Design Portability for FPGA/ASIC Conversion

Design portability is a measure of how easily a design can be ported from one technology, be it FPGA or ASIC, into another technology. The easier a design can be ported, the less bound the design is to a given vendor: it is in the interest of designers to make their designs waferfab independent.

Designers who have reduced their production costs by converting their FPGA designs into ASICs know that design portability allows independence from FPGA vendors. This allows designers to more easily cut costs.

Designers who have had to migrate their ASIC designs from the obsolescence of their ASIC vendors know that portability ensures component production longevity. This allows designers to continue selling their product.

When a design needs to be ported it’s common that knowledge of the design may have been lost, or that logic design rules may not have been adhered to, making a retarget both difficult and risky.

ATMEL regularly assists customers with designs originating from either FPGA or ASIC foundry, and recognizes the importance of portability. Portability is comprised of many issues, and this document presents common pitfalls in FPGA design and is meant to serve as a quick reference to ensure faster, smoother and safer FPGA->ASIC and ASIC->ASIC conversion.
1. Pinout

On a printed circuit board, once the partitioning of the different functions has been determined, there is the layout of the board to consider. Layout is subject to electrical rules and mechanical rules, and must also account for other devices on the board.

With ever smaller technologies, supply pins have to be positioned in such a way as to minimize noise and electromigration issues, for every component. Designers must be aware that assigning signals to pins consists not only of position considerations on the board, but also in the device itself.

For FPGA, these considerations may seem irrelevant or at least transparent, but when considering conversion into ASIC, the following rules must be adhered to:

- In Atmel's 0.35um processes, a maximum of 70mA is allowed between the two nearest VCC and VSS supplies. This value drops to 48mA in 0.18µm. The maximum drive current between supplies must be calculated, even for FPGA conversions, to be sure that a pin-to-pin replacement is possible.
- Drive current of each output buffer should not be assigned a greater value than necessary, since both the switching noise and power consumption increase rapidly as the drive strength increases.
- PLL or DLL blocks are noise sensitive; to ensure performance, it is best to have specific supplies for these blocks (see PLL/DLL section for more information).
- With input to output critical paths, placement of such IOs as close to each other as possible, to avoid these signals crossing more of the die than necessary, which subjects them to greater propagation delay and more noise.
- It's recommended to reserve two pins which will not be connected on the board, which may be used as test pins for Atmel's production test.

2. Specifications

Before writing many lines of RTL code, it is naturally important to get a clean specification, not only concerning behavior but also timing. The device typically shares the board with other devices, with which it communicates. FPGAs can be reprogrammed to verify functionality, whereas ASICs cannot. Preparing timing specifications (input data setup, critical paths, maximum frequency, etc.) will accelerate timing analysis and minimize risk.

3. Memory

As with Xilinx® or Altera® memories, ASIC memories come in many different flavors varieties the technologies. Atmel memories generated with our 0.35um Avanti™ memory compiler produce very different memories than those generated by our 0.18um Virage™ compiler.

Atmel typically adds glue logic, ”wrapper” logic, to a RAM block in order to make the block compatible with the RAM module of the original design. However, this logic is not always easy to implement, and it decreases performance. To avoid this, the easiest solution is to use memories in accordance with the following rules:

- synchronous mode for read and write operation
- same format between left and right ports for dual port memories
- no initialization
- no read-before-write in the same cycle
• no data used on output bus during the write operations.

Synchronous memory power consumption depends directly on clock frequency. The only way to reduce consumption is to gate the clock, such that the clock is active only when necessary.

4. Power Consumption

Technology and supply voltage are significant factors in power consumption, as is the design itself. As an example, consider a design implemented in a 0.18um FPGA for prototyping. It has been validated in the application, and performs satisfactorily. The consumption is measured and meets or exceeds expectations. The design is supplied at 1.8V for the core and 3.3V for the periphery. The objective is to enter production as soon as possible, using a ULC or ASIC to reduce the cost. The same pinout must also remain intact.

Depending on volume, Atmel may propose both a 0.18um solution and a 0.35um solution. Looking closer at the 0.35um technology offer, the designer requests a feasibility study to confirm the offer. To maintain pin-for-pin compatibility, Atmel’s implementation of the design will be supplied totally at 3.3V (core and periphery), as Atmel’s 0.35um technology does not operate at 1.8V. Using the netlist and the testbench, power consumption is analyzed to find that consumption exceeds requirements.

The designer considers Atmel’s 0.18um offer, which has better consumption results but for this designer’s production volume, doesn’t decrease the cost as much as the 0.35um solution does. By instead reviewing his design, the designer finds that by gating the clock such that it reached flops only when the flops’ input is relevant, some power is saved, and that by disabling memories when no access is necessary, still more power is saved, and he meets his budget.

