AC 2012-4835: HARD CORE VS. SOFT CORE: A DEBATE

Dr. Antonio Francisco Mondragon, Rochester Institute of Technology

Antonio F. Mondragon-Torres received a B.Sc. degree with honors from Universidad Iberoamericana, Mexico City, Mexico, a M.Sc. degree from Universidad Nacional Autonoma de Mexico, Mexico City, Mexico, and a Ph.D. degree (as a Fullbright-CONACYT scholarship recipient) from Texas A&M University, College Station; all degrees in electrical engineering in 1990, 1996, and 2002, respectively. From 1988 to 1995, he worked in a telecommunications company TVSCOM, Mexico City, Mexico, designing teletext products, first as a Design Engineer and later as a Design Manager. In 1995, he joined the Mechanical and Electrical Department, Universidad Iberoamericana, as an Associate Professor. From 2002 through 2008, he was with the DSPS R&D Center's Mobile Wireless Communications Technology branch, Texas Instruments Dallas, Texas, and in 2008, he moved to the nanoMeter Analog Integration Wireless branch, where he worked as Analog IP verification technical lead. In 2009, he worked for Intel Guadalajara, Design Center in Mexico as Front-End/Back-End Technical Lead. In 2009, he joined the Electrical, Computer, and Telecommunications Engineering Technology Department at the Rochester Institute of technology, where he currently is a tenure-track Assistant Professor. His research interests include analog and digital integrated circuit implementation of communications systems and system-on-a-chip methodologies.

Ms. Jeanne Christman, Rochester Institute of Technology

Hard Core vs. Soft Core: A Debate

1. Abstract

Today's Computer Engineering Technology students have the opportunity to learn embedded design in both physical and virtual environments. Current technologies allow for such a high level of integration that digital and embedded designs can no longer be implemented using discrete components that are easily accessible to students for prototyping and debugging. Educational platforms currently available are in the form of microcontroller populated boards (hard core processors) or programmable logic device boards. In the later, students can instantiate a configurable, soft core processor comparable to the one provided in the former. This leaves educators with two distinct options for teaching embedded systems and low level programming courses (Note: there can be hard core processors within a programmable logic device, however this paper is referring to a hard core processor as a stand-alone component).

This paper is a dialogue between two faculty members, one defending design using hard components, assembly and laboratory testing, and the other using soft components, simulation and verification. The objective of the paper is to highlight the tradeoffs that both methodologies offer and the equilibrium that needs to be reached in order for students to receive the maximum exposure to both domains. Some of the tradeoffs debated are in terms of: environment, visibility to internal signal behavior, testability, design flexibility, cost and availability, power consumption, student learning curve, industry training, hardware-software partitioning, and student's engagement.

While evaluating the advantages and disadvantages of both approaches to teaching embedded design, the underlying goal that is always considered is that students become proficient designers while obtaining the skills that are current by today's industry standards.

2. Introduction

In current digital hardware curricula there is a tendency to use programmable logic devices as early as freshman year in introductory and fundamental courses. Many of the basic digital design textbooks such as "Digital systems: principles and applications" [1], propose the use of hardware description languages concurrent with the introduction of hardware equivalent logic gates to learn the fundamentals of logic design. This integrated approach presents both advantages and disadvantages that can enhance or impede the student learning process.

One of the advantages of using programmable logic on a virtual development platform is that the complexity of a student's design has the potential to explode. Except in their earliest digital fundamentals classes, today's students rarely deal with systems containing only a couple dozen gates or components. Through programmable logic devices, they have hundreds, thousands and even millions of gates at their disposal which they can use to create robust modern designs.

Not only does a programmable logic board provide students with millions of gates for digital design, it often includes a dedicated or configurable microprocessor core. With its potential to be used for introductory digital design, microprocessor programming and interfacing as well as complex custom embedded systems, a programmable platform can be a one solution fits all for a computer engineering technology curriculum.

With what seems to be the perfect solution of platforms that could potentially be used throughout the engineering technology curriculum, why do we still want to teach students to use discrete components such as gates and microcontrollers? The answer is that students need a variety of skills at different levels. One disadvantage of using programmable logic boards is that students are no longer able to prototype their designs using discrete components on a breadboard or a wire-wrapped board like previous generations of digital design students were able to. This is due to the fact that components today are rarely available on packages that can be assembled in such a manner. If students are not exposed to hardware prototyping in their undergraduate studies, they reach the graduate level and even the workplace lacking the ability to perform hardware debug and testing; critical skills for product development. Additionally, since the students are not used to building prototypes, their assembly skills are deficient and even a simple connection of less than ten wires can end up unstructured and difficult to debug.

Faculty are often in the position of having to decide which approach to use to teach an embedded systems design class. In this paper a position will be taken on each side, debating the merits of each type of system. One side will defend hardware design, programming, prototyping and testing, while the other will defend the programmable, single platform approach for design, simulation, programming and verification.

