

# Floorplanning Methodology Guide

UG633 (v14.5) April 10, 2013



#### Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at <http://www.xilinx.com/warranty.htm>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: <http://www.xilinx.com/warranty.htm#critapps>.

© Copyright 2009-2013 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. All other trademarks are the property of their respective owners.

## Revision History

The following table shows the revision history for this document.

Date	Version	Revision
4/10/2013	14.5	Minor updates throughout.
4/24/2012	14.1	Updated figures. Made minor language edits.

# Table of Contents

---

Revision History .....	2
<b>Chapter 1: Floorplanning Overview</b>	
About Floorplanning .....	5
Timing Closure .....	6
Floorplanning Basics .....	8
Floorplanning Considerations .....	11
Working With Hierarchical Netlists .....	11
Logic Synthesis Recommendations .....	12
Increasing Consistency and Other Benefits of Floorplanning .....	12
Using Clock Resources to Guide Floorplanning .....	13
<b>Chapter 2: Floorplanning Flows</b>	
Re-Use Flow (Design Meets Timing Some of the Time) .....	15
Hierarchical Floorplanning Flow (Design Has Never Met Timing) .....	20
<b>Chapter 3: Using Floorplanning for Timing Closure</b>	
Floorplanning Questions .....	23
Place and Route Results .....	23
Timing Results .....	25
Gates and Hierarchies .....	25
Shaping the Floorplan for the Critical Hierarchy .....	28
Deciding What Else Should Be Floorplanned .....	29
<b>Chapter 4: Floorplanning Iteratively</b>	
General Recommendations .....	31
Revise Critical Paths .....	31
Improve Timing in Critical Hierarchies .....	32
<b>Appendix A: Additional Resources</b>	
Xilinx Resources .....	33
ISE Documentation .....	33
PlanAhead Documentation .....	34



# ***Floorplanning Overview***

---

This chapter provides an overview of floorplanning.

## **About Floorplanning**

Floorplanning allows you to:

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

## **Goals of Floorplanning**

The goals of floorplanning are to:

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

## **Benefits of Floorplanning**

Floorplanning can:

- Improve performance.
- Enable a placed and routed design to meet timing.
- Help you achieve:
  - Higher system clock frequency
  - Shorter implementation run times
  - Greater consistency in timing
  - All of these benefits together

## **Limitations of Floorplanning**

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.

## When To Floorplan

Consider floorplanning when a design:

- Does not meet timing consistently.

See [Re-Use Flow \(Design Meets Timing Some of the Time\)](#) in Chapter 2, Floorplanning Flows.

OR

- Has never met timing.

See [Hierarchical Floorplanning Flow \(Design Has Never Met Timing\)](#) in Chapter 2, Floorplanning Flows.

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

During implementation, the software:

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

Floorplanning can reduce path delays, leading to timing closure. As a first step in floorplanning, ensure that the timing constraints are accurate.

## Determining 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 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 Closure User Guide (UG612)* cited in [Appendix A, Additional Resources](#).

## Clock Jitter and Clock-to-Clock Skew

The allowed time is modified by:

- Clock jitter
- 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.

