

HDL Coding and Design Practices for Improving Virtex-5 Utilization, Performance, and Power

These tips and techniques can lead to better Virtex-5 designs.

by Brian Philofsky
Staff Software Technical Marketing Manager
Xilinx, Inc.
brian.philofsky@xilinx.com

FPGAs have been very flexible in accommodating any HDL coding or design style for digital logic; Xilinx® Virtex™-5 devices are no exception. Although Virtex-5 FPGAs can accommodate many different types of designs written in many different methods, certain recommended constructs and manners can achieve improved optimization in terms of area, performance, and power.

Know Your Target Architecture and Synthesis Tool

Before beginning any project, you should understand the device architecture you are targeting. For Virtex-5 FPGAs, I recommend reading the Virtex-5 Users Guide (<http://direct.xilinx.com/bvdocs/userguides/ug190.pdf>) before starting your first line of code. Once you have a better understanding and vision as to how your code will ultimately result in the base hardware, you can make both large and small design and coding decisions confidently.

For example, if you know of and use Bitslip technology within the ISERDES, you could save time, effort, and resources by capturing input data rather than attempting to describe and build similar circuitry.

In another example, if you know the structure and capability of the DSP48E, you can make better choices as to when and where to place pipeline registers. Dedicated features like the wider multiplier or post adder can also help you achieve better area, performance, and power.

Similarly, knowing the capabilities and current limitations of your synthesis tool can not only help when choosing coding styles to properly infer primitives but can also give you greater insight as to when to instantiate a component or use inference. Review synthesis manuals, application notes, or other relevant materials before starting so that you know the recommended coding styles for the synthesis tool you are using.

You should also update and use the latest versions of synthesis and ISE™ tools before beginning a project. Although initial synthesis support for the Virtex-5 architecture is strong, many improvements in optimization and inference support are still to come with new releases. One easy way to ensure more optimal designs in

terms of area, performance, and power is to install the latest version of the software.

Control Signal Polarity

The Virtex-5 architecture can support different control signal polarity (clock enables, resets, or sets). However, to have the most optimal design, I recommend consistent use of active high control signals in your design. The Virtex-5 slice control logic is active high, and when described in this same manner in the code should never require additional LUT resources for a simple signal inversion.

If the signal comes from an external pin and needs an active low polarity, I suggest inverting the signal in the top-level code and using a positive polarity in all processes and sub-modules requiring that signal. This is critical for designs that have several cores, use bottom-up synthesis techniques, have KEEP_HIERARCHY constraints, or employ the use of partitions (Figure 2).

Designs that fall into these categories are more susceptible to the use of additional LUTs per core/netlist/hierarchy/partition for the sole purpose of inverting these control signals, which not only consume extra LUT resources but may also have negative effects on performance and slice

packing. As a general rule, always code sets, resets, and enables with an active high (logic 1 activates) polarity.

Use of Resets

It is common practice to use a global asynchronous reset in the source HDL code to initialize the design; however, in many cases this consumes additional resources. Instead, think synchronous and local. I suggest describing a synchronous set/reset logic to the portions of the design that do need periodical resets. For those portions of the design that do not, you can initialize the signals defined to be registered in the HDL code at the time they are declared (for example, when defining a reg in Verilog or a signal in VHDL). This methodology allows for improved packing density, enhances timing analysis and performance, and can improve area resources.

In terms of FPGA behavior, without a global reset described in the code, a GSR (global set/reset) will occur upon completion of the configuration cycle, initializing all registers to known specified values. This same cycle is also simulated in the gate-level simulation netlist, giving the same known starting point as in the FPGA.

In terms of RTL simulation, having the registers initialized in the code allows for proper RTL or behavioral simulation; this same initialization will be picked up by the synthesis tool and applied to the implemented design. Therefore, for simulation at any stage, a global reset is redundant and unnecessary.

Using a synchronous reset instead of an asynchronous reset also allows for more predictable behavior upon the assertion and release of the reset, because the synchronous signals are automatically analyzed and their behavior is more deterministic when all timing constraints are met. It also allows for the possibility of greater logic optimization and performance because it is not global.

When using synchronous control signals, you can move portions of the logic function to the synchronous set or reset of the flip-flop; this is not possible with asynchronous signals. By only describing a reset where necessary, the synthesis tool can use alternative resource choices like SRLs (shift

register LUTs); distributed RAM (LUT-based RAM) memory; or block RAM for the implementation, which would not be otherwise possible nor optimal. The synthesis tool has maximum flexibility to choose the best resource for the described code.