3. Challenges with Current Systems

When most of the current generation's instructors were students, more prototyping and hardware debugging was required and feasible. Simulations were sometimes not required and often not available so a "burn & crash" methodology was regularly followed, with testing not occurring until after the hardware was built. Hardware measurement equipment such as oscilloscopes and logic analyzers were indispensable in the design process. It is becoming increasingly more difficult to give today's students experience with this equipment since everything is now embedded within the development platform. Interfacing skills are being lost as more and more hardware is prebuilt. Three main challenges that affect the way embedded design is currently taught are miniaturization, surface mount and integration.

As technology has advanced, it is now possible to have all components required to form a complete microprocessor system on a single silicon die. This creates the problem that students are not able to visualize how all parts such as memory and peripherals are interconnected and what glue logic is required to have a complete system working. The best that can be done is to show a block diagram of how everything has already been connected. However, the chance to design it, connect it wrong and learn from the process is lost.

The technology has not only incorporated all components into a single die, but into a single package as well. These modern packages are so small that students are unable to solder them by themselves unless previously trained in surface mount technology (SMT) soldering. This means that the components can no longer be used on a protoboard for student experimentation and designs must rely only on the interconnections available on the development board. Again, the learning opportunity in selecting an incorrect configuration is no longer present.

When students today are introduced to the laboratory material, they are presented with a fixed board with all the interfaces required for their laboratory exercises in place. LEDs, switches,

pushbuttons, displays, serial ports, general purpose input/output ports are at fixed locations. With these interfaces fixed in the memory map it is difficult for students to rearrange, extend or alter the design configuration. While this may actually shorten the learning curve in the early courses, it can hinder design innovation in advanced courses.

4. Advantages of current systems

Although, it would appear that there are many disadvantages to today's embedded platforms, the reality is not as bleak as it may appear. The knowledge that students are acquiring may be different than in years past, but does have its own advantages. Today's students are still graduating with qualifications required in many of the current job descriptions. As predicted by Moore's Law, the complexity of student-affordable systems has increased at an exponential rate, giving students the ability to work with components and development tools that cost thousands of dollars just five to ten years ago. Now, complete development systems with computing power rivaling that of a smart phone can be purchased for as little as twenty dollars. These development systems not only provide computing power, they also include integrated hardware debugging capabilities, field programmability, and free software coding and debugging tools. When a microcontroller can do a job for less than one dollar, the time it took previous students to build a custom logic based application, can now be spent creating more advanced software applications.

In the past, hardware debuggers were very expensive and only big companies had access to emulators. Everybody else had to work using a "burn and crash" methodology which was not very efficient to detect infrequent and random problems. Today most microcontroller development systems and programmable platforms come with debugging capabilities that allow students to attack problems in a wide variety of forms from on chip hardware debugging and tracing to embedded logic analyzing. These capabilities are all available through the integrated development environment (IDE) associated with the platform and are typically available for free to students.

The open source movement in the embedded market has also changed the landscape for today's embedded design students. Designers all around the world are sharing libraries of hardware cores and software drivers that allow students to control components that are either integrated on the platform or connected externally. With the available libraries of sample code, students are able to realize more complex designs by building on something that already exists rather than starting from square one. Prevailing electronic systems have reached such a complexity that reutilization is necessary from both hardware and software aspects. Oftentimes, in order to meet the modern time to market requirements, the electronic industry creates large systems by combing small to medium size systems that have already been developed, instead of starting from scratch.

Along with having access to code libraries, students can also select hardware functions to be integrated into their design. Examples of these functions are processor cores, memory cores, and peripheral cores. These are available on companies' websites, sites such as Opencores.org[2], and different university websites. The availability of cores enables students to start at a higher level of abstraction in order to solve more complex problems. In the past, reusable components could have taken months or years to reach the state of maturity that is currently available in the open source market.

There now exists a large selection of platforms that contain driver libraries that are able to control all the internal peripherals and external components very efficiently. The low level requirements of these drivers, such as accessing a particular set of peripheral registers, is transparent to the end user. Two good examples of this, with very active online communities for the sharing ideas and code, are the Arduino[3] open-source electronics prototyping platform and the mbed[4] tool for rapid prototyping with microcontrollers.

5. Hard Core and Soft Core Definitions

While the advent of such devices as personal data assistants, smart phones and tablet computers has changed the definition for 'embedded system', this paper will address the traditional definition which is "a physical system that employs computer control for a specific purpose, rather than for general-purpose computations[5]." All embedded systems, regardless of how they are implemented contain the same underlying components: central processing unit, memory for program and data, input and output interfaces and control circuitry such as timers.

