Nanometer Prototyping

Dr. Danny Rittman

January 2006

Introduction

As the semiconductor industry moves into new dimensions nanometer areas, keeping up with ASIC design requirements has become extremely challenging. Since the introduction of system-on-chip (SoC) design methodologies in the mid '90's, silicon technologies have been pushed to support higher performance, lower power and finer geometries. In order to deliver SoC designs on time, on budget and on market target, FPGA prototyping method was developed. FPGA prototyping enables an affordable SoC design hardware and software validation in order to ensure desired SoC functionality.

Designing an ASIC or a SoC today is an expensive project that involves complex infrastructure. The design has to meet its market window schedule in order to be profitable. The increasing cost of mask-sets and engineering efforts requires that getting a device right the first time is imperative. Simulation, verification, and validation of a design need to be performed to ensure that the ASIC or SoC design is correct before tape-out and after silicon has been received. Re-spins are practically disastrous from both an increased cost of engineering plus mask sets but also from a lost market opportunity perspective. Using FPGAs as an ASIC or SoC prototype vehicle provide a validation tool to address these development challenges and achieve pre-tape out results. Efficient ASIC prototyping requires extensive engineering efforts in design stages like hardware/software trade-off testing, design checking, debug, incremental compilation, pin planning, partitioning, security and tool automation. This article presents nanometer ASIC/SoC prototyping using FPGAs method. We'll observe SoC development, its challenges and the requirements for FPGA-based prototyping. We'll discuss potential solutions, their pros & cons and limitations. We will present one of the most effective methods to reduce the risk of ASIC/SoC design; structured ASICs.

The Evolution of platform-based SoC designs

Initially, ASICs were used to replace glue logic. These were assembled manually as a schematic at the transistor or gate level with almost no reuse of previously used logic or functions. With the rise of RTL languages such as Verilog and VHDL, EDA tools emerged to simulate and synthesize logic from an RTL description to a gate-level netlist. Reusability emerged in cell-based libraries and portions of reusable HDL code. The ability to reuse HDL functional code from one design to the next led to the beginnings of a block-based design methodology. Blocks could be described in RTL, synthesized into gates and laid out in a physical implementation as virtual components (VC) referred to as soft, firm or hard cores.

The advancement of process technology approaching 60 nm from 0.13 micron only a few years ago has opened up a significant number of new applications that can be integrated onto a single chip. Complexities of few millions gates are now moving to ten's million-plus gates with hundreds million gates in sight. It would be a challenge to simply maintain the design cycles of 8 to 16 months of a few years ago with this increased complexity. However, demand in consumer and communications products for new features and capabilities is driving market windows down; the upshot is that those 8- to 16-month design cycles are now approaching even shorter time with derivative products requiring shorter introduction times. Consumers are demanding more functionality in smaller packages at a lower price, which is yielding to the requirement for full systems to be integrated onto a single chip, known as system-on-chip, or SoC.

