Multiprocessor System-on-Chip (MPSoC) Technology

Wayne Wolf, *Fellow, IEEE*, Ahmed Amine Jerraya, and Grant Martin, *Senior Member, IEEE*

***Abstract*—The multiprocessor system-on-chip (MPSoC) uses multiple CPUs along with other hardware subsystems to implement a system. A wide range of MPSoC architectures have been developed over the past decade. This paper surveys the history of MPSoCs to argue that they represent an important and distinct category of computer architecture. We consider some of the technological trends that have driven the design of MPSoCs. We also survey computer-aided design problems relevant to the design of MPSoCs.**

***Index Terms*—Configurable processors, encoding, hardware/software codesign, multiprocessor, multiprocessor system-on-chip (MPSoC).**

I. Introduction

MULTIPROCESSOR systems-on-chips (MPSoCs) have emerged in the past decade as an important class of very large scale integration (VLSI) systems. An MPSoC is a system-on-chip—a VLSI system that incorporates most or all the components necessary for an application—that uses multiple programmable processors as system components. MPSoCs are widely used in networking, communications, signal processing, and multimedia among other applications.

We will argue in this paper that MPSoCs constitute a unique branch of evolution in computer architecture, particularly multiprocessors, that is justified by the requirements on these systems: real-time, low-power, and multitasking applications. We will do so by presenting a short history of the MPSoCs as well as an analysis of the driving forces that motivate the design of these systems. We will also use this opportunity to outline some important computer-aided design (CAD) problems related to MPSoCs and describe previous works on those problems.

In an earlier paper [1], we presented some initial arguments as to why MPSoCs existed—why new architectures were needed for applications like embedded multimedia and cell phones. In this paper, we hope to show more decisively that MPSoCs form a unique area of multiprocessor design that is distinct from multicore processors for some very sound technical reasons.

We will start with our discussion of the history of MPSoCs and their relationship to the broader history of multiprocessors in Section II. In Section III, we will look at several characteristics of embedded applications that drive us away from traditional scientific multiprocessing architectures. Based on that analysis, Section IV more broadly justifies the pursuit of a range of multiprocessor architectures, both heterogeneous and homogeneous, to satisfy the demands of high-performance embedded computing applications. Section V describes some important CAD problems related to MPSoCs. We close in Section VI with some observations on the future of MPSoCs.

II. Multiprocessors and the Evolution of MPSoCs

MPSoCs embody an important and distinct branch of multiprocessors. They are not simply traditional multiprocessors shrunk to a single chip but have been designed to fulfill the unique requirements of embedded applications. MPSoCs have been in production for much longer than multicore processors. We argue in this section that MPSoCs form two important and distinct branches in the taxonomy of multiprocessors: homogeneous and heterogeneous multiprocessors. The importance and historical independence of these lines of multiprocessor development are not always appreciated in the microprocessor community.

To advance this theme, we will first touch upon the history of multiprocessor design. We will then introduce a series of MPSoCs that illustrate the development of this category of computer architecture. We next touch upon multicore processors. We conclude this section with some historical analysis.

A. Early Multiprocessors

Multiprocessors have a very long history in computer architecture; we present only a few highlights here. One early multiprocessor was Illiac IV [2], which was widely regarded as the first supercomputer. This machine had 64 arithmetic logic units (ALUs) controlled by four control units. The control units could also perform scalar operations in parallel with the vector ALUs. It performed vector and array operations in parallel. Wulf and Harbison [3] describe C.mmp as an early multiprocessor that connected 16 processors to memory through a crossbar.

Culler *et al.* [4] describe the state-of-the-art in parallel computing at the end of the 1990s. They identified Moore's Law as a major influence on the development of multiprocessor architectures. They argue that the field has converged on a

Manuscript received June 11, 2007; revised November 20, 2007. Current version published September 19, 2008. The work of W. Wolf was supported in part by the National Science Foundation under Grant 0324869. This paper was recommended by Associate Editor V. Narayanan.

W. Wolf is with the Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: wolf@ece.gatech.edu).

A. A. Jerraya is with CEA-LETI, 38054 Grenoble Cedex 9, France (e-mail: ahmed.jerraya@cea.fr).

G. Martin is with Tensilica, Inc., Santa Clara, CA 95054 USA (e-mail: gmartin@tensilica.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2008.923415

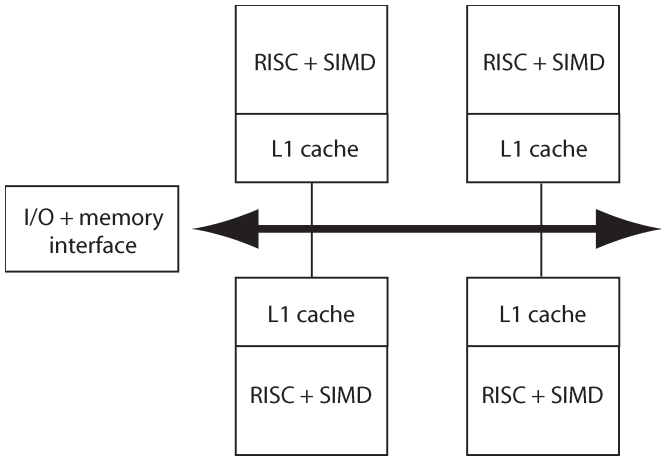


Fig. 1. Lucent Daytona MPSoC.

generic model of multiprocessors in which a collection of computers, each potentially including both CPU and memory, communicate over an interconnection network. This model is particularly aimed at unifying the shared memory and message passing models for multiprocessors. Chapter 2 of their book describes parallel programs. They discuss four applications that they consider representative: ocean current simulation, simulating the evolution of galaxies, ray tracing for computer graphics, and data mining.

Almasi and Gottlieb provide a definition of highly parallel processors that they believe covers many such designs: "A large collection of processing elements that communicate and cooperate to solve large problems fast" [5, p. 5]. They see three motivations for the design of parallel processors: solving very large problems such as weather modeling, designing systems that satisfy limited budgets, and improving programmer productivity. They identify several types of problems that are suitable for parallel processors: easy parallelism such as payroll processing (today, we would put Web service in this category), scientific and engineering calculations, VLSI design automation, database operations, and near- and long-term artificial intelligence.

B. History of MPSoCs

In this section, we will survey the history of MPSoCs. Given the large number of chip design groups active around the world, we do not claim that the list of MPSoCs given here is complete. However, we have tried to include many representative processors that illustrate the types of MPSoCs that have been designed and the historical trends in the development of MPSoCs.

The 1990s saw the development of several VLSI uniprocessors created for embedded applications like multimedia and communications. Several single-chip very long instruction word (VLIW) processors were developed to provide higher levels of parallelism along with programmability. Another common approach was the application-specific IC, which stitched together a number of blocks, most of which were not general-purpose computers. The architecture of these systems often closely corresponded to the block diagrams of the applications for which they were designed.

The first MPSoC that we know of is the Lucent Daytona [6], shown in Fig. 1. Daytona was designed for wireless base stations, in which identical signal processing is performed on a number of data channels. Daytona is a symmetric architecture with four CPUs attached to a high-speed bus. The CPU architecture is based on SPARC V8 with some enhancements: 16×32 multiplication, division step, touch instruction, and vector coprocessor. Each CPU has an 8-KB 16-bank cache. Each bank can be configured as instruction cache, data cache, or scratch pad. The Daytona bus is a split-transaction bus. The processors share a common address space in memory; the caches snoop to ensure cache consistency. The chip was 200 mm^2 and ran at 100 MHz at 3.3 V in a 0.25-μm CMOS process.

Shortly after the Daytona appeared, several other MPSoCs were announced. These processors were designed to support a range of applications and exhibit a wide range of architectural styles.

The C-5 Network Processor [7] was designed for another important class of embedded applications, which is packet processing in networks. The architecture of the C-5 is shown in Fig. 2. Packets are handled by channel processors that are grouped into four clusters of four units each. Three buses handle different types of traffic in the processor. The C-5 uses several additional processors, some of which are very specialized, whereas the executive processor is a reduced instruction set computing (RISC) CPU.

A third important class of MPSoC applications is multimedia processing. An early example of a multimedia processor is the Philips Viper Nexperia [8], shown in Fig. 3. The Viper includes two CPUs: a MIPS and a Trimedia VLIW processor. The MIPS acted as a master running the operating system, whereas the Trimedia acted as a slave that carried out commands from the MIPS. The system includes three buses, one for each CPU and one for the external memory interface. Bridges connect the buses. However, the multiprocessing picture is more complicated because some hardware accelerators are attached to the buses. These accelerators perform computations such as color space conversion, scaling, etc. The Viper could implement a number of different mappings of physical memory to address spaces.

