INVITED PAPER

System-on-Chip: Reuse and Integration

Pre-designed and pre-verified hardware and software blocks can be combined on chips for many different applications—they promise large productivity gains.

By Resve Saleh, *Fellow IEEE*, Steve Wilton, *Senior Member IEEE*, Shahriar Mirabbasi, *Member IEEE*, Alan Hu, *Member IEEE*, Mark Greenstreet, *Member IEEE*, Guy Lemieux, *Member IEEE*, Partha Pratim Pande, *Member IEEE*, Cristian Grecu, *Student Member IEEE*, and Andre Ivanov, *Fellow IEEE*

ABSTRACT | Over the past ten years, as integrated circuits became increasingly more complex and expensive, the industry began to embrace new design and reuse methodologies that are collectively referred to as system-on-chip (SoC) design. In this paper, we focus on the reuse and integration issues encountered in this paradigm shift. The reusable components, called intellectual property (IP) blocks or cores, are typically synthesizable register-transfer level (RTL) designs (often called soft cores) or layout level designs (often called hard cores). The concept of reuse can be carried out at the block, platform, or chip levels, and involves making the IP sufficiently general, configurable, or programmable, for use in a wide range of applications. The IP integration issues include connecting the computational units to the communication medium, which is moving from ad hoc bus-based approaches toward structured network-on-chip (NoC) architectures. Design-for-test methodologies are also described, along with verification issues that must be addressed when integrating reusable components.

KEYWORDS | Analog intellectual property (IP); intellectual property (IP) cores; network-on-chip (NoC); platform-based design; programmable intellectual property (IP); system-on-chip testing; system-on-chip verification; system-on-chip (SoC)

Manuscript received September 1, 2005; revised February 9, 2006. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Individual Discovery Grants and Strategic Grant STPGP 257684, in part by PMC-Sierra, in part by Micronet, in part by Gennum, in part by Altera, in part by the Canadian Foundation for Innovation, in part by CityTel of Prince Rupert and in part by the Advanced Systems Institute of British Columbia.

R. Saleh, **S. Wilton**, **S. Mirabbasi**, **G. Lemieux**, **C. Grecu**, and **A. Ivanov** are with the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC V6T 1Z4, Canada (e-mail: res@ece.ubc.ca).

A. Hu and **M. Greenstreet** are with the Department of Computer Science, University of British Columbia, Vancouver, BC V6T 1Z4, Canada.

P. P. Pande is with the School of Electrical Engineering and Computer Science, Washington State University, PO Box 642752, Pullman, WA 99164-2752 USA.

Digital Object Identifier: 10.1109/JPROC.2006.873611

I. INTRODUCTION

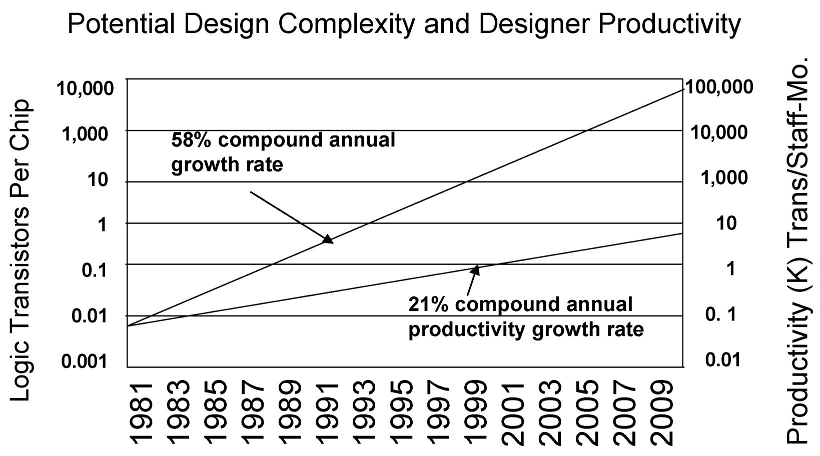
The semiconductor industry has continued to make impressive improvements in the achievable density of very large-scale integrated (VLSI) circuits [1]. In order to keep pace with the levels of integration available, design engineers have developed new methodologies and techniques to manage the increased complexity inherent in these large chips. One such emerging methodology is system-on-chip (SoC) design, wherein predesigned and preverified blocks—often called intellectual property (IP) blocks, IP cores, or virtual components—are obtained from internal sources, or third parties, and combined on a single chip. These reusable IP cores [2] may include embedded processors, memory blocks, interface blocks, analog blocks, and components that handle application specific processing functions. Corresponding software components are also provided in a reusable form and may include real-time operating systems and kernels, library functions, and device drivers.

Large productivity gains can be achieved using this SoC/IP approach. In fact, rather than implementing each of these components separately, the role of the SoC designer is to integrate them onto a chip to implement complex functions in a relatively short amount of time. The integration process involves connecting the IP blocks to the communication network, implementing design-for-test (DFT) techniques [3] and using methodologies to verify and validate the overall system-level design. Even larger productivity gains are possible if the system is architected as a platform [4] in such as way that derivative designs can be generated quickly. The purpose of this paper is to address the reuse and integration issues in SoC design today.

In the past, the concept of SoC simply implied higher and higher levels of integration. That is, it was viewed as

0018-9219/$20.00 ©2006 IEEE

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.



Year	Technology	Chip Complexity	ASIC Frequency
1997	250 nm	50M Tr.	100MHz
1999	180 nm	150M Tr.	200MHz
2002	130 nm	250M Tr.	400MHz
2004	90 nm	500M Tr.	600MHz

Fig. 1. ***Productivity gap [1].***

migrating a multichip system-on-board (SoB) to a single chip containing digital logic, memory, analog/mixed-signal, and RF blocks. The primary drivers for this direction were the reduction of power, smaller form factor, and lower overall cost. It is important to recognize that integrating more and more functionality on a chip has always existed as a trend by virtue of Moore's Law, which predicts that the number of transistors on a chip will double every 18–24 months [5]. The challenge is to increase designer productivity to keep pace with Moore's Law. Therefore, today's notion of SoC is defined in terms of overall productivity gains through reusable design and integration of components. Fig. 1 illustrates the point by superimposing the compound annual growth rate of transistors on a chip (58%) with the productivity growth of the designer in terms of transistors per month (21%) [1]. The increasing separation between the two lines is referred to as the *productivity gap*. In the associated table, we provide estimates of the chip complexity (transistor counts) and performance (frequency of operation) for four technology nodes, assuming an application-specific integrated circuit (ASIC) design. The implementation of a chip at the 250-nm node with 50 million transistors provided major challenges; it is significantly more complex at the 90-nm node, where designs may contain up to 500 million transistors.

In industry, the ultimate driver for any paradigm shift is the overall cost of a design. Today, the cost of a 50 million transistor chip can run between US$20–30 million [1], consisting of the cost of engineering resources and the rising tooling costs in the form of masks required for manufacturing. Currently mask costs account for 10% of the overall cost, but this can become much more significant if there are multiple design "spins" that require new mask sets. Other important factors driving the trend toward SoC design are requirements for lower power and a smaller form factor. These factors are a natural by-product of the higher levels of integration, so we will not discuss them further in this paper. However, it is well worth noting that on-chip power reduction is currently an active area of research [6] and will continue to be so in the foreseeable future.

With the increasing adoption of SoC design over the years, it has now become the driver for many other improvements in the integrated circuit (IC) industry. In fact, SoC is now a driver for the development and use of industry-wide standards. For example, standards for buses [7], bus interfaces [8], IP exchange formats, documentation, IP protection and tracking [9], and test wrappers [10], [11] have been developed and standardized. It has also forced suppliers to improve the quality of reusable IP. For many years, the quality of IP sold in the marketplace, and the business and legal issues surrounding licensing, have limited the growth of the third-party IP industry. Today, there are improvements in the licensing process and new targets for the overall IP quality levels to qualify as "Star-IP" [2]. Another side effect of SoC is that it has enabled designers and developers to migrate up to the system-level to address hardware/software codesign issues, which have been dealt with separately up to this

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.

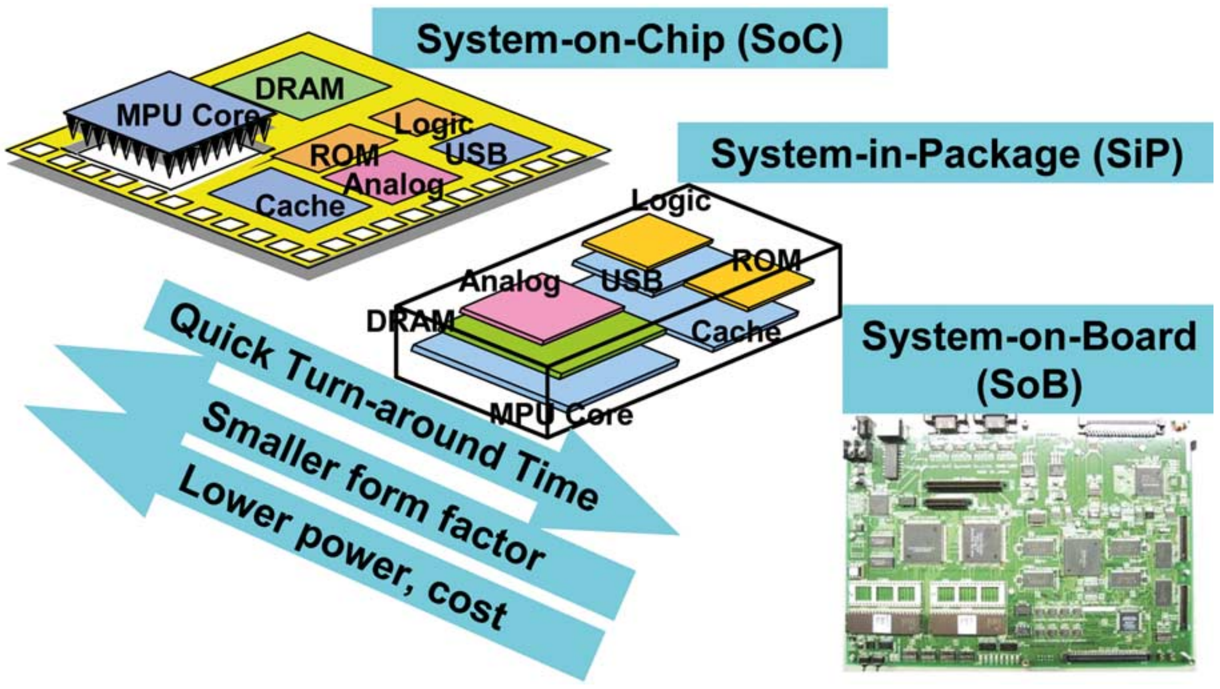


Fig. 2. *SoC versus SiP versus SoB [13].*

point. It is now possible to treat these issues in a holistic way, since components of the SoC can be viewed as commodity items.

An often-viewed competitor to SoC is the notion of system-in-package (SiP) [12] whereby separate chips are packaged together into a system with a very small form factor using a common two-dimensional (2-D) or three-dimensional (3-D) substrate. It is most suitable for the integration of heterogeneous technologies where single-chip integration is difficult or too expensive to pursue. In reality, SoC, SiP, and SoB are not in direct competition with one another, but rather provide different tradeoffs in terms of cost (including differences in both development costs and production or unit costs), power, testability, time-to-market, and packaging. The key issue is to decide which option provides the best solution for a given product. Moreover, an SoC can be part of an SiP or SoB solution if appropriate from cost, system performance, and time-to-market perspectives. Fig. 2 illustrates first-order tradeoffs of the three options in terms of turnaround time, form factor, and power.

The rest of this paper is organized as follows. The issues surrounding reusability and designer productivity are discussed at the IP block level in Section II and at the chip level in Section III. Challenges of integrating numerous IP blocks on a single chip give rise to communication infrastructure for IP design presented in Section IV. Of course, SoC designs are very large, complex designs that require extensive testing and verification. These topics are treated in Sections V and VI, respectively. Finally, an overall summary is provided in Section VII.

II. REUSABLE IP

The main prerequisite for creating SoC designs is a set of reusable IP blocks that support plug-and-play integration. IP blocks are the highest level building blocks of an SoC. A library of reusable IP blocks with various timing, area, power configurations is the key to SoC success as it allows mix-and-match of different IP blocks so that the SoC integrator can apply the tradeoffs that best suit the needs of the target application. The process of creating a reusable IP block, however, differs from the traditional ASIC design approach. Typically, it may require five times as much work to generate a reusable IP block compared to a single-use block [2]. Details on the IP design process may be found in a wide variety of sources. However, the most noteworthy reference on the subject is the *Reuse Methodology Manual* (RMM) [2], which provides a comprehensive set of guidelines on the reusable IP design process. In the sections to follow, we provide an overview of the issues in design issues for reusable digital IP, analog IP, and programmable IP.

