AC 2007-2341: TRANSFORMING THE MICROPROCESSOR CLASS: EXPANDING LEARNING OBJECTIVES WITH SOFT CORE PROCESSORS

Lynne Slivovsky, California Polytechnic State University

Lynne Slivovsky received her B.S. in Computer and Electrical Engineering and her M.S. and Ph.D. in Electrical Engineering from Purdue University in 1992, 1993, and 2001, respectively. She worked with the Engineering Projects In Community Service (EPICS) Program from 2001 to 2003. In Fall 2003, she started a tenure-track assistant professor position in Electrical Engineering and Computer Engineering at California Polytechnic State University, San Luis Obispo. She received a Frontiers In Education New Faculty Fellow Award in 2003. In 2006, she was named the Hood Professor of Electrical Engineering. Her research is in the areas of haptics, human computer interaction, computer vision, and engineering education. In her free time, she enjoys mountaineering, kayaking, and photography.

Albert Liddicoat, California Polytechnic State University

Albert A. Liddicoat received his M.S. and Ph.D. degrees in Electrical Engineering and his M.S. degree in Engineering Management from Stanford University in 1996, 2002 and 1999, respectively. Dr. Liddicoat worked for IBM's Storage Technology Division from 1990 until 2002 where he held many positions in disk drive development including: servo system test and integration, ASIC development, system electronics and architecture, program management, and business line management. Currently, he is the Forbes Associate Professor and the Director of the Computer Engineering Program at Cal Poly State University in San Luis Obispo. His research interests include computer architecture, computer arithmetic, networks, and re-configurable computing.

Transforming the Microprocessor Class: Expanding Learning Objectives with Soft Core Processors

Abstract

The rapid evolution of semiconductor technology over the past four decades has fueled the information age and an era of ubiquitous computing. Furthermore, the exponential increase in the number of transistors available in integrated circuits has drastically changed the field of electrical and computer engineering. Computer Aided Design (CAD) tools allow systems engineers to work at higher-levels of abstraction to design increasingly more complex systems. Embedded system design has been a keystone course in electrical and computer engineering curricula. Following the migration from discrete components to programmable logic devices in introductory digital design courses we expect to see a similar, yet more selective, shift to the use of soft core processors in future microprocessor and embedded systems courses. Soft core processors and peripheral devices can readily be implemented on a programmable logic device, typically a Field Programmable Gate Array (FPGA), and can be customized with respect to system requirements. Off-the-shelf processors cannot offer a customized computer system or the ability to design user-specified hardware as part of a system-on-a-chip. These aspects are the most advantageous characteristics of the soft core approach to embedded systems. Students themselves will design their computing platform using only the necessary peripherals. They will define the memory system and assign addresses to memory mapped peripheral registers. They will analyze system performance based on hardware and software tradeoffs against a backdrop of the utilization of hardware resources, thus vastly increasing the design space they consider for their projects. In this paper we predict a shift in the pedagogical approach to teaching the microprocessor course from one that uses off the shelf processors to one that will include the teaching of the soft core processor. We support our claim by reviewing advances in the programmable logic industry from which these processors have emerged, outlining current soft core processor applications and trends in industry, detailing learning objectives for a soft core-based approach (patterned after the course we currently teach), and summarizing resources available to those interested in using soft core processors at their schools.

Introduction

The introductory microprocessor/microcontroller/embedded systems course (which we will refer to as the introductory embedded systems course for brevity) is integral to the electrical and computer engineering curriculum. Variations exist in processor, programming language, and textbook yet there are similarities in the topics covered and the labs performed[1]. Students are introduced to a processor, its architecture and instruction set, and the fundamentals to program and interface the processor to the outside world with D/A and A/D converters, for example. Students learn about memory mapped I/O, configure timers and serial interfaces (UARTs, SPI and I^2C), and design real-time systems with interrupt-based schemes. Historically, this course has used commercial off-the-shelf processors; ARM, Atmel AVR, and flavors of the Motorola and Freescale 68HCxx processors are commonplace.

Today, modern digital systems often include a combination of custom logic designed using hardware description languages and blocks that have been designed by a third party. The third party hardware blocks are often referred to as Intellectual Property (IP) cores. Many soft core processors have been designed using hardware description languages and implemented on programmable logic devices, typically FPGAs. Like their off-the-shelf counterparts, they have a teachable architecture and instruction set. They have on-chip memory, provided by FPGA block RAM. Sophisticated development tools configure the soft core processors with various on-chip peripheral cores, such as general purpose parallel ports, timer/counters, UARTs, and external memory controllers, resulting in a customized system-on-a-chip implementation that meets performance constraints while minimizing on-chip logic resources. Commercial off-the-shelf microcontrollers include a fixed set of peripheral devices that cannot be changed by the user and thus limit the design space of the digital system.

Following the migration from discrete components to programmable logic devices in introductory digital design courses we expect to see a similar, yet more selective, shift to the use of soft core processors in future microprocessor and embedded systems courses. Off-the-shelf processors cannot offer a customized computer system or the ability to design user-specified hardware as part of the system-on-a-chip. These aspects are the most advantageous characteristics of the soft core approach to embedded systems. The same FPGA development board can be used for many diverse design projects. For example, one student may be designing a distributed system and configure the soft core computer system to have 5 UARTS for communication with several embedded systems and Personal Computers. Another student may design a robotic arm and have the need to control 5 pulse width modulated servos. Most off-the-shelf microcontrollers include two or fewer UARTS and PWM outputs. Using the soft core computer system approach, students themselves will have access to a library of peripheral devices and they will design their computer platform using as many peripherals as needed. They will configure their computer system using controllers for instruction and data memories and the minimum set of peripheral devices needed for their system architecture. They will define the memory map by assigning address ranges to the memory blocks and peripheral hardware registers. They will analyze system performance based on hardware and software tradeoffs against a backdrop of the utilization of hardware resources thus vastly increasing the design space they consider for their projects.

