Building Reliable and Efficient FPGA Designs

Adapted from Frank Nelson, the senior terchnical trainer and course designer at Xilinx Inc.

- These tips are intended to show you how to build reliable designs for your FPGA.
- These tips apply to any vendor of FPGAs, and are not used to “fix” the FPGA, but the design.
- Lots of designers create designs that fail because they do not follow similar guidelines.

1. Don’t gate clocks

- Don’t route clock signals through the Look-Up Tables (LUTs). These “internally generated clock signals” can glitch easily.
- Instead of routing this signal to the clock port of the synchronous elements in the FPGA, route them to the clock enable port.
- Not gating clocks also reduces the number of clock signals routed inside the FPGA.
- If you do not have a clock enable resource on each register, this will be implemented as an input to the combinatorial logic driving the register (LUT).

2. Use the global routing resources

- This reduces clock skew which makes the design more likely to run reliably.
- Virtex has fewer global routing resources that are intended for the distribution of clock signals only.
- Older families have more global routing resources that can be used for any control signal (CE, Set, Reset, etc.).
- Use the MaxSkew attribute in the UCF on control signals that are routed on general interconnect, especially clocks.

3.Register Asynchronous input signals

- This means signals which are not registered by the same clock frequency as the FPGA registers may need to be registered, or use a synchronization circuit.
- Consider using the IOB flip-flop to register your input. Note that these flip-flops often have variable set-up and hold times.
- Synchronization circuits may also be necessary for transferring signals between clock domains within the FPGA.
  - Remember that synchronization circuits can prevent set-up and hold time violations on asynchronous inputs, which can cause failure of the circuit due to metastability.

4. Design with a Synchronous Set/Reset

- Use of Asynchronous Set/Reset can easily create circuits that glitch.
- Use the Global Set/Reset (GSR) resources in the older device families to reduce the skew on a set/reset.
- Don’t use the GSR in Virtex. The GSR has too much delay and general interconnect will distribute this signal quickly.

5. Choose the best State Machine Encoding scheme

- Binary encoding is good for creating small (less than 16 states) FSMs in FPGAs.
  - Larger FSMs require too much Next state logic and perform slower when binary encoded.
- One-Hot encoding is good for building larger FSMs since it requires very little Next state logic.
  - OHE is more reliable than Binary encoding, because fewer bits change at once making the circuit less likely to glitch.
  - OHE leaves a lot of unused states in the FSM and also uses a lot of registers (but there are a lot of registers in FPGAs).

7

State Machine Encoding scheme

- Gray encoding of your state machine uses less resources than OHE and also gets good speed.
  - Gray encoding is also the most reliable encoding technique, because only one bit changes per transition.
  - However, Gray encoding requires the designer to be very aware of the coding technique chosen when simulating the FSM, and this is often a pain.
- Note that custom encoding can be an excellent solution if there are not many cases where more than one bit changes at once.

8

FSM Encoding Summary

- Simulate your circuitry in binary encoding and then convert it to another coding technique to get better speed or reliability.
- Binary encoding works best with 8 or fewer states.
- OHE works best for medium sized FSMs (less than 32 states).
- Gray encoding works best for the largest FSMs (it uses fewer registers than OHE and is more reliable).

9

6. Properly code your FSM

- Do not leave undefined states in your FSM. Use a default statement or point the unused states to a reset state.
- Do not infer latches in your FSM. This builds unreliable circuits with designs that mix registers and latches.
  - Assign all outputs a value when in every branch of an if/then and case statement.
  - Use an Else or When Others statement to fix this.
- Use a Case statement. This will create a faster FSM since If/Then statements create priority encoders, which infer multiple levels of logic.

10

Build your state machine with two Always blocks (speed)

- Next State decoding and Output decoding should be placed in one block, while the State and Output registering should be in another.
- This will assure better speed of the FSM when it is targeted for an FPGA. The synthesis tends to optimize the combinatorial logic when these pieces of logic are separated.

11

End your combinatorial processes/always blocks (reliability)

- Use an “else” statement when using if-then when using a Case statement or the “default” in Verilog when using a Case statement.
- This will avoid inferring a latch.
- Use this to make sure you have covered all possible conditions.
- Also be sure that you define all outputs in each branch.

12

Register the outputs of your state machine. (speed)

- This will prevent the logic from being optimized with other hierarchical components combinatorial logic.
- This assures that the outputs of the state machine will not be optimized away and can be simulated and probed.
- Registering the outputs of every component assures that team designers know that each of their inputs are registered and encourages pipelining. *Pipelining is a primary method of improving the speed of FPGA designs.*
- Since state machines are often in their own level of hierarchy, registering their outputs only requires adding a component that is entirely registers.

