

FPGAs Replacing ASICs in SoC Applications

Jay Southard, Principal Engineer Design Creation and Synthesis Division, Mentor Graphics jay_southard@mentor.com

FPGAs now deliver ASIC-like density and performance, while their flexibility and operational characteristics offer distinct advantages over their ASIC counterparts. As innovative architectures with embedded processors, memory blocks and DSP emerge, more designers are turning to FPGAs for new system-on-chip (SoC) designs. Switching from an ASIC to an FPGA design flow, however, is not easy. Key differences between the two silicon platforms mandate unique tool features and design methods when developing and implementing an FPGA-based SoC.

ASICs/FPGAs: Similarities and Differences

Complex FPGA design shares some commonality with ASIC design, but underthe-hood, many steps are fundamentally different. The pre-built nature of FPGAs drives a "use or lose" approach to features/capabilities. FPGA design, more often than ASIC design, must match functional requirements with the device architecture. Thus common steps such as synthesis or place and route (P&R) have all proved to differ subtly in the FPGA domain. But steps such as RTL simulation remain basically unchanged.

Some ASIC design tasks are simply not part of an FPGA flow. For example, each FPGA platform has already implemented the on-chip clock trees, boundary scan circuitry and PLLs. FPGA designers also need not perform buffer insertion or scan-chain insertion.

As high-end FPGAs encroach on ASIC performance, many advanced ASIC techniques are being adapted for FPGA design. Two such techniques: physical synthesis for high-performance timing closure, and C++ synthesis for system-level design, can illustrate the subtleties involved when the same general approach is used for both ASICs and FPGAs.

C++ synthesis techniques for ASICs and FPGAs are almost identical. C++ specifications are much less tied to any specific hardware than the corresponding RTL coding would be. To make full use of physical synthesis, however, requires an understanding of the FPGA's hardware structure. At the very least, the physical synthesis tools must be more specifically targeted to FPGA architectures.

Complex FPGAs must get Physical

Timing closure solutions using standalone logical synthesis and P&R are iterative and non-deterministic by nature. Without physical synthesis, designers typically write/re-write RTL code, provide guidance to the P&R tools by grouping cells, and possibly attempt some floorplanning. An alternative is to simply do numerous P&R runs. Usually, the HDL code/constraints are modified with only some heuristic notion that these changes will improve timing. Designers must iterate through P&R—the most time-consuming step in FPGA design—before gaining any visibility if the changes were a step in the right direction (or only served to exacerbate the problem). This unpredictability negates the reduced cost benefits and time-to-market advantages of using programmable logic in the first place.

To reduce design iterations and improve accuracy, interconnect delay and physical effects must be considered up front (**figure 1**); combined logic and physical synthesis is critical for efficient design of high-performance FPGAs. In today's chips, interconnect delay dominates performance for both ASIC and FPGA platforms. Yet there are major differences between the physical-implementation processes used for each design type. In ASICs, routing is not predetermined, so designers can do intelligent pre-placement floorplanning to reduce wire lengths and thus minimize delay.

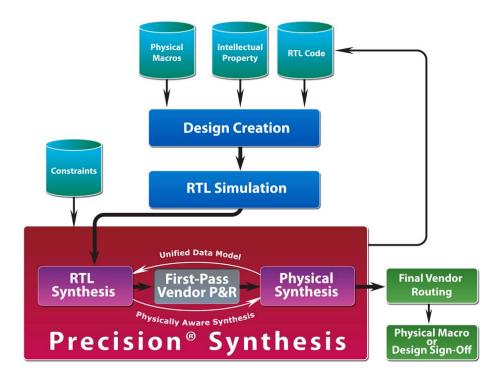


Figure 1 – High-end FGPA synthesis tools should consider FPGA vendor placement results up-front, and only then begin to manipulate the design using physical synthesis – integrated with logic synthesis in a unified data model – to converge on timing.

