9-1-2006

Using System-on-a-Programmable-Chip Technology to Design Embedded Systems

Tyson S. Hall
Southern Adventist University, tyson@southern.edu

James O. Hamblen
Georgia Institute of Technology - Main Campus

Using System-on-a-Programmable-Chip Technology to Design Embedded Systems

J. O. Hamblen*
Georgia Institute of Technology, Atlanta, GA 30332, USA

T. S. Hall†
Southern Adventist University, Collegedale, TN 37315, USA

Abstract

This paper describes the tools, techniques, and devices used to design embedded products with system–on-a-chip (SoC) type solutions using a large Field Programmable Gate Array (FPGA) with an internal processor core. This new FPGA-based approach is called system-on-a-programmable-chip (SoPC). The performance tradeoffs present in SoPC systems is compared to more traditional design approaches. Commercial devices, processor cores, and CAD tool flows are described.

The issues in SoPC hardware/software design tradeoffs are examined and three example SoPC designs are presented as case studies.

Key Words: SoC, SoPC, FPGA, VHDL, verilog, IP core, ASIC, embedded systems

1 Introduction

Traditional system-on-a-chip (SoC) designs require the development of a custom IC or Application Specific Integrated Circuit (ASIC) [2]. Unfortunately, ASIC costs have risen dramatically in recent years along with the vast improvements in VLSI technology feature size and transistor counts. Current ASIC commercial development costs run several million dollars per device. Only a few high volume embedded products can support long ASIC development times and high costs. As a consequence, the number of new traditional ASIC designs has fallen dramatically in recent years.

A promising new alternative technology has emerged that enables designers to utilize a large FPGA that contains both memory and logic elements along with an intellectual property (IP) processor core to rapidly implement a computer and custom hardware for SoC embedded systems [17]. This new FPGA-based methodology is called system-on-a-programmable-chip (SoPC).

The traditional design modalities are ASIC and fixed-processor design. In a fixed-processor design, a commercial microprocessor chip is used along with several selected support chips on a printed circuit board. In an ASIC design, the end user combines user designed logic and other elements from the vendor's IP core library functions. ASICs then require final expensive custom manufacturing steps to fabricate and test the device.

SoPC design has advantages and disadvantages to both of these alternatives as highlighted in Table 1. The strengths of SoPC design are a reconfigurable, flexible nature, and the short development cycle. However, the trade-offs include lower maximum processor performance, higher unit costs in production, and relatively high power consumption.

Table 1: Comparing SoPC, ASIC, and fixed-processor design modalities [10]

Feature	SoPC	ASIC	Fixed-processor
S/W flexibility	●	●	●
H/W flexibility	●	○	○
Reconfigurability	●	○	○
Development time/cost	●	○	●
Peripheral equipment costs	●	●	○
Performance	⊙	●	●
Production cost	⊙	●	●
Power efficiency	○	●	●

Legend: ● – Good; ⊙ – Moderate; ○ – Poor

The benefit of having a flexible hardware infrastructure can not be overestimated. In many new designs, features and specifications are modified throughout the design cycle. For example, marketing may detect a shift in demand requiring additional features (e.g., demand drops for cell phones without cameras), a protocol or specification is updated (e.g., USB 2.0 is introduced), or the customer requests an additional feature. In traditional design modalities (including ASIC and fixed-processor designs), these changes can dramatically affect the ASIC design, processor selection, and/or printed circuit board

* Dept. of Electrical and Computer Engineering.
† School of Computing.

design. Since the hardware architecture is often settled upon early in the design cycle, making changes to the hardware design later in the cycle will typically result in delaying a product's release and increasing its cost.

Flexible infrastructure can also be beneficial in extending the life (and thus reducing the cost) of a product's hardware. A fixed-processor option will often require additional hardware and perhaps even a new printed circuit board (PCB) design for each product variation. With the flexible, reconfigurable logic present in an SoPC device a single printed circuit board can be designed for use in multiple product lines and in multiple generations/versions of a single product. Using reconfigurable logic as the heart of a design allows it to be reprogrammed to implement a wide range of systems and designs. Extending the life of a board design even one generation can result in significant savings and can largely offset the increased per-unit expense of reconfigurable devices.

The SoPC approach can offer advantages for many embedded systems. ASIC development times are too long and mask setup fees are too high to be considered for most products. However, ASICs can still offer the lowest per unit production cost when quantities are large enough to recoup the very high mask setup and development costs. ASICs may still be required for very low power applications that need long run times using small capacity batteries. Traditional processor and ASIC vendors have also seen the advantage of the SoPC approach. For their next generation of larger ICs, several ASIC and processor vendors are developing embedded FPGA cores to add to their devices to make them more flexible for a wider range of designs.

2 Hardware/Software Design Alternatives

The SoPC-based approach offers new design space alternatives to explore. It is possible to consider design options that use software, dedicated custom hardware, or a mixture of both. Software implementations require less development time, but in many cases, a general-purpose processor will be too slow to perform all of the calculations using only software. To speedup the system, some frequently executed functions can be implemented in hardware in an SoPC design using the FPGA's logic.

SoPC hardware/software design space tradeoffs are shown in Figure 1 with the *Y* axis indicating increasingly complex algorithms or programs, and the *X* axis being the computation or program execution time required. As seen in Figure 1, for simple algorithms a hardware solution offers faster computation times, but it offers less flexibility since the hardware remains dedicated for that calculation and it may require a larger FPGA that consumes more power and increases system cost. Potential SoPC applications to implement in hardware in this region include pre-processing of real-time data with high sample rates, multimedia encoding and decoding, low-level communication protocols, and digital signal processing algorithms such as filters and FFTs. Hardware IP cores designed for FPGAs are available for many of these more common functions such as encoders/decoders, communication protocols, filters, and FFTs. The new hardware can also be designed using the traditional VHDL or Verilog FPGA synthesis tools.