In this paper *hard core* refers to a system that cannot be reconfigured and all of the components are already fixed by the manufacturer and integrated into a development board. By contrast *soft core* will refer to a system on a programmable chip (SOPC). In an SOPC, the processor, memories and components are created by using the available resources in a programmable logic device and it can be customized according to a particular set of specifications.

6. The Debate

One of the fundamental themes in teaching embedded systems design is that every design decision involves a tradeoff. Students must be taught to weigh the advantages and disadvantages carefully in deciding what the best solution is. Whether it is size, power consumption, flexibility or speed, every design consideration is about tradeoffs. In much the same manner, the choice of which design platform to use in an embedded systems lab course involves a plethora of tradeoffs. When tradeoffs are involved, there is never a universal 'perfect' solution. Rather, the optimal solution for the given set of constraints is the goal that should be aimed for.

6.1. Environment

Two types of environments exist today; physical and virtual. As its name implies, a physical environment is the actual hardware that is being used and includes the microcontroller and all of its peripheral devices. In contrast, modern design tools allow for modeling of a complete system design without having the hardware available. This emulating of hardware components for design verification is known as a virtual environment.

Hard: When using a hard core development system, students must have access to the hardware in order to develop and verify their designs. While most microcontrollers have software simulators built into the integrated design environment (IDE), the interaction of the processor with the hardware peripherals cannot be fully modeled. The lack of a complete system model limits the amount of work that can be accomplished in the virtual environment for a hard core system.

Soft: The virtual environment is best suited for a soft-core solution. Since the processor core and all of the peripheral devices with which it interacts are written in a hardware description language, the design software has the ability to generate a simulation netlist that includes real-time operating factors such as routing and propagation delays. With such a netlist, the entire system can be simulated and verified using a tool such as Mentor Graphics' Modelsim without ever downloading to the development board. In situations where the students may not have access to the development board outside of scheduled lab time, this permits them to work at any hour of the day provided that they have the software. While most scheduled lab periods are only a few hours per week, sufficiently challenging designs may take students up to ten times that amount of time to complete. Thus development in a virtual environment enables more challenging assignments that will further student competency. The ability to work when and where they please is also seen as a benefit by the students.

6.2. Visibility to Signal Behaviors

Even with meticulous planning and diligent simulations, rarely does an embedded system work the first time the software is run within the microprocessor. Whether it is a soft core or a hard core development system, hardware debug is an integral part of the iterative design process and a crucial skill for students to acquire.

In the days when digital systems were built from discrete components all component pins and thus all system signals were accessible to perform diagnostics using traditional laboratory equipment such as oscilloscopes and logic analyzers. Visibility of all signals interconnecting one component to another allowed for critical timing relationships between signals to easily be viewed and parallel bus traffic to be monitored. As modern digital systems have become more and more integrated and less discrete, the interconnecting signals between system components are hidden in the silicon and no longer available for diagnostic probing. The inability to trace internal signals increases the difficulty in debugging a system and today's students lose insight into the complex relationships between internal signals.

Hard: Because a modern microcontroller integrates a processor, memory, I/O and control circuitry on a single integrated circuit, it is impossible to monitor signal transitions with either a virtual logic analyzer or a traditional logic analyzer. The understanding of how the processor interacts with memory and I/O can only be demonstrated via a schematic and not witnessed in operation.

Soft: As previously explained, the soft core embedded system is developed entirely in a hardware description language (HDL) for realization in a field programmable gate array (FPGA). The synthesis and mapping processes result in a netlist containing all of the signals in the components as well as signals that interconnect the components. Development environments for FPGAs often include a system-level debugging tool that captures and displays any signal in the circuit. Such tools act as virtual logic analyzers and provide visibility into internal bus and signal transactions. Like real logic analyzers, virtual logic analyzers display a snapshot of the signals at a given instance in time.

Both: The advantage of being able to view internal signals clearly is with a soft core processor. However both soft core and hard core development systems provide the opportunity to view

signal behavior using modern tools. The new generation of mixed signal oscilloscopes (MSO) is equipped with logic analyzer functions. While limited in the number of signals that can be connected, serial communication protocols such as I2C, SPI, USB, and CAN have largely substituted parallel communications with peripherals. Exploring and debugging these serial protocols does not require the large number of ports available in a traditional logic analyzer. Using the protocol analysis capabilities within the MSO, students gain a deeper understanding of the serial communication protocols that are being used more and more frequently in commercial products.

6.3. Testability

Embedded systems lab activities must challenge the students to be innovative and develop a system to solve a complex problem. In order to do so, students need to be able to iteratively test and modify their work until the correct solution is found. As much as instructors and textbooks stress the importance of planning, flow-charting and state transition diagrams, students still prefer to jump in with both feet and immediately start coding. This approach typically results in unstructured code and many rounds of test and edit. While not the method preferred by the instructor, the reality is that a majority of the student's time is spent trying things to see if they work.

