PlanAhead Methodology Guide

Release 11.1

UG633(v 11.1.0) April 27, 2009

Included in the PlanAhead source code is source code for the following programs:

Centerpoint XML

The initial developer of the Original Code is CenterPoint – Connective Software Software Engineering GmbH. Portions created by CenterPoint – Connective Software Software Engineering GmbH. are Copyright © 1998-2000 CenterPoint - Connective Software Engineering GmbH. All Rights Reserved. Source Code for CenterPoint is available at http://www.cpointc.com/XML/

NLView Schematic Engine

Copyright © Concept Engineering.

Static Timing Engine by Parallax Software Inc.

Copyright © Parallax Software Inc.

Java Two Standard Edition

Copyright © 1995 - 2006 Sun Microsystems

Includes portions of software from RSA Security, Inc. and some portions licensed from IBM are available at http://oss.software.ibm.com/icu4j.

Powered By JIDE – http://www.jidesoft.com

The BSD License for the JGoodies Looks

Copyright© 2001-2004 JGoodies Karsten Lentzsch. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

o Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
o Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
o Neither the name of JGoodies Karsten Lentzsch nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTI/ON) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Table of Contents

About this Manual

FPGA designers face a variety of challenges during design implementation. The types of challenges encountered depend primarily on the target application, project goals and priorities. The PlanAhead™ tool supports multiple flows and methodologies to help designers overcome some of these difficulties. This document is intended to assist designers with the various PlanAhead flows. It is intended more as a "helpful hints" guide than a "must follow" procedural outline. Please use it with proper consideration for the context of your design and apply suitable techniques with reasonable care.

The main sections of this document are listed below:

1. Design Flow with PlanAhead
2. I/O Pin Assignment with PinAhead
3. Performance Driven Design
4. IP Block Creation and Re-use
5. Maximizing Device Utilization
6. Using Platform Studio and the EDK with PlanAhead

The IP creation flows are most effective when implemented at the beginning of an FPGA design project. These techniques require up front design partitioning and hierarchical synthesis strategies.

This guide assumes the user has performed the ***PlanAhead Tutorial*** or is otherwise familiar with the PlanAhead tool.

Additional Resources

Perform the ***PlanAhead Tutorial*** to learn most of the PlanAhead functionality using a sample design walk-through.

Refer to the ***PlanAhead User Guide*** for information about specific PlanAhead functionality and more details on specific commands.

For more information, including video demonstrations of the benefits of using PlanAhead, visit the following web site:

http://www.xilinx.com/planahead

Design Flow using PlanAhead

The PlanAhead tool supports many powerful flows. This document will talk about timing closure and results repeatability. Other flows are covered by the *PlanAhead User Guide*. The PlanAhead tool sits between logic synthesis and the ISE® place and route tools. Any synthesis tool, targeting Xilinx® FPGAs, can be used for your design. In this flow the PlanAhead tool uses the synthesized netlist and design constraint files for its powerful analysis capabilities (Figure 1) . From the PlanAhead tool, you can export an EDIF netlist and a single design constraint UCF file to drive the ISE tools. The ***PlanAhead User Guide*** talks in detail about other PlanAhead flows.

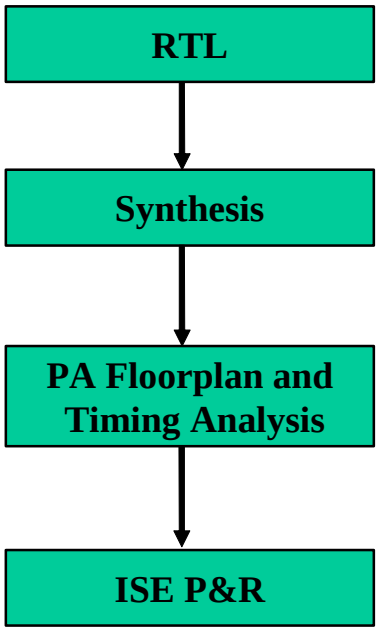

Figure 1: FPGA Design flow using PlanAhead

The PlanAhead tool supports top-level netlists in EDIF or NGC format. Module netlists can be in EDIF, NGC or NGO format. If NGC or NGO netlists are being used, PlanAhead will invoke the ***ngc2edif*** utility from within the ISE tools. The ISE software environment needs to be setup correctly before invoking the PlanAhead tool. The PlanAhead tool reads the NGC/NGO files by first translating them into EDIF. When NGC or NGO cores are imported into the PlanAhead tool, they are used for floorplanning and analysis purposes only. Upon export for implementation, the original NGC and NGO cores are copied to the export directory.

Design constraints can be imported from one or more UCF or NCF file(s). For export, all input UCF and NCF constraints are combined into a single UCF file.

1. Logic Synthesis Recommendations

The following are suggestions on a logic synthesis methodology:

1. To the extent possible, partition the design at the RTL level such that critical timing paths are confined to individual modules. Critical paths that span large numbers of hierarchical modules can be difficult to floorplan.
2. Register the outputs of all the modules to help limit the number of modules involved in a critical path.
3. Replicate the drivers of nets that will be 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 a difficult task. Consider dividing large hierarchical blocks in the RTL. It is easier to work with smaller hierarchical blocks.
5. If the design is expected to change often, consider an incremental approach to synthesis. In an incremental approach, individual blocks can be synthesized separately or the synthesis attributes (SYN_HIER=HARD) can be used to preserve the hierarchy. Hierarchy preservation will help an incremental flow but may hurt performance since global optimizations across hierarchy are disabled. This tradeoff needs to be considered before you embark on an incremental RTL synthesis methodology.
6. Constrain the synthesis engine to rebuild, or otherwise preserve the hierarchy in the synthesized netlist. Flattened netlists may be optimal from a synthesis perspective, but they make it very difficult to reliably floorplan and constrain placement. Consider using the options and synthesis pragmas to rebuild the hierarchy. For XST add *–netlist_hierarchy = rebuilt*.

2. Creating a New Project

Create a new project using the **File | New Project** command to import the available netlists for the design. This may include all of the netlists associated with the entire design or only the netlists associated with the blocks desired for analysis. Alternatively, you can choose to create a project with no netlist, and import it later, or with HDL sources.

The design does not have to be complete. The top-level netlist may contain black boxes for any modules that are not yet implemented. Module netlists in EDIF, NGC or NGO format can be imported into your project at any time.

PlanAhead allows you to update your entire design netlist with a newly synthesized netlist at any time during your project. PlanAhead will also allow you to update a single module in the design with a newly synthesized module. Use the **File | Update Netlist** command for such netlist updates.

I/O Pin Assignment with PinAhead

PlanAhead provides the capability to do pin assignment with PinAhead technology. Designers can generate I/O package pin assignments manually on a pin-by-pin drag and drop basis, by semi-automatically dragging and dropping groups of ports, or with a fully automatic pin placement algorithm. This process can begin with a synthesized EDIF netlist, an un-synthesized HDL netlist, a comma separated value (CSV) file, or with a completely blank project in which the design ports are created inside the tool for export. PlanAhead may be placed in the PinAhead mode by executing the **Tools | Open PinAhead** command, which opens package and device views beside each other, as well as brings up a list of the defined I/O ports for the top level of your design.

