Design for Speed

These tips are intended to show you how to increase the speed of your FPGA design. These tips apply to Xilinx FPGAs and some of the design solutions apply to other vendors, and are not used to "fix" the FPGA, but the design. Lots of designers have trouble increasing the performance of their designs, while others do not understand how to build reliable FPGA designs. Always create a reliable design by following the steps outlined in the FPGA Design Tips and then increase the speed of the design with these tips. Note that these tips can vary somewhat for CPLDs.

1) Use Timing Constraints to communicate your timing objectives

- Failing to use timing constraints will yield modest performance.

- Early Pin-Locking can also limit the ability of the Implementation Tools to reach your timing goal. Give the tools as much flexibility to meet your timing goal by making your pin assignments as late in the design cycle as possible. If you/must lock your pins early, follow the Locking Your Pins Tips sheet.

- Take the time to understand Basic Global Timing Constraints (Period, Offset In, Offset Out, and Pad-to-Pad constraints). Add these constraints as a foundation to constraining your design. Keep these constraints as loose as possible.

- Take the time to understand the use and creation of Path Specific Timing Constraints (Multi-Cycle Paths, Pin Specific Offset In/Out, Constraining between Clock Domains, Creating Custom Groups, Identifying False Paths, etc.). Make these constraints as tight as necessary.

- Note that the Virtex devices (Virtex/E/EM and Spartan II) can have their timing constraints pro-rated from the Constraints Editor. This utility allows you to enter your worst case operating conditions, and have the timing information reported by the Timing Analyzer pro-rated. Note that if your operating conditions ever get worse than the
values you enter into the Constraints Editor, you risk your device failing in-system. Be certain you always enter the **WORST** case conditions that you ever expect to encounter.

2) Use the Logic Level Timing Report (or your Synthesis Tools estimates) to verify that your timing constraints are realistic.

- This is important especially on the first implementation.

- I recommend doubling the delays reported by the LLTR if you are targeting a newer device (Virtex/E/EM or Spartan II) because routing delays are considered zero. If your constraint is shorter than this your compile time is going to be large and the Implementation Tools may not be able to reach your timing goal. Be **AWARE** that this may explain a very long compile time.

- I recommend adding 20% extra delay reported by the LLTR if you are targeting an older device (XC4000/E/XL/XV, Spartan/XL, or older) because routing delays are estimated.

- Note that the LLTR will not change unless the device, speed grade, or the design changes.

3) Use the Post Layout Timing Report to verify that your timing constraints were met by the Implementation Tools.

- This is easier than opening the Timing Analyzer on a very large Virtex design, which might take a couple minutes.

4) Use the Timing Analyzer to generate detailed timing information about your design.

- The Timing Analyzer will provide a wealth of timing information on designs that use timing constraints. Unconstrained designs will generate a Default Path Analysis that is only slightly helpful.

- The Timing Analyzer reports can show users how many levels of logic are being inferred. **This is very important**, since most designers are not aware of how much logic they are generating with their synthesis tool, or how much optimization the synthesis tool is doing for them. If your delay path infers multiple levels of
logic, it will have to be re-synthesized (with code changes or different synthesis option settings) to meet your timing objective.


- The Custom Report, shows all the delay paths between groups of path end points created by selecting Sources and Destinations. This report can be used to find the timing information for a particular delay path without having to review a large report.

- The Report Paths Not Covered Report, shows the all of the delay paths in the design, in descending order of length. This report can be used to find any unconstrained delay paths.

5) Increase the Place and Route Effort Level
   - This will enable the software to work longer trying to implement your design. It is very effective at improving your design's performance, but increases the compile time considerably.

6) Use MPPR to improve the performance of your design
   - This allows the tools to use different algorithms to improve the performance of your design while focusing on improving your placement. Placement is responsible for 80% of your designs performance, not routing.

   - Consider decreasing your routing iterations with each pass of MPPR, and after implementation is completed polishing the routing with the Re-entrant Router.

7) Pipeline purely combinatorial functions.
   - Pipelining involves the addition of registers to a design, especially designs that have multiple levels of logic between synchronous elements.

   - This design technique has been particularly useful since FPGAs are register rich and most designers do not use all of their flip-flops.
• Pipelining also creates latency in a design and requires designers to create additional logic to signal when data is valid.

• In cases where a design has multiple pipelined stages, consider using the Shift Register LUT to balance the stages and save registers.

8) Choose the best State Machine Encoding scheme.

• One-Hot and Gray encoded FSMs tend to have the fewest levels of Next state logic and hence the best speed.

• This generally assures that larger FSMs (greater than 16 states) will be faster if they are not binary encoded.

• Note that custom encoding can be an excellent solution if there is not very much decode logic inferred.

• Creating your Binary FSM in the Virtex Block Ram resources can create a faster and more reliable FSM by transitioning the outputs with a 3.3Ns read access time.

9) Use the Carry Logic resources for fast arithmetic functions

• Carry Logic is the easiest and fastest way to create fast adders, accumulators, counters, subtracters, comparators, etc.
The Virtex MuxCY can also be used to create fast decoders, although this can be challenging to create since these resources will have to be instantiated.

Pre-Scale counters can be used to get even better speed than Carry Logic. Likewise, Linear Feedback Shift Registers can be used in FIFO applications to get better speed than Carry Logic implementations.

10) Duplicating logic on high fan-out nets can decrease long net delays.

- This enables the placement tools to place the replicated logic in different areas of the die, which shortens the associated net delay.

- Net delay and fan-out can be found with the Timing Analyzer reports and the FPGA Editor.

- Duplicate address and control lines to large memories, clock enables, output enables, and synchronous resets.

- The only caveat to this technique is that it increases the total area of the design.

11) Use the IOB resources appropriately.

- They are designed to register your inputs (fast set-up times) and to register your outputs (fast clock-to-output times).

- Input register have variable set-up and hold times. ONs hold time comes at the expense of an increased set-up time.

- Fast slew rate decreases your output transition times (decreases output delays by 30-40%).

12) Use the Re-entrant router.

- This utility is designed to route designs that failed to completely route during the implementation process.
- It can also be used to improve the performance of SOME nets by as much as .3Ns. However, the net you may need to be shorter may not improve at all, so this is not a very good technique toward improving your design speed.
- The Delay Based Clean-up option can be used to improve net delays as described above, and can be very effective at improving unconstrained net delays.

13) Change your device's speed grade.

- This option allows you to choose a faster or slower device to meet your performance specifications.

- The Timing Analyzer can access a new speed file and recalculate your designs internal delays to determine if a faster or slower device will meet your timing specifications. Use the command: Report -> Speed Grade. Moving to a faster device will cost more, but it may save you from costly re-design. Likewise, a slower speed grade can save you money.

14) In general, when these design techniques are followed, your design has a greater chance of running very fast. Note that the Implementation Tools and these tips cannot tell you whether your design will actually function when downloaded to a prototype. Refer to the FPGA Design Tips sheet for more information on building reliable FPGA designs.

*** Note that although these techniques enable you to get better speed out of your design, creative use of the FPGAs resources can sometimes produce even better results.

*** The Timing Analyzer only reports the static delays of your placed and routed design at worst case operating conditions. Your design will probably run faster in your prototype, especially if you are not going to be operating the finished product at worst case operating conditions.