A. Digital IP

Digital IP blocks are the most popular and ubiquitous form of reusable IP in industry today. Many of the tools and design methodologies for creating digital IP were

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.

already in place when the concepts of reusability emerged. However, there have been wholesale changes in the design flow to fully enable reusable design. In addition, there are many technical issues that need to be addressed, as the IP developer must anticipate all of the applications in which the IP block may be used. The technical issues associated with the design will not be repeated here, since they are well documented in the RMM [2].

The three stages in the design process are as follows:

- specification and documentation of the reusable IP;
- implementation using standardized coding practices;
- full verification including code coverage and behavioral (or functional) coverage.

The first step includes the generation of suitable documentation for the IP block. The second step includes code design, synthesis, and design for test. Somewhat surprisingly, the second step of implementation is only a small part of the reusable IP design process. In fact, depending on the size and type of the IP, the third step of verification may take up to 50% of the total time. Since the IP may be reused many times in a variety of unanticipated applications, design errors within the block cannot be tolerated. The goal of verification is to achieve 100% code coverage and close to 100% functional coverage to ensure the IP block works correctly.

The actual form of a reusable IP core can vary depending on the way in which the IP developer/vendor chooses to provide the core to the system designer. There are three main categories [2]: *soft*, *firm*, and *hard*. These forms are described below and their relationships and tradeoffs are depicted in Fig. 3.

- *Soft IP* blocks are specified using RTL or higher level descriptions. They are more suitable for digital cores, since a hardware description language (HDL) is process-independent and can be synthesized to the gate level. This form has the advantage of flexibility, portability, and reusability, with the drawback of not having guaranteed timing or power characteristics, since implementation in different processes and applications produces variations in performance. Sometimes, the HDL is obtained from a third party in an *encrypted* form which does not allow it to be modified. This makes it harder to adapt it for use in an unanticipated way, but it also prevents the user from introducing any new design errors into the block.
- *Hard IP* blocks have fixed layouts and are highly optimized for a given application in a specific process. They have the advantage of having predictable performance. This, however, comes with additional effort and cost, and lack of portability that may greatly limit the areas of application. This form of IP is usually prequalified, meaning the provider has tested it in silicon. This adds some assurance to its correctness.
- *Firm IP* blocks are provided as parameterized circuit descriptions so that designers can optimize cores for their specific design needs. The flexible parameters allow the designers to make the performance more predictable. Firm IP offers a compromise between soft and hard, being more flexible and portable than hard IP, yet more predictable than soft IP.

Until very recently, most digital IP blocks came in the form of hard IP. For example, ARM and MIPS core vendors would provide behavioral models and black-box layouts of the processors for use during design and verification, and then "drop in" the hard IP at the foundry facility before fabrication. This afforded the vendor some level of IP protection while allowing the customer to carry out designs using the IP. More recently, soft IP has become the preferred handoff level. Typical soft IP (cf. [109]) include interface blocks (USB, UART, PCI), encryption blocks (DES, AES), multimedia blocks (JPEG, MPEG 2/4), networking blocks (ATM, Ethernet), and microcontrollers

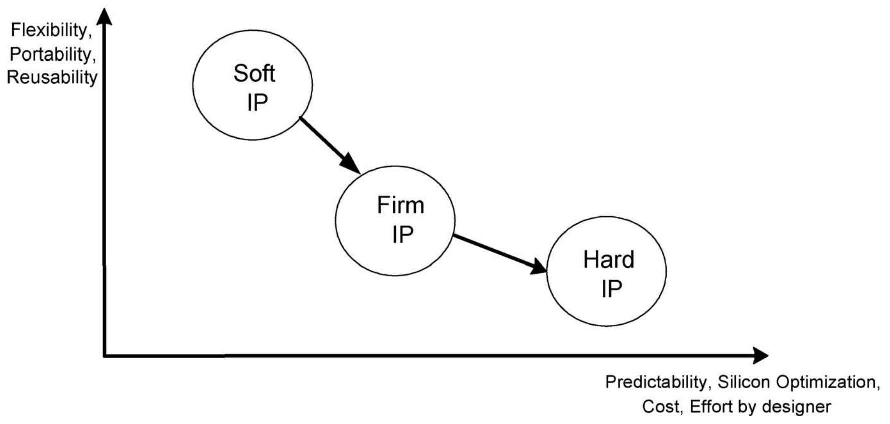

Flexibility, Portability, Reusability
Soft IP
Firm IP
Hard IP
Predictability, Silicon Optimization, Cost, Effort by designer

Fig. 3. ***Different types of IP blocks [4].***

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.

(HC11). The RTL descriptions usually are *configurable* in that certain parameters are user-definable to tailor the block to the needs of the customer. This allows selective inclusion or exclusion of distinguishing features which can impact the final implementation's performance, cost, and power. In the case of a processor core, parameters such as the bus width, number of registers, cache sizes, and instruction set may vary from customer to customer, so the flexibility of soft IP allows for their modification before synthesis. Commercial tools exist for this purpose in which a configurable processor with specific attributes can be automatically generated [15]. Such flexibility provides lower area, power, and improved performance, since the IP block can be tuned for each application. In addition, some features in a given IP block that are not needed by all customers can be removed. With some effort, the final design generated from a soft IP block can be within 20%–30% of the power and timing of a hard IP implementation.

B. Analog/Mixed-Signal Design for Reuse

While design productivity can be improved significantly with the use of digital IP blocks, another bottleneck exists if the designs include analog and mixed-signal components. Digital design has a well-defined, top-down design methodology but analog/mixed-signal (AMS) design has traditionally been an *ad hoc* custom design process. When analog and digital blocks coexist on the same substrate, the analog portion of the design can be more time-consuming to develop even though it may represent a smaller percentage of the chip area. In a few years, it is anticipated that more than 70% of all SoC designs will have some form of analog/mixed-signal blocks [1]. This increase is consistent with the expected growth of the wireless industry over the same period.

In order to keep pace with rapidly evolving markets, the productivity of AMS design can be improved using a mixed-signal SoC design flow [16], [17] employing AMS IP [18]–[20]. One of the main advantages of the use of AMS IP in SoCs is the potential reduction in power, which is especially important in battery-operated applications such as personal digital assistants (PDAs), wireless local area networks (LANs), etc. Typical AMS components include operational amplifiers, analog-to-digital converters (ADCs), digital-to-analog converters (DACs), phase-locked loops (PLLs), delay-locked loops (DLLs), serializer/deserializer transceivers, filters, voltage references, radio-frequency (RF) modules, voltage regulators, analog comparators, etc. Many of these blocks are delivered in the form of hard IP and targeted to one application in a specific fabrication technology. Therefore, they cannot be easily migrated to other applications and other technologies by the end user.

Compared to digital IP, AMS IP must provide an even greater degree of flexibility in the design parameters and performance characteristics. While the general function of an analog block may be the same in different applications, the design specifications may vary widely between the applications. Furthermore, the performance of AMS IP blocks is significantly influenced by parasitics and interactions with the surrounding environment, often in the form of power supply fluctuations and substrate noise effects.

Currently, because of the complexity of analog/mixed-signal design and its sensitivity to the surrounding environment, AMS blocks are most commonly presented in the form of hard IP. However, this form has limited scope of applications. Hard IP will reduce the design cycle significantly when the specifications and fabrication processes are identical, but will not greatly improve the design cycle if it has to be modified in any way or migrated to a new process. This calls for a more flexible definition for the format in which the AMS IPs are provided. Firm IP appears to be the most appropriate format to deliver the AMS IP library components. In this form, the IP captures suitable schematics of the analog blocks with parameters that are adjustable to optimize the design for specific applications. Unlike hard IP, this form allows ease of migration of IP from foundry to foundry, customer to customer, and application to application.

The traditional flow for AMS design relies heavily on the expertise and experience of the designer. The design process begins with the performance specification of the component for a target application. Ideally, the AMS block should be described at a high level in the form of soft IP as in the digital case. System-level designers typically use tools such as MATLAB [14] for specification and simulation. In addition, analog/mixed-signal hardware description languages (AMS-HDL) are increasingly used to model these types of circuits. The languages are a relatively new addition to the design process, and the most commonly used are *Verilog-AMS* [21] and *VHDL-AMS* [22]. Automatic generation of AMS architectures from AMS-HDL is still in its infancy because of the large number of variables associated with AMS design. However, the current contribution of these AMS-HDLs to system-level design is highly significant. They provide the necessary platform for system-level verification, an important part of design quality. Verification of mixed-signal SoCs requires cosimulation of analog and digital behavioral models to reduce simulation costs [24].

Since AMS blocks cannot be easily synthesized from a high-level specification without low-level support, designers must follow a design process such as the firm *IP hardening flow* [3], [20] illustrated in Fig. 4. The starting point of the flow is the set of selected library components that comprise the unoptimized schematic view of the design. This library consists of parameterized reusable components and is an essential part of the design flow. The model parameters are set by an optimization tool to achieve the derivative design specifications. After an architecture is chosen, the firm IP is taken through the IP hardening flow for optimization of the circuit parameters to maximize

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.

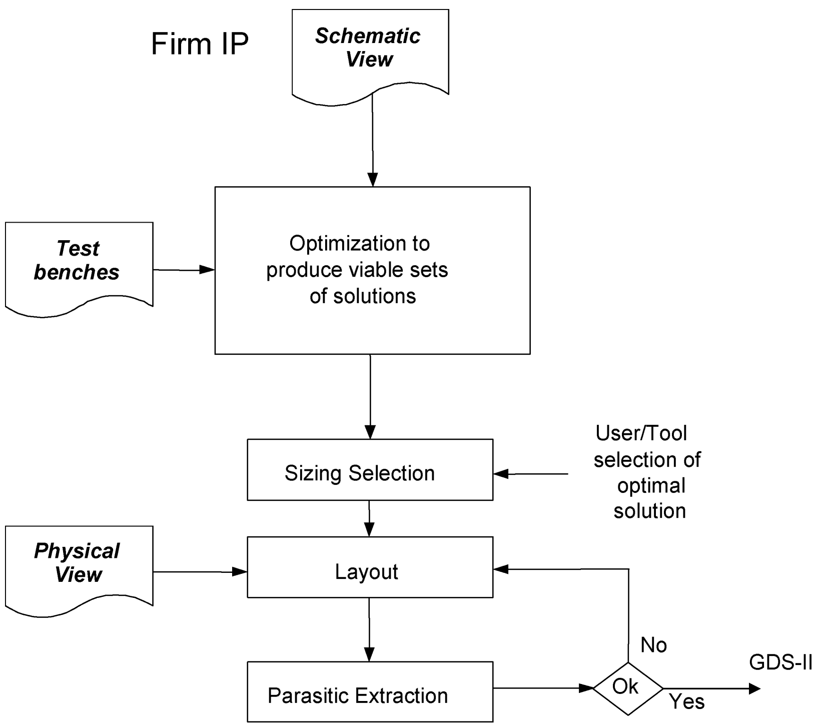


Fig. 4. ***Proposed analog/mixed-signal IP hardening flow.***

performance and to generate the final GDSII layout of the block. Proper modeling of the interfaces between the different blocks is important in the design process to account for different effects such as loading and coupling. This is needed to achieve correct performance when used in the overall system-level design [23].

When developing firm reusable AMS cores [18], [20], the usual precepts of a good design must be followed. That is, there should be a good formal specification, a good architectural design and a good circuit implementation. In addition, there are a number of steps that must be followed to achieve reusability, as discussed below. A successful IP block should be parameterized, easily verified through reusable test benches, well documented, and have associated *views* to ease the derivative design process. Specifically, an AMS IP block should have a behavioral/analytical view (in AMS-HDL), a parameterized schematic view (transistor level), and a layout view (floor plan). In addition, test benches are needed to validate the performance of the circuit under different operating conditions and at various process corners. They are used as the basis for verification of specifications and for exploration of the design space for the system.

It is clear that the development of AMS IP must take a different approach compared to digital IP development. The IPs must be able to handle and transfer both design experience and heuristics from the original design to subsequent design derivatives. In reality, this constitutes the reusable IP in the analog design process.

C. Programmable IP

As SoCs become larger and more complex, the design costs are so high that it becomes important to incorporate programmability within the SoC to allow for reuse at the chip level. This programmability can appear in a number of forms: *hardware programmability* using programmable logic cores and *software programmability* using an embedded processor. The key to programmable SoC designs is to provide some form of flexible hardware and/or software infrastructure, often called the *programmable fabric*. We distinguish between two types of flexibility: *prefabrication* and *postfabrication*.

Prefabrication flexibility involves the use of a structured ASIC fabric (cf., [25], [26]). In this approach, the lower layout layers of the structured ASIC fabric are predetermined and chips are already partially fabricated. The programming is performed by modifying the upper *metal* and *via* layout layers just before final fabrication. Since only a few steps are performed to complete fabrication, the turnaround time is relatively short. With this approach, however, the design cannot be changed after the chip is manufactured.