Pin placement can affect the timing of the final design. It is far easier to write code that will meet timing for pins in a single bank or adjacent banks, then for banks on opposite sides of the chip. Take some time when doing the pin planning to consider, what RTL will talk to MGTs, BlockRams, DSP48s, and other embedded elements. Keep in mind the RTL hierarchy that will also talk to these elements. Keep in mind which hierarchy will be pulled apart by a pinout and, if necessary use this information when writing the RTL, The following pin assignment suggestions can help increase productivity for optimal I/O placement.

1. Defining and Configuring Ports

Ports are defined in your design in multiple ways. The most common is if the project was created by importing an EDIF netlist. Ports are defined by the structural definition of the netlist. If RTL synthesis has not yet been completed, you may define ports in your design by importing HDL sources using the **File | Import I/O Ports | From HDL** menu item. You may also define the pins by importing a UCF or CSV file with the appropriate **File | Import I/O Ports** sub-command. If you have none of these available, you may start from a blank project, and define ports inside the tool by right-clicking inside the I/O Ports view, and choosing **Create I/O Ports**.

The **Create I/O Interface** command can be used to create I/O Port interfaces. An interface is a grouping of any number of bus and scalar pins which are related and may need to be placed together or otherwise manipulated. An example of an interface is a DDR interface, which has address and data busses, as well as clocking and control logic pins which all need to be placed in close proximity.

Ports may be configured by selecting the port(s) or interface(s), and then selecting the Configure **I/O Ports** popup menu command. You may set I/O standard, drive strength, slew, and pull type for one or more ports in this manner. Pin assignments and configuration information defined in UCF or in CSV imported into PlanAhead are also fully supported. *Config Prohibit* pin statements in UCF are also fully supported, as well as the ability to set and clear these properties by selecting the **Set Prohibit** or **Clear Prohibit** popup menu commands.

2. Placing Pins Manually

Ports may be assigned manually by selecting the ports in the I/O Ports view, and dragging them onto valid package pins in the Package view. Differential pairs will be snap to the appropriate p-n pair. If dragging over a pin already assigned, a prohibited site, or otherwise invalid placement, PinAhead will snap to the closest valid location and display the site name in the ToolTip popup.

3. Placing Groups of Pins Semi-Automatically

Groups of pins or interfaces may be placed together by selecting multiple pins and dragging them onto valid site locations. By default, PinAhead attempts to assign all the selected ports to legal sites within the same bank. If there aren't enough valid sites to place all the selected pins, a Tooltip popup will tell you how many of the selected pins can be placed.

You may also select a group of pins or interfaces and place them sequentially, one mouse click at a time, rather than dragging them as a group. To do this, select the I/O ports, then choose the **Place I/O Ports Sequentially** toolbar button, then click once on a valid site for each selected pin to place the port. The tool will generate a status pop up if there are issues placing the IO on the package pin.

You may place a group of ports in an arbitrary area by selecting the **Place I/O Ports in Area** toolbar button. Then you may click to define a rectangular region into which ports will be placed. The group of ports selected can be assigned to I/O banks by selecting them and using the **Place I/Os in an I/O Bank** placement mode.

4. Placing Pins Automatically

PinAhead has an automatic pin placement feature that will attempt to place all ports in the design. To run this, select **Tools | Autoplace I/O Ports** menu item. You may place every port in the design, or only the unplaced ports. PinAhead attempts to keep interfaces and vector signals together as much as possible, and will report the number of successfully placed pins, as well as warnings when it is unable to meet placement constraints.

Finally, you can run automatic placement on a group of pins or interfaces. To do this, select the pins you wish to place, and run **Tools | Autoplace I/O Ports** and follow the instructions in the automatic port placement wizard. You will be prompted to place only the selected pins or all pins in the design.

5. Running DRC and Output Integrity Analysis

Once the design I/O ports have been placed, various DRCs can be run to ensure legal placement. To run DRC, select the **Tools | Run DRC** command.

PlanAhead can also do a quick check for some common output drive problems. The tool runs a WASSO (Weighted Average of Simultaneously Switching Outputs) analysis to help the user identify potential signal integrity related I/O problems. To run select **Tools | Run WASSO Analysis**.

6. Exporting Pin Assignment

Once the designer is satisfied with the quality of the pin assignment and configuration, the pin assignment may be exported in the form of a CSV file, UCF, or structural HDL netlist (VHDL or Verilog). CSV export defines all standard information relevant to the pin assignment, including signal name, bank, I/O standard, drive strength, slew, pullup, trace delay, etc. To export the pin assignment for other tools, select the **File | Export I/O Ports** command and choose the output format that is desired.

Performance Driven Design

Sometimes the designer may want to improve design performance or help improve consistency. The following floorplanning suggestions are intended to help increase design performance.

1. Creating a new Floorplan

The New Project Wizard walks you through importing the netlist, new floorplan creation, part selection and importing the design constraints. After the floorplan is created, you can import additional constraints by using the **File | Import Constraints** command. PlanAhead allows multiple constraint files to be imported for the same design. I/O, timing and physical constraints can be in separate constraint files for easier constraint creation and maintenance during the design.

When importing module level constraints such as NCF files associated with cores, select the module instance in the PlanAhead Netlist view and use the **File | Import Constraints** command to import the constraints relative to the instance. The tool will assemble all top-level UCF files and all module level NCF files into a single UCF to be exported to ISE for place and route.

Physical constraints, such as I/O Ports constraints, and LOC, and BEL constraints, appear on the designated sites within PlanAhead. Area Group constraints are converted to Pblocks. RPMs defined through the UCF will show up in the Physical Hierarchy view.

2. Performance Analysis

To see actual timing on a design, run place and route on the unfloorplanned design. Import the timing results using the **File | Import TRCE Results** command to get an understanding of the critical paths. You could also run the **Tools | Run TimeAhead** command in the *No Interconnect* mode on the unfloorplanned, unplaced design to view the paths with the least amount of timing margin. This can help identify paths that will need RTL improvements prior to implementation.

You can use PlanAhead to improve performance of your design by reducing the route delay in the design through floorplanning. Logic delay limits the amount of performance gain you can achieve. If logic delay is modified by DCM phase shifts for paths at an I/O and the logic delay exceeds the timing constraint, relax the constraint or modify the RTL to reduce the logic delay.

For internal register to register paths with a lot of logic delay, place the logic from the critical path into a temporary Pblock. Use ExploreAhead to quickly run place and route on the Pblock to judge the feasibility of meeting timing targets. The results of this Pblock run can help you understand if these gates can make timing in isolation. This is a quick check for critical sections of your design. Timing can degrade on these paths in the context of the entire design. If the critical path does not make timing in isolation, consider revising the RTL to meet your timing constraints. This method will not work on paths starting or ending at an I/O pad.

