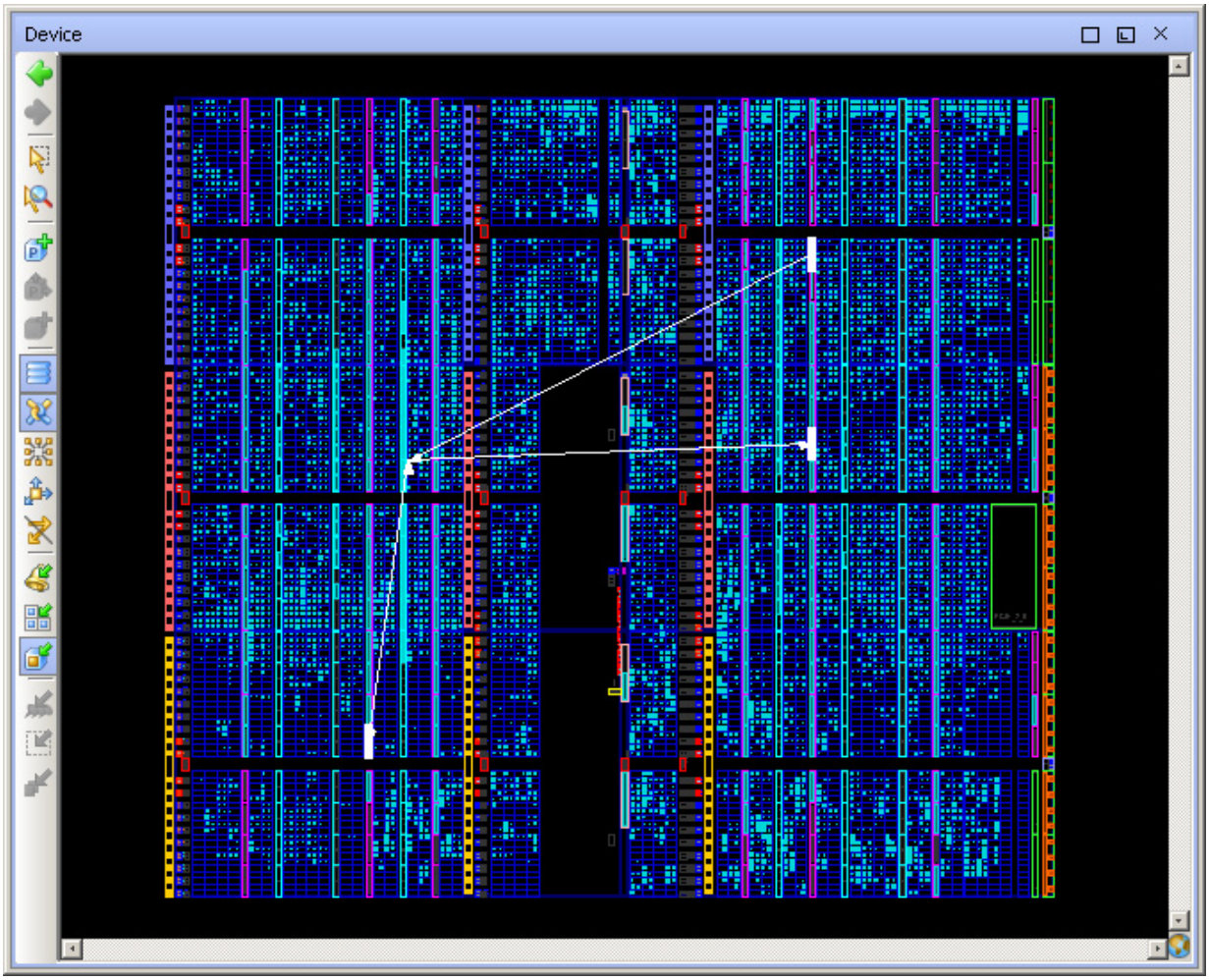

Figure 9: Placement of Paths Failing Timing

The block RAMs with the critical paths are spread out over more of the chip then they need to be. Floorplanning can be used to generate a tighter placement. The timing problem occurs in the paths between block RAM. These paths are good candidates for floorplanning.