A fourth important class of MPSoC applications is the cell phone processor. Early cell phone processors performed baseband operations, including both communication and multimedia operations. The Texas Instruments (TI) OMAP architecture [9] has several implementations. The OMAP 5912, as shown in Fig. 4, has two CPUs: an ARM9 and a TMS320C55x digital signal processor (DSP). The ARM acts as a master, and the DSP acts as a slave that performs signal processing operations.

The STMicroelectronics Nomadik [10], shown in Fig. 5, is another MPSoC for cell phones. It uses an ARM926EJ as its host processor. At this level, the architecture appears to be a fairly standard bus-based design. However, two of the units on the bus are programmable accelerators, one for audio and another for video, which make use of the MMDSP+, which is a small DSP. The architectures of the video and audio accelerators are shown in Figs. 6 and 7, respectively. The video accelerator is a heterogeneous multiprocessor, including an MMDSP+ and hardwired units for several important stages

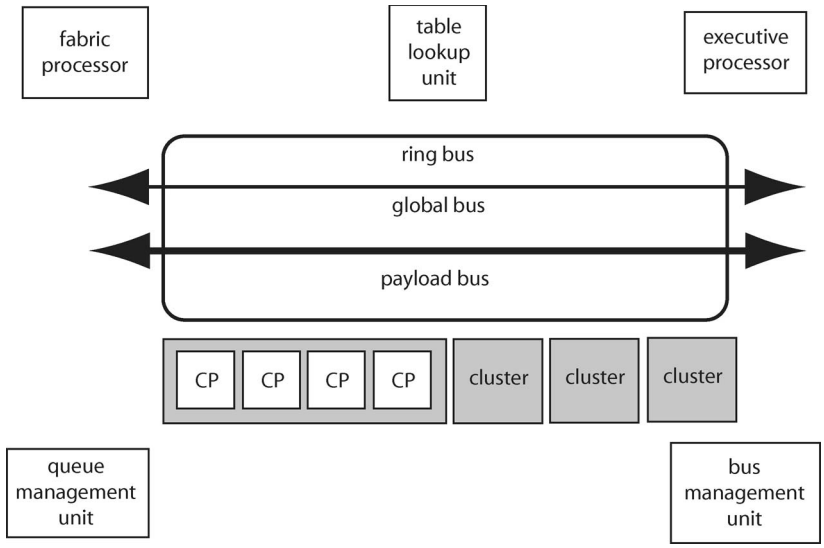


Fig. 2. C-5 network processor.

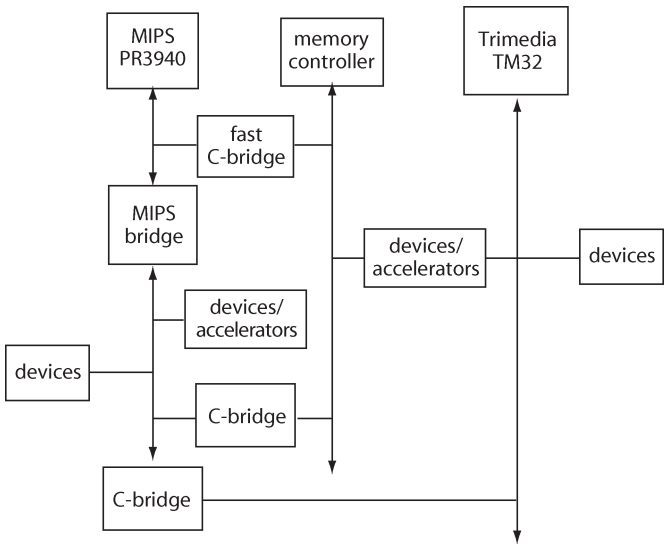


Fig. 3. Viper Nexperia processor.

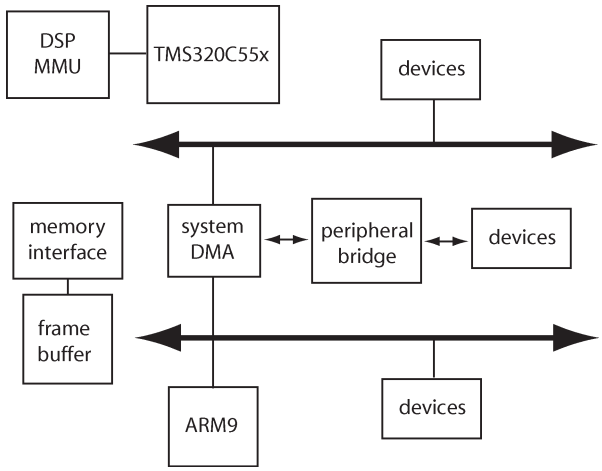


Fig. 4. TI OMAP 5912.

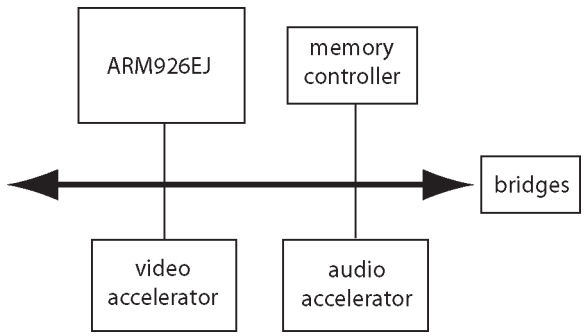


Fig. 5. ST Nomadik SA.

in video processing. The audio processor relies more heavily on the MMDSP+, thanks to the lower throughput requirements of audio.

As shown in Fig. 8, the ARM MPCore [11] is a homogeneous multiprocessor that also allows some heterogeneous configurations. The architecture can accommodate up to four CPUs. Some degree of irregularity is afforded by the memory controller, which can be configured to offer varying degrees of access to different parts of memory for different CPUs. For example, one CPU may be able only to read one part of the memory space, whereas another part of the memory space may not be accessible to some CPUs.

The Intel IXP2855 [12], shown in Fig. 9, is a network processor. Sixteen microengines are organized into two clusters to process packets. An XScale CPU serves as host processor. Two cryptography accelerators perform cryptography functions.

The Cell processor [13] has a PowerPC host and a set of eight processing elements known as synergistic processing elements. The processing elements, PowerPC, and I/O interfaces are connected by the element interconnect bus, which is built from four 16-B-wide rings. Two rings run clockwise, and the other two run counterclockwise. Each ring can handle up to three nonoverlapping data transfers at a time.

The Freescale MC8126 was designed for mobile basestation processing. It has four Starcore SC140 VLIW processors along with shared memory.

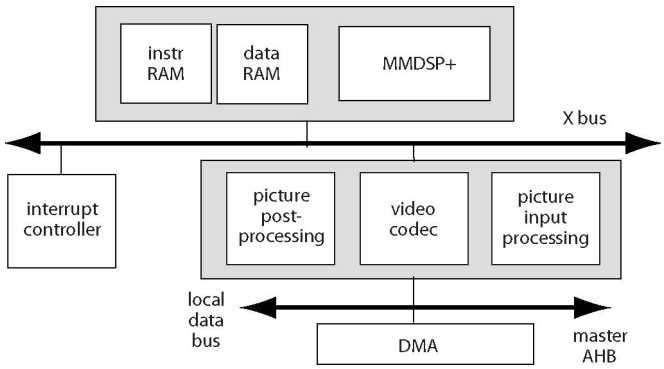


Fig. 6. ST Nomadik video accelerator.

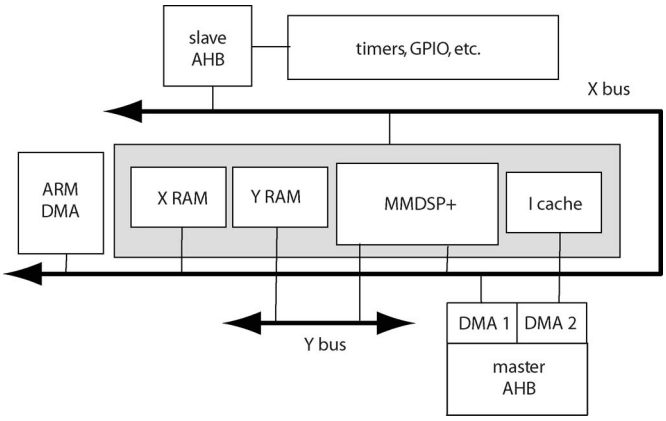


Fig. 7. ST Nomadik audio accelerator.

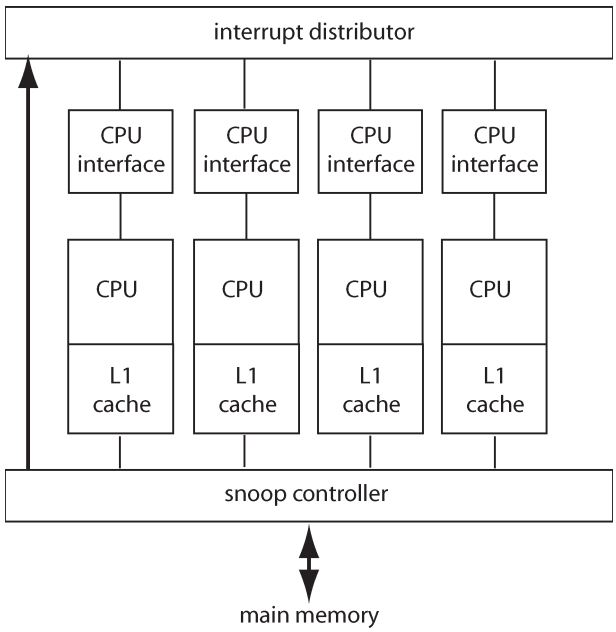


Fig. 8. ARM MPCore.