3. Finding Critical Logic

If place and route has been run, import the .twx file generated by the ISE *trce* command, using the **File | Import TRCE Results** command. This will have the actual timing information from the place and route run. ISE placement should be imported from the placed and routed .ncd file using the **File | Import Placement** command. If ExploreAhead is being used to run place and route, the placement and timing results can be loaded into a PlanAhead floorplan in a single step using the ExploreAhead **Import Run** popup menu command.

To get a better feel for the placement of timing critical and non-critical logic, run the PlanAhead timing analyzer using the **Tools | Run TimeAhead** command in the *Estimated* mode, and then enable the **Metrics | Min Slack per placed BEL** display. The PlanAhead tool can highlight the placed elements in the design based on user definable slack ranges. This lets the user see the distribution of timing critical logic and non-timing critical logic.

Use this information to determine the hierarchical instances involved in timing critical paths. Primitive gate names can change with each synthesis run. Therefore, try to avoid floorplanning the primitive gates. Instead, floorplan the hierarchical instances that contain the critical paths. If the critical paths span multiple levels of the hierarchy, floorplan them all. The parent hierarchy may be much bigger then the child hierarchy that contains the critical paths. Check relative sizes in the Hierarchy View using the Show hierarchy popup menu command before floorplanning the modules.

4. Placing Pblocks

You may want to visualize the connectivity and the data flow through the design by running the **Tools | Partition** and **Tools | Place Pblocks** commands at the design's top level. Repeat the commands for large Pblocks. The resulting top level floorplan will provide an excellent view of the design data flow highlighting

potential congestion areas or large fanout busses (Figure 2). In some cases, design bottlenecks can be seen through this process even as the RTL is being developed.

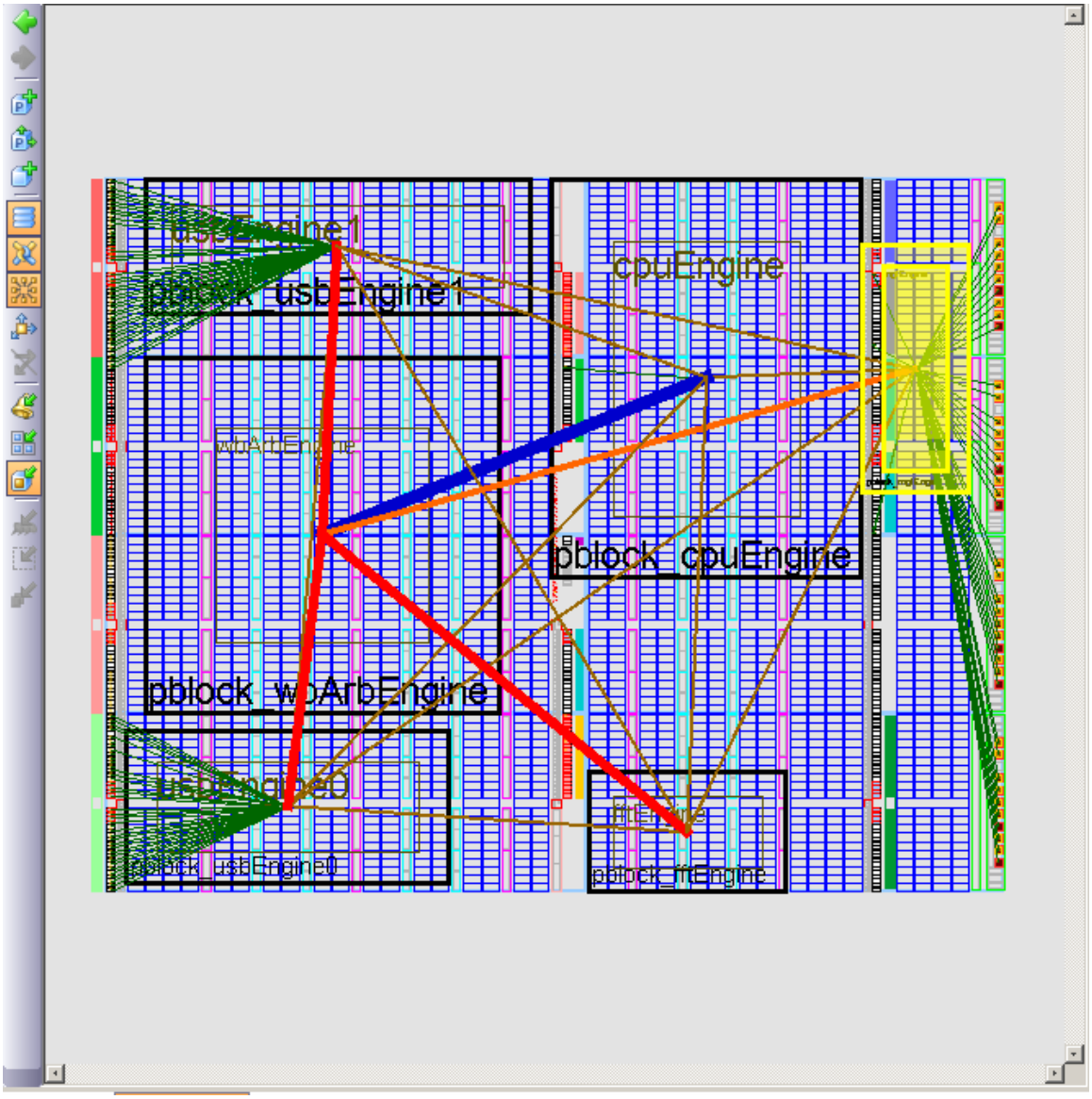

Figure 2: Visual Feedback of Design Connectivity

It is obvious from the connectivity analysis which blocks connect to which I/Os and which Pblocks are heavily interconnected. Viewing the Pblock Properties for each Pblock also gives an understanding of the resources required for each one. You may need to repartition larger Pblocks using lower levels of hierarchy in order to create tighter rectangles for them. This top-level floorplanning technique is typically used for analysis purposes only as it tends to isolate all modules leading to poor implementation results.

PlanAhead provides analysis capabilities for designs that are not finished. The user can import the netlists and lay out the top-level hierarchy. The bundles will show what blocks are heavily connected. This can be quite useful when doing initial pin assignment. You can easily see if the I/Os are grouped or spread out. With only a little more work you can see if the pinout is pulling logic away from fixed silicon resources (DSP48s, PPCs, and etc). Sometimes problems can be seen in time to change the pinout. In other cases designers can change the logic hierarchy while the RTL is still in the development phase to later reduce place and route iterations.

5. Using the Schematic View

Another way to view the design interconnects and data flow is to use the **Schematic** view on the top-level of the design (Figure 3).