Looking at Timing Results

Analyzing the paths between block RAM will enable you to determine whether you need to floorplan, change logic, or both to close timing. The path delay for the above critical path shows two nets with long route delays. The details are shown in Figure 10. The path is failing timing by over 1 ns. The first net has 2.25 ns route delay. The third net has 1.5 ns route delay. Even though the fanouts are 40 and 256, the route delay can be reduced with improved placement.

Data Path

Delay Type	Delay	Cumulative	Location	Logical Resource
RAMB36E1 (Trcko_DOB)	1.591	1.591	RAMB36_X4Y19	usbEngine1/usb_dma_wb_in/buffer_fifo/Mram_fifo_ram
net (fanout=40)	2.245	3.836		usbEngine1/ma_adr[14]
LUT6 (Tilo)	0.053	3.889	SLICE_X28Y64	usbEngine1/u5/ma_we1
net (fanout=1)	0.283	4.172		usbEngine1/ma_we
LUT4 (Tilo)	0.053	4.225	SLICE_X28Y64	usbEngine1/u2/Mmux_sram_we11
net (fanout=256)	1.548	5.773		usbEngine1/sram_we_o
RAMB36E1 (Trcck_WEA)	0.437	6.210	RAMB36_X4Y13	usbEngine1/usbEngineSRAM/Mram_snoopyRam1
Total	6.210	6.210 Logic: 2.134 Net: 4.076		

Figure 10: Detailed Data Path

A hierarchical floorplan can reduce the route delay in the critical logic. Logic delay limits the amount of performance gain you can achieve. For designs with a large percentage of logic delay, you can change the code or update synthesis to modify the gates.

Looking at the Gates and the Hierarchies

You can floorplan gates through individual LOC and placement constraints. Moving the gates around by hand to improve timing is not recommended because identifying the gates is a slow process and placing the gates is a slow and difficult process. Also, remember that when the logic in the gate floorplan changes, the floorplan needs to be redone.

Instead ask the question, what hierarchy is timing critical? Implementation reports timing problems for usbEngine1 in Figure 10. This level of hierarchy or one or more levels of sub-hierarchy are candidates for hierarchical floorplanning. You must investigate the design to determine which hierarchy should be floorplanned.

Start off by loading the critical paths into the schematic. As shown in Figure 11, the schematic will show which gates are involved in the critical path and in which hierarchy the gates are located. You can trace the logic around the critical gates in the schematic to see how the non-critical logic is structured.

