Vivado Design Suite User Guide: Design Analysis and Closure Techniques (UG906)

Floorplanning

Floorplanning

This section discusses Floorplanning and includes:

About Floorplanning

Floorplanning can help a design meet timing. Xilinx recommends that you floorplan when a design does not meet timing consistently, or has never met timing. Floorplanning is also helpful when you are working with design teams, and consistency is most important.
Floorplanning can improve the setup slack (TNS, WNS) by reducing the average route delay. During implementation, the timing engine works on resolving the worst setup violations and all the hold violations. Floorplanning can only improve setup slack.
Manual floorplanning is easiest when the netlist has hierarchy. Design analysis is much slower when synthesis flattens the entire netlist. Set up synthesis to generate a hierarchical netlist. For Vivado synthesis use:

- `synth_design -flatten_hierarchy rebuilt`
  or
- The Vivado Synthesis Defaults strategy

Large hierarchical blocks with intertwined logical paths can be difficult to analyze. It is easier to analyze a design in which separate logical structures are in lower sub-hierarchies. Consider registering all the outputs of a hierarchical module. It is difficult to analyze the placement of paths that trace through multiple hierarchical blocks.

Understanding Floorplanning Basics

Not every design will always meet timing. You may have to guide the tools to a solution. Floorplanning allows you to guide the tools, either through high-level

hierarchy layout, or through detailed gate placement.
You will achieve the greatest improvements by fixing the worst problems or the most common problems. For example if there are outlier paths that have significantly worse slack, or high levels of logic, fix those paths first. The Reports > Timing > Create Slack Histogram command can provide a view of outlier paths. Alternatively, if the same timing endpoint appears in several negative slack paths, improving one of the paths might result in similar improvements for the other paths on that endpoint.
Consider floorplanning to increase performance by reducing route delay or increasing logic density on a non-critical block. Logic density is a measure of how tightly the logic is packed onto the chip.
Floorplanning can help you meet a higher clock frequency and improve consistency in the results. There are multiple approaches to floorplanning, each with its advantages and disadvantages.

Detailed Gate-Level Floorplanning

Detailed gate-level floorplanning involves placing individual leaf cells in specific sites on the device.

Advantages of Detailed Gate-Level Floorplanning

- Detailed gate-level floorplanning works with hand routing nets.
- Detailed gate-level floorplanning can extract the most performance out of the device.

Disadvantages of Detailed Gate-Level Floorplanning

- Detailed gate-level floorplanning is time consuming.
- Detailed gate-level floorplanning requires extensive knowledge of the device and design.
- Detailed gate-level floorplanning may need to be redone if the netlist changes.

✎ **Recommended:** Use detailed gate-level floorplanning as a last resort.

Information Reuse

Reuse information from a design that met timing. Use this flow if the design does not consistently meet timing. To reuse information:

1. Open two implementation runs:

  a. One for a run that is meeting timing.
  b. One for a run that is not meeting timing.

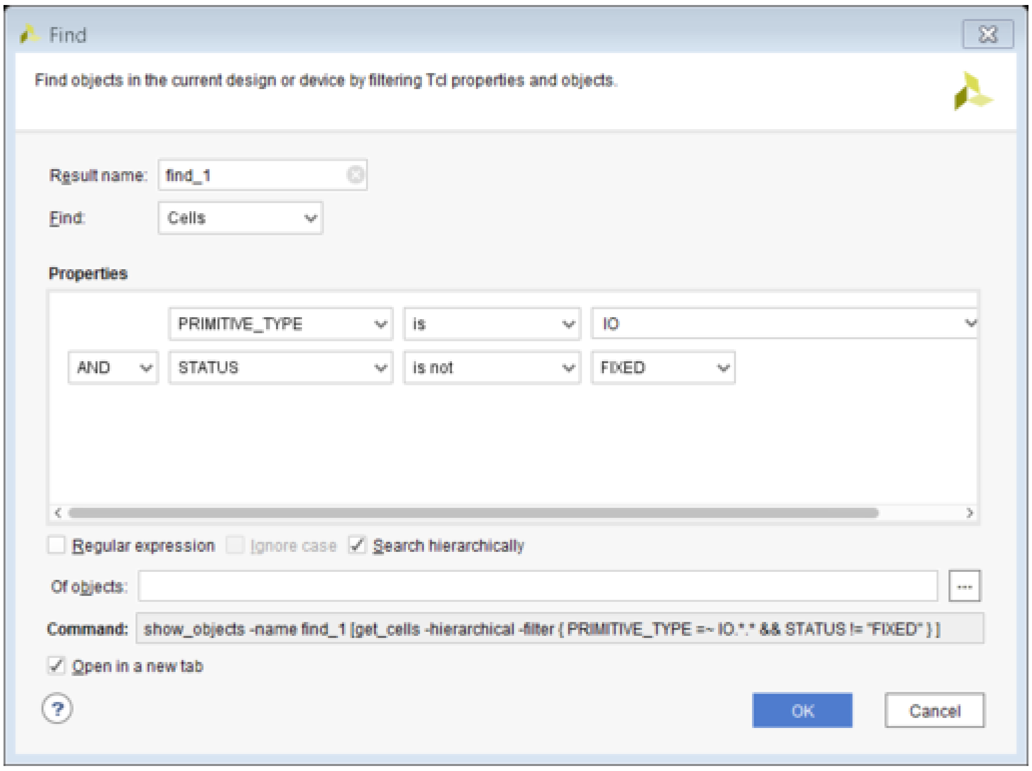
> ★ **Tip:** On a computer with multiple monitors, select Open Implementation in New Window to open a design in a new window.

