Floorplanning Methodology Guide

UG633 (v13.1) March 1, 2011

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/01/2011	13.1	Minor updates throughout.

Table of Contents

Chapter 1

Introduction to Floorplanning

Floorplanning is the process of:

- Choosing the best grouping and connectivity of logic in a design, and
- Manually placing blocks of logic in an FPGA device.

The goals of floorplanning are to:

- Increase density, routability, or performance.
- Reduce route delays for selected logic by suggesting a better placement.

Floorplanning has become necessary as designers create ever-more complex designs for ever-larger FPGA devices. Implementation software has improved to meet these complexities.

On some designs, you can guide the implementation software by means of a floorplan to:

- Higher system clock frequency
- Shorter implementation run times
- Greater consistency in timing
- In some cases, all of these benefits together

Benefits of Floorplanning

A good floorplan can:

- Improve performance.
- Enable a placed and routed design to meet timing.

Xilinx recommends floorplanning when a design:

- Does not meet timing consistently, or
- Has never met timing.

When To Floorplan

When to floorplan varies greatly among design teams. Design teams may floorplan:

- Before the first iteration through place and route.
- When a problem is identified before floorplanning.
- When a design does not consistently meet the setup timing constraint.

Timing Closure Basics

Floorplanning reduces path delays, leading to timing closure.

During implementation, the software:

- Compares the delay of the logic and routing against the time allowed by the timing constraint.
- Takes Clock Jitter and Clock-to-Clock Skew into account.
- Reports the amount of time by which the paths:
  - Beat timing constraints (meet timing), or
  - Exceed timing constraints (fail timing).

Figure 1-1, Example Timing Report, shows an example timing report.

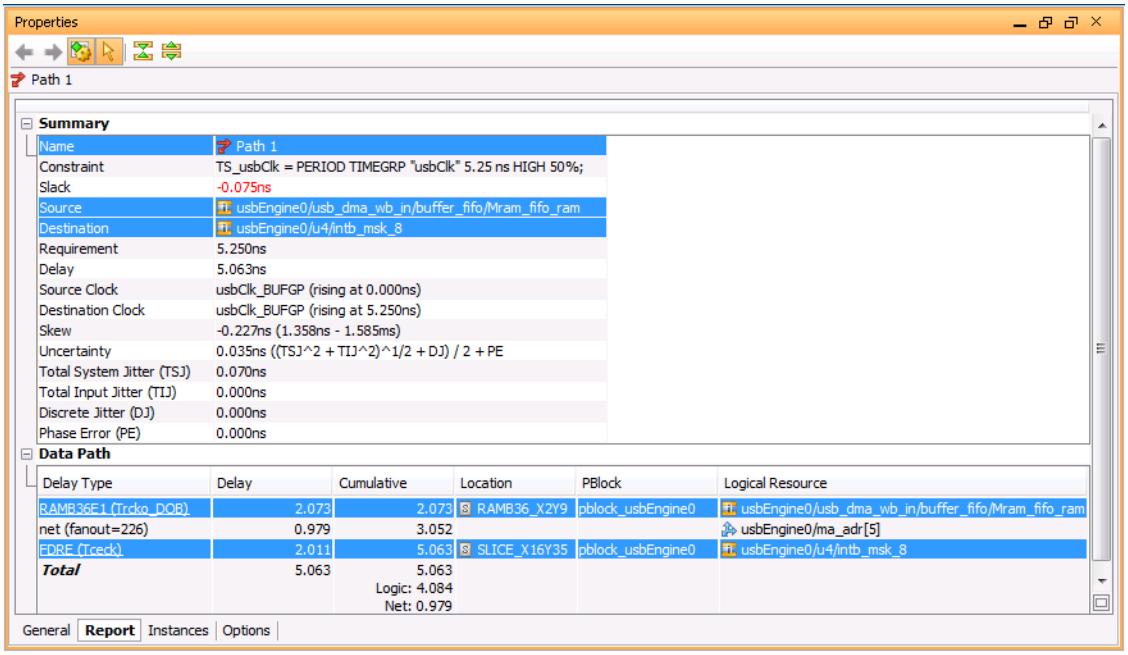

Figure 1-1: **Example Timing Report**

Ensure That Timing Constraints Are Accurate

As a first step in floorplanning:

- Ensure that the timing constraints are accurate.
- Determine whether the path is a multi-cycle path or a false path.

Multi-Cycle Paths and False Paths

Sections of some designs are not clocked every clock cycle, or the paths may not be reached due to the control structure. The implementation software 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.

For a discussion of multi-cycle paths and false paths, see the *Xilinx Timing Constraints User Guide (UG612)* referred to in Appendix A, Additional Resources.