Postfabrication flexibility allows the behavior of an IP block to be modified after the chip has been fabricated, by either rewriting software/firmware to run on an embedded processor (software configurability) or by reconfiguring embedded programmable hardware, which involves changing the state of configuration bits embedded in the fabric (hardware programmability). The most compelling

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.

benefit of postfabrication flexibility is the ability to change the behavior of the IC as requirements change (such as communication standards or customer demands). If the "personality" of the design can be changed by rewriting software or reconfiguring the programmable hardware, the creation of a new version of the IC can be avoided; this can typically save several months and significant design costs. The primary disadvantages of postfabrication flexibility (compared to a fixed-function chip) are higher power, lower density (and hence a reduction in the amount of circuitry that can be implemented on a single IC), and reduced speed. However, for many applications, the advantages outweigh the disadvantages.

1) Hardware Programmability: Hardware programmability is enabled by the incorporation of one or more *programmable logic cores* into an SoC [27], [28]. The programmable logic core is a flexible logic fabric that can be customized to implement any digital circuit after fabrication. Such an embedded core may look very much like the programmable fabric in a stand-alone field-programmable gate array (FPGA) [29]–[31]. Before fabrication, the designer embeds the fabric (consisting of many uncommitted gates and programmable interconnects between the gates) onto the chip. After the fabrication, the designer can then program these gates and the connections between them.

Fig. 5 shows a hypothetical SoC containing such a core. In this example, the PC interface, the data processing block, and the network interface datapath are implemented using fixed-function IP blocks. However, rather than implementing the network interface control functions using fixed logic, the SoC designer embeds a generic programmable logic core onto the IC. The fabric consists of logic blocks and routing channels with switch blocks at their intersections. Both the logic and interconnect are programmable. After fabrication, the network interface control functions are mapped to and implemented on the programmable logic fabric. This postfabrication mapping is typically done using automated tools that are different than the tools used to implement the SoC before fabrication. Other SoCs containing programmable logic cores have been described in [32]–[35]. The internal structure (architecture) of programmable logic cores often inherits much from the architecture of stand-alone FPGAs. The optimization of stand-alone FPGAs have been well studied [36]–[38]. Flexibility in these architectures occurs in two ways: the individual logic elements that make up the fabric can be configured, and the interconnection between these elements can be configured. The logic elements themselves are often lookup tables which can be configured to implement any function of between three and eight inputs [39], [40]. Fig. 6(a) shows an example lookup table which can implement any function of three inputs. Eight configuration bits are used to program the function implemented by the lookup table. In addition, a configuration bit controls whether the function output is registered or combinational. The state of each of these configuration bits is set when the programmable logic core is programmed, usually at power-up. Other logic elements based on multiplexers [31], product terms [104] or arithmetic units [41]–[43] can also be used.

The lookup tables are connected using fixed metal tracks; the metal tracks are connected to each other and to lookup tables using pass transistors or buffered bidirectional connections controlled by programming bits, as shown in Fig. 6(b). Again, the state of each of these programming bits, and hence the connections between the lookup tables, are set when the programmable logic core

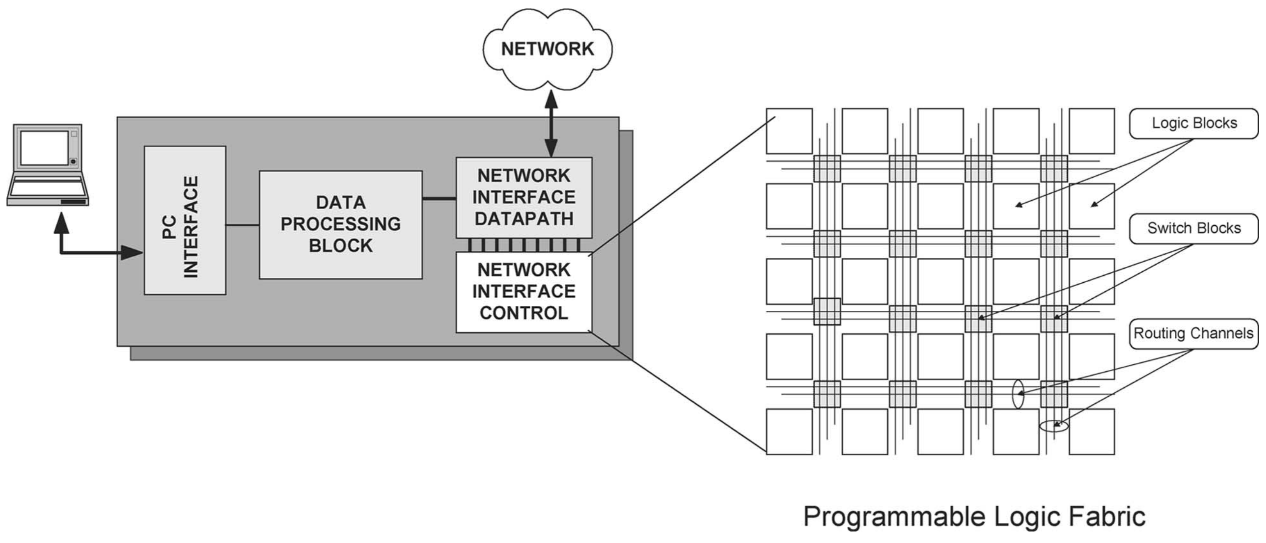


Fig. 5. ***SoC application containing a programmable logic fabric.***

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.

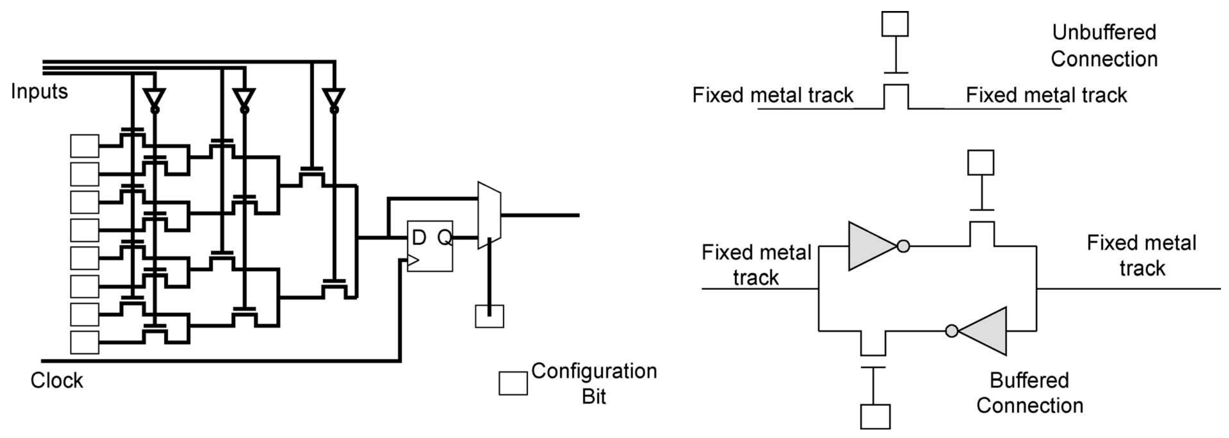


Fig. 6. ***Programmable fabric architecture.***

is configured. Considerable research has been aimed at optimizing the topology of these fixed metal tracks and the pass transistors and programming bits used to connect them for stand-alone FPGAs [37], [44]; some optimizations have been presented which specifically target general-purpose embedded programmable logic cores [45] and domain-specific ones [105]. In addition to logic and interconnect, programmable logic cores also often contain flexible memory arrays [46], dedicated arithmetic blocks, and high-speed communication blocks [29], [30].

The programming bits can be stored in a number of ways, typically by using static RAM (SRAM) bits. Implementing configuration bits using SRAM bits has the additional advantage that no extra processing steps beyond those needed to construct the other IP blocks on the SoC are required. Alternatively, in a structured ASIC fabric, the programmable switches in the connection fabric can be implemented using *programmable metal and vias*, in which connections between adjacent metal layers can be made by adding or removing metal [47], [48]. Typically, these programmable vias can be programmed only once (during fabrication), while programmable SRAM configuration bits can be programmed any number of times. While there is considerable overhead for programmable fabrics, the viability of the embedded programmable logic core approach is expected to improve at the 90- and 65-nm nodes.

As described earlier, IP cores can either be hard cores or soft cores, depending on whether the core is delivered as a layout or in the form of a hardware design language (HDL) description. The same is true for programmable logic cores. The programmable logic cores described above are hard cores. A soft programmable logic core is described in [32]; in this architecture, the behavior of the core is described using an HDL (note that this is different than the behavior of the circuit to be implemented in the core). The soft core makes integration simpler, since standard SoC tools can be used. However, the overhead for a soft programmable logic core is typically six times larger than that for a hard programmable logic core, meaning it is only suitable if only small amounts of programmable logic are required. Development of automated hard-core generators for programmable logic cores is also a continuing area of research [106]–[108].

2) *Software Programmability:* Software is the most natural form of programmability for an SoC. Embedded software [49], [50] allows a single SoC to have different "personalities" and serve different customers or market segments. From a design perspective, as much of the solution should be implemented in software as possible to maximize the flexibility of the design. Complex control functions are better suited to software implementations whereas data operations are better implemented in hardware. If performance is an issue, then hardware must be used for the implementation which can take the form of a programmable logic fabric, which is roughly five times faster than software, or an ASIC implementation, which is typically 50 times faster than software (based on the authors' experience).

Reuse in software is provided through the use of libraries of code and data structures, along with off-the-shelf kernel and real-time operating systems (RTOSs) to improve productivity. Unlike hardware, the software or firmware is never actually completed; it is under continuous development but the code is *frozen* and released as a version with the intention to update it in the future (and perhaps provide bug fixes when needed). Since SoC programmability is headed in this direction, there will be a growing need for more software/firmware designers in an industry typically dominated by hardware engineers.

III. PLATFORM-BASED DESIGN

Productivity gains obtained strictly from reusable IP has its limits, since the process of chip integration can be very

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.

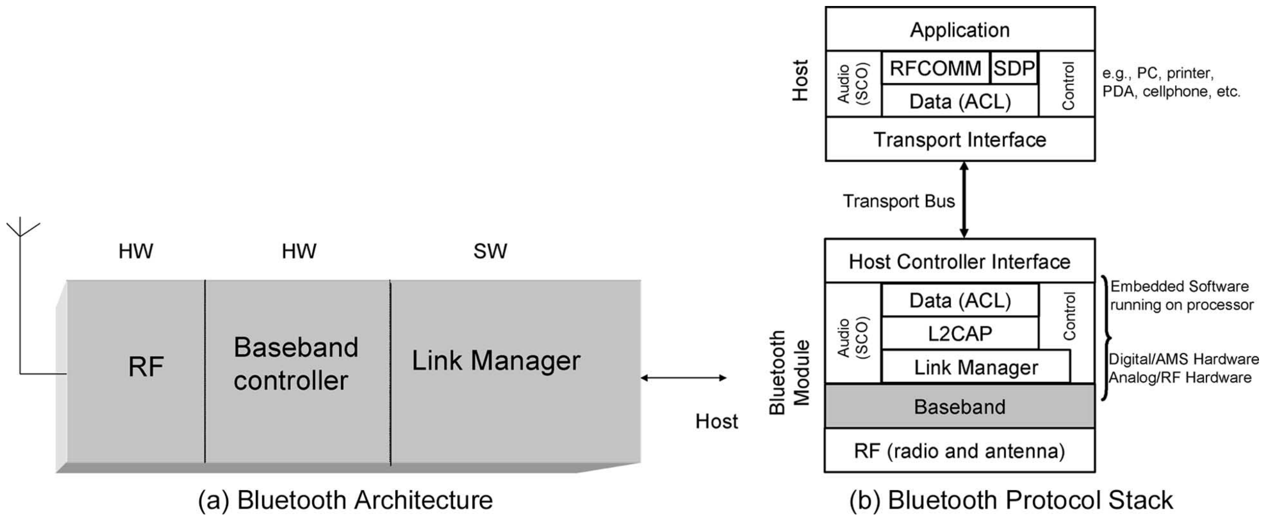


Fig. 7. ***Bluetooth architecture and protocol stack.***

time-consuming and expensive even with predesigned blocks. What is needed to improve productivity is a higher level of design abstraction. Economies of scale can be derived from the fact that a single architecture may serve many different customers in one market segment. Furthermore, as the needs of a marketplace change, it should be possible to use a configurable architecture as a basis for new designs. For these reasons, the concept of *platform-based design* [4] was introduced where new designs could be quickly created from an original platform to amortize costs over many design derivatives. More specifically, a platform is an abstraction level that covers a number of refinements to a lower level. It allows significant improvements in designer productivity, since many high-level and low-level decisions have already been incorporated into the platform, and all associated tools and flows are in place to quickly generate new designs.