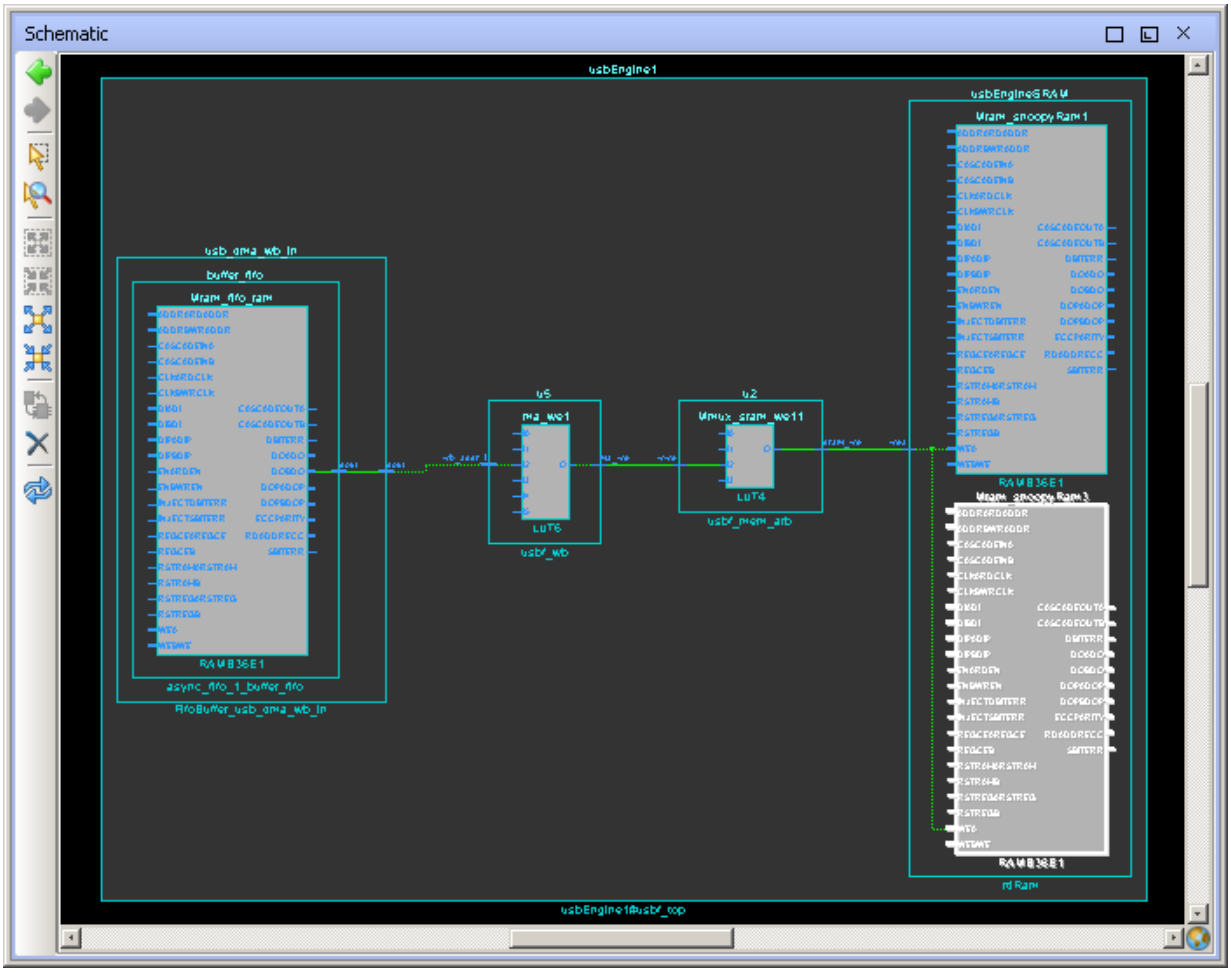

Figure 11: The Gates and Hierarchy in the Critical Path

The floorplan should constraint at least the timing critical paths between block RAMs inside usbEngine1. So far, usbEngine1 appears to be a good candidate for floorplanning. If usbEngine1 is a large portion of the chip, instead we would try to floorplan the four levels of sub hierarchy that contain the critical path.

To quickly determine which gates should be floorplanned, look at the placement in the Device view. In Figure 12, the gates in the critical hierarchies are colored green. The gates in the non-critical hierarchies are colored yellow. In the critical hierarchies there is high utilization of the block RAM. The non-critical hierarchies contain a lot of LUT/FF logic that can be placed between the block RAM. The entire hierarchy is approximately 20% of the design. Before floorplanning usbEngine1, examine the pinout and design connectivity. The design may show that usbEngine1 is not a good candidate.