Clock Jitter and Clock-to-Clock Skew

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 software modifies the allowed time.

The timing report shows the modifications. For failing timing paths, make sure the jitter and skew numbers are reasonable.

Reducing Path Delay

Once the timing constraints and clocking structures are verified, timing can be met by reducing path delay. The path delay is split between logic and 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 requires additional work on the gates. You can:

- Modify the RTL to add in registers, or
- Change the constraints, or
- Set up the synthesis engine differently.

If a large percentage of the path delay comes from routing, placement may be a problem.

Check to see if high fanout nets, pin placement, or other structures force a spread out of placement. If not, use floorplanning to either:

- Reduce route delay, or
- Determine how RTL needs to be modified.

Floorplanning Basics

Floorplanning can reduce the route delay in a critical path. You can:

- Identify logic that is contributing to timing problems.
- Guide the place and route software to keep the logic close together.

The 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 software to structure the gates to support the floorplan. If most of the delay in the critical path comes from logic delay, re-synthesizing the design may bring larger gains than floorplanning.

During floorplanning, you may discover other issues that might benefit from re-synthesis. Designers often replicate registers to stay local to clusters of dispersed loads.

Even a good floorplan does not guarantee that a design will meet timing. The floorplan does not fix routing. The floorplan only provides a placement seed.

You can use incremental design techniques with floorplanning for designs in which design consistency is valued over absolute performance.

For more information, see "Floorplanning Partitions" in Chapter 2, "Design Considerations" of the *Hierarchical Design Methodology Guide* (*UG748*) referred to in Appendix A, Additional Resources.

Approaches to Floorplanning

- Detailed Gate-Level Floorplanning

  Places individual logic elements of a critical path to a specific site on the chip.

- Hierarchical Floorplanning

  Constrains levels of hierarchy to specific regions on the chip.

Detailed Gate-Level Floorplanning

You can hand place every gate for a tight timing critical path, as shown in Figure 1-2, Floorplanned Logic by Hand.

Xilinx recommends detailed gate-level floorplanning only as a last resort:

- Detailed gate-level floorplanning is time consuming.
- Detailed gate-level floorplanning requires comprehensive knowledge of the device to achieve the proper routing.
- The resulting detailed gate-level placement is fragile. If gates or gate names change during synthesis, the placement may no longer be valid.

Figure 1-2: **Floorplanned Logic by Hand**

Hierarchical Floorplanning

Xilinx recommends hierarchical floorplanning instead of gate level floorplanning. Hierarchical floorplanning allows you to:

- Place one or more levels of hierarchy on a small region of the chip.

  See Figure 1-3, Floorplanned Hierarchy.

- Provide quick guidance to the placer.

The hierarchy contains all the gates. Gate changes do not render the floorplan invalid as long as the hierarchy names do not change. The placer relies on a comprehensive knowledge of the device and timing arcs in order to generate a fine grain placement. The resulting floorplan is typically resistant to design changes.

Figure 1-3: **Floorplanned Hierarchy**

Generating a High-Level Floorplan

You may need to generate a high-level floorplan for a design while the RTL is being architected, and the pinout is being implemented. The high-level floorplan:

- Enables you to visualize data flow across the device.
- May help you see how to generate better RTL and a better pinout.

Note: Do not use this floorplan for place and route.

Xilinx recommends that you:

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

Floorplanning Considerations

- Floorplanning is often an iterative process.

  The first pass at a floorplan may address issues in one section of the design, only to reveal that a different section is failing.

- Floorplanning can hurt timing as well as improve it.

  This is especially true 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 you create a working floorplan.

Floorplanning Timing-Critical Logic

When you initially floorplan a design, Xilinx recommends that you:

- Floorplan only the logic that the implementation software considers timing-critical.
- Begin with the lower-level hierarchies that the implementation software considers timing-critical.

Do Not Floorplan the Entire Design

Most FPGA designs, as presented to the implementation software in the post-synthesis netlist form, support full design floorplanning.

Xilinx does not recommend floorplanning the entire design.

Floorplanning the entire design based on the data flow diagrams almost always hurts timing.

Working With Hierarchical Netlists

The RTL structure can help or hinder floorplanning for timing closure. You can floorplan the hierarchy that is coded into the RTL as presented by the synthesis software.

Set up the synthesis software to generate a hierarchical netlist. Working with a hierarchical netlist is easier than working with a netlist without a hierarchy.

Timing can be met more easily if you understand how the design will be spread out on the chip when you construct the hierarchy.

If two similar memory interfaces must be placed on opposite sides of the chip, you can give each interface its own copy of high fanout control signals in the RTL source.

The synthesis software often does 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.