Our claim in a shift from the commercial off-the-shelf microprocessor to a soft core processor in introductory embedded systems courses stems from a number of observations. It is interesting to note the overlap inherent in these observations and the parallelism that exists between the areas from which they emerge.

Technology Trends and the Digital Designer

Gordon Moore first observed an exponential growth rate in the number of transistors per integrated circuit[2]. This trend became know as Moore's Law and has held true for over four decades. Since Moore's observation, the number of transistors per integrated circuit has doubled every couple of years. Moore realized the technological impact that this trend would have on commercial products and in 1965 wrote, "With unit cost falling as the number of components per circuit rises, by 1975 economics may dictate squeezing as many as 65,000 components on a single silicon chip." Forty years ago, 1 billion transistors per integrated circuit was inconceivable.

This technology trend of an exponential increase in the number of transistors per die has fueled the growth in the electronics industry over the last forty years. Not only does this technology trend have a significant impact on commercial products but it also has a significant impact on digital system designers, the tools they use, and the level of abstraction in which digital designers work. Obviously we do not have an exponential increase in the number of engineers that are available each year to design integrated circuits. Therefore, to keep pace with this technology trend, engineers must become more productive as measured by the number of transistors they can design into a digital system per man year. In other words, engineers will constantly be driven to design digital systems at higher levels of abstraction because they simply cannot continue to design exponentially more complex digital systems using the same design methodologies and tools. Colleges and universities must therefore be educating their students to be competitive when they graduate, especially in today's global economy.

I. Increasing Levels of Abstractions

Both hardware and software design have followed a similar evolutionary path over the last forty years and may converge in the future into a single design methodology. Figure 1 provides a visual image of the levels of abstraction that digital designers may work at. As CAD tools improve, engineers are able to work at higher levels of abstraction and effectively become more productive. When integrated circuits were first produced engineers designed ICs at the transistor level. Transistors were connected in networks to implement simple logic functions and processors were initially programmed directly using machine code, both of which were tedious for large and complex designs.

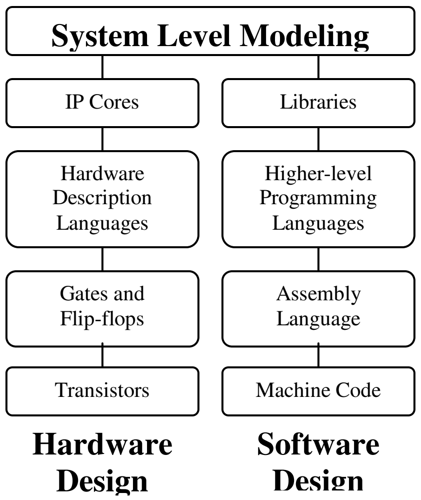


Figure 1. Hardware design and software programming levels of abstraction. Increased abstraction ultimately leads to hardware-software co-design.

Next, hardware designers created logic gates that allowed them to design Boolean functions using primitives such as AND, OR and INVERTER logic gates and memory devices such as flip-flops. Working at the logic gate and flip-flop level of abstraction increased the hardware designer's productivity. In software design, assembly programming was introduced. This allowed programmers to write assembly language programs with human readable mnemonics much easier than writing machine instructions.

The next leap in hardware design was the advent of hardware description languages. The two most common HDLs are VHDL and Verilog. Both allow designers to enter Boolean expressions directly in a symbolic form. Designers can simply code a correct unreduced Boolean function or truth table and let the HDL synthesis tool optimize the design for performance and low cost and generate the correct logic gate level design for the HDL code. Finite State Machines (FSMs) can be designed in HDLs using a higher level of abstraction by enumerating the states of a state machine and defining the state transitions. Similarly, higher-level languages such as C and Java were defined and compilers were written to translate the higher-level language programs into machine code. These languages made it much easier to implement software algorithms at a higher-level of abstraction that is independent of the processor instruction set. HDLs and higher-level programming languages again increased the productivity of hardware and software designers.

The next step up in the hardware design productivity came by reusing the designs of subcomponents. The subcomponents are considered intellectual property (IP) of the designer and may be licensed or freely distributed to be used as subcomponents in digital systems. Examples of Intellectual Property Cores include memory controllers, bus arbiters, serial communication devices, Ethernet modules, PCI controllers, Arithmetic units, and many more. View the OPENCORES website for some freely available IP Cores, http://www.opencores.org/. On the software side, software libraries are analogous to the hardware IP Cores. Software functions that may be of general use are often included in libraries so that every programmer does not have to write similar functions. Examples of software functions include printf, scanf, sine, and cosine to name a few. This hardware IP Core and software function reusability increases the productivity of hardware and software designers since they can reuse modules that have been developed for general use.

