Good FPGA Design Practices Aid FPGA Conversion to a ULC, and Improve the Quality of Both the FPGA and the ULC

Scope

This Application Note describes design practices that make a ULC conversion Schedule shorter, and accomplished with reduced risk. This note is recommended for a designer considering a conversion to a ULC, or for a designer before starting an FPGA design. For the designer considering a conversion to a ULC, this Application Note will give background on the reasons for the questions in the ULC Customer Questionnaire. For the designer just starting and FPGA design, this Application Note shows that much can be done during the FPGA design process to reduce the ULC conversion schedule and risk. This application note is probably not needed by an experienced ASIC designer, because the experienced ASIC designer is almost certainly following these design practices already.

This Application note is in three sections:
1. Overcoming Timing Difficulties - An Introduction to Good Design Practices
2. Good Design Practices.
3. Good Simulation Practices.

These “good design practices” also apply to good FPGA design, even if not converting to a ULC, and to good ASIC design.

Although following these practices is not mandatory, it is recommended. In all cases, for a ULC conversion, a feasibility study is done first to determine if a conversion should be successful. The feasibility study includes a determination of the degree of conformance to these practices, and an assessment of the degree of difficulty of a conversion. If these good design practices have been followed, it is virtually assured that the results of the feasibility study will be positive; if the practices have not been followed, there is a chance that the results of the feasibility study will be negative.

Note that the effective use of CAD/CAE tools depend on the designer following these design practices. If good design practices have been followed, the tools work much better, faster, with much less manual intervention, and report fewer errors and warnings that must be resolved by the ULC designer.

1. Overcoming Timing Difficulties - An Introduction to Good Design Practices

First a word about good FPGA design practices. Good FPGA design practices include allowing margins for timing variations. Timing variations occur as the chip operates over a temperature range, and due to fabrication process variations. When a designer uses trial and error in the system lab to design an FPGA, with no system-level specification for the FPGA and little knowledge of timing margins, the resulting FPGA could be undependable over temperature and process. FPGA designers should use good design practices, and simulation, to make sure that this does not happen. However, note that if the FPGA operates in production without problems for some time, this is proof that it is likely that adequate timing margins exist.

Now, on to conversion considerations. Converting the FPGA design to anything else (not just ULCs) means re-targeting the design to a new set of timing parameters, similar to those of the FPGA, but not exactly the same. A ULC is smaller and generally faster, like most ASICs, relative to an FPGA. The fastness of the ULC can be a

problem if the FPGA requires long delays on some paths. On the other hand, the ULC can be slower than Global Clock (fast-path) features on FPGAs, and this can be a problem if the signal travels across the chip. Also, ULC setup and hold are usually different; ULC setup is about zero, and hold is about 2-5ns, whereas FPGA setup and hold is generally the opposite. ULC designers know how to handle the conversions, but the conversion is much easier if the FPGA designer uses good design practices, and simulation.

If the timing tolerances are known, TEMIC assures that a successful FPGA conversion can be achieved, whether or not the FPGA has been simulated. This is assured by using ATPG (Automatic Test Program Generation) and associated fault simulation/grading. Before tapeout of the ULC, the ULC vectors, whether generated by the customers' simulations or by TEMIC ATPG, are tested on the FPGA in an IC tester, where timing is analyzed, and functionality is verified if tester or simulation vectors are provided by the customer.

Note that only customer-provided vectors will check for correct functional operation; the TEMIC ATPG vectors are used to check logic conformance to a customer specification and/or within tolerance of the FPGA timing, but these latter vectors do not check functionality (so the logic in the FPGA had better represent the desired functionality). Also note that only pin-to-pin vectors contribute to assuring the correctness of the conversion; simulation vectors that set internal nodes during simulation are welcome information but do not directly aid in checking the conversion. Also, note that adding internal scan to achieve a high fault coverage contributes nothing to the success of the conversion; scan "just" increases fault coverage for potential fabrication faults. As there are other ways to check for fabrication faults, scan is not recommended for conversions, but scan will be provided (at extra cost) if desired. Of course, JTAG is supported.

TEMIC has proven techniques that have been used to convert many hundreds of FPGAs, many done when the original FPGA designer was not available to answer questions. However, we always do a (free) feasibility study to make sure that the conversion should succeed, and the result of the feasibility study is generally positive.