You can duplicate the register by hand to create two copies with lower fanout:

- One register drives the loads on one side of the chip.
- The other register drives the loads on the opposite side of the chip.

Logic Synthesis Recommendations

- Structure the RTL logic to confine critical timing paths to individual modules. Critical paths that span large numbers of hierarchical modules are difficult to floorplan.
- Register the outputs of all the modules to reduce the number of modules involved in a critical path.
- Replicate the drivers of nets that are separated on the die. Synthesis may need an attribute to preserve logically equivalent logic.
- Long paths in single large hierarchical block can make floorplanning difficult. Xilinx recommends dividing large hierarchical blocks in the RTL, since to is easier to work with smaller hierarchical blocks.
- Intermingled critical paths can be difficult to floorplan. Divide large critical blocks into blocks that are smaller and easier to isolate.
- If you expect the design to change often, consider using incremental synthesis.
  - Synthesize individual blocks separately, or
  - Use **SYN_HIER=HARD** to preserve the hierarchy.

  Hierarchy preservation helps an incremental flow, but may hurt performance since global optimizations across hierarchy are disabled. Consider this trade-off before using incremental RTL synthesis methodology.
- Constrain the synthesis engine to rebuild or otherwise preserve the hierarchy in the synthesized netlist.
  - Flattened netlists may be optimal for synthesis, but make it difficult to:
    - Floorplan.
    - Constrain placement.
  - Use the synthesis option to rebuild the hierarchy.
    - For XST, use **-netlist_hierarchy = rebuilt**.
    - The PlanAhead™ software includes the synthesis option by default.

Increasing Consistency

A successful floorplan can:

- Increase design consistency.
- Improve quality of results (QOR).
- Take a design from failing timing to meeting timing.

Many hierarchical floorplans 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 trade-offs 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 achieve consistency at the cost of some QOR. You should decide which flow to use at the beginning of your design cycle, not after the design is well underway.

For more information, see Chapter 2, "Design Considerations" in the *Hierarchical Design Methodology Guide (UG748)* referred to in Appendix A, Additional Resources.

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. Consider the clock rules of the device when placing the logic.

The PlanAhead software can:

- Help constrain some clocks to certain regions on the chip.
- Graphically display the various clock regions or clock quadrants within the chip.

The Clock Region Properties or Pblock Properties Statistics:

- Show where clock resources are located in the Clock Resources view.
- Show which clock nets and clock regions:
  - Are present in all Pblocks, and
  - Are defined by AREA_GROUP constraints.

The schematic view can show the logic and hierarchy attached to each clock net.

Chapter 2

Floorplanning Flows

Xilinx supports the following Floorplanning Flows:

- Re-Use Flow

  Close timing on a design that meets timing some of the time.
- Hierarchical Floorplanning Flow

  Close timing on a design that has never met timing.

Re-Use Flow

The Re-Use Flow can close timing on a design that meets timing some of the time. You can re-use some of the block RAM and DSP48 placement from a successful implementation run to seed a later run.

Advantages and Disadvantages of Re-Use Flow

The Re-Use Flow has the following advantages:

- Can be applied quickly.
- Can reduce implementation run times.
- Can improve consistency of meeting timing.
- Does not require extensive knowledge of the device to place the design.

The Re-Use Flow has the following disadvantages:

- Does not work if the design does not meet timing at all.
- Limits design change.
- May not consistently meet timing.

How Re-Use Flow Works

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 and FF placement. By re-using macro placement from an implementation run that meets timing, you can reduce some of the variability from one implementation run to the next. This allows the implementation software to find a placement that meets timing, then 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 suggest a placement for the other gates. Timing may be more stable and, in some cases, implementation run times may 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). If multiple runs meet timing, start with the run that has the shortest implementation run time.

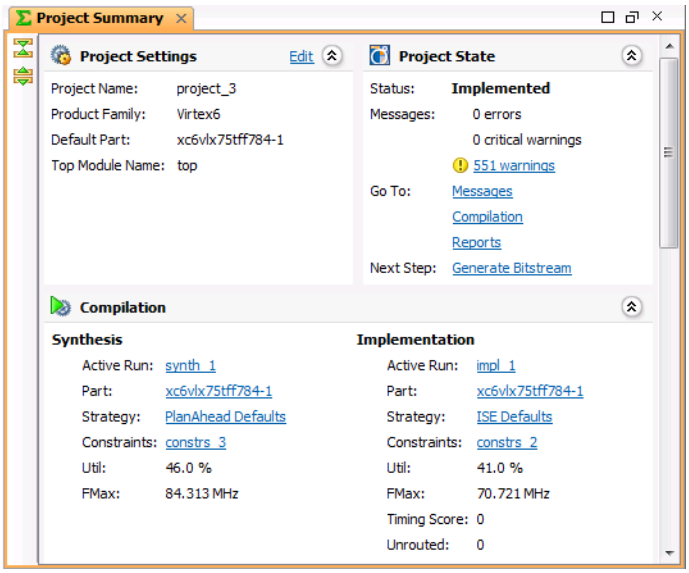

Figure 2-1: **Project Summary**

Using an Implementation Run

You can use an implementation run in:

- Scripts
- Project Navigator
- The PlanAhead™ software

Load the design that meets timing into the PlanAhead software to constrain placement.