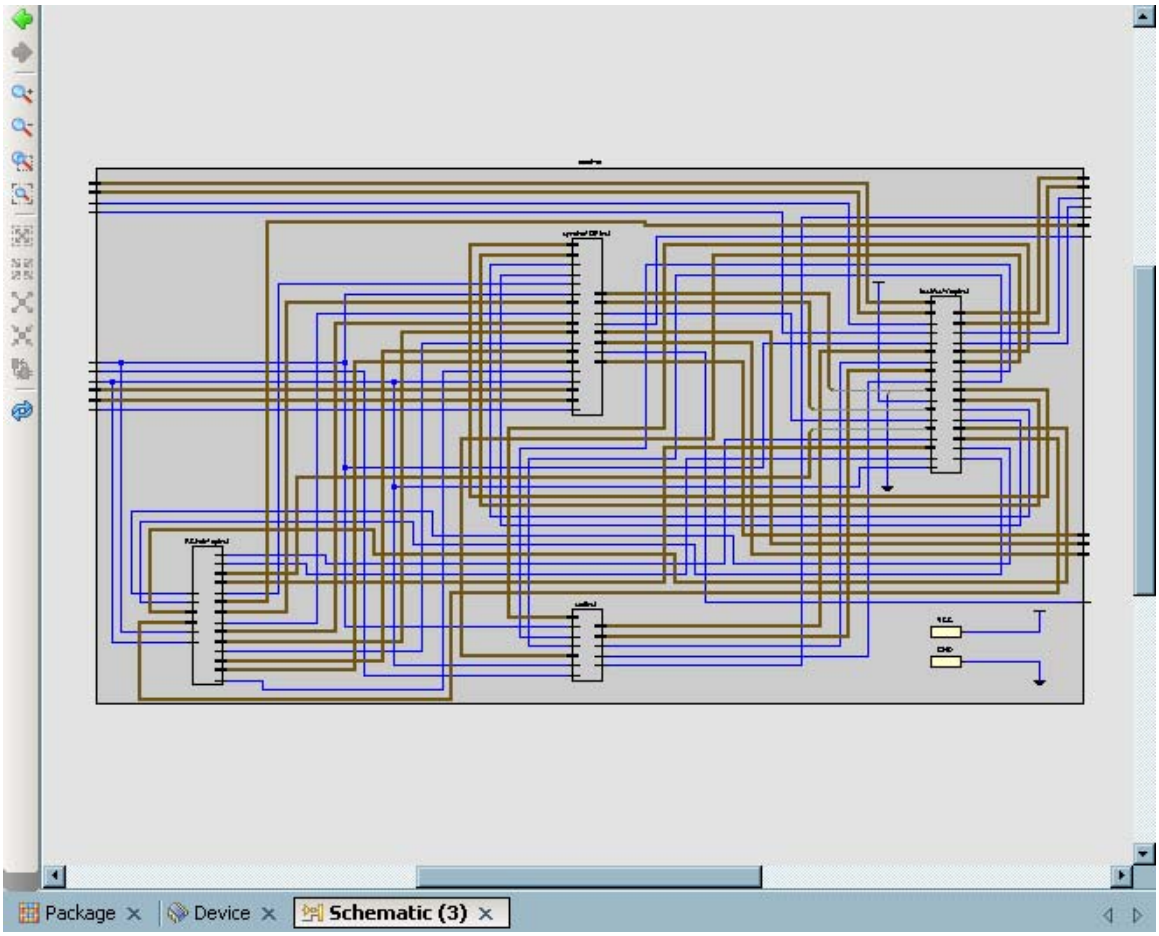

Figure 3: Schematic View for Design Analysis

Critical signals can be traced through the design to view the primitives connected to the signals. The hierarchy containing these instances can then be selected and grouped into Pblocks to constrain the logic involved. In the Schematic view, right

click on inputs or outputs and select the **Expand Cone | To Flops** popup menu command to view logic connectivity.

6. Using the Hierarchy View

PlanAhead recommends a hierarchical floorplanning and analysis approach. The hierarchy acts as a handle for all the gates under it. The user does not need to worry about synthesis changing the names on the logic in the hierarchy. One way to quickly see and analyze the netlist structure is in the Hierarchy view. Figure 4 shows an example of a design with a lot of hierarchy. Notice that there are many levels of hierarchy available to constrain logic to small regions of the chip.

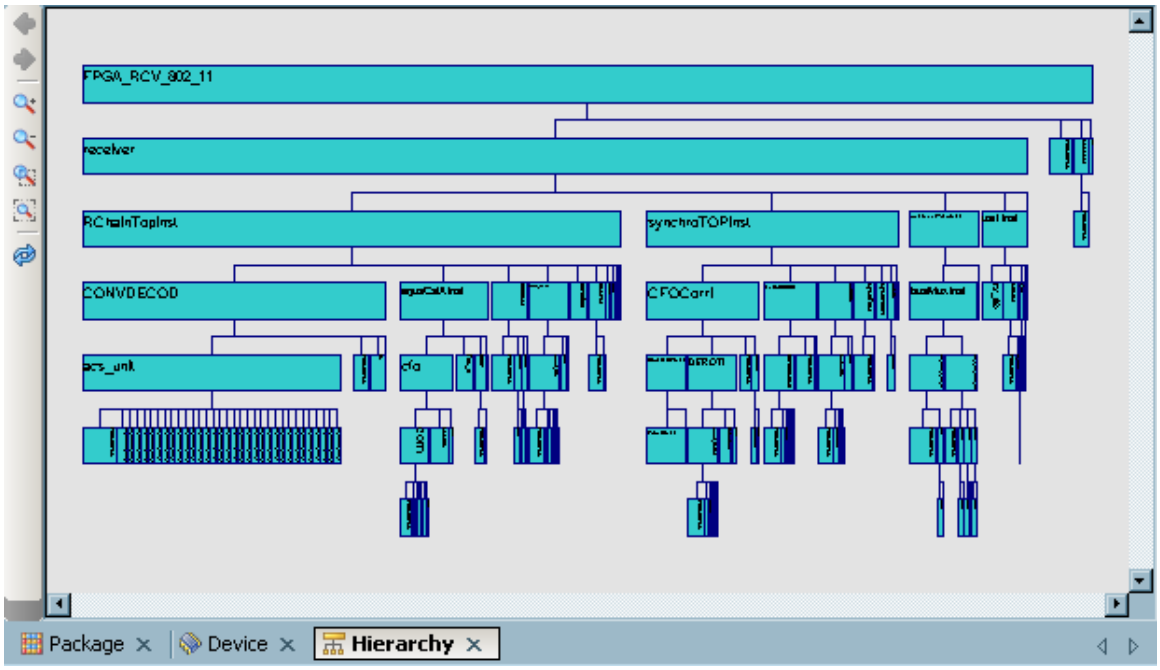

Figure 4: Well Designed Hierarchy as Shown in the Hierarchy View

A design with few or no hierarchical branches does not give the designer many choices on how to floorplan. Such designs are not amenable to good floorplanning.

7. Implementing Design and Using Results to Guide Floorplanning

If the design has been run through ISE, the results can help drive analysis and floorplanning. Use the **File | Import Placement** or the ExploreAhead **Import Run** commands to import existing placement into the PlanAhead floorplan. The **File | Import TRCE Results** command can be used to load in the timing results. When a path is selected the user can easily see where PAR placed the path. The **Highlight Primitives** command can be used to highlight where levels of hierarchy were placed. Different hierarchical blocks can be highlighted using

different colors to get a better understanding of the design placement. This can help guide the placement of Pblocks to further constrain that logic (Figure 5).