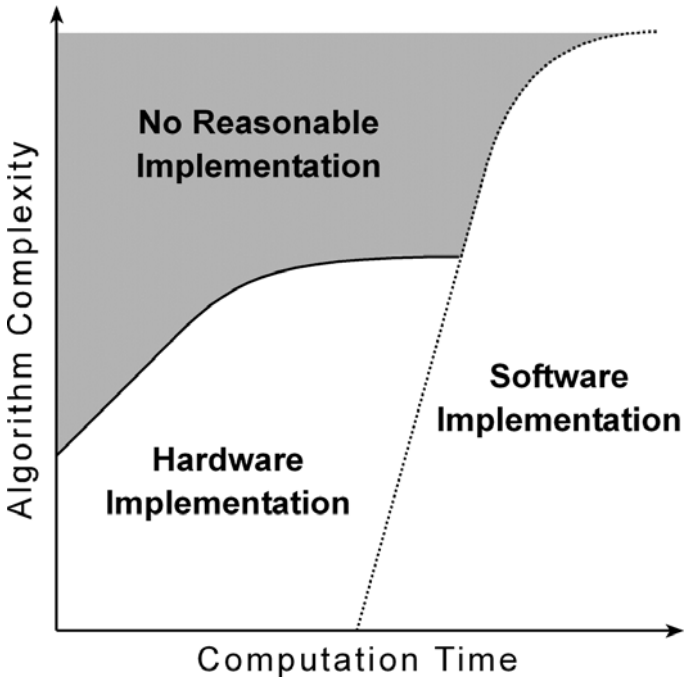


Figure 1: SoPC hardware & software design space tradeoffs

As the size of the hardware needed for implementation of more complex algorithms increases, the hardware starts to slow down, achieves diminishing performance levels, and becomes increasingly more difficult for designers to implement. This occurs due to increasing numbers of gate delays in logic circuits and increases in the distance and communication time needed to transfer data values between hardware units. Pipelining and parallel processing techniques can be used to extend the useable range of hardware solutions, especially for non-recursive algorithms with a high degree of parallelism. (Typically, hardware pipelining works well in FPGAs since they have a register-rich architecture.)

Moving to the upper right in Figure 1, as the algorithm complexity increases, implementation of algorithms using software becomes much easier to design and implement (assuming that the increased computation time still meets system design goals). As seen in the upper left of Figure 1, there can still be some applications that require complex algorithms that are beyond the real-time performance range of a single current generation FPGA's hardware and software.

It is also possible to consider a combination of both hardware and software approaches. Some processor cores allow the designer to add custom instructions. If an application program requires the same calculation repeatedly in loops, adding a custom instruction using extra hardware to accelerate the inner loop code can greatly speed up the application. A similar technique was initially developed in automatic compilers for microprogrammed processors with writeable control store in the 1970s. In an SoPC system, it is possible to add extra hardware to the processor including

ALUs and memory to provide additional speedup for a new instruction. One SoPC processor core from Stretch Incorporated now features a *C* compiler that automatically attempts to implement inner loop code or simple *C* functions in FPGA hardware [18]. In suitable application programs, speedups of one to two orders of magnitude are possible with this approach. For cores that do not allow new instructions to be added, it is also possible to use FPGA logic to build a co-processor for the main processor. The larger FPGAs contain sufficient logic so that several processors working in parallel can be placed in a single FPGA. Current tools provide only limited support for multiple processor cores [7]. Automatic hardware and software partitioning of SoPC systems and partitioning of SoPC systems with multiple processor cores working in parallel is still an active area of research and development.

3 SoPC Processor Cores

SoPC systems require an FPGA with a processor core. Processor cores are classified as either "hard" or "soft." This designation refers to the flexibility/configurability of the core. Hard cores have a custom VLSI layout that is added to the FPGA and they are less configurable; however, they tend to have higher performance characteristics than soft cores [17] .

Hard processor cores use an embedded processor core (in dedicated silicon) in addition to the FPGA's normal programmable logic elements. Hard processor cores added to an FPGA are a hybrid approach, offering performance tradeoffs that fall somewhere between a traditional ASIC and an FPGA; they are available from several manufacturers with a number of different processor flavors [1, 6, 11-12, 16, 18-19]. For example, Altera offers an ARM processor core embedded in its APEX 20KE family of FPGAs that is marketed as an Excalibur™ device. Xilinx's Virtex-II Pro family of FPGAs include up to four PowerPC processor cores on-chip. Stretch Inc. offers SoPC devices with a Tensilica Xtensa RISC configurable processor core. Cypress Semiconductor also offers a variation of the SoPC system. Cypress's Programmable-System-on-a-Chip (PSoC™) is formed on an M8C processor core with configurable logic blocks designed to implement the peripheral interfaces, which include analog-to-digital converters, digital-to-analog converters, timers, counters, and UARTs. Clock rates on commercial hard processor cores are currently in the 300-500 MHz range.

Soft cores, such as Altera's Nios II and Xilinx's MicroBlaze and PicoBlaze processors, use existing programmable logic elements from the FPGA to implement the processor logic. As seen in Table 2, soft-core processors can be very feature-rich and flexible, often allowing the designer to specify the memory width, ALU functionality, number and types of peripherals, and memory address space parameters at compile time. However, such flexibility comes at a cost. Soft cores have slower clock rates and use more power than an equivalent hard processor core.

FPGAs that include hard processor cores require long development times and costs similar to ASICs for the FPGA vendor. By the time a hard processor core is developed, it may compete with a soft core that can run on a newer generation FPGA. With current pricing on large FPGAs, the addition of a soft processor core costs as little as thirty-five cents based on the logic elements it requires. The remainder of the FPGA's logic elements can be used to build application-specific system hardware.

Table 2: Features of commercial soft processor cores for FPGAs [3]

Feature	Nios II 5.0	MicroBlaze 4.0
Gate Count	26,000 – 72,000	30,000 – 60,000
Frequency	50-200 MHz	50-200 MHz
Datapath Width	32 bits	32 bits
Pipeline Stages	1-6	3
Register File	32 general purpose & 6 special purpose	32 general purpose & 32 special purpose
Instruction Word	32 bits	32 bits
Instruction Cache	Optional	Optional
Hardware Multiply & Divide	Optional	Optional
Hardware Floating Point	Third Party	Optional

FPGAs are available with several million gates of programmable hardware logic, a few million bits of internal memory, and optional multiple hard processor cores. Current generation FPGAs are large enough that a mixture of multiple hard and soft cores can even be placed in a single FPGA. FPGAs can also implement relatively fast integer add circuits using their programmable logic making them a useful design platform in DSP-intensive applications. FPGA device families that are specifically designed for the DSP market also include multiple integer hardware multiply circuits, and floating point IP cores are available for some FPGA families.

Traditional system-on-a-chip devices (ASICs and full custom VLSI ICs) still offer higher performance, but they also have large development costs and longer turnaround times. For products requiring a custom hardware implementation, the FPGA-based SoPC approach is easier, faster, and more economical in low to medium quantity production.

