WP361 (v1.1) March 31, 2010

Maintaining Repeatable Results

By: Kate Kelley

Meeting the timing requirements in a design can be difficult in itself, but producing a design whose timing is 100% repeatable can sometimes seem nearly impossible. Fortunately, there are design flow concepts that can help to maintain repeatable timing results.

The four areas that have the most impact are:

- HDL Design Practices
- Synthesis Optimizations
- Floorplanning
- Implementation Options

HDL Design Practices

Designs with very high resource utilization and frequency (QoR) requirements are the most challenging from which to get repeatable results. They are also the designs that need a repeatable results flow the most. The first step in getting repeatable results is to use good design practices. There are several papers already written on this topic, but here are some highlights and general recommendations.

HDL Modifications

Often, HDL modifications are the best way to fix timing issues. If two different paths require the same resource (component or routing) to meet timing then one of the paths will fail. If the HDL is modified to make timing easier, then this will not be an issue.

Good Hierarchical Boundaries

Following good hierarchical boundary practices helps to keep logic together, which helps to maintain repeatable results when making design changes. Several rules to follow include:

- Put logic that needs to be optimized, implemented, and verified together in the same hierarchy.
- Register inputs and outputs of modules. This keeps the timing paths contained within a module.
- Keep all logic that needs to be packed into larger FPGA resources (block RAM or DSP, for example) in the same level of hierarchy.

Logic Levels

Repeatable results are very difficult to obtain from designs that have too many LUT logic levels for the required QoR results. Often, it is not the LUT delay that is the issue, but rather the routing delay *between* the LUTs. This is extremely important in the high-performance areas of the design.

Here are some common sources of too many logic levels:

- Large *if/else* constructs.
- Large *case* statements — When appropiate, use "full_case" and "parallel_case" Verilog directives to optimize the case statement with less logic. This often results in fewer logic levels.
- Multiplexers/Decoders — Large multiplexers/decoders can create routing congestion resulting in unrepeatable results. A multi-stage registered multiplexer/decoder path can help with this issue.
- Adders — Use an adder chain instead of an adder tree. Adder chains improve performance while lowering device utilization and power. (They do, however, introduce latency into the design.)

For more information on best practices for coding, see WP231, *HDL Coding Practices to Accelerate Design Performance*.

Resets and Other Control Signals

The choice of resets affects the performance, area, and power of a design. A global reset is not necessary for circuit initialization on power-up, but a global reset can have a major effect on the type of resources that can be used in a design. For example:

- Shift registers (SRLs) cannot be inferred if there is a global reset in the HDL. One shift register versus ten registers produces more repeatable results.
- The DSP and block RAM registers only contain synchronous resets. If an asynchronous reset is used in the code, these registers cannot be used, forcing the use of CLB registers. The same results are easier to maintain if the registers are packed into the DSP and/or block RAM.
- Using synchronous resets on general logic might reduce the levels of logic. The slice registers can have asynchronous or synchronous resets. If the design uses the synchronous reset, then the synchronous set is available for use by the combinatorial logic. This could reduce the logic levels by one LUT.
- A control set consists of a unique grouping of clock, clock enable, set, reset, and in the case of distributed RAM, write-enable signals. Control set information is important because registers must share the same control set to be packed in the same slice. This can affect packing and utilization, creating repeatable result issues.

For more information on resets, see WP272, *Get Smart About Reset: Think Local, Not Global*. For more information on control sets, see WP309, *Targeting and Retargeting Guide for Spartan-6 FPGAs*. While this white paper is specific to Spartan-6 FPGAs, it contains good general information applicable to all FPGAs.

Understanding FPGA Resources

It is important to understand what FPGA resources are available and when it is best to use them. Often, there are synthesis directives to define which resources should be used.

- Block RAM — Best for deep memory requirements.
- Distributed RAM — Works well for wide buses, especially where high-speed data is being clocked by regional clocks.
- RAM control signals — Both block RAM and distributed RAM can have issues with large fanout on control signals. Duplicating control signals and using floorplanning techniques to keep blocks with the same signals together can help maintain repeatable results.
- Shift Registers — SRLs can reduce the utilization of a design, which helps repeatability. There are several performance issues to keep in mind:
    - Clock-to-out of an SRL is slower than clock-to-out of a FF; therefore, it is useful to use a FF as the last stage of a shift register. Most synthesis tools do this automatically, but if there is an issue with a path involving shift registers, it is good to confirm that the last stage is a register.
    - Having a FF in front of an SRL gives the placer more options to meet timing, therefore maintaining results. Again, most synthesis tools do this automatically, but if there is an issue with a path involving shift registers, it is good to confirm that the first stage is a register.
- Registers — FPGAs have many registers, making pipelining a useful technique to improve performance. It is important to disable SRL inference on multiple pipelined flip-flops.

