Efficient Resource Allocation for FPGA Demo Board Based Digital Laboratories

Chia-Jeng Tseng

Department of Electrical Engineering
Bucknell University
Lewisburg, Pennsylvania 17837

Abstract

Due to its low cost and convenience, a field-programmable gate array (FPGA) demo board is often used in universities for teaching digital design. The major limitations of an FPGA board include a small number of input and output options and limited high-level software capability. In order to show students how to overcome the resource scarcity, we have developed several digital laboratories to help students creatively explore possible solutions. In this paper, we discuss design considerations for managing various resource limitations. Also, we present several laboratory assignments for students to practice these design considerations using an FPGA board. These laboratories not only provide students with opportunities to practice subsystem design, but also teach them various system integration techniques.

1. Introduction

FPGA boards are widely used for digital laboratories in universities[12,15,16]. Normally, an FPGA board contains an FPGA chip, input and output devices, a clock source, and supporting circuitry for downloading a bit-stream into the FPGA. Commonly seen input devices include dual in-line package (DIP) and push-button switches; output devices are seven-segment and bar-graph light-emitting-device (LED) displays. Some FPGA boards may contain computer, codec, and network interfaces. Since there are only a limited number of input and output options available, creative methods are required for efficient application of these resources.

In this paper, we address the issues of overcoming resource constraints normally encountered when using an FPGA demo board. Section 2 discusses considerations for exploring efficient resource allocation. Section 3 presents several laboratories for students to practice the design considerations described in Section 2. These basic laboratories were instrumental for teaching system design ideas such as stored program control, embedded systems, as well as rapid prototyping. Finally, Section 4 contains concluding remarks of the paper.

2. Design Considerations for Efficient Resource Allocation

In this section, we identify some efficiency considerations for input devices, output interfaces, memory structures, data conversions, logic implementation options, and clocking signals generation. Normally, we instruct students to use DIP switches to define the system modes of a digital circuit. For example, two DIP switches can define four different system modes such as initialization, input, output, and a function-specific mode. A push-button pulse signals the triggering of the operations for a system mode.

2.1 Input Capture

Commonly seen input devices include a DIP switch, a push-button switch, and a transducer input. A DIP switch is considered to be a simple input device due to its stable input status. The input value can be latched through regular sampling or read using a second push-button. The input pulse of a push-button can be captured by a flip-flop; a debouncing mechanism may be needed to ensure its proper operation[11]. An alternative is to use a finite state machine to reliably capture an input pulse request[9]. An analog-to-digital converter is generally needed to retrieve the input value from an input transducer. For example, a CODEC may be used to sample a speech input and to convert it into digital form[12].

Numeric data are often required for a circuit. DIP switches are convenient for defining a binary number; the methods described so far are adequate for reading this type of data. A binary-coded-decimal (BCD) number is, however, a much more user-friendly interface. How can we enter a large decimal number into an FPGA board? In this subsection we describe three alternative methods of capturing input data in the BCD format.

The first method uses four DIP switches to define a decimal digit between zero and nine. Then, it uses a push-button to capture the decimal digit identified by the four DIP switches. If a number between 10 and 15 is entered, the number is ignored. If a decimal number consists of multiple BCD digits, use the following procedure to read the input number. Starting from the leftmost digit, capture one BCD digit at a time by pressing a push-button switch once. The input capture process is terminated by switching the system mode of the circuit.

In the second approach, a counter and a push-button is used to trigger the generation and display of a decimal digit. Starting from zero, the value stored in the counter is incremented by one each time a push-button is pressed. A push-button press while the number nine is being displayed resets the counter to zero. This decimal number generation process can be repeated. A second push-button is then used to capture a displayed decimal digit. The input capture is deactivated when the system mode of input capturing is terminated.

Finally, the third method uses a free-running BCD counter to generate a decimal digit between zero and nine at a low speed. A push-button press is then used to read the digit being displayed. The input capture process is terminated by changing the system mode.

Assume that the input is from a transducer and an analog-to-digital converter (ADC) is used to convert analog signals into digital format. If the ADC periodically converts an analog input into

a twelve-bit digital data and the FPGA chip does not have enough pins to input the sampled data, a scanner consisting of a multiplexer driven by a counter may be used to transform the parallel data into a serial one. Therefore, a single pin may be used for data input. Inside the FPGA chip, the serial input data are stored into a shift-register bit by bit. If there is a need, a second register may be used as a buffer to synchronize updating of the twelve-bit data. The data processing inside the FPGA chip can be isolated from the input interface register. Without a doubt, the design works only if appropriate clocking schemes are implemented. Figure 1 depicts the data flow of the input capture scheme and Figure 2 shows the structure of a three-bit shift-register.