4 SoPC Design Flow

Typically, additional software tools are provided along with each processor core to support SoPC development. A special CAD tool specific to each soft processor core is used to configure processor options, which can include register file size, hardware multiply and divide, floating point hardware, interrupts, and I/O hardware. This tool outputs an HDL synthesis model of the processor core in VHDL or Verilog. In addition to the processor, other system logic is added and the resulting design is synthesized using a standard FPGA

synthesis CAD tool. The embedded application program (software) for the processor is typically written in C or C++ and compiled using a customized compiler provided with the processor core tools.

The traditional flow of commercial FPGA design tools typically follows a path from hardware description language (HDL) or schematic design entry through synthesis and place and route tools to the programming of the FPGA. FPGA manufacturers provide CAD tools such as Altera's Quartus II and Xilinx's ISE software, which step the designer through this process. As shown in Figure 2, the addition of a processor core and the tools associated with it are a superset of the traditional FPGA tools. The standard synthesis, place and route, and programming functionality is still needed, and in the case of both Altera and Xilinx, the same CAD tools (Altera's Quartus II or Xilinx's ISE) are used to implement these blocks.

4.1 Processor Core Configuration Tools

Currently, a number of pre-defined processor cores are available from various sources. GPL-licensed public processor cores can be found on the web (i.e., www.opencores.org and www.leox.org), while companies such as Altera (Nios II processor), Xilinx (MicroBlaze and PicoBlaze processor), and Tensilica (Xtensa processor) provide their processors and/or development tools for a fee.

Processor cores provided by FPGA manufacturers are typically manually optimized for the specific FPGA family being used, and as such, are more efficiently implemented on the FPGA than a user-designed core (especially given the time and resource constraints of most projects). Additionally, FPGA companies provide extensive support tools to ease the customization and use of their cores, including high-level compilers and debuggers targeted at the custom cores.

In the case of Altera and Xilinx, the Processor Core Configuration Tool block shown in Figure 2 is realized in a user-friendly GUI interface that allows the designer to customize the processor for a particular project. A screen shot of Altera's processor configuration wizard is seen in Figure 3. The configurable parameters can include the datapath width, memory, address space, and peripherals (including arbitrarily defined general-purpose I/O, UARTs, Ethernet controllers, memory controllers, etc.). Once the processor parameters are specified in the GUI interface, the processor core is generated in the form of an HDL file (in Altera) or a netlist file (in Xilinx). This file can then be included within a traditional HDL or schematic design using the standard CAD tools. Specific pin assignments and additional user logic can be included at this point like any other FPGA design. Next, the full hardware design (processor core and any additional user logic) is compiled (synthesis, place and route, etc.), and the FPGA can be programmed with the resulting file using the standard tools. The hardware design is complete, and the FPGA logic has been determined.

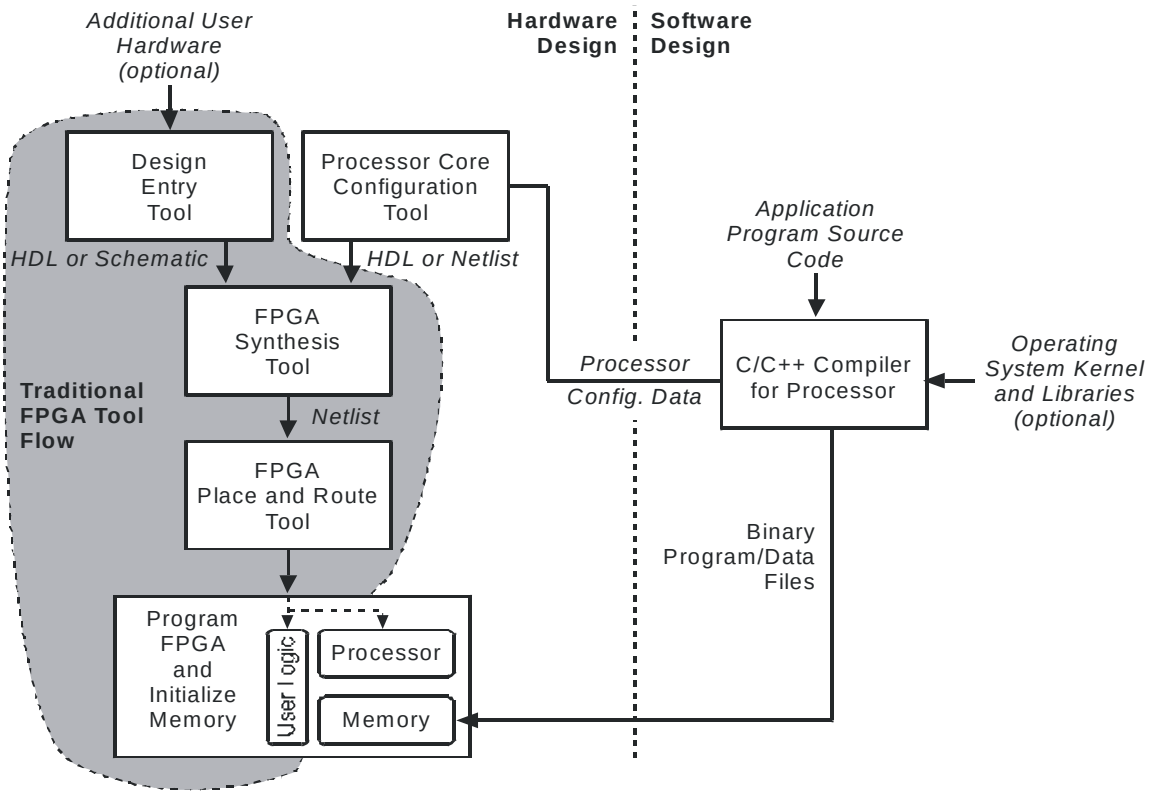


Figure 2: The CAD tool flow for SoPC design is comprised of the traditional design process for FPGA-based systems with the addition of the processor core configuration tool and software design tools. In this figure, the program and data memory is assumed to be on-chip for simplicity [4]