An SoC platform consists of hardware IP, software IP, programmable IP, standardized bus structures and communication networks, computer-aided design (CAD) flows for hardware/software codesign, verification and implementation, system validation tools, design derivative creation tools, and dedicated hardware for system prototyping. As an example of an SoC platform, consider a Bluetooth device [51] for low-power wireless personal area networks (WPANs). Bluetooth operates in the unlicensed ISM band at 2.4 GHz and uses frequency hopping spread-spectrum techniques. The overall architecture of a Bluetooth device consists of an RF front-end, a baseband controller, and software to implement the Bluetooth protocol stack. A simplified architecture is shown in Fig. 7(a) and the protocol stack is shown in Fig. 7(b) [51].

When considering all of the hardware, software, design and verification flows, technology, and testing issues, it is clear that the amount of effort required to carry out a chip design is prohibitive in the market windows associated with this type of product. In particular, as the Bluetooth standard is upgraded or customer requirements shift, the design may become obsolete. Furthermore, there are other personal area networking standards under development that have similar characteristics. These considerations make WPAN is a suitable application for platform-based design.

The platform concept will be illustrated using the baseband processing unit of Bluetooth assembled and verified at the University of British Columbia (UBC) using SoC/IP design and verification methodologies. Since all of the IP was obtained through third-party vendors, the tasks of design, verification, and logic synthesis required five engineers and roughly one year to accomplish in a university research environment. A block diagram of the reference platform for the mixed-signal SoC is shown in Fig. 8. The software portion of the protocol stack runs on the ARM7TDMI processor.[1] It is responsible for the Bluetooth functions associated with establishing connections between devices and interpreting the packets to determine the services being requested by other devices.[2]

The processor requires high-speed connections to certain blocks such as the shared memory controller (SMC), the direct memory access (DMA) controller, the test interface controller (TIC), and the power and clock control units. The connections are implemented using the synchronous advanced high-speed bus (AHB) of the AMBA bus standard [7]. This bus requires the development of a bus arbiter and decoder functions. The other components in the system operate at different speeds and

[1]ARM7TDMI is a trademark of ARM Holdings Ltd.

[2]This platform was originally developed by Tality, which is part of Cadence Design System Inc.

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.

are connected using the asynchronous advanced peripheral bus (APB). These components include the watchdog timers, other timers, general-purpose input/output (GPIO) devices, and programmable interrupt controller (PIC), UART, and USB interface standards. The baseband controller (BBC) that performs the DSP functions on the incoming and outgoing bit streams and radio control functions is also interfaced to the APB. A bus bridge is used to interface the AHB to the APB. The data received from the ADC and transmitted out through the DAC (and eventually to/from the RF module) would be implemented as a separate chip in this case.

To construct the platform,[3] the aforementioned components were acquired through third parties or developed in-house. The ARM7TDMI processor was delivered in the form of hard IP, along with RTL models for simulation and verification. The software protocol stack was separately acquired. The AMBA bus components were developed by a third party as soft IP, along with the baseband controller and all the APB functional units. The memory blocks were generated using a commercial memory compiler. The AMP IP functions required are the ADC, DAC, and PLL, and would be purchased separately; we used behavioral models for these blocks. With these IP components and the associated CAD flows in place, the platform was built, verified, and synthesized within one year.

Programmability can be built into the platform in three ways. First, the ARM processor itself is programmable. Any changes to the protocol stack functions can be easily accommodated. It can also be enhanced with a local embedded FPGA to speed up critical sections of the code. Other interface blocks can be replaced by a similar embedded FPGA, so long as the resulting fabric does not consume a large area or exceed the power specifications of Bluetooth. The BBC is too large for such a fabric but may benefit from the use of a structured ASIC implementation. In this way, as the standard changes or new algorithms are devised, the BBC can be modified accordingly.

[3]The IP components and CAD tools were acquired through the services of the Canadian Microelectronics Corporation.

IV. ON-CHIP COMMUNICATION INFRASTRUCTURE

A typical SoC today consists of many cores operating at different clock frequencies and this presents additional issues for IP integration. For example, digital signal processors (DSPs) and network interface cores include circuits whose clock frequencies must match the sample rate or bit-rate of the data that they are processing. Apart from these types of constraints, different cores may be designed to operate at different clock rates, a situation that is nearly unavoidable when cores are obtained from several sources, both internal and external to a company.

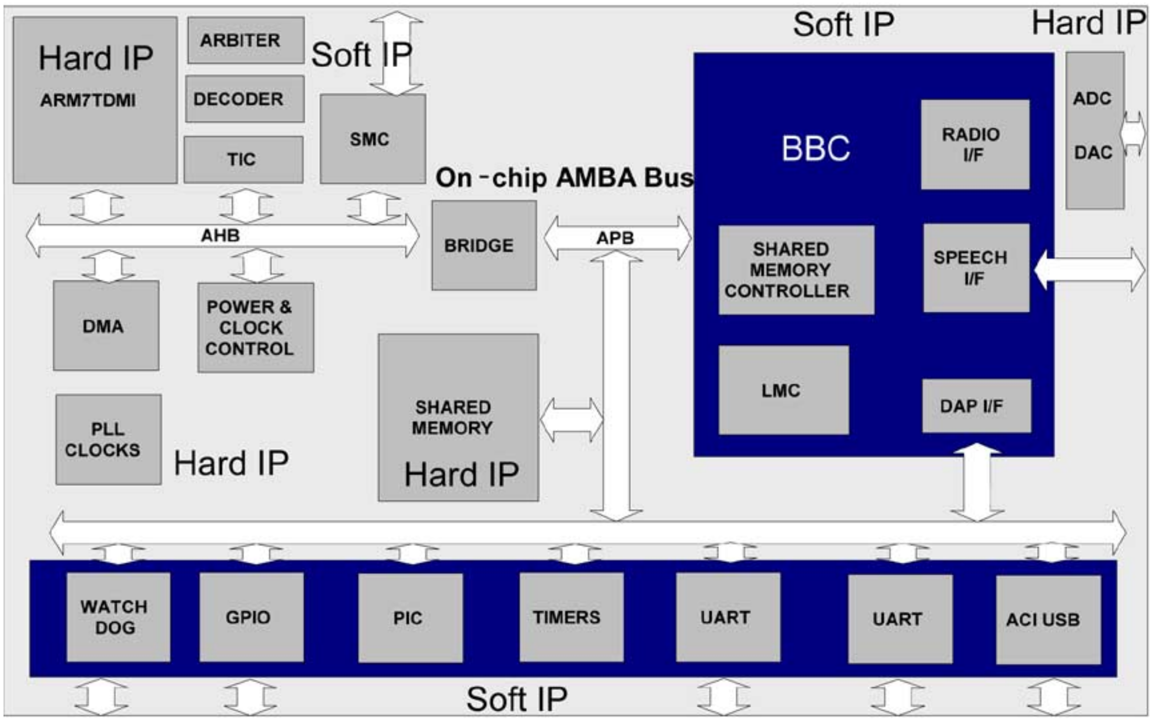


Fig. 8. ***Bluetooth baseband platform.***

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.

Design flows based on traditional logic synthesis assume a synchronous implementation: all timing issues are regulated by a single, global clock. More recently introduced system level design languages such as System-C [52] and System Verilog [53] allow designs to be described as communicating processes by using transactional models. With this decoupling, each core in an SoC design can be viewed as a separate synchronous island, and the interfaces between these islands are provided by the chip's global interconnect. This approach is commonly referred to as globally asynchronous, locally synchronous (GALS) design [54], [55]. The work in [56] adapted the GALS idea to define "latency insensitive" designs, wherein the interconnect can be either synchronous or asynchronous. The key idea in this approach is to restrict the system-level design to one where correct functionality can be guaranteed regardless of the latency of links between cores. This allows the interconnect to be pipelined and provides a separation of timing and functionality similar to that of traditional synchronous design. However, latency remains critical for the performance of many SoC designs. Integrating latency and performance requirements into an SoC design flow remains an area of active research [57], [58]. Synthesizing designs from multitimed, transactional descriptions is another topic of ongoing research.

When communicating between different clock domains, the possibility of synchronization failures arises. Without a fixed relationship between sender and receiver clocks, the data from the sender clock may change at an arbitrary time with respect to the receiver clock. This can lead to set-up and hold violations or other failures in the receiver's core. The underlying cause is that any device with two stable states (e.g., a flip-flop) must have a metastable state that can occur under marginal triggering conditions [59], [60]. While such a state can persist indefinitely, the probability of a failure drops exponentially with the amount of time allowed for the device to settle. Mathematically, this can be expressed as

$$\text{Prob}\{\text{failure}\} = e^{-t/\tau} \qquad (1)$$

where t is the time allowed for the flip-flop to settle, and τ is the time constant for the device.

A common rule of thumb is to use two flip-flops to implement a reliable synchronizer. As shown by (1), the probability of failure is reduced by the time allowed for the signal to settle rather than the number of clock cycles. Thus, for designs with fine-grain pipelining, a two flip-flop synchronizer will result in unacceptable failure rates. Conversely, designs with low clock frequencies can safely perform synchronization in a half clock cycle or less [61]. A survey of common pitfalls in synchronizer design is presented in [62].

While a synchronizer can be used to ensure that individual bits are conveyed with acceptable reliability, SoC interconnect typically employs some form of parallel interconnect, and care must be taken to make sure that all of the bits of a word are transferred together. While this can be accomplished with double-buffering and synchronizing the full/empty signals, higher throughput can be achieved by pipelining and synchronization operations. Researchers in [63] have implemented a general set of interfaces between timing domains that exploit such pipelining [64]. If special timing relationships are known between the two domains in advance, then more optimized interfaces can be implemented [66]. For example, a family of simple, low-latency interfaces are described in [65]. We expect that future SoC solutions will use dedicated synchronization solutions such as those presented in [65] for interfaces where performance and latency are critical, and more generic interfaces such as those in [64] for less critical interfaces between cores.

A. Network-on-Chip

A more structured interconnect fabric is being pursued in the research community for commercial designs that must integrate a large number (10–100) of IP blocks in a single SoC. Today, there exist many SoC designs that contain a number of processors in applications such as set-top boxes, wireless base stations, HDTV, mobile handsets, and image processing [67]. Such systems behave as multiprocessors, and require a corresponding design methodology for both their hardware and software implementations. Power budgets and cross-chip signaling constraints are forcing the development of new design methodologies to incorporate explicit pipelining and provide a more structured communication fabric [67], [68]. Many researchers have suggested that future designs will be dominated by arrays of processors that form the basis of new multiprocessor SoC platforms (the so-called MP-SoC platforms [67]).

Network-on-chip (NoC) [68], [69] infrastructures are emerging as the new paradigm that characterizes the on-chip data communication architecture of large-scale SoCs. The integration of several components into a single system gives rise to new challenges. As shown in Fig. 9, there are a wide variety of topologies that have been proposed for NoC [74], including SPIN [85], CLICHÉ [83], torus [69], folded torus [69], [72], and irregular and bufferfly fat-tree [72], [82], [84]. The practical implementation and adoption of the NoC design paradigm faces various unsolved issues related to design methodologies, test strategies, and system reliability.

B. Design Considerations

It is well known that it can take several clock cycles for a global signal to travel from one end of a chip to the other [81]. To cope with this issue, the end-to-end communication medium can be divided into multiple pipelined stages, with delays in each stage comparable with

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.