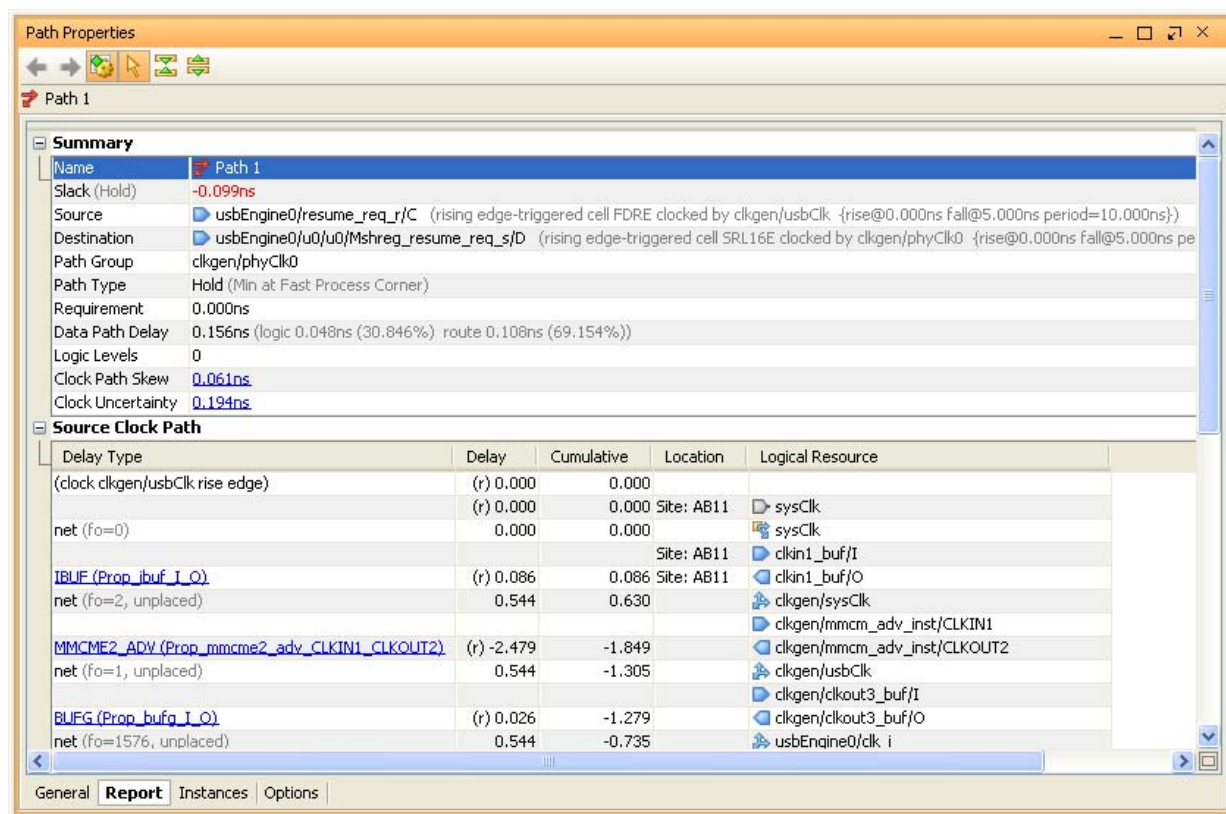


Figure 1-1: Example Timing Report

## 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

Following are some basic principles of floorplanning.

### Reducing Route Delay

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 route 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.

### Incremental Design

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” in the *Hierarchical Design Methodology Guide (UG748)* cited in [Appendix A, Additional Resources](#).

### Detailed Cell-Level Floorplanning

Detailed cell-level floorplanning, or hand-placing logic, places individual logic elements of a critical path on a specific site on the chip.

You can hand place some or all of the gates for a timing critical path. Hand-placed logic shows in orange in [Figure 1-2, page 9](#).

Consider detailed cell-level floorplanning only as a last resort.

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





Figure 1-2: Floorplanned Logic by Hand

## Hierarchical Floorplanning

Use hierarchical floorplanning to constrain levels of hierarchy to specific regions on the chip.

Xilinx recommends hierarchical floorplanning instead of cell-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

## 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

Following are some considerations into account during floorplanning.

## Floorplanning is Often Iterative

Floorplanning is often an iterative process. The first pass at a floorplan may address issues in one section, only to reveal that a different section is failing.

## Floorplanning Can Hurt Timing

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.

## Use Multiple Trials and Notes

Multiple trials and notes about the design can help you create a working floorplan.

## Floorplan Timing-Critical Logic

When you initially floorplan a design:

- 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 floorplanning the entire design.

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

1. 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.
2. 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.
3. Timing can be met more easily if you understand how the design will be spread out on the chip when you construct the hierarchy.
4. 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.
5. 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.
6. You can duplicate the register by hand, in the RTL source file, 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

Follow these logic synthesis recommendations:

1. 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.
2. Register the outputs of all the modules to reduce the number of modules involved in a critical path.
3. Replicate the drivers of nets that are separated on the die. Synthesis may need an attribute to preserve logically equivalent logic.
4. Long paths in single large hierarchical block can make floorplanning difficult. Because it is easier to work with smaller hierarchical blocks, Xilinx recommends dividing large hierarchical blocks in the RTL.
5. Intermingled critical paths can be difficult to floorplan. Divide large critical blocks into blocks that are smaller and easier to isolate.
6. 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 because global optimizations across hierarchy are disabled. Consider this trade-off before using incremental RTL synthesis methodology.

7. 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 and Other Benefits of Floorplanning

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)* cited 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 device clock rules 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.



# ***Floorplanning Flows***

---

Xilinx® supports the following floorplanning flows:

- [Re-Use Flow \(Design Meets Timing Some of the Time\)](#)
- [Hierarchical Floorplanning Flow \(Design Has Never Met Timing\)](#)

Both flows can significantly impact timing.

## **Re-Use Flow (Design Meets Timing Some of the Time)**

The Re-Use Flow can close timing on a design that meets timing some of the time. You can re-use some block RAM and DSP48 component 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 source of timing variability is the macro placement, such as block RAM and DSP48 components. 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 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, and
- The names and structures for the macros do not change.

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

Start with an implementation run that routes and meets timing. Look 0 unrouted, and a Timing Score of 0 in the Design Runs view. If multiple runs meet timing, use the implementation run that has the shortest elapsed time.

Name	Part	Constraints	Strategy	Status	Progress	Elapsed	Util (%)	FMax (MHz)	Timing Score
✓ synth_1	xc7k70tfgb676-2	constrs_2	PlanAhead Defaults (XST 14)	XST Complete!	<div><div></div></div> 100%	00:04:35	53.000	113.608	
✓ impl_1 (active)	xc7k70tfgb676-2	constrs_2	ISE Defaults (ISE 14)	PAR Complete!	<div><div></div></div> 100%	00:18:35	48.000	51.424	0
✓ impl_2	xc7k70tfgb676-2	constrs_2	MapTiming (ISE 14)	PAR Complete!	<div><div></div></div> 100%	00:32:35	48.000	50.216	0
✓ impl_3	xc7k70tfgb676-2	constrs_2	MapGlobalOptParHigh (ISE 14)	PAR Complete!	<div><div></div></div> 100%	00:32:28	48.000	50.803	0
✓ impl_4	xc7k70tfgb676-2	constrs_2	MapLogicOptParHighExtra (ISE 14)	PAR Complete!	<div><div></div></div> 100%	00:44:52	48.000	50.216	0

Figure 2-1: Design Runs view

## Using an Implementation Run

You can use an implementation run in:

- Scripts
- ISE® 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** and select **Tools > PlanAhead > Analyze Timing/Floorplan Design (PlanAhead)**, or
2. Launch PlanAhead and from **Flow Navigator** select **Implementation > Open Implemented Design** to open the implemented design, or
3. If implementation was run in stand-alone scripts:
  - a. Create a new project in the PlanAhead tool, and
  - b. Select **Import ISE Place and Route Results** as the Project Type in the New Project dialog box.



