Integrating the Hardware-Software Codesign Concept in an Introductory Digital Design Course

Pong P. Chu
Department of Electrical Engineering and Computer Science
Cleveland State University, Cleveland, OH 44115, US

Abstract

The hardware-software codesign paradigm divides the computation tasks into both software and hardware. The modern FPGA devices provide the "hardware programmability" and make this scheme more relevant and accessible. We introduce this concept in an introductory digital design course via the end-of-semester case studies on a pre-configured platform. The carefully selected cases incorporate the key course topics and at the same time demonstrate how the customized hardware accelerators can complement the software at the system level.

1. Introduction

Hardware-software codesign is broadly defined as incorporating hardware components and software components in a single design effort [1]. It divides the computation tasks between the software and the customized hardware. This design practice is frequently used in a "high-end" embedded system, such as a digital camera and a smartwatch. It includes a processor, which realizes the "general-purpose" housekeeping functions, and a collection of *hardware accelerators*, which perform computation-intensive functions and specialized I/O operations.

An FPGA (field programmable gate array) device contains generic logic cells and interconnects that can be configured to perform customized functions. Its capacity and capability increase significantly over the years and a device can easily incorporate an advanced embedded system. The FPGA based system introduces a new dimension – the *hardware programmability* – in the design process and the availability of the devices makes the hardware-software codesign paradigm more relevant and accessible. It allows a designer to explore both software and hardware to obtain optimal efficiency and performance. It will be an important paradigm for future development. Intel (the largest manufacturer of processors) recently acquired Altera (the second largest manufacturer of FPGA devices) and had discussed plans to integrate the FPGA fabric into its future processors [2].

Most computer engineering curricula follow a "layered model," in which transistor, logic gate, module, processor, operating system, and application programs constitute the individual layers. The software development and the hardware design are covered by separate courses in isolation. The subject of software-hardware codesign is normally offered as a senior- or graduate-level elective course and thus many students are not exposed to this subject. This article suggests a way to incorporate this concept in an introductory digital design course. Our approach is to use customized accelerators and I/O controllers as case studies. These cases incorporate basic digital design topics and show the partition and integration of a large system. They demonstrate how to integrate software and hardware in the same system and how the customized hardware and the software complement each other.

The remaining article is organized as follows: Section 2 discusses the development and the platform setup; Section 3 provides the detailed description of one case study - an ultrasonic distance sensor controller core; Section 4 lists other possible project ideas; and the last section summarizes the article.

2. Curriculum Development and Codesign Platform

2.1 Current Digital Design Curriculum

To manage complexity, computer system development emphasizes the abstraction and adopts a layered model. The computer engineering curricula basically follow the model and organize the courses according to the layers. The digital design course covers the gate layer and module layer and is a key knowledge area specified by IEEE/ACM Computer Engineering Curricula 2016 [3]. The basic topics are Boolean algebra and logic gates, module-level combinational circuits (such as a decoder, a multiplexer, and an adder), module-level sequential circuits (such as register and counter), FSM (finite state machine), and controller and data-path.

Most digital design curricula include a few more sophisticated case studies in the end. A commonly used case is the implementation of a “toy CPU” with a control unit and an ALU (arithmetic and logic unit).

2.2 Hardware accelerators as case studies

The toy CPU case study is adequate for the layered model since it provides a preview of the next abstraction layer. However, the hardware-software codesign concerns the “vertical integration” over multiple layers and incorporates the hardware programmability. In our development, we select a new set of case studies – hardware accelerators – to introduce the paradigm and concept.

The hardware acceleration uses a customized digital circuit to realize a software algorithm. It can reduce processor overhead and exploit parallelism and thus can speed up the computation. The hardware accelerators are mainly used for computation-intensive algorithms, such as those in signal processing and computer vision. However, many simple functions and specialized I/O controllers can be derived as examples in an introductory digital design course.

Software-hardware co-development is a complex process and involves sophisticated EDA (electronic design automation) software packages. It is unlikely to provide comprehensive coverage in the introductory course. Our approach is to provide an established hardware platform with a predefined hardware accelerator slot and a preconfigured software development environment. After a brief overview of the platform, the main discussion can be steered to the design and development of the example hardware accelerator.