5. Synchronous Design

Synchronous design facilitates static timing analysis. For a design with a single clock, STA results are obtainable by merely declaring this one clock frequency. With asynchronous designs, it’s much more complicated to verify timing, and asynchronicity should be avoided whenever possible, to minimize risk of incomplete analysis. With its ease of timing analysis, synchronous design is more portable than asynchronous design.

For synchronous designs with multiple clock domains, special care must be taken in the handling of signals which cross domains. Synchronization registers should always be employed.

Clock tree synthesis in an ASIC is very efficient even for clocks driving huge numbers of flip-flops, and the skew can be held very low (<0.2ns). The delay from input clock to flip-flop will depend of course on matrix size, which for a relatively large matrix may range up around 2ns.

When there are multiple clock trees in the design, the secondary clocks also enjoy minimal clock skew, but their latencies will of course vary based on the numbers of flops they drive. When a signal must cross from domain D1 to domain D2 with minimal skew between ck1 and ck2, then clock latencies l1 and l2 must be adjusted so as to be very similar; this is also accomplished by the clock tree synthesizer.

If the above clock tree synthesis is run assuming typical conditions, best and worst corners must also be considered. Under worst conditions, ck1 latency will increase as will ck2 latency, but not necessarily by the same amount. If ck1 latency increases by 0.5ns and ck2 increases by 0.3ns, timing may break.
One approach to resolve this problem is to employ a PLL/DLL, which minimizes the skew between the 2 clocks, but it may not be feasible to add a PLL/DLL due to pin assignments or pin count. Synchronizing a signal when it crosses clock domains will resolve this problem.

6. DLL/PLL

FPGAs offer many DLL/PLL/DCMs; such matrices employ multiple clock trees, each tree being local to some region of the matrix. In one region of the die, skew and latency might be perfect, but for logic spread across two regions, the same challenge arises as with multiple clock domains: skew.

Liberal use of DLL/PLL/DCMs makes sense for this situation, but it's not required when porting the design into an ASIC. In fact, it's quite rare to see many PLLs on an ASIC, as clock trees can be created efficiently across the full die. A PLL/DLL/DCM being used in the FPGA does not mean that the ASIC will require it; this relaxation of requirements reduces risk, effort, and time, and also frees up pins, possibly to reduce the die size or even for additional signals.

When a PLL or DLL really is required in a design, whether for frequency synthesis, clock latency reduction or phase control, the PLL/DLL in an ASIC must generally receive dedicated supplies in order to avoid noise and increased jitter. This point should be considered during the assignment of signals to pins.

7. Multipliers, Blocks

Multipliers and other functions are implemented in FPGAs as hard blocks, allowing superior performance and rapid implementation, as the design simply instantiates such blocks within the RTL code. Unfortunately, instantiation of such hard blocks is not portable, forcing such functionality to be coded in HDL. Performance, however, is rarely a concern: synthesis (ULCs employ Synopsys’ Designware as well as PKS) allows intelligent implementation of arithmetic functions, offering performance superior to conventional synthesis.

8. IP

Intellectual property is typically not portable, unless the IP is held by the designer in HDL form and the license between the designer and the IP vendor specifically allows porting of the RTL code and all its associated testbenches.

9. Reset

Reset can be implemented either with an external signal or an internal, “power-on-reset”. External reset is easier to implement, because not all technologies include power-on-reset capability. Global reset does not require much logic, and ensures correct initialization for production test.

10. Internal Tri-State Buffer

Internal tri-states can significantly increase power consumption, and fault-coverage suffers during production test. Muxes should be used instead.

11. Boundary-Scan Testing

Boundary-scan is offered in all modern FPGAs, and can be used both to program the FPGA and to test either the FPGA or (via daisy-chaining) other devices on the PC board. If boundary-scan
pins are used only for the purpose of programming the FPGA, conversion into ASIC allows these pins to either be removed or used for other (either application or test) purposes, since programming no longer occurs.

If boundary-scan pins are in fact used for boundary-scan testing, they will be implemented in the ASIC; no modification is required in the RTL code.

12. Test Considerations

When a design is converted from FPGA into ASIC, the ASIC production parts are fully tested in production. To ensure high fault coverage (and minimal vectors), internal scan chains are inserted by Atmel. Functional vectors, while critical for detecting timing issues in asynchronous logic (where STA falls short), generally fail to provide high fault coverage, and even when they do provide high coverage, they typically are far more numerous than scan vectors with the same coverage. No modification of source code is required.