Proceedings,
The 2013 ASEE North Midwest Section Conference,
October 17-18, 2013, Fargo, North Dakota.

Experience of Teaching Embedded System Design using FPGAs

Department of Electrical, Computer Engineering and Technology
Minnesota State University, Mankato, MN 56001
Han-Way Huang, han-way.huang@mnsu.edu
Nannan He, nannan.he@mnsu.edu

Abstract

An embedded system is a product using one or more computers as its controller. Traditionally, the controller of an embedded system is an off-the-shelf microcontroller from microcontroller vendors. However, an off-the-shelf microcontroller may not provide the required peripheral functions or cannot achieve the desired performance required by the application. In this situation, the embedded system designers may either design their own special microcontroller chip or configure the FPGA chips to meet the functional and performance requirement. Designing a dedicated microcontroller chip is only justified when the embedded system is going to be duplicated many million copies. When the volume is not high enough or time-to-market is very tight to justify the design of a dedicated microcontroller, configuring an off-of-the shelf FPGA chip becomes the only viable approach.

An FPGA-based embedded system design approach starts with selecting the processor core, amount of on-chip memory, and peripheral modules from the FPGA vendor library using the design software and then generates the HDL file that describes this controller module. The second step is to write an upper-level module to instantiate the controller module generated in the previous step and also instantiate certain special peripheral functions provided by third parties or from the designer's own library to meet the performance requirement. The third step is to write the application software in C or C++ language to implement the embedded functions. This paper describes our experience of teaching a combined senior/graduate course on embedded system design using this approach in our electrical and computer engineering program.

Microcontroller features

The heart of an embedded system is the microcontroller. Depending on the number of bits that the processor can manipulate in one operation, the microcontroller can be classified as 8-bit, 16-bit, 32-bit, and so on. In general, 8-bit and 16-bit microcontrollers are targeted toward simpler applications that do not require high performance whereas 32-bit microcontrollers are targeted toward more complicated applications that require much higher performance. The usefulness of a microcontroller is greatly determined by the peripherals added to the chip. Over the years, microcontroller peripheral functions to a large extent have been standardized. For example, the following peripherals [1, 2, 3, 4] are common to most microcontrollers:

- Parallel I/O ports: An I/O port consists of a set of I/O pins (8, 16, or even 32) and associated registers. I/O pins are used to drive I/O devices such as light-emitting diodes (LEDs), seven-segment displays, liquid crystal displays (LCDs), key pads, keyboards, and so on.
- Timer functions: Timers are often used to create time delay, measure signal parameters, generate waveforms to be used in control functions, and keep track of time-of-day.
- Serial interfaces: The most common serial interfaces include universal synchronous asynchronous receiver and transceiver (USART), serial peripheral interface (SPI), inter-integrated circuit (I^2C), local interconnect network (LIN), controller area network (CAN), universal serial bus (USB), and so on. A microcontroller uses these interfaces to communicate with peripheral devices or another microcontroller.
- Analog-to-digital converter (ADC): ADC allows the microcontroller to measure non-electric physical quantities such as pressure, humidity, temperature, weight, and so on. The analog-to-digital conversion results are represented in digital values. The result of the measurement can be used to carry out certain control operations.
- Digital-to-analog converter (DAC): A DAC converts a digital value into a voltage. DAC is used to generate waveforms, playback music, and so on.
- Analog comparator: This function is often used to detect whether certain signal's voltage exceeds a predefined threshold value.
- Interrupt controller: Peripherals of the microcontrollers use interrupts to request attention from the CPU. The CPU executes a short program (called interrupt service routine) in response to an interrupt request. With the interrupt function, the CPU can continue to perform useful operations while peripherals are busy in their operations.
- Direct memory access (DMA): DMA is a data transfer method in which the CPU does not execute instructions to perform the actual transfer. The CPU sets up the source address, destination address, and transfer count and let the DMA controller carry out the actual transfer. DMA transfer is much more efficient than data transfers performed by the CPU.
- Watchdog timer: Watchdog timer is implemented to detect software errors.
- Sleep modes: Sleep modes are added to reduce power consumption whenever the microcontroller is not actively perform any useful functions.

In addition to these common peripheral functions, microcontroller vendors may also implement other specialized peripheral functions to gain advantage over their competitors. When teaching a microcontroller course, it is desirable to cover as many peripheral functions as possible. Readers must be aware that common peripherals are designed and implemented differently in each microcontroller family even from the same company.