2. Look for the differences between the two designs.
   a. Identify some failing timing paths from `report_timing_summary`.
   b. On the design that is meeting timing, run `report_timing` in `min_max` mode to time those same paths on the design that meets timing.
3. Compare the timing results:
   a. Clock skew
   b. Datapath delay
   c. Placement
   d. Route delays
4. If there are differences in the amount of logic delay between path end points, revisit the synthesis runs.

Review I/O and Cell Placement

Review the placement of the cells in the design. Compare two I/O reports to review the I/O placement and I/O standards. Make sure all the I/Os are placed. A simple search finds all I/Os without fixed placement as shown in the following figure.

Figure: I/O Is Not Fixed


Find
Find objects in the current design or device by filtering Tcl properties and objects.
Properties
Regular expression
Ignore case
Search hierarchically
OK
Cancel

Result name: find_1
Find: Cells

	PRIMITIVE_TYPE	is	IO
AND	STATUS	is not	FIXED

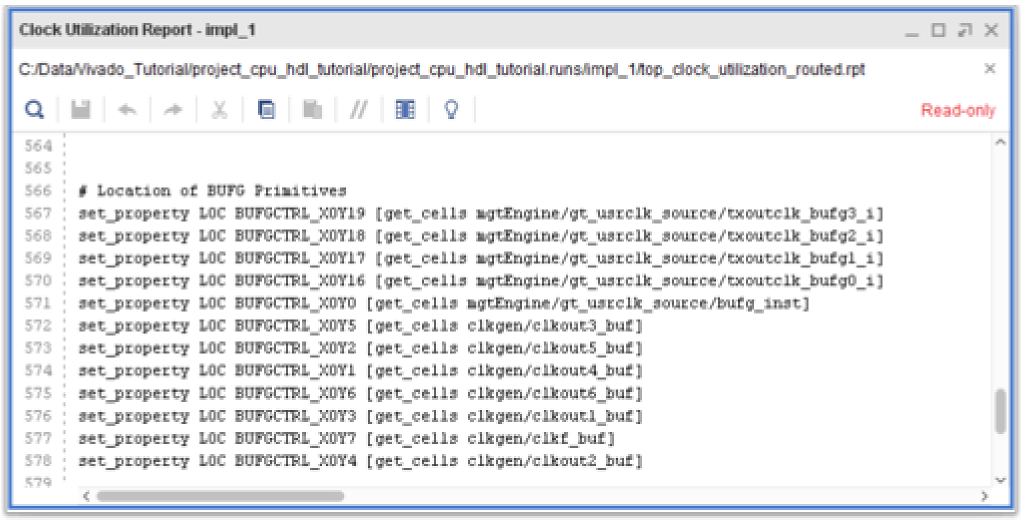
Of objects:

Command:
```
show_objects -name find_1 [get_cells -hierarchical -filter { PRIMITIVE_TYPE =~ IO.*.* && STATUS != "FIXED" } ]
```

Open in a new tab

If clock skew has changed between the runs, consider re-using the clock primitive placement from the run that met timing. The Clock Utilization Report lists the placement of the clock tree drivers, as shown in the following figure.