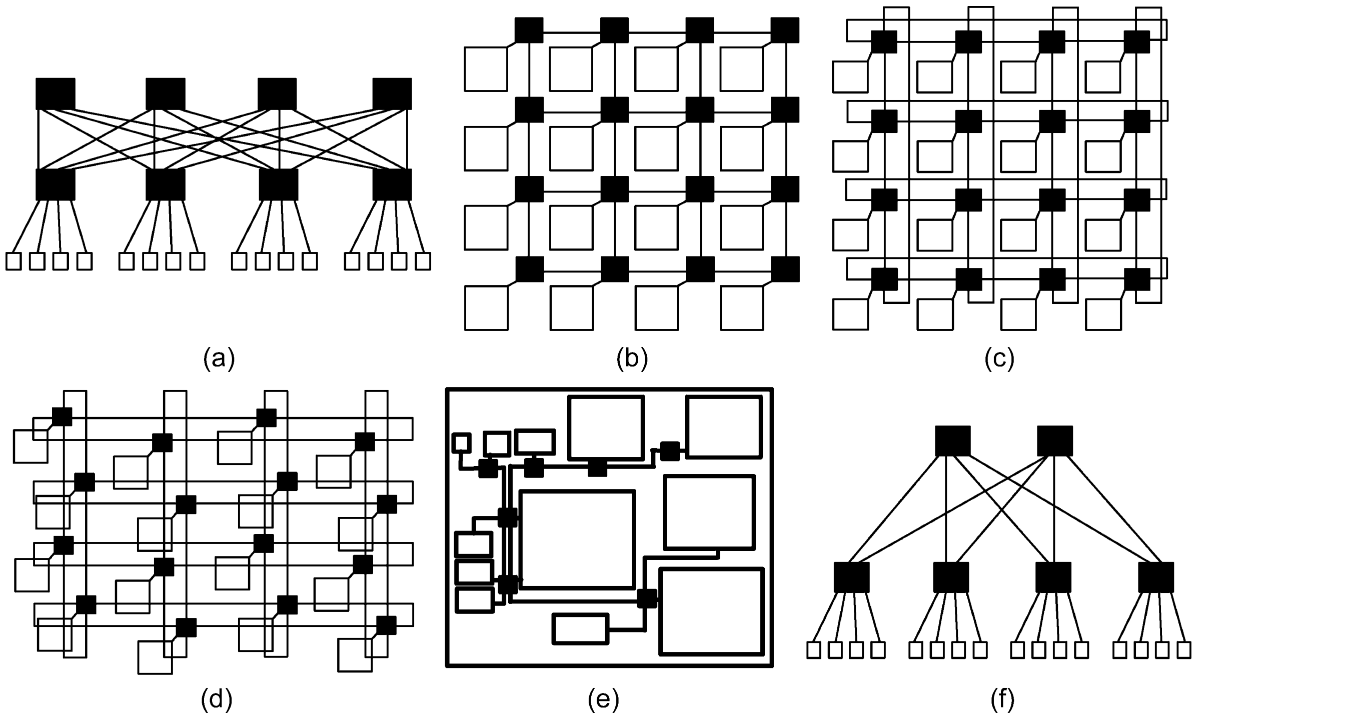


Fig. 9. *NoC topologies. (a) SPIN. (b) CLICHÉ. (c) Torus. (d) Folded torus. (e) Irregular. (f) BFT.*

the clock-cycle budget. In NoC architectures, the inter-switch wire segments together with the switch blocks constitute a highly pipelined communication medium characterized by link pipelining, deeply pipelined switches, and latency-insensitive component design [76]. Link pipelining is inherently built-in to regular NoC topologies. The switch designs generally consist of multiple pipeline stages and the delay of each stage is less than the target clock period in a particular technology node [78].

The design process of NoC-based systems borrows some of its aspects from the parallel computing domain, but it is driven by a significantly different set of constraints. From the performance perspective, high-throughput and low latency are desirable characteristics of MP-SoC platforms. However, from a VLSI design perspective, the power dissipation profile of interconnect is increasing in importance as it represents a significant portion of the overall energy budget. The silicon area overhead due to the interconnect fabric is also important.

In the context of NoC architectures, there is a trend toward using packet-based on-chip communication. The common characteristic of these kinds of architectures is that the IP cores communicate with each other through high-performance links and intelligent switches. Various schemes for packetized communication are viable according to the local/global requirements on *quality of service* (QoS). Therefore, switching and routing schemes will require switch blocks with various characteristics such as data buffering, virtual channels, priority-based schemes, etc. [71]. The switch design also depends on the routing scheme adopted. There are two broad categories of routing: deterministic and adaptive. Deterministic routing algorithms always supply the same path between a given source/destination pair. Adaptive routing algorithms use information about routing traffic and/or channel status to avoid the congested or faulty part of the network. If deterministic routing schemes are adopted, the switches can be designed to be fast and compact [73].

The block-level representation of a switch is shown in Fig. 10. It mainly consists of input/output FIFO buffers, input/output arbiters, and a routing block. Recent publications indicate that *wormhole switching* [86] is the solution of choice for NoCs [72]. In wormhole switching, the packets are divided into fixed length flow control units (*flits*) and the input and output buffers are expected to store only a few flits. As a result, the buffer space requirement in the switches can be small compared to that generally required for other schemes. Thus, using a wormhole technique, the switches will be small and compact. The first flit, i.e., *header flit*, of a packet contains routing information. Header flit decoding enables the switches to establish the path and subsequent flits simply follow this path in a pipelined fashion. As a result, each incoming data flit of a message packet is simply forwarded along the same output channel as the preceding data flit, and no packet reordering is required at the destination. If a certain flit faces a busy channel, subsequent flits have to wait at their current locations.

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.

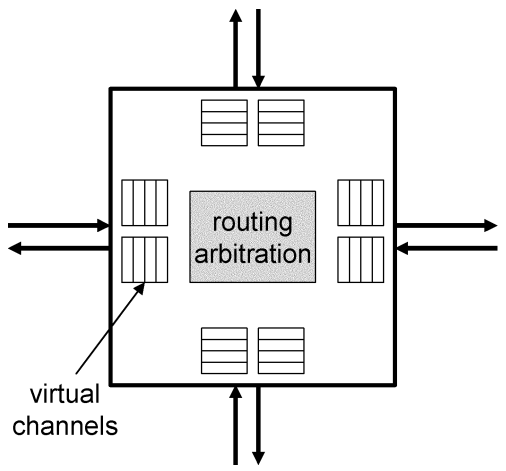


Fig. 10. ***NoC switch architecture.***

One drawback of this simple wormhole switching method is that the transmission of distinct messages cannot be interleaved or multiplexed over a physical channel. This will decrease channel utilization if a flit from a given packet is blocked in a buffer. In order to have a considerably high throughput, a set of virtual channels can be used (two per input/out port are shown in Fig. 10). If a flit belonging to a particular packet is blocked in one of the virtual channels, then flits of alternate packets can use the other virtual channel buffers and, ultimately, the physical channel to continue on their path to the destination.

Streamlined and fast switches can also be realized to support simple adaptive routing schemes. For example, in the Nostrum NoC [74], the switches realize a congestion-driven deflective routing scheme for a mesh/torus network architecture. In [75], this routing scheme was combined with wormhole switching. The switches are combinational blocks, where the decision to reroute a packet (i.e., one that cannot be routed efficiently toward the destination) is based on the analysis of traffic congestion of the neighboring nodes.

It is likely that NoC architectures will make their way into SoC designs in an evolutionary manner rather than a revolutionary manner, driven initially by MP-SoC applications [77] and then later by the need to have manufacturable structured interconnect as technology scales below 90-nm CMOS.

V. SoC TEST METHODOLOGIES

Another important aspect of SoC integration is the development of a test methodology for postmanufacturing tests. Core testing strategies often accompany a third-party IP block when it is purchased or otherwise acquired. However, system-level test integration is left to the SoC platform designer. Conceptually, testing of traditional SoB and the current SoC designs have many similarities. ICs on a printed circuit board are the components of SoB, whereas cores in a core-based system are the virtual components of an SoC. However, the similarities stop there; in fact, the manufacturing test procedures of SoBs and SoCs are quite different [70].

In the SoB approach, IC design, manufacturing, and testing are all performed by the IC provider. The system integrator is responsible for design and manufacturing of the board-level design using these ICs. As the provider tests the ICs, the system integrator can assume fault-free ICs. In SoC, the core provider only delivers a description of the core design to the system integrator; the system integrator then designs any proprietary blocks, called User Defined Logic (UDL), and assembles the predesigned cores. It is not possible for the core provider to do the manufacturing test, as the system is yet to be manufactured. Therefore, the system integrator is responsible for testing the core logic and the wires between cores. The best the system integrator can expect from the core provider is that the core's design description is delivered with a set of test patterns with high fault coverage. Ideally, test development for IP blocks should be carried out with reuse and system-level integration in mind.

The core tests from the core providers are originally described at the input/output terminals of the core itself. When these cores are integrated in an SoC, the final test is to be applied at the input/output pins of the SoC. However, the core may be embedded deep into the SoC; its I/O pins may not be directly accessible from the external pins. This is another key difference between the SoB and SoC approaches. In an SoB, direct physical access to the chip peripherals is available to the system integrator for testing, whereas in an SoC core pins are not accessible to the user for manufacturing test. Consequently, an electronic test access mechanism is required from the SoC pins to the cores and *vice versa*. The test access to the embedded cores is the responsibility of the system integrator. This access mechanism requires additional logic and wiring; ultimately, this leads to the development of architectures for core test access.

A. IP Core Level Test

The test of an IP core typically consists of internal DFT structures and the required set of test patterns to be applied and captured on the core periphery. The test patterns need to include data and protocol patterns. The data patterns contain the actual stimulus and response values, whereas the protocol pattern specifies how to apply and capture the test data. The core internal test should be carried out by the core provider, as the system integrator, in most cases, has very limited knowledge about the structural content of the adopted core and hence considers it as a black box. It may not be possible for the system integrator to prepare the necessary test for it. Consequently, the core creator should be responsible for

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.

delivering: 1) the DFT hardware inside the core; 2) the test patterns of the core; and 3) the validation of those test patterns.

Another major task of the core provider is to determine the internal test requirements of the core without knowing the target process and application. For instance, which test method needs to be adopted, what types of faults to target, and what level of fault coverage is desired, are not known to the core provider. In the SoC scenario, a core provider may not know the target process and the desired test coverage level. Hence, the provided quality level may or may not be adequate. A built-in self-test (BIST) strategy [79] is another promising alternative to hide the test problem from the system integrator. However, to date, high test coverage levels have been difficult to achieve.

B. SoC Level Test

A conceptual architecture suitable for testing embedded core based SoCs is presented in Fig. 11, and has three principal components as explained below.

1) ***Test Pattern Source and Sink:*** The test pattern source generates the test stimuli and the sink receives the test responses.
2) ***Test Access Mechanism (TAM):*** The test access mechanism performs the on-chip test pattern transport. It can be used to transport either test stimuli from the test pattern source to the core under test or to transport the test responses from core under test to a test pattern sink.
3) ***Core Test Wrapper:*** The core test wrapper forms the interface between the embedded core and its environment. It connects the embedded core to the rest of the IC and also to the TAM.

To facilitate SoC testing, new standards to express the test procedure for both the core provider and the system integrator were developed and recently approved as IEEE 1500, which includes a core test language (CTL) [10]. Its purpose it to provide a uniform interface between the cores and the chip-level test access mechanism, analogous to how JTAG facilitates board-level testing [80]. In fact, P1500 (i.e., its previous name) is very similar to the legacy JTAG boundary scan in both architecture and operation. The most notable difference is the absence of the test access port (TAP) controller and the addition of parallel test port in P1500 wrappers. By detaching the TAP controller and providing more access ports, the serial-input constraint of JTAG is removed and a greater variety of test access mechanisms are supported.

The P1500 wrapper has four control inputs and one pair of serial data input and output as shown in Fig. 12. The serial wrapper scan input (WSI) is used to transport wrapper instruction and test data. Instructions for the wrapper are shifted serially into the wrapper instruction register (WIR) and various enable signals are generated from the control logic based on the content of the WIR and the four control inputs. The core data registers (CDRs) are used to capture test results or provide signatures to the BIST circuitry. The ring of flip-flops around the core form the boundary data register (BDR) that isolates the core's functional interface from the other blocks during testing. When exercising full-scan test on the wrapped core, the test vector is serially shifted in through WSI, and scan output is serially shifted out through to the wrapper scan output (WSO).

Fig. 13 shows the integration of the P1500 standard in SoC testing. Each core is encapsulated in a P1500 wrapper to provide a unified interface for test control purposes. The wrapper control signals can be generated by a user-defined test controller which is enabled by external sources. In addition, a user-defined parallel TAM can be implemented for test data transportation to/from individual IP cores. All of these items compose the infrastructure that supports the actual test of IP cores in an SoC design.

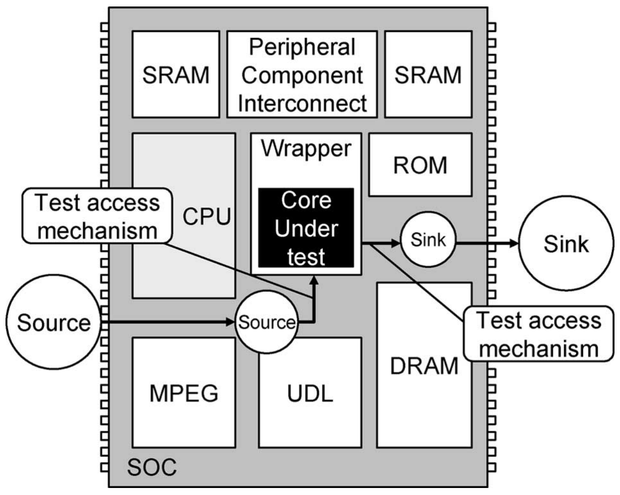


Fig. 11. *Core-based SoC test architecture [70].*

VI. SoC VERIFICATION

The verification challenges for SoC largely parallel those facing design and test—the challenge is the unprecedented complexity, and the hoped-for solution is through reuse—but the verification problem is in some respects harder. As the ITRS has noted [1], while design sizes have grown exponentially over time in accordance with Moore's Law, theoretical verification complexity has been growing double-exponentially, because the number of states that must, in theory, be verified is exponential in the size of the design. For example, consider an SoC built from n IP cores, with the ith core having some measure of verification complexity (e.g., number of reachable states, number of functional coverage points) of $v(i)$. If we simply assemble and connect the cores together and try to verify the entire SoC without exploiting the structure, we must

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.