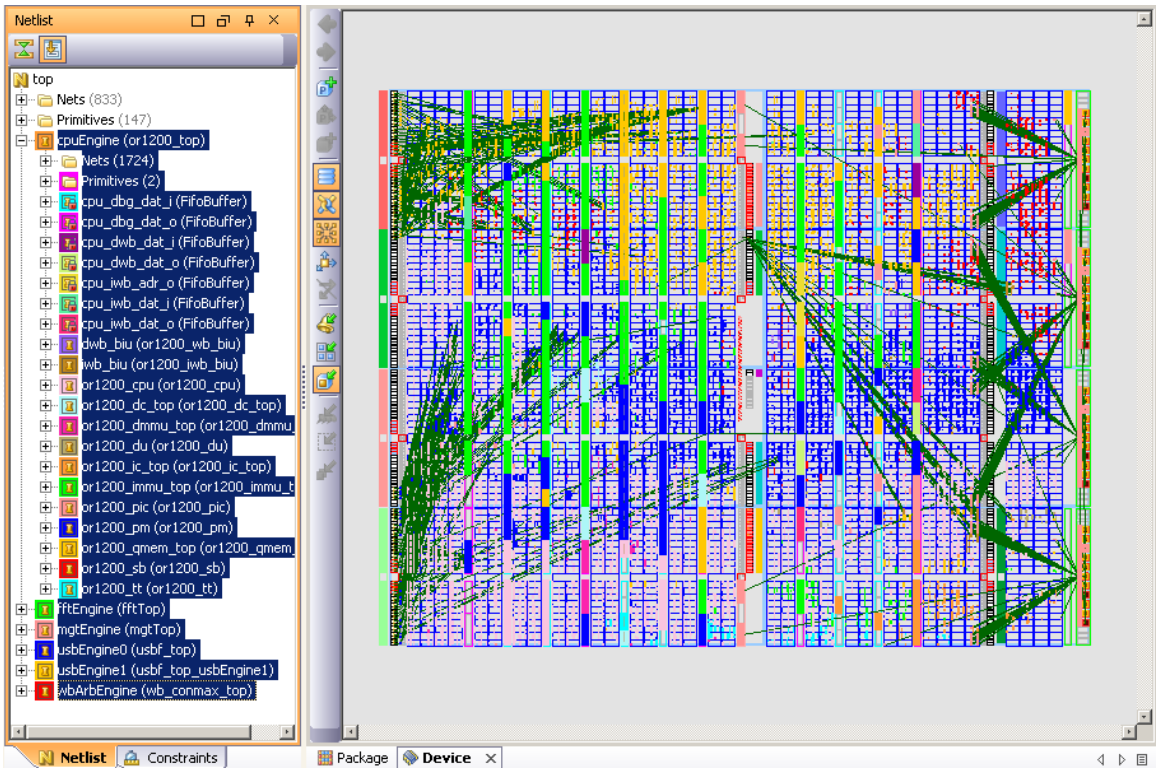

Figure 5: Placement Analysis through Selective Logic Highlighting

8. Constraining Logic with Pblocks

Use the methods described earlier to gain an understanding of the design data flow and the resource requirements of the various modules. Modules with block memory and block arithmetic sites must be taken into consideration since their placement sites can be somewhat restrictive.

Where possible, use your knowledge of the design to create Pblocks for critical modules. The analysis from TimeAhead, the Schematic view, and/or place and route results can provide guidance.

When floorplanning for performance, it is good practice to only constrain the hierarchies that contain the critical path. In some cases, the hierarchy that is connected to fixed silicon resources (I/Os or PPCs) should be floorplanned as well. Floorplanning all the logic in a design, as one might do in an ASIC flow, will generally hurt performance. Since FPGA tools work differently from ASIC tools, just as FPGA architecture is different from ASIC architecture. It is rare that floorplanning an entire FPGA design will help performance. The **Tune the Floorplan to Meet Timing** section of this document discusses how to group logic to improve timing.

Use the **Pblock Properties | Statistics** to make the Pblock as small as practical to limit interconnect length. Use these same statistics to ensure that the RPMs and Carry Chains will fit in the Pblock rectangle.

Larger Pblocks may need to be partitioned further to provide a finer level of placement constraint granularity. As a rule of thumb keep the size of a single Pblock less than 30% of the chip. Smaller Pblocks are generally better.

Working with RPMs and RLOCs – Some of the Xilinx CORE Generator™ System cores and some synthesized blocks include RLOC constraints. These groups of RLOC constraints are referred to as RPMs. The PlanAhead Physical Hierarchy view shows all of the RPMs in a separate folder. Users can create Pblocks at desired locations on chip and then directly or indirectly assign these RPMs to various Pblocks. Occasionally users can create very specific RPMs through the RLOC constraint and then floorplan them to chosen regions on the chip through Pblocks.

Often the RPMs are intended only to keep logic from being spread across a chip. If the RPM is in a Pblock, the Pblock's range is constraining the logic. Unless the RPM is being used to force a particular packing, it can be useful to delete a RPM contained in a Pblock. To erase an RPM, select it in the Physical Hierarchy view and select the **Delete** popup menu command.

The PlanAhead Pblock statistics display the height and width requirements of the largest RPMs. This can guide Pblock size and shape so the RPM will fit inside.

9. Manually Assigning LOC and BEL Constraints

A common practice is to define physical constraints to lock certain RAMs or multipliers into specific sites. Many designs see increased consistency when the user re-uses the RAM and MAC placement from a Place and Route run that met timing. After back annotating the Place and Route results, select this logic using the find command and "fix" the placement. Occasionally timing critical I/O constraints need to have logic placed into specific nearby slice or BEL sites. Refer to the *Using Placement Constraints* chapter of the User Guide for information on placing this logic.

10. Constraining I/O Registers

If I/O Registers have tight Max Delay or Max Skew constraints, Pblocks can be created adjacent to the I/O Pads to contain these registers, reducing distance and the subsequent delay and skew. To ensure that registers are packed into the pads, **map** should be given the **–pr b** switch. For more information, see the ISE software documentation.

11. Using Clock Resources to Guide Floorplanning

Different FPGA device families have different restrictions on the placement of logic in a design utilizing a high percentage of the device's clock resources. The user may have to consider the device's clock rules when placing the logic. The PlanAhead tool can help constrain certain clocks to certain regions in the chip. It is possible to graphically display the various clock regions or clock quadrants within the chip. The **Clock Region Properties** or **Pblock Properties | Statistics** will show what clock nets and clock regions are in all Pblocks. The schematic view can show what logic and hierarchy is attached to each clock net.

12. Constraining Hierarchy Containing the Critical Paths

Avoid constraining individual logic instances unless absolutely necessary. It is easier to create and maintain floorplans based on levels of hierarchy since gate instance names can change during synthesis.

From the timing report, failing critical paths can be highlighted and selected for placement into Pblocks, as shown below. Multiple path elements can all be selected and included for placement into a new Pblock (Figure 6). This can drastically reduce the delay on that path. Care must be taken to not affect other areas of the design with this approach.