Several multiprocessor architectures are being developed for cell phones. One example is the Sandbridge Sandblaster processor [14], [15]. It can process four threads with full interlocking for resource constraints. It also uses an ARM as a host processor. The Infineon Music processor is another example of a cell phone processor. It has an ARM9 host and a set of single-instruction multiple-data (SIMD) processors.

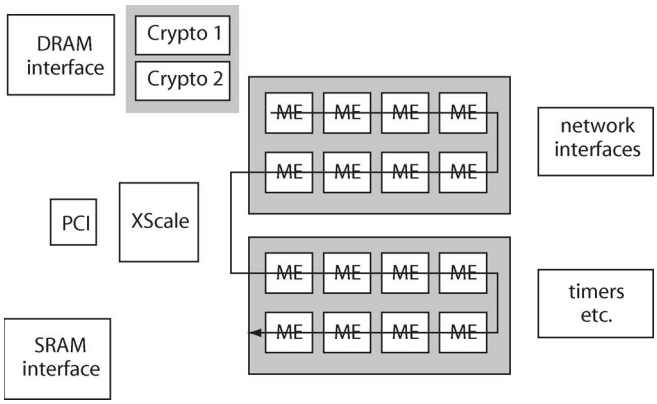


Fig. 9. Intel IXP2855.

The Cisco Silicon Packet Processor [16] is an MPSoC used in its high-end CRS-1 router. Each chip includes 192 configured extended Tensilica Xtensa cores. Multiple chips are placed on a board, multiple boards in a rack, and multiple racks in a router, so that the final product may have up to 400 000 processors in it, which are dedicated to network packet routing. This is an interesting MPSoC example in that it incorporates some design-for-manufacturability considerations in its architecture; the architecture requires 188 working processors on a die, and the extra four processors are added and dynamically used in order to increase the yield of the SoC.

A Seiko-Epson inkjet printer "Realoid" SoC [17] incorporates seven heterogeneous processors, including a NEC V850 control processor and six Tensilica Xtensa LX processors, each of them differently configured with different instruction extensions to handle different parts of the printing image processing task.

An increasing number of platforms include field programmable gate array (FPGA) fabrics: Xilinx, Altera, and Actel all manufacture devices that provide programmable processors and FPGA fabrics. In some cases, the CPUs are separately implemented as custom silicon from the FPGA fabric. In other cases, the CPU is implemented as a core within the FPGA. These two approaches are not mutually exclusive. FPGA platforms allow designers to use hardware/software codesign methodologies.

C. MPSoCs versus Multicore Processors

Hammond *et al.* [18] proposed a chip multiprocessor (CMP) architecture before the appearance of the Lucent Daytona. Their proposed machine included eight CPU cores, each with its own first-level cache and a shared second-level cache. They used architectural simulation methods to compare their CMP architecture to simultaneous multithreading and superscalar. They found that the CMP provided higher performance on most of their benchmarks and argued that the relative simplicity of CMP hardware made it more attractive for VLSI implementation.

Commercial multicore processors have a much shorter history than do commercial MPSoCs. The first multicore general-purpose processors were introduced in 2005. The Intel Core

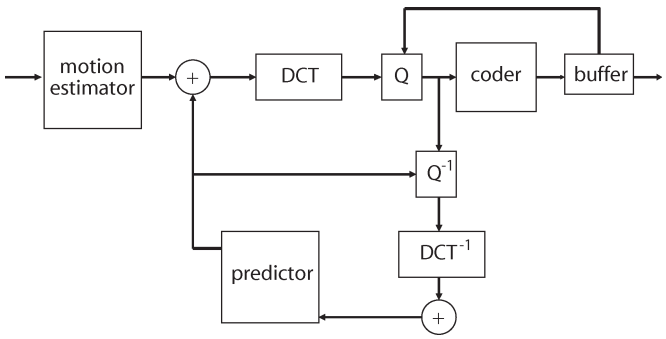


Fig. 10. Block diagram of an MPEG-2 encoder.

Duo processor [19] combines two enhanced Pentium M cores on a single die. The processors are relatively separate but share an L2 cache as well as power management logic. The AMD Opteron dual-core multiprocessor [20] provides separate L2 caches for its processors but a common system request interface connects the processors to the memory controller and HyperTransport. The Niagra SPARC processor [21] has eight symmetrical four-way multithreaded cores.

D. Analysis

Let us now try to fit MPSoCs into the broader history of multiprocessing. MPSoCs generally conform to today's standard multiprocessor model of CPUs and memories attached to an interconnection network. However, that model was generally conceived for fairly homogeneous multiprocessors.

As we have seen, many MPSoCs are very heterogeneous. We believe that MPSoCs provide at least two branches in the taxonomy of multiprocessors: the homogeneous model pioneered by the Lucent Daytona and the heterogeneous model pioneered by the C5, Nexperia, and OMAP. We would argue that these two branches are distinct from the multicore systems designed for general-purpose computers. The CPUs in multicore processors have fairly independent hardware units and provide a fairly independent programming model.

III. How Applications Influence Architecture

In this section, we will consider how the applications for which MPSoCs are designed influence both the design process and the final architecture. We will study three influences: complex applications, standards-based design, and platform-based design.

A. Complex Applications

Many of the applications to which MPSoCs are applied are not single algorithms but systems of multiple algorithms. This means that the nature of the computations done in different parts of the application can vary widely: types of operations, memory bandwidth and access patterns, activity profiles, etc. Such variations argue for heterogeneous architectures.

As an example, consider the structure of an MPEG-2 encoder [22], as shown in Fig. 10. The two most computationally intensive tasks are motion estimation and discrete cosine transform (DCT) computation, which have very different computational profiles. Motion estimation uses 2-D correlation with fairly simple operations performed on a large number of data elements; efficient motion estimation algorithms are also highly data dependent. DCT computation performs a large number of multiplications and additions but with well-established and regular data access patterns. Other tasks, such as variable length coding, operate on much lower volumes of data than either motion estimation or DCT. As a result, the memory bandwidth requirements of the encoder vary widely across the block diagram. Xu *et al.* [23] studied the operational characteristics of the H.264 reference video decoder and described some of those effects in more detail.

B. Standard-Based Design

MPSoCs exist largely because of standards—not standards for the chips themselves but for the applications to which the MPSoCs will be put. Standards provide large markets that can justify the cost of chip design, which runs into the tens of millions of U.S. dollars. Standards also set aggressive goals that can be satisfied only by highly integrated implementations.

When crafting a standard, most committees try to balance standardization and flexibilities. One common technique is to define the bit stream generated or consumed but not how that bit stream is generated. This allows implementers to develop algorithms that conform to the bit stream but provide some competitive advantage such as higher quality or lower power consumption. Another common technique for balancing standards with competition is to exclude some aspects of the final product from the standard. User interfaces, for example, are generally not included in standards.

Standards that allow for multiple implementations encourage the design of MPSoCs. Multiple products with different characteristics can occupy the design space for the standard.

However, the presence of standards is not an unalloyed blessing for MPSoC designers. Standards committees often provide reference implementations, which are executable programs that perform the operations specified by the standard. Whereas these reference implementations may seem to be a boon to system designers, they are generally not directly useful as implementations. Reference implementations are generally very single threaded. Breaking these programs into separate threads for multiprocessor implementation requires considerable effort. Improving the cache behavior of the threads requires more effort. MPSoC design teams generally spend several person-months refining the reference implementation for use.

Second, standards have grown in complexity. The reference implementation for H.264, for example, consists of over 120 000 lines of C code.

Complex reference implementations compel system implementations that include a significant amount of software. This software generally must conform to the low power requirements of the system; it makes no sense to operate at ultralow power levels in the hardware units while the software burns excessive energy. Depending on the system architecture (SA), some of the software modules may need to operate under strict real-time requirements as well.

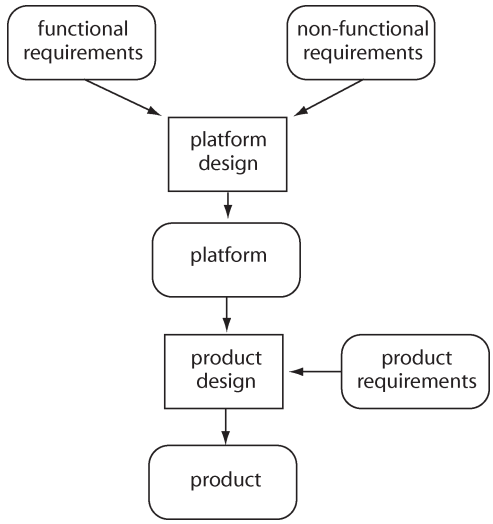


Fig. 11. Two-stage platform-based design methodology.