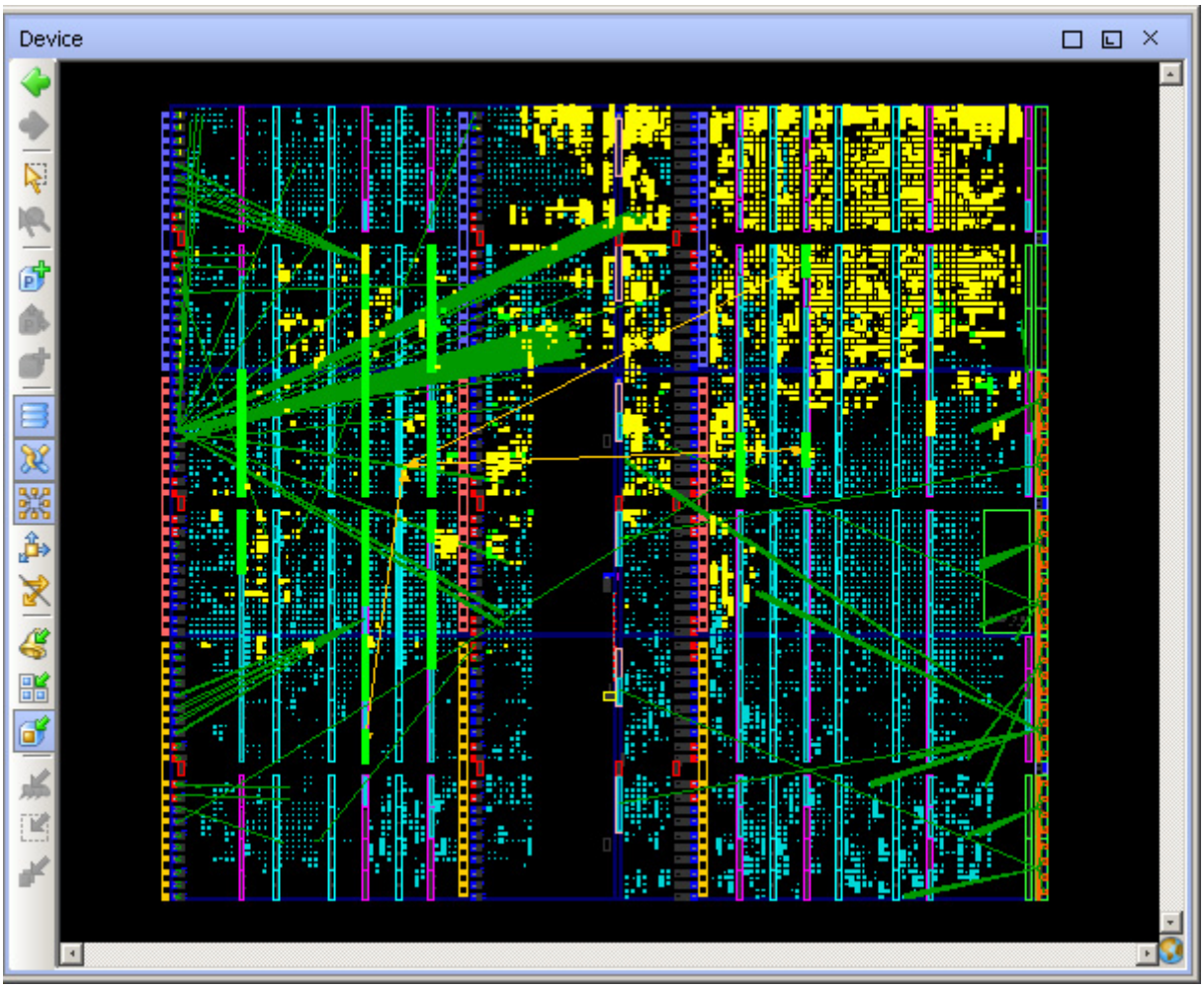

Figure 12: Critical and Non-Critical parts of usbEgine1

The next step is to confirm that usbEngine1 is a good candidate for floorplanning and to figure out where it should be placed. It is helpful to create a top-level floorplan on the device. The top-level floorplan can provide hints about what logic is influencing the placement of other logic. The blocks that are spread out across the chip are bad candidates for floorplanning.

IO connectivity is displayed as green IO lines. An example is shown in Figure 13. Look for the lines going from the middle IO bank on the left side of the chip to the yellow logic in the middle. Connectivity between hierarchical blocks displays as bundles of nets between the placed hierarchies. An example of this is shown in Figure 13. At quick glance, you can see that there are many inter-connected hierarchies. You can see when a pinout draws a hierarchy across the chip.