More information on block RAM can be found in WP231, *HDL Coding Practices to Accelerate Design Performance*.

For more information on shift registers, see WP271, *Saving Costs with the SRL16E*.

Clock Domain Issues

Paths that cross unrelated clock domains must be correctly constrained. Clocks from the same source clock (e.g., a DCM) are automatically related by the tools. The PERIOD constraint can also relate external clocks.

Unrelated clocks that are not created internal to the device take special consideration. By default, these clocks are not constrained by the tools. If there are special timing considerations, the FROM:TO constraint needs to be used to correctly constrain the path. The DATAPATHONLY keyword tells the tools to not include clock skew in the equations.

For more information, see the "Asynchronous Clock Domains" section in UG625, *Constraints Guide*, or WP257, *What Are PERIOD Constraints?*

It is also important to ensure that race conditions do not occur. FIFOs can be used when crossing from one domain to another. If not, one and only one control signal should be double-synchronized, and this control signal should be used in the receiving clock domain to receive other signals.

High-Fanout Signals

Often, high-fanout signals can be the gating factor in a design. Even though most synthesis tools have fanout control, it is suggested that these signals be duplicated in the HDL to get more repeatable results. This needs to be combined with directives to ensure that the duplicates are not removed by the synthesis tools. If a high-fanout signal is in the top-level logic, one method is to duplicate the signal and then drive each top-level module with a separate signal.

If the synthesis tool fanout control is not giving the desired results and modifying the HDL is not palatable, then using Register Duplication within MAP along with the max fanout constraint often makes better register duplication choices than synthesis. For more information, see MAX_FANOUT in UG625, *Constraints Guide*.

Signal Names

As a general debug issue, it is easier to trace a problem path if a signal name is kept constant when crossing hierarchies. If the name constantly changes, it is difficult to follow in the timing reports and other debug output.

It is also helpful to put the signal direction on the port definitions for all modules or entities.

Synthesis Optimizations

Synthesis has a big effect on repeatable results. If the output netlist from synthesis is not optimal, then it is impossible to have ideal conditions in the implementation tools.

Several synthesis techniques can be used to help improve implementation results.

- Use timing constraints when running synthesis. It is common to over-constrain during synthesis, then relax the timing constraints in the Xilinx® implementation tools. This makes the synthesis tool work harder, relieving the burden on the implementation tools.
- Use the timing report from the synthesis tools. If a path is failing timing in synthesis and implementation, modify the HDL or synthesis options to meet timing after synthesis. This saves time during implementation runs.
- The best way to have repeatable results in the implementation tools is to have

repeatable results during synthesis. Most synthesis tools support a bottom-up flow, with separate synthesis projects for the top level of the design and each of the lower-level modules. The user is in control of which netlist is updated, based upon HDL changes.
- Most commercially available synthesis tools have an incremental flow.

Floorplanning

Floorplanning locks placement of components to a specific location or to a range. This reduces the variability of placement, increasing the repeatability of a design. Better performance can almost always be obtained by floorplanning and/or using location constraints. That said, a bad floorplan or poor location constraints can make it impossible to meet timing. Floorplanning is somewhat of an art and requires advanced knowledge of the tools and the design. Implementation results that meet timing can be used as a guide to create a good floorplan.

There are several different layers of floorplanning:

- Pinout Selection
- Area Group Floorplanning
- Locating Core Blocks
- Locating the Entire Module
- Locating the Critical Path

Pinout Selection

If pinouts are selected mainly by board requirements, FPGA implementation tools might have a difficult time maintaining repeatable results. The following suggestions can help achieve repeatability:

- Be aware of the data flow. For example, data can go from the center I/Os to the side I/Os.
- Keep all of the pins associated with the bus in the same area of the FPGA to limit the routing distance on control signals.
- I/O bus control signals should be placed near the address and data buses.
- Signals that are to be optimized together need to be placed together.
- If board routing is a bigger concern, pipelining registers on the I/Os can help FPGA routing with less than ideal pinouts.