In order to design a digital system that includes a processor or microcontroller and custom logic the conventional approach is to design the hardware first and then when the hardware is available write the firmware or software for the system. Experienced programmers know that it is difficult to write code without being able to test and debug the code on the hardware platform as the code is written. Often, assumptions are made in the hardware architecture and design because the software is not available when the hardware is being designed. These assumptions lead to sub-optimal designs. Recent research in the area of system level modeling languages and hardware/software co-design has brought together both the hardware and software design phases into one development phase. A system level modeling language is used to specify the system. The modeling language does not require the designer to partition the system into a hardware-based design and a microcontroller-based design. Rather a universal language is used to define the system in its entirety. Then a CAD tool may be used to determine the performance critical operations and hardware is synthesized to implement the components of the design directly in hardware such that the system performance constraints are met. For example, in a real time video compression system the arithmetic transforms may be synthesized into hardware units that can perform parallel computation to compress the video image. On the other hand less performance critical system functions can be compiled to run on a processor. Specifying the system using a system level modeling language allows the system designer to work at yet a higher level of abstraction. In addition, this level of abstraction blurs the distinction between hardware design and software design and aims at producing an optimal system level design.

The exponential increase in the number of transistors per integrated circuit has held for the last forty years. Today’s best predictions expect the trend to continue into the foreseeable future. The challenge for the digital designer is to determine how to best use 1 billion or 10 billion transistors on a single integrated circuit. In addition, designers have had to adopt new design methodologies and ever more powerful computer aided design (CAD) tools so that they can effectively take advantage of this technology trend. This rapid advancement in technology drives the design methodology and in turn requires engineers to constantly learn new design techniques and tools. Digital designers are therefore required to acquire new skills and be committed to a career that includes livelong learning.

II. Advances in the Programmable Logic Industry

A brief history of the advent of programmable logic devices best begins with the advent of integrated circuits, Figure 2. In the late 1950s, the first integrated circuits which contained single transistors were manufactured. Texas Instruments introduced the first family of integrated circuits in 1962 to support the NASA space program. The TTL 74xx family of discrete parts used bipolar junction transistors with chips containing just a few logic gates. Over the last 40 years engineers have witnessed the introductions of dozens of IC families. Each successive family uses a faster and/or lower power technology than the one before, the most recent being high-speed low-power CMOS devices.

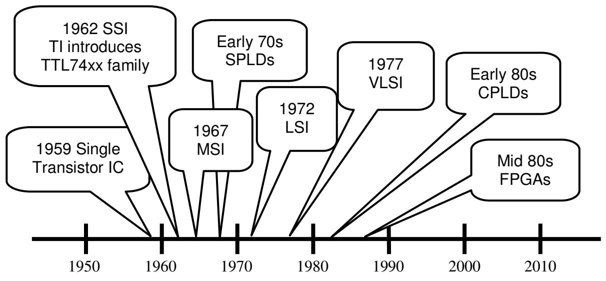


Figure 2: The Evolution of Integrated Circuits

The first programmable logic devices, now typically referred to as Simple PLDs (SPLDs), arrived in the early 1970s and were based on a programmable AND-OR array structure. Complex PLDs (CPLDs), devices that contained multiple SPLDs on a single chip, appeared in the early 1980s. Later that decade, Field Programmable Gate Arrays (FPGAs) were introduced. These are devices whose underlying programmable architecture implements multi-level logic functions through the use of Look Up Tables (LUTs). The FPGA fabric contains LUTs, programmable interconnects to route signals, I/O blocks to interface to off-chip components, and on-chip Block RAM for memory. Higher-end FPGAs now offer hardware multipliers for signal processing applications and hard core (in silicon) processors. The later provide capabilities for complete system-on-a-chip designs.

Coincidentally, the Very-High Speed Integrated Circuit (VHSIC) program was responsible for the specification of the first hardware description language in the early 1980s. This specification became known as the VHSIC Hardware Description Language (VHDL) and was formally adopted as the IEEE standard 1076 in 1987 by the IEEE.

The cost of early programmable devices limited their use when compared to the cost associated with an equivalent number of discrete parts. Over the last 20 years, advances in process technology have driven production costs of programmable devices down. Today, CPLDs and FPGAs often provide advantages over discrete components in flexibility, board area savings, operating performance, reliability, time to market, power consumption, EMI, and design security and hardware reusability. Because of this, they are found in numerous commercial products ranging from home entertainment systems to planetary rovers. Their widespread use necessitates inclusion in the digital curriculum with attention paid to the advantages stated above.

III. Implementations of Digital Systems

Today digital systems are designed using one of many drastically different implementation techniques. Each of these implementation techniques has a different set of advantages and disadvantages in the area of cost, performance, power consumption, design time, manufacturing time, and flexibility. Programmable logic is often used for hardware prototyping, applications that require concurrent hardware and software development, and for applications where it is not economically beneficial to design a custom integrated circuit (IC). Systems implemented using programmable logic often benefit from the performance advantage of a hardware implementation as compared to systems implemented using a general purpose processor or microcontroller while avoiding the large initial investment that custom integrated circuits require.

IV. Shifting Pedagogy in Digital Education

Following the advent of PLDs, digital designers migrated from discrete logic to SPLDs and CPLDs. A pedagogical shift was seen in the Digital Design course in the 1990s. What had been traditionally taught using SSI and MSI components in circuits designed, unbelievably at first by hand and later, by schematic with CAD tools slowly gave way to the use of PLDs and hardware description languages. Today, most digital design courses offer a mix of the two. Students first implement designs with discrete components and then shift to implementations on SPLDs, CPLDs, and FPGAs using HDLs. The use of designing circuits with an HDL on a reconfigurable device has added benefits in reducing errors and time spent troubleshooting due to mistakes in wiring, for example. Additionally, students are now introduced to the concepts of reusability and modular design.