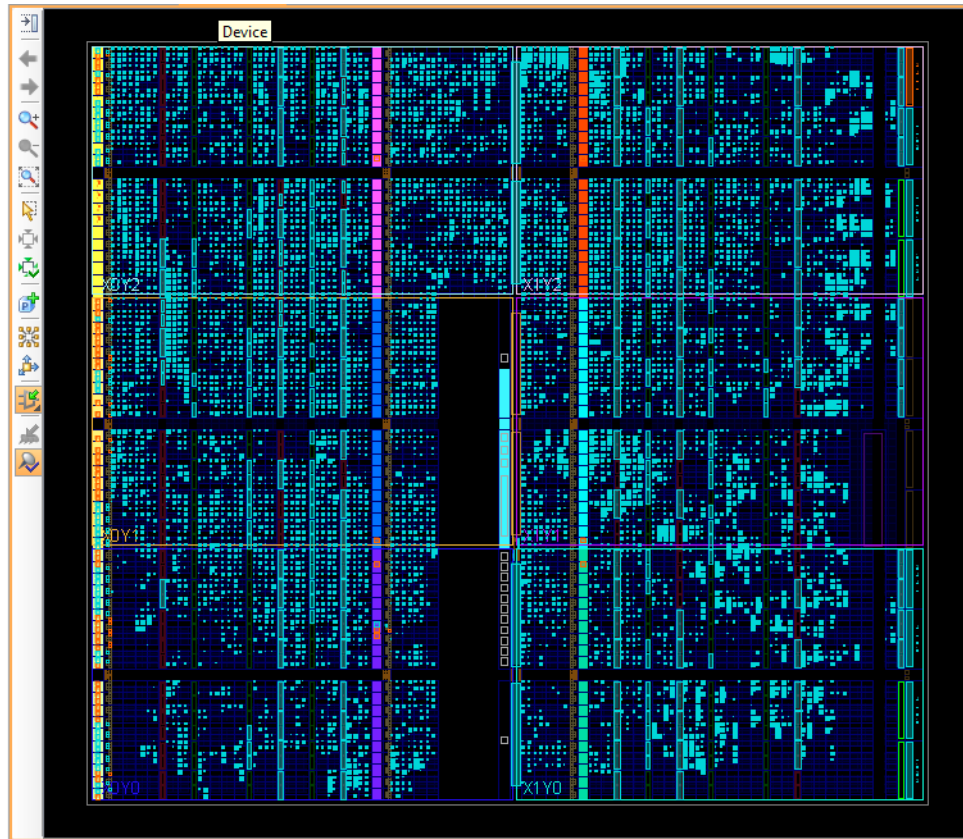


Figure 2-2: Viewing Implementation Placement

## Re-Using the Placement

When the design meets timing, you can re-use the placement. Do not fix everything in place, because 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 primitives 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**. See the following figure.

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.

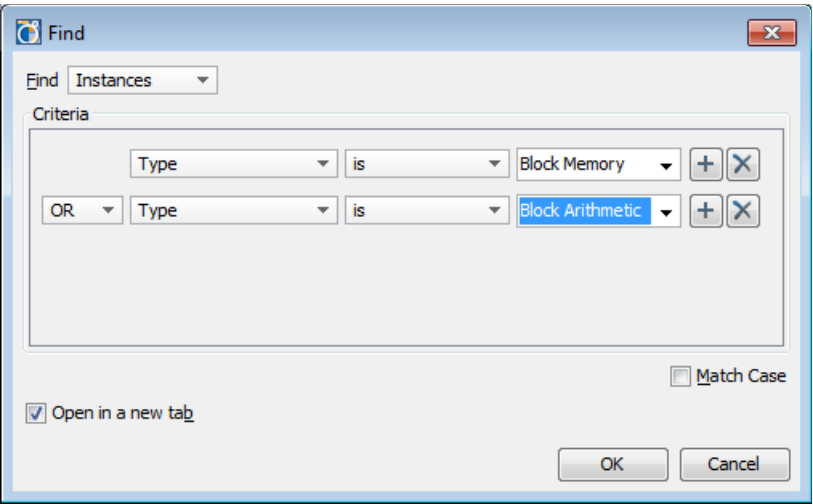


Figure 2-3: Searching for the Memory and Arithmetic Blocks

# Fixed and Unfixed Placement

The PlanAhead software has two types of placement:

- Fixed Placement
- Unfixed Placement

Table 2-1: Placement Types

	Fixed Placement	Unfixed Placement
<b>Created From</b>	<ul style="list-style-type: none"> <li>Placed from a User Constraints File (UCF), or</li> <li>Hand-placed by the user in the PlanAhead software, or</li> <li>Mark “fixed” by the user.</li> </ul>	<ul style="list-style-type: none"> <li>Placed by the PAR implementation software.</li> </ul>
<b>Reused</b>	Yes	No

# Fixing the Logic

To fix the logic:

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

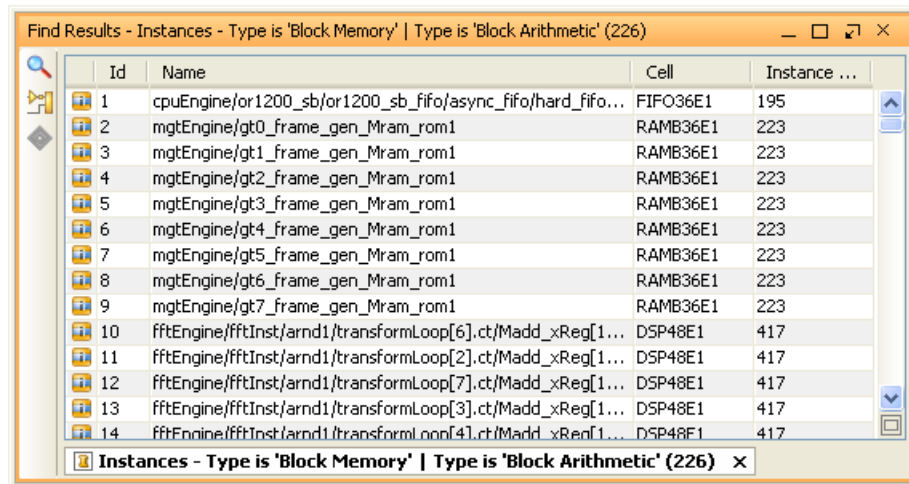


Figure 2-4: Selecting Logic in the Find Results Dialog Box

## Analyzing and Modifying Placement

The placed logic in the Device view changes color to show 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 components (advanced users).

For more information on analyzing and modifying placement, see:

- Chapter 10, “Analyzing the Implementation Results” in the *PlanAhead User Guide (UG632)* cited in [Appendix A, Additional Resources](#).
- Chapter 11, “Floorplanning the Design” in the *PlanAhead User Guide (UG632)* cited in [Appendix A, Additional Resources](#).

## Hierarchical Floorplanning Flow (Design Has Never Met Timing)

The Hierarchical Floorplanning Flow is more powerful than the [Re-Use Flow \(Design Meets Timing Some of the Time\)](#). This flow can:

- Close timing on a design that has never met timing.
- Suggest design and logic changes to meet timing more easily and consistently.
- 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.

### Advantages and Disadvantages of Hierarchical Floorplanning Flow

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.

### Using Hierarchical Floorplanning Flow in Designs That Have Not Met Timing

Hierarchical floorplanning is the best flow for closing timing in designed that have not met timing.

Hierarchical floorplanning 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 I/Os must often be placed near their I/Os. 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 components 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.



# *Using Floorplanning for Timing Closure*

---

This chapter discusses using floorplanning for timing closure.

## **Floorplanning Questions**

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

1. What are the timing failures?
2. What is the critical hierarchy?
3. Are changes to floorplanning or logic alone enough to close timing?
4. Does anything else need to be floorplanned?
5. Can the critical hierarchies be floorplanned?
6. What should be placed where?

To answer these questions:

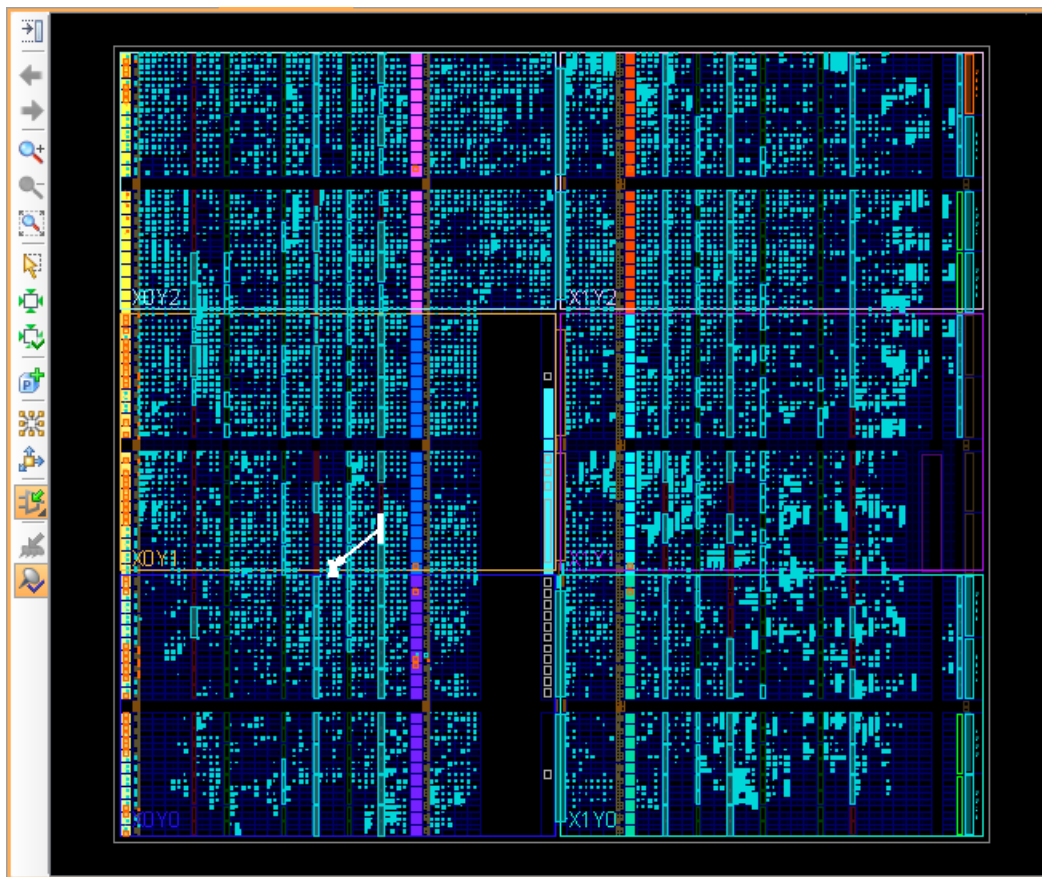
- Look at the timing paths, placement, and logic structure in the paths, and
- Understand the pinout and the design.