The Word-length of the microcontroller

The number of bits that a microcontroller can manipulate in one operation is referred to as the **word length** of the microcontroller. A 32-bit CPU requires many more transistors to implement than an 8-bit or 16-bit CPU and, therefore, consumes more power. However, after adding many peripheral functions and the on-chip flash memory, a 32-bit microcontroller may not consume much more power than an 8-bit or 16-bit microcontroller.

The 8- and 16-bit microcontrollers may have some advantage in program size for simple and non computation-intensive applications. However, when moderate to high arithmetic computation operations are required for the application then 32-bit microcontrollers have significant advantage in program size over the 8- and 16-bit microcontrollers. One 32-bit multiplication or 32-bit divide operation is performed by executing one multiply or divide instruction in the 32-bit microcontroller. For the same 32-bit multiply or divide operation, an 8- or 16-bit microcontroller is required to call a subroutine. For this type of applications, the 8- and 16-bit microcontrollers may consume more power because they require much longer time to complete the operations and hence will be busy at a much higher percentage in time compared with the 32-bit microcontroller. The microcontroller can be put to sleep and save power when it is not busy performing useful work.

The 8- and 16-bit microcontrollers do not have price advantage over the 32-bit microcontroller for the same amount of flash memory and the same number of pins. For example, the 8-bit Atmel ATMEGA1280 (with 100 pins and 128 kB flash memory) is more than twice as expensive as the 32-bit SAM4L2C (with 100 pins and 128kB flash memory).

Considering the factors of power consumption, program size, and cost, 8-bit and 16-bit microcontrollers are more suitable for simple applications. To give students more options when they enter the job market, we teach both the 8-bit (or 16-bit) and 32-bit microcontrollers.

Sources for microcontrollers

When choosing a microcontroller as the controller for the embedded system, the designer has three options:

- Buy the off-the-shelf microcontroller from the distributor or directly from the manufacturer. This approach involves the least design effort and is also the least expensive approach. If the chosen microcontroller satisfies the design specification, this approach would have the shortest time-to-market. Apparently, most embedded systems use one or multiple off-the-shelf microcontrollers as their controller.
- Configure an FPGA chip into a microcontroller and use it in the embedded system. The greatest advantage of the FPGAs is its configurability. The user can configure the FPGA on-chip resources to accelerate certain operations. FPGAs are used as the controller of an embedded system whenever no off-the-shelf microcontroller can satisfy the performance

requirement of the applications and the volume of the embedded system to be produced is not high.

- Design and fabricate a customized microcontroller. This is the most expensive and risky approach. However, the designer may adopt this approach whenever the volume of the embedded products is extremely high (for example, many millions) and no off-the-shelf microcontroller meets the performance requirement. When adopting this approach, the company often wants to minimize the risk by licensing a well-established processor core such as ARM or MIPS and add certain peripherals or special circuitry appropriate to the application. The most famous example is probably the ARM Cortex-A processors designed by Apple for its cell phones and tablet products.

Overview of the FPGA

A **field programmable gate array** (FPGA) chip consists of an array of programmable (or reconfigurable) logic components called "**logic blocks**" and a hierarchy of reconfigurable **interconnects** that allow the logic blocks to be wired together. Different company would use different names for the logic block. The logic blocks may be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

The programming of the FPGA chip is carried out by injecting a bit stream to configure every used logic block inside the FPGA chip. The programming point inside the FPGA chip can be implemented by using the SRAM cell or fuse. When SRAM cells are used, the FPGA chip can be reconfigured (or reprogrammed) over and over. However, when fuses are used, the FPGA chip is one-time programmable only. Fuse-based FPGAs have better performance (in terms of propagation delay) when using the same semiconductor technology. It appears that Actel (acquired by MicroSemi) is the only company that has ever used the fuse technology as the programming technology for the FPGA chip. When SRAM is used as the programming method, the programming information will be lost whenever power is turned off. A separate chip (called **configuration chip**) is required to hold the programming information for the FPGA chip. Whenever power is turned on, the FPGA chip shifts in the programming information from the configuration chip to configure itself. This process may take many milliseconds depending on the length of the bit stream.

Some FPGAs have analog features in addition to digital functions. The most common analog features are programmable slew rate and drive strength on each output pin, allowing the engineer to set slow rates on lightly loaded pins that would otherwise ring unacceptably, and to set stronger, faster rates on heavily loaded pins on high-speed channels that would otherwise run too slowly. Another relatively common analog feature is differential peripheral comparators on input

pins designed to be connected to differential channels. A few mixed signal FPGAs have integrated peripheral analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) with analog signal conditioning blocks allowing them to operate as system-on-a-chip.