The shift in digital design education can be examined with respect to the Learning Style Model of Felder and Silverman[5]. Of their five dimensions, Perception, Processing, Input, Organization, and Understanding[6,7], have shown perception to be the most applicable. The authors state that a shift from discrete components to PLDs is advantageous for *intuitive* learners whereas the ability to touch ICs and physically trace signals through circuits is advantageous to *sensing* learners. With most classes using a mix of discrete and programmable components, both learning styles are addressed.

With an increase in the use of FPGAs in industry, Wolf notes in[8] the importance of an understanding of VLSI concepts when designing large and complex digital systems on FPGAs. This includes the application of concepts from circuits and VLSI to digital design which is often taught away from physical devices and in the abstract Boolean logic level. A logical follow-on to the work in[6,7] would be to examine this with respect to Felder and Silverman's Understanding dimension.

In the introductory embedded systems course, processor, programming language, development environment, and textbook are selected with learning objectives and fit within the curriculum in mind[1]. Recently, Bloom's Taxonomy and problem-based learning concepts have directly been applied to these courses. The 3C5I Model[9] incorporates the two. The authors present a context in which to apply the 5I's (Introduction, Illustration, Instruction, Investigation, and Implementation) to the 3C's (Concepts with Courses within a Curriculum) in a thematic project throughout their course.

Building on the fundamentals taught in introductory level courses, Fisher and Baladi[10] present a framework for embedded computer system design. They identify design criteria of real-time requirements, fault-tolerance, testability requirements, time-to-market, and product life cycle. They demonstrate the need for a hierarchical design process, judicious selection of a system specification language, reuse of IP, and hardware software co-design. This is in concert with[11] that also addresses technology trends.

Common to digital design and introductory and advanced embedded systems courses are the needs for students to learn at ever increasing levels of abstraction and to incorporate holistic design methodologies.

Soft Core Processor Pedagogy

Technology trends in digital design have followed an exponential increase in the number of transistors on an integrated circuit. This has led to engineers designing at increased levels of abstraction: from transistor to hardware description language, paralleling the development of programming from that of machine code to high level language. The development and use of reconfigurable logic devices, CPLDs and FPGAs, has progressed along similar lines. Engineers now design digital systems in hardware description languages, providing modularity and portability unseen with discrete components.

Pedagogical shifts in the digital curriculum began with introductory digital design courses which now include exposure to and design with CPLDs, FPGAs, and HDLs. Considerations in flexibility, board area savings, operating performance, reliability, time to market, power consumption, EMI, and design security are part of the design process. Sustainability is addressed by hardware reuse through the reprogrammability of most devices.

With traditional introductory embedded systems courses, many tradeoffs, while obvious at a superficial level of understanding, are at times too abstract for students with little experience synthesizing and applying concepts from prior, and seemingly unrelated, courses such as linear circuit analysis and digital integrated circuits. The use of a soft core processor on an FPGA provides an optimal platform to expand these courses. Though designed in an HDL and implemented on an FPGA, learning objectives for introductory embedded systems courses are still met. Traditional topics include:

- Processor architecture and instruction set
- Programming fundamentals
- Memory mapped I/O
- Common serial interfaces (e.g., UARTs, SPI and I2C)
- Timer and Counters (e.g., PWM)
- Interfacing (e.g., buttons, switches, LEDs, LCD displays, A/D, D/A)
- Polling vs. interrupt schemes
- Real-time systems

Unique to the soft core approach is the ability to address the trends and needs outlined in this paper. Soft core processors offer a customized computer system and the ability to design user-specified hardware as part of the system-on-a-chip. The same FPGA and accompanying development board can be used for many diverse design projects. Students will configure their computer system using controllers for instruction and data memories and the minimum set of peripheral devices needed for their system architecture. They will define the memory map by assigning address ranges to the memory blocks and peripheral hardware registers.

On-chip peripherals and on-board memory configurations are limited for off-the-shelf processors and development boards. The selection of a processor and development board by an instructor is subject to tradeoffs between capability and cost and place real limitations on the type of projects that students can implement with supplied resources. Using the soft core computer system approach, students themselves will have access to a library of peripheral devices and they will design their computer platform using as many peripherals as needed. For example, one student may be designing a distributed system and configure the soft core computer system to have a number of UARTS for communication with several embedded systems in an instant messaging project. Another student may be designing a video game that uses a VGA controller to interface with a monitor. Students will also be able to design their own peripherals using a hardware description language to incorporate into their custom computer system.

Students will make choices between software algorithms running on the soft core processor and hardware implementations on the FPGA fabric for components in their designs. They are, in fact, running a stored-program system on a processor implemented on a device that could also implement a hardware-based solution. Yet, performance tradeoffs are not limited to these two choices. Propagation delays of signals routed on a programmable interconnect are greater than that of ASIC or full custom implementations. This will require students to incorporate material from digital integrated circuits into their decision making processes. They will impose constraints among multiple dimensions, for example, the hardware spectrum (from transistor to IP Core) and the digital system spectrum (from stored program to fully custom IC). They will analyze system performance based on hardware and software tradeoffs against a backdrop of the utilization of hardware resources thus vastly increasing the design space they consider for their projects.