See the following example of a design walkthrough.

## **Place and Route Results**

Only post-implementation timing results can identify which logic is failing timing. If the design stills fails timing after implementation, load the results into the PlanAhead™ software.

The placement results, timing results, and gates can all be viewed in one place. To obtain ideas for troubleshooting, select multiple critical paths and view the placement See [Figure 3-1, Placement of Paths Failing Timing](#).



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

This figure shows that the block RAM components 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 from block RAM components. These paths are good candidates for floorplanning.



## Timing Results

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

- Floorplan, or
- Change logic, or
- 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 28 ps. The first net has 937 ps route delay. The route delay can be reduced with improved placement.

**Data Path**

Delay Type	Delay	Cumulative	Location	Logical Resource
RAMB36E1 (Trcke_DOB)	(r) 2.073	2.073	RAMB36_X2Y9	usbEngine0/usb_dma_wb_in/buffer_fifo/Mram_fifo_ram
net (fanout=23)	(r) 0.937	3.010		usbEngine0/ma_adr[14]
LUT4 (Tilo)	(r) 0.201	3.211	SLICE_X26Y41	usbEngine0/u5/state_rf_we_d1
net (fanout=17)	(r) 0.341	3.552		usbEngine0/rf_we
LUT5 (Tilo)	(r) 0.198	3.750	SLICE_X26Y41	usbEngine0/u4/adr[6]_we_AND_2411_o11
net (fanout=3)	(r) 0.406	4.156		usbEngine0/u4/N11
LUT4 (Tilo)	(r) 0.191	4.347	SLICE_X26Y41	usbEngine0/u4/adr[6]_we_AND_2415_o1
net (fanout=6)	(r) 0.388	4.735		usbEngine0/u4/adr[6]_we_AND_2415_o
EDRE (Tceck)	(r) 0.318	5.053	SLICE_X25Y40	usbEngine0/u4/intb_msk_6
<b>Total</b>	5.053	5.053		

**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.

To modify the gates in designs with a large percentage of logic delay:

- Change the code, or
- Update synthesis

## Gates and Hierarchies

You can floorplan gates through individual LOC and placement constraints.

Do not move the gates by hand to improve timing.

- 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 **usbEngine0** 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.

Trace the logic around the critical gates in the schematic to see how the non-critical logic is structured.

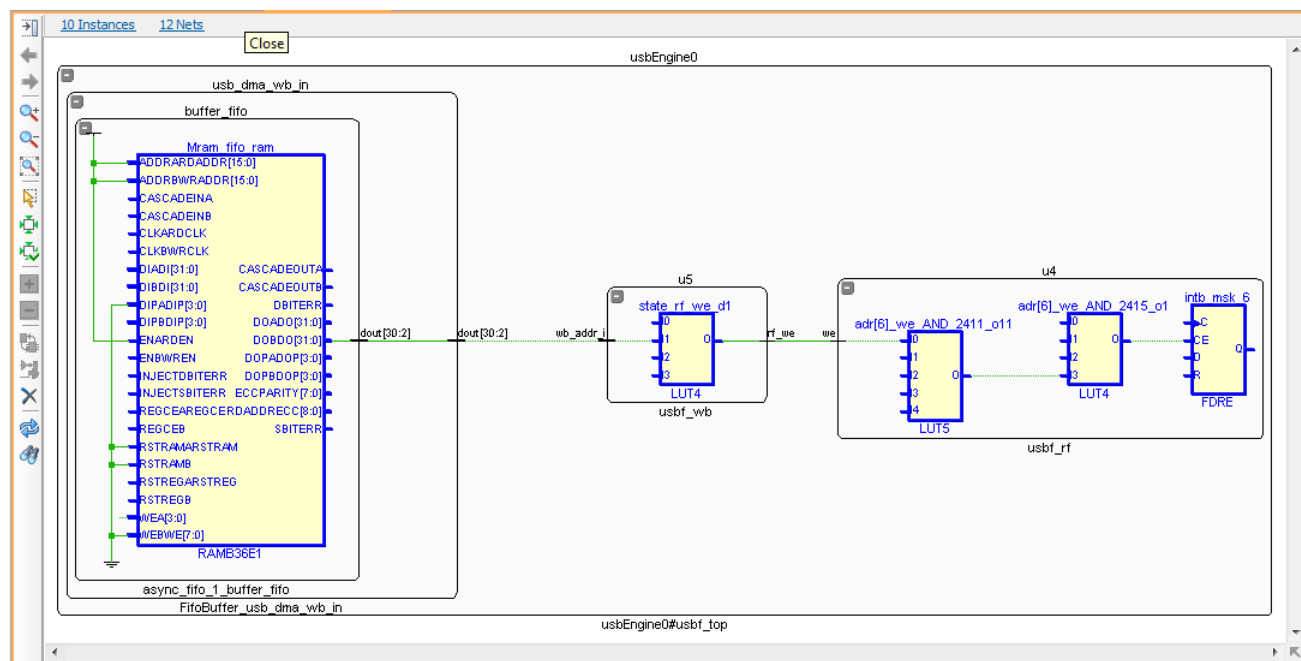


Figure 3-3: Gates and Hierarchy in the Critical Path

The floorplan should constrain at least the timing critical paths involving block RAM components 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 block RAM components.
- The *non-critical* hierarchies contain considerable LUT and FF logic that can be placed between the block RAM components.
- The entire hierarchy is approximately 20% of the design.