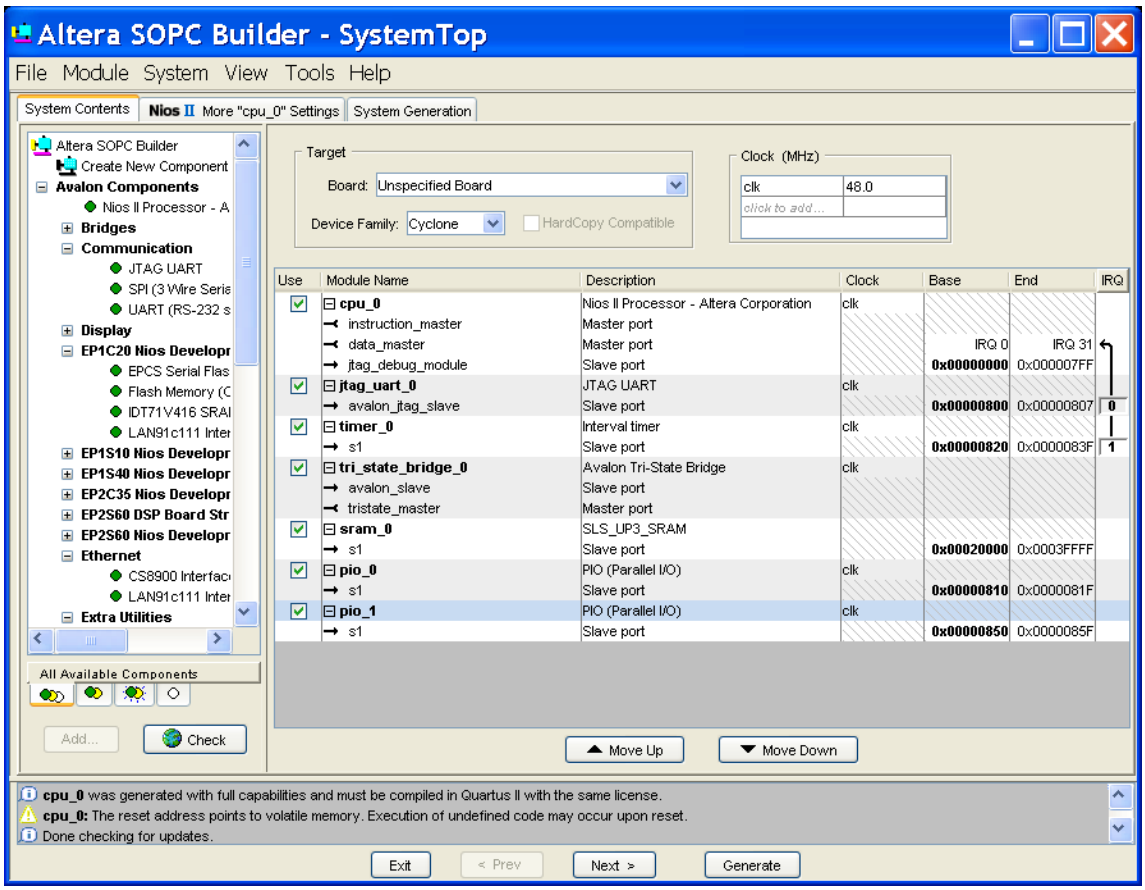


Figure 3: Processor core configuration tool GUI for the Nios II soft processor core. A drag-and-drop style interface can be used to add I/O hardware and set processor options

4.2 High-Level Compiler for Processor Core

As shown on the right side of Figure 2, the next step is to write and compile the software that will be executed on the soft processor core. When the Processor Core Configuration Tool generates the HDL or netlist files, it also creates a number of library files and their associated *C* header files that are customized for the specific processor core generated. A C/C++ cross compiler targeted at this processor is also provided for the development system. The designer can then program stand alone programs to run on the processor. Optionally, the designer can compile code for an operating system targeted for the processor core (see Sect. 6).

4.3 Memory

Once a program/data binary file has been generated, it must be loaded into the processor's program and data memories. This loading can be done several ways depending on the memory configuration of the processor at hand.

If the application program is small and can fit into the memory blocks available on the FPGA, then the program can be initialized in the memory when the hardware configuration is programmed. This initialization is done through the standard FPGA tools, such as Altera's Quartus II software or Xilinx's ISE software. However, on-chip memory is typically very limited, and this solution is not usually a realistic option. Most SoPC systems have one or two small external memory chips in addition to the FPGA. If memory controllers are needed, they are implemented using internal FPGA logic.

4.4 Initializing Program Memory

In a prototyping environment, the application program will most likely be modified a number of times before the final program is complete. In this case, the ability to download the application code from a PC to the memory on an FPGA board must be provided. This functionality, typically called a "bootloader" or "boot monitor," can be implemented in either software or hardware.

A software bootloader is comprised of code that is loaded into an on-chip memory and starts running on power up. This program is small enough (1-2 KB) to fit in most on-chip memories, and its primary function is to receive a program binary file from the development PC, load it into external memory, and then start the new code executing. In this way, a new program can be stored into external memory (SRAM, SDRAM, Flash memory, etc.) by downloading it over a USB

(or other) interface on the fly without having to reload the FPGA's hardware configuration. Xilinx provides a boot monitor for their MicroBlaze soft-core processor that includes the ability to download a program binary over USB (or other interface), store it in memory, and start the code executing. They also provide a more enhanced version called XMDstub that adds debugging capabilities. Altera's legacy Nios processors included a software bootloader; however, a hardware bootloader is the preferred solution in Nios II.

A hardware bootloader provides functionality very similar to a software bootloader; however, it is implemented in hardware logic within the processor core. Typically, the processor will be paused or stalled upon power up and the hardware bootloader will have direct access to memory or the memory registers in the processor's datapath. The bootloader hardware can start and stop the processor and can control the downloading of a program over the USB, JTAG, or serial interface to the desired memory locations. Altera's hardware bootloader is a part of the JTAG debug module, which resides within the Nios II processor. This logic uses the JTAG interface with the PC to receive the execution code, and it then writes the code to the appropriate memory. Finally, the bootloader hardware overwrites the processor's program counter with the start address of the code just downloaded and releases the pause bit to allow the processor to begin executing the downloaded code.

4.5 External Non-Volatile Storage

The application program code can be stored on an external Flash memory, EEPROM, or other forms of non-volatile memory. As with most embedded systems, hard disk drives are rarely used in smaller SoPC-based devices since they have shorter lifetimes than other non-volatile memory components and most systems do not require extremely large amounts of non-volatile storage. The application program and OS code can either be pre-programmed in the external memory module (for a production run) or a bootloader program can be used to store the application program in non-volatile storage. For low-speed applications, the code can be executed directly from the external memory. However, if high-speed functionality is required, then a designer could use three memories as shown in Figure 4. In this scheme, the on-chip memory is initialized with a bootloader, which handles the movement of the application program between the memories. (On-chip memory is replaced with a hardware bootloader on some systems including the Nios II processor.)