Hard: In a hard core environment, the microcontroller and interconnections have already been verified by the semiconductor manufacturer and the development platform has already been verified and tested by the board manufacturer. Starting from a known good system allows students to devote their time to writing the overall software application, adding components via serial communication interfaces, and writing the drivers for these devices. The methodologies to test these stages are well defined and developing the application becomes the main focus. When operation is not as expected, testing and debug can be isolated to the software portion of the system as it is highly improbable that the fault lies in the hardware (although blaming the hardware for failure seems to be the students' first reaction).

Having a known good system has its advantages in that hardware testing is rarely necessary, but it requires that the student have the hardware in their hands to carry out the testing. Some amount of testing can be done virtually with a simulator or emulator, but full system verification requires access to the full system. Unless each student has their own development system, they are limited on when and where they can work and this in turn limits the complexity of the designs that can be assigned.

Soft: Configurable soft cores introduce an additional level of testing since the hardware that has been implemented in the programmable device has to first be verified before software development can begin. In upper level embedded design courses student projects often include both hardware and software development. Each hardware component or IP created needs to be first simulated and then tested for interoperability with the processor and other cores in the system. If the soft core system is not thoroughly verified prior to running the application software, the debug process becomes more complicated as it may not be discernable whether the error exists in the software or the hardware.

The advantage of using a soft core or programmable development board is that students are able to do much of the testing in the virtual environment as previously described. It is not uncommon that the application programmers on a large project in industry do not have access to the hardware when development begins. In that case they must be able to design and test in a virtual environment and have an application that is ready for integration once the hardware is complete. Having the capability for development and verification in a virtual environment and forcing students to use it, provides them with a valuable experience and necessary job skill. Students who only have experience with "try it and see how it works on the hardware" may face a steep learning curve once they are in the real world.

6.4. Design for Flexibility

An important feature of a development platform is how easily it can be expanded to add and communicate with devices other than the ones integrated into the board. In advanced classes students need to incorporate components that fit outside the boundaries of their development system and learn how to integrate off the shelf components, available from different companies, in order to meet a particular objective. There are two main avenues to accomplish this. The first is to customize the platform by adding intellectual property (IP) blocks and the other is to extend the platform through the use of parallel and serial interfaces.

Hard: Since a hard core development system is not configurable, a way to customize the platform using IP blocks does not exist. When an additional digital or analog component needs to be added because it is not already available on the same die, for example an analog to digital converter (ADC), a digital to analog converter (DAC), a filter, a sensor, a port extender, and LCD controller, the only way is by using standard interfaces.

Soft: The programmable logic device on the soft core platform provides a means for the system to be expanded by the use of digital IP cores. These cores can be obtained by purchase or through open source channels. Additionally, the creation of cores can be a part of the coursework. A course that takes a truly integrated hardware/software approach to embedded system design should include core creation as an intended learning outcome. Starting with a blank slate for a system, students can use a hardware description language to create their own components. Most soft-core development systems contain a custom component editor that allows a designer to easily integrate their custom component with that processor core. The application arena is greatly increased when students are no longer limited to using just hardware that has already been placed on the board. If there is a requirement to add analog components, than the soft core solution is similar to the hard core solution.

Both: Whether hard core or soft core, all development platforms come with a set of peripheral devices already on the board. On the simplest of boards it may only be some switches and LEDs while on the more expensive boards it may include UART, PCIe, VGA, Ethernet, and USB. Most boards also include general purpose input/output (GPIO) pins for expansion through asynchronous and synchronous serial port interfaces such as: I2C, SPI, and CAN. The peripherals on the board provide that opportunity for students to gain experience in various communication protocols and also in interfacing to other components. On a hard core development platforms that experience is limited to software development, while on a soft core platform both software and hardware design experience is possible. Some of the low and high

level drivers required by complex interfaces such as USB and Ethernet may be provided with the platform or available for purchase. Exposing students to a myriad of interfaces is beneficial in preparing them for the workplace. Most of modern industrial designs use these interfaces and protocols to design complete networked systems. A good example is an automobile which is connects a large number of processors and sensors using the CAN bus.

6.5. Cost

When considering the cost factors in choosing a development platform for an embedded systems design course a usage model for the platforms needs to be addressed. Will each student be provided a platform for their sole use for the entirety of the semester or quarter? Will each student be required to purchase their platform for the course? Will the university provide one lab set of platforms to be shared among the students only during lab and open lab hours? With the escalating costs of a college education, many students cannot afford the additional financial burden of purchasing a development platform. However, they appreciate having their own board that they are able to work with at any time of the day or night. If it is a priority for all students to have their own board and it is expected that they will purchase them, than a lower cost development platform should be sought. If the university plans to provide a set of boards to be shared among students, a costly platform can be considered since far fewer will need to be purchased. Finally, if the university plans on providing a system for each student, than departmental budget will be the driving force in deciding on an appropriately priced platform.