Viewing Implementation Placement

To see where the implementation software placed the gates:

1. Run **Project Navigator > Analyze Timing/Floorplan Design**, or
2. Select **PlanAhead > Flow Navigator > Implement Design** to open the implemented design, or
3. If implementation was run in stand-alone scripts:
   a. Create a new PlanAhead project
   b. Select **New Project wizard > Import ISE Place and Route Results**.

Figure 2-2: **Viewing Implementation Placement**

Re-Using the Placement

When the design meets timing, you can re-use the placement. Avoid fixing 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 only the block RAM and DSP48 helps maintain timing as other gates change.

Searching for Primitives

The PlanAhead software allows you to easily find all:

- Block Memory (RAMB and FIFO primitives)
- Block Arithmetic (MULT and DSP primitives)

To search for these primitives, select **Edit > Find** as shown in Figure 2-3, 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 needed for a seed must be isolated from the other placement.

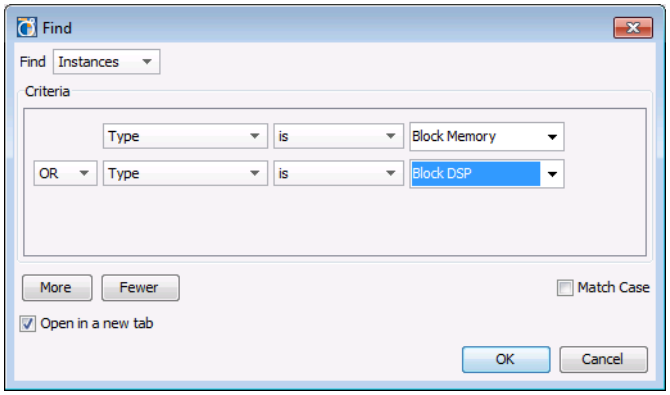

Figure 2-3: **Searching for the Memory and Arithmetic Blocks**

Fixed and Unfixed Placement

The PlanAhead software has two types of placement, Fixed and Unfixed.

	Fixed Placement	Unfixed Placement
Created From	Placed from a User Constraints File (UCF), or	Back-annotated from the implementation software
	Designated by the user in the PlanAhead software	
Reused	Yes	No

To fix the logic:

1. Select the placed objects to be fixed.
2. Right-click.
3. Select **Fix Instances**.

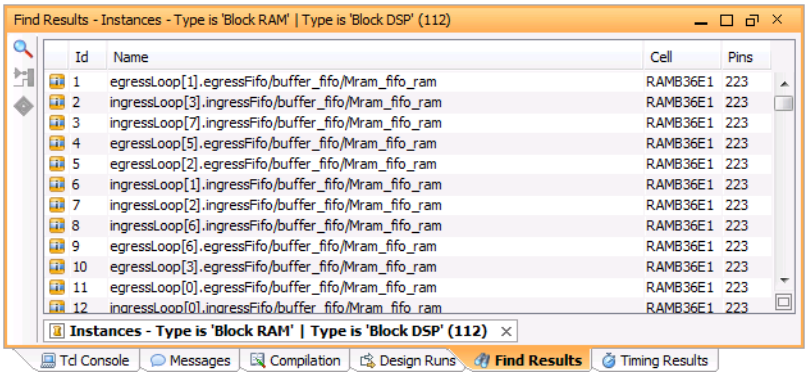

Figure 2-4: **Selecting Logic in the Find Results Dialog Box**

Analyzing and Modifying Placement

The placed logic in the Device view changes color to denote the change in how the software handles the placement.

The UCF is updated with the new constraints only after a save.

The UCF now has multiple gate level constraints in 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, re-run the placement to update the references defined in the LOC constraints.
- If the macros or the logic around the macros change, clear and rerun the placement.
- If the design regularly fails timing:
  - Run PAR without the LOC constraints on the macros.
  - Tweak placement of individual block RAM or DSP48 (advanced users).

For more information on analyzing and modifying placement, see:

- Chapter 10, "Analyzing the Implementation Results" in the *PlanAhead User Guide (UG632)* referred to in Appendix A, Additional Resources.
- Chapter 11, "Floorplanning the Design" in the *PlanAhead User Guide (UG632)* referred to in Appendix A, Additional Resources.