The fast, volatile memory (i.e., SDRAM) is used to store the application program during execution, while the slower, non-volatile memory (i.e., Flash or EEPROM) is used for the permanent storage of the application program. The bootloader can be modified to initialize the system, retrieve a program from non-volatile memory, store it in the faster, volatile memory, and then start it executing from the faster memory. This scheme provides the advantages of permanent storage, fast execution, and the ability to modify the application program when needed. Of course, it comes at the expense of having additional memory.

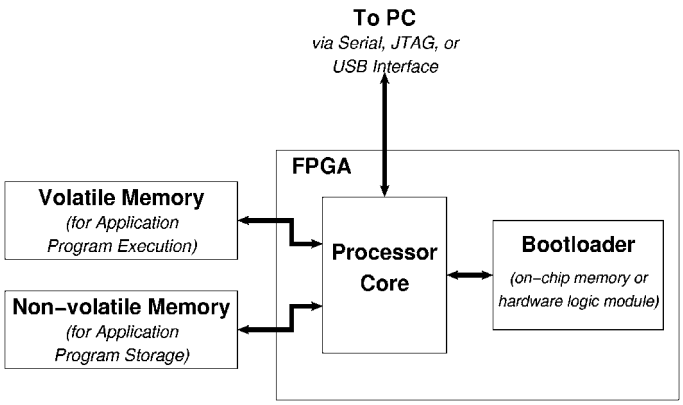


Figure 4: This arrangement of on-chip and external memories provides flexibility and performance to an SoPC system

5 SoPC Development Boards

To enable designers to learn the complicated tool flow and to provide for an early start of hardware/software co-design projects, most FPGA vendors offer SoPC development boards. These boards offer a large FPGA with several megabytes of external memory and a variety of built-in I/O features that are capable of supporting a soft processor core. Many also include FPGAs with hard processor cores. While a custom PCB is being designed for the new SoPC-based product, work can start in parallel on the hardware configuration and software development tasks using the development board (assuming that the board has a similar FPGA device and I/O features).

Two new SoPC development boards from Altera and Xilinx can be seen in Figures 5 and 6. These boards both support a wide array of I/O hardware interfaces including VGA, audio, PS/2, USB, Ethernet, serial I/O, parallel I/O, LCD displays, LEDs, switches, and additional general purpose I/O pins on headers that can be connected to external user hardware. The Altera DE2 board in Figure 5 supports SoPC design using the Nios II soft-core processor on the Cyclone II FPGA family.

The Xilinx XUP-V2P board in Figure 6 contains a Virtex II Pro family FPGA. It contains two PowerPC hard processor cores and can also be used to develop MicroBlaze soft-core processor designs [4]. The FPGAs on both boards contain integer hardware multiply circuits that can be used for DSP applications. A USB cable connected to a PC development system is typically used to download FPGA hardware configuration data and software to the board's memory devices. SoPC development tools also support remote debugging on the boards.

6 Embedded Operating Systems for SoPC Systems

Many embedded systems now require multitasking, scheduling, threads, and perhaps support for networking. In such cases, a commercial embedded operating system is typically used rather than developing a custom OS for individual products. Some FPGA vendors provide a tiny microkernel for their devices. Linux, Nucleus PLUS, NORTi,

Wind River VxWorks AE X, OSE RTOS, and KROS are available for Altera's SoPC systems through third party vendors. Linux, QNX Neutrino, Wind River, and uC/OS-II RTOS are available for Xilinx's SoPC systems [1, 4, 19].

For smaller designs that do not require full O/S support, IP cores and supporting software for basic communication and networking are provided. Third-party tools are also available with complete support for networking including full TCP/IP protocol stacks, USB, and Bluetooth communication.

License fees and legal agreements for the OS, networking support, and other IP cores can add to the overall cost of the SoPC system, and they need be carefully evaluated early in the design process. IP issues tend to be less involved when dealing directly with the major FPGA chip vendors; however, IP cores can still vary widely in cost from free to tens of thousands of dollars.

7 Two SoPC Design Examples

Two system design examples will now be presented as case studies that show some of the advantages and design tradeoffs present in SoPC technology. Both systems have been successfully implemented using current SoPC devices and tools. The first system uses an FPGA with Altera's soft-core Nios processor, and the second system uses an FPGA with Xilinx's hard-core PowerPC processor.

7.1 SoPC-Based Digital Autopilot Design

The miniature SoPC-based autopilot system seen in the photographs in Figure 7 is used for unmanned aerial vehicles. This system makes an interesting case study in SoPC design. The autopilot system continuously reads in sensor data that indicates attitude, altitude, speed, and location via GPS. It then uses this data to solve the control system motion equations for the aircraft and outputs updated signals to control the aircraft [3].

The flexibility of SoPC design allows the use of the FPGA's logic elements to interface to a wide range of sensors without the need for additional I/O support chips that would be requireed if a more traditional fixed-processor option was used. This results in an extremely small and low weight PCB design. An ASIC could be used instead of the FPGA, but the small production quantities needed for this system do not justify the greatly increased development time and cost needed for an ASIC.

Different types of small unmanned aircraft also require markedly different I/O standards for the control outputs. Some