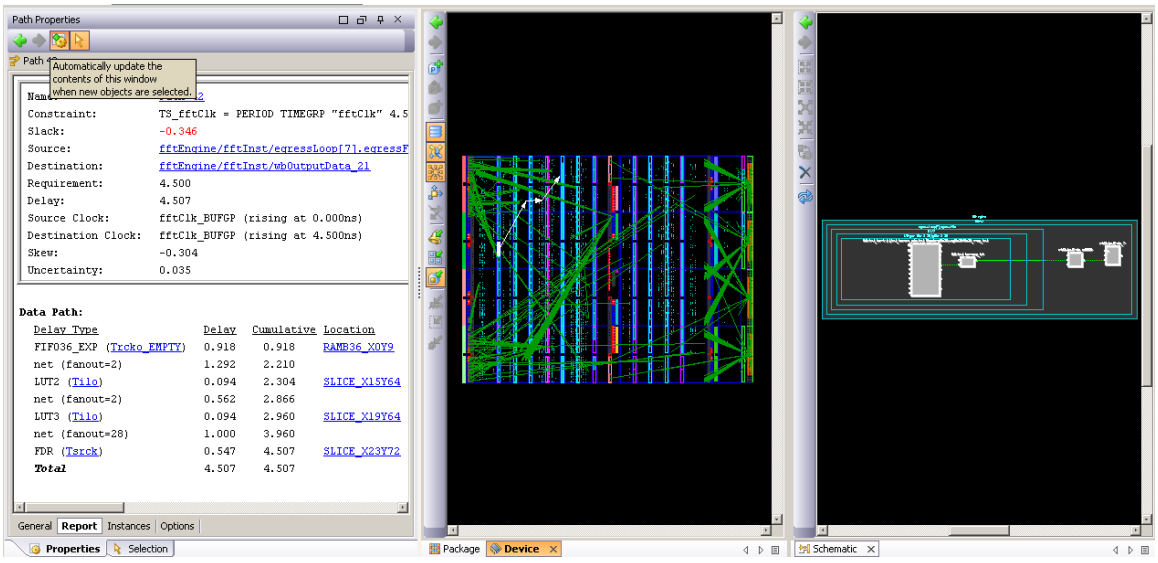

Figure 6: **Visual Feedback of Timing Critical Paths**

Groups of critical paths can also be viewed in the Schematic view, as shown below. Paths can be further explored in the timing report and the Schematic view, and they can be selected to form a basis for placement into Pblocks.

As mentioned earlier, it is a good practice to constrain the hierarchy containing the critical paths rather than the gates. In the Schematic view below (Figure 7),

multiple levels of hierarchy are available to choose from. Use the **Highlight Primitives** popup menu command discussed in Step 7 to determine size and placement of parents versus children. If a parent hierarchy is much larger than the children containing the critical path, it is generally a good idea to constrain the children. For the example shown below, the two levels of hierarchy highlighted in yellow were floorplanned into a single Pblock to improve timing.

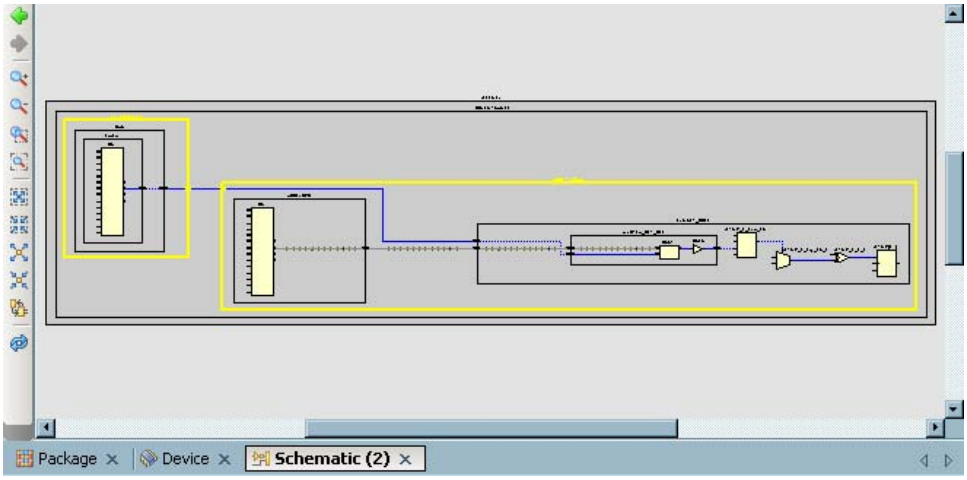

Figure 7: Visual Feedback of Timing Critical Path Schematic

13. Using LOC Constraints to Lock Placement

If performance is satisfactory for any of the Pblocks in the design, the placement can be locked through LOC constraints. Rather than place the logic by hand, LOCs can be imported from a top-level or Pblock-level ISE run. Unwanted LOCs can be cleared using the **Tool | Clear Placement Constraints** command with the rest left in place for subsequent floorplan exports. This method does not work well if the logic in the Pblock is going to change or if any of the outputs of the Pblock are unregistered.

14. Running DRC and ISE Place and Route

Run the design rule checks using the **Tools | Run DRC** command to see if common errors in the floorplan will cause problems in the downstream tools. Use the Run Implementation command to export the netlist and run through ISE to see if timing has been met.

15. Performing Detailed Performance Analysis

Use the PlanAhead tool's analysis capabilities to understand and improve upon the design performance bottlenecks. After running place and route, read back the placement and the timing from place and route.

During the floorplanning process, run the **Tools | Run TimeAhead** command in *Estimated* mode to estimate and highlight any new critical path delays. This can be performed prior to and after running ISE. Various netlist-level or Pblock-level

–from, –through, and/or –to filters can help analyze the floorplan timing without having to go through place and route.

16. Tuning the Floorplan to Meet Timing

Floorplanning can be an iterative process. Most designs do not go from failing timing to making timing on the first floorplan. After the first floorplan, you may need to revise the floorplan. It is helpful to save each floorplan in case you want to revisit your work later.

If critical paths are within unfloorplanned logic, create a new Pblock. Identify the levels of hierarchy that contain the critical path. Assign them to a new Pblock and place it on the chip. If the placement is reasonable, keep this Pblock around for place and route.

If critical paths are within a single Pblock, revise the Pblock. Consider creating an embedded Pblock to more tightly constrain the critical hierarchy. Alternately, work with lower levels of hierarchy and consider removing some logic from the Pblock.

If critical paths are between a Pblock and unconstrained hierarchy, add the unconstrained logic to a Pblock. The first option is to create a new Pblock to hold the critical path and place it nearby. The second option which works if the unconstrained logic is small, is to create a Pblock to hold both the critical path as well as the unconstrained logic.

If critical paths are between two Pblocks, revise the Pblocks. Consider moving or reshaping the Pblocks so they are closer. Consider embedding one Pblock inside the other. Consider moving logic from one Pblock to the other.