2. Good Design Practices

TEMIC ULC designers use a "Feasibility Risk Assessment" checklist to record their analysis of the FPGA submitted for a feasibility study. This checklist covers the design practices described in this Application Note, and it makes sure that the ULC designer to assesses the potential problems in those areas where good design practices have not been followed and therefore there is increased risk that the conversion will take longer, or be unsuccessful, unless appropriate measures are taken. This sheet is shown in Figure 1. The following paragraphs are a description of these items, in the same order, and same numbering, as the table.

Figure 1.
Feasibility Risk Assessment (Rev. 1) **Mask & Company Name:_____**
Date:
Feasibility Study Designer(s):

☐ With Vectors ☐ Without Vectors

ANALYZE THE FOLLOWING (Assign a risk factor if the "good design practice" has not been used)	Risk Rating (0-10; 10 is bad)*	Estimated Extra Days if Need	COMMENTS:
1. Timing Well-specified (internal and external), i.e., a specification (with timing)			
2. Master Clear (Initialization); or small set of vectors to get to known state; a way to circumvent POR (if any)			
3. Only FFs drive reset (No combinatorial logic driving reset)			
4. Avoided Gated Clocks			
5. Avoided internally-generated clocks (used on-chip)			
6. Avoided Counters or control states over 10 bits without taps			
7. Glitches into a FF data-in before clk			
8. Avoided combinatorial loops (including no home-made FFs)			
9. Avoided internal tri-states			
10.Avoided redundant or fault tolerant circuits			
11.Avoided Global Clocks too fast to meet timing; & delay blocks or programmed delays; & deglitching circuits			
12.Avoided demand for high fault coverage, e.g., 95%			
13.Avoided special noise standards			
14.Avoided Asynchronous circuits			
15.No dynamic programming (fatal)			
TOTALS:			

1. Write a Specification on functionality and timing of the FPGA. This is for the reasons described above. Also, provide schematics and other items listed on the Customer questionnaire.

2. Use Master Clear, i.e., an asynchronous reset from an external pin. Second best is a small set of vectors to get to a known state. This means that there must be a way to circumvent any POR (Power On Reset), and rapidly get to a known state. Timing cannot be verified using simulation if the logic cannot be efficiently taken to a known state. The same is true for fault simulation/grading (used to generate tests that determine if the fabrication process is OK, and if enough logic has been tested to assure a successful conversion). (Don't use a bi-direct buffer on reset, or clock inputs, because it will make a mess of the simulation vectors when it goes to the Z state.)

3. Use Only FFs To Drive Reset (in addition to Master Clear). In other words, the reset to a FF needs to be synchronized and stabilized, and this is done by driving the reset from a data output of another FF, or else glitches can occur, causing errors. So, for example, the correct construction of a divide-by-3 counter would be as shown in Figure 2.

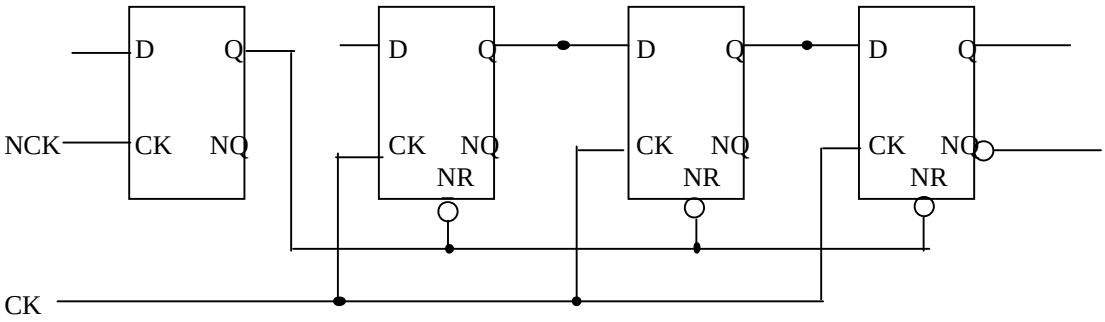

Figure 2. Correct Construction of a Divide-by-3 Counter

Also, if you need to generate a short pulse, instead of connecting a FF's Q to its reset, do as shown in figure 3.