Figure 13 shows the top-level floorplan for this design. It is easy to see that only one hierarchy is spread around the chip. A second hierarchy spans the length of the right side. The pinout would support floorplanning usbEngine1. Based on the pinout, usbEngine1 (in white) should be placed in the upper left corner of the device.

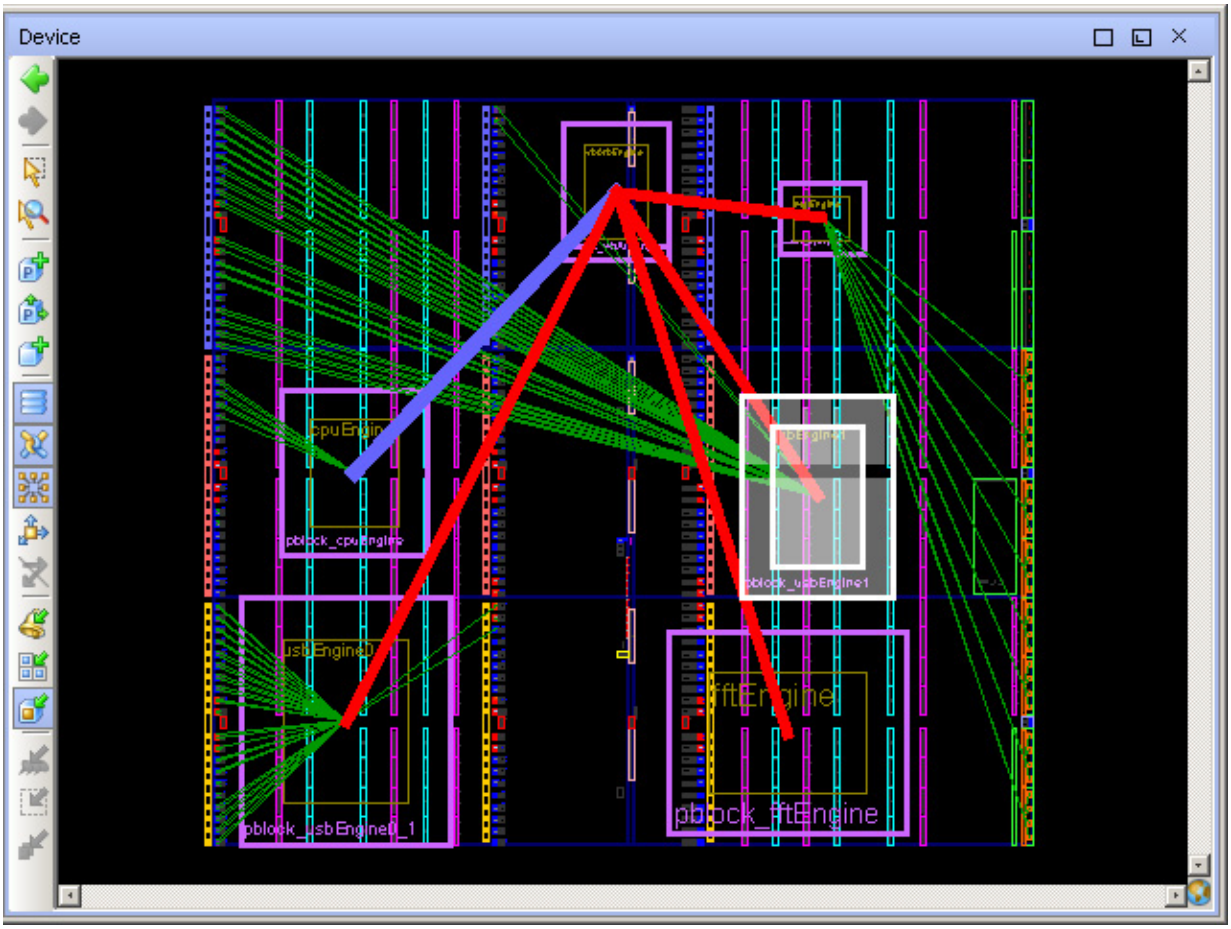

Figure 13: A Top-Level Floorplan for Analysis

Shaping the Floorplan for the Critical Hierarchy

The floorplan suggests the critical hierarchy should be in the upper left corner. Design analysis shows that the critical hierarchy uses multiple block RAM sites. The pinout shows the critical hierarchy connects to the two IO banks on the top left of the chip. It makes sense to try to floorplan the logic to use slices and block RAM between these banks. A good target is to try to size the block to use 100% of the block RAM (or DSP, if applicable) and about 80% of the slices.

Deciding What Else Should Be Floorplanned

This design has two copies of the same gates: usbEngine1 and usbEngine0. Implementation has shown that there is a timing problem with usbEngine1, which will likely appear in usbEngine0 as well. You will need to solve the timing problems of each block separately. Consider both USB blocks as two separate timing critical hierarchies, and floorplan each hierarchy separately. A final floorplan that meets timing is shown below in Figure 14.