In all cases, if the logic in a critical hierarchy is large, is heavily interconnected, or is being pulled around the chip by scattered loads, do not place it at first. Start working with the timing critical hierarchy that has a good placement. Revisit the hierarchy on a later pass if it is still a problem. If paths are a persistent timing problem consider revising the RTL and resynthesizing.

IP Block Creation and Re-use

Introduction

Often, design teams will implement the same function in multiple designs. To save time and preserve performance, designers may want to re-use block(s) from one design to the next. Different approaches have to be taken depending on the design. If the functionality of the block has to change or if the device family changes, the user may have to modify the RTL and re-synthesize. If the block behaves the same way across multiple chips in

the same device family, the gates and placement may be able to be moved from one chip to the next.

The goal of re-using IP is to save time from one chip to the next. IP re-use will work best if the RTL and placement satisfy the following conditions:

1) The IP logic should exist under a single level of hierarchy
2) The IP block should have all inputs and outputs registered
3) The IP block should meet timing targets
4) The IP block should have the same port list in all versions

The next section talks about how to move a qualifying IP block from one design to the next.

IP Creation

1. Complete the Physical Design

The first step in IP re-use is to create a usable placement within the Pblock. Use the techniques described above to either compete the Pblock only or the top-level design. Place and route the Pblock or entire design to meet the desired size, density and performance targets.

There are some limitations involved with the device type and placement location of the reusable Pblock. If the IP block contains block memories or block arithmetics, the IP block should be placed in the new design with the columns of the block functions in the same relative location inside the Pblock.

2. Import Placement

Import the placement from the successful PAR run. The top-level run is preferred but a Pblock run can be used.

3. Export an IP Block

Export the IP block as an EDIF and UCF with LOC constraints for the Netlist instance with the **File | Export IP** command. The **File | Export IP** command will export the files in a logical format consistent with the original EDIF netlist. This makes re-use much easier at the RTL level since a static netlist is being implemented. Placement can be exported for the IP block as LOC constraints or as a Relatively Placed Macro (RPM). We recommend the using LOC flow. There are more issues with re-using a block created in the RPM flow.

IP Re-use

4. Create RTL Black Box for IP Pblock

The new design should contain a black box in the RTL and a netlist for the reusable IP Pblock.

If using XST synthesis, the exported IP Pblock .edn file can be used to derive a timing shell to be used during synthesis. Refer to the Xilinx *XST User Guide* for more information on how this is done.

5. Import the New Design

Import the EDIF and UCF files for the new design using the **File | New Project** command. Ensure the path for the Import Netlist step includes the folder where the exported IP resides. Create a Floorplan for the new design.

6. Import the Placement Constraints for the IP Pblock

Select the reusable IP instance in the Netlist view. Select the **File | Import Constraints** command and use the dialog box to select the exported UCF file for the IP block. Make sure that the ***Instance*** mode is set properly in the *Import Constraints* dialog.

The Pblock rectangle and the placement constraints should now be imported into the floorplan.

Maximizing Device Utilization

Compressing Pblocks

It can be useful to export a single Pblock and run place and route on it in isolation. Afterwards, the map report will indicate how the block may be resized. Repeat until just before the placer fails, which indicates the smallest Pblock possible has been achieved. By doing this on each Pblock, place and route will have a much better chance of fitting a design with high resource utilization into the device.

Containing Multiple Object Types (Block RAMs, MULTs, DSPs, etc…)

When many Pblocks contain multiple object types such as Block RAMs, MULTs, DSPs, etc. it makes it difficult to draw rectangles to accommodate them. The number of rectangles vs. sites required becomes limited. Often the Pblock has little slice utilization after it has been sized to fit the Block RAMs, MULTs, DSPs, etc.. There are a couple approaches to try here.

Adding Pblock Rectangles

Additional Pblock rectangles may be added to include the other required resources using the **Add Pblock Rectangle** popup menu command. Pblocks with multiple rectangles appear with a dashed line connecting them. It is possible to have one or more rectangles constraining only slices, and other rectangles constraining only other resources. Refer to the *Adding Rectangles to a Pblock* section of the *PlanAhead User Guide* for more information. You can merge or nest one Pblock inside a larger Pblock to more tightly constrain a critical path or allow for tighter logic packing.

Assigning Objects Outside the Pblock Rectangles

The PlanAhead tool allows users to assign certain types of logic to sites outside of the Pblock rectangle in which it is assigned. Refer to the *Using Placement Constraints* section of the *PlanAhead User Guide* for more information.

Helpful Synthesis Tips to Reduce Area

The PlanAhead tool integrates seamlessly into the FPGA design and implementation flow. Therefore users can explore area optimization opportunities using synthesis and ISE tools in conjunction with the PlanAhead features mentioned above. Following are some of the features that users can explore for area optimization through their synthesis tools.

1. RTL Coding –

If the RTL is coded inefficiently, the synthesis tools will create structures that are redundant or unnecessary. Some examples of inefficient RTL code are as follows.

There may be a logic block for which the conditions that can activate it are defined in the RTL code and no other conditions can trigger it. If the RTL source is Verilog, the user can employ a “full_case” directive to indicate this scenario to the synthesis tool. Synthesis tools can make use of this information to produce logic to cover a minimal set of conditions. If the “full_case” directive is not specified, the synthesis tool has to synthesize logic to cover all possible conditions.

There may be a case structure coded to ensure that out of all the conditions that can activate it, only one condition can occur at a time. If the RTL source is Verilog, the user can employ a “parallel_case” directive to indicate this scenario to the synthesis tool. The synthesis tool can make use of this information and unnecessary priority logic will not be generated.

2. Utilizing Virtex-4 and Virtex-5 device resources –

Virtex-4 and Virtex-5 devices have a number of new features including embedded DSP functions and embedded FIFO blocks. The CLB structure has also changed in these devices. Designs should be resynthesized for these families. PlanAhead has DRCs to check whether a netlist is using the DSP48 and FIFO16 blocks optimally.

3. Synthesis Tool Options –

With increasing design sizes users are looking to quickly verify their designs. Use of formal verification is growing, and synthesis tools have started providing options to enable the formal verification flow. These options typically result in disabling optimizations that can break the formal verification flow and can result in increased design area. Users who are not interested in enabling formal verification flow should turn the formal verification option OFF.

For large fanout nets, synthesis tools typically replicate logic to optimize the design performance. Users who are more interested in reducing area than improving timing can make use of proper directives to disable this replication (such as syn_replicate and syn_maxfan for the Synplify tools).

Synthesis tools typically perform various optimizations depending on the number of states in a state machine. These optimizations are typically geared towards improving timing performance and increase the design area substantially. Users who are interested in reducing area can disable state machine optimizations completely or instruct the synthesis tool to use sequential encoding to minimize area through proper directives (such as syn_encoding for the Synplify tools).

Depending on the design constraints, synthesis tools choose the level of parallelism in the design implementation. Users who are more interested in optimizing a design for area can instruct the synthesis tool to share resources across different logic processes (in turn reducing the parallelism in the design and possibly affecting design timing). This can be achieved through proper directives or options (such as resource sharing ON for the Synplify tools).