New content in introductory embedded systems courses using a soft core approach can be summarized as:

- Increased design space considerations
- Hardware software co-design
- Increased exposure to hardware
- Hierarchical, modular, reconfigurable, and reusable design processes
- Increased cost, performance, and power considerations
- Exposure to the benefits of PLDs and FPGAs: flexibility, board area savings, operating performance, reliability, time to market, power consumption, EMI, and design security
- Customized peripheral selection and student-defined memory mapped I/O
- Custom peripheral design – parallel execution for speed optimization

Off-the-shelf processors can offer neither a customized computer system nor the ability to design user-specified hardware as part of a system-on-a-chip. A soft core processor approach expands the design space in which students develop embedded systems incorporating numerous design decisions along the way.

An Introductory Microprocessor Class Using Soft Core Processors on FPGAs

We teach an introductory class on microprocessors that uses the Xilinx Embedded Developer's Kit with the MicroBlaze processor. MicroBlaze is a 32-bit RISC processor that is implemented in the programmable logic (lookup tables) on a Xilinx Spartan 2e FPGA. The course covers the design, implementation and testing of programmable logic microprocessor-based systems. Hardware/Software tradeoffs (such as timing analysis and power considerations), system economics of programmable logic and microprocessor-based system design and interfacing hardware components (such as ADCs/DACs, sensors, and transducers) are the main areas of the class.

The class begins with a study of the internal architectures of CPLDs and FPGAs with special attention paid to performance of programmable logic based systems compared to ASIC implementations. Students examine propagation delay, area, and cost tradeoffs between the two.

With an understanding of hardware-based advantages for system implementation, the class moves on to study the MicroBlaze soft core processor. In classes that use commercial off the shelf processors, this would be the time at which the memory map of registers and peripherals are presented. Being a soft core processor, students not only learn about memory mapped I/O well but also construct the memory map of their hardware system. This brings them a level closer to the hardware and introduces the discussion of minimum hardware resources necessary for system implementation.

Figure 5 depicts a basic MicroBlaze hardware system and the first computer system that students construct. It consists of the MicroBlaze CPU Core, two memory controller cores, on-chip block RAM for memory, and an 8-bit general purpose I/O (GPIO) core configured as an output port. The hardware system is developed using the Embedded Developer's Kit (EDK) from Xilinx. Once built, the EDK generates a downloadable BIT file for the FPGA target device that includes both the synthesized VHDL code from the soft cores and the binary executable code that has been compiled for execution on the MicroBlaze processor.

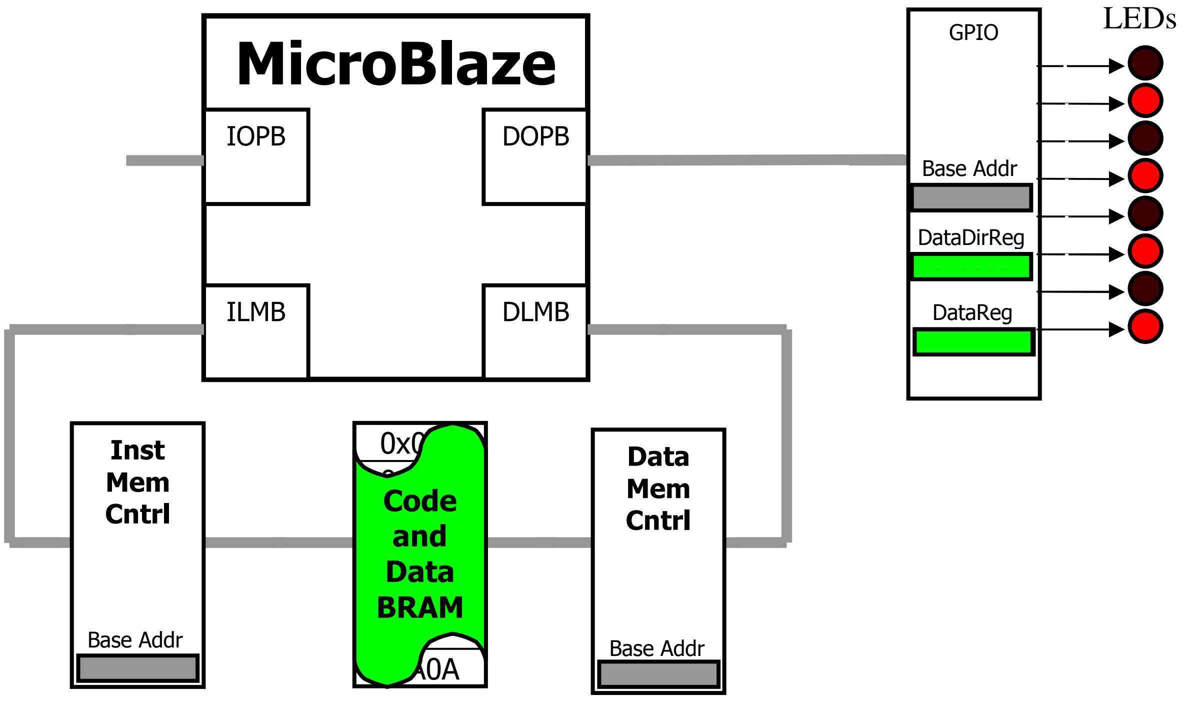
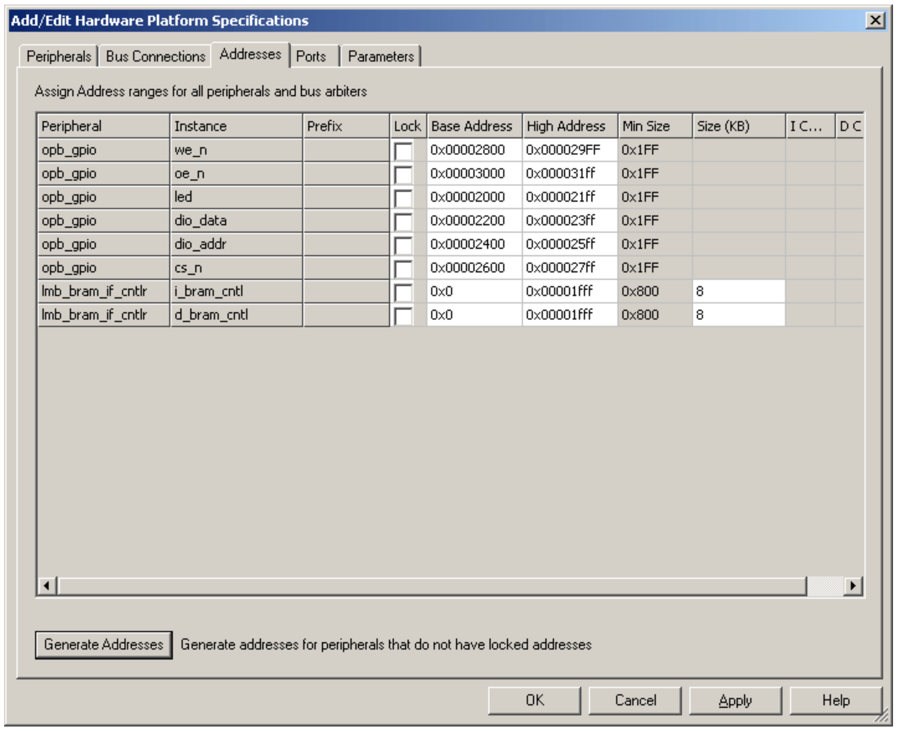