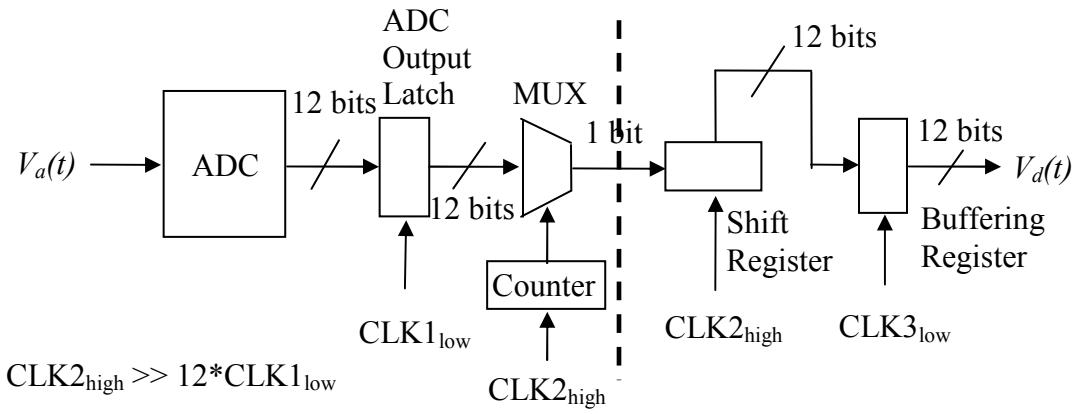

Figure 1: A High-Speed Data Transfer and Buffering Scheme for Input Capture

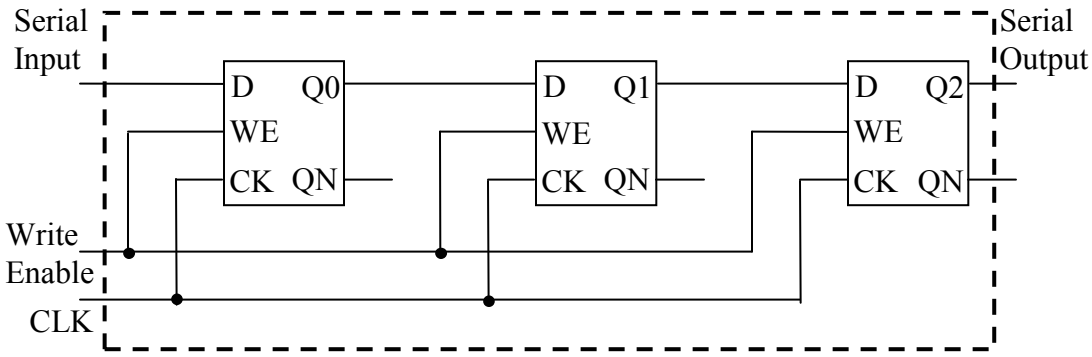

Figure 2: A Shift-Register

2.2 Output Presentation

When the circuit is set to the output mode, a push-button is used to initiate the output display. If a binary format is used, an output vector can be displayed using bar-graph LEDs. If the BCD format, which is more user-friendly, is preferred, the output data should be converted into a BCD representation. Starting from the leftmost digit, one digit is displayed at a time. Each digit is displayed for a short period of time. The seven-segment display can be turned off briefly between the display of two digits. After the rightmost digit is displayed, the seven-segment display is turned off. This process can be repeated by pressing the push-button again. The output display is deactivated by switching the system out of the output mode.

Similar to the input interface described in Subsection 2.1, a major concern for output interface is how to satisfy a demand of a large number of output pins. For example, an alarm-clock design may require 42 output pins to display the six decimal digits using a normal BCD to seven-segment mapping approach. One may consider sharing output pins for the hour-, minute-, and second-data. Then, the number of output pins required is reduced to fourteen. The simplest approach may use one bit for the hour-, minute-, and second-communication link. Inside the FPGA chip, the data are scanned in a bit-by-bit manner and transferred to a 42-bit shift-register placed in a breadboard outside the FPGA demo board. Six seven-bit flip-flops may be used as buffers to synchronize the update of LED display. Figure 3 depicts the data flow. In general, a scanner coupled with high-speed data transfer and buffering is an efficient method for overcoming input and output pin constraints.