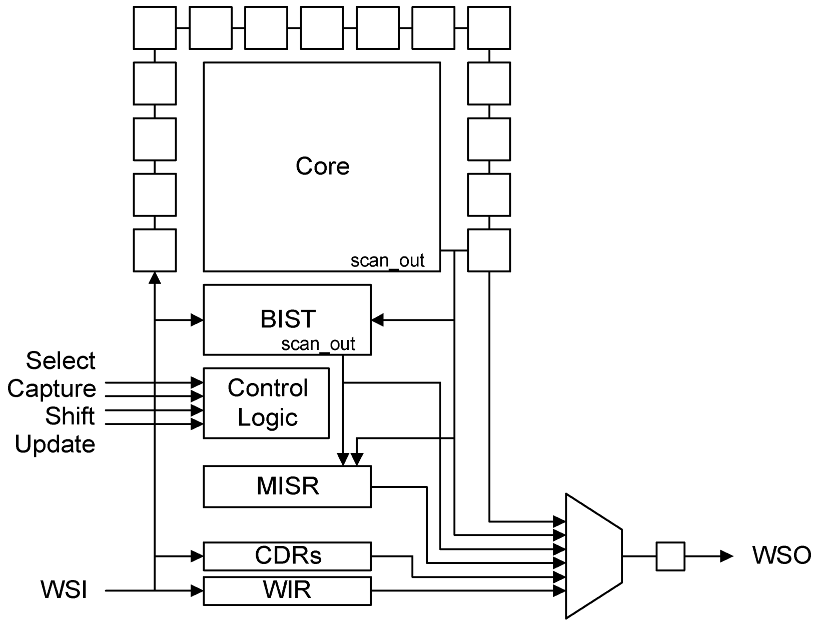


Fig. 12. ***Block diagram of a P1500 wrapper for a core using BIST DFT.***

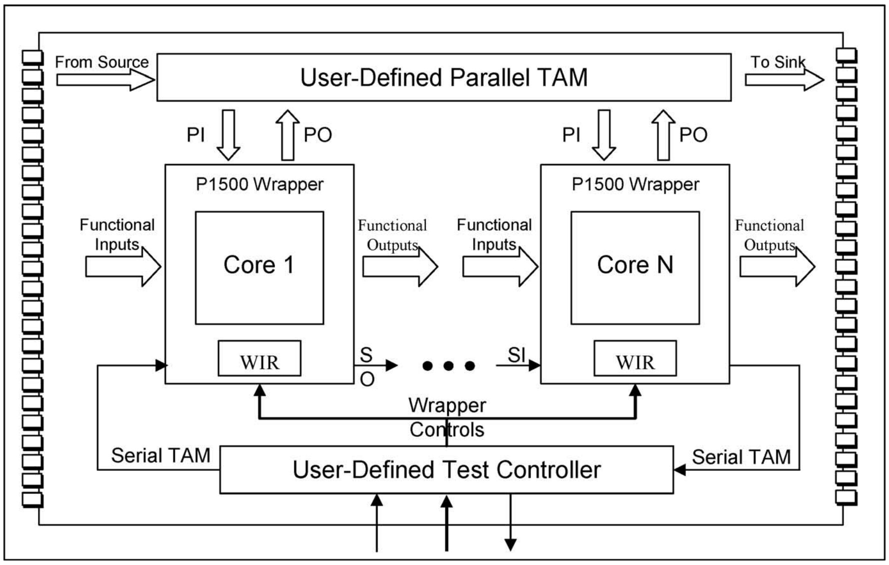


Fig. 13. ***Integration of cores using P1500 wrapper.***

consider the cross product of all possible verification states, yielding complexity $\prod_{i=1}^{n} v(i)$, which is exponential in n. The goal is to try to find a way to reuse the verification effort for each core, so that the cores can be verified once in isolation. Doing so would reduce the verification effort to

$$\sum_{i=1}^{n} v(i) + v_{\text{sys}}$$

where v_{sys} is a (hopefully smaller) term reflecting the effort of verifying the integrated system under the assumption that each core is correct. Furthermore, as cores are reused, we would ideally need expend only the v_{sys} for the next SoC.

Progress toward this ideal has been modest. In industrial practice, the main effort has been two-pronged: encapsulating verification information within the IP cores in an effort to make as much verification effort reusable as possible, and standardizing on-chip interconnection

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.

protocols to reduce bugs introduced during integration of the cores. Mainstream verification continues to rely heavily on dynamic methods (simulation and emulation), so the obvious initial step at encapsulation consisted of vendors simply supplying simulation testbenches along with their cores. Significant progress came with the widespread adoption of assertion-based verification [87]. In this methodology, designs carry with them, embedded in the code and on interfaces, assertions specifying correct behavior. Thus, a large amount of the specification of correct behavior is clearly and unambiguously spelled out. The acceptance of assertion-based verification, in turn, stimulated interest in standardizing languages for specifying more complex assertions, such as PSL [88] or SVA [89]. These languages have the expressive power of formal temporal logic (on which they are based [90], [91]), and have been accepted as industry standards. On the prong of standardizing interconnect protocols, several companies have proposed standards for on-chip communication, such as the AMBA family of busses [7] or the OCP protocol [8]. Obviously, adopting a standard interconnect does not eliminate all bugs arising from integration (since the system may not use individual cores correctly), but it does eliminate the common errors that occur when designing an interconnection protocol or from incorrect interfacing of cores and interconnect.

The above incremental advances have helped reduce the verification problem, by increasing confidence in individual cores and reducing integration bugs. However, dynamic verification methods intrinsically suffer from increasingly poor coverage as design complexity increases, due again to the fundamental fact that the set of behaviors of a system grows exponentially in its size.[4] Poor coverage increases the likelihood of undetected bugs in IP cores, undermining the goal of reusing IP cores without needing to completely reverify them. If each IP core has some residual probability p of containing undetected bugs (which could have been detected before integration, but were not due to poor coverage), then the probability of an error-free SoC with n cores again falls off exponentially as $(1-p)^n$, not even including errors introduced from integration. In the past, when individual logic blocks were simpler (so $p \approx 0$), or when systems were much smaller (so n was small), this probability of having bugs could have been acceptable. As SoCs grow, however, only a more formal verification process, which minimizes p and documents any assumptions being made, has any hope of scaling.

[4]It is important to distinguish different concepts of coverage used by different communities. In VLSI test, coverage is the fraction of faults, for a given fault model, that can be detected via a test set. In dynamic validation, coverage typically refers to the percentage of some coverage metric that the simulation test vectors have stimulated. Coverage metrics range from very crude measures like line coverage to the theoretical ideal of exhaustively simulating every possible behavior of the system, which we dub "behavior coverage." Formal verification has 100% behavior coverage by definition. In formal verification, "coverage" sometimes means to what degree a set of formal specifications completely specifies the desired behavior. We dub this concept "specification coverage" and note that it is an issue for both formal and informal verification. The scalability problem of dynamic validation is due to poor behavior coverage.

Most research activity, therefore, has focused on formal verification (primarily *model checking* [92]). Formal verification provides a 100% proof that a design meets its specifications. Furthermore, the specifications that are verified and the assumptions under which they are verified are precisely documented and can be exploited during integration verification. Methodological barriers to formal verification are falling, because the methodology advances in dynamic verification mentioned earlier are exactly what is needed to enable more formal verification. Formal verification is already indispensable industrial practice for certain applications, such as RTL-to-gate equivalence checking [93] and microprocessor verification [94], [95]. Formal verification is no magic bullet, however—the exponential state explosion that manifested itself as poor coverage in simulation reappears as the high computational complexity of formal verification algorithms. The on-going research effort in formal verification has been to improve the scalability of formal methods to larger, more complex designs. Specifically for SoCs, the same ideas of reuse and integration that simplify SoC design also hold promise for simplifying SoC verification. Well-specified cores interacting via clean and well-defined interfaces ought to admit easier verification than an arbitrary assemblage of logic. The key research areas are *compositional model checking* [96], which decomposes the verification of an entire system composed of several blocks into multiple, smaller verification tasks on the individual blocks; and *assume-guarantee reasoning* [97], which emphasizes verification of cores under assumptions about the behavior of the rest of the system. Both of these verifications can be performed separately. The other key research direction for formal verification of SoCs is in handling nonconventional IP: most research has emphasized blocks of digital hardware, but preliminary research is starting to appear on embedded software verification [98] and formal verification of analog circuits [99].

Finally, despite all the research advances now and in the future, it is unlikely the verification challenge can be solved without help from designers. Historically, whenever formerly second-order issues grew into first-order productivity challenges, e.g., sequential testability or low power, the solution ultimately involved design changes and some sacrifices, e.g., the overhead of scan latches and power-management logic. There is no reason to presume that verification will be different. Already, leading companies involve verification experts early in the design process to steer the design toward better verifiability [100] and ambitious projects are underway that completely integrate the design and verification process [101]. These sorts of efforts will likely solidify into standard practices for design-for-verifiability. There is even recent research where the very architecture of a system is optimized for easier verification; when done well, performance and cost

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.

penalties can be minimal [102], [103]. In the future, design-for-verifiability and architecture-for-verifiability will be absolutely essential.

VII. SUMMARY

This paper provided a broad perspective on the reuse and integration issues associated with mixed-signal SoC design. Reusable forms of digital, analog/mixed signal, and programmable IP component were described. The platform-based design concept was illustrated using a Bluetooth baseband processor. Integration issues associated with interconnect, testing, and verification were presented. The authors believe that almost all designs in the future will make use of reusable IP and that commercial tool vendors will continue to advance their tools to address the more challenging issues of system level hardware/software codesign and coverification. ■

Acknowledgment

The authors wish to thank their students and research engineers for the significant contributions to the work at the System-on-Chip Research Laboratory at the University of British Columbia. The authors also wish to thank the Canadian Microelectronics Corporation (CMC) for licensing the infrastructure used in this work, and for invaluable discussions regarding SoC design and reuse.

REFERENCES

[1] International Techbology Roadmap for Semiconductors. [Online]. Available: http://www.public.itrs.net

[2] M. Keating and P. Bricaud, *Reuse Methodology Manual: For System-on-a-Chip Designs*, 3rd ed. Boston, MA: Kluwer, 2002.

[3] R. Rajsuman, *System-on-a-Chip Design and Test*. Boston, MA: Kluwer, 2000.

[4] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, and L. Todd, *Surviving the SOC Revolution: A Guide to Platform-Based Design*. Boston, MA: Kluwer, 1999.

[5] G. Moore, "Cramming more components onto integrated circuits," Electronics, vol. 38, pp. 114–117, Apr. 19, 1965.

[6] A. Chandrakasan and R. Brodersen, *Low Power Digital CMOS Design*. Boston, MA: Kluwer, 1995.

[7] AMBA Specification (Rev 2.0), ARM Ltd., May 13, 1999.

[8] Open Core Protocol Specification 2.1. OCP-IP Assoc. [Online]. Available: http://www.ocpip.org

[9] Virtual Socket Interface Alliance. [Online]. Available: http://www.vsi.org

[10] M. Richhetti, "Overview of proposed IEEE P1500 scalable architecture for testing embedded cores (slide presentation," presented at the *Design Automation Conf.*, Las Vegas, NV, 2001, pp. 1–26.

[11] IEEE 1500 Standard. [Online]. Available: http://www.grouper.ieee.org/groups/1500

[12] R. Tummala and V. Madisetti, "System on chip or system on package?" *IEEE Des. Test Comput.*, vol. 16, no. 2, pp. 48–56, Apr.–Jun. 1999.

[13] T. Sakurai, "Low-voltage design or the end of CMOS scaling?" in *Int. Solid-State Circuits Conf.*, San Francisco, Feb. 2002, Evening Panel.

[14] The MathWorks. [Online]. Available: http://www.mathworks.com

[15] Xtensa Architecture and Performance, White Paper, Tensilica Inc., Sep. 2002.

[16] K. Kundert, H. Chang, D. Jeffries, G. Lamant, E. Malavasi, and F. Sendig, "Design of mixed-signal systems-on-a-chip," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 12, pp. 1561–1571, Dec. 2000.