Hierarchical Floorplanning Flow

The Hierarchical Floorplanning Flow:

- Is more powerful than the Re-Use Flow.
- Can close timing on a design that has never met timing.
- Can suggest design and logic changes to meet timing more easily and consistently.

Both the Hierarchical Floorplanning Flow and the Re-Use Flow can significantly impact timing.

The Hierarchical Floorplanning Flow has the following advantages:

- Resists design change.
- Can close timing.
- Can bring consistency.

The Hierarchical Floorplanning Flow has the following disadvantages:

- Requires significant engineering time.
- May require iterations.

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

Designs That Have Not Met Timing

Hierarchical floorplanning is the best flow for closing timing in a design that has not met timing. It allows you to:

- Take smaller levels of hierarchy.
- Constrain the hierarchy to a region on the chip.
- Use that as a guide to implementation.

This involves more work than a design that meets timing.

Implementation:

- Has comprehensive knowledge of the critical paths and the structure of the chip.
- Generally does a good job of the fine grain placement.
- 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.

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

These lines define the shape on the chip, and what to place into it.

You can:

- Set up a region that does not constrain all these ranges.
- 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.

Chapter 3

Using Floorplanning for Timing Closure

When creating a floorplan, keep the following questions in mind:

- What are the timing failures?
- What is the critical hierarchy?
- Are changes to floorplanning or logic alone enough 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
- Understanding the pinout and the design, as described in the following example of a design walkthrough.

Place and Route Results

Only post-implementation timing numbers can identify which logic is failing timing. If the design stills fails timing after it has been run through implementation, load the results into the PlanAhead software. The placement results, timing results, and gates can all be seen in one place. Select multiple critical paths and view the placement to obtain ideas for troubleshooting. See Figure 3-1, Placement of Paths Failing Timing.

Figure 3-1: **Placement of Paths Failing Timing**

Figure 3-1 shows that the block RAMs with critical paths are spread out over more of the chip than necessary. Use floorplanning to generate a tighter placement. The timing problem occurs in the paths between block RAM. These paths are good candidates for floorplanning.

Timing Results

In order to close timing, analyze the paths between block RAM to determine whether to:

- Floorplan
- Change logic
- Both floorplan and change logic

The path delay for the above critical path shows two nets with long route delays. See Figure 3-2, Detailed Data Path. The path is failing timing by over 837 ps. The third net has 1.177 ns route delay. The fourth net has 0.958 ns route delay. The route delay can be reduced with improved placement.

Data Path

Delay Type	Delay	Cumulative	Location	Logical Resource
RAMB36E1 (Trcko_DOB)	2.073	2.073	RAMB36_X2Y13	usbEngine0/usb_dma_wb_in/buffer_fifo/Mram_fifo_ram
net (fanout=1118)	0.886	2.959		usbEngine0/ma_adr[2]
LUT3 (Tilo)	0.196	3.155	SLICE_X30Y60	usbEngine0/u4/Mmux_adr[6]_dout[31]_wide_mux_97_OUT1817
net (fanout=1)	0.236	3.391		usbEngine0/u4/Mmux_adr[6]_dout[31]_wide_mux_97_OUT1816
LUT6 (Tilo)	0.068	3.459	SLICE_X30Y60	usbEngine0/u4/Mmux_adr[6]_dout[31]_wide_mux_97_OUT1818
net (fanout=1)	1.177	4.636		usbEngine0/u4/Mmux_adr[6]_dout[31]_wide_mux_97_OUT1817
LUT3 (Tilo)	0.205	4.841	SLICE_X30Y39	usbEngine0/u4/Mmux_adr[6]_dout[31]_wide_mux_97_OUT1819
net (fanout=1)	0.958	5.799		usbEngine0/u4/Mmux_adr[6]_dout[31]_wide_mux_97_OUT1818
LUT5 (Tas)	0.070	5.869	SLICE_X29Y57	usbEngine0/u4/Mmux_adr[6]_dout[31]_wide_mux_97_OUT1820
FDE	0.000	5.869	SLICE_X29Y57	usbEngine0/u4/dout_14
Total	5.869	5.869 Logic: 2.612 Net: 3.257		

Figure 3-2: **Detailed Data Path**

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

Gates and Hierarchies

You can floorplan gates through individual LOC and placement constraints. Xilinx® does not recommend moving the gates by hand to improve timing for the following reasons:

- Identifying and placing the gates is slow and difficult.
- If the logic in the gate floorplan changes, the floorplan must be redone.

Ask instead, What hierarchy is timing critical? Implementation reports timing problems for **usbEngine1** in Figure 3-2, Detailed Data Path. 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 to floorplan.