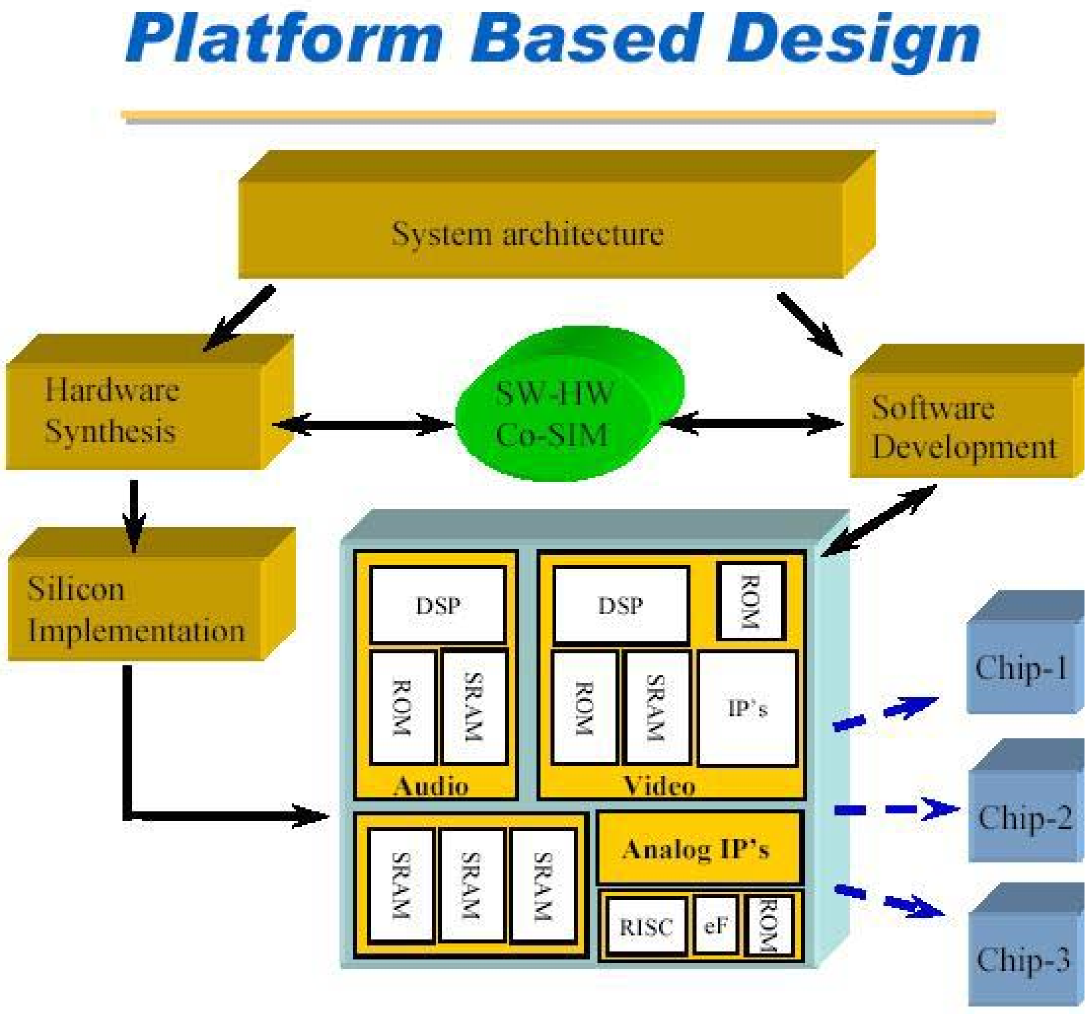

Image Source: National Semiconductors

ASIC vs SoC

What are the main differences between ASIC and SoC design? Typical SoC includes components like CPUs, DSPs, and memory that have traditionally been in separate chips. In SoC designs, external IP plays a much larger role. The designs are simply too big and complex to be developed from scratch within a single organization. Where custom logic is used, there is tremendous emphasis on re-use of existing design work, rather than re-inventing the wheel for each design. Another difference with SoCs is the emphasis on being able to roll out derivative products quickly. Companies want to capitalize on market successes by making small changes to existing designs to target new market niches. The biggest single difference between SoCs and traditional ASICs, however, is the importance of software. Software has become the key differentiator among products and often requires major development time.