Traditionally a custom hard microcontroller solution is more cost effective than a completely programmable one. The price differential that exists between two comparable boards, one with a hard processor and one with a soft processor, can be attributed to the difference in price between a microcontroller and an FPGA. Advanced technology and high volume production have enabled powerful microcontrollers to be available for less than a few dollars each while FPGA volumes and technology force their price to be several times that amount, resulting in an overall more expensive system.

Tables 1 and 2 below list various development platforms and their associated cost. It can be seen that a robust hard core platform can be purchased for price that is affordable for the average college student or that fits well in a department's tight budget. Some lower cost soft core development platforms have been introduced, but care should be taken in choosing one of those as they are limited in their usefulness in upper level courses due to the size of the programmable device.

Table 1. Examples of Hard Core Platforms

Development platform	Processor Core	Interfaces	Cost
LaunchPad Texas Instruments	MSP430F2211/31	GPIO, I2C, SPI, ADC, UART, PWM. LED, PB	$4.30
MSP430FG4618/F2013 Experimenter Board – TI	MSP430FG4618/F2013	GPIO, I2C, SPI, ADC, UART, PWM. LED, PB, CT, DAC	$99.00
DK-LM3S9D96 Development Kit – TI	Stellaris ARM® Cortex™-M3	GPIO, EPI, ADC, UART, SPI, I2C, I2S, CAN, ETH, USB, PWM, AC, QEI, CT, LCD	$425.00
EK-EVALBOT – TI	LM3S9B92 ARM® Cortex™-M3	GPIO, USB, PWM, UART, I2C, I2S, ETH, SPI, LED, PB, SW	$149.00
LPCXpresso - NXP	LPC1XXX ARM® Cortex™-M3/M0	GPIO, USB, PWM, UART, I2C, ETH, SPI, CAN, LED	$29.95
STM32VLDISCOVERY – ST Microelectronics	STM32F100RBT6B LPC1XXX ARM® Cortex™-M3	GPIO, USB, ADC, DAC, PWM, I2C, SPI, USART, LED, PB	$9.88
Explorer 16 Starter Kit (DV164037) - Microchip	PIC24FJ128GA010 dsPIC33FJ256GP710	GPIO, USB, LCD, PB, LED, SPI, ADC, SPI, I2C, CAN, PWM, ETH	$129.99
TWR-K60N512-KIT - Freescale	MK60N512VMD100 ARM® Cortex™-M4	GPIO, CT, LED, PB, I2C, SPI, I2S, CAN. ADC, USB, DAC, PWM, UART, ETH	$139.00
CY8CKIT-001 PSoC® Development Kit - Cypress	CY8C28 M8C/ CY8C38 Single cycle 8051 CPU/ CY8C55 ARM® Cortex™-M3	GPIO, UART, USB, SPI, LCD, CT, PB, PWM, ADC, DAC, AC, CPLD	$248.94
Actel A2F-EVAL-KIT SmartFusion Evaluation Kit -	A2F200M3F-FGG484ES FPGA, ARM® Cortex™-M3	GPIO, UART, USB, I2C, SPI, DAC, ADC, AC, LED, LCD, PB, FPGA	$99.00

Table 2. Examples of Soft Core Platforms

DE0-Nano Development and Education Board - Terasic	Altera Cyclone® IV EP4CE22 – NIOS II	GPIO, USB, ADC, SW, LED, PB	$59.00
Altera DE2-115 Development and Education Board - Terasic	Altera Cyclone IV EP4CE115 – NIOS II	GPIO, USB, ETH, UART, LCD, Audio, Video, SW, LED, PB	$299.00
Altera DE0 Board - Terasic	Altera Cyclone III EP3C16F484 NIOS II	GPIO, USB, UART, LCD, Video, SW, LED, PB	$79.00
Nexys™2 Spartan-3E FPGA Board - Digilent	Spartan-3E-500	GPIO, USB, UART, Video, SW, LED, PB	$99.00
Virtex-5 OpenSPARC Evaluation Platform - Digilent	Xilinx Virtex®-5 XC5VLX110T	GPIO, USB, ETH, UART, LCD, Audio, Video, SW, LED, PB	$750.00
Avnet Spartan-6 LX9 MicroBoard	MicroBlaze™ soft processor	GPIO, USB, SW, LED, PB	$89.00

Acronyms

AC – Analog Comparators
ADC – Analog to Digital Converter
CAN - Controller Area Network Interface
CPLD – Complex Programmable Logic Device
CT – Capacitive Touch
EPI – External Peripheral Interface
ETH – Ethernet Controller
FPGA – Field Programmable Gate Array
GPIO – General Purpose I/O
I2C – Inter Integrated Circuit Interface
I2S – Inter Integrated Sound Interface
LCD – Liquid Cristal Display
LED – Light Emitting Diode
PWM – Pulse Width Modulation
QEI – Quadrature Encoder Interface
SPI – Serial Peripheral Interface
SSI – Synchronous Serial Interface (SPI compatible)
SW – Switches
USB – Universal Serial Bus controller
U(S)ART Universal (Synchronous) Asynchronous Receiver and Transmitter