For FPGAs, the pre-built programmable routing fabric creates specific structural limitations. Fanout-based delay estimates in FPGAs do not model even a simplified version of physical reality – calling them timing "estimates" is misleading. Optimization decisions based on a wire-load estimate are mostly random choices. Even physical proximity is not always directly related to delay, so traditional floorplanning also falls painfully short in complex FPGA design.

Some "physical synthesis" alternatives available today are based solely on technology borrowed from the ASIC implementation space. In reality, forcing an ASIC methodology (and mentality) on the FPGA world cannot work. Such approaches essentially try to outsmart the vendor placement and may show promise in certain instances, but most cannot match the performance of a tool that leverages the FPGA vendor's post-P&R information to provide accurate physically-aware synthesis. Any tool used in complex FGPA design should consider vendor placement results as soon as possible, and only then begin to manipulate the design using physical synthesis--integrated with logical synthesis in a unified data model--to converge on timing.

C/C++ Raises the Level of Abstraction

For the first time, professional design engineers are literally struggling to keep pace with Moore's Law, which makes it difficult to fully utilize the capacity of 90 nm ASICs or efficiently target the complex structures found in domain-specific FPGAs. Algorithmic C synthesis (**figure 2**) promises to raise the abstraction of hardware design by providing a new, more abstract entry point, benefiting both ASIC and FPGA hardware designers. But to understand the need for higher abstraction languages, you must first analyze the problems with existing RTL methodologies.

The design complexity of new DSP applications has outpaced traditional RTL capabilities. To create hardware implementations for blocks of computationally intensive algorithms using RTL, design teams must iterate through several steps, including micro-architecture definition, handwritten RTL, and area/speed optimization through RTL synthesis. This manual process is slow and error-prone. In the final result, both the micro-architecture and technology characteristics become hard-coded into the RTL description. This hard coding renders the whole notion of RTL reuse or retargeting impractical in real applications.

An optimized C-to-RTL synthesis flow not only promotes a higher level of abstraction, it also gives the design team the flexibility to transition from one implementation technology to another. You can tune the hardware for high-performance parallel implementations or smaller, more serial implementations. Using this approach to describe functional intent (offered in Mentor Graphics' Catapult™ C Synthesis tool), you can move up to a far more productive abstraction level for designing hardware. As hardware designers, you can reduce

implementation efforts by as much as 20X while creating a more repeatable and reliable design flow.

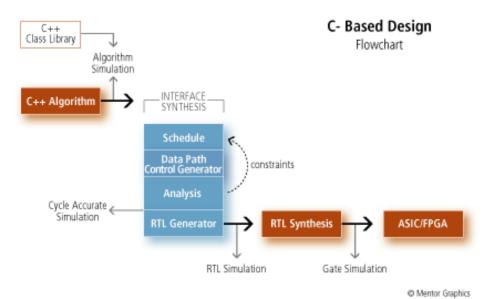


Figure 2 – An optimized C-to-RTL synthesis flow promotes a higher level of design abstraction and gives you the flexibility to easily transition from one implementation technology to another.

The ability to select fundamentally superior micro-architectural alternatives allows you to create designs of better quality than traditional RTL methods. Finally, this approach closes the conceptual gap between algorithm designers modeling in C/C++ and hardware designers working at the RTL abstraction level.

From Point Tools to ESL Design Flows

Every designer stands poised to benefit from the new standard set by new high-performance FPGAs. The next-generation challenge faced by mainstream FPGA EDA tool vendors is to leverage point tool expertise and thus meld apparently contradictory trends – higher levels of abstraction on the one hand and greater dependence on specific physical characteristics on the other – into a coherent design methodology and highly productive flow. In keeping with these advances, EDA tool companies will continue to extend and improve their comprehensive, integrated design flows spanning all levels of abstraction. Mentor Graphics continues to be a technology leader in this space. Designers must take advantage of EDA tools that now address both physical and electronic system-level (ESL) challenges of high-end FPGAs, and thus realize the unprecedented potential of these devices as ASIC replacements in new SoC designs.

To access the latest product news, application notes and case studies, evaluate new design flows, or schedule a product demonstration, visit www.mentor.com/fpga.