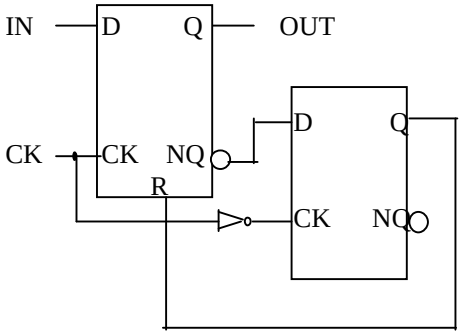

Figure 3. If Must Make Pulses, Do It This Way

4. Try to Minimize the Use of Gated Clocks, because they cause more skew between the clock and the data. In layout, the skew can be a real challenge to accommodate. If the skew cannot be accommodated, spikes occur, ruining the simulation and thus the ability to verify timing on a conversion. Of course, an enable on the clock that has been built into the cell library is OK, because this circuit is specially controlled by the CAE tools, including the layout tools, but designer-generated gated clocks cause errors and warnings by the CAE tools and, if there are enough of them spread around the logic, layout may not be able to get rid of the skew everywhere, and the conversion will have to be aborted (aborted toward the end of the design phase, which means that much design time will have been wasted).

5. Try to Minimize Internally Generated and Used Clocks (created using sequential or combinatorial logic), because the distribution of such clock lines inside the chip are subject to skew that is not accounted for by clock tree generation software and thus will likely have skew problems. This is similar to the problems in items 3 and 4. (So use of equations to generate clock signals in an FPGA design is not advised.)

6. Try to Keep Counters, Dividers, Control State Logic, Etc., Segments to Less than 10 Bits, as Seen from the Input/Output Pins, or Add Taps. If more than 10 bits, long counters should have taps for monitoring, or at least have a pre-load function. These taps should somehow easily propagate their data to output pins. The reason for this good practice is that the number of simulation vectors and test times required are too much without it.

7. Construct Logic to Minimize Glitches, and When a Glitch is Unavoidable, Make Sure That It Settles Out at the Data Input of a FF (Flip Flop) Before this FF is Clocked. Also, glitches into reset, set, or clock inputs to a flip flop must be avoided; as stated above, we recommend not using combinatorial logic in reset, set, or clock in the first place.

8. Avoid Combinatorial Loops (including combinatorial latches), and Pulse Generators. Any combinatorial design whose proper operation depends on delays through logic elements is dangerous. This is especially true in ULCs and ASICs but is also true for FPGAs. So don't design assuming that the delay down any combinatorial path is predictable, and that the difference in time down two different paths to the same destination is predictable. It isn't (unless very carefully controlled, as is done when creating the logic inside the cell of a cell library). Pulse widths and delays vary substantially over temperature, process variations, re-layouts, and voltage variations.

9. Internal Tri-States are Best Avoided, although we probably can handle them with some difficulty. If internal tri-states are unavoidable, avoid floating nodes. These can occur when all of the buffers driving a bus are disabled. Add a buffer to drive the bus when all of the other buffers are OFF.

10. Try to Avoid Redundant or Fault Tolerant Circuits, although we can handle them with some difficulty. The difficulty comes during re-synthesis to the ULC, where redundant logic is minimized out (and thus must be manually added back in). Please inform us if this type of circuitry is in your design.

11. Minimize the use of Global Clocks (fast paths), Delay Blocks, Programmed Delays, and Deglitching Circuits, because it is difficult (but generally not impossible) to match the timing. The Global Clock is only a problem when converting (to a ULC) if the signal travels across the chip. The delay blocks and programmed delays are problems because any function that depends on delay of the logic elements is difficult to reproduce (the min/max delay window in the converted material is too big), as explained in item 8 above. Deglitching circuits are also difficult to reproduce for the same reason - the min/max delay window in the converted material is too big.