Pipelining

As with previous FPGA generations, properly pipelining your design is necessary to achieve top performance and improved power characteristics. With the introduction of the Virtex-5 architecture, a new logic structure dictates slightly different rules regarding when and how to pipeline.

The Virtex-5 device departs from the traditional four-input LUT in previous FPGA families and has an enhanced six-input LUT (6-LUT), allowing for wider logic functions between pipeline registers while maintaining top performance. You should keep this in mind, as logic functions coded into HDL as optimal code should include six inputs to the logic function between registers to get the most optimal pipelining and LUT resource management.

In cases where it is not practical or possible to have exactly six inputs in a given logic function, the wider input 6-LUT still allows for good performance by

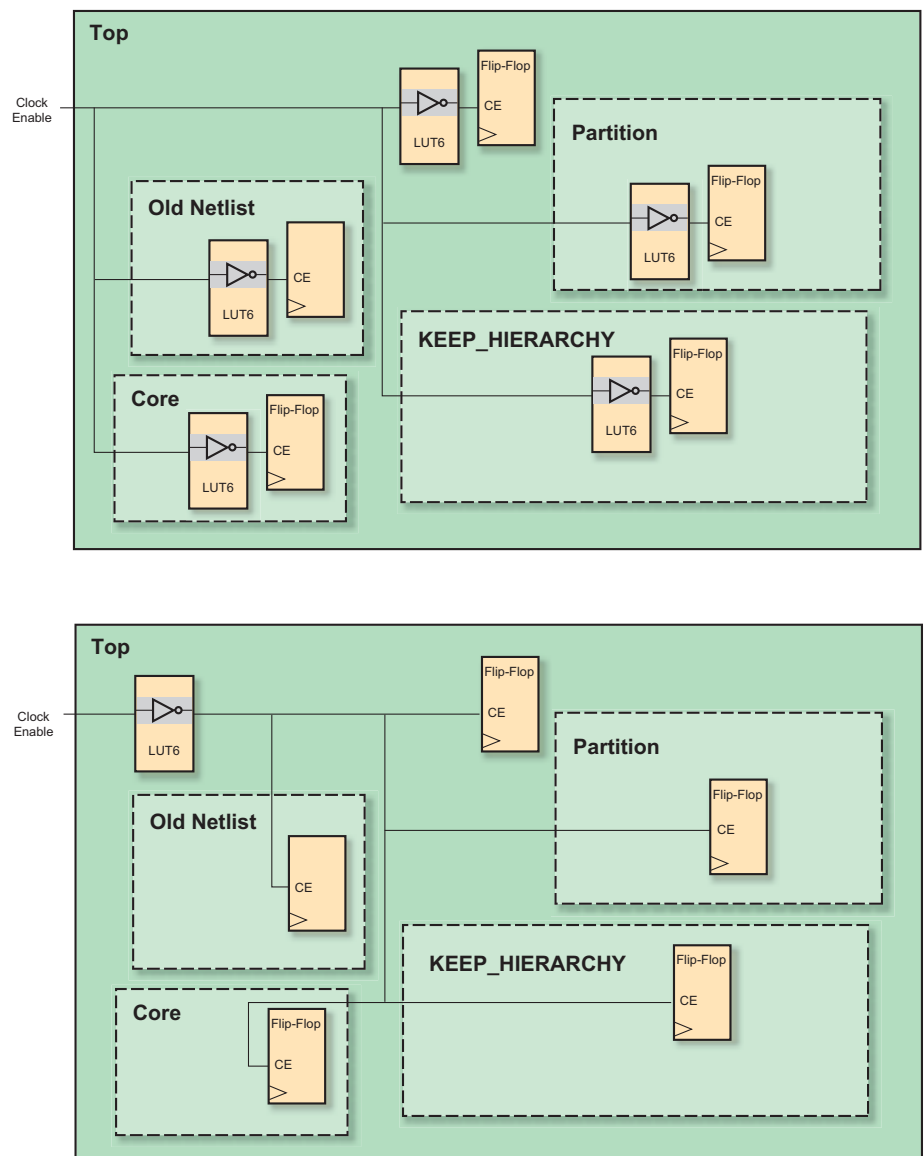


Figure 2 – How clock enable polarity affects LUT utilization in a design



reducing the number of logic levels, thus requiring fewer pipeline stages to achieve the same as or better performance than previous FPGA architectures.

A good goal is to aim for less than 10 inputs to a given logic function between I/Os, registers, or synchronous blocks (like block RAM or DSP48Es), which generally would represent two logic levels. When you need a significantly higher number of inputs for the design path to meet latency or other requirements, you can attempt to reduce the fan-in to that logic function (when possible) if high performance or low power are your design objectives.

