The Future of Multiprocessor Systems-on-Chips

Wayne Wolf
Department of Electrical Engineering
Princeton University
Princeton NJ 08544
wolf@princeton.edu

Abstract

This paper surveys the state-of-the-art and pending challenges in MPSoC design. Standards in communications, multimedia, networking, and other areas encourage the development of high-performance platforms that can support a range of implementations of the standard. A multiprocessor system-on-chip includes embedded processors, digital logic, and mixed-signal circuits combined into a heterogeneous multiprocessor. This mix of technologies creates a major challenge for MPSoC design teams. We will look at some existing MPSoC designs and then describe some hardware and software challenges for MPSoC designers.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**] Special-purpose and application-specific systems. Real-time and embedded systems.

General Terms

Design.

Keywords

MPSoC, system-on-chip, real-time, low power, embedded software.

1. INTRODUCTION

Multiprocessor systems-on-chips (MPSoCs) have started to enter the marketplace over the past several years and are expected to be available in even greater variety over the next few years. An MPSoC combines embedded processors, specialized digital hardware, and often mixed-signal circuits to provide a complete integrated system. MPSoCs are hard to design partly because they implement sophisticated functions and partly because they use such a wide variety of technologies to do the job.

In this paper we'll take a look at what we know about MPSoC design and what challenges need to be solved. We will start with a survey of the present state of MPSoC systems. We will then look at hardware challenges and software challenges that MPSoC designers currently face.

DAC 2004, June 7-11, 2004, San Diego, California, USA Copyright 2004 ACM 1-58113-828-8/04/0006...$5.00.

2. THE PRESENT

Before looking at the future of MPSoC design, we will consider the present state of the art. After briefly discussing the applications of MPSoCs, we will review three commercial MPSoC designs.

2.1 Applications

Multiprocessor systems-on-chips make the most sense in high-volume markets that have strict performance, power, and cost goals. Communications, multimedia, and networking are examples of markets that meet these requirements. These areas are defined by standards that encourage large markets for compliant products. These standards require not just large amounts of computation but complex algorithms that cannot be supported by simple hardware. The cost pressures inherent in large markets, as well as the fact that many of these applications require battery operation, put intense pressure on cost and energy consumption that means performance goals must be met with heterogeneous architectures.

Many embedded applications are commonly referred to as *streaming* applications. Many architectures have been designed to perform relatively simple, predictable operations on periodic data. However, modern embedded applications perform complex processing that does not fit the streaming mold. Consider MPEG compression, for example, which takes in a stream of pixels but turns that into a more complex data set as quickly as possible. Since the goal of data compression is to reduce the data volume, sophisticated algorithms are used to analyze the incoming video and identify features that can be compressed. Motion estimation is the most time-consuming part of video compression; the most effective motion estimation algorithms use sophisticated heuristics to search the space of possible motion. That control ends up as branches, changes in memory fetch behavior, and other phenomena that do not resemble the simple, periodic behavior expected of streaming processors.

2.2 Intel IXP2850 Network Processor

The Intel IXP2850 [1] is designed for network applications that require packet processing, content processing, and security. Basic packet processing requires high throughput. Modern applications such as secure Web browsing add a great deal of processing to the packet stream.

The IXP2850 is a heterogeneous multiprocessor. It has an array of 16 multi threaded microengines to handle packets. An Xscale processor handles control operations. Two

cryptography engines accelerate basic security algorithms. The chip also supports multiple ports for packet ingress and egress and a PCI interface.

The IXP2850's software development environment includes a simulator that allows development of the network application without having a complete board design. The SDK also includes libraries for commonly used operations in networking and security.

2.3 Philips Nexperia™

The Philips Nexperia™ Digital Video Platform [2] is an architecture for digital video entertainment systems. It is designed support digital television, home gateway and networking, and set-top box applications. The first member of this family was the Viper PNX-8500.

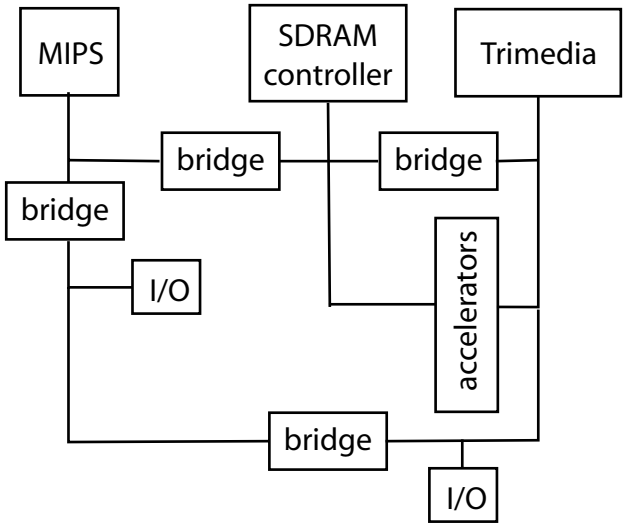

Figure 1. Architecture of the Philips Nexperia PNX-8500.

The architecture is shown in Figure 1. It contains two CPUs and three busses. One CPU is a Trimedia TM32 5-issue VLIW processor. The other processor is a MIPS PR3940 32-bit CPU. The system includes three main busses: one for the memory system, and one each for the MIPS and Trimedia processors. It also includes bridges to connect the busses to each other.

The PNX-8500 contains several accelerators: a 2-D rendering engine, MPEG-2 video decoder, image composition processor, video input processor, scaler, and system processor. It has a memory controller for the external DRAM interface as well as DMA units for each processor. It also includes a variety of I/O units, include UART, PCI, IEEE 1394, and SPDIF digital audio output.

The MIPS core runs the operating system and control processing tasks. It can execute man different operating systems, such as Windows CE, WindRiver, VxWorks, and Linux. The CPUs share some peripherals and use semaphores to negotiate ownership of those shared resources.

2.4 TI OMAP™

The TI OMAP™ architecture [3] designed to support 2.5G and 3G wireless applications. In addition to basic voice services, it is intended for speech processing, location-based services, security, gaming, and multimedia.

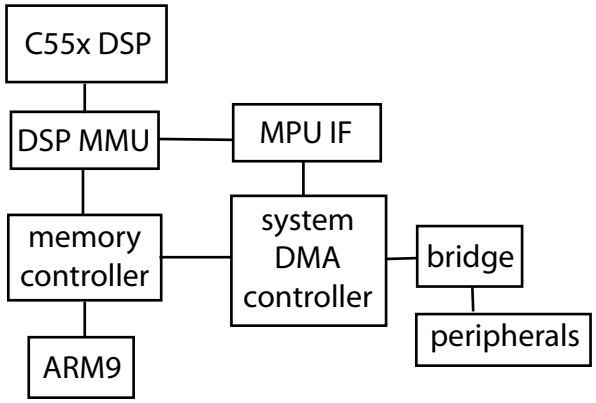

Figure 2. Block diagram of the TI OMAP5910.

Figure 2 shows the block diagram of the TI OMAP 5910. It includes a TI C55x DSP and an ARM9TDMI processor. The two processors share an external DRAM interface. The chip includes 192K Bytes of shared internal SRAM. Peripherals include USB, I^2C, UARTs, GPIOs, and four interprocessor mailboxes.

The DSP/BIOS™ bridge supports heterogeneous multiprocessing. The DSP Resource Manager runs on the ARM to control tasks on the DSP. The DSP has its own real-time OS to implement the control of DSP tasks. The bridge supports interprocessor communication, based upon the mailbox-interrupt mechanism.

2.5 ST Nomadik™

The ST Nomadik™ architecture [4] is designed for mobile multimedia applications. This requires supporting MPEG-4 encoding and decoding. It also requires supporting a range of display sizes, ranging from cell phone (160 x 160) to PDA (320 x 240 up to 680 x 480).

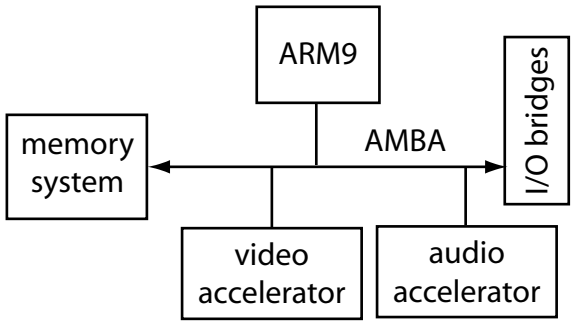

Figure 3. Block diagram of an ST Nomadik system.

Figure 3 shows the block diagram of the first implementation of Nomadik's heterogeneous multiprocessor architecture. The main CPU for the system is the ARM926E-JS. However, this CPU concentrates on coordination and control; most of the media functions are performed by application-specific accelerators. The first version of Nomadik includes audio and

video accelerators. All these units are connected by an AMBA bus.

Each accelerator is itself a small heterogeneous processor. Both are built around ST's MMDSP+ embedded core, which provides 16- or 24-bit integer arithmetic and 32-bit floating point arithmetic. The video accelerator includes an MMDSP+, a picture input formatting unit, video codec unit, picture post-processing unit, and interrupt controller. The video accelerator includes two busses: one which connects the MMDSP+, hardware units, memory, and interrupt controllers; the other connects the hardware units to each other and to the AMBA bus. The audio accelerator relies less on hardware units. It includes an MMDSP+, memory, I/O units, and two internal busses.

Nomadik uses techniques at all levels of abstraction to minimize energy consumption: distributed heterogeneous processors, efficient algorithms for motion estimation and other compute-intensive steps, data compression on the bus, embedded memory, specialized instruction sets, and aggressive power management. Nomadik consumes 20 mW while decoding MPEG-4 in 15 frames/sec QCIF resolution, including MP3 audio.

Nomadik's ARM core allows software to easily be ported to the platform. Nomadik supports the OMAP architecture.

3. HARDWARE CHALLENGES

In order to understand the hardware challenges presented to us by multiprocessor systems-on-chips, we need to ask ourselves two questions: what is new about MPSoC systems and how many different MPSoC platforms do we need?

3.1 What Is New About MPSoCs?

The first question that we must ask ourselves is how much of the architectural technology for MPSoCs has already been invented for machines built with lower levels of integration. The history of computer system design shows that techniques developed for one technology have often been translated to newer technologies: mainframes to minicomputers to microprocessors, etc. And parallel processing has a rich history going back to the earliest days of computing.

However, most parallel processing systems have been developed for scientific computing, databases, or other applications that are very different from embedded computing applications. Two important application characteristics will cause us to rethink some aspects of traditional parallel processing for MPSoCs:

- **real-time operation** Real-time operation requires predictable performance from the architecture. This does not mean that the architecture has to be trivial, such as eliminating caches. It does mean that the architectural elements have to behave predictably enough so that the compiler and programmer can plan how to achieve the required computation rates in critical parts of the system.
- **low-power/energy operation** Most embedded applications operate under either power (heat) or energy constraints. In traditional supercomputers power problems have been tackled largely by packaging and mechanical engineering, freeing the programmer from thinking about the issue. In contrast, power and energy constraints in MPSoCs must be tacked at every level of abstraction. The architecture must provide support for programmers to measure and control energy/power consumption. Furthermore, the architecture's energy and power consumption characteristics must be as predictable as possible.

A key area of concentration to handle both real-time and energy/power problems is the memory system. Memory systems have long been a focus of architectural optimizations that improve average performance at the expense of a greater variance in access times. Many MPSoC applications are memory intensive, which amplifies the effect of those variations in processing time. The memory system is also a prime determinant of energy consumption.

One simple but important question is the memory system programming model: shared or partitioned memory? Scientific and business applications long ago embraced the shared memory model because it greatly simplifies programming. However, very few MPSoCs or even embedded multiprocessors built from multiple chips use a shared memory model for their time-critical applications. Consider, for example, the multiprocessors used in radar, which generally pass messages along high-speed serial links that are incorporated into most high-performance DSPs.

There are some good reasons why the time-critical parts of embedded applications should be built on more heterogeneous platforms. Memory access times need not be uniform but they must be predictable. Building a specialized memory system allows the architect to more carefully characterize its behavior and the effect of run-time variations. A more specialized structure can also conserve energy. A variety of software techniques can be used [5] to tune memory system behavior, but architecture must work with the software to provide predictable performance.

3.2 How Many Platforms Are There?

We also need to ask ourselves how many different MPSoC platforms will be needed in the long term. One simple way to determine the number of platforms would be to categorize problems by data rate. We could then develop a multiprocessor platform for each data rate and rely on software to customize the platform for various applications. This approach has its appeal—a homogeneous multiprocessor is easier to program and could be manufactured in very large volumes.

While data rate is one useful metric for categorizing architectures, power and energy consumption will probably drive us to more heterogeneous architectures over the long run. As we eliminate some functions and add others in order to specialize the machine for a particular purpose, we make it more useful for some applications and less useful for others. We end up with several platforms that operate in the same general operations-per-second range but provide substantially different performance and energy consumption on a given application.

A more subtle design point that could lead to more platforms is buffering and memory management. Because many embedded multiprocessors use heterogeneous memory systems to efficiently meet real-time requirements, MPSoCs could use the same set of processing elements but with different memory architectures. An interesting problem in embedded computing architecture is to find flexible memory architectures that can support a variety of real-time requirements.

I/O is a factor that could drive us to more platforms. Applications with similar computational loads may require very different devices. But we can handle that to some extent with support chips that let us reuse the core MPSoC. In the case of analog I/O, this approach is standard. A DVD chip set, for example, includes a digital processor and an analog support chip, with the analog chip designed in a different technology.

4. SOFTWARE CHALLENGES

Probably the most difficult thing for hardware designers to get used to as they move to MPSoC design is that they must worry about software design from the beginning. The hardware architect can't simply create a machine and throw it over the wall for someone else to program. The architects must understand the application to know what can be thrown out from the hardware and what must be left in. The architects must also understand the characteristics of the application software that affect real-time and low-power operation.

Software plays a critical role in MPSoC design because the chip won't do anything without software. But the fullest view of software challenges relating to MPSoCs starts with the design environment and then moves to the chip itself.

4.1 Development Environments and Tools

Development environments are necessary to allow programmers to create code for the system, starting well before the chip is fabricated. We normally think of the host software when we talk about a development environment, but in fact the development environment includes the target hardware as well. If the development environment requires a complete chip to operate, then software development will be intolerably delayed. Software and hardware designers must work together to develop methodologies that allow them to develop as much software as possible without a working MPSoC. At least part of the solution is a software architecture that partitions functions such that a great deal of software can be implemented on a subset of the architecture's processors, using a configuration that can be supplied by existing software and development environment configurations.

Because MPSoCs are often designed to comply with standards, software development does not start with a blank slate. A great deal of the software effort is to port the reference implementation of the standard to the platform. Because reference implementations are usually written with functionality, not performance in mind, porting the code requires the use of software analysis tools.

A great deal of the performance analysis for MPSoC designs is done using execution traces. A trace is gathered from an existing system, or perhaps synthesized, and then used to drive a simulation. The simulation is monitored to evaluate a variety of characteristics: execution time, memory system behavior, subsystem utilization, etc.

While trace-driven design can be very powerful, it also represents the first software difficulty in the MPSoC design process. System designers need to be able to collect interesting, realistic traces of their system to feed simulators. This requires either a working system of some sort—perhaps an existing chip, perhaps a software design running on a general-purpose processor—or a reliable way of generating synthetic traces. In some cases, standards dictate input formats to the extent that plausible input sequences can be constructed. In other cases, traces must be gathered from executing systems.

The designers also need to invest the time to build simulation models that can make use of the traces. The proper simulation tools depend on the level of abstraction to be modeled. SystemC is popular for system-level modeling, while SimpleScalar [6] is a well-known CPU simulator. SimpleScalar is designed to be quickly reconfigured to different CPU characteristics; however, there is still some work to be done to create easy-to-configure highly accurate simulators for heterogeneous multiprocessors.

4.2 Operating Systems and Middleware

A great many real-time operating systems (RTOSs) have been created for embedded computing systems. However, most commercial RTOSs have been developed for the industrial, automotive, or other markets that permit large images and emphasize functionality. MPSoCs, in contrast, generally require their core functions to be implemented in a very small amount of software, both for performance and memory limitations. MPSoCs will make extensive use of middleware that provide advanced features like Internet access on top of their microkernels.

The operating system and middleware must provide several categories of services, including scheduling, memory management, I/O management, and communication. The exact division between OS and middleware is a choice for software architects, but MPSoCs clearly need to execute some key functions very efficiently. Interprocessor communication primitives, for example, must execute in a very small number of cycles in order to meet real-time performance goals as well as energy consumption goals. The OS and middleware should be optimized to take advantage of the features of the heterogeneous multiprocessor to provide the necessary mix of high-performance, time-critical functions with less-critical operations.

4.3 Embedded System Security

Any type of programmable system has potential security problems. However, as MPSoCs start to include Internet connections, they will become vulnerable to a variety of security attacks. When MPSoCs are used in safety-critical applications, such as cars and airplanes, those security problems must be a central concern of the system architects. Even in non-safety-critical systems, such as home entertainment systems, poor security on the MPSoC can make the chip unusable in practical systems.

As Phil Koopman has pointed out, traditional computer security methodologies and systems are designed to protect transactions. However, MPSoCs that are used in real-time systems must secure the continuous operation of that real-time control function. Attackers may try to disrupt the MPSoC's real-time characteristics in a variety of ways, such as quality-of-service attacks. The MPSoC must be designed from the ground up for secure operation. Software and hardware architects must work together to provide the proper system modes and operations that allow the chip to do its job with a minimum threat from the outside.

5. CONCLUSIONS

Multiprocessor systems-on-chips have a bright future. They are being incorporated into some of the key electronic products of the next decade. We can learn a lot about the design of MPSoCs by studying traditional computer system designs. However, the real-time and low-power/energy characteristics of embedded computing applications provide some unique challenges that should keep MPSoC designers busy for quite some time.

Acknowledgments

Thanks to Ahmed Jerraya, Alain Mellan, Faraydon Karim, Santanu Dutta, Pierre Paulin, and Phil Koopman for fruitful discussions about multiprocessor systems-on-chips.

References

[1] Intel, "Product Brief: Intel IXP2850 Network Processor," 2002, Available at http://www.intel.com.

[2] S. Dutta, R. Jensen, and A. Rieckmann, "Viper: A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems," *IEEE Design and Test of Computers*, September/October 2001, pp. 21-31.

[3] Justin Helmig, "Developing core software technologies for TI's OMAP™ platform," Texas Instruments, 2002. Available at http://www.ti.com.

[4] Alain Artieri, Viviana D'Alto, Richard Chesson, Mark Hopkins, and Marco C. Rossi, "Nomadik™ Open Multimedia Platform for Next-generation Mobile Devices," STMicroelectronics Technical Article TA305, 2003, available at http://www.st.com.

[5] Wayne Wolf and Mahmut Kandemir, "Memory system optimization of embedded software," *Proceedings of the IEEE*, 91(1), January 2003, pp. 165-182.

[6] Doug Burger and Todd M. Austin, "The SimpleScalar Toolset, Version 2.0," University of Wisconsin-Madison Computer Sciences Department Technical Report #1342, June 1997.