2.3 Predesigned Hardware Platform

The block diagram of the pre-designed hardware platform is shown in Figure 1. It is a simple computer system with four basic I/O peripherals, which are a timer core, a UART core, a general-purpose input core (GPI) core, a general-purpose output core (GPO) core, and a reserved hardware accelerator (HA) slot. Today’s FPGA devices are very capable and can support a soft-core processor-based system (such as Altera’s Nios II or Xilinx’s MicroBlaze). The entire system can be easily implemented on entry-level FPGA boards [4].

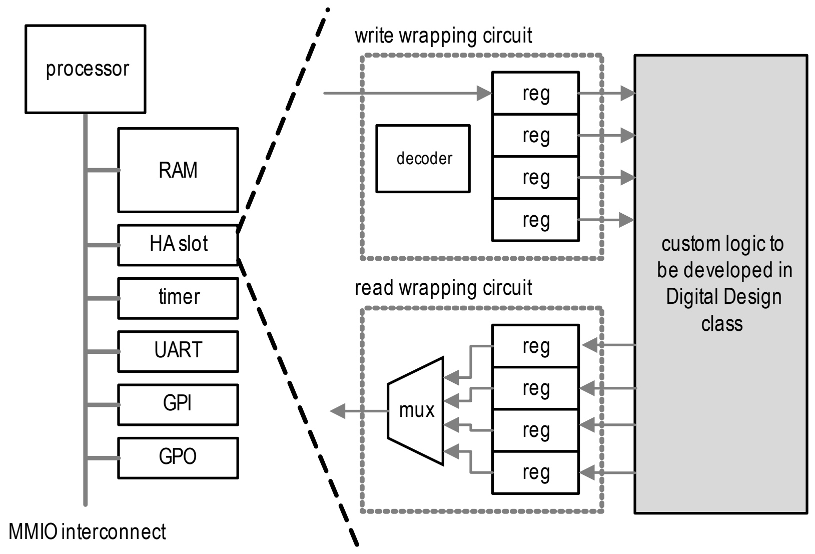

Figure 1. Hardware platform

Deriving the complete system requires to use FPGA vendor's integration tools and IP (intellectual property) cores. It is done in advance and provided to students. The hardware accelerator slot is included and reserved as a "placeholder." In the subsequent development, the design (HDL codes) can be inserted into the slot, which is shown as the shaded area of Figure 1. The subsequent processing only requires re-synthesis but does not involve any IP tool.

2.4 Memory-Mapped I/O Accelerator Core

The modern embedded processor uses the memory-mapped I/O scheme, in which an I/O module is treated as a collection of registers and shares the same memory address space. A hardware accelerator consists of the customized circuit to perform the desired functionality and a "wrapping circuit" to interface with the bus (which is actually FPGA's "interconnect" structure). The read and write warping circuits with four registers are shown in the enlarged portion of Figure 1.

2.5 Pre-Configured Software Environment

The software development of a soft-core based system consists of two major steps: creating the BSP (board supporting package) library and deriving the application program. The BSP library contains the startup code and device drivers and is derived based on the hardware configuration. It provides the utility routines to be used in an application program.

Our hardware setup includes a UART core and a timer core and thus the BSP will infer routines to display text on a serial console (such as `printf()`) and timing related utilities (such as `delay()` and `now()`).

During the hardware construction, the hardware accelerator slot is assigned a base address. The application code can access its I/O registers by reading and writing the designated addresses (as pointers). A pair of macros can make the register read and write operations more expressive:

```
#define io_rd(addr) (*(volatile int *)(addr)
#define io_wr(addr, data) (*(volatile int *)(addr) = (data))
```

Since the configuration of the predesigned hardware platform does not change, the BSP library only needs to be created once and the hardware accelerator's address remains the same. The software development environment can be created in advance and students just need to derive the C code to access the I/O registers and recompile the main program.