Figure: Clock Locations

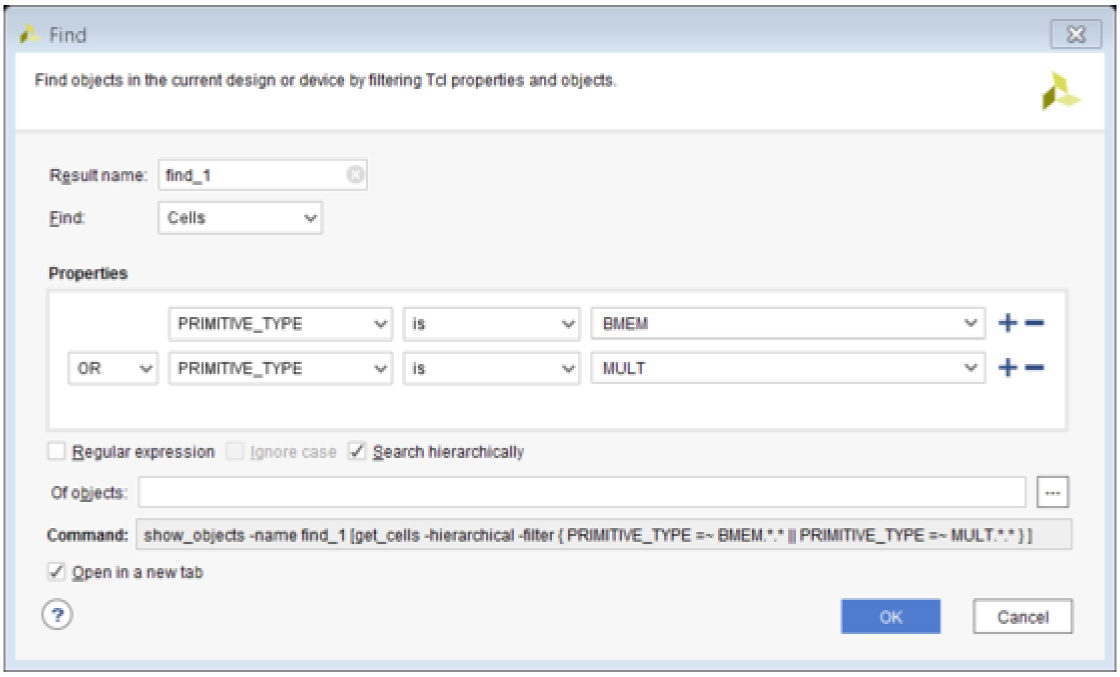

Clock Utilization Report - impl_1
C:/Data/Vivado_Tutorial/project_cpu_hdl_tutorial/project_cpu_hdl_tutorial.runs/impl_1/top_clock_utilization_routed.rpt
Read-only

```


# Location of BUFG Primitives
set_property LOC BUFGCTRL_X0Y19 [get_cells mgtEngine/gt_usrclk_source/txoutclk_bufg3_i]
set_property LOC BUFGCTRL_X0Y18 [get_cells mgtEngine/gt_usrclk_source/txoutclk_bufg2_i]
set_property LOC BUFGCTRL_X0Y17 [get_cells mgtEngine/gt_usrclk_source/txoutclk_bufg1_i]
set_property LOC BUFGCTRL_X0Y16 [get_cells mgtEngine/gt_usrclk_source/txoutclk_bufg0_i]
set_property LOC BUFGCTRL_X0Y0 [get_cells mgtEngine/gt_usrclk_source/bufg_inst]
set_property LOC BUFGCTRL_X0Y5 [get_cells clkgen/clkout3_buf]
set_property LOC BUFGCTRL_X0Y2 [get_cells clkgen/clkout5_buf]
set_property LOC BUFGCTRL_X0Y1 [get_cells clkgen/clkout4_buf]
set_property LOC BUFGCTRL_X0Y6 [get_cells clkgen/clkout6_buf]
set_property LOC BUFGCTRL_X0Y3 [get_cells clkgen/clkout1_buf]
set_property LOC BUFGCTRL_X0Y7 [get_cells clkgen/clkf_buf]
set_property LOC BUFGCTRL_X0Y4 [get_cells clkgen/clkout2_buf]
```

The LOC constraints can easily be copied into your XDC constraints file.
Many designs have met timing by reusing the placement of the Block RAMs and DSPs. Select Edit > Find to list the instances.

Figure: DSP or RAM



Find objects in the current design or device by filtering Tcl properties and objects.

Result name: find_1

Find: Cells

Properties

	PRIMITIVE_TYPE	is	BMEM
OR	PRIMITIVE_TYPE	is	MULT

☐ Regular expression ☐ Ignore case ☑ Search hierarchically

Of objects:

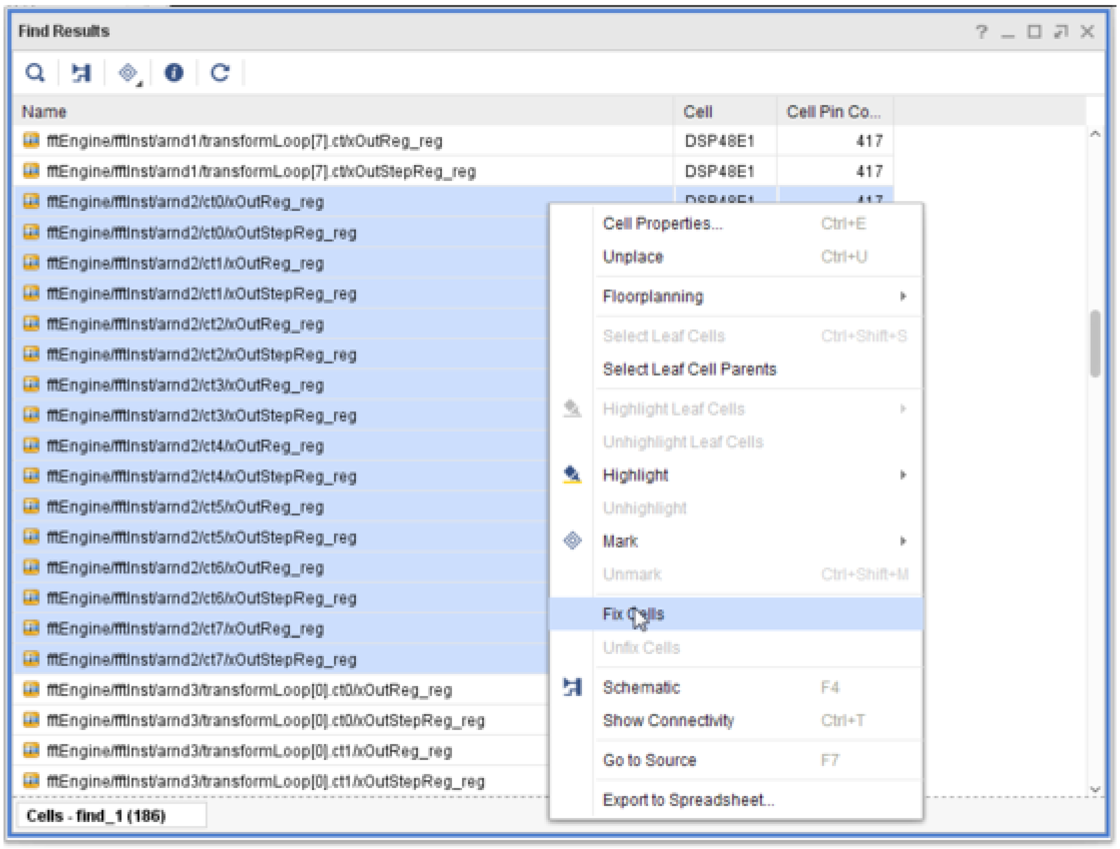
Command:
```
show_objects -name find_1 [get_cells -hierarchical -filter { PRIMITIVE_TYPE =~ BMEM.*.* || PRIMITIVE_TYPE =~ MULT.*.* } ]
```

☑ Open in a new tab

Adding Placement Constraints

Fix the logic to add the placement constraints to your XDC.