12. Try Not to Insist on High Fault Coverage (e.g., 95%). Although it sounds good to demand the same fault coverage as on an ASIC, this adds to the time (and thus cost) of the conversion and does not provide the degree of benefit it does for an ASIC. First of all, ASICs are designed with design-for-test rules so that scan logic can be efficiently added, and its supporting CAE tools will work efficiently. FPGAs usually are not designed using these rules. The good design practices of this Application Note are nearly the same as the design-for-test rules for scan, so if these good design practices are followed completely, adding scan should not be a problem. However, if the practices are not rigorously followed, adding scan can take much time, and, in a few cases, may not be made to work properly. (Scan is a method where logic is added to each flip flop to allow all of the flip flops to be connected in a serial string in test mode so that test bits can be inserted and extracted in/out of the FFs between each normal clock pulse, effectively turning the flip flops into I/O pins for test, thus eliminating the very negative effect of the flip flops on generating an effective test using just real I/O pins.) We try to achieve 85% fault coverage without adding scan. Keep in mind that the merit of high fault coverage is that it provides a somewhat better check on the fabrication process - it contributes nothing to the accuracy of the conversion. We have other ways to check the fabrication process, including conservative process and packaging yield monitoring, and IDDQ testing can be added if the customer deems it necessary. Of course all of this will not be an issue if the fault coverage percentage gets to 95% without adding scan logic, as is often the case. On the other hand, at the other extreme, if the design has a big function but few pins in and/or out, e.g., a digital filter, we may not be able to achieve a great enough fault coverage percentage to be confident of the conversion, not to mention the monitoring of the processing. In other words, our primary concern is an accurate conversion, which depends on getting the fault coverage up using testing from the I/O pins only, and adding scan logic will not help this (at least at present-day levels of relatively low logic usage in FPGAs as converted). We have found that 85% test coverage is good; 60% is OK; and under 50% is definitely risky.

13. Let Us Know About Any Noise Requirements or Standards That Must Be Met, because we can use our slow Slew rate buffers to avoid noise problems (we have slew rate control).

14. Try to Use Synchronous Design as Much as Possible, because asynchronous design often depends on element delays, which, as we said in several places above, is difficult to duplicate in a conversion.

15. No Dynamic Reprogramming if you plan to do a conversion. We can handle on-board RAM, but not if used for dynamic re-programming. In other words, we can generally assure timing with one set of program parameters in an FPGA but it gets too complicated when these parameters are changed on the fly. So, in fact, we only hard-wire the programmed parts. (Note that this means that, for FPGAs that use the daisy chain method of programming, we can convert all of this set of FPGAs, or any FPGAs that are not the master, but we can't be the master when the slaves are not our conversions, i.e., not if the slaves are expecting to be programmed. That is, our converted slaves simply pass the programming data down the daisy chain. Our master cannot generate daisy chain programming information.)

3. Good Simulation Practices

Why Simulate?

An FPGA designer should simulate to:
1. Make sure that the FPGA timing margins are adequate.
2. Save time in the lab by designing on the computer, and using simulation to make sure that functionality is right and that all functions have been provided.
3. End up with an FPGA in good shape for conversion to a ULC.
4. Learn the same design techniques that will prepare the designer for ASIC design.

Don't forget to follow the Good Design Practices too. Also, remember to always go back to the design file and simulations on the computer as the "master" of your design.

Two Kinds of Simulation

As has been discussed somewhat above, we are interested in two different kinds of simulation - functional, and ATPG-based simulation with fault grading. Definitions of these terms is not consistent in the industry, so let's make it clear that "functional" simulation is generated by the designer and is used to verify the design, whereas, ATPG (Automatic Test Program (testlist) Generation) is software that is applied against a design file by a test engineer and it analyzes the circuit and generates test vectors for testers that exercise the logic such that any stuck-at-1 or stuck-at-zero faults will propagate to the output pins and thus be detected. So note that ATPG vectors have no idea about the function of the logic; they only try to exercise it to detect faults (whatever the functional use of the logic).

ATPG test was invented when functional test on testers started taking too long. It started with circuit boards, long before ASICs were invented. The ATPG-generated test exercises the combinational logic in just a few seconds on the tester. It is the sequential logic that causes the prohibitively long test times. A popular solution for this problem for ASICs, in conjunction with ATPG, is adding scan logic to effectively get rid of the sequential logic in the test generation. As explained in item 12 above, scan makes more sense for ASICs than for ULCs.