[17] G. Gielen and R. Rutenbar, "Computer-aided design of analog and mixed-signal integrated circuits," *Proc. IEEE*, vol. 88, no. 12, pp. 1825–1852, Dec. 2000.

[18] N. Madrid, E. Peralias, A. Acosta, and A. Rueda, "Analog/mixed-signal IP modeling for design reuse," *Proc. IEEE Design, Automation and Test in Eur. Conf.*, Mar. 2001, pp. 766–767.

[19] Z. Li, L. Luo, and J. Yuan, "A study on analog IP blocks for mixed-signal SoC," in *IEEE Int. Conf. ASIC*, vol. 1, 2003, pp. 564–567.

[20] M. Hamour, R. Saleh, S. Mirabbasi, and A. Ivanov, "Analog IP design flow for SoC applications," in *Proc. IEEE Int. Symp. Circuits and Systems Conf.*, 2003, pp. IV-676–IV-679.

[21] Verilog-AMS. [Online]. Available: http://www.eda.org/verilog-ams/

[22] VHDL AMS. [Online]. Available: http://www.vhdl.org/analog/

[23] Y.-C. Ju, V. Rao, and R. Saleh, "Consistency checking and optimization of macro-models," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 10, no. 8, pp. 957–967, Aug. 1991.

[24] P. Rashinkar, P. Paterson, and L. Singh, *System-on-a-Chip Verification*. Boston, MA: Kluwer, 2001.

[25] eASIC. [Online]. Available: http://www.easic.com

[26] Faraday. [Online]. Available: http://www.faraday-tech.com

[27] S. Wilton and R. Saleh, "Programmable logic IP cores in SoC design: opportunities and challenges," in *Proc. IEEE Custom Integrated Circuits Conf.*, 2001, pp. 63–66.

[28] J. Greenbaum, "Reconfigurable logic in SoC systems," in *Proc. IEEE Custom Integrated Circuits Conf.*, 2002, pp. 5–8.

[29] Stratix II device handbook, Altera Corp., May 2005, datasheet.

[30] Virtex-4 family overview, Xilinx Inc., Mar. 2005, datasheet.

[31] ProASIC3 Flash family FPGAs, Actel Corp., Jan. 2005, datasheet.

[32] S. J. E. Wilton, N. Kafafi, J. Wu, K. Bozman, V. Aken'Ova, and R. Saleh, "Design considerations for soft embedded programmable logic cores," *IEEE J. Solid-State Circuits*, vol. 40, no. 2, pp. 485–497, Feb. 2005.

[33] M. Borgatti, F. Lertora, B. Foret, and L. Cali, "A reconfigurable system featuring dynamically extensible embedded microprocessor, FPGA, and customisable I/O," *IEEE J. Solid-State Circuits*, vol. 38, no. 3, pp. 521–529, Mar. 2003.

[34] T. Vaida, "PLC advanced technology demonstrator TestChipB," in *Proc. IEEE Custom Integrated Circuits Conf.*, 2001, pp. 67–70.

[35] L. Cali, F. Lertora, C. Gazzina, M. Besana, and M. Borgatti, "Platform IC with embedded via programmable logic for fast customization," in *Proc. IEEE Custom Integrated Circuits Conf.*, 2004, pp. 419–422.

[36] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli, "Architecture of field-programmable gate arrays," *Proc. IEEE*, vol. 81, no. 7, pp. 1013–1029, Jul. 1993.

[37] G. Lemieux and D. Lewis, *Design of Interconnection Networks for Programmable Logic*. Boston, MA: Kluwer, Nov. 2003.

[38] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Boston, MA: Kluwer, 1999.

[39] J. S. Rose, R. J. Francis, D. Lewis, and P. Chow, "Architecture of field-programmable gate arrays: the effect of logic block functionality on area efficiency," *IEEE J. Solid-State Circuits*, vol. 25, no. 5, pp. 1217–1225, Oct. 1990.

[40] S. Singh, J. Rose, P. Chow, and D. Lewis, "The effect of logic block architecture on FPGA performance," *IEEE J. Solid-State Circuits*, vol. 27, no. 3, pp. 281–287, Mar. 1992.

[41] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. Taylor, "PipeRench: A reconfigurable architecture and compiler," *Computer*, vol. 33, no. 4, pp. 70–77, Apr. 2000.

[42] H. Singh, M.-H. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, and E. Chaves, "MorphoSys: An integrated reconfigurable system for data-parallel and compute intensive applications," *IEEE Trans. Comput.*, vol. 49, no. 5, pp. 465–481, May 2000.

[43] T. Todman, G. A. Constantinides, S. Wilton, O. Mencer, W. Luk, and P. Cheung, "Reconfigurable computing: Architectures and design methods," *IEE Proc. Comput. Digital Tech.*, vol. 152, no. 2, pp. 193–208, Mar. 2005.

[44] J. Rose and S. Brown, "Flexibility of interconnection structures for field-programmable gate arrays," *IEEE J. Solid-State Circuits*, vol. 26, no. 3, pp. 277–282, Mar. 1991.

[45] P. Hallschmid and S. Wilton, "Detailed routing architectures for embedded programmable logic IP cores," in *Proc.*

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.

ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays, 2001, pp. 69–74.

[46] S. Wilton, J. Rose, and Z. Vranesic, "The memory/logic interface in FPGA's with large embedded memory arrays," *IEEE Trans. Very-Large Scale Integr. (VLSI) Syst.*, vol. 7, no. 1, pp. 80–91, Mar. 1999.

[47] Y. Ran and M. Marek-Sadowska, "Designing a via-configurable regular fabric," in *Proc. IEEE Custom Integrated Circuits Conf.*, 2004, pp. 423–426.

[48] K. Y. Tong, V. Kheterapal, S. Rovner, H. Schmit, L. Pileggi, and R. Puri, "Regular logic fabrics for a via patterned gate array (VPGA)," in *Proc. IEEE Custom Integrated Circuits Conf.*, Sep. 2003, pp. 53–56.

[49] D. Lewis, *Fundamentals of Embedded Software: Where C and Assembly Meet*. Upper Saddle River, NJ: Prentice-Hall, 2002.

[50] S. Edwards, *Languages for Digital Embedded Systems*. Boston, MA: Kluwer, 2000.

[51] D. Bakker and D. Gilster, *Bluetooth End to End*. New York: Hungry Minds, 2002.

[52] SystemC. [Online]. Available: http://www.systemc.org

[53] SystemVerilog. [Online]. Available: http://www.systemverilog.org

[54] D. Shapiro, "Globally-asynchronous, locally-synchronous systems," Ph.D. Thesis, Dept. Comput. Sci., Stanford Univ., Stanford, CA, Oct. 1984, Tech. Rep. STAN-CS-84-1026.

[55] J. Mutterbach, T. Villiger, and W. Fichtner, "Practical design of globally-asynchronous, locally-synchronous systems," in *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, 2000, pp. 52–59.

[56] L. P. Carloni, K. L. McMillan, A. Saldanha, and A. Sangiovanni-Vincentelli, "A methodology for correct-by-construction latency insensitive design," in *Proc IEEE Int. Conf. Computer-Aided Design*, 1999, pp. 309–315.

[57] K. Goossens, J. Dielissen, *et al.*, "Guaranteeing the quality of services in networks on chip," in *Network on Chip*, A. Jantsch and H. Tenhunen, Eds. Boston, MA: Kluwer, 2003, ch. 4, pp. 61–82.

[58] T. Sparso, "Scheduling discipline for latency and bandwidth guarantees in asynchronous network-on-chip," in *Proc. IEEE Int. Symp. Asynchronous Circuits and Systems*, 2005, pp. 34–43.

[59] T. Chaney and C. Molnar, "Anomalous behavior of synchronizer and arbiter circuits," *IEEE Trans. Comput.*, vol. C-22, no. 4, pp. 421–422, Apr. 1973.

[60] L. Marino, "General theory of metastable operation," *IEEE Trans. Comput.*, vol. C-30, no. 2, pp. 107–115, Feb. 1981.

[61] R. Dobkin, R. Ginosar, and C. Sotiriou, "Data synchronization issues in GALS SoCs," in *Proc. IEEE Int. Symp. Asynchronous Circuits and Systems*, 2004, pp. 170–180.

[62] R. Ginosar, "Fourteen ways to fool your synchronizer," in *Proc. IEEE Int. Symp. Asynchronous Circuits and Systems*, 2003, pp. 89–96.

[63] J. Seizovic, "Pipeline synchronization," in *Proc. IEEE Int. Symp. Asynchronous Circuits and Systems*, 1994, pp. 87–96.

[64] T. Chelcea and S. Nowick, "Robust interfaces for mixed-timing systems," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 12, no. 8, pp. 857–873, Aug. 2004.

[65] A. Chakraborty and M. Greenstreet, "Efficient self-timed interfaces for crossing clock domains," in *Proc. IEEE Int. Symp.* Asynchronous Circuits and Systems, May 2003, pp. 78–88.

[66] D. Messerschmitt, "Synchronization in digital systems design," *IEEE J. Sel. Areas Commun.*, vol. 8, no. 8, pp. 1404–1419, Oct. 1990.

[67] P. Magarshack and P. G. Paulin, "System-on-chip beyond the nanometer wall," in *Proc. Design Automation Conf.*, 2003, pp. 419–424.

[68] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," Computer, vol. 35, no. 1, pp. 70–78, Jan. 2002.

[69] W. Dally and B. Towles, "Route packets, not wires: On-chip interconnect networks," in *Proc. Design Automation Conf.*, 2001, pp. 684–689.

[70] Y. Zorian, "Guest editor's introduction: what is infrastructure IP?" *IEEE Des. Test Comput.*, vol. 19, no. 3, pp. 3–5, May–Jun. 2002.

[71] A. Radulescu, J. Dielissen, S. G. Pestana, O. P. Gangwal, E. Rijpkema, P. Wielage, and K. Goossens, "An efficient on-chip NI offering guaranteed services, shared-memory abstraction, and flexible network configuration," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 1, pp. 4–17, Jan. 2005.

[72] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design trade-offs for network on chip interconnect architectures," *IEEE Trans. Comput.*, vol. 54, no. 8, pp. 1025–1040, Aug. 2005.

[73] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks—An Engineering Approach*. San Mateo, CA: Morgan Kaufmann, 2002.

[74] A. Jantsch and H. Tenhunen, Eds., Networks on Chip. Boston, MA: Kluwer, 2003.

[75] T. Ye, L. Benini, and G. De Micheli, "Packetization and routing analysis of on-chip multiprocessor networks," *J. Syst. Integr.*, vol. 50, pp. 81–104, Feb. 2004.

[76] L. Benini and D. Bertozzi, "Xpipes: A network-on-chip architecture for gigascale systems-on-chip," *IEEE Circuits Syst. Mag.*, vol. 4, no. 2, pp. 18–31, 2004.

[77] R. Saleh, "An approach that will NoC your SoCs off!" *IEEE Des. Test Comput.*, vol. 22, no. 5, p. 488, Sep.–Oct. 2005.

[78] C. Grecu, P. P. Pande, A. Ivanov, and R. Saleh, "Timing analysis of network on chip architectures for MP-SoC platforms," *Microelectron. J.*, vol. 36, no. 9, pp. 833–845, Sep. 2005.

[79] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory, & Mixed-Signal VLSI Circuits*. Boston, MA: Kluwer, 2000.

[80] K. Parker, *The Boundary-Scan Handbook: Analog and Digital*, 2nd ed. Boston, MA: Kluwer, 1998.

[81] R. Ho, K. W. Mai, and M. A. Horowitz, "The future of wires," *Proc. IEEE*, vol. 89, no. 4, pp. 490–504, Apr. 2001.

[82] R. Greenberg and L. Guan, "An improved analytical model for wormhole routed networks with application to butterfly fat-trees," in *Proc. Int. Conf. Parallel Processing*, 1997, pp. 44–48.

[83] S. Kumar *et al.*, "A network on chip architecture and design methodology," in *Proc. IEEE Symp. VLSI*, 2002, pp. 105–112.

[84] C. Leiserson, "Fat trees: Universal networks for hardware efficient supercomputing," *IEEE Trans. Comput.*, vol. 34, no. 10, pp. 892–901, Oct. 1985.

[85] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *IEEE Design, Automation and Test in Europe Conf. Exhibition*, 2000, pp. 250–256.

[86] L. Ni, Y. Gui, and S. Moore, "Performance evaluation of switch-based wormhole networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, no. 5, pp. 462–474, May 1997.

[87] L. Bening and H. Foster, *Principles of Verifiable RTL Design: A Functional Coding Style Supporting Verification Processes in Verilog*, 2nd ed. Boston, MA: Kluwer, 2001.

[88] *Accellera Property Specification Language 1.1 Reference Manual*, Accellera Org., Inc., Napa, CA, Jun. 9, 2004.