The first commercial FPGA product XC2064 was introduced by Xilinx in 1985. This device has only 64 configurable logic blocks (CLBs). Each CLB has two 3-input lookup tables (LUTs). Xilinx continued unchanged and quickly growing from 1985 to mid-1990s, when competitors sprouted out, taking away Xilinx's market share. By 1993, Actel has about 18 percent of the FPGA market. The 1990s were an explosive period for FPGAs, both in sophistication and the volume of production. In the early 1990s, FPGAs were primarily used in communications and networking. By the end of the decade, FPGAs found their way into consumer, automotive, PC, and industrial applications. However, communications and networking are still the two largest markets for FPGA products.

The FPGA market is dominated by Xilinx and Altera with 49% and 40% market share, respectively. The remaining market is divided by several smaller companies including Lattice semiconductor (6%) and MicroSemi (4%, acquired Actel in 2010) [5].

To attack the embedded system market, FPGA companies offered the soft processor cores that can be implemented using the FPGA logic. Nios II, MicroBlaze, and Mico32 are the soft processor cores from Altera, Xilinx, and Lattice Semiconductors, respectively. Soft core processors are described in one of the hardware description languages such as Verilog and VHDL. To use the soft core processor, the FPGA user uses the software to invoke the library to implement these processor core and peripheral module into the FPGA chip and uses it as the controller of the embedded system. We are using this approach to teach students.

In the last few years, major FPGA vendors also provide an embedded processor inside the FPGA chip. In this approach, the FPGA chip consists of logic blocks, interconnects and an embedded microprocessor. Xilinx's Zynq-7000 includes a 1.0 GHz dual-core Cortex-A9 processor. Altera's Arria V FPGA includes an 800 MHz dual-core Cortex-A9 MPCore. The Actel SmartFusion devices incorporate a Cortex-M3 hard processor core and ADC and DACs.

Design flow of embedded system using FPGA with soft core processor

Design embedded system using the FPGA with a software core processor as its controller has two major parts:

Part 1
Configure the FPGA into a microcontroller. To do this, the FPGA vendor provides a software package that allows the user to select the processor core and appropriate peripherals such as

timers, serial interfaces, and so on and generate the HDL descriptions for them. The user then write a short HDL description (in Verilog or VHDL) to instantiate the HDL description generated previously. The resultant HDL file is then compiled to generate the bit stream that can configure the FPGA chip into the desired microcontroller. The processor core and peripherals are described in either Verilog or VHDL and placed in the library and are called **intellectual property** (IP) cores. These IP core descriptions are read-only and cannot be modified by the user. The users can also write their own IP cores if they are not available from the FPGA vendor. Keep in mind that the reason for using this approach is because there are performance requirements that cannot be met by the off-the-shelf microcontroller.

Part 2
Develop application software (often called **firmware**) to be executed by the microcontroller implemented using the FPGA. This step is no different from using the off-the-shelf microcontroller. The designer will use an integrated development environment (IDE) to enter, compile, and debug the program. An IDE [2, 3, 4] consists of a text editor, assembler, compiler, simulator, debugger, and device drivers. Once the firmware has been debugged, it must be programmed into a nonvolatile memory so that it will be available for the CPU to access and execute. Xilinx, Altera, and Lattice Semiconductor all offer IDEs based on the popular freeware **Eclipse** IDE.

Our experience

We taught embedded system design using the FPGA to both senior and graduate students as an elective. These students have different background. Some have learned VHDL or Verilog before. However, more than half of the class never learned any hardware description language. All of the students in the class have learned some type of microcontroller. With this background in mind, we decided to teach Verilog and a soft core processor in this class. Our goal is to enable students to use Verilog to design digital systems using FPGA and to implement embedded systems using the FPGA.

We have used the design tools from Xilinx, Altera, and Lattice Semiconductor on teaching digital system courses in the last fifteen years. The design tools from these three companies are all very good. However, Altera seems to have the best demo boards for lab experiments. When using Lattice Semiconductor's design tools, we designed our own demo kit using the CPLD chips from Lattice Semiconductor. According to our experience, using programmable logic devices (including FPGA and CPLD), hardware description languages (VHDL or Verilog), and design tools in teaching digital system courses has always increase students enthusiasm in the lab projects. We have seen many students spent extra hours in the lab in order to get their lab assignment done. When we taught introductory digital design course, we always assign students to do the first few lab assignments by using traditional TTL chips, which requires a lot of wiring

and causes a lot of frustration to students. However, after shifting to programmable logic devices and hardware description language and CAD tools, students become very happy working on the lab assignments.

Teaching embedded system design using the FPGA and teaching digital system design are not quite the same. Additional design tools are needed. Again, we decide to choose the soft core processor from one of the top FPGA vendors Xilinx, Altera, and Lattice Semiconductor. To choose among Microblaze, Nios II, and Mico32, we considered the following factors:

- Development software support
- Availability of tutorials and books for the development software
- Availability of affordable demo boards for lab projects
- Availability of lab projects

After comparing these factors, we chose Altera's Nios II soft core processor for our course.

Nios II is a soft-core processor targeted for Altera's FPGA devices. As opposed to a fixed prefabricated processor, a soft-core processor is described in HDL codes and then mapped onto FPGA's generic logic blocks (logic element (LE) in Altera's term). A soft-core processor can be configured and tuned by adding or removing features on a system-by-system basis to meet performance or cost goals.

The Nios II processor follows the basic design principles of a reduced instruction set computer (RISC) architecture and uses a small, optimized set of instructions similar to those supported by the MIPS processor. Its main characteristics are:

- Load-store architecture—Only load and store instructions can access memory. Arithmetic and logic instructions operate only on registers and immediate values.
- Fixed 32-bit instruction format
- 32-bit internal data path
- 32-bit address space
- Memory-mapped I/O space—Peripheral devices share the same memory space with memory components.
- 32-level interrupt requests
- 32 general-purpose registers

There are three basic versions of Nios II:

- Nios II/f: The fast core is designed for optimal performance. It has a 6-stage pipeline, instruction cache, data cache, and dynamic branch prediction.
- Nios II/s: The standard core is designed for small size while maintaining good performance. It has a 5-stage pipeline, instruction cache, and static branch prediction.

- Nios II/e: The economy core is designed for optimal size. It is not pipelined and contains no cache.

The **Quartus II** package is used to configure the FPGA into a microcontroller. After starting the Quartus II, the user invokes the **Qsys** tool from the **Tools** menu to select the processor, memory, peripherals, and so on and request Qsys to generate the HDL (Verilog or VHDL) description for the resultant microcontroller. The user then creates a project and instantiate the generated microcontroller in it. After that, the user compiles the project and generates the bit stream for configuring the FPGA chip. A screen of Qsys after adding several components but before adding interconnections is shown in Figure 1.

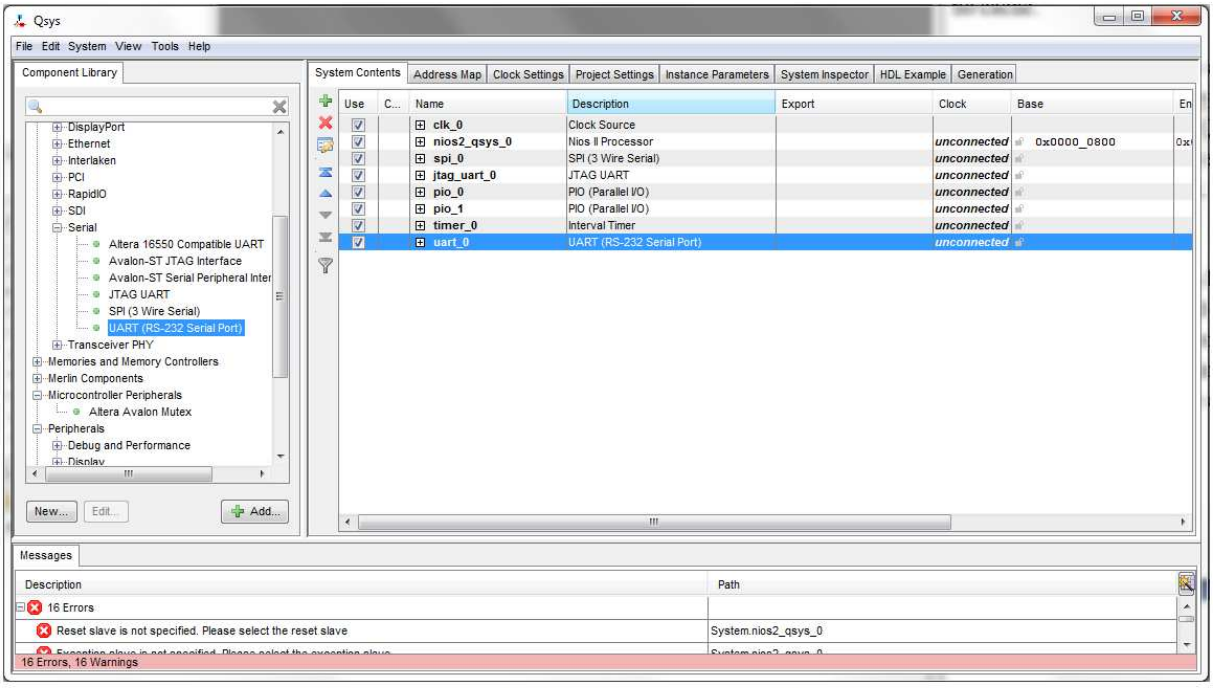

Figure 1. A snap shot of Qsys after adding several components but without adding connections

The next step is to invoke the **Nios II Build Toos for Eclipse** from the tools menu of Quartus II. The user then enters his/her assembly or C program, assemble/compile the program. To write application programs in assembly or C, the user is required to read the related documentations provided by Altera. After eliminating syntax and semantics errors, the user invokes the programming tool to download the configuration file and the firmware into the FPGA chip on the demo board for debugging. Debug commands such as setting breakpoints, adding variables to watch list, execute to cursor position, and so on can be invoked to debug the program.

For the university program customers, Altera also provides the software called **Altera Monitor Program** to help the user to develop his/her programs in assembly or C to be executed on the demo board (for use with the DE-series boards). The user can enter, assemble/compile, and

download his/her program onto the DE demo boards using this program. The Monitor program also provides some simple debugging functions such as single stepping through the program, setting breakpoints, and examining register and memory contents.

Altera provides several well-designed FPGA boards for users to learn how to use their FPGA design software and the device. Among them, the DE1 and DE2-115 are most suitable for learning the Altera design software and experimenting with the FPGA devices. The DE1 is less expensive and is based on the Cycle II device whereas the DE2-115 is based on the Cyclone IV device with more hardware resources. Both the DE1 and DE2-115 have been used by many universities worldwide in teaching digital design and computer organization courses.

Altera provide a tutorial on Qsys [6] and lab assignments for users to get started with embedded system design using the Nios II soft processor. Textbooks are also available for using the Nios II processor [7, 8].

We spent about four weeks on teaching Verilog programming and get students to work out a few design projects using the DE1 kits. After that we taught students to start from a very simple Nios II microcontroller configuration and write assembly and C programs to control its operation. We let students to work out a subset of the labs provided by Altera. Most students did not have difficulty to get those lab assignments to work. Because of the need to teach Verilog, we didn't have time to teach students to design their own peripheral functions and integrate with the Nios processor. We plan to find some way to achieve this in the future.

Conclusion

Embedded system design using the FPGA is more challenging than using the off-the-shelf microcontroller because more work is required. Using the FPGA provides an advantage unavailable from the off-the-shelf microcontroller: the user can dedicate some hardware resource on the FPGA to accelerate certain operations to meet the performance requirement of the target application. Instructors of this course should remind students about this throughout the course. The instructor should also teach students to learn how to write their own intellectual property cores.

Teaching embedded system design using FPGA provides the following advantages to students:

- Familiarizing with digital design skills using one of the hardware description languages.
- Exposure to a new embedded system design alternative.

Because designing embedded system using this approach involves many additional steps than using the off-the shelf microcontroller, it is important to start with a simple configuration and get it to work to build up confidence. After building a successful simple system, the user can then add a few more peripherals into the system and try to get it to work. In this manner, the user can

learn this embedded system design approach successfully. In a semester time frame, the instructor may not have time to also teach students to design their own peripheral functions.

It is important to remind students the reason for designing embedded system using the FPGA instead of the off-the-shelf microcontroller—there are some performance requirements that cannot be met by using the off-the-shelf microcontroller.

References

1. Atmel, “SAM4L Data Sheet”, Jul, 2013.
2. Han-Way Huang, “The Atmel AVR Microcontroller MEGA and XMEGA in Assembly and C”, Delmar Cengage Learning, Clifton Park New York, 2013.
3. Han-Way Huang, “Embedded System Design with the C8051”, Cengage Learning, Stamford, Connecticut, 2009.
4. Han-Way Huang, “HCS12/9S12 An Introduction to Software and Hardware Interfacing”, 2nd edition, Delmar Cengage Learning, Clifton New York, 2010.
5. Jeff Johnson, “List and Comparison of FPGA Companies”, FPGA Developer, Jul, 2011.
6. Altera, “Introduction to the Altera Qsys System Integration Tool”, May, 2012.
7. Pong P. Chu, “Embedded SoPC Design with NIOS II Processor and Verilog Examples”, Wiley, Hoboken, New Jersey, 2012.
8. Pong P. Chu, “Embedded SoPC Design with NIOS II Processor and VHDL Examples”, Wiley, Hoboken, New Jersey, 2011.