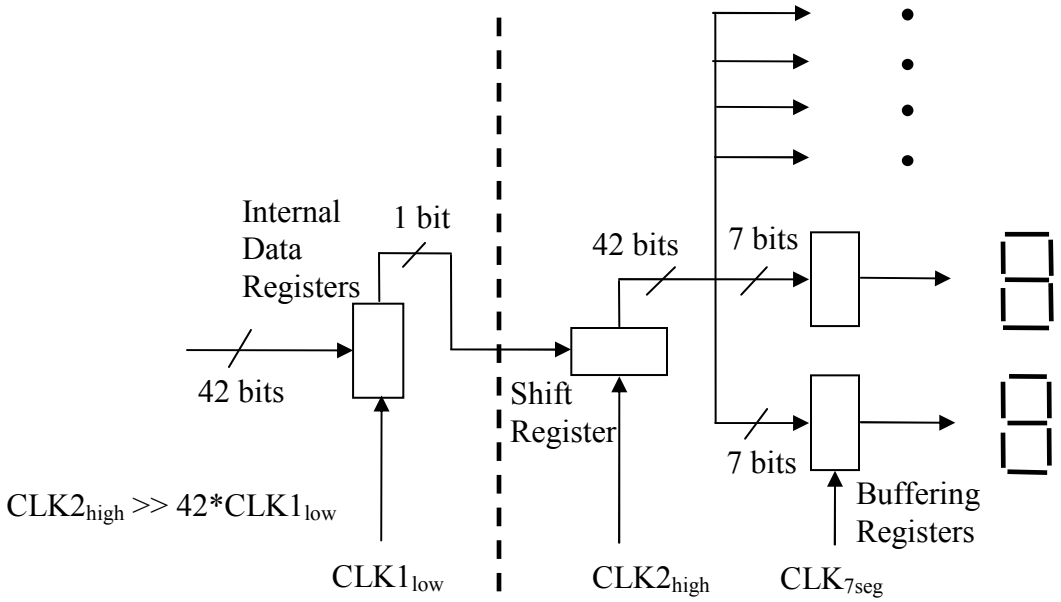

Figure 3: A High-Speed Data Transfer and Buffering Scheme for Output Processing

2.3 Memory Structures

Memory is an important component in modern computer design. As a matter of fact, memories form the fundamentals for many powerful design schemes such as stored program control and embedded systems. As detailed in Subsections 2.1 and 2.2, incremental approaches are used for input and output interfaces to overcome the resource constraints in our laboratories. Memories become critical hardware structures for students to learn.

Memory is often defined as a two-dimensional array, which consists of a specified number of words. Each memory word is assumed to contain a fixed number of bits. There are numerous ways of organizing a memory block. However, it is difficult to generate a working design in some VHDL specification methods using commercial synthesis tools[13,14]. For example, the following statements are a concise way of specifying a memory of 8 sixteen-bit words. Most synthesis tools were not able to generate a working structure. In fact, it is difficult for a synthesis tool to determine an appropriate access structure for a behavioral specification of a memory block.

```
type memory is array (7 downto 0) of std_logic_vector (15 downto 0);
signal memory_block: memory;
```

One way of specifying the aforementioned memory block, as shown in the following VHDL statement, is to define it as a one-dimensional array of bits.

```
signal memory_block: std_logic_vector (127 downto 0);
```

To enable a synthesis tool to generate a working memory-block design, the access mechanism must be clearly specified. The organization depicted in Figure 4 was adopted. In addition to a memory array, the memory block contains a memory address register (MAR), an input memory data register (IMDR), and an output memory data register (OMDR). To write a vector of bits into a memory cell, the following three data transfers are described for the memory access interface: a memory address is stored into the MAR, a data vector is moved into the IMDR, and a memory write enable signal is asserted. To read a memory cell, one needs to set the MAR and to latch the memory data into the OMDR. Indeed, a working memory block can be synthesized only if the access mechanism is properly defined. To create a generic memory component, a parameterized VHDL process which details the access mechanism using these interface registers must be defined.