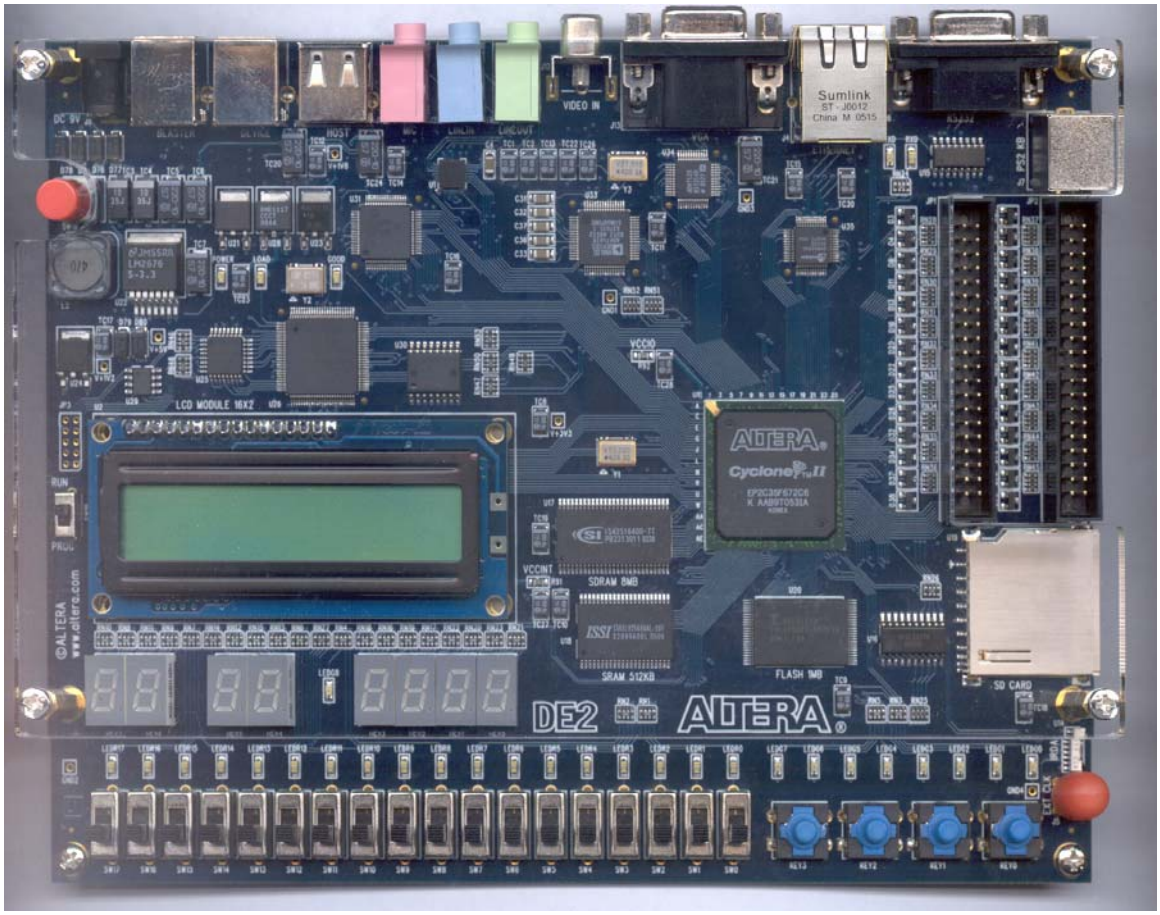


Figure 5: Altera SoPC board with Cyclone II FPGA that can run the Nios II soft-core processor

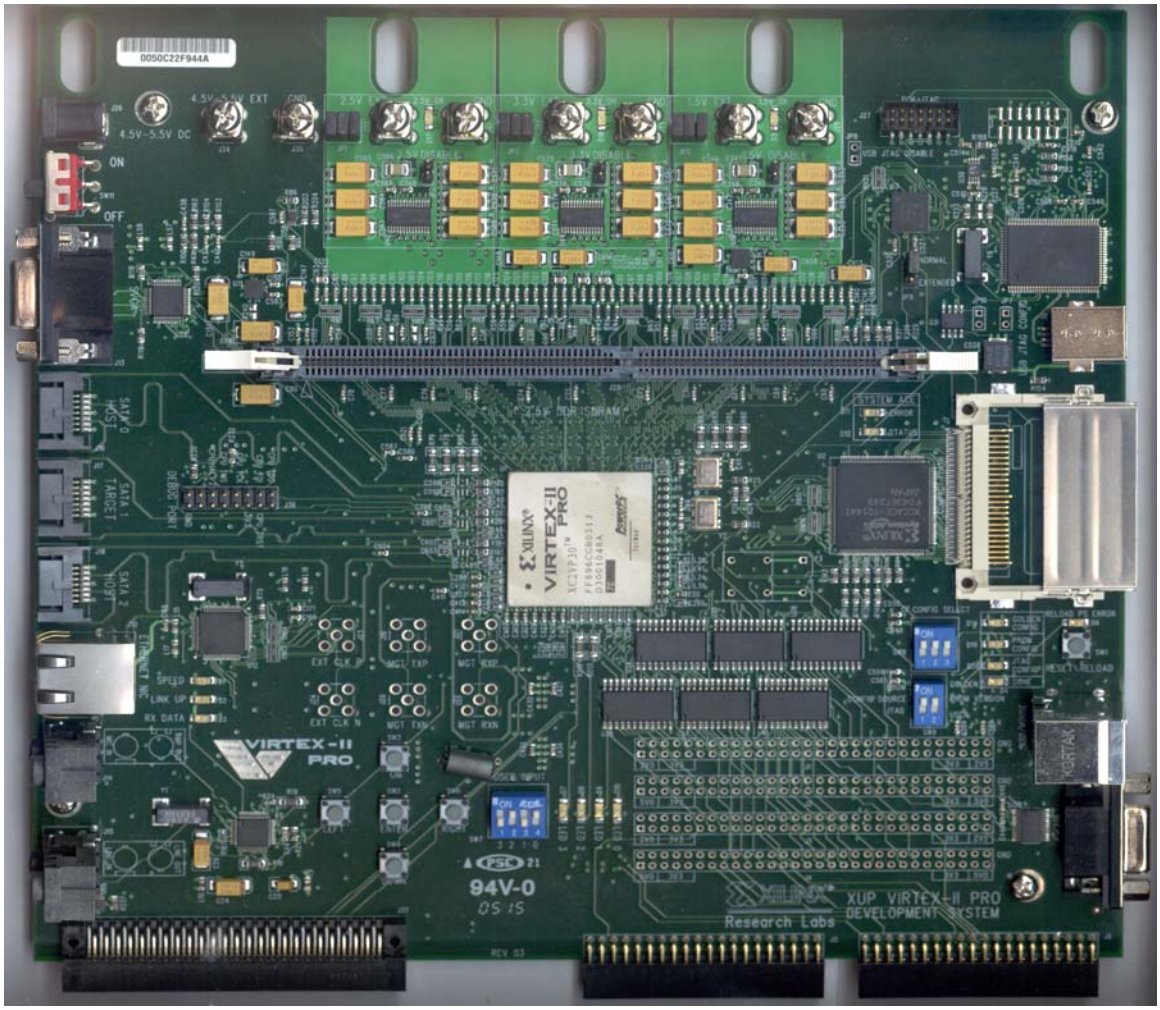
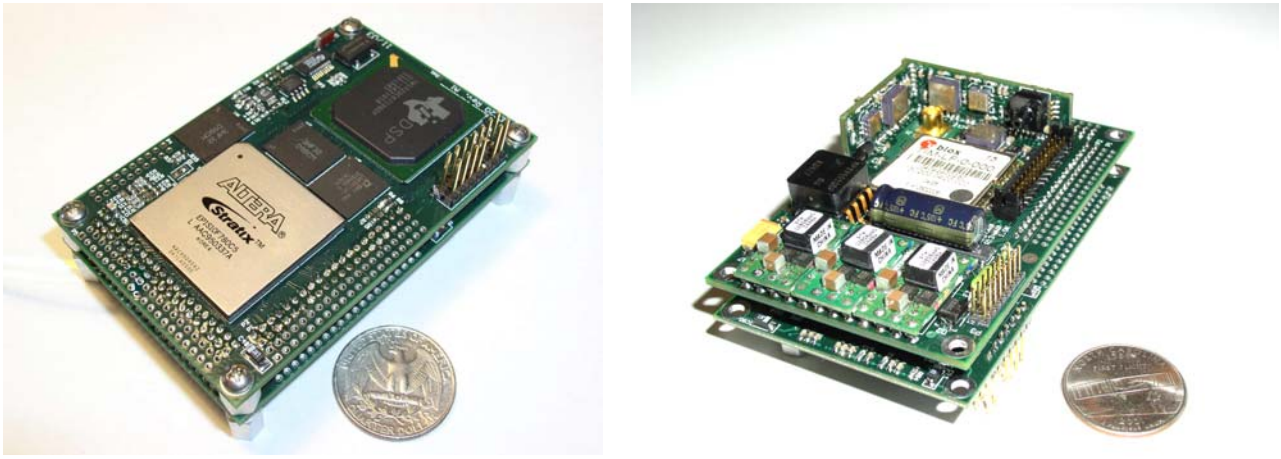