6.6. Power Consumption

As today's consumers have an insatiable appetite for high-performing, lightweight portable devices that can go long periods of time between recharging, industry must respond by providing systems that meet the customers' desires. To fulfill these industry needs, students need to be taught the skills to design systems for higher energy efficiency, to be battery operated or even to operate on harvested energy[6]. It is essential that they are trained to be aware of optimization at all levels to address the issues of power consumption. The importance of training students in low power design is evident in current student contests where the goal is to show "how low can you go" in terms of energy consumption. This new goal is changing the traditional paradigm in which performance was the metric.

There are two basic types of power consumption: static and dynamic. Static power is due to leakage and is technology dependent. Dynamic power is principally a function of the frequency of operation, loading capacitance at each node and the power supply voltage. Dynamic power can be determined by computing the switching rate at each node. Both static power and dynamic power can be controlled through various techniques.

Hard: One of the characteristics of modern processors is that they possess power saving modes. By utilizing these modes one or more operational units can be shut down, the processor itself can be turned off and clock generation can be disabled. When the whole system is in sleep mode, the only power consumed will be due to leakage of the operation units that monitor for an event to occur. The speed at which the processor can wake up and be fully operational again is a key aspect on these implementations. In many commercial processors available to students, that time has been scaled down to less than 10μs. Implementation of various low power features has led to the claim by hard core processor manufacturers that a product can be designed that will operate more than a decade on a single battery. A ten year battery life is of particular concern to students who plan to make a career of designing embedded systems for use in inaccessible locations such as the human body.

In addition to training students for low power design, power consumption issues come into play with the student development system. By having the processor and other components all contained in the same chip, a microcontroller and thus a hard core system has already been optimized for low power consumption, making it the better choice for battery operation.

Soft: Because they tend to have the low gate utilization and rely on memory technology and lookup-tables, FPGAs by nature are not power efficient. As a result, a battery powered soft core development platform, would require frequent battery replacements. The inconvenience of having to change or recharge batteries repeatedly is viewed very negatively by students who want their systems to be operation when, and for as long, as they are ready to work.

To design for power efficiency on a soft core is achievable, but the efforts to accomplish this would surpass the benefits. The absence of sleep modes and selective power-down on programmable boards makes them not very suitable for teaching low-power design optimization.

6.7. Student Learning Curve

Every time a new development platform is introduced in a class, a student learning curve exists for the electronic design automation (EDA) tools associated with the platform as well as for the platform itself. Valuable class and lab time in the first few weeks of the semester/quarter must be spent introducing the platform and completing tutorials. This reduces the amount of time available for teaching course concepts, especially in a quarter-based system. The steepness of the learning curve determines how quickly students can be creating meaningful designs on their development systems.

Aside from the obvious, complexity of the tool and platform, another factor that affect the steepness of the learning curve is academic level. As students progress through the academic levels, they are exposed to various tools and gain confidence in learning something new. The more EDA tools a student works with, the easier it is for them to learn a new one. Since many of the features are similar between tools and many are based on a common platform such as Eclipse the learning curve for seniors is much less of a concern than it is for freshmen.

Hard: Hard core development systems are the easiest way to introduce students to microcontroller design and programming. A simple platform can initially be used to learn the basics of programming and interfacing. As previously mentioned, the hardware of a hard core processor platform is a known good system. Entry-level students can focus primarily on simple software applications to be run on the board. As students progress in their program, more advanced features such as specialized serial communications, interface with analog domains, wireless communications, motor control, real time operating systems, and direct memory access can be introduced. With a hard core, the students learn how to use a particular architecture, rather than designing their own architecture.

Soft: Soft core development boards are not ideal for introducing embedded systems programming. The integrated hardware/software environment of the programmable system forces the student to have to work in two different development programs. For freshmen and sophomores just being introduced to the topic, having to learn both at once can be overwhelming. Juniors and seniors who already have microprocessor experience and HDL experience are a better target audience for a soft core platform.

One advantage that the soft core platform does have over a hard core platform is that it can be used for introductory courses in digital systems design and then later used, with the addition of the processor core, for embedded systems design. Students become familiar with the board and the hardware design tools in the introductory digital classes and only have to learn the software development environment after adding a processor core. Since a soft core platform can be used in several different courses in the curriculum, it has the ability to "grow" with the students. If students are being asked to purchase their own development boards, they appreciate being able to reuse it.