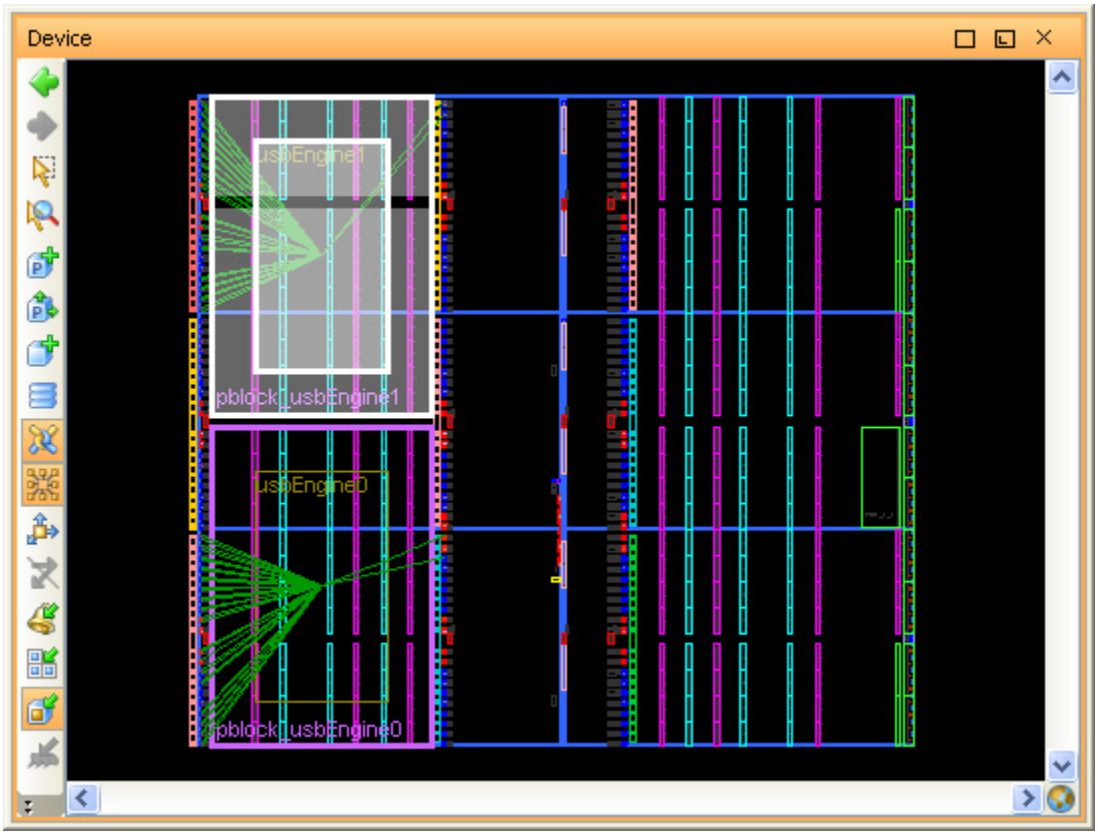

Figure 14: A First Pass Floorplan

PlanAhead creates a construct that enables you to constrain any subset of netlist hierarchy to a region on the chip. They are created using the **New Pblock** and **Assign to Pblock** commands. The Pblocks are turned into AREA_GROUP constraints in the UCF to guide implementation and they keep the level(s) of hierarchy to various regions on the chip.

```
INST "usbEngine1" AREA_GROUP = "pblock_usbEngine1";
AREA_GROUP "pblock_usbEngine1" RANGE=SLICE_X0Y60:SLICE_X43Y119;
AREA_GROUP "pblock_usbEngine1" RANGE=DSP48_X0Y24:DSP48_X2Y47;
AREA_GROUP "pblock_usbEngine1" RANGE=RAMB18_X0Y24:RAMB18_X2Y47;
AREA_GROUP "pblock_usbEngine1" RANGE=RAMB36_X0Y12:RAMB36_X2Y23;
```

These lines define the shape on the chip, and what to place into it. It is possible to set up a region that does not constrain all these ranges. It is possible to constrain only the block RAM to sites on the chip by using:

```
INST "usbEngine1" AREA_GROUP = "pblock_usbEngine1";
AREA_GROUP "pblock_usbEngine1" RANGE=RAMB18_X0Y24:RAMB18_X2Y47;
AREA_GROUP "pblock_usbEngine1" RANGE=RAMB36_X0Y12:RAMB36_X2Y23;
```

The slices and DSP are now unconstrained.

Floorplanning Iteratively

Floorplanning is an iterative process. When it is not obvious what hierarchy should be floorplanned, use trial and error until some timing improvement is seen. If timing degrades in the blocks that are floorplanned, analyze why. The design may have connections that are not obvious on the first analysis. After the first floorplan, you may need to revise the floorplan. It is helpful to save each floorplan in case you want to revisit your work later. A simple approach generally works better and takes less time, so keep things simple.

Some helpful hints when working iteratively:

- If critical paths are located within logic that is not floorplanned, create a new Pblock. Identify the levels of hierarchy that contain the critical paths, assign them to a new Pblock, and place the Pblock on the chip. If the placement is reasonable, keep this Pblock for place and route.