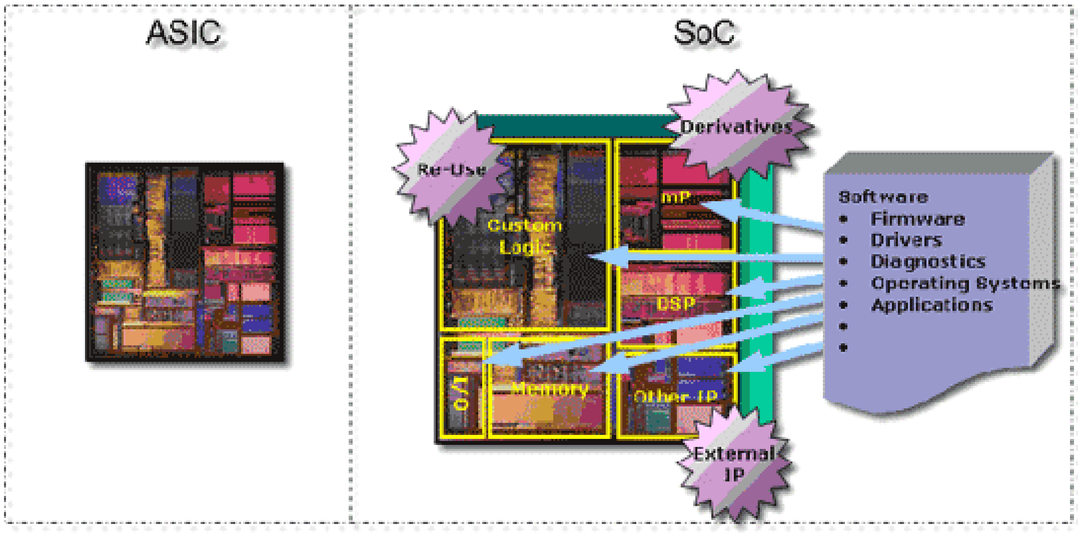

Figure 1: The difference between classic ASIC & SoC
Image Source: Aptix

FPGA Prototyping of ASICs/SoC - Advantages

One of the major issues of today's ASIC's and SoC is the design size. Validating or verifying large amount of devices within ASIC's or SoC has become a challenging task. Using FPGA's for prototyping can relieve the time bottleneck and remove the compute resources necessity required to functionally verify a medium-to-large sized design. In addition, prototyping can provide other, more enticing, benefits. A single prototype can serve to verify hardware, firmware and application software design functionality all before 1st article silicon is brought in-house. System performances are orders of magnitude faster than RTL simulation and the reliance on accurate IP models is removed since you're now running in actual hardware. Further more, multiple IP's can be integrated and functionally verified expeditiously. Hardware/software architectural trade-offs can be evaluated in hardware rather than at just a high-level of abstraction. So, if the architectural design initially put too much in hardware, then design blocks can be moved to software therefore lowering gate counts and lowering power consumption. If, on the other hand, the architect put too much in software, then design blocks can be moved into hardware thus increasing performance—a set of trade-offs you can make during hardware run-time with real world testing instead of just relying on high-level abstraction tools to make decisive trade-offs.

In some cases with off-the-shelf prototypes, pre-silicon prototypes or emulators, the hardware can be hooked up to a simulation environment linking system level verification tools allowing design co-verification with both simulation and hardware. This provides enormous advantage using FPGA's prototypes to validate ASIC's or SoC designs.

SoC Design & Prerequisites

Typical SoC design starts where the market need for a product is established. The next stage is to make a decision at the corporate level. A development process begins, and a hardware project and a software project are defined, varies from team to team. Each team of developers has its own special needs and concerns. Hardware developers are concerned about how they will integrate external IP that they don't fully understand. The IP can be hard or soft, with each type presenting different integration challenges. The designs are getting larger, and time-to-market pressures are getting greater. The need to develop derivatives must be accounted for in the original design—if not, developing a derivative can become as tough as a new product.

SoC development process can be segmented into five zones. See Figure 2.