Load the critical paths into the schematic. In Figure 3-3, Gates and Hierarchy in the Critical Path, the schematic shows:

- The gates involved in the critical path, and
- The hierarchy in which 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.

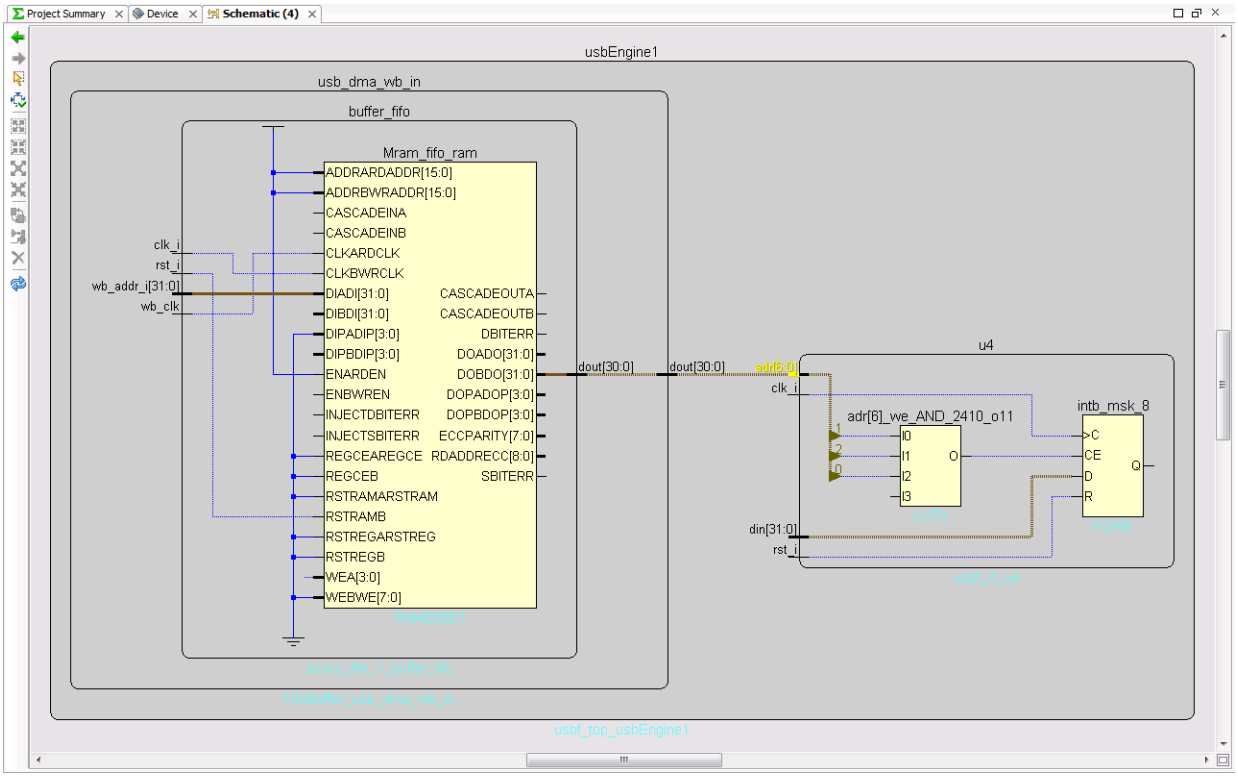

Figure 3-3: **Gates and Hierarchy in the Critical Path**

The floorplan should constrain at least the timing critical paths involving block RAMs inside each **usbEngine**. So far, both **usbEngine** blocks appear to be good candidates for

floorplanning. However, if **usbEngine** block is a large portion of the chip, try to floorplan the levels of sub-hierarchy that contain the critical path.

Critical and Non-Critical Hierarchies

To determine which gates should be floorplanned, look at the placement in the Device view. In Figure 3-4, Critical and Non-Critical Parts of a usbEngine:

- The gates in the *critical* sub-hierarchies are colored red.
- The gates in the *non-critical* sub-hierarchies are colored green.

Figure 3-4 shows that:

- In the *critical* hierarchies, there is high utilization of the block RAM.
- The *non-critical* hierarchies contain considerable LUT and FF logic that can be placed between the block RAM.
- The entire hierarchy is approximately 20% of the design.

Figure 3-4: **Critical and Non-Critical Parts of a usbEngine**

Before floorplanning **`usbEngine1`**, examine the pinout and design connectivity. The design may show that **`usbEngine1`** is not a good candidate

Confirming Good Candidates

The next step is to:

- Confirm that **usbEngine1** is a good candidate for floorplanning
- Determine where it should be placed.
- Create a top-level floorplan on the device (optional).