1. Select the macros from the find results.
2. Right click and select Fix Cells (shown in the following figure).



Name	Cell	Cell Pin Co...
fftEngine/fftInst/arnd1/transformLoop[7].ct/xOutReg_reg	DSP48E1	417
fftEngine/fftInst/arnd1/transformLoop[7].ct/xOutStepReg_reg	DSP48E1	417
fftEngine/fftInst/arnd2/ct0/xOutReg_reg	DSP48E1	417
fftEngine/fftInst/arnd2/ct0/xOutStepReg_reg		
fftEngine/fftInst/arnd2/ct1/xOutReg_reg		
fftEngine/fftInst/arnd2/ct1/xOutStepReg_reg		
fftEngine/fftInst/arnd2/ct2/xOutReg_reg		
fftEngine/fftInst/arnd2/ct2/xOutStepReg_reg		
fftEngine/fftInst/arnd2/ct3/xOutReg_reg		
fftEngine/fftInst/arnd2/ct3/xOutStepReg_reg		
fftEngine/fftInst/arnd2/ct4/xOutReg_reg		
fftEngine/fftInst/arnd2/ct4/xOutStepReg_reg		
fftEngine/fftInst/arnd2/ct5/xOutReg_reg		
fftEngine/fftInst/arnd2/ct5/xOutStepReg_reg		
fftEngine/fftInst/arnd2/ct6/xOutReg_reg		
fftEngine/fftInst/arnd2/ct6/xOutStepReg_reg		
fftEngine/fftInst/arnd2/ct7/xOutReg_reg		
fftEngine/fftInst/arnd2/ct7/xOutStepReg_reg		
fftEngine/fftInst/arnd3/transformLoop[0].ct0/xOutReg_reg		
fftEngine/fftInst/arnd3/transformLoop[0].ct0/xOutStepReg_reg		
fftEngine/fftInst/arnd3/transformLoop[0].ct1/xOutReg_reg		
fftEngine/fftInst/arnd3/transformLoop[0].ct1/xOutStepReg_reg		

Cells - find_1 (186)

✎ **Recommended:** Analyze the placement based on hierarchy name and highlight before fixing the placement.

Reusing Placement

It is fairly easy to reuse the placement of I/Os, Global Clock Resources, Block RAM macros, and DSP macros. Re-using this placement helps to reduce the variability in results from one netlist revision to the next. These primitives generally have stable names. The placement is usually easy to maintain.

★ **Tip:** Do not reuse the placement of general slice logic. Do not reuse the placement for sections of the design that are likely to change.

Reusing Placement with Incremental Compile

Incremental Compile allows reuse of place and route data from a previous run. To set it up, simply reference an existing placed or routed DSP before `place_design`. It is possible to reuse a full design, a hierarchy level, or a cell type like DSPs or block RAMs. Incremental Compile also automatically handles changes made to parts of a design.
For more information, see the *Vivado Design Suite User Guide: Implementation* (UG904).

Floorplanning Techniques

Consider gate-level floorplanning for a design that has never met timing, and in which changing the netlist or the constraints are not good options.

✎ **Recommended:** Try hierarchical floorplanning before considering gate level floorplanning.

Hierarchical Floorplanning

Hierarchical floorplanning allows you to place one or more levels of hierarchy in a region on the chip. This region provides guidance to the placer at a global level, and the placer does the detailed placement. Hierarchical floorplanning has the following advantages over gate-level floorplanning:

- Hierarchical floorplan creation is fast compared to gate-level floorplanning. A good floorplan can improve timing. The floorplan is resistant to design change.
- The level of hierarchy acts as a container for all the gates. It will generally work if the netlist changes.

In hierarchical floorplanning:

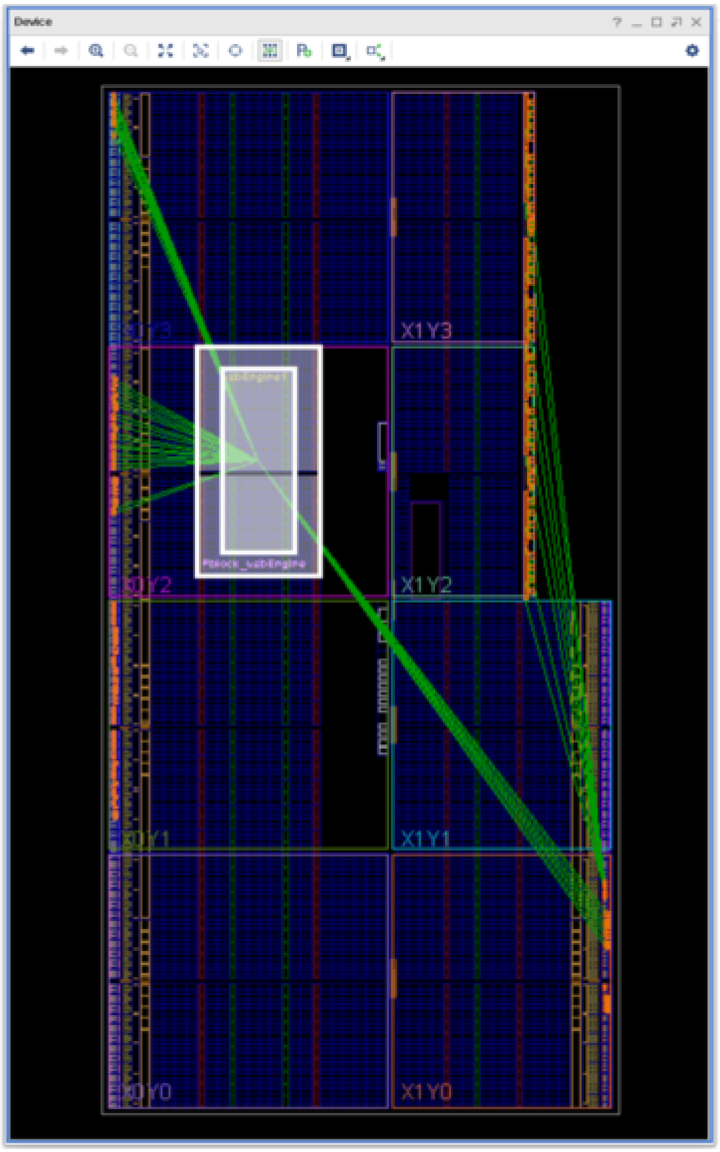
- Identify the lower levels of hierarchy that contain the critical path.
- Use the top level floorplan to identify where to place them.
- Implementation places individual cells.
- Has comprehensive knowledge of the cells and timing paths.
- Generally does a good job of fine grain placement.