Figure 6: Xilinx SoPC board with Virtex II Pro FPGA with two PowerPC hard-core processors



Top board

Bottom board

Figure 7: Minature SoPC-based Autopilot System. Left: Top board contains an FPGA with a soft core Nios processor, SRAM, Flash, and a DSP processor. Right: Bottom board contains 3-axis MEMs gyros and accelerometers, GPS receiver, altitude sensor, airspeed sensor, and three ADCs. Photograph ©2004 courtesy of Henrik Christophersen, Georgia Institute of Technology Unmanned Aerial Research Facility

aircraft controls use serial interfaces, while others use PWM or even parallel I/O. Here again, the flexibility of using the FPGA's logic elements to implement the I/O interface is of great benefit. By varying the logic in the interface peripherals, the same programmable processor core and PCB board has been used to support a wide range of aircraft without any hardware changes to the PCB.

The autopilot system requires intensive floating-point calculations to solve the complex control system equations for the aircraft. While it would be possible to perform floating-point calculations using a larger FPGA, the decision was made to use a fixed-processor DSP chip for the intensive floating-point calculations. By offloading the algorithmic computations to a fixed DSP processor, the Nios II processor is primarily acting as an intelligent I/O processor for the system.

This partitioning of the system between a fixed-processor DSP and soft-core processor results in higher computational performance than using just an FPGA (with floating-point hardware logic) and higher interface flexibility than using just a fixed DSP processor in the system. However, new generations of FPGAs with DSP features such as hardware multipliers and floating-point IP cores are currently changing this set of design tradeoffs.

7.2 SoPC-Based Data Acquisition System

Another SoPC design example is a data acquisition system developed for a seismic landmine detection system [15]. In this system, a seismic shaker sends sound waves through the soil. An array of seismic sensors in contact with the soil measure the response. Since mines have vastly different acoustic properties than the surrounding soil, it is possible to determine the location of plastic mines buried in the soil through analysis of this response. (The sensor contact pressure and seismic excitation signals are well below the threshold for detonation of the mine.)

The full 32 by 32 seismic sensor array used in this system contains 1024 low-cost accelerometers each spaced several inches apart. The sample rate of a single sensor is a relatively modest 4-8 KHz. Simultaneously sampling a thousand sensors synchronized to a master clock at this rate for several seconds without dropping any samples poses a challenge for most current data acquisition systems. Current commercial off-the-shelf data acquisition solutions would be very large and cost prohibitive, since they do not scale well for such a large number of channels that must all be synchronized to a common sample clock signal.

The new data acquisition computer system that was developed for the seismic sensor array uses SoPC technology to achieve the desired goal of scalability. Each sensor in the array has its own low-cost 16-bit SPI Analog to Digital converter. Each row of sensors clocks its digital sample data into the FPGA using a high-speed 1-bit daisy chained SPI serial data stream. Serial interfaces work well on an FPGA since FPGAs implement high-speed shift registers efficiently. Serial interfaces also require fewer I/O pins, which can become a scarce resource when the system is implemented on a single chip. A custom hardware interface was designed using VHDL for each row of sensors. This hardware interface, implemented in the FPGA, performs serial to parallel conversion using a shift register and also contains a FIFO memory buffer to store data samples. All rows can work in parallel, since each row has its own independent FPGA hardware interface and shift register. Since all rows work in parallel, the system scales well for large numbers of sensors. A 256MB SDRAM memory module is used to buffer data samples, and a PowerPC hard-core processor in the FPGA runs a web server application program written in *C* that provides an easy-to-use web-based interface to the system. A standard Ethernet network interface is used to access the web interface and to download data sample files to a PC for analysis in Matlab.

The ability to add extensive custom hardware to interface thousands of sensors to a processor was ideally suited for this application. The SoPC solution was able to meet the desired performance, small packaging, and power consumption goals. A commercial microprocessor would require a large number of I/O expansion chips or perhaps even an external FPGA to interface to the sensor array. Also, since the production quantities are small, the high development and setup costs for an ASIC would likely not be recovered.

8 SoPC Design in Education

The SoPC approach is ideally suited for student projects. The vast hardware, software, and I/O flexibility of an SoPC development board allows a single board to be used and reused for a wide variety of student projects. Low-cost SoPC boards have been used for a number of team-based undergraduate student projects including image processing systems, robots, internet appliances, web and email servers, simple video games, and various multimedia systems.

SoPC developers and students need a background in C/C++ programming, digital logic, computer architecture, operating systems, and VHDL or Verilog to design complete SoPC systems. In most cases, this restricts SoPC projects to senior year undergraduate or graduate courses. Students can also concentrate on software development tasks, if they are provided with a hardware reference design that implements the features needed for their project on the SoPC development board. The same boards can also be used without the processor cores for digital logic hardware projects in lower level courses [13]. FPGA CAD tools are available free to schools from the major FPGA vendors, and SoPC development boards are available with educational discounts [1, 4, 7]. Tools, operating systems, and IP cores from smaller third-party vendors are still difficult for schools to obtain at a discount.