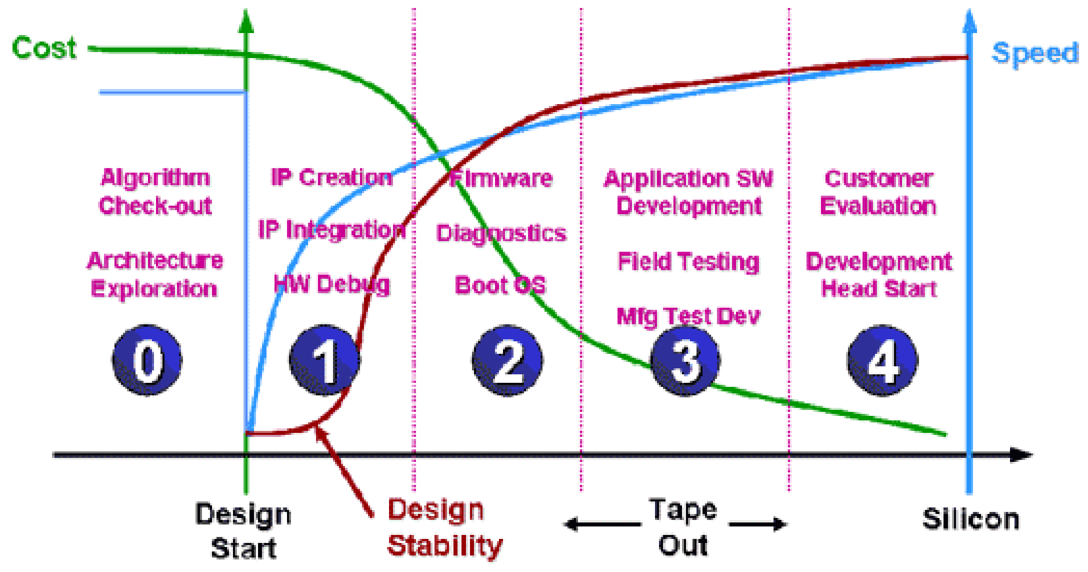

Figure 2: SoC Development & Prerequisites
Image Source: Aptix

Zone 1: IP Creation, Architecture exploration. Design stability is finalized.
Zone 2: Firmware and diagnostic development, as well as tasks like booting the OS.
Zone 3: Application software development and early field testing.
Zone 4: A prototype product is delivered to customers so that they can get a head start on their development and provide feedback to the design teams.

Using FPGA-based prototypes, this process can happen before silicon is available. Time-to-market factor is significantly shortened!

The development life cycle requires other prerequisites conditions. First, the cost has to go down as you move through the cycle. Supplying software developers with multi-hundred-thousand-dollar platforms would simply be cost-prohibitive. Second, speed has to go up as you move though the cycle. Applications software can't be developed effectively at sub-megahertz speeds. Software developers need near-real-time speed. Third, the stability of the design moves from low to high somewhere between Zone 1 and 2.

The requirements for an FPGA based prototype are the next key subjects.

- Re-configurable
It must be re-configurable to permit hardware developers to accommodate design iterations. For lower-cost units or replicates, the software teams will need to download new bit-streams from the hardware development team as new design iterations are created. In addition, the interconnect between the FPGAs and other system components that comprise the SoC design should be reprogrammable as well, allowing the FPGA vendor tools the freedom of deciding which pins are used during place-and-route The FPGA bit-streams should support optional encryption, in order to permit hardware designers to ship their prototypes to partners or customers for evaluation, while maintaining IP security.

- Flexible or "Open" system interface
It must maintain an 'openness' to it, allowing DSPs, µ-controllers, memories and other off-the-shelf components to be plugged straight into the prototype. It must be able to interface to standard peripherals such as RS232, parallel, USB, Ethernet, etc.

- **Fast**
It must run at real-time or near-real-time speeds. Booting an OS at multi-MHz is required for software developer productivity. Sub-MHz speeds won't get the job done. The faster, the better.

- Debug environment
It must provide debugging capabilities, whether visibility is built-in to the

FPGA prototype or it is interfaced to readily available debug tools. It must co-exist with a simulation environment allowing non-synthesizable blocks to remain in simulation until the entire implementation is ready for in-circuit testing. A block-by-block verification strategy can reduce required IP integration effort required to get the design running in an in-circuit environment.

- Self-test
It must also be reliable and have built-in self-test capabilities. If a design doesn’t work in hardware, the designer does not want to spend time debugging hardware-related issues instead of focusing on design debugging issues.

SoC Design Mapping & Debugging

The next step is to map the SoC design into FPGA. FPGA vendors and synthesis providers create products to help designers map their designs into an FPGA. They even provide debug instrumentation and run-time hooks. However, mapping and debugging a design into multiple FPGAs from SoC design coding styles requires additional capabilities in order to harness the benefits and features provided by the FPGA vendors and synthesis providers.