Manual Cell Placement

Manual cell placement can obtain the best performance from a device. When using this technique, designers generally use it only on a small block of the design. They may hand place a small amount of logic around a high speed I/O interface, or hand place Block RAMs and DSPs. Manual placement can be slow.
All floorplanning techniques can require significant engineering time. They might require floorplan iterations. If any of the cell names change, the floorplan constraints must be updated.
When floorplanning, you should have an idea of final pinout. It is useful to have the I/Os fixed. The I/Os can provide anchor points for starting the floorplan. Logic that communicates to I/Os migrates towards the fixed pins.

★ **Tip:** Place blocks that communicate with I/Os near their I/Os. If the pinout is pulling a block apart, consider pinout or RTL modification.

Figure: I/O Components Pulling Design Apart



The floorplan shown in the previous figure might not help timing. Consider splitting the block apart, changing the source code, or constraining only the Block RAMs and DSPs. Also consider unplacing I/O registers if external timing requirements allow.

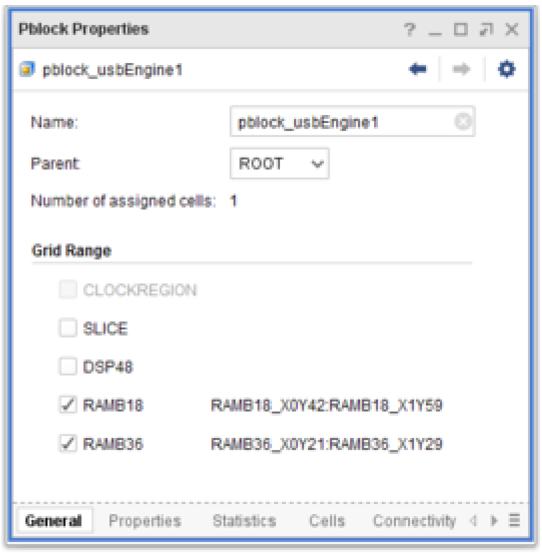
The Pblock mentioned in this section is represented by the XDC constraints:

```
create_pblock Pblock_usbEngine
add_cells_to_pblock [get_pblocks Pblock_usbEngine] [get_cells
-quiet [list
usbEngine1]]
resize_pblock [get_pblocks Pblock_usbEngine] -add
{SLICE_X8Y105:SLICE_X23Y149}
resize_pblock [get_pblocks Pblock_usbEngine] -add
{DSP48_X0Y42:DSP48_X1Y59}
resize_pblock [get_pblocks Pblock_usbEngine] -add
{RAMB18_X0Y42:RAMB18_X1Y59}
```

```
resize_pblock [get_pblocks Pblock_usbEngine] -add
{RAMB36_X0Y21:RAMB36_X1Y29}
```

The first line creates the Pblock. The second line (`add_cells_to_pblock`) assigns the level of hierarchy to the Pblock. There are four resource types (SLICE, DSP48, RAMB18, RAMB36) each with its own grid. Logic that is not constrained by a grid can go anywhere in the device. To constrain just the Block RAMs in the level of hierarchy, disable the other Pblock grids.