Synthesis tools can employ advanced optimizations like pipelining and retiming to improve the design timing performance. Timing can be significantly improved. However, there can be a huge register increase due to the replication of entire cones of logic. Users who are more interested in optimizing a design for area should turn these advanced optimizations OFF through proper directives or options (such as retiming OFF for the Synplify tools).

Synthesis tools provide various options to direct the level of timing and area optimizations. In the extreme case where a user is interested only in fitting the design in to a given device and is not concerned about the operating speed (e.g. in an ASIC prototyping application), users can remove timing constraints and instruct the synthesis tool to optimize for area only through proper directives or options (such as setting global frequency to 0 MHz and removing all timing constraints for the Synplify tools).

Using Platform Studio and EDK with PlanAhead

The Xilinx Platform Studio (XPS) tool is used by FPGA designers to build embedded systems on Xilinx FPGA devices. XPS provides a comprehensive environment for designers to integrate their hardware and software system components. To ease the integration process, XPS tries to make use of preset design tool options while implementing the embedded system. Using preset options makes the process of system implementation simpler but can result in the system not meeting desired performance goals. In such cases, designers need to have an option to enhance their system performance. The PlanAhead tool has been proven to help users in hardware design analysis and physical design for performance improvement. Design flow using XPS and PlanAhead is shown in the block diagram below (Figure 8).

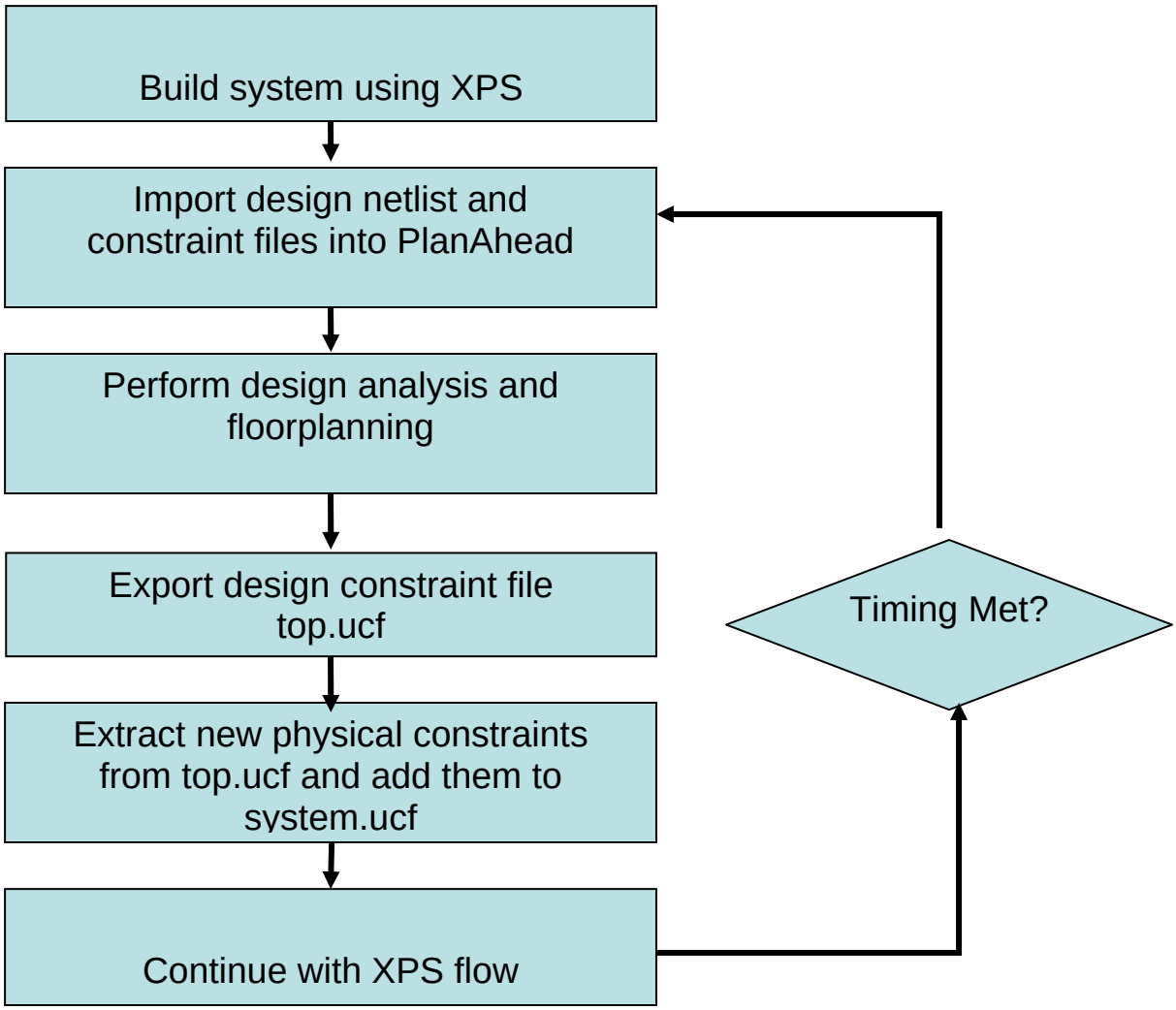

Figure 8: PlanAhead Design Flow with EDK

In XPS, use either the **Tools | Generate Netlist** or **Tools | Generate Bitstream** options to create "synthesis" and "implementation" sub-directories. Synthesis scripts (.scr), project files (.prj) and report (.srp) files are stored in the "synthesis" sub-directory. Design netlist files created as a result of the synthesis process are stored in the "implementation" sub-directory. XPS also creates .bmm files in the "implementation" sub-directory for configuring the Block RAMs on the device. The PlanAhead tool is inserted in the design flow after synthesis has been performed.

XPS uses the XST (Xilinx Synthesis Technology) tool for design synthesis. The synthesis netlists generated are in NGC format. The top-level file is named "system.ngc" and all other files have the nomenclature "module_name.ngc". The design constraints are stored in a file named "system.ucf".

The following steps are used in this flow:

1. Build the system using XPS.
2. Use the **Tools | Generate Netlist** or **Tools | Generate Bitstream** options in XPS to create the system.ngc and system.ucf files, along with the NGC and NCF files for sub modules.
3. Import the system.ngc, system.ucf and core files into the PlanAhead tool.
4. Use the analysis and physical design capabilities of PlanAhead to explore the design.
5. Create physical design constraints if necessary, as determined in Step. 4.
6. Use the **File | Export floorplan** command to export the top-level UCF file, top.ucf. Under the *File types to generate* header, clear the check mark next to ***Netlist***.
7. Manually extract the new physical constraints from the top.ucf file and add them to the XPS generated system.ucf file.
8. Run place and route through XPS.
9. If the design performance has improved as desired then continue with the post placement and routing XPS flow.
10. If design goals are still not met, return to the PlanAhead environment for further analysis. Determine the best choice of RTL changes, floorplanning, etc. to proceed.