6.8. Training for Industry

Part of the job of an educator is to equip students with the skills necessary to be productive in industry upon graduation. Unfortunately, when it's in a high-tech field knowing what skills they will need can be a moving target. On one side of the argument, having experience with a

particular processor or platform could give student leverage when job hunting, especially when applying for a company that is looking for someone with that skill set. The company will save money by not having to train the new hire and can make them productive immediately. The other side of the argument is that we may not know that skill set that industry will require when today's freshmen are looking for jobs. Additionally, companies may prefer a new hire with a diverse skill set so that they can move from one project to the next. Due to downsizing, companies expect employees to fill more and varied positions.

Hard: The debater for hard core systems is on the first side of the argument, that students should be trained for a specific architecture being used in industry. Today, ARM is the most widely used microcontroller architecture in the world. ARM processors can be found in products ranging from smartphones to automobiles to giant neutrino detectors[7] and are especially popular in portable communication devices. One of the reasons for ARM's popularity in the portable market is that it has been optimized for low power operation over high performance. Believing that the popularity of ARM will continue to grow, a hard core development platform would be the logical choice to best prepare students with a specific skill set that will provide an advantage for employment. At all price points, development platforms featuring an ARM microcontroller are available in abundance.

Soft: A soft core system is the platform of choice for the other debater. While the demand for portable and hence low power, electronics continues to grow, there has also been expansion in the FPGA market. Traditionally FPGAs have been used mainly in the communications and data processing markets but have recently seen increased usage in consumer, industrial and automotive arenas[8]. A high degree of reprogrammability, quick time to market and complete system integration on one device contribute to the rising popularity of FPGA solutions. By training students with the use of a soft core platform, they graduate with skills that will enable them to choose a path in hardware design, software design or a combination of the two. Current industrial trends and market predictions indicates that there will continue to be a demand for employees with both skills.

6.9. Hardware-Software Partitioning

An important tradeoff that is stressed in an embedded systems design course is the partitioning of software and hardware. Typically functions are faster when run in hardware, yet offloading to hardware utilizes more resources. Increased resource utilization results in a larger system that consumes more power. If the decision to implement some functionality in hardware rather than software is made for a system that has already been built or is in the final stages of the design flow, it could mean extensive redesign and rework. Having all functionality residing in software allows for a smaller overall system with lower power needs but may not fulfill some high-speed requirements and may increase the overall memory needs.

Hard: In a hard core development system, the hardware resources are already in place and the capability to create a custom instruction does not exist. Unless a hardware accelerator has already been included on the board, students are not able to try out different configurations of hardware and software. When working with the fixed hardware configuration of a hard core development system, the main focus for the student is that of developing software for a particular application.

The absence of hardware to carry the load of high performance forces students using a hard core platform to make their code as efficient as possible in order to meet the time allocated for each function. Students are presented with the traditional tradeoffs of polling vs. interrupts, assembly vs. C, and super-loop vs. real time operating systems (RTOS). By carefully creating laboratory assignments, instructions can guide students in analyzing these options and determining which will lead to the most efficient solution.

Soft: In a soft core development platform there is extra logic available in the FPGA for student experimentation and development. Not only can students design their own peripheral cores, but they can also develop hardware modules to replace computationally intensive software algorithms. Known as custom instructions or hardware accelerators, these hardware modules are written in a hardware description language and synthesized as part of the processor core. An instruction mnemonic is then assigned by the EDA tool for inclusion in the embedded program. During execution of the embedded program, when the given instruction is reached, the processor offloads it to the hardware, retrieves the result and then continues. With this capability, students are able to design systems with various configurations of hardware and software and then perform a comparative analysis to find the best solution. Instead of just discussing design tradeoffs in a lecture environment, students are able to put the theory to the test and measure exactly the speedup obtained by offloading to hardware or the difference in memory usage when leaving all functionality in software[9].

6.10. Student Engagement

Engaging a large number of students with different career goals, varying interests and diverse experiences, is one of the most difficult challenges for any computer engineering technology program. Adding to the challenge is that electrical engineering technology students, many of whom have decided that they do not like programming are required to take these courses also. Even the electrical engineering technology students who are interested in programming present a challenge as they often have in mind a different set of applications that can be pursued with these platforms.

Today's students, dubbed the millennial generation, live in a connected world of multi-media and entertainment. In order to engage many of today's students, projects must have an end product that they find entertaining or exciting. The correct platform, assignments, laboratory exercises and projects to spark and hold students' interest depends on what year they are in the program. In the first couple of years students like to control a device. A robotic platform[5] could be ideal for accomplishing this objective but if the task of programming the platform becomes very complex, it can be discouraging. When students are advancing in their programs, they get excited by controlling other type of devices such as data converters, motors, LCD displays, touch screens, sensors, wireless, etc. In senior years in addition to the experiences mentioned before, they also get excited by running multi-cores, operating systems such as Linux or Android as well as RTOS to include on a capstone experience. The ideal scenario would be to have a common platform that could be used and extended throughout their program

Hard: A hard core processor platform offers some advantages in the area of student engagement, especially in the early years of the program. As previously mentioned, a hard core system allows students to be up and running in a relatively short amount of time. When students are able to

download their program to the board and then see the result manifest itself as a robot moving or even an LED blinking they get excited. That excitement translates into the desire to make the platform do even more. The combination of the excitement gained from being able to control something and the confidence built from being able to do it after just a short amount of time working with the tools leads to increased engagement with the platform.