Figure: Pblock Grids



Name: pblock_usbEngine1
Parent ROOT
Number of assigned cells: 1

Grid Range

☐ CLOCKREGION	
☐ SLICE	
☐ DSP48	
✓ RAMB18	RAMB18_X0Y42:RAMB18_X1Y59
✓ RAMB36	RAMB36_X0Y21:RAMB36_X1Y29

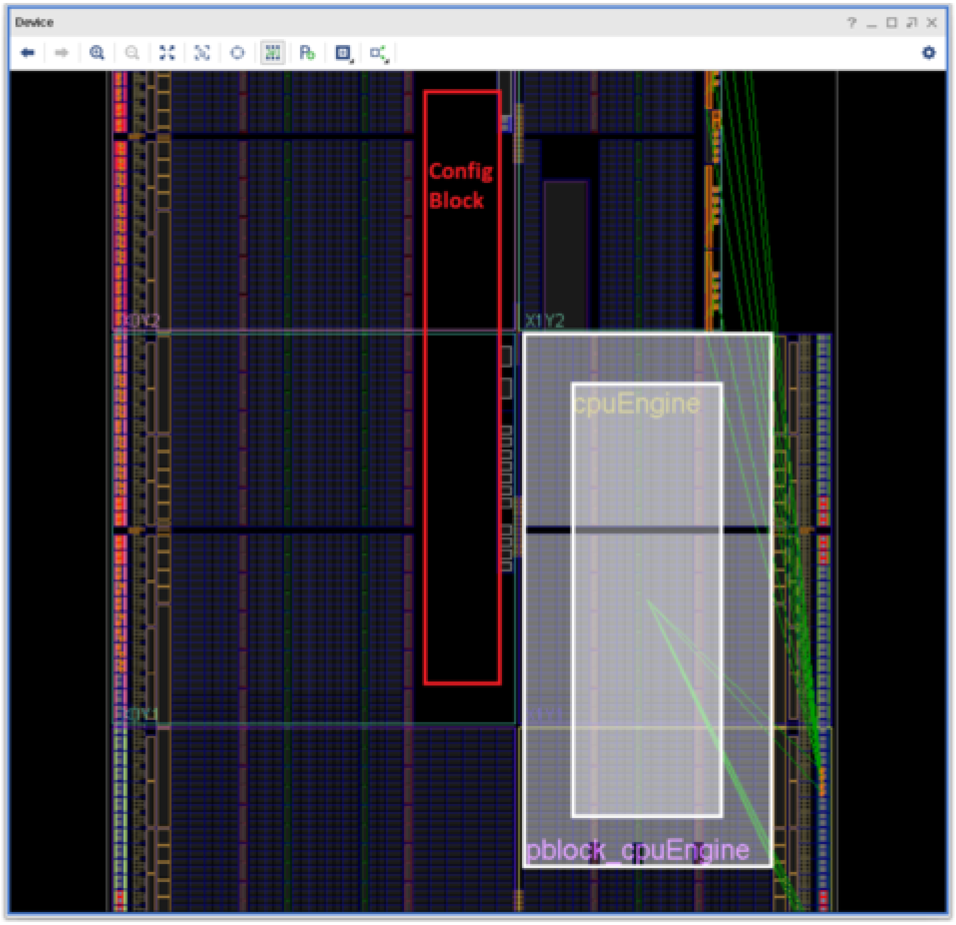
The resulting XDC commands define the simplified Pblock:

```
create_pblock Pblock_usbEngine
add_cells_to_pblock [get_pblocks Pblock_usbEngine] [get_cells
-quiet [list
usbEngine1]]
resize_pblock [get_pblocks Pblock_usbEngine] -add
{RAMB18_X0Y42:RAMB18_X1Y59}
resize_pblock [get_pblocks Pblock_usbEngine] -add
{RAMB36_X0Y21:RAMB36_X1Y29}
```

The Block RAMs are constrained in the device, but the slice logic is free to be placed anywhere on the device.

★ **Tip:** When placing Pblocks, be careful not to floorplan hierarchy in such a manner that it crosses the central config block.

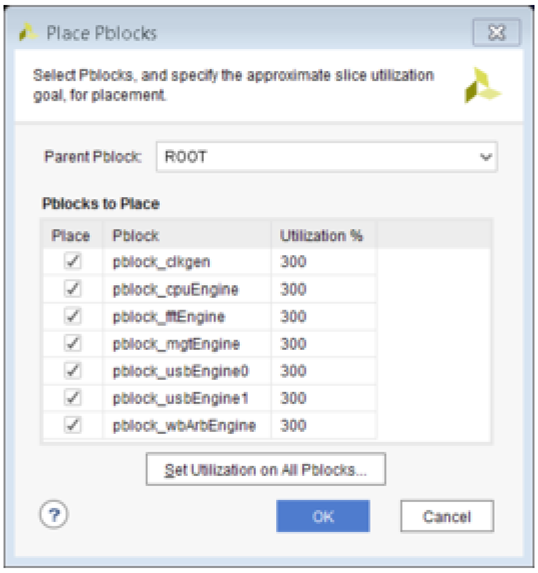
Figure: Avoiding the Config Block



Using Pblock-Based Floorplanning

When you integrate RTL into a design, it helps to visualize the design inside the device. Graphically seeing how the blocks interconnect between themselves and the I/O pinout after synthesis helps you to understand your design.
To view the interconnect, generate a top level floorplan using Pblocks on upper levels of hierarchy. To break apart the top level RTL into Pblocks, select Tools > Floorplanning > Auto Create Pblocks.
To place the blocks in the device, select Tools > Floorplanning > Place Pblocks. The tool sizes the Pblocks based on the slice count and target utilization.
Pblocks can be more than one hundred percent full during analysis, but not during implementation. Overfilling the Pblock makes them smaller on the device. This is a useful technique for getting an overview of the relative size of your design top-level blocks, and how they will occupy the device.

Figure: Place Pblocks Utilization



Place Pblocks

Select Pblocks, and specify the approximate slice utilization goal, for placement.