C. Platform-Based Design

Platform-based design divides system design into two phases. First, a platform is designed for a class of applications. Later, the platform is adapted for a particular product in that application space. Platform-based design leverages semiconductor manufacturing; a platform can be manufactured in the large volumes that are required to make chip manufacturing economically viable; the platform can then be specialized for use in a number of products, each of which is sold in smaller volumes. Standard-based design encourages platform-based design: The standard creates a large market with common characteristics as well as the need for product designers to differentiate their products within the scope of the standard.

MPSoCs are ideally suited to be used as platforms. CPUs can be used to customize systems in a variety of ways. Moreover, because MPSoCs are not limited to regular architectures, they can be designed to more aggressive specifications than can be met by general-purpose systems.

Fig. 11 shows the methodology used to design systems using platforms. Here, we distinguish between the platform, which is used in a number of different systems, and the product, which is one of those end-use systems. The platform design must take into account both functional and nonfunctional requirements on the system. Those requirements may be expressed in a more general form than would be typical in a nonplatform design. A reference implementation may provide detailed functional requirements for part of the system but no guidance about other parts of the system; for example, most reference implementations do not specify the user interface.

Product design tends to be software driven. In a very few cases, platform vendors may allow a customer to modify the platform in ways that require new sets of masks, but this negates many of the benefits of platform-based design. Much of the product customization comes from software. As a result, the quality and capabilities of the software development environment for the platform in a large part determines the usefulness of the platform. Today's software development environments (SDEs) do not provide good support for heterogeneous multiprocessors.

IV. Architectures for Real-Time Low-Power Systems

Based upon our survey of MPSoC architectures and applications, we are now ready to confront two important questions. First, why have so many MPSoCs been designed—why not use general-purpose architectures to implement these systems? Second, why have so many different styles of MPSoC architecture appeared? In particular, why design heterogeneous architectures that are hard to program when software plays such a key role in system design using MPSoCs?

The answer to the first question comes from the nature of the applications to be implemented. Applications like multimedia and high-speed data communication not only require high levels of performance but also require implementations to meet strict quantitative goals. The term "high-performance computing" is traditionally used to describe applications like scientific computing that require large volumes of computation but do not set out strict goals about how long those computations should take. Embedded computing, in contrast, implies real-time performance. In real-time systems, if the computation is not done by a certain deadline, the system fails. If the computation is done early, the system may not benefit (and in some pathological schedules, finishing one task early may cause another task to be unacceptably delayed).

High-performance embedded computing is a very different problem from high-performance scientific computing. Furthermore, these high-performance systems must often operate within strict power and cost budgets. As a result, MPSoC designers have repeatedly concluded that business-as-usual is not sufficient to build platforms for high-performance embedded applications.

This argument also helps to explain why so many radically different MPSoC architectures have been designed. Because we have strict, quantitative goals, we can design more carefully to meet them. The differing nature of the computations done in different applications drives designers to different parts of the design space.

A. Performance and Power Efficiency

To consider these driving forces in more detail, we will first start with the power/performance budget. Fig. 12 shows a plot of power consumption trends that was taken from a paper by Austin *et al.* [24]. Their article looked at the performance and power requirements of next-generation mobile systems. They estimated the performance and power requirements of an advanced set of applications that might appear on mobile systems such as cell phones in the near future. This benchmark set included high-performance data networking, voice recognition, video compression/decompression, and other applications. They estimated that this benchmark set would require a 10 000 SpecInt processor. They then showed that publicly available information estimated that general-purpose processors in this class would consume several hundred watts of power. Because batteries will only be able to produce 75 mW in that time frame, general-purpose processors consume several orders of magnitude more power than that available to support this mobile application set.

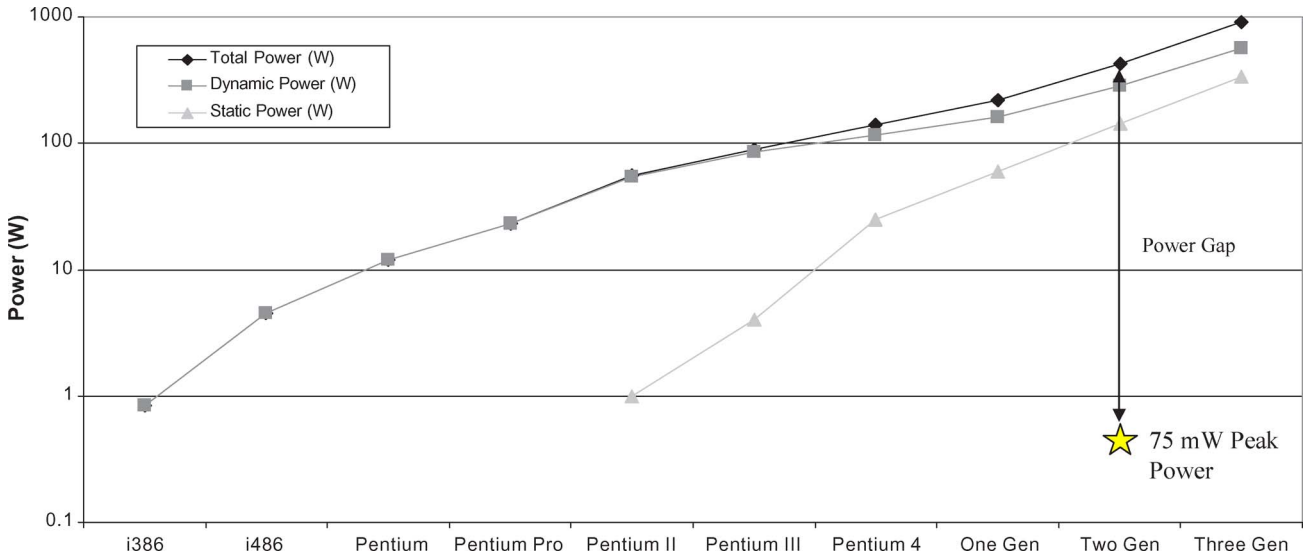


Fig. 12. Power consumption trends for desktop processors from Austin *et al.* [Aus04] 2004 IEEE Computer Society.

An MPSoC can save energy in many ways and at all levels of abstraction. Clearly, device, circuit, and logic-level techniques [25] are equally applicable to MPSoCs as to other VLSI systems. Irregular memory systems save energy because multiported RAMs burn more energy; eliminating ports in parts of the memory where they are not needed saves energy. Similarly, irregular interconnection networks save power by reducing the loads that must be driven in the network. Using different instruction sets for different processors can make each CPU more efficient for the tasks it is required to execute.

B. Real-Time Performance

Another important motivation to design new MPSoC architectures is real-time performance. Heterogeneous architectures, although they are harder to program, can provide improved real-time behavior by reducing conflicts among processing elements and tasks. For example, consider a shared memory multiprocessor in which all CPUs have access to all parts of memory. The hardware cannot directly limit accesses to a particular memory location; therefore, noncritical accesses from one processor may conflict with critical accesses from another. Software methods can be used to find and eliminate such conflicts but only at noticeable cost. Furthermore, access to any memory location in a block may be sufficient to disrupt real-time access to a few specialized locations in that block. However, if that memory block is addressable only by certain processors, then programmers can much more easily determine what tasks are accessing the locations to ensure proper real-time responses.

C. Application Structure

The structure of the applications for which MPSoCs are designed is also important. Different applications have different data flows that suggest multiple different types of architectures; some of which are quite different than the multiprocessors designed for scientific computing. The homogeneous architectural style is generally used for data-parallel systems. Wireless base stations, in which the same algorithm is applied to several independent data streams, is one example; motion estimation, in which different parts of the image can be treated separately, is another.

Heterogeneous architectures are designed for heterogeneous applications with complex block diagrams that incorporate multiple algorithms, often using producer–consumer data transfers. A complete video compression system is one important example of a heterogeneous application. The scientific computing community, in contrast, was mainly concerned with single programs that were too large to run in reasonable amounts of time on a single CPU, as evidenced by Almasi and Gottlieb's statement cited previously that multiprocessors were designed to "solve large problems fast."

V. CAD Challenges in MPSoCs

MPSoCs pose a number of design problems that are amenable to CAD techniques. General-purpose computer architects [26] perform extensive experiments but then go on to apply intuition to solve specific design problems; they have no choice but to do so because they have eschewed application-specific optimizations. Embedded computing, in contrast, is all about tuning the hardware and software architecture of a system to meet the specific requirements of a particular set of applications while providing necessary levels of programmability. Because embedded system designers have specific goals and a better-defined set of benchmarks, they can apply optimization algorithms to solve many design problems.

Many embedded system design and optimization problems can be approached from either the hardware or software perspective. This hardware/software duality allows the same basic design challenge to be solved in very different ways.

In this section, we will survey previous works in several problem areas that we consider important CAD problems for MPSoCs: configurable processors and instruction set synthesis, encoding, interconnect-driven design, core- and platform-based

designs, memory system optimizations, hardware/software codesign, and SDEs.

A. Configurable Processors and Instruction Set Synthesis

Instruction sets that are designed for a particular application are used in many embedded systems [27]. The design of customized processors entails a substantial amount of work but can result in substantially reduced power consumption and smaller area. CPU configuration refers to tools that generate a hardware description language (HDL) of a processor based on a set of microarchitectural requirements given by a user. Configurable processors usually fall into two categories: those that are based on a preexisting processor architecture, usually implementing a RISC instruction set, and those that create a complete new instruction set architecture during processor generation as specified by the user. Configurable processors utilize an automated processor generation process and toolset, which is driven by a specification that may be based on a more formalized architectural definition language or may be based on parameter selection and structural choices given in a processor configuration tool.

Processor configuration is usually of two types.

1) Coarse-grained structural configuration. This includes the presence or absence of a variety of interfaces to associated objects. These might include system bus interfaces, local memory interfaces (instruction, data and unified memories, both RAM and ROM, and in one or more instantiations), direct first-in first-out queues that can be used to interconnect processors or connect processors to hardware accelerating blocks and peripherals, direct IO ports for the same purpose, coprocessor interfaces, and extra load-store units. The interface widths and specific protocols may also be configurable or selectable. Other parameterized structural choices may include special instruction inclusion (e.g., multipliers, dividers, multiply-accumulate units, and shifters) or the inclusion of various subsets (such as DSP instruction sets, vectorizing or SIMD instructions, and floating point units), the locations and nature of reset vectors, and the numbers, kinds, levels, and priorities of interrupts, and allowing multioperation instructions and encodings, all fall into this structural configuration category. Additional structural parameters may include the degree of pipelining, on-chip debug, trace and JTAG, register file size, timers, and exception vectors.
2) Fine-grained instruction extensions. This adds extra instructions directly into the datapath of the processor that are decoded in the standard way and may even be recognized by the compiler automatically or least as manually invoked instrinsics. The instruction extensions are usually offered in some kind of compiled architectural description language (e.g., nML or LISA) or may be defined through a combination of HDL code together with templates for instruction formats, encoding, and semantics.

Architectural optimization refers to the design or refinement of microarchitectural features from higher level specifications such as performance or power. This is often in conjunction with configurable extensible processors or coprocessors. The optimization flow is usually an automated one, which is supported by tools of various levels of sophistication.

The MIMOLA system [28] was an early CPU design tool that performed both architectural optimization and configuration. ASIP Meister [29] is a processor configuration system that generates Harvard architecture machines. LISA [30] uses the LISA language to describe processors as a combination of structural and behavioral features. LISA was used as the basis for a commercial company, LISATek, which was acquired by CoWare, and the technology continues to be offered on a commercial basis.

The Tensilica Xtensa processor [31] is a commercial configurable processor that allows users to configure a wide range of processor characteristics, including instruction sets, caches, and I/O. The Toshiba MeP core is a configurable processor optimized for media processing and streaming.

Instruction set synthesis is a form of architectural optimization that concentrates on instructions. Holmer and Despain [32] used a 1% rule to determine what instructions to add to the architecture—an instruction must produce at least 1% improvement in performance to be included. They used microcode compaction techniques to generate candidate instructions. Huang and Despain [33], in contrast, used simulated annealing. Atasu *et al.* [34] developed graph-oriented instruction generation algorithms that generate large candidate instructions, possibly including disjoint operations.

The XPRES tool from Tensilica, Inc. [35] combines the notions of configurable processor, instruction set extensions, and automated synthesis into a tool flow that starts with user application code and ends up with a configured instruction-extended processor tuned to the particular application(s). XPRES uses optimization and design space exploration techniques that allow users to select the right combination of performance improvement, area increase, and energy reduction that best meets their design objectives. The user may be presented with a Pareto-style tradeoff curve that graphically illustrates the choices available between area and cost increases versus performance improvement.

Several commercial approaches that generate application-specific instruction set processors or coprocessors from scratch exist. These start with either application source code (e.g., Synfora PICO, which is based on research from HP Laboratories [36]) or compiled binary code (e.g., Critical Blue Cascade [37]) and generate a custom highly application-tuned coprocessor.

B. Encoding

MPSoCs provide many opportunities for optimizing the encoding of signals that are not directly visible by the user. Signal encoding can improve both area and power consumption. Code compression and bus encoding are two well-known examples of encoding optimizations.

Code compression generates optimized encodings of instructions that are then decoded during execution. Compressed instructions do not have to adhere to traditional constraints on

instruction set design such as instruction length or fields. Code compression was introduced by Wolfe and Chanin [38], who used Huffman coding. A similar scheme was provided on one model of PowerPC [39]. A number of groups have studied code compression since then; we mention only a few here. Lefurgy *et al.* [40] introduced the branch table, which improves the handling of branches. Larin and Conte [41] tailored the encoding of fields in instructions based on the program's use of those fields. Lekatsas and Wolf [42] used arithmetic coding and Markov models.

Data compression is more complex because data is frequently written, which means that the architecture must be able to both compress and decompress on the fly. Tremaine *et al.* [43] developed an architecture that uses Lempel-Ziv compression to compress and uncompress data moving between the level 3 cache and main memory. Benini *et al.* [44] used a data compression unit to compress and decompress data transfers.

Buses are good candidates for encoding because they are heavily used and have large capacitances that consume large amounts of power. Stan and Burleson introduced bus-invert coding [45], which transmits either the true or complement form of a word, depending on the bits of the previous transfer, to reduce the number of bit flips on the bus. Mussoll *et al.* [46] developed working-zone encoding, which divides programs into sets of addresses for encoding. Benini *et al.* [47] clustered correlated address bits. Lv *et al.* [48] used a dictionary-based scheme.

C. Interconnect-Driven Design

Traditionally, in SoC design, the design of on-chip interconnects has been an afterthought or a choice made obvious by the buses available with the single processor chosen for the SoC. Because early SoCs were driven by a design approach that sought to integrate onto a single chip the architecture of a complete board, most interconnect choices were based on conventional bus concepts. Thus, the two best known SoC buses are ARM AMBA (ARM processors) and IBM CoreConnect (PowerPC and POWER processors). AMBA is actually a bus family, following conventional memory-mapped, single, or multiple master bus controls, with bridges between buses of different performance and sometimes to split the bus into domains. AMBA buses include APB for peripherals, the AMBA AHB for the main processor, and now a crossbar-based split transaction bus for multiple masters and slaves (AXI). The classic bus-based architecture places low-speed peripherals on a lower speed peripheral bus, which is bridged to a higher speed system bus on which sits the main processor and high speed hardware accelerators, as well as bridges to other processors with their own high performance system buses.

It has long been predicted that as SoCs grow in complexity with more and more processing blocks (and MPSoCs will of course move us in this direction faster than a single-processor SoC), current bus-based architectures will run out of performance and, in addition, consume far more energy than desirable to achieve the required on-chip communications and bandwidth [49]. The search for alternative architectures has led to the concept of network-on-chip (NoC) (for a good survey, see the book edited by De Micheli and Benini [50]). After several years of primarily being a research area, there are now several credible commercial NoCs. The key idea behind an NoC is to use a hierarchical network with routers to allow packets to flow more efficiently between originators and targets and to provide additional communications resources (rather than, for example, a single shared bus) so that multiple communications channels can be simultaneously operating. In this, the energy and performance inefficiencies of shared-medium bus-based communications can be ameliorated. Many NoC architectures are possible and the research community has explored several of them.

Sonics SiliconBackplane was the first commercial NoC [51]. Sonics offers a TDMA-style interconnection network with both fixed reservations of bandwidth along with the grantback of unneeded communication allocations. Today, several other vendors provide NoCs, including Arteris [52] and Silistix [53]. The Silistix interconnect is an asynchronous interconnect that has adaptors for common bus protocols and is meant to fit into an SoC built by using the globally asynchronous locally synchronous paradigm.

D. Core- and Platform-Based Design

Bergamaschi *et al.* [54] developed a core-based synthesis strategy for the IBM CoreConnect bus. Their Coral tool automates many of the tasks required to stitch together multiple cores using virtual components. Each virtual component describes the interfaces for a class of real components. If the glue logic supplied with each core is not sufficient, Coral can synthesize some combinational logic. Coral also checks the connections between cores using Boolean decision diagrams.

Cesario and Jerraya [55] use wrappers to componentize both hardware and software. Wrappers perform low-level adaptations like protocol transformation. A golden abstract architecture defines the platform, which can then be specialized to create an implementation.

CoWare N2C [56] and Cadence VCC [57] were very early tools for SoC integration and platform-based design from the mid to late 1990s. N2C supported relatively simple platforms with one main processor, a standard bus architecture, and hardware peripherals of various types including accelerators. Its primary distinguishing feature was interface synthesis, which is a way of describing high-level communication protocols between software functions and hardware blocks, and via template-driven generation, the capability of directly generating relevant software and hardware that would implement the selected protocols. VCC was a slightly more generic tool that implemented the function-architecture codesign concept. The function of an intended system would be captured independent of the architecture, and an explicit mapping between functional blocks and architectural elements and communications channels between functional blocks and associated architectural channels would be used to carry out the performance estimation of the resulting mapped system. A variety of links to both hardware and software implementation were also generated from VCC.

VCC survived for a few years as a commercial product before being abandoned by Cadence. N2C was gradually replaced by

CoWare with the Platform Architect tool. Platform architect follows a more conventional line in common with other existing commercial system level tools such as ARM Realview ESL SoC Designer (now Carbon Design Systems).

In these kinds of tools, a variety of IP blocks such as processors, buses, memories, DMA controllers, and other peripherals are selected from libraries and are interconnected, assuming that the right kinds of protocol matching bridges exist. From the architecture, simulation models based on SystemC can be generated and precompiled software is loaded into the appropriate processor instruction set simulator models. The systems can be simulated, often in a choice between a cycle-accurate mode and a fast functional mode as a "virtual prototype," and various system performance information can be tracked and analyzed. Bus contention and traffic and the relationship between processor activity and bus activity can all be assessed, leading to a refined microarchitecture that is more optimized to the application.

These more modern system level design tools offer less methodology assistance in deciding on the mapping between software tasks and processors and between communication abstractions and implementations than N2C and VCC offered. However, because they are not restricted in the architectural domain to just the elements that supported the N2C and VCC methodologies, they are arguably more general and also easier for a design team to adopt as part of a more gradual move to system-level MPSoC design.

Additional information about system level design tools is available in the book by Bailey *et al.* [53].

E. Memory System Optimizations

A variety of algorithms and methodologies have been developed to optimize the memory system behavior of software [58]. Catthoor *et al.* developed an influential methodology for memory system optimization in multimedia systems: data flow analysis and model extraction, global data flow transformations, global loop and control flow optimization, data reuse optimization, memory organization design, and in-place optimization. A great deal of work at the IMEC, such as that by Franssen *et al.* [59], De Greef *et al.* [60], and Masselos *et al.* [61], developed algorithms for data transfer and storage management.

Panda *et al.* [62] developed algorithms that place data in memory to reduce cache conflicts and place arrays accessed by inner loops. Their algorithm for placing scalars used a closeness graph to analyze the relationships between accesses. They used an interference graph to analyze arrays. Kandemir *et al.* [63] combined data and loop transformations to optimize the behavior of a program in cache.

Scratch pad memories have been proposed as alternatives to address-mapped caches. A scratch pad memory occupies the same level of the memory hierarchy as does a cache, but its contents are determined by software, giving more control of access times to the program. Panda *et al.* [64] developed algorithms to allocate variables to the scratchpad. They statically allocated scalars. They defined a series of metrics, including variable and interference access counts, to analyze the behavior of arrays that have intersecting lifetimes.

Gordon-Ross *et al.* [65] optimized the cache hierarchy: first, the cache size for each level, then line size for each level, then associativity per level. Balasubramonian *et al.* [66], among others, have proposed configurable caches that can be reconfigured at run time based on the program's temporal behavior.

F. Hardware/Software Codesign

Hardware/software codesign was developed in the early 1990s. Hardware/software cosynthesis can be used to explore the design space of heterogeneous multiprocessors. Cosyma [67], Vulcan [68], and the system of Kalavade and Lee [69] were early influential cosynthesis systems, with each taking a very different approach to solving all the complex interactions due to the many degrees of freedom in hardware/software codesign: Cosyma's heuristic moved operations from software to hardware, Vulcan started with all operations in hardware and moved some to software, and Kalavade and Lee used iterative performance improvement and cost reduction steps.

Cost estimation is one very important problem in cosynthesis because area, performance, and power must be estimated both accurately and quickly. Cost estimation methods were developed by Herman *et al.* [70], Henkel and Ernst [71], Vahid and Gajski [72], and Xie and Wolf [73] among others.

A number of other algorithms have been developed for cosynthesis: Vahid *et al.* used binary search [74], Eles *et al.* [75] compared simulated annealing and tabu search, Lycos [76] used heuristic search over basic scheduling blocks, Wolf [77] used heuristic search that alternated between scheduling and allocation, Dick and Jha [78] applied genetic algorithms, COSYN [79] used heuristics optimized for large sets of tasks, some of which were replicated, and Serra [80] combined static and dynamic task scheduling.

G. SDEs

SDEs for single processors can now be quickly retargeted to new architectures. A variety of commercial and open-source SDEs illustrate this technique. No comparable retargeting technology exists for multiprocessors. As a result, MPSoC development environments tend to be collections of tools without substantial connections between them. As a result, programmers and debuggers often find it very difficult to determine the true state of the system.

Popovici *et al.* [81] developed a software development platform that can be retargeted to different multiprocessors. It provides four levels of abstraction. The SA level models the system as a set of functions grouped into tasks connected by abstract communication links. The SA model is formulated by using standard Simulink functions. The virtual architecture level refines software into a C code that communicates by using a standard API. The transaction accurate architecture level models the machine and operating system in sufficient detail to allow the debugging of communication and performance. The virtual prototype level is described in terms of the processor and low-level software used to implement the final system.

VI. Conclusion

We believe that MPSoCs illustrate an important chapter in the history of multiprocessing. In conclusion, we need to ask whether that chapter will close as general-purpose architectures take over high-performance embedded computing or whether MPSoCs will continue to be designed as alternatives to general-purpose processors.

If processors can supply sufficient computational power, then the ease with which they can be programmed pulls system designers toward uniprocessors. Much audio processing, except for very low-cost and low-power applications, is performed on DSPs. Whereas DSPs have somewhat different architectures than do desktop and laptop systems, they are von Neumann machines that support traditional software development methods.

However, even today's applications can be handled by future generations of general-purpose systems; thanks to Moore's Law advances, we believe that new applications that will require the development of new MPSoCs will emerge. Embedded computer vision is one example of an emerging field that can use essentially unlimited amounts of computational power but must also meet real-time, low-power, and low-cost requirements. The design methods and tools that have been developed for MPSoCs will continue to be useful for these next-generation systems.

Acknowledgment

The authors would like to thank B. Ackland and S. Dutta for the helpful discussions of their MPSoCs.

References

[1] W. Wolf, "The future of multiprocessor systems-on-chips," in *Proc. 41st Annu. Des. Autom. Conf.*, 2004, pp. 681–685.

[2] W. J. Bouknight, S. A. Denenberg, D. E. McIntyre, J. M. Randall, A. H. Sameh, and D. L. Slotnick, "The Illiac IV system," *Proc. IEEE*, vol. 60, no. 4, pp. 369–388, Apr. 1972.

[3] W. A. Wulf and S. P. Harbison, "Reflections in a pool of processors—An experience report on C.mmp/Hydra," in *Proc. AFIPS Nat. Comput. Conf.*, Jun. 1978, pp. 939–951.

[4] D. E. Culler, J. P. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*. San Francisco, CA: Morgan Kaufmann, 1999.

[5] G. S. Almasi and A. Gottlieb, *Highly Parallel Computing*, 2nd ed. Redwood City, CA: Benjamin Cummings, 1994.

[6] B. Ackland, A. Anesko, D. Brinthaupt, S. J. Daubert, A. Kalavade, J. Knobloch, E. Micca, M. Moturi, C. J. Nicol, J. H. O'Neill, J. Othmer, E. Sackinger, K. J. Singh, J. Sweet, C. J. Terman, and J. Williams, "A single-chip, 1.6-billion, 16-b MAC/s multiprocessor DSP," *IEEE J. Solid-State Circuits*, vol. 35, no. 3, pp. 412–424, Mar. 2000.

[7] *C-5 Network Processor Architecture Guide*, C-Port Corp., North Andover, MA, May 31, 2001. [Online]. Available: http://www.freescale.com/files/netcomm/doc/ref_manual/C5NPD0-AG.pdf?fsrch=1

[8] S. Dutta, R. Jensen, and A. Rieckmann, "Viper: A multiprocessor SOC for advanced set-top box and digital TV systems," *IEEE Des. Test. Comput.*, vol. 18, no. 5, pp. 21–31, Sep./Oct. 2001.

[9] *OMAP5912 Multimedia Processor Device Overview and Architecture Reference Guide*, Texas Instruments Inc., Dallas, TX, Mar. 2004. [Online]. Available: http://www.ti.com

[10] A. Artieri, V. D'Alto, R. Chesson, M. Hopkins, and M. C. Rossi, *Nomadik—Open Multimedia Platform for Next Generation Mobile Devices*, 2003. technical article TA305. [Online]. Available: http://www.st.com

[11] J. Goodacre and A. N. Sloss, "Parallelism and the ARM instruction set architecture," *Computer*, vol. 38, no. 7, pp. 42–50, Jul. 2005.

[12] *Intel IXP2855 Network Processor*, Intel Corp., Santa Clara, CA, 2005. [Online]. Available: http://www.intel.com

[13] M. Kistler, M. Perrone, and F. Petrini, "Cell multiprocessor communication network: Built for speed," *IEEE Micro*, vol. 26, no. 3, pp. 10–23, May/Jun. 2006.

[14] J. Glossner, D. Iancu, J. Lu, E. Hokenek, and M. Moudgill, "A software-defined communications baseband design," *IEEE Commun. Mag.*, vol. 41, no. 1, pp. 120–128, Jan. 2003.

[15] J. Glossner, M. Moudgill, D. Iancu, G. Nacer, S. Jinturkar, S. Stanley, M. Samori, T. Raja, and M. Schulte, *The Sandbridge Sandbridge Convergence Platform*, 2005. [Online]. Available: http://www.sandbridgetech.com

[16] W. Eatherton, "The push of network processing to the top of the pyramid," in *Proc. Symp. Architectures Netw. Commun. Syst.*, Princeton, NJ, Oct. 26–28, 2005.

[17] S. Otsuka, "Design of a printer controller SoC using a multiprocessor configuration," in *Proc. Nikkei Embedded Processor Symp.*, Tokyo, Japan, Oct. 31, 2006. Presentation in Japanese/English.

[18] L. Hammond, B. A. Nayfeh, and K. Olukotun, "A single-chip multiprocessor," *Computer*, vol. 30, no. 9, pp. 79–85, Sep. 1997.

[19] S. Gochman, A. Mendelson, A. Naveh, and E. Rotem, "Introduction to Intel Core Duo processor architecture," *Intel Technol. J.*, vol. 10, no. 2, pp. 89–97, May 15, 2006.

[20] K. Quinn, J. Yang, and V. Turner, *The Next Evolution in Enterprise Computing: The Convergence of Multicore X86 Processing and 64-bit Operating Systems*, 2005. [Online]. Available: http://www.amd.com

[21] A. S. Leon, J. L. Shin, K. W. Tam, W. Bryg, F. Schumacher, P. Kongetira, D. Weisner, and A. Strong, "A power-efficient high-throughput 32-thread SPARC processor," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2006, pp. 98–107.

[22] B. G. Haskell, A. Prui, and A. N. Netravali, *Digital Video: An Introduction to MPEG-2*. New York: Chapman & Hall, 1997.

[23] J. Xu, W. Wolf, J. Henkel, and S. Chakradhar, "A design methodology for application-specific networks-on-chips," *ACM Trans. Embed. Comput. Syst.*, vol. 5, no. 2, pp. 263–280, May 2006.

[24] T. Austin, D. Blaauw, S. Mahlke, T. Mudge, C. Chakrabarti, and W. Wolf, "Mobile supercomputers," *Computer*, vol. 35, no. 7, pp. 81–83, May 2004.

[25] W. Wolf, *Modern VLSI Design: System-on-Chip Design*, 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 2004.

[26] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, 4th ed. San Francisco, CA: Morgan Kaufmann, 2006.

[27] P. Ienne and R. Leupers, Eds., *Customizable Embedded Processors*. San Francisco, CA: Morgan Kaufmann, 2006.

[28] P. Marwedel, "The MIMOLA design system: Tools for the design of digital processors," in *Proc. 21st Des. Autom. Conf.*, 1984, pp. 587–593.

[29] S. Kobayashi, K. Mita, Y. Takeuchi, and M. Imai, "Rapid prototyping of JPEG encoder using the ASIP development system: PEAS-III," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, Apr. 2003, vol. 2, pp. 485–488.

[30] A. Hoffman, T. Kogel, A. Nohl, G. Braun, O. Schliebusch, O. Wahlen, A. Wieferink, and H. Meyr, "A novel methodology for the design of application-specific instruction-set processors (ASIPs) using a machine description language," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 11, pp. 1338–1354, Nov. 2001.

[31] C. Rowen, *Engineering the Complex SoC: Fast, Flexible Design With Configurable Processors*. Upper Saddle River, NJ: Prentice-Hall, 2004.

[32] B. K. Holmer and A. M. Despain, "Viewing instruction set design as an optimization problem," in *Proc. 24th Int. Symp. Microarchitecture*, 1991, pp. 153–162.

[33] I.-J. Huang and A. M. Despain, "Synthesis of application specific instruction sets," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 14, no. 6, pp. 663–675, Jun. 1995.

[34] K. Atasu, L. Pozzi, and P. Ienne, "Automatic application-specific instruction-set extensions under microarchitectural constraints," in *Proc. 40th Des. Autom. Conf.*, 2003, pp. 256–261.

[35] D. Goodwin and D. Petkov, "Automatic generation of application specific processors," in *Proc. CASES*, 2003, pp. 137–147.

[36] V. K. S. Aditya, B. R. Rau, and V. Kathail, "Automatic architectural synthesis of VLIW and EPIC processors," in *Proc. 12th Int. Symp. Syst. Synthesis*, 1999, pp. 107–113.

[37] R. Taylor and P. Morgan, "Using coprocessor synthesis to accelerate embedded software," in *Proc. Embedded Syst. Conf.*, 2005.

[38] A. Wolfe and A. Chanin, "Executing compressed programs on an embedded RISC architecture," in *Proc. 25th Annu. Int. Symp. Microarchitecture*, 1992, pp. 81–91.

[39] M. T. Kemp, R. K. Montoye, J. D. Harper, J. D. Palmer, and D. J. Auerbach, "A decompression core for PowerPC," *IBM J. Res. Develop.*, vol. 42, no. 6, pp. 807–812, Nov. 1998.

[40] C. Lefurgy, P. Bird, I.-C. Chen, and T. Mudge, "Improving code density using compression techniques," in *Proc. 30th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 1997, pp. 194–203.
[41] S. Y. Larin and T. M. Conte, "Compiler-driven cached code compression schemes for embedded ILP processors," in *Proc. 32nd Annu. Int. Symp. Microarchitecture*, 1999, pp. 82–92.
[42] H. Lekatsas and W. Wolf, "SAMC: A code compression algorithm for embedded processors," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 12, pp. 1689–1701, Dec. 1999.
[43] R. B. Tremaine, P. A. Franaszek, J. T. Robinson, C. O. Schultz, T. B. Smith, M. E. Wazlowski, and P. M. Bland, "IBM memory expansion technology (MXT)," *IBM J. Res. Develop.*, vol. 45, no. 2, pp. 271–285, Mar. 2001.
[44] L. Benini, D. Bruni, A. Macii, and E. Macii, "Hardware-assisted data compression for energy minimization in systems with embedded processors," in *Proc. Conf. Des. Autom. Test Eur.*, 2002, pp. 449–453.
[45] M. R. Stan and W. P. Burleson, "Bus-invert coding for low-power I/O," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 3, no. 1, pp. 49–58, Mar. 1995.
[46] E. Musoll, T. Lang, and J. Cortadella, "Working-zone encoding for reducing the energy in microprocessor address buses," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 6, no. 4, pp. 568–572, Dec. 1998.
[47] L. Benini, G. De Micheli, E. Macii, M. Poncino, and S. Quer, "Power optimization of core-based systems by address bus encoding," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 6, no. 4, pp. 554–562, Dec. 1998.
[48] T. Lv, J. Henkel, H. Lekatsas, and W. Wolf, "A dictionary-based en/decoding scheme for low-power data buses," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 5, pp. 943–951, Oct. 2003.
[49] W. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. Des. Autom. Conf.*, 2001, pp. 684–689.
[50] G. De Micheli and L. Benini, Eds., *Networks on Chips: Technology and Tools*. San Francisco, CA: Morgan Kaufmann, 2006
[51] D. Wingard, "MicroNetwork-based integration for SOCs," in *Proc. Des. Autom. Conf.*, 2001, pp. 273–677.
[52] A. Fanet, "NoC: The arch key of IP integration methodology," in *Proc. MPSoC Symp.*, 2005.
[53] B. Bailey, G. Martin, and A. Piziali, *ESL Design and Verification: A Prescription for Electronic System Level Methodology*. San Francisco, CA: Morgan Kaufmann, 2007.
[54] R. Bergamaschi, S. Bhattacharya, R. Wagner, C. Fellenz, M. Muhlada, F. White, W. R. Lee, and J.-M. Daveau, "Automating the design of SOCs using cores," *IEEE Des. Test. Comput.*, vol. 18, no. 5, pp. 32–45, Sep./Oct. 2001.
[55] W. O. Cesario and A. A. Jerraya, "Component-based design for multiprocessor systems-on-chips," in *Multiprocessor Systems-on-Chips*, A. A. Jerraya and W. Wolf, Eds. San Francisco, CA: Morgan Kaufmann, 2004, ch. 13.
[56] K. Van Rompaey, I. Bolsens, H. DeMan, and D. Verkest, "CoWare—A design environment for heterogeneous hardware/software systems," in *Proc. Conf. Eur. Des. Autom.*, 1996, pp. 252–257.
[57] S. J. Krolikoski, F. Schirrmeister, B. Salefski, J. Rowson, and G. Martin, "Methodology and technology for virtual component driven hardware/software co-design on the system-level," in *Proc. Int. Conf. Circuits Syst.*, 1999, vol. 6, pp. 456–459.
[58] W. Wolf and M. Kandemir, "Memory system optimization of embedded software," *Proc. IEEE*, vol. 91, no. 1, pp. 165–182, Jan. 2003.
[59] F. Franssen, I. Nachtergaele, H. Samsom, F. Catthoor, and H. De Man, "Control flow optimization for fast system simulation and storage minimization," in *Proc. Int. Conf. Des. Test*, 1994, pp. 20–24.
[60] E. De Greef, F. Catthoor, and H. De Man, "Memory organization for video algorithms on programmable signal processors," in *Proc. ICCD*, 1995, pp. 552–557.
[61] K. Masselos, F. Catthoor, C. E. Goutis, and H. De Man, "A performance oriented use methodology of power optimizing code transformations for multimedia applications realized on programmable multimedia processors," in *Proc. IEEE Workshop Signal Process. Syst.*, 1999, pp. 261–270.
[62] P. R. Panda, N. D. Dutt, and A. Nicolau, "Memory data organization for improved cache performance in embedded processor applications," *ACM Trans. Des. Autom. Embed. Syst.*, vol. 2, no. 4, pp. 384–409, Oct. 1997.
[63] M. Kandemir, J. Ramanujam, and A. Choudhary, "Improving cache locality by a combination of loop and data transformations," *IEEE Trans. Comput.*, vol. 48, no. 2, pp. 159–167, Feb. 1999.
[64] P. R. Panda, N. D. Dutt, and A. Nicolau, "On-chip vs. off-chip memory: The data partitioning problem in embedded processor-based systems," *ACM Trans. Des. Autom. Embed. Syst.*, vol. 5, no. 3, pp. 682–704, Jul. 2000.
[65] A. Gordon-Ross, F. Vahid, and N. Dutt, "Automatic tuning of two-level caches to embedded applications," in *Proc. DATE*, 2004, pp. 208–213.
[66] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dworkadas, "A dynamically tunable memory hierarchy," *IEEE Trans. Comput.*, vol. 52, no. 10, pp. 1243–1258, Oct. 2003.
[67] R. Ernst, J. Henkel, and T. Benner, "Hardware-software cosynthesis for microcontrollers," *IEEE Des. Test Comput.*, vol. 10, no. 4, pp. 64–75, Dec. 1993.
[68] R. K. Gupta and G. De Micheli, "Hardware-software cosynthesis for digital systems," *IEEE Des. Test Comput.*, vol. 10, no. 3, pp. 29–41, Sep. 1993.
[69] A. Kalavade and E. A. Lee, "The extended partitioning problem: Hardware/software mapping, scheduling, and implementation-bin selection," *Des. Autom. Embed. Syst.*, vol. 2, no. 2, pp. 125–163, Mar. 1997.
[70] D. Hermann, J. Henkel, and R. Ernst, "An approach to the adaptation of estimated cost parameters in the COSYMA system," in *Proc. 3rd Int. Workshop Hardware/Software Codes.*, 1994, pp. 100–107.
[71] J. Henkel and R. Ernst, "An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation techniques," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 9, no. 2, pp. 273–289, Apr. 2001.
[72] F. Vahid and D. D. Gajski, "Incremental hardware estimation during hardware/software functional partitioning," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 3, no. 3, pp. 459–464, Sep. 1995.
[73] Y. Xie and W. Wolf, "Hardware/software co-synthesis with custom ASICs," in *Proc. ASPDAC*, 2000, pp. 129–133.
[74] F. Vahid, J. Gong, and D. D. Gajski, "A binary-constraint search algorithm for minimizing hardware during hardware/software partitioning," in *Proc. Conf. Eur. Des. Autom.*, 1994, pp. 214–219.
[75] P. Eles, Z. Peng, K. Kuchinski, and A. Doboli, "System level hardware/software partitioning based on simulated annealing and tabu search," *Des. Autom. Embed. Syst.*, vol. 2, no. 1, pp. 5–32, Jan. 1997.
[76] J. Madsen, J. Grode, P. V. Knudsen, M. E. Petersen, and A. Haxthausen, "LYCOS: The Lyngby co-synthesis system," *Des. Autom. Embed. Syst.*, vol. 2, no. 2, pp. 195–235, Mar. 1997.
[77] W. Wolf, "An architectural co-synthesis algorithm for distributed, embedded computing systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 5, no. 2, pp. 218–229, Jun. 1997.
[78] R. P. Dick and N. K. Jha, "MOGAC: A multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 17, no. 10, pp. 920–935, Oct. 1998.
[79] B. P. Dave, G. Lakshminarayana, and N. K. Jha, "COSYN: Hardware-software co-synthesis of heterogeneous distributed systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 7, no. 1, pp. 92–104, Mar. 1999.
[80] V. J. Mooney, III and G. De Micheli, "Hardware/software co-design of run-time schedulers for real-time systems," *Des. Autom. Embed. Syst.*, vol. 6, no. 1, pp. 89–144, Sep. 2000.
[81] K. Popovici, X. Guerin, F. Rousseau, P. S. Paolucci, and A. Jerraya, "Efficient software development platforms for multimedia applications at different abstraction levels," in *Proc. IEEE Rapid Syst. Prototyping Workshop*, 2007, pp. 113–122.

Wayne Wolf (F'98) received the B.S., M.S., and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1980, 1981, and 1984, respectively.

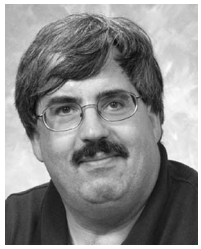
He is Farmer Distinguished Chair and a Research Alliance Eminient Scholar with the Georgia Institute of Technology, Atlanta. He was with AT&T Bell Laboratories from 1984 to 1989. He was with the faculty of Princeton University, Princeton, NJ, from 1989 to 2007. His research interests include embedded computing, embedded video and computer vision, and VLSI systems.

Dr. Wolf is the recipient of the American Society for Engineering Education Terman Award and the IEEE Circuits and Systems Society Education Award. He is a Fellow of the Association for Computing Machinery.

Ahmed Amine Jerraya received the Engineer degree from the University of Tunis, Tunis, Tunisia, in 1980 and the "Docteur Ingénieur" and "Docteur d'Etat" degrees from the University of Grenoble, Grenoble, France, in 1983 and 1989, respectively, all in computer sciences.

In 1986, he held a full research position with the Centre National de la Recherche Scientifique (CNRS), France. From April 1990 to March 1991, he was a Member of the Scientific Staff at Nortel in Canada, where he worked on linking system design tools and hardware design environments. He got the grade of Research Director within CNRS and managed at the TIMA Laboratory the research dealing with multiprocessor systems-on-chips. Since February 2007, he has been with CEA-LETI, where he is the Head of Design Programs. He served as the General Chair for the Conference Design, Automation and Test in Europe in 2001, coauthored eight books, and published more than 200 papers in international conferences and journals.

Dr. Jerraya is the recipient of the Best Paper Award at the 1994 European Design and Test Conference for his work on hardware/software cosimulation.



Grant Martin (M'95–SM'04) received the B.S. and M.S. degrees in combinatorics and optimisation from the University of Waterloo, Waterloo, ON, Canada, in 1977 and 1978, respectively.

He is a Chief Scientist with Tensilica, Inc., Santa Clara, CA. Before that, he was with Burroughs, Scotland for six years, with Nortel/BNR, Canada for ten years, and with Cadence Design Systems for nine years, eventually becoming a Cadence Fellow in their laboratories. He is the coauthor or Co-editor of nine books dealing with system-on-chip (SoC) design, SystemC, UML, modeling, electronic design automation for integrated circuits, and system-level design, including the first book on SoC design published in Russian. His most recent book, "ESL Design and Verification," which was written with Brian Bailey and Andrew Piziali, was published by Elsevier Morgan Kaufmann in February, 2007. He was the Cochair of the Design Automation Conference Technical Program Committee for Methods for 2005 and 2006. His particular areas of interest include system-level design, IP-based design of SoC, platform-based design, and embedded software.