- If critical paths are within a single Pblock, revise the Pblock. Consider creating a Pblock within the Pblock that contained the failing timing path to constrain the critical hierarchy more tightly. Alternately, work with lower levels of hierarchy, remove some logic and use a smaller Pblock.
- If critical paths are between a Pblock and unconstrained hierarchy, add the unconstrained logic to a Pblock. The first option is to create a new Pblock to hold the critical path and place it nearby. The second option, which works if the unconstrained logic is small, is to create a Pblock to hold both the critical path as well as the unconstrained logic.
- If critical paths are between two Pblocks, revise the Pblocks. Consider moving or reshaping the Pblocks so they are closer. Consider embedding one Pblock inside the other. Consider moving logic from one Pblock to the other.
- In all cases, if the logic in a critical hierarchy is large, heavily interconnected, or being pulled around the chip by scattered loads, do not place it at first. Start working with the timing critical hierarchy that has a good placement. Revisit the hierarchy on a later pass if it is still a problem. If paths are a persistent timing problem consider revising the RTL and re-synthesizing.
- If sections of the design are floorplanned and consistently failing timing it may be time to take a step back. Consider removing the floorplanning constraints to see what happens. If timing improves, try something different. Sometimes a new approach suggests itself.
- When upgrading from one ISE® Design Suite release to the next, run the design through implementation unconstrained. A new release may obviate the need for floorplanning.

Summary

Floorplanning can help improve timing performance and consistency. The re-use flow can bring consistency to a design that has met timing. The hierarchical floorplanning flow can help a design that has not met timing or can improve consistency. As the design changes it may be necessary to revisit either approach.