Parent Pblock: ROOT

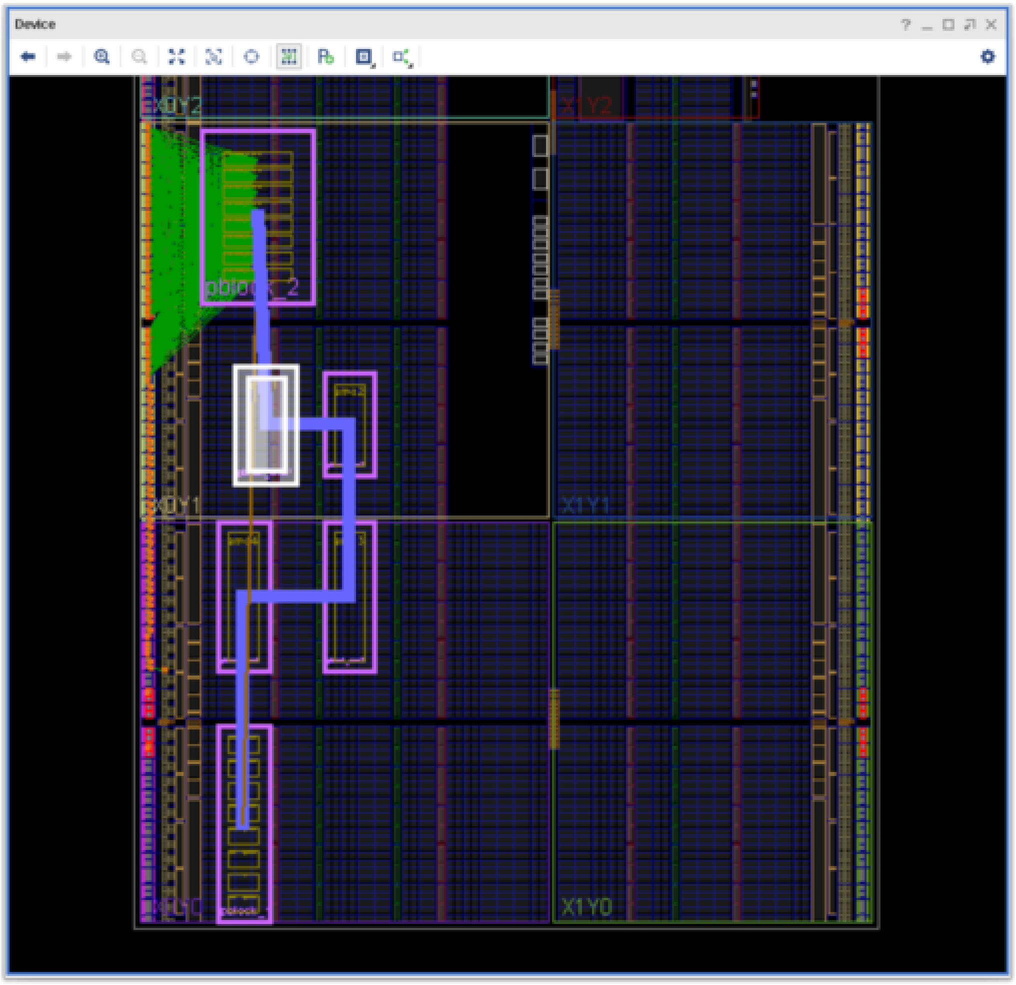
Pblocks to Place

Place	Pblock	Utilization %
✓	pblock_clkgen	300
✓	pblock_cpuEngine	300
✓	pblock_fftEngine	300
✓	pblock_mgtEngine	300
✓	pblock_usbEngine0	300
✓	pblock_usbEngine1	300
✓	pblock_wbArbEngine	300

Top-Level Floorplan

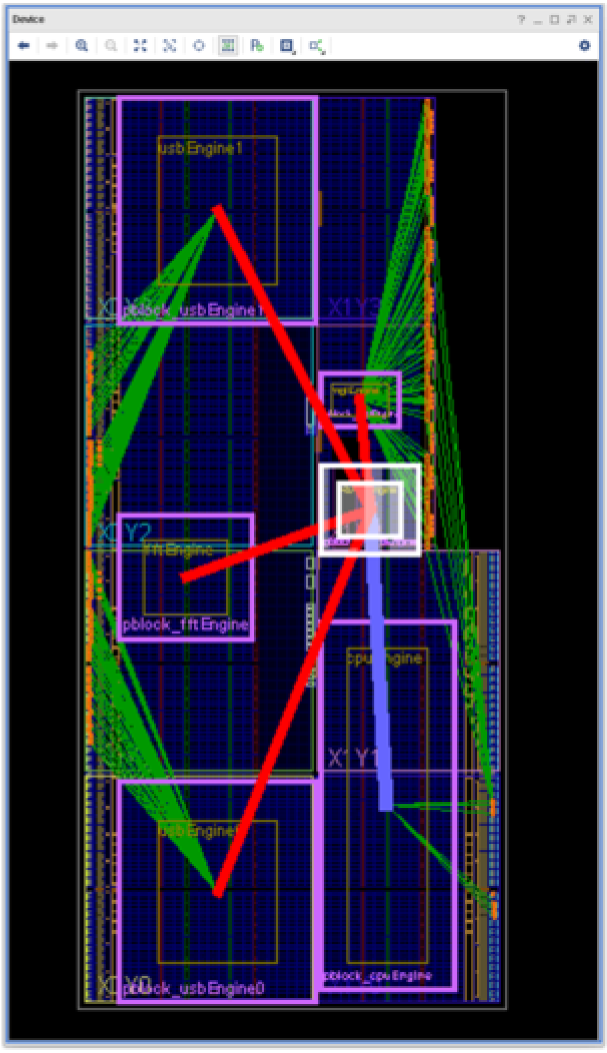
The top-level floorplan shows which blocks communicate with I/Os (green lines). Nets connecting two Pblocks are bundled together. The bundles change size and color based on the number of shared nets. Two top-level floorplans are shown in the following figures.
The Data Path Top Level Floorplan shows how the data flows between the top-level blocks of the design. Each block communicates only with two neighbors. The green lines show well-placed I/Os that communicate with a single block.

Figure: Data Path Top Level Floorplan



The Control Path Floorplan displays a design in which all the blocks communicate with a central block. The largest connection is between the central block and the block in the bottom right. The central block must spread out around the design to communicate with all the other loads.