Hard core development systems also have an advantage with the number and types of hardware interfaces built into them. A wide variety of hardware interfaces allows for labs to be developed at each level to best meet the interest of the students. Hard core boards often contain an analog input and an analog to digital converter on the board. Soft core boards most often are limited to digital signals only. The presence of an analog input enables students to develop projects that take in an audio or video data stream, for example. Most students today become very engaged when working with and manipulating this type of data.

Soft: The additional time needed to become familiar with the two design environments (hardware and software) necessary in order to get a system up and running on a soft core platform makes it overwhelming for students who are just starting out. A soft core platform works better in a curriculum where it is used first solely for hardware development with the software development piece added in later classes. Overwhelming, and thus frustrating students early on could lead to disengagement from the board and even the curriculum.

The soft core platform does offer one advantage in the most advanced classes in embedded systems design. Since adding a processor to the system is as simple as invoking another core in the system builder, students are able to experiment with multi-core applications. A hard core platform limits students to use the single processor that already exists on the board. Multi-core processing is a current trend that students want to know more about and if the platform provides the environment in which they can experiment with a technology that they are interested in, they will be more engaged. Team-based projects can also benefit by utilizing a dual core. For example, one student can design the audio portion of a project while another student designs the display portion of the project, each using a dedicated processor. The two portions, developed independently, can be integrated onto one programmable device on the board[10].

7. Conclusions

After carefully weighing the advantages and disadvantages of hard core and soft core development platforms, it can be concluded that there is no right or wrong answer. Each provide their own advantages and there is not a single platform that will allow educators to prepare students with every single skill required for a computer engineering technology graduate. The best that can be done is to present the most appropriate platform to achieve the course goals and to achieve the overall program goals at each level in the program. Students must be presented with an equal exposure to both fixed and programmable design environments. A senior capstone project should be a combination of the two where they are able to leverage on the experience of hardware/software partitioning, hardware acceleration, wired and wireless communications, networking, sensor interfacing, graphical user interface designs, systems control, etc. In general, the students should see the culmination of their program with a project that represents what they have learned and can be used to market themselves when searching for a job.

In recent years there have been some new platforms introduced that could potentially cover the gap between hard and soft cores as viewed in this paper. These platforms come with a hard core processor plus programmable logic and include programmable analog components as well. Examples include the Actel Smartfusion and Cypress PSoc platforms. Will platforms like these become the best tradeoff to expose students to a variety of design decisions and prepare them to join the workforce? The debaters agree that these platforms may very well be the system of choice for embedded systems design education in the near future.

References

[1] R. J. Tocci, *Digital systems : principles and applications*, 11th ed ed. Upper Saddle River, N.J. :: Prentice Hall, 2011.

[2] (2012, 01/04/12). *Home :: OpenCores*. Available: http://opencores.org/

[3] (2012). *Arduino - HomePage*. Available: http://www.arduino.cc/

[4] (2012). *Rapid Prototyping for Microcontrollers | mbed*. Available: http://mbed.org/

[5] C. Hamacher, Z. Vranesic, S. Zaky, and N. Manjikian, *Computer Organization and Embedded Systems*, 6th ed.: McGraw Hill - Higher Education, 2012.

[6] A. F. Mondragon-Torres and IEEE, "Work in Progress - Ultra-Low Power and the Millennium Generation," *2010 Ieee Frontiers in Education Conference (Fie)*, 2010 2010.

[7] C. Edwards, "20 years ago today - The making of ARM," *Engineering & Technology*, vol. 5, pp. 66-69, 2010.

[8] D. Kish. (2012, 01/11/2012). *FPGAs: Architectural Innovations Open Up New Applications*. Available: http://electronicdesign.com/article/digital/fpgas-architectural-innovations-open-up-new-applic.aspx

[9] J. Christman and E. Alley, "*A Hands-on Approach to Demonstrating Hardware/Software Tradeoffs in an Embedded System Design*," in *Proc. 2011 American Society for Engineering Education Annual Conference & Exposition*, Vancouver, BC, 2011.

[10] A. F. Mondragon-Torres, A. Kozitsky, C. Bundick, E. Mc Kenna Jr, E. Alley, M. Lloyd, P. Stanley, and R. Lane, "*WORK IN PROGRESS - AN AGILE EMBEDDED SYSTEMS DESIGN CAPSTONE COURSE*," in *2011 Frontiers in Education Conference*, Rapid City, South Dakota, 2011.