Figure 5: Standard MicroBlaze hardware configuration of MicroBlaze CPU Core, Data and Instruction Memory Controller Cores, dual-port Block RAM, and one OPB General Purpose I/O Core driving eight LEDs.

The MicroBlaze CPU supports four buses: Instruction Local Memory Bus (ILMB) and Data Local Memory Bus (DLMB) to interface to on-chip memory for program and data storage, and Instruction On-chip Preipheral Bus (IOPB) and Data On-chip Peripheral Bus (DOPB) to interface with peripherals. Since the Spartan 2e has dual-port Block RAM, the Harvard architecture of the MicroBlaze is configured in what resembles a Von Neuman architecture. Configuring the memory controllers is accomplished by making proper bus connections and assigning the base address from which to index both instructions and data in memory. The GPIO core is connected to the DOPB as a slave. It contains two registers: a Data Direction Register set to indicate whether the port is acting as an input port or an output port and a Data Register to write out or read in data. When the Data Direction Register contains 0x00, the GPIO is configured as an output. When it contains a 1 at each bit position (0xFF), it is configured as an input port. The GPIO is memory mapped in the MicroBlaze address space by assigning a base address and minimum size (in bytes) to the device as shown in Figure 6. Writing to a memory mapped register in C code is done with a low level driver XIo_Out32(*address*, *data*); and reading from a memory mapped register is accomplished by assigning the return value of a call to XIo_In32(*address*) to a variable. In both cases, *address* is the memory mapped address of the register. One point of importance to note is that the GPIO cores, as well as most all other peripheral cores, have a configurable width up to the 32 bits.



Assign Address ranges for all peripherals and bus arbiters

Peripheral	Instance	Prefix	Lock	Base Address	High Address	Min Size	Size (KB)	I C...	D C
opb_gpio	we_n		☐	0x00002800	0x000029FF	0x1FF			
opb_gpio	oe_n		☐	0x00003000	0x000031ff	0x1FF			
opb_gpio	led		☐	0x00002000	0x000021ff	0x1FF			
opb_gpio	dio_data		☐	0x00002200	0x000023ff	0x1FF			
opb_gpio	dio_addr		☐	0x00002400	0x000025ff	0x1FF			
opb_gpio	cs_n		☐	0x00002600	0x000027ff	0x1FF			
lmb_bram_if_cntlr	i_bram_cntl		☐	0x0	0x00001fff	0x800	8		
lmb_bram_if_cntlr	d_bram_cntl		☐	0x0	0x00001fff	0x800	8		

Generate addresses for peripherals that do not have locked addresses

Figure 6: User-defined memory map is done while building the MicroBlaze hardware system.

Once student have mastered the basic hardware configuration with a number of GPIOs communicating to external devices, more advanced soft core peripherals are introduced. The OPB Timer/Counter Core consists of two independently programmable timer/counters that can be configured to generate timing intervals, measure timing intervals, or by using the two together in pulse width modulation mode. Each timer has a Control/Status Register (read/write), and Load Register (read/write) and a Count Register (read only), as shown in Figure 7. Again, the peripheral is configured as a slave on the DOPB and memory mapped to an available portion of MicroBlaze's address space.

In lab, our students use the OPB Timer Core in two experiments. The first one is the implementation of a digital clock using an interrupt scheme. The core is configured to generate interrupts by setting appropriate bits in the Control/Status Register. In the second experiment, students implement a Digital Function Generator using the timer core and off-chip digital to analog converters. Their function generator systems produce sine waves, sawtooth waves, and square waves whose frequencies vary between 100 Hz and 1 kHz in steps of 100 Hz. In their implementations students must make decisions on tradeoffs between hardware and software utilization for speed, accuracy and hardware resource utilization.

