Technical Brief: Planning a High-Reliability FPGA Project

Kurt Aronow, PE

Designing logic for a high-reliability application field programmable gate array (FPGA) requires considerably more due diligence than a design for basic functionality in the lab. Thinking things through ahead of time and doing them in the right order can help considerably. (This discussion is not meant to be comprehensive.) Consider the following:

1. Specification or User's Manual: This normally includes an overall block diagram (and sometimes many lower-level block diagrams). Collectively, the block diagram should show the movement and control of data along with any other important functionality. The specification should also include table(s) of registers that an external processor can access. If the function of these registers needs to change in any way, the specification should be updated and agreed on before touching the code. The specification should also include detailed descriptions of what the FPGA is doing especially from the point of view of circuitry and processors connected to the FPGA. The User's Manual should be updated frequently so that it keeps up with the state of an actual design.

   A key initial tradeoff for the FPGA design is often about minimizing overall complexity versus having flexibility. For high-reliability designs that have significant calendar or budget constraints, simpler designs are highly desirable. If the specification is evolving, it's often better to start off with a relatively straight-forward design and then add complexity only as it's needed. Also, remember that relatively slow, complicated decisions are often best made in software running on a processor and not in FPGA logic.

2. Script Files for Running Design Tools: Synthesis, place & route, and simulation can generally all be run out of the graphical user interfaces (GUIs) for the software tools being used. However, for complex designs and especially for designs with more than one engineer, it is often worth the time to create script files (e.g., in TCL) for running the tool flows. These allow all of the engineers to run these functions in the same way each time and avoid confusion with compile options, etcetera. You will get the most out of these scripts if these are created for synthesis and place & route near the start of the design. (The scripts will likely get edited over time.)

3. Revision Control: If you have more than one engineer working on the design, it will probably be useful to set up a data repository with a system like Git that all of the engineers can access and check their files into. You'll need to agree on a few rules of use (unless an administrator does this for you). It is worth considering whether you want to put verification simulation files in a separate data repository than the source code and module level simulations. This will more easily allow the verification engineers to run their tests on different releases of the source code. It also removes the temptation for the verification engineers to modify the source code. Finally, revision control makes it much safer to remove unused/unwanted code since one can always go back to an earlier version and retrieve such code.

4. Timing Constraints and Diagrams: Engineers often wait until the end (and after the printed circuit board for the FPGA is built) to write timing constraints for their FPGA designs including for interfacing

to external memories, microcontrollers, etcetera. However, this can makes completing the design much more difficult because items contributing to the timing are often difficult to change. Instead, the timing constraints should be considered near the start of a design for especially certain external I/O. The gold standard for writing timing constraints for an FPGA to access external circuitry such as memory is to use a timing diagram in conjunction with a spreadsheet. One can use MS Office tools such as Excel and Visio for this although several specialized tools exist for this purpose.

The work process involves thinking through what constraints will be needed in the FPGA and (for example) the external memory and carefully showing these graphically in plots of time (and clock cycles) versus the logic levels for the various signals involved. Concurrently one enters all of these parameters in a spreadsheet and includes the propose timing constraints for the FPGA which will need to be met. (The purpose of the diagram is to help define what all of these parameters mean.) Then, preliminary logic can be written and tried out in the FPGA synthesis and place & route tools to see if the timing is reasonable. At this point, you may realize you need to change some of the FPGA pin assignments, I/O register assignments, oscillator speeds, etcetera. Even though you'll likely have to revisit this again in the design, this initial work on timing will likely shorten the overall design time.

5. Internal Buses and Clocked I/O: Although most FPGA design tools prevent the use of internal tri-state buses, deciding which modules will access which internal buses can be critical to creating simple, reliable designs. Creating a bus with multiple masters should be avoided unless it is absolutely necessary since all of the possible contention scenarios between the masters must be carefully considered. Also, clocking the I/O between various internal modules (and at the FPGA pins) can really aid in meeting timing constraints.

6. State Machines: Radiation and other factors can occasionally cause the state machines in FPGAs to wander into unknown states. For space-based deployment (and some other applications) these should be planned for in advance. Consider including pragma statements into your source code (or use equivalent techniques) to force your code synthesis tool to create your state machines explicitly in the way that you want. Make sure that all "unknown" states that can be decoded will converge back to a known safe state (e.g., an idle state). Consider adding external counters or other logic to make sure that your FPGA processes are completed in a reasonable number of clock cycles. These counters themselves may need to be triple-mode redundant (TMR) even if each flip-flop in the design has TMR. Consider TMR options in the compiler if your FPGA technology doesn't have this inherently. Consider adding in built-in self-test (BIST) options to allow an external processor (or a simulation) to force the FPGA into such states to see if the FPGA recovers as expected. These test features can be disabled (e.g., by an external test pin) before final deployment. Here is a useful PowerPoint presentation from NASA on this topic: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20150022459.pdf.

7. Module-Level Simulation: Module-level simulation (usually written by the design engineer) is almost always faster than debugging a design in hardware on a circuit board. These simulations are most useful when you can specifically test both the nominal and the corner cases for each piece of logic. You can also use these module simulations to check that your source code is conducive to allowing your simulation tool to checking for code coverage and for robust state machines. Checking for code coverage and robust state machines can be especially useful during system level verification.

8. System-Level Verification: In addition to module-level simulation, system-level verification simulation is highly-useful for revealing issues that might not show up quickly in board-level testing or that might be otherwise difficult to solve. Usually, verification is done by someone different than the original designer.

9. Closing Timing: This refers to making sure that the place & route tool for your FPGA indicates that your design meets all of its timing constraints (i.e., that you gave it). For high-reliability designs, this should include having the place & route tool check four-corners timing, i.e., timing with min and max delays at min and max temperatures. It is desirable that the tool be able to meet this timing fairly consistently--which depends on (among other things) on how full the FPGA is, how many constraints you have, how strict the constraints are, and how well you've constrained the design physically. As you do the design, you may find it useful to place internal area constraints on various modules so that as the design grows, the routing remains more consistent. (This can be especially useful if the modules use internal resources such as SRAMs.) Do not make the timing constraints more severe than they need to be. Try not to plan on filling up the FPGA too much. *Consider checking the ability of your design to meet four-corners timing all through the development of the FPGA code.*

10. Design Documentation: It is useful to decide up-front what level of documentation should be created for your design in addition to the User's Manual or specification. This will allow building in time and budget to create this documentation up-front and for filling in the documentation as you go along. The main purpose of this documentation is to reveal shortcomings in the design by using a methodical process. Your description and analysis document may include these sections:
    a. List the source files including constraints, scripts, source code, and simulation code. Include the location of each file and a description of each.
    b. Warnings: Disposition of all warnings from the synthesis and the place & route tools.
    c. Methodically check and account for: (1) correct synthesis of internal memories; (2) redundant & replicated logic; and (3) non-standard flip-flops (i.e., those without both a clock and a reset or preset input).
    d. Voltage & power analysis: These should cover all of the voltage levels used on the various pins, their compatibility with the rest of the circuitry, and the FPGA power consumption (and the ability of the PCB to deliver the necessary power).
    e. Provide a disposition of all FPGA pins including pin number, pin logic family , drive strength, edge rate, trace length (and latency), I/O buffers, and signal integrity on the board
    f. Provide an analysis of all clocks, clock domain crossings, resets, and interrupts. (Consider both description and diagrams).
    g. Tie together all timing constraints with respective timing diagrams and spreadsheets; all pins should have an explicit constraint even if the constraint is that it should be ignored.
    h. Provide a disposition for all state machines to assure the robustness of all of the state machines in the design.