Figure: Control Path Floorplan



Analyzing Utilization Statistics

A common cause of implementation issues is not considering the explicit and implicit physical constraints. The pinout, for example, becomes an explicit physical constraint on logic placement. Slice logic is uniform in most devices. The following specialized resources, however, represent implicit physical constraints because they are only available in certain locations, and impact logic placement:

- I/Os
- Gigabit transceivers
- DSP slices
- Block RAM
- Clock management blocks such as MMCM
- Clock buffers such as BUFG

Blocks that are large consumers of these specialized resources might have to be spread around the device, physically constraining placement and routing when

designing the interface with the rest of the design. Additionally, Pblocks are explicit physical constraints used to define allowable placement areas for specified logic.
Use a combination of the following methods to analyze block resource usage on the device:

- Report utilization
- Netlist properties
- Pblock properties

Locking Specific Logic to Device Sites

You can place cells on specific locations on the FPGA, such as placing all the I/O ports on a Xilinx 7 series FPGA design. Xilinx recommends that you place the I/Os before attempting to close timing.
The I/O placement can impact the cell placement in the FPGA fabric. Hand placing other cells in the fabric can help provide some consistency to clock logic and macro placement, with the goal of more consistent implementation runs.

Table: Constraints Used to Place Logic

Constraint	Use	Notes
LOC	Places a gate or macro at a specific site.	SLICE sites have subsites called BEL sites.
BEL	Specifies the subsite in the slice to use for a basic element.	

Fixed and Unfixed Cells

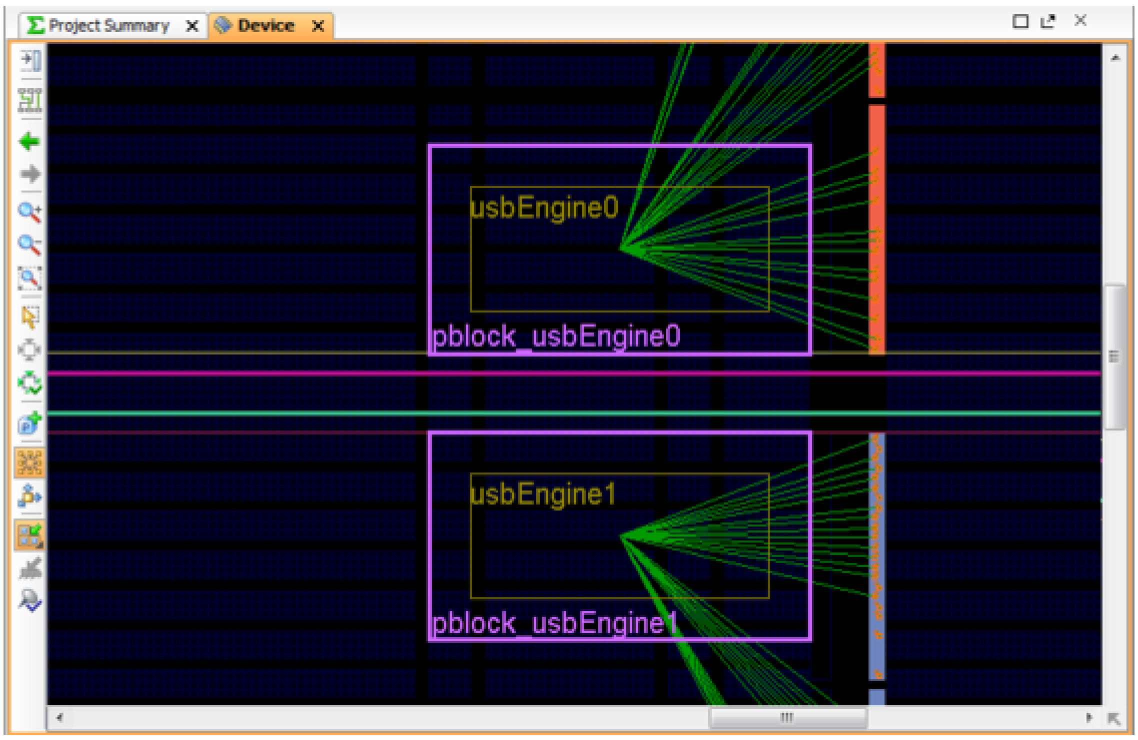
Fixed and Unfixed apply to placed cells. They describe the way in which the Vivado tools view placed cells in the design.
For more information about Fixed and Unfixed Cells, refer to this link in the *Vivado Design Suite User Guide: Implementation* (UG904).

✎ **Recommended:** After the I/Os are placed, use a hierarchical Pblock floorplan as a starting point for user-controlled placement. Hand placing logic should be used when Pblocks have been found not to work.

Floorplanning With Stacked Silicon Interconnect (SSI) Devices

There are extra considerations for Stacked Silicon Interconnect (SSI) parts. The SSI parts are made of multiple Super Logic Regions (SLRs), joined by an interposer. The interposer connections are called Super Long Lines (SLLs). There is some delay penalty when crossing from one SLR to another.
Keep the SLRs in mind when structuring the design, generating a pinout, and floorplanning. Minimize SLL crossings by keeping logic cells of critical timing paths inside a single SLR.

Figure: Minimize SLR Crossings



The I/Os must be placed in the same SLR as the relevant I/O interface circuitry. You must also carefully consider clock placement when laying out logic for SSI parts.

✎ **Recommended:** Let the placer try an automatic placement of the logic into the SSI parts before doing extensive partitioning. Analyzing the automatic placement may suggest floorplanning approaches you were not considering.