3. Detailed Case Study: Ultrasonic Distance Sensor

3.1 Overview of HC-SR04

The HC-SR04 module is an inexpensive (about US $3) ultrasonic distance sensor, as shown in Figure 2 [5]. It measures the elapsed time of a reflected ultrasonic pulse. The distance then can be calculated by a simple formula:

```
distance = elapsed_time * speed_of_sound / 2
```

The measurement range of the module is between 2 cm to 400 cm and the resolution is 3 mm (which corresponds to the17-μs elapsed time). Its main application is to help the robotic navigation by detecting the obstacles in the path.

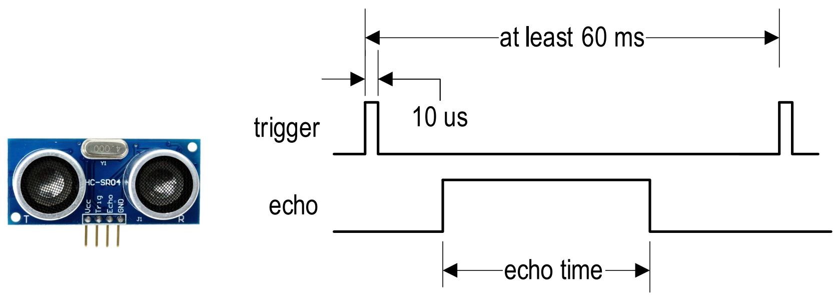

Figure 2. HC SR04 module and timing

The module's interface includes a `trigger` signal (as input) and an `echo` signal (as output). When the `trigger` signal is asserted, the module generates an ultrasonic pulse and waits for the reflected pulse. The `echo` signal is asserted during the waiting period and the interval corresponds to the elapsed time. The timing diagram is shown in Figure 2. To prevent interference, the module imposes a 60-ms cycle time (i.e., the triggers must be separated by at least 60 ms).

3.2 Software code and its caveats

The module can be incorporated into the system by connecting the trigger pin and the echo pin to the GPO and GPI cores. The software code can be derived following the timing diagram in Figure 2:

```
// generate 10-us pulse over trigger pin
io_wr(gpo_addr, 1);
delay(10);
io_wr(gpo_addr, 0);
// wait for 1 on echo pin and record the start time
while (io_rd(gpi_addr) == 0){};
t_start = now();
// wait for 0 on echo pin and record the end time
```

```
while (io_rd(gpi_addr) == 1){};
t_end = now();
// calculate distance in cm
time = (t_end – t_start)/1000000; // in sec
sound_speed = 34000;              // in cm/sec
distance = time * sound_speed / 2.0;
// wait for 60-ms turn-around time
delay(60000 - t_start);
```

While the code is simple and straightforward, it is not very efficient because of the "busy-waiting" time, in which the processor is idle and the execution is blocked. The polling or interrupt scheme can alleviate the problem to some degree. However, to maintain the accuracy of 3 mm, the assertion interval of the echo signal must be measured promptly and the polling cycle or the interrupt service latency must be within 17 μs. This leads to a tight timing constraint. Furthermore, a sophisticated robot prototype may use multiple modules to check the obstacles in all directions at the same time. It can further complicate software development.

Within the software-hardware codesign context, an alternative is to develop a customized hardware controller core to implement the desired functionality.

3.3 Hardware controller

The HC-SR04 controller contains a control path and a data path and its design follows the timing diagram. The controller includes the `start` signal, which initiates the measurement, and the `ready` status signal, which indicates whether it is available (i.e., not busy). The measurement is placed on the `t` signal, which is the elapsed time in terms of number of clock cycles.

The control path is an FSM that traverses through the trigger pulse generation, elapsed time counting, and turn-around waiting. The data path contains a counter (`c`), which runs continuously and a register (`t`), which stores the elapsed time. The conceptual state diagram is shown in Figure 3. It assumes that the system clock is 100 MHz (i.e., 10-ns period).

Note that the basic operation of the controller is to wait and measure various events. It can be thought that a dedicated counter to do the "busy-waiting" previously performed by the processor.