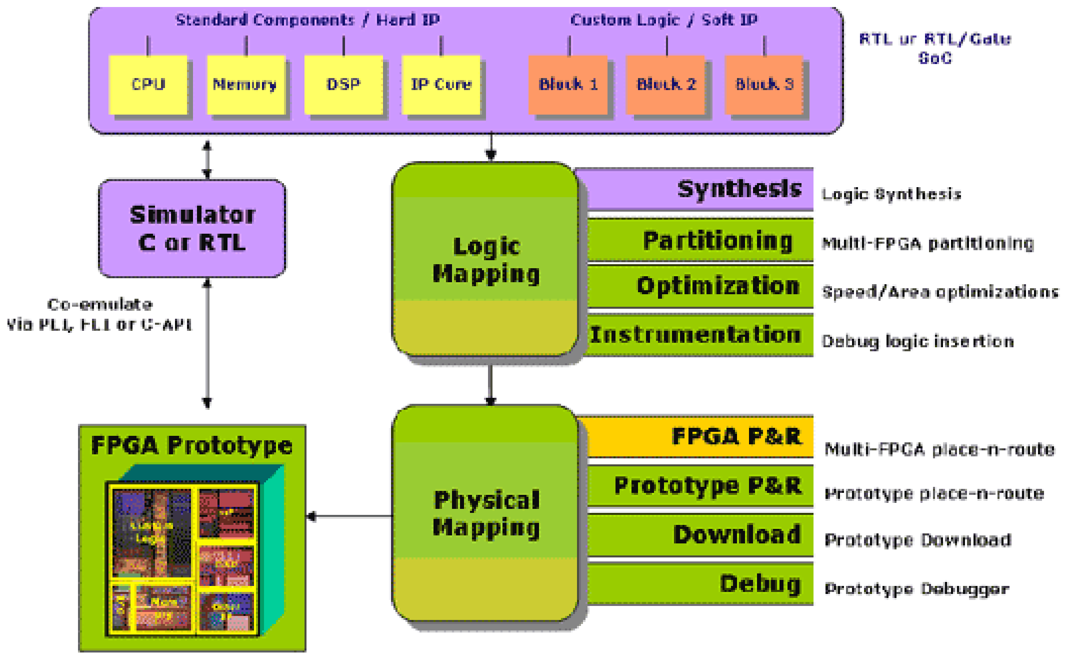

Figure 3: Mapping flow of a SoC design to a multi-FPGA prototype
Image Source: Aptix

to undergo logic mapping, which consists of logic synthesis, logic partitioning, logic optimization and logic debug instrumentation. The design is then physically mapped, where the partitioned FPGAs' netlists are placed-and-routed, and the prototype is placed-and-routed if it contains any (re)programmable circuits other than FPGAs. The design is then downloaded into the FPGA prototype and debugged using the prototype debugger. Mapping SoC design into FPGA prototype is going through some challenges.

1. Design for Prototyping (DFP)

SoC designers code their designs for a sea-of-gates target environment. In general, a sea-of-gates design *may* not easily compile into or function in the defined architectures of FPGAs. To address this challenge, the *offending* coding techniques must be detected and fixed—automation is preferred here. For instance, gated clocks *may* need to be converted into clock enables, and, since there are several different gated clock types, caution in detecting and fixing them is a must. As another example, simulation-friendly coding techniques are sometimes not synthesis-friendly. Or, certain coding techniques can result in functional differences between pre- and post-synthesis simulations. The sooner these coding issues are flagged to the user in the logic mapping flow, the sooner designers can analyze the problems without having to spend days debugging the problem in the hardware prototype. Both Mentor's HDL Designer and DesignAnalyst tool provide RMM-compliant (Reuse Methodology Manual) linters that check the code for FPGA-based coding violations. The tools will provide flags for the user allowing the user to edit their source code using, for instance, conditional directives such as *ifdef* when compiling the design to FPGAs. When the design has to be compiled to the SoC, the conditional directive setting will switch to the SoC implementation. Granted, this implies that not all of the SoC targeted logic will be functionally verified using FPGAs, but this should only be a sub-set of the entire design. Alternatively, the user can modify their code to enable their design to be compiled into the FPGA and pass the linter's coding checker.

2. Logic Synthesis

A SoC ASIC design usually does not fit into a single FPGA. Synthesis tools are developed and qualified to handle design sizes that fit into the largest FPGA available in the market when the tool was released. In fact, designs that span two or more FPGAs quickly start to bump the limits of synthesis functionality. In these cases, you're faced with CPU power and memory restrictions on designs that span two or more high-end FPGAs.

During the synthesis process, a strategy to retain hierarchy as well as overcome synthesis restrictions using a bottom-up or middle-down synthesis is often necessary to synthesize the large design.