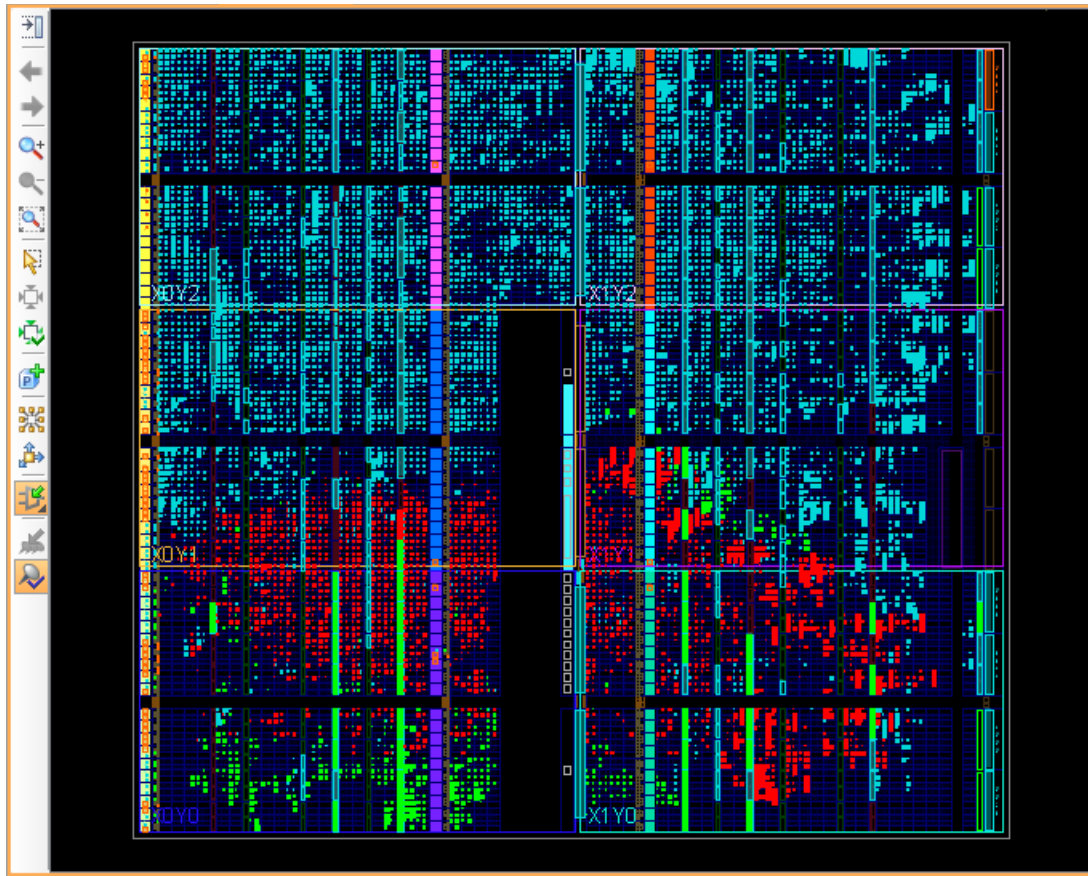


Figure 3-4: Critical and Non-Critical Parts of a usbEngine

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](#):

- I/O connectivity is displayed as **green** I/O lines.  
Look for the lines going from the middle I/O 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 is highlighted in white.
  - 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. The design hierarchy is spread around the chip. The pinout would support floorplanning **usbEngine1**. Based on the pinout, **usbEngine1** (at the bottom right of the device) should be placed in the upper left corner of the device.

**Note:** To view the net connections and the net bundles, as seen in Figure 3-5, you must enable the **Show I/O Nets** command on the Device view toolbar menu, and enable the display of Bundle Nets in the Device Options. See the *PlanAhead User Guide (UG632)*, as referenced in Appendix A, “Additional Resources,” for more information.