The top-level floorplan can suggest which logic is influencing the placement of other logic. Blocks spread out across the chip are bad candidates for floorplanning.

In Figure 3-5, Top-Level Floorplan for Analysis:

- IO connectivity is displayed as green IO lines.

  Look for the lines going from the middle IO bank on the left side of the chip to the block in the middle towards the bottom of the device.

- Connectivity between hierarchical blocks is displayed as bundles of nets between the placed hierarchies. The block highlighted in yellow:
  - Communicates to most other blocks.
  - Is not a good candidate for floorplanning because it has to spread around the chip.

  You can see at a glance that there are many inter-connected hierarchies. You can see when a pinout draws a hierarchy across the chip.

Figure 3-5 shows the top-level floorplan for this design. 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 the middle towards the bottom of the device) should be placed in the upper left corner of the device.

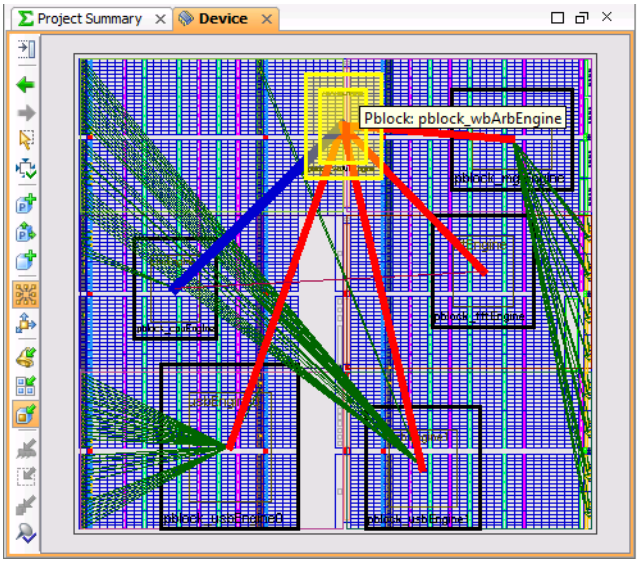

Figure 3-5: **Top-Level Floorplan for Analysis**

Shaping the Floorplan for the Critical Hierarchy

The floorplan suggests that 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 that 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**
- **usbEngine0**

Implementation showed a timing problem in **usbEngine0**. The same timing problem will probably occur in **usbEngine1** as well.

To deal with this problem, Xilinx recommends that you:

- Solve the timing problems of each block separately.
- Consider the USB blocks as separate timing critical hierarchies.
- Floorplan each hierarchy separately.

A final floorplan that meets timing is shown in Figure 3-6, First Pass Floorplan.

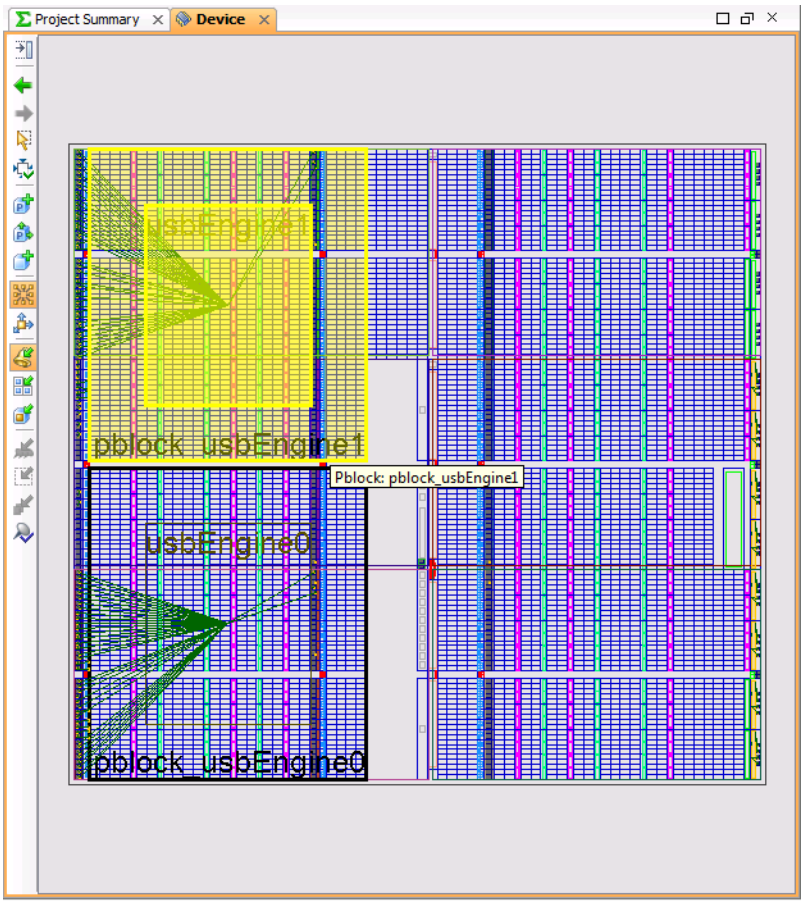

Figure 3-6: **First Pass Floorplan**

Constraining Subsets of the Netlist Hierarchy

The PlanAhead software 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 become AREA_GROUP constraints in the User Constraints File (UCF) in order to guide implementation and confine the levels 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.