3. Multi-FPGA Partitioning

Some SoC designs may simply fit into a single largest density FPGA device. 90-nm technology has increased gate capacity of FPGAs up to 2.2 million ASIC gate equivalents with additional dedicated resources such as memory and DSP functionality. In the case where the design can fit into two FPGA devices, the hardware engineer may hand-partition the design ensuring, for instance, that performance critical blocks are placed in the same FPGA. However, if the need to partition the design grows past three or more FPGA devices, the number of combinations of blocks partitioning quickly increases to an extent where graphical tools and heuristic algorithms are needed to help automate the multi- FPGA partitioning process. For instance, consider partitioning a design with 15 major blocks into three identical FPGAs, of which any combination of 5 blocks can fit into an FPGA. Using combination mathematics, there are 756,756 unique combinations of partitioning the design! The number additionally increases as the number of FPGAs increase. This brief analysis is based on gate consideration but there's another complexity—namely block I/Os. When moving blocks from one FPGA to another, the shift in I/O counts between the FPGAs is another factor to consider. Needless to say, both the gate and I/O considerations can make multi-FPGA partitioning a daunting task. Despite the magnitude and complexities of these numbers, there are solutions. Mentor's HDL Designer provides a graphical group/ungroup capability providing I/O impact analysis which assists the hardware designer during the partitioning process.

4. Incremental Compilation

Incremental approach is a must in order to reduce the time required to re-implement changes into the prototype. Frequently, developers focus their attention on a single mapping and getting their designs into a FPGA prototype. Yet, once design bugs are detected in hardware, a new revision of the design needs to be mapped. But, it's not the entire netlist that was rev'ed – more often only a small subset of the design needs to be re-mapped. So, why redo the mapping portions for design blocks that do not need to be re-mapped since the RTL didn't change? The SoC-to-Prototype mapping flow must be incremental.

EDA Solutions

The EDA world has provided efficient prototyping solutions for the past decade. Typically, ASIC and SoC hardware designers have an affinity for scripting. Their flows use and apply automation within a single tool flow and between multiple tools used in projects. Initially, as a SoC design is compiled into FPGA(s), designers may require of use of graphical tools to assist them. However, as design iterations increase, the need to automate becomes dominant. With today's EDA tools, developing and running tool command language (Tcl) scripts to control the compilation allows the designer to perform a wide range of functions, such as compiling a design or writing procedures to automate common tasks.

Structured ASIC's – Furthermore Risk Reduction

No doubt, verifying a SoC design using FPGAs for prototyping is a significant risk reducer. The risk can be further reduced by migrating the FPGA-verified design into structured ASICs. Structured ASICs can shrink time-to-market further as the additional design effort required for migration is minimal compared to re-targeting the design to an ASIC or SoC technology. Prototyping saves time in more ways than one. A structured ASIC is a non-reprogrammable device seamlessly migrated from a design that is prototyped in a FPGA. Based on a fine-grained architecture of transistor cells, structured ASICs are fabricated on the same advanced nanometer process from the Semiconductor Manufacturing Company as the FPGA family.

Conclusions

System-on-a-chip (SoC) ASIC technology is one of the most effective ways to produce high-speed, low-power products. The continuing advances in process technology give us the ability, in principle, to design ever-more-complex systems-on-chip at higher speeds. Hence those complexities, combined with the more complex device and interconnect models that these processes require, create a design crisis in which designers spend more and more time on iterating through cycles of synthesis, place and route, physical design and verification. System-on-chip solutions typically include high-speed, high-bandwidth mixed-signal interfaces; large, complex digital blocks that implement multilayer protocols; and significant amounts of on-chip memory. Designers of those devices push the limits of our EDA tools. ASIC and SoC prototyping using nanometer FPGAs has several benefits and fewer drawbacks to get the design out-the-door on time and on budget. Nanometer technology has enabled more ASIC and SoC designs to be prototyped in a single FPGA although multiple FPGAs prototyping is still predominant. Some of the more important associated requirements for logic mapping and debugging tool flow from RTL-to-gates have been discussed in this paper. Silicon and tool solutions are available to the ASIC and SoC developers providing automation, flexibility and visibility to reduce development risk. The risk can be further reduced by migrating prototyped FPGAs to structured ASICs.