In the ULC & ASIC worlds, ATPG testing is mainly intended to provide good parts testing to determine that the fabrication process for the ULC or ASIC is working properly. It has nothing to do with verifying the design or proper functionality of the chip. So the less percentage of fault coverage, the less confident that the ATPG test will catch IC processing faults. Note however that ATPG testing is essentially a static test, and dynamic faults may not be detected by it. Also, stuck-at faults don't include bridging faults, etc. So we're not talking a perfect percentage when we achieve a certain percent of fault coverage. However, one of the big appeals of this method is that is does indicate some relative degree of test coverage in the form of a percentage, whether it's exactly right or not.

When generating ATPG tests, the test engineer often starts with a subset of the functional tests vectors, if these type of vectors are available from the designer. The designer selects a few hundred or thousand functional

vectors to "prime" the ATPG testing. These vectors usually result in about 60% fault coverage. Then the designer goes on to generating the ATPG vectors, adding to the fault coverage.

Take special note that the designer, in functional simulation, may often set and monitor nodes other than the inputs or output to the chip. These parts of the functional simulation are of no direct benefit to the ATPG testing. ATPG testing requires input/output vectors only. (But send all vectors for a conversion; all information is useful.)

When creating functional vectors for simulation, think in terms of a limited number of vectors to achieve functional verification - don't think in terms of real time, because real time takes too long. One second of real time of chip operation can be days of simulation time. One cannot afford to simulate POR (Power On Reset) or things like real diagnostic programs. Instead, one must devise simulation vectors that test the functionality in just a few hundred or thousand vectors. A 15K vector test on the tester should be the target (no extra charge); however well over 100K can be accommodated but may cost more (because of the cost of the tester time, which is hundreds of dollars per hour).

Simulation and Conversion to ULCs

In converting an FPGA to a ULC, the ATPG vectors provide another use besides increasing the quality via better testing for detecting processing faults, and this additional use is that these vectors are used to check the timing, especially I/O pin-to-pin timing, to make sure that the timing of the converted design into the ULC is within the necessary windows as compared to the timing of the FPGA being converted. In other words, the ULC designer takes any customer-provided functional test vectors whose "I/Os" are at the I/Os of the FPGA and uses them in the front of his ATPG program. The ULC designer then runs the ATPG software to add more vectors. The resulting test is analyzed for timing compatibility with the FPGA being converted. This is done before and after layout of the ULC. These tests are also applied to the FPGA itself in an IC tester, after layout but before tapeout, of the ULC. These test are also used in wafer probe test, prototype test (before sending the fabricated part to the customer for checkout), and production test. So these test vectors are targeted to several testers, using software made for that purpose. (Note that the FPGA designer-generated ATPG vectors would be welcome but an FPGA designer likely won't be producing them because FPGAs are built without benefit of fault grading. ASIC designs generally use fault grading.)

What You Can Do

So what can an FPGA designer do to make conversions easier, in light of what is needed for the conversion? One thing is that the FPGA designer can use simulations at the I/O pins as much as possible. It often works to simulate pieces of the design, and then combine the pieces and simulate the whole (from the I/O pins). Note however that each piece that is simulated requires the designer to set up a new simulation environment, which does take considerable time. So it pays to plan how this will be done and how much time it will take to do it. Doing all simulation from the I/O pins from the beginning is best, but only if the designer is confident that it will take less time to debug the design than if it were done in pieces first. Obviously, large design must be done in pieces. Also, the more experienced designers can probably do larger pieces than the less experienced.

Another thing that will help in conversion is vectors that do not try to stress the clock skew or variations due to processing, temperature, layout, etc. Depend on the environment options of the simulator for this (worst case temperature, etc., best case, normal, etc.). Use good design practices and conservative design rules so that you are not tweaking the logic to make it work. If good design rules are followed, it is fine if the data "just makes it" to the next clock pulse.

FPGA Vendor Simulation Tools

The FPGA vendors all provide simulation tools. They may charge extra for these tools. The books provided by the vendor teach how to do simulation. Most also have on-line help, and hotlines. Further information, and the latest information, is found on the home page of the vendor's World Wide Web site. They all have email as well, which is very convenient. In addition, most have regularly-scheduled training classes.

See the ULC Product book for the formats that are needed to be sent to TEMIC for your ULC conversion (simulation vectors and netlist).

Conclusion

Give simulation a try if you haven't already. It will result in an FPGA less likely to have problems in its application, and it will make conversion to a ULC much easier. Give the Good Design Practices a try too, and thus increase the quality of the parts, and conversion to a ULC will be a breeze!