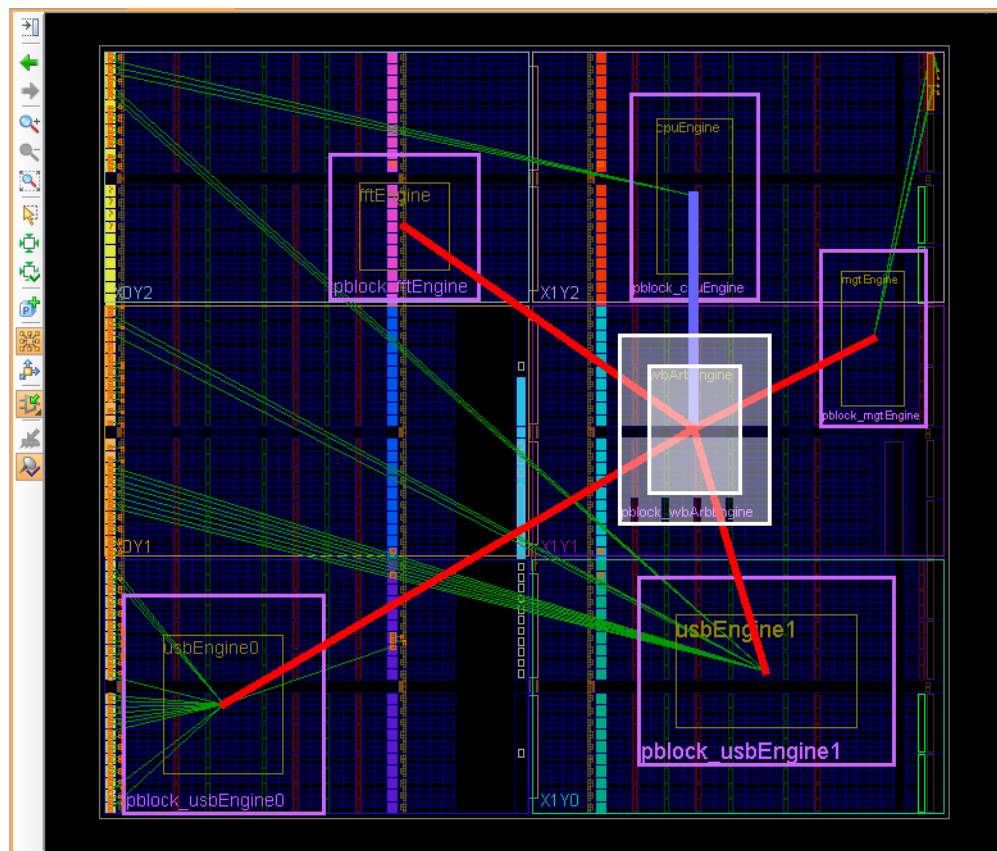


Figure 3-5: Top Level Floorplan for Analysis

## 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 I/O banks on the top left of the chip. It makes sense to try to floorplan the logic to use slices and block RAM components 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](#).

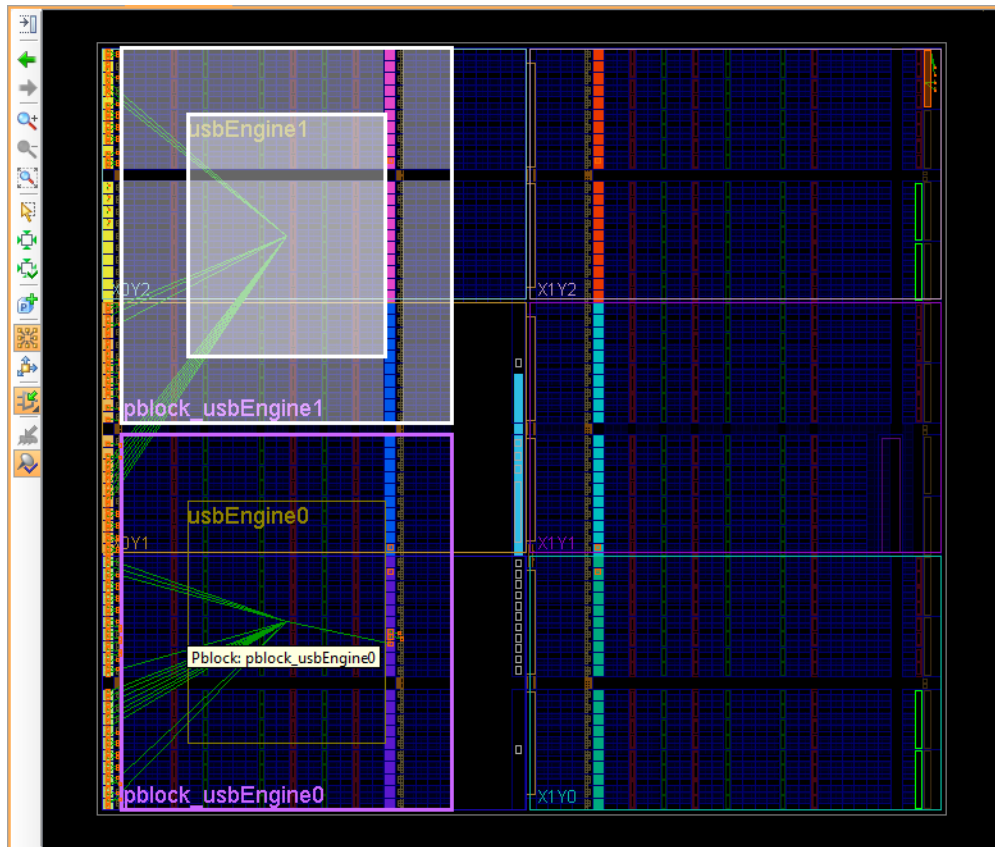


Figure 3-6: First Pass Floorplan

### 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 components 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***

---

Xilinx® recommends that you floorplan iteratively, and that you consider the options shown in the sections below. Not all options will help in all cases. Try several options where necessary to find the best solution for your design.

## **General Recommendations**

Following are some general recommendations for floorplanning iteratively.

### **Use Trial and Error**

When it is not clear which hierarchy to floorplan, use trial and error until timing improves.

### **Look for Hidden Connections**

If timing degrades in the floorplanned blocks, look for hidden connections. The design may have connections that are not immediately obvious.

### **Revise the Floorplan**

Revise the floorplan if necessary. Save each floorplan in case you want to revisit your work later.

### **Keep Floorplans Simple**

Keep floorplans simple. Simple floorplans usually work better and take less time than complicated floorplans.

### **Re-Run the Design After Upgrading**

After upgrading an ISE® Design Suite release, run the design through implementation unconstrained. A new release may make floorplanning unnecessary.

## **Revise Critical Paths**

The following sections contain suggestions for improving floorplanning based on the location of critical paths.