Area Group Floorplanning

Area group floorplanning is very easy to do, but it is often misused, creating poor floorplans that create more issues than they resolve.

Some general guidelines for good floorplans include:

- Keep utilization similar across all area groups. For example, do not have one at 60% and another at 99%.
- Do not use overlapping area groups. The one exception is if two different area groups have some logic elements that need to be placed together, it is acceptable to overlap by one or two rows or columns of CLBs. The user is then responsible for making sure there are enough resources for both area group constraints. If there are two different logical portions of the design that need to be in the same physical location, put both logical portions into the same area group.

- One level of nesting is typically acceptable. This can be necessary if a small portion of a larger area group needs to be located in a tight region.
- Floorplan only the critical portion of the design.
- Logic connected to fixed resources (e.g., I/O, transceiver, or processor blocks) might benefit from floorplanning.
- Use the results of a good implementation run as a guideline to identify placement or timing issues. Tools like PlanAhead™ software and Timing Analyzer can help visualize the issues.
- It is often helpful to minimize the number of regions used for each global clock and the number of clocks (regional and global) in each region. Do not overconstrain and plan accordingly if more logic is going to be added to a clock region. When all clocks in a clock region are used, it can be difficult to find a valid placement. Use the PlanAhead software's ability to snap to a clock region to make this floorplanning easier. For Virtex® FPGA designs with more than 10 global clocks, the clock regions used in the current implementation are in the `.map` report file. UCF constraints are also included in this file.

For more information on area group floorplanning, see UG632, *PlanAhead User Guide* and UG633, *Floorplanning Methodology Guide*.

Locating Core Blocks

Often, locating the core components, such as the block RAMs, FIFOs, DSPs, DCMs, and global clocking resources can help achieve repeatability. This is best done by looking at a good placement and using the design knowledge to verify these are good locations. Control signal and data flow (bus alignment) should be taken into account when locating these block RAM, FIFO, and DSP components. Constraints to locate the clock regions of an existing design are found in the `.map` report file. Keeping the same clock regions prevents the placer from making changes to the clock region partitioning, which could change the floorplanning of the design. Use **`reportgen -clock_regions design.ncd`** to create the report.

Locating the Entire Module

PlanAhead software has the ability to lock down all the placement information on critical modules. On the next run, the placement is the same, but the routing information is not saved. More information on location constraints in the PlanAhead software can be found in the Floorplanning the Design chapter in UG632, *PlanAhead User Guide* and in the PlanAhead Tutorial.

Locating the Critical Path

If locking down an entire module is overkill, it is possible within PlanAhead software to lock down a critical path. This technique should be used in a very limited manner. If there is a specific path that is causing the majority of issues, it is better to fix the timing issue by changing the HDL. If this is not possible, limited use of locating specific timing paths can be helpful.

Implementation Options

Several options in the implementation tools improve repeatability. Here are some general guidelines that can help maintain repeatable results:

- Design Preservation using Partitions is the best methodology to preserve implementation, but it is not a good fit for all designs, and it does have HDL design requirements.
- SmartGuide™ technology is another option for maintaining repeatable results. This is best for designs that are not pushing the absolute maximum QoR or utilization.
- If Design Preservation or SmartGuide technology is not appropriate for the design, then use SmartXplorer or PlanAhead software strategies to maintain timing.
- For designs with high QoR requirements, there are advanced implementation options to help maintain timing.
- Managing utilization is key to maintaining repeatable results.
- Staying with the same software release for the entire design phase helps achieve repeatable results.

Design Preservation

The Design Preservation flow makes use of Partitions. This is the only method that guarantees repeatable results. The main goal of design preservation is to enable consistent module performance to reduce the amount of time in the timing closure phase. It also requires the greatest commitment from the user to following good design practices.

Partitions preserve unchanged portions of the design that have been previously implemented. If a Partition's netlist is unchanged, the implementation tools use a "copy-and-paste" process to guarantee that the implementation data for that Partition is preserved. By preserving implementation results, Partitions enable the modified portions of the design to be implemented without affecting the preserved portion of the design.

In 12.1 and future releases, Design Preservation is supported in the PlanAhead software and command line tools. For more information on Design Preservation, see WP362, *Repeatable Results with Design Preservation*, and UG748, *Hierarchical Design Methodology Guide*.