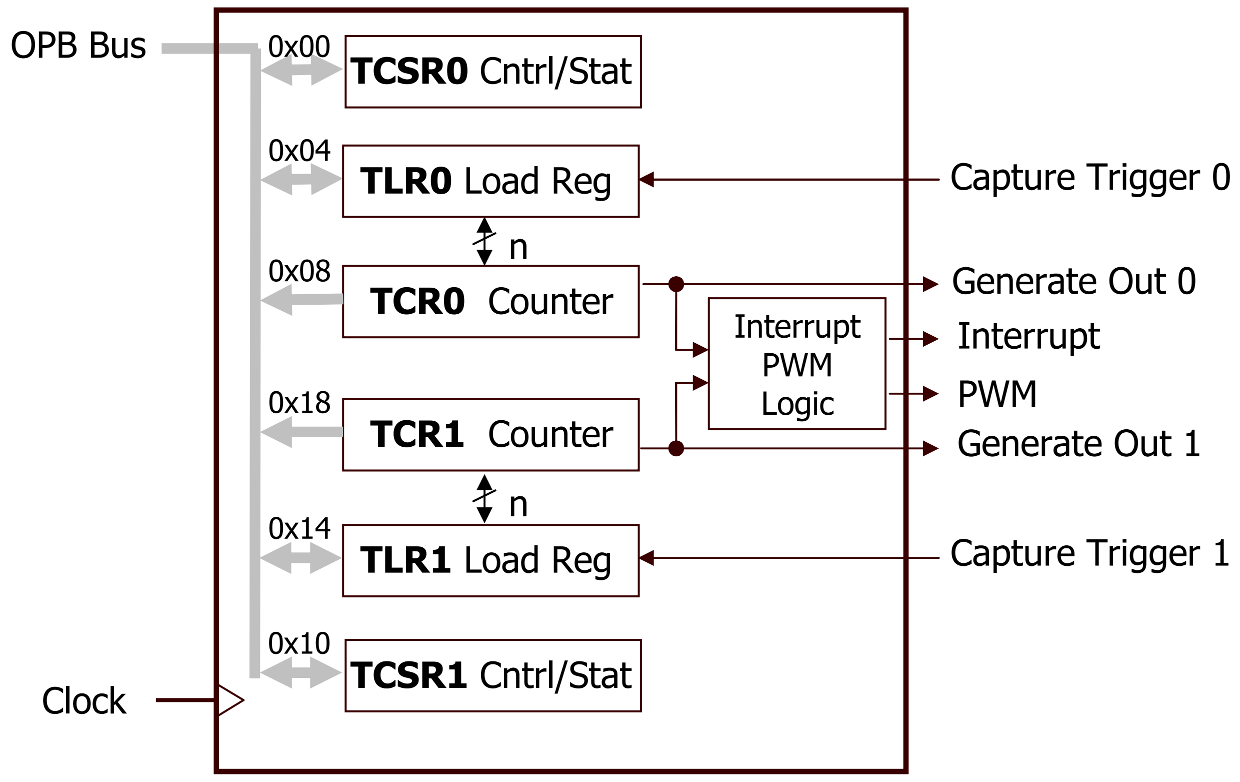


Figure 7: Xilinx OPB Timer Core

The OPB UART Lite Core is studied for asynchronous serial communication between devices. The peripheral core has Receive and Transmit I/O, one Receive Data Register, one Transmit Data Register, a Control Register and a Status Register. Unique to the core, as compared to traditionally used microprocessors, is the ability to include 16 word first-in first-out (FIFO) queues between the data registers and their respective I/O. This option is configured in EDK while the hardware system is being designed. Figure 8 depicts a UART Lite core connected to the DOPB. The "Lite" part of the core comes from the fact that all communication parameters (baud rate, number of data bits, parity on/off, parity odd/even) are configured in the development environment and cannot be changed once the hardware system is synthesized. This yields a much lower resource utilization at the cost of runtime flexibility. If the ability to change these parameters is needed, a fully programmable UART IP core is available. The UART Lite core can be configured to use a polling scheme or an interrupt driven scheme to receive and transmit characters. Like the other cores mentioned, this one is memory mapped in the development environment.