Coding Memories

Among other innovations within the Virtex-5 architecture, Xilinx has enhanced both block RAM and distributed RAM memories with greater capacity and capability. You must make different decisions early in the design process and while coding to get the most from these valuable resources.

General guidelines call for inferring RAMs when possible for easier code changes, faster simulation, and more portable code. However, even when behaviorally describing the RAM, you should keep some important things in mind. The first and most obvious thought is RAM capacity. In terms of block RAMs, the base memory block increased in Virtex-5 devices to 36 Kb of memory storage space. You can configure this block to the wider but shallower 512 x 72 configuration, the deeper single-bit width 32 Kb x 1, or several configurations in between. It is also possible to cascade two 36-Kb RAMs to form a 64-Kb x 1 configuration or break up the 36-Kb RAMs into two separate 18-Kb RAMs capable of 512 x 36 to 16-Kb x 1 configurations.

Distributed RAM have benefited from the larger LUT structure and can now efficiently accommodate 64-bit depths without any area or performance penalties. This is the most optimal size for this type of RAM in the Virtex-5 device, although other sizes can be accommodated. The base RAM sizes are important to remember during memory selection and coding to most efficiently use the limited RAM resources in the device and achieve the best performance.

Both block RAM and distributed RAM memories also have additional capabilities that require different coding and design considerations. For performance, perhaps the most important is the proper use of output registers. For block RAMs, this means enabling the output registers to the block RAM whenever possible. By enabling the output registers, a reduced clock-to-out is realized from the RAM, thus improving timing for the data leaving the RAM. However, an extra clock cycle of latency is added during reads, for which you must account.

Similarly, when using distributed RAM, the output of the RAM can be asynchronous; however, coding it synchronously will allow the use of the register within the slice, providing better timing characteristics and reducing the chance of the RAM being part of the timing bottleneck.

There are more advanced features of the block RAMs, such as FIFO and ECC (error correction circuitry) capabilities. The distributed RAM also has new capabilities such as a quad-port configuration. In some cases, these features cannot be realized by inference within synthesis and instantiation is necessary. If you need such functionality, I suggest instantiating the RAMs either by generating cores within Xilinx CORE Generator™ software or by instantiating the base primitive. Taking advantage of these advanced features can save RAM and logic resources as well as improve area, performance, and power.

Some General Guidelines

A few other general recommendations do not fall into any specific categories but can result in better coding and design choices. First, you should make wise choices in terms of your design hierarchy right from the start. Your choice of hierarchy can have effects on the synthesis and implementation tools' ability to optimize the logic paths.

In general, do not allow timing paths to cross multiple boundaries of hierarchy. This not only limits the tool's ability to optimize logic but may also limit your options for design implementation and design debugging. For instance, you may not be able to use partitions or KEEP_HIERARCHY on certain hierarchies with this practice.

For designs in which some or most of the code was created for an architecture other than Virtex-5 FPGAs, I suggest that you review the code to ensure that it is well suited for implementation into the new architecture. A few minutes of time spent here can save several hours later if you identify and correct suboptimal code.

If your design contains cores or pre-compiled netlists (EDIF or NGC files) from a previous architecture, you should regenerate those targeting Virtex-5 devices. Unless regenerated, netlists optimized for a previous architecture are more likely than not far less optimal when targeting Virtex-5 architectures.

One last suggestion is to use the HDL language templates within the ISE tools. They not only help with accelerating the generation of VHDL or Verilog code, but also provide assistance in creating more optimal code for FPGAs. They also cut down on the possibility of creating syntax or other simple but common mistakes that can hold up the testing and verifying of HDL code.

Figure 3 shows both Verilog and VHDL code following the guidelines discussed here.

Conclusion

Coding styles are very individual; however, following these suggestions makes it more likely that you will achieve a more optimal result. These guidelines do not represent absolutely everything you need to know to achieve the best Virtex-5 design possible, but I have provided some common strategies that can help in achieving more optimal designs.

Almost any set of valid HDL code likely will result in a functioning design, but following a few simple guidelines can help in terms of improved density, performance, and power, and many times may reduce the amount of time it takes to ultimately complete a design.

For more information, see the Synthesis and Simulation Design Guide at <http://toolbox.xilinx.com/docsan/xilinx82/books/docs/sim/sim.pdf> or White Paper 231, "HDL Coding Practices to Accelerate Design Performance," at <http://direct.xilinx.com/bvdocs/whitepapers/wp231.pdf>.