## Within Logic That is Not Floorplanned

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.

## Within a Single Pblock

If critical paths are *within a single Pblock*, revise the Pblock.

- Create a Pblock within the Pblock that contained the failing timing path to constrain the critical hierarchy more tightly, or
- Work with lower levels of hierarchy, remove some logic, and use a smaller Pblock.

## Between a Pblock and an Unconstrained Hierarchy

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.

## Between Two Pblocks

If critical paths are *between two Pblocks*:

- Move or reshape the Pblocks so they are closer.
- Embed one Pblock inside the other.
- Move logic from one Pblock to the other.

## Improve Timing in Critical Hierarchies

1. If the logic in a critical hierarchy is large, heavily interconnected, or being pulled around the chip by scattered loads, do not place it initially.
  - 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, revise the RTL, then re-synthesize.
2. If floorplanned sections still fail timing, remove the floorplanning constraints.
3. If timing still fails to improve, try a new approach.



# Additional Resources

---

## Xilinx Resources

- **Device User Guides:**  
[http://www.xilinx.com/support/documentation/user\\_guides.htm](http://www.xilinx.com/support/documentation/user_guides.htm)
- **Xilinx Glossary:** <http://www.xilinx.com/company/terms.htm>
- **ISE Design Suite 14: Release Notes, Installation, and Licensing (UG631)**  
<http://www.xilinx.com/cgi-bin/docs/rdoc?v=14.5;t=release+notes>
- **Product Support and Documentation:** <http://www.xilinx.com/support>

## ISE Documentation

- **Libraries Guides:**  
[http://www.xilinx.com/support/documentation/dt\\_ise14-5\\_librariesguides.htm](http://www.xilinx.com/support/documentation/dt_ise14-5_librariesguides.htm)
- **ISE Design Suite Documentation:**  
[http://www.xilinx.com/support/documentation/dt\\_ise14-5.htm](http://www.xilinx.com/support/documentation/dt_ise14-5.htm)
  - **Command Line Tools User Guide (UG628):**  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/devref.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/devref.pdf)
  - **Constraints Guide (UG625):**  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/cgd.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/cgd.pdf)
  - **Data2MEM User Guide (UG658):**  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/data2mem.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/data2mem.pdf)
  - **ISim User Guide (UG660):**  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/plugin\\_ism.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/plugin_ism.pdf)
  - **Synthesis and Simulation Design Guide (UG626):**  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/sim.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/sim.pdf)
  - **Timing Closure User Guide (UG612):**  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/ug612.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug612.pdf)
  - **Xilinx/Cadence PCB Guide (UG629):**  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/cadence\\_pcb.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/cadence_pcb.pdf)
  - **Xilinx/Mentor Graphics PCB Guide (UG630):**  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/mentor\\_pcb.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/mentor_pcb.pdf)
  - **XPower Estimator User Guide (UG440):**  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/ug440.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug440.pdf)
  - **XST User Guide for Virtex-4, Virtex-5, Spartan-3, and Newer CPLD Devices (UG627):**  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/xst.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/xst.pdf)

- *XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices (UG687):*  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/xst\\_v6s6.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/xst_v6s6.pdf)
- **ISE Methodology Guides:**
  - *Power Methodology Guide (UG786):*  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/ug786\\_PowerMethodology.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug786_PowerMethodology.pdf)
  - *Large FPGA Methodology Guide (UG872):* [http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/ug872\\_largefpga.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug872_largefpga.pdf)
- **ISE Tutorials:**
  - *ISE In-Depth Tutorial (UG695):*  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/ise\\_tutorial\\_ug695.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ise_tutorial_ug695.pdf)
  - *ISE RTL Technology Viewer Tutorial (UG685):*  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/ug685.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug685.pdf)
  - *ISim In-Depth Tutorial (UG682):*  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/ug682.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug682.pdf)
  - *Using Xilinx ChipScope Pro ILA Core with Project Navigator to Debug FPGA Applications (UG750):*  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/ug750.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug750.pdf)
  - *Xilinx Power Tools Tutorial (UG733):*  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/ug733.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug733.pdf)

## PlanAhead Documentation

- *PlanAhead User Guide (UG632):*  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/PlanAhead\\_UserGuide.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/PlanAhead_UserGuide.pdf)
- *Hierarchical Design Methodology Guide (UG748):*  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/Hierarchical\\_Design\\_Methodology\\_Guide.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/Hierarchical_Design_Methodology_Guide.pdf)
- *Pin Planning Methodology Guide (UG792):*  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/ug792\\_pinplan.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug792_pinplan.pdf)
- *PlanAhead Tcl Command Reference Guide (UG789):*  
[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_5/ug789\\_pa\\_tcl\\_commands.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug789_pa_tcl_commands.pdf)
- **PlanAhead Tutorials:**  
[http://www.xilinx.com/support/documentation/dt\\_planahead\\_planahead14-1\\_tutorials.htm](http://www.xilinx.com/support/documentation/dt_planahead_planahead14-1_tutorials.htm)
  - *Quick Front- to-Back Flow Overview (UG673)*
  - *I/O Pin Planning (UG674)*
  - *RTL Design and IP Generation (UG675)*
  - *Design Analysis and Floorplanning (UG676)*
  - *Debugging with ChipScope (UG677)*
  - *Team Design (UG839)*
  - *Design Preservation (UG747)*
  - *Partial Reconfiguration (UG743)*
  - *Reconfiguration with Processor Peripheral (UG744)*