Redgarden Engineering LLC
Electronics Engineering Consulting Services

Technical Brief: Planning a High-Reliability FPGA Project

Kurt Aronow, PE and Dave Williams

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. Consider the following. (This discussion is not meant to be comprehensive.)

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. Remember that relatively slow, complicated decisions are often best made in software running on a processor and not in FPGA logic.

   Avoid the temptation to create a lot of custom logic to debug a processor core—if this can be avoided by software more carefully staying within its memory map, etcetera. Alternatively, choose a processor or processor core that supports things like out flagging-of-bounds memory accesses.

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 and Defect Tracking: If you have more than one engineer working on the design, it will be useful to use revision control. 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 archive verification simulation files in a separate data repository from the FPGA source code targeted for synthesis. Doing so will more easily allow the verification engineers to run their tests on different releases of the FPGA source code to be verified. 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 revert back to an earlier version and retrieve code. When code is under revision control, we have found that there is less resistance to re-factor and make the needed source code changes.

Defect or 'bug' tracking is recommended on larger team-based FPGA designs. Consider utilizing Git systems that have integrated 'bug' or defect tracking. Examples are GitHub and GitLab. Having integrated defect tracking with Git is useful because it makes it harder for engineers to ignore some issues that may be buried in private emails and communications. GitHub invented the 'fork' where you download your own copy of the repo – and work can continue on a local copy but the user is not permitted to push or move data back to the main repo. This prohibits the user from pushing potentially buggy code onto the main branch without some code review.

Rules should be established on how defects are closed. Consider permitting select users with advanced privileges to be permitted to close a defect (like a team leader). This prohibits the user who is assigned a defect, from having the authority to close a defect because they do not like the person who posted the bug. We recommend giving revision control and defect tracking systems careful attention and to consider adopting some of the current best practices in industry.

4. Timing Constraints and Diagrams: Engineers often wait until the end of an FPGA design (and after the printed circuit board for the FPGA has been built) to write timing constraints for their FPGA designs. However, this can makes completing the design much more difficult because items contributing to the timing are often difficult to change (e.g. logic that interfaces to external memories and microcontrollers). After running static-timing analysis (STA), it may become apparent the FPGA design will need restructuring. However, since the FPGA design is considered functionally complete at this stage in the project, there can be major resistance to make changes to the FPGA. Instead, the timing constraints should be considered early in the design cycle, especially for certain external I/O as well as any clock-domain-crossings (CDC).

   The gold standard for writing timing constraints for an FPGA, especially when accessing external circuitry such as memory, is to use a timing diagram in conjunction with separate timing parameter computations. A spreadsheet can be used to make these computations and to link parameters to timing diagrams. One can use MS Office software 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 proposing timing constraints for the FPGA which will need to be met. (Part of the purpose of the diagram is to help define what all of these parameters mean.) Then, preliminary logic can be written and tested 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 these issues again in the design, the initial work on timing will likely shorten the overall design time. It is wrong to wait {until after the PCB is complete and the FPGA design has been functionally verified} to consider FPGA timing constraints.

5. The FPGA on the Circuit Board: FPGAs interface with PCB traces and cables. For many of these signals (e.g., the faster ones), signal quality needs to be considered. Board level simulation can be done before PCB design and fabrication. Signal quality can be modeled using simulation tools like Mentor Graphics' HyperLynx. Most IC vendors provide simulation models for their devices. IBIS is a popular simulation file format. For high-speed designs, the board stack-up will need to be designed to support desired trace impedances. We advocate considering PCB signal quality early in the design process. This will help in deciding if board-level terminations are required. (Adding dead-bug style resistors on many dense PCBs can be challenging.) Often, the resistor termination cannot be added at the desired end of a signal trance (e.g., at the FPGA), and a different type of termination will need to be considered. It is also important to consider how the board may be tested. If direct probing or in-circuit-test methods will be used, net access may not be possible. How do you place a probe on the net driver when it is buried under a ball grid array? In these situations, board-level

simulation becomes even more important. If you add an outer layer stub, it may ring. Adding many through-hole vias through power planes, to provide net access for probing, can degrade internal PCB power plane decoupling. Power plane decoupling and ultimately FPGA signal quality can suffer if the test strategy is to add through hole vias to every net under a BGA or chip-scale package.

6. 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 an internal bus with multiple bus masters should be given careful consideration. For data path driven designs, multiple bus masters may be required, so data can be transferred efficiently. But for more control-oriented designs, multiple bus masters may not be a good choice. Bus arbitration creates timing latency in control-oriented designs. Any design using multiple bus masters for internal FPGA designs will require testing of all of the possible contention scenarios between the masters. This increases the verification difficulty and if not thoroughly verified, can be an ongoing design risk.

   Registering FPGA external I/O can really aid in meeting timing constraints.

7. State Machines: Radiation and other factors can occasionally cause the state machines in FPGAs to transition into unknown or undesired states. For space-based deployment (and some other applications) these should be planned for in advance. Consider including pragma statements in 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.

8. 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.

9. System-Level Verification: In addition to module-level simulation, system-level verification and 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. It is generally advantageous to have system level verification done by someone other than the logic designer(s). On larger and high-reliability designs, consider budgeting for additional computing capability (both for long simulations and complex place and route). System level and chip-level simulations can be very computer intensive. When run times become many hours or days, consider building or leasing cloud-based servers to offload these tasks from less capable laptops and some desktop computers.

   If the FPGA has to fit hand-in-glove with embedded software, avoid the temptation to let the software team drive the FPGA and hardware development timing. The result can be that incomplete and untested FPGAs are delivered--which can create additional confusion and debug time for the software team. Rather, we advocate that the FPGA and hardware development team should have more authority to say when the design is ready for software test. It is best-practice to have the verification activity occurring in-step with the FPGA code

development. After a certain amount of basic verification is complete, then the design can be released to the software team. Our experience at Redgarden is this process can save overall project development time. Also, consider starting hardware and FPGA development before software development begins.

10. 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 preferably using timing diagrams where useful). 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.) Also, clocking the I/O between various internal modules is best-practice and can be a big aid in meeting timing constraints. Avoid making timing constraints much more severe than they need to be. Also timing problems can be aggravated by the device utilization. Having a 'full' FPGA can make timing closure very difficult with long place-and-route iterations. As a FPGA design is ongoing – periodic synthesis runs should be made to assess device utilization. *Consider checking the ability of your design to meet four-corners timing through-out the development of the FPGA code.*

11. 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. Link 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.

12. Design Reviews: Consider multiple, topic focused, design reviews for larger FPGA designs. The one BIG FINAL review is to be avoided. Consider a separate design review for FPGA timing and a separate design review for functional verification, for example. Give external reviewers multiple opportunities to review the design and documentation as it progresses. For space and some other critical applications, triple mode redundancy (TMR) and radiation risks should be reviewed in every design review.