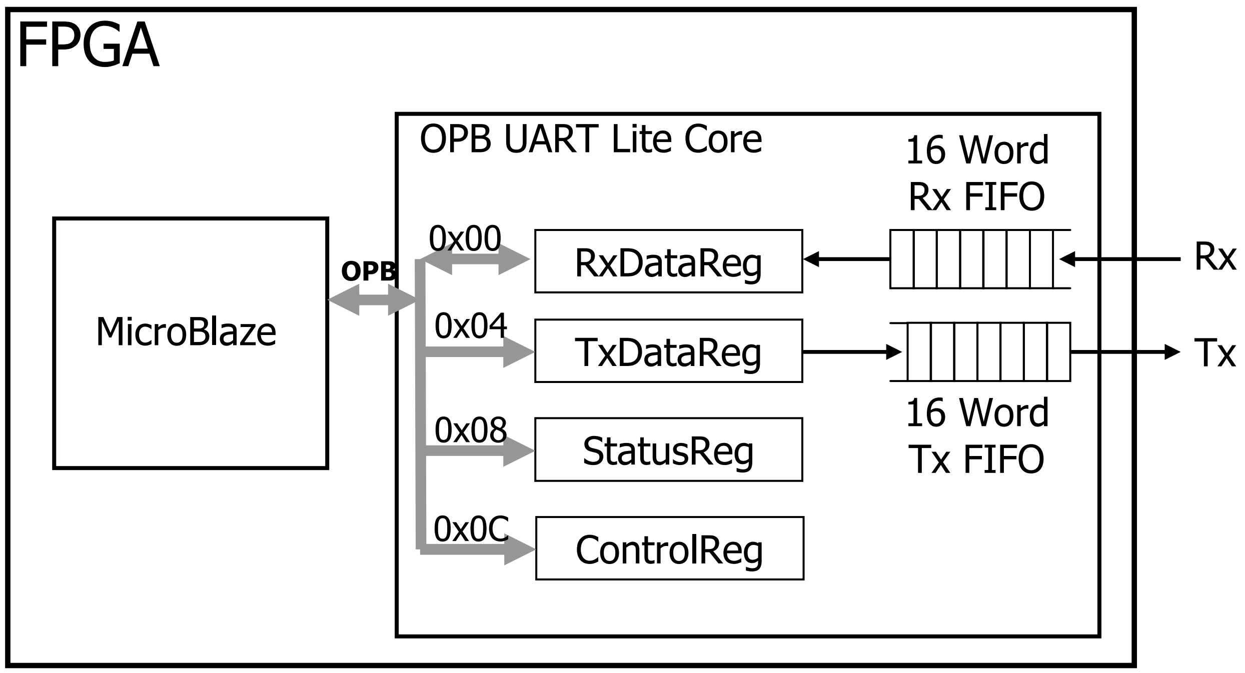


Figure 8: Xilinx OPB UART Lite Core

In addition to the cores just discussed, Interrupt Controller, External Memory Controller and Serial Peripheral Interface cores are also presented in class. Students complete a final design project in lab in which they are required to use a new core, use a prior core in a new mode, or develop their own core. The use of soft cores in the laboratory and the closeness to the hardware implementation required by memory mapping cores and building a hardware system shed new light on the microcontroller architecture for students.

Resources for the Instructor

Soft core processors and accompanying embedded development software packages are available from the two leading programmable logic manufacturers. Xilinx Inc. offers two soft core processors, the 8-bit PicoBlaze and the 32-bit MicroBlaze, both of which are RISC machines. Altera Corp. offers three variations of their 32-bit soft processor the Nios II. Both companies have university programs that offer generous support in outfitting laboratories with programmable logic solutions.

Additional soft core processors and peripheral cores based on the architecture and instruction sets of commercial off-the-shelf processors can be found through third-party vendors such as the DF6811CPU microcontroller CPU, based on the 8-bit Motorola 6811, from Digital Core Design. Systems may be customized with IP cores included with the soft processor developers kits or from a number of free and licensed cores from third-party vendors. Opencores.org has an extensive repository of freely available open source cores. For those interested in real-time operating system (RTOS) designs, the Nucleus RTOS by Accelerated Technology (a division of mentor graphics) is available for the Xilinx MicroBlaze processor, among others.

Conclusion

We expect to witness increased use of soft core processors in introductory embedded systems courses. Our claim in a shift from the commercial off-the-shelf microprocessor to a soft core processor is supported by a number of observations in the semiconductor industry and digital education pedagogies. Trends indicate the need for design at increasing levels of abstraction, for hardware software co-design, for the inclusion of constraints such as performance, cost, and power consumption in the design process. Unifying concepts from circuits to embedded systems can be accomplished with a soft core processor approach to digital system design.

References

1. Broberg, H.L. and Thompson, E., "Selection of Processor, Language, and Labs in Introductory Microprocessor/Microcontroller Courses", Proc. 2005 ASEE Annual Conference & Exhibition, Portland, OR, June 12-15, 2005.
2. Moore, Gordon, *Cramming More Components onto Integrated Circuits*, Electronics, Volume 38, Number 8, April 1965.
3. Semiconductor Industry Association, *International Technology Roadmap for Semiconductors 2003*, http://www.sia-online.org.
4. Liddicoat, A.A. and Slivovsky, L.A., "Programmable Logic" in *The Electrical Engineering Handbook, 3rd Ed.: Computers, Software Engineering, and Digital Devices*, Richard C. Dorf Ed., Boca Raton, FL.: Taylor & Francis, 2006.
5. Felder, R.M., & Silverman, L.K. "Learning styles and teaching styles in engineering education". Engineering Education, 78(7), 674-681, 1998.
6. Nickels, K., "Pros and Cons of Replacing Discrete Logic with Programmable Logic in Introductory Digital Logic Courses", Proc. 2000 ASEE Annual Conference and Exhibition, St. Louis, MO, June 18-21, 2000.
7. Nickels, K., Aminian, F. and Giolma, J.P., "Bridging the Gap Between Discrete and Programmable Logic in Introductory Digital Logic Laboratories", Proc. 2001 ASEE Annual Conference & Exhibition, Albuquerque, NM, June 24-27, 2001.
8. Wolf, W., *FPGA Based System Design*, Upper Saddle River, N.J.: Prentice-Hall, 2004.
9. Striegel, A. and Rover, D.T., "Problem-based Learning in an Introductory Computer-Engineering Course", Proc. 32nd ASEE/IEEE Frontiers in Education Conference, Boston, MA, Nov. 2002.
10. Fisher, P.D. and Baladi, M., "Embedded Computer System Design: A Framework", Proc. 2004 ASEE Annual Conference & Exhibition, Salt Lake City, UT, June 20-23, 2004.
11. Wolf, W., "Rethinking Embedded Microprocessor Education", Proc. 2001 ASEE Annual Conference & Exhibition, Albuquerque, NM, June 24-27, 2001.