[89] *SystemVerilog 3.1a Language Reference Manual*, Accellera Org. Inc., Napa, CA, May 13, 2004.

[90] I. Beer, S. Ben-David, C. Eisner, D. Fisman, A. Gringauze, and Y. Rodeh, "The temporal logic sugar," *Computer-Aided Verification: 13th Int. Conf.*, 2001, pp. 363–367.

[91] R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M. Y. Vardi, and Y. Zbar, "The ForSpec temporal logic: A new temporal property-specification language," *Tools and Algorithms for the Construction and Analysis of Systems: Int. Conf.*, 2002, pp. 296–311.

[92] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA: MIT Press, 2002.

[93] S.-Y. Huang and K.-T. Cheng, *Formal Equivalence Checking and Design Debugging*. Boston, MA: Kluwer, 1998.

[94] B. Bentley, "High level validation of next generation microprocessors," in *Proc. IEEE Int. Workshop High-Level Design Validation and Test*, 2002, pp. 31–35.

[95] T. Schubert, "High level formal verification of next-generation microprocessors," in *Proc. ACM/IEEE Design Automation Conf.*, 2003, pp. 1–6.

[96] E. Clarke, D. Long, and K. McMillan, "Compositional model checking," in *Proc. 4th Annu. Symp. Logic in Computer Science*, 1989, pp. 353–362.

[97] A. Pnueli, "In transition from global to modular temporal reasoning about programs," in *Logics and Models of Concurrent Systems*, New York: Springer, 1989, pp. 123–144.

[98] X. Feng and A. Hu, "Cutpoints for formal equivalence verification of embedded software," in *Proc. ACM Int. Conf. Embedded Software*, 2005, pp. 307–316.

[99] Workshop Formal Verification of Analog Circuits, Edinburgh, U.K., Apr. 9, 2005, affiliated with the 2005 Eur. Joint Conf. Theory and Practice of Software.

[100] N. Mokhoff, "Trust designs, but verify, say panelists," EE Times, Jun. 17, 2005. [Online]. Available: http://www.eetimes.com/conf/dac/showArticle.jhtml?articleID=164900577

[101] G. Spirakis, "Designing for 65 nm and beyond: Where's the revolution?!? (keynote speech)," presented at the *EDP 2005 Workshop, Electronic Design Process Subcommittee of IEEE DATC*, Monterey, CA, Apr. 7–8, 2005.

[102] T. M. Austin, "DIVA: A reliable substrate for deep submicron microarchitecture design," in *Proc. 32nd IEEE/ACM Int. Symp. Microarchitecture*, 1999, pp. 16–27.

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.

[103] M. Martin, M. Hill, and D. Wood, "Token coherence: Decoupling performance and correctness," in *Proc. Int. Symp. Computer Architecture*, 2003, pp. 182–193.

[104] M. Holland and S. Hauck, "Improving performance and robustness of domain-specific CPLDs," in *ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2006, pp. 50–59.

[105] K. Compton and S. Hauck, "Track placement: Orchestrating routing structures to maximize routability," in *Int. Conf. Field-Programmable Logic and Applications*, Aug. 2003.

[106] S. Phillips and S. Hauck, "Automating the layout of reconfigurable subsystems using circuit generators," in *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, 2005, pp. 203–212.

[107] I. Kuon, A. Egier, and J. Rose, "Design, layout and verification of an FPGA using automated tools," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2005, pp. 215–226.

[108] V. Aken'Ova, R. Saleh, and G. Lemieux, "An improved soft eFPGA design and implementation strategy," in *Proc. IEEE Custom Integrated Circuits Conf.*, 2005, pp. 179–182.

[109] Soft IP Cores. [Online]. Available: http://www.us.design-reuse.com

ABOUT THE AUTHORS

Resve Saleh (Fellow, IEEE) received the B.Sc. degree in electrical engineering from Carleton University, Ottawa, ON, Canada, and the M.S. and Ph.D. degrees in electrical engineering from the University of California at Berkeley.

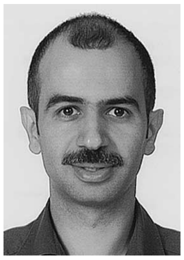
He is currently Professor and the NSERC/PMC-Sierra Chairholder in the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, Canada, working in the field of system-on-chip design, verification, and test. He was a Founder and served as Chairman from 1995 to 2000 at Simplex Solutions which developed CAD software for deep-submicrometer digital design verification. He was also VP of R&D from 1995 to 1999 for the extraction and analysis product lines. Prior to starting Simplex, he spent nine years as a Professor in the Department of Electrical and Computer Engineering, University of Illinois in Urbana, and one year teaching at Stanford University. Previous to his academic career, he worked for Mitel Corporation, Ottawa, Toshiba Corporation, Japan, Tektronix, Beaverton, OR, and Nortel, Ottawa. He has published two books on mixed-mode simulation and over 80 papers in conferences and journals.

Dr. Saleh received the National Science Foundation Presidential Young Investigator Award in 1990. He has served as General Chair (1995), Conference Chair (1994), and Technical Program Chair (1993) for the Custom Integrated Circuits Conference, and recently held the positions of Technical Program Chair, Conference Chair, and Vice-General Chair of the International Symposium on Quality in Electronic Design (2001). He is Cofounder and Chair of the Vancouver Chapter of the Solid-State Circuits Society. He is a Professional Engineer of British Columbia and currently consults for a number of startup companies in San Jose, CA. He has served as Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN. From 1992 to 1995, he also held the position of chairman of the IEEE Standards Coordinating Committee 30—Analog Hardware Description Languages (AHDL). He was the Technical Program Chair of the International Symposium on Quality in Electronic Design 2002.

Steve Wilton (Senior Member, IEEE) received the M.A.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Toronto, Toronto, ON, Canada, in 1992 and 1997, respectively.

In 1997, he joined the Department of Electrical and Computer Engineering at the University of British Columbia, Canada, where he is now an Associate Professor. During 2003 and 2004, he was a Visiting Professor in the Department of Computing at Imperial College, London, U.K., and at the Interuniversity MicroElectronics Center (IMEC), Leuven, Belgium. He has also served as a consultant for Cypress Semiconductor and Altera Corporation. His research focuses on the architecture of FPGAs, and the CAD tools that target these devices.

In 2005, Dr. Wilton was the Program Chair for the ACM International Symposium on Field-Programmable Gate Arrays and the Program Cochair for the International Conference on Field-Programmable Logic and Applications. He is also a member of the program committee for the IEEE Custom Integrated Circuits Conference, the International Conference on Field-Programmable Logic and Applications, and the International Conference on Field-Programmable Technology, and has served as a Guest Editor for two issues of the IEEE JOURNAL OF SOLID-STATE CIRCUITS. In 1998, he won the Douglas Colton Medal for Research Excellence for his research into FPGA memory architectures. He received best paper awards at the International Conference on Field-Programmable Technology in 2003 and 2005, and at the International Conference on Field-Programmable Logic and Applications in 2001 and 2004.



Shahriar Mirabbasi (Member, IEEE) received the B.Sc. degree in electrical engineering from Sharif University of Technology, Tehran, Iran, in 1990, and the M.A.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Toronto, Toronto, ON, Canada, in 1997 and 2002, respectively.

During the summer of 1997, he was with Gennum Corporation, Burlington, ON, Canada, working on the system design of cable equalizers for serial digital video and HDTV applications. From January 2001 to June 2002, he was with Snowbush Microelectronics, Toronto, ON, Canada, as a Designer, where he worked on high-speed mixed-signal CMOS integrated circuits including ADC and serializer/deserializer blocks. Since August 2002, he has been an Assistant Professor in the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, Canada. His current research interests include analog and mixed-signal integrated circuits and systems design for high-speed wireless and wireline data communications applications, wireless sensor networks, and biomedical implants.

Alan Hu (Member, IEEE) received the B.S. and Ph.D. degrees from Stanford University, Stanford, CA, in 1989 and 1996, respectively.

He is an Associate Professor in the Department of Computer Science at the University of British Columbia, Vancouver, BC, Canada. For the past 15 years, his main research focus has been automated, practical techniques for formal verification.

Dr. Hu has served on the program committees of most major CAD and formal verification conferences, and chaired or cochaired CAV (1998), HLDVT (2003), and FMCAD (2004). He was also a Technical Working Group Key Contributor on the 2001 International Technology Roadmap for Semiconductors, and is a member of the Technical Advisory Board of Jasper Design Automation.

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.

Mark Greenstreet (Member, IEEE) received the B.Sc. degree in electrical engineering from Caltech (1981), and the M.A. and Ph.D. degrees in computer science from Princeton University, Princeton, NJ, in 1988 and 1993, respectively.

He is a Professor in the Department of Computer Science at the University of British Columbia, Vancouver, BC, Canada, where he has been teaching and directing research since 1992. He has active research collaborations with Intel and SUN Microsystems, and consults regularly at SUN. His research interests include asynchronous design, high-performance interconnect, formal verification, and hybrid and dynamical systems.

Guy Lemieux (Member, IEEE) received the B.A.Sc. degree from the division of engineering science and the M.A.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Toronto, Toronto, ON, Canada.

In 2003, he joined the Department of Electrical and Computer Engineering at the University of British Columbia, Canada, where he is now an Assistant Professor. His research interests include computer-aided design algorithms, VLSI and SoC circuit design, FPGA architectures, and parallel computing. His specialization is in interconnection network design and routing algorithms. He is coauthor of the book Design of Interconnection Networks for Programmable Logic.

Dr. Lemieux received the Best Paper award at the IEEE International Conference on Field-Programmable Technology in 2004. He is a member of the technical program committee for the ACM/SIGDA International Symposium on FPGAs, IEEE International Conference on Field-Programmable Technology, International Conference on Field-Programmable Logic and Applications, and the ACM/IEEE Design Automation Conference Ph.D. Forum.

Partha Pratim Pande (Member, IEEE) received the B.S. in electronics and communication engineering from Calcutta University, the M.S. degree in computer science from the National University of Singapore, and the Ph.D. degree in electrical and computer engineering from the University of British Columbia.

He is an Assistant Professor in the School of Electrical Engineering and Computer Science at Washington State University, Pullman, WA. His research interests focus on design and test of network-on-chip (NoC) architectures, and robust and fault-tolerant multiprocessor SoC (MP-SoC) platforms.

Cristian Grecu (Student Member, IEEE) received the B.Eng. and M.Eng. degrees in electrical and computer engineering from the Technical University of Iasi, Romania, in 1996 and 1997, respectively, and the M.A.Sc. degree from the University of British Columbia, Vancouver, BC, Canada, in 2003. He is currently working toward the Ph.D. degree in the System-on-Chip (SoC) Research Laboratory, Department of Electrical and Computer Engineering, University of British Columbia.

His research interests include design and test of SoCs, fault-tolerant on-chip communication infrastructures, and reliability issues in VLSI systems.

Andre Ivanov (Fellow, IEEE) received the B.Eng. (Hon.), M.Eng., and Ph.D. degrees in electrical engineering from McGill University, Montreal, QC, Canada.

He is a Professor in the Department of Electrical and Computer Engineering at the University of British Columbia, Vancouver, BC, Canada. In 1995–1996, he spent a sabbatical leave at PMC-Sierra, Vancouver. He has held Invited Professor positions at the University of Montpellier II, the University of Bordeaux I, and Edith Cowan University, in Perth, Australia. In 2001, he cofounded Vector 12, a semiconductor IP company. He has published over 100 papers in conference and journals and holds four U.S. patents. He serves on the Editorial Board of Kluwer's *Journal of Electronic Testing: Theory and Applications*. His primary research interests lie in the area of integrated circuit testing, design for testability and built-in self-test, for digital, analog and mixed-signal circuits, and systems-on-chip (SoCs). He has published widely in these areas and holds several patents in IC design and test. Besides testing, Ivanov has interests in the design methodologies of large and complex integrated circuits and SoCs.

Dr. Ivanov has served and continues to serve on numerous national and international steering, program, and/or organization committees in various capacities. Recently, he was the Program Chair of the 2002 VLSI Test Symposium (VTS 02) and the General Chair for VTS 03 and VTS 04. In 2004, he founded and cochaired the 1st IEEE International GHz/Gbps Test Workshop. He serves on the Editorial Board of the *IEEE Design and Test Magazine*. He is currently the Chair of the IEEE Computer Society's Test Technology Technical Council (TTTC) and Vice Chair of the IEEE Computer Society Conference and Tutorials Board. He is a Golden Core Member of the IEEE Computer Society, a Fellow of the British Columbia Advanced Systems Institute, and a Professional Engineer of British Columbia.

Authorized licensed use limited to: Washington State University. Downloaded on June 16,2010 at 22:22:10 UTC from IEEE Xplore. Restrictions apply.