13

Registered Mealy outputs are preferable to registered Moore outputs. (speed)

- The outputs of the Moore FSM are 1 clock cycle behind the active state (causing additional latency).
- The outputs of the Mealy FSM reflect the active state, resulting in simplified timing and reduced latency.
- Recall that a Moore FSM has outputs that are based on the current state of the machine. Mealy FSMs have outputs that are based on the current state and the inputs.

14

"look-ahead" Mealy

- It is suggested that you use a "look-ahead" Mealy state machine if possible.
  - By using the inputs to look-ahead at the state you will be in on the next rising clock edge (next state) and set the output during the transition to reflect that state.
  - A "look-ahead" Mealy state machine looks at the current state, the inputs, and what the next state will be, so that the outputs always reflect the current state.
- Mealy…
  - based on inputs and current state
- Mealy look-ahead…
  - based on inputs, current state, and the next state (again, looking ahead to what the next state will be)

15

7. Try to avoid causing ground bounce in your design

- Ground bounce can occur when many output pins simultaneously switch (SSO).
  - It is accentuated when many of the output pins are assigned fast slew rate.
- Ground bounce causes the functionality of the circuit to appear random.
  - This is caused by the device's internal ground rising above the board level ground.
  - When this happens some of the registers may trigger twice.
- Always use sufficient decoupling capacitors.

16

More on avoiding ground bounce

- Never assign fast slew rate to all of your output pins. Instead assign fast slew rate to those output paths that need a little bit better performance.
- Consider alternative coding techniques for your counters and state machines so that all of the bits do not transition at once.
- Add an extra ground pin to the design.
  - Tie an unused output pin near the SSO pins to ground, and externally connect this to the board's ground plane.
  - This pin should be using the LVTTL I/O standard with the highest drive capability.

17

8. Consider alternative techniques to create your counters

- Binary encoded counters can suffer from the same problems as binary encoded FSMs.
- Gray code and Johnson counters have only one bit transition at once, and are thus more reliable than binary encoded counters. Consider using these when possible or necessary.

18

Summary-- Design rules for reliability

- In general, when these design techniques are followed, your design has a greater chance of working properly. This also means that you can reduce your simulation effort and complete your design sooner. However, these steps sometimes cause the design to run a little slower.

19

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 our Design Rules for Reliability and then increase the speed of the design with these tips.

20

1. Don’t gate clocks

- 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.
- 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.

21

More on Clocks

- Take the time to understand the use and creation of Path Specific Timing Constraints. 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.

22

2. Use the Logic Level Timing Report to verify that your timing constraints are realistic

- This is important especially on the first implementation
- Double 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 very AWARE of this).
- Add 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.

23

3. 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 Report Paths in Timing Constraints Report, shows each constraints delay path in descending order of slack. The Report Paths Failing Timing Constraints Report, shows each failing delay path.
- The Custom Report, shows all the delay paths between groups of path endpoints created by selecting Sources and Destinations. This report can be used find the timing information for a particular delay path without having to review a large report.

24

More on the Timing Analyzer

- 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.
- 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.

25

4. 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.
- It increases the compile time considerably.

26

5. 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.

27

6. 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 techniques 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.

28

7. Choose the best State Machine Encoding scheme

- One-Hot and Gray encoded FSMs tend to have the least Next state logic and hence infer the fewest levels of logic.
- 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 fast and reliable FSM by transitioning the outputs with a 3.3Ns read access time.

29

8. Use the Carry Logic resources for fast arithmetic functions

- Carry Logic is the easiest and fastest way to create fast adders, accumulators, counters, subtractors, comparators, etc.
- 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.

30

9. Duplicate logic can decrease net delay on high fan-out nets

- 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 increases the total area of the design.

10. Use the IOB resources

- 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. 0Ns hold time comes at the expense of an increased set-up time.
- Fast slew rate decreases your output transition times (decrease output delays by 30-40%).

11. Use the Re-Entrant router

- This utility is designed to route designs that failed to completely route during the implementation process.
- It can 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.
- The Delay Based Clean-up option can be used to improve net delays as described above, but can be very effective at improving unconstrained net delays.

12. Change your 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. Moving to a faster device will cost more, but if may save you from costly re-design. Likewise, a slower speed grade can save you money.

Summary--Go faster!

- 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.
- Although these techniques enable you to be get better speed out of your design, creative use of the FPGAs resources can sometimes produce even better results.
- 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.