You can:

- Set up a region that does not constrain all these ranges.
- 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.

Chapter 4

Floorplanning Iteratively

Xilinx recommends that you floorplan iteratively, and that you follow these guidelines:

- When it is not obvious which hierarchy to floorplan, use trial and error until timing improves.
- If timing degrades in the blocks that are floorplanned, determine why. The design may have connections that are not immediately obvious.
- After the first floorplan, revise the floorplan if necessary.
- Save each floorplan in case you want to revisit your work later.
- Keep things simple. A simple approach generally works better and takes less time.
- If critical paths are located within logic that is not floorplanned:
  - Identify the levels of hierarchy that contain the critical paths.
  - Assign them to a new Pblock.
  - Place the Pblock on the chip.
  - Keep this Pblock for place and route if the placement is reasonable.
- 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 an unconstrained hierarchy, add the unconstrained logic to a Pblock.
  - Create a new Pblock to hold the critical path and place it nearby, or
  - If the unconstrained logic is small, create a Pblock to hold both the critical path and the unconstrained logic.
- If critical paths are between two Pblocks, consider doing the following:
  - Move or reshape the Pblocks so they are closer.
  - Embed one Pblock inside the other.
  - Move logic from one Pblock to the other.
- 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. Begin with the timing critical hierarchy that has a good connectivity. 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, but still consistently fail timing, consider removing the floorplanning constraints. If timing still fails to improve, 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.

Appendix A

Additional Resources

Xilinx Resources

- ***ISE® Design Suite: Installation and Licensing Guide*** *(UG798)*: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/iil.pdf
- ***ISE Design Suite 13: Release Notes Guide*** *(UG631)*: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/irn.pdf
- **Xilinx® Documentation**: http://www.xilinx.com/support/documentation.htm
- **Xilinx Global Glossary**: http://www.xilinx.com/support/documentation/sw_manuals/glossary.pdf
- **Xilinx Support**: http://www.xilinx.com/support.htm
- **Xilinx Data Sheets:** http://www.xilinx.com/support/documentation/data_sheets.htm

ISE Documentation

- **ISE Documents**: http://www.xilinx.com/support/documentation/dt_ise13-1.htm
  - ***Command Line Tools User Guide*** *(UG628)*: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/devref.pdf
  - ***Constraints Guide*** *(UG625)*: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/cgd.pdf
  - ***Xilinx Synthesis and Simulation Design Guide*** *(UG626)*: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/sim.pdf
  - ***XST User Guide for Virtex-4, Virtex-5, Spartan-3, and Newer CPLD Devices*** *(UG627)*: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/xst.pdf
  - ***XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices*** *(UG687)*: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/xst_v6s6.pdf

PlanAhead Documentation

- **PlanAhead Documentation**: http://www.xilinx.com/support/documentation/dt_planahead_planahead13-1_userguides.htm
- **Methodology Guides**: http://www.xilinx.com/support_documentation

  - ***Floorplanning Methodology Guide*** *(UG633)*:
    http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/Floorplanning_Methodology_Guide.pdf
  - ***Hierarchical Design Methodology Guide*** *(UG748)*:
    http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/Hierarchical_Design_Methodology_Guide.pdf
  - ***Pin Planning Methodology Guide*** *(UG792)*:
    http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/ug792_pinplan.pdf
- ***PlanAhead User Guide*** *(UG632)*:
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/PlanAhead_UserGuide.pdf
- ***PlanAhead Tcl Command Reference Guide*** *(UG789)*:
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/ug789_tcl_commands.pdf

- **PlanAhead Tutorials:**
  http://www.xilinx.com/support/documentation/dt_planahead_planahead13-1_tutorials.htm
  - ***Design Analysis and Floorplanning*** *(UG676)*:
    http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/PlanAhead_Tutorial_Design_Analysis_Floorplan.pdf
  - ***I/O Pin Planning*** *(UG674)*:
    http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/PlanAhead_Tutorial_IO_Pin_Planning.pdf
  - ***Leveraging Design Preservation for Predictable Results (UG747):***
    http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/PlanAhead_Tutorial_Design_Preservation.pdf