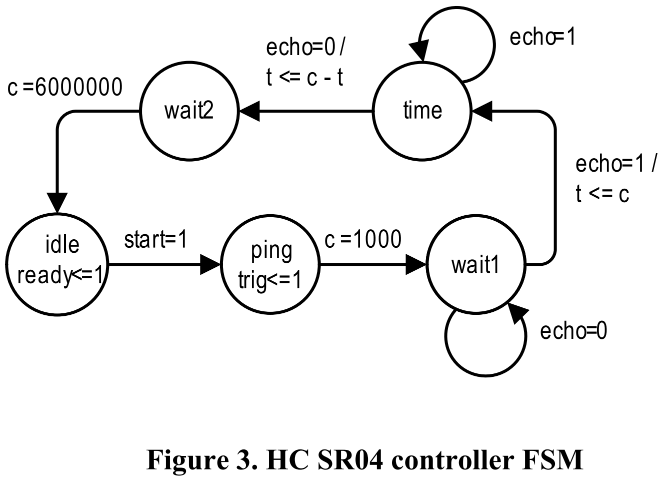

Figure 3. HC SR04 controller FSM

3.4 Customized I/O core

A custom I/O accelerator core can be derived by "wrapping" the controller with a proper bus interface circuit. To make the core more flexible, we can define two operation modes. In the single-measurement mode, the controller is normally idle and software must send a command (to assert the `start` signal) for each measurement. In the continuous mode, the controller performs the measurement continuously, which is achieved by connecting the `start` signal to 1.

Based on the specification, four I/O registers can be defined: register 0 (write only) for the operation mode, register 1 (write only) for the initiation of a single-measurement operation, register 2 (read only) for the ready status, and register 3 (read only) for the elapsed time in terms of system clock cycles. The wrapping circuits are similar to those in Figure 1.

The HDL code can be inserted into the previously reserved accelerator slot and the hardware platform then can be resynthesized accordingly. Assume that the continuous mode is used. The revised software code becomes:

```
// set the mode; do it only once
io_wr(accelerator_addr+0, 1);
. . .
// read the last measurement as needed
cycle = io_rd(accelerator_addr+3);
// convert 10-ns clock cycles to sec
time = cycle * 10 / 1000000000;
// calculate distance in cm
sound_speed = 34000;      // in cm/sec
distance = time * sound_speed / 2.0;
```

The core can be easily expanded to facilitate multiple modules. For example, assume that four modules are needed in a robotic platform. The core can be constructed by instantiating four HC-SR04 controllers and expanding the registers. Since all operation is done by the controllers, the number of modules does not affect software execution.

3.5 Topics learned

The design and development of the SR04 controller core are not very complicated and can be done in one or two lectures. Despite its simplicity, it includes all the key topics of the digital design course and integrates them together in a single case study. The read and write interfaces cover the decoder, the encoder, and the multiplexer of the module-level combinational circuits, the registers, and the counters covers the module-level sequential circuits, and the controller demonstrates the use of an FSM and the construction of a data-path. Furthermore, the case study introduces the basic concept of hardware-software codesign and illustrates the digital design in a system-level context.

4. Hardware Accelerator Modules

There are several other topics that can be used as the end-of-class case studies and projects. Like the SR04 controller core, they are based on simple algorithms and can illustrate the hardware-software codesign concept. These topics are discussed in the following subsections.

4.1 Simple Mathematics Functions

Certain arithmetic functions involve only integer operation and can be described by simple algorithms. They can be realized by hardware accelerators. One function is to determine the GCD (greatest common divisor) of two numbers, which is defined as

$$\gcd(a,b)=\begin{cases} a & a=b \\ \gcd(a-b,b) & a>b \\ \gcd(a,b-a) & b>a \end{cases}$$

Other functions include the binary Euclid's algorithm, the Fibonacci numbers, and Newton's method of finite differences. These functions can be realized by software routines or constructed by an FSM controller with a data-path [4].

4.2 Digital Filters

A digital filter performs mathematical operations on a sampled, discrete-time signal. Its general format is

$$y=\sum_{i=0}^{n-1} k_i * x(i)$$

where k_i is a coefficient and *x(i)* is a sampled value.