References

M.R. Stan, Wayne P. Burleson. Bus-Invert Coding for Low Power I/O. IEEE Transactions on Very Large Scale Integration Systems, 1995.

A. Malik, B. Moyer, D. Cermak. A Programmable Unified Cache Architecture for Embedded Applications. International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, 2000.

D.H. Albonesi. Selective Cache Ways: On-Demand Cache Resource Allocation. MICRO, 1999.

S.M. Kang. Accurate Simulation of Power Dissipation in VLSI Circuits. IEEE Journal of Solid-State Circuits, vol. CS21, no. 5, 1986.

G.Y Yacoub, W.H. Ku. An Accurate Simulation Technique for Short- Circuit Power Dissipation Based on Current Component Isolation. International Symposium on Circuits and Systems, 1989.

R. Tjarnstorm. Power Dissipation Estimate by Switch Level Simulation. International Symposium on Circuits and Systems, 1989.

T.H. Krodel. PowerPlay - Fast Dynamic Power Evaluation Based on Logic Simulation. International Conference on Computer Aided Design, 1991.

E. Macii, M. Pedram. High-Level Power Modeling, Evaluation, and Optimization. IEEE Transactions on Computer Aided Design, vol. 17, no. 11, 1998.

D. Marculescu, R. Marculescu, M. Pedram. Information Theoretic Measures for Power Analysis. IEEE Transactions on Computer Aided Design, vol. 15, no. 6, 1996.

M. Nemani, F. Najm. Toward a High Level Power Evaluation Capability. IEEE Transactions on Computer Aided Design, vol. 15, no. 6, 1996.

V. Tiwari, S. Malik, A. Wolfe. Power Analysis of Embedded Software: A First Step Toward Sofware Power Minimization. IEEE Transactions on Very Large Scale Integration Systems, vol. 2, no. 4, 1994.

C.T. Hsieh, M. Pedram, H. Mehta, F. Rastgar. Profile Driven Program Synthesis for Evaluation of System Power Dissipation. Design Automation Conference, 1997.

C. Barndolese, W. Fornaciari, F. Salice, D. Sciuto. Energy Evaluation for 32-bit Microprocessor. International Workshop on Hardware/Software Co-Design, 2000.

R.J. Evans, P.D. Franzon. Energy Consumption Modeling and Optimization for SRAMs, IEEE Journal of Solid-State Circuits, vol. 30,
no. 5, 1995.

T. Givargis and F. Vahid. Interface Exploration for Reduced Power in Core-Based Systems, International Symposium on System Synthesis, 1998.

High Density Altera FPGAs - http://www.altera.com

B. Ackland et al., "A Single Chip, 1.6-Billion, 16-b MAC/s Multiprocessor DSP," *IEEE J. Solid-State Circuits*, Mar. 2000, pp. 412-424.
A. Agrawal, "Raw Computation," *Scientific Am.*, Aug. 1999, pp. 60-63.
January 2002

L. Benini and G. De Micheli, "System-Level Power Optimization: Techniques and Tools," *ACM Trans. Design Automation of Electronic Systems*, Apr. 2000, pp. 115-192.

R. Hegde and N. Shanbhag, "Toward Achieving Energy Efficiency in Presence of Deep Submicron Noise," *IEEE Trans. VLSI Systems*, Aug. 2000, pp. 379-391.

W. Dally and J. Poulton, *Digital Systems Engineering*, Cambridge Univ. Press, New York, 1998.

J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach,* IEEE CS Press, Los Alamitos, Calif., 1997.

P. Guerrier and A. Grenier, "A Generic Architecture for On-Chip Packet-Switched Interconnections," *Proc. IEEE Design Automation and Test in Europe* (DATE 2000), IEEE Press, Piscataway, N.J., 2000, pp. 250-256.

R. Ho, K. Mai, and M. Horowitz, "The Future of Wires," *Proc. IEEE*, Apr. 2001, pp. 490-504.

D. Sylvester and K. Keutzer, "A Global Wiring Paradigm for Deep Submicron Design," *IEEE Trans. CAD/ICAS*, Feb. 2000, pp. 242-252.

Todd Moore and Rick Schmalbach Sep 1, 2003 - RFDesign