SmartGuide Technology

SmartGuide technology uses the previous implementation results as a starting point when running implementation. The main goal of SmartGuide technology is to reduce runtime. Guided placement and/or routing can be moved in order to route the design or meet timing. SmartGuide technology works best for designs that are not trying to push the limit on QoR or utilization.

In previous versions of the tools, there was an exact guide and a leveraged guide. Often, the exact guide methodology resulted in unroutable designs. If exact preservation is required, then the suggested flow is Design Preservation. Leveraged guide has been replaced with SmartGuide technology.

It is often asked whether SmartGuide technology or Partitions should be used. The answer depends upon where the designer is in the design flow. SmartGuide technology should be used at the end of the design cycle when making small design changes. Using this flow, it is easy to see if the proposed changes work with the design. Partitions require a greater commitment up front to following good design hierarchy rules. The Design Preservation flow using Partition should be decided when starting to organize the HDL. An exception to this rule is when the design already follows the hierarchal rules for Partitions.

For more information, refer to the Design Considerations chapter of UG626, *Synthesis and Simulation Design Guide*.

SmartXplorer

SmartXplorer and PlanAhead software strategies are similar tools that help achieve timing closure. Both of these tools run different sets of implementation options to find the best fit for the design. These results can then be used to see what placements tend to have better timing results. These placement results can then be used to create good area group floorplanning. The different results can also point to a design issue. If the same path is failing across all runs, it is beneficial to change the HDL to remove the timing issue.

Another good use of SmartXplorer (Variability Passes) and PlanAhead software is to run multiple implementations with different cost tables. A cost table affects the initial placement of design. Often, a different starting point allows a design that is on the edge of meeting timing to meet timing. Running the first ten cost tables is usually sufficient. Once a good cost table is found, it can usually be used for several design iterations. When it no longer works, the user needs to search for a new one. SmartXplorer also supports different options for XST synthesis. The best synthesis options for user designs should be found using this approach.

For detailed information on both of these tools, refer to WP287, *Timing Closure Exploration Tools with SmartXplorer and PlanAhead Tools*.

Advanced Implementation Options

In the beginning of the design, it is recommended to use the default effort levels for MAP and PAR. Using too many advanced options in the beginning can hide a timing issue that might be best solved by modifying the HDL. When the device utilization increases, it becomes harder and harder for the tools to converge on a solution that meets timing. If the default options are used, then the higher effort options are available to get the last few picoseconds of timing later in the design flow, allowing timing results to be maintained.

More information on the implementation options can be found in UG628, *Command Line Tools User Guide*.

Managing Utilization

Designs with low utilization of LUTS/FFS (<25%) or high utilization of LUTS/FFS (>75%) can be difficult to place and route with consistency. For designs with high utilization, look at slice control sets, resets (often synchronous resets/sets are not required with FPGAs), modules with higher than expected logic usage (easily done in PlanAhead), or if SRL/DSP48 inference is happening.

The flip side of high utilization is low utilization. With designs that have 25% utilization or less of all component types, the low utilization algorithm takes effect and keeps components tightly placed. However, if I/O utilization exceeds 25%, then the implementation tools could spread out the design in order to keep logic near the I/Os. Careful placement of I/Os and use of area groups can minimize this issue.

Software Releases

Try to use the same major software release during the timing closure phase. As algorithms change from one release to another, a technique that worked in one might not work in the other. Also, methods that rely on the previous results (Partitions and SmartGuide technology) might not work across major releases.

Conclusion

There are many different ways to influence design repeatability. The best way is to follow good design methodology in the HDL and fix any timing issues by changing the HDL. If that is not possible, there are several different methods using synthesis, floorplanning, and implementation techniques. Design Preservation using Partitions is the flow that guarantees instance performance. SmartGuide technology is another solution that uses previous implementation results. If a design has very high utilization and QoR requirements, the best methodology might be the simple flat flow using different floorplanning and implementation options via PlanAhead software or SmartXplorer.

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
02/04/10	1.0	Initial Xilinx release.
03/31/10	1.1	Updated for ISE Design Suite 12.1 release. Updated Understanding FPGA Resources, page 3, Area Group Floorplanning, page 5, Design Preservation, page 7, SmartGuide Technology, page 7, and Advanced Implementation Options, page 8.

Notice of Disclaimer