The operation is referred to as multiply-accumulation (MAC) and is the most common steps used in digital signal processing. It includes n multiplications and is inherently parallel.

The digital filter is a good case study to demonstrate the exploration of parallelism with hardware. Let n be 8. In the software implementation, the function is done by a simple for-loop, in which the multiplications are performed sequentially eight times. There is a variety of ways for the hardware implementation. The operation can be realized by an FSM with a data path. A different number of multipliers (such 1, 2, 4, or 8) can be assigned to the data path to obtain the desired speed-area trade-off.

4.3 Customized I/O controllers

An embedded system contains a variety of sensors and actuators. Since the data rate is low, most devices communicate via the standard serial UART, I^2C, or SPI interface. The processor has pre-built I/O controllers for these interfaces. However, some devices have less common 1-wire interface, such as the DS18S20 digital thermometer, or ad hoc interfaces, such as the DHT22 temperature and humidity sensor and the WS2812 "addressable" tricolor LED. The SR04 module discussed in Section 3 is also in this category.

To interact with a low-rate non-standard interface, the processor can use software code to generate the output patterns and sample the input with the general-purpose I/O cores. This technique is referred to as the bit-bang scheme. The code in Section 3.2 demonstrates this scheme. As discussed in Section 3.2, the bit-bang code wastes processor's time and imposes unnecessarily tight timing constraints. The code becomes very complexes when a system needs to interact with multiple devices. A customized I/O core, like the one in Section 3.4, is a more effective alternative. Based on the timing specification of each device, a dedicated hardware controller can be developed and integrated into the system to offload the processor.

4.4 Square Wave Generator

A square generator produces a wave oscillating between 0 and 1. Software implementation can be realized by using the bit-bang scheme:

```
// assume that the period of the square wave is t us
while(1){
   io_wr(gpo_addr, 0);
   delay(t/2);
   io_wr(gpo_addr, 1);
   delay(t/2);}
```

The code suffers the same efficiency problem discussed earlier. Furthermore, even for a dedicated processor, the maximal clock rate is limited because of the clock rate of the processor and the overhead associated with software execution.

The hardware alternative is to use the DDFS (direct digital frequency synthesis) scheme [4]. Its design includes an adder and a register and can achieve a much higher rate (up to one half of the system clock rate). Furthermore, an additional lookup table and a DAC can be added to generate a continuous analog wave, which cannot be accomplished by software. With the accelerator, the processor only needs to write the frequency control word to an I/O register to set the frequency.

5. Assessment

The work is part of the development to create a "spiral lab framework" for the entire computer engineering curriculum [6]. The effectiveness of the development is evaluated by an array of assessment instruments, including contents tests, lab works, student survey, and student interviews, and the result is reported in [6]. Since the assessment is aimed at the semester-long course, it does not evaluate individual topics or projects. However, we believe that the introduction of hardware-software integration helps to obtain overall positive responses.

6. Summary

The advancement and availability of FPGA offer a new opportunity for "hardware programmability" and make the hardware-software codesign scheme more relevant. We use the updated case studies to introduce this subject in a digital design course. These case studies integrate the key topics studied in the course and at the same time demonstrate how to incorporate dedicated hardware accelerators in a computer system and how the customized software and the customized hardware can complement each other to obtain the optimal efficiency and performance.

Acknowledgments

Part of the material is based upon work supported by the National Science Foundation under Grant No. #1504030. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Bibliography

[1]. Schaumont, Patrick, *A practical introduction to hardware/software codesign,* Springer Science, 2012.

[2]. https://insidehpc.com/tag/intel/, Inside HPC/Intel, Are FPGAs the answer to the "Computer Gap?" 2016.

[3]. ACM/IEEE Computer Society, *Computer Engineering Curricula*, 2016.

[4]. P. Chu, *FPGA Prototyping by VHDL Example, 2nd ed.*, Wiley-Interscience, 2017.

[5]. www.elecfreaks.com, *HC-SR04 User Guide*.

[6]. P. Chu, Chansu Yu, and Karla Mansour, "Integrating Computer Engineering Lab Using Spiral Model," *Proceedings of ASEE Annual Conference, 2017*.