An example of a student SoPC project is seen in Figure 8. A student design team modified a hobbyist R/C truck to build an autonomous robot. A low-cost CMUCAM color vision system [14] is used to guide the vehicle down hallways. The path to follow in the hallway of the building is marked with colored tape. The solid-state camera detects and tracks color blobs and sends out data on a serial port to the processor. A program written in *C* running on the processor reads the tracking data

and determines how to control the speed and steer the vehicle. Like most R/C models, pulse width modulation (PWM) servo signals control the speed and steering. Students on this team decided to build hardware PWM controllers with additional FPGA logic rather than having several complex processor timer interrupt routines to generate the PWM signals for each servo. The processor writes the pulse width value to an I/O register on a parallel output port. State machine based PWM controllers written in VHDL read the I/O registers and generate the appropriate PWM timing signals for the servos. An additional serial interface was added to the processor's I/O system to interface the camera, and a parallel I/O port was used to interface an LCD status display. This was all accomplished by programming the FPGA hardware and writing C/C++ software without the need for additional external hardware, permanent modifications to the SoPC board, or add-on I/O boards.

The complexity of the senior-level student design projects has increased since introduction of the SoPC boards. Using a general purpose SoPC board saves both time and money. These boards have been successfully reused several semesters for a wide variety of projects. Newer generation SoPC tools are more user friendly, but there is still a significant learning curve for students to overcome, when using the complex commercial SoPC CAD tools. Students should successfully complete a system level tutorial during the first few weeks of the course to force them to start early and learn the tool flow. The flexibility of the new SoPC boards has worked very well for student design projects and it has improved the overall quality of the design projects [5, 8-9].

9 Conclusions

This paper has provided a brief overview of SoPC systems and designs. The SoPC approach should be considered for many embedded devices. In many cases, it can reduce development time and costs. The exceptions are low-end devices easily implemented on a low-cost single-chip microcontroller, devices produced in very large quantities, devices that need very high performance, and devices that require very low power consumption. Traditional processor and ASIC vendors have also seen the advantage of the SoPC approach using FPGAs for custom user logic. More information about specific SoPC devices, tools, and systems is available from manufacturers such as Altera, Xilinx, Cypress Semiconductor, Stretch Incorporated, and Tensilica [1, 4, 18].

Figure 8: Student robot project controlled by an SoPC board with a camera and PWM servos

References

[1] Altera Corporation, Embedded Processor Web Site, www.altera.com.

[2] H. Chang, L. R. Cooke, M. Hunt, G. Martin, A. McNelly, and L. Todd, *Surviving the SOC Revolution A Guide to Platform-Based Design*, Kluwer Academic Publishers, 1999.

[3] H. Christophersen, R. Pickell, J. Neidhoefer, A. Koller, S. Kannan, and E. Johnson, "A Compact Guidance, Navigation, and Control System for Unmanned Aerial Vehicles," to appear in *Journal of Aerospace Computing, Information, and Communication*, 2006.

[4] Digilent Inc, XUPV2P Reference Manual, www.digilentinc.com.

[5] ECE 4006 Senior Design Project Class Web Site, Georgia Institute of Technology, www.ece.gatech.edu/~hamblen/4006/projects.

[6] J. Fisher, P. Faraboschi, and C. Young, *Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools*, Morgan Kaufmann, 2004.

[7] T. S. Hall and J. O. Hamblen, "System-on-a-Programmable-Chip Development Platforms in the Classroom," *IEEE Transactions on Education*, 47(4):502-507, Nov. 2004.

[8] J. O. Hamblen, "Using Second Generation SoPC Boards for Student Design Projects" *Proceedings of the 2005 International Conference on Microelectronic Systems Education*, Anaheim, CA, pp. 69-70, June 2005.

[9] J. Hamblen and T. Hall, "Engaging Undergraduate Students with Robotic Design Projects," *Proceedings of the Second IEEE International Workshop on Electronic Design, Test and Applications*, Perth, Australia, pp. 140-145, January 28-30, 2004.

[10] J. O. Hamblen, T. S. Hall, and M. Furman, *Rapid Prototyping of Digital Systems Quartus II Edition*, Springer, August 2005.

[11] A. Jerraya, H. Tenhunen, and W. Wolf, "Multiprocessor Systems-on-Chips," S. Liebson and J. Kim, "Configurable Processors: A New Era in Chip Design," *IEEE Computer*, 38(7):51-59, July 2005.

[12] M. Mar, B. Sullam, and E. Blom, "An architecture for a configurable mixed-signal device," *IEEE J. Solid-State Circuits*, 38:565–568, Mar. 2003.

[13] K. Newman, J. O. Hamblen, and T. Hall, "An Introductory Digital Design Course Using a Low-cost Autonomous Robot," *IEEE Transactions on Education*, 45(3):289-296, August 2002.

[14] A. Rowe, C Rosenberg, and I. Nourbakhsh, "A Low Cost Embedded Color Vision System," IROS 2002 Proceedings, http://www-2.cs.cmu.edu/~cmucam/.

[15] W. Scott, J. Hamblen, J. Martin, and G. Larson, "Large Vibrometer Arrays for Seismic Landmine Detection," SPIE Defense and Security Symposium, Orlando, Florida, April 2006.

[16] D. Seguine, "Just add sensor - integrating analog and digital signal conditioning in a programmable system on chip," *Proceedings of IEEE Sensors*, 1:665–668, 2002.

[17] C. Snyder, "FPGA Processor cores get serious," in *Cahners Microprocessor Report*, http://www.MPRonline.com/, Sept. 2000.

[18] Stretch Incorporated, Software Configurable Processor Web Site, www.stretchinc.com.

[19] Xilinx Corporation, Embedded Processor Web Site, www.xilinx.com.

James O. Hamblen is a Professor in Electrical and Computer Engineering at Georgia Tech. His current research interests include rapid prototyping, high–speed parallel and VLSI computer architectures, computer–aided design, and reconfigurable computing. He received a PhD in Electrical Engineering from Georgia Tech, an MSEE from Purdue University, and a BEE from Georgia Tech. Prior to earning his PhD, he worked as a Systems Analyst for Texas Instruments in Austin, Texas, and as a Senior Engineer for Martin Marietta in Denver, Colorado.

Tyson S. Hall received the PhD, MSECE, and BSCMPE degrees in Electrical and Computer Engineering from the Georgia Institute of Technology in 2004, 2001, and 1999. He is currently an Assistant Professor in the School of Computing at Southern Adventist University in Collegedale, Tennessee. Dr. Hall's research interests include rapid prototyping of mixed-signal systems, cooperative analog/digital signal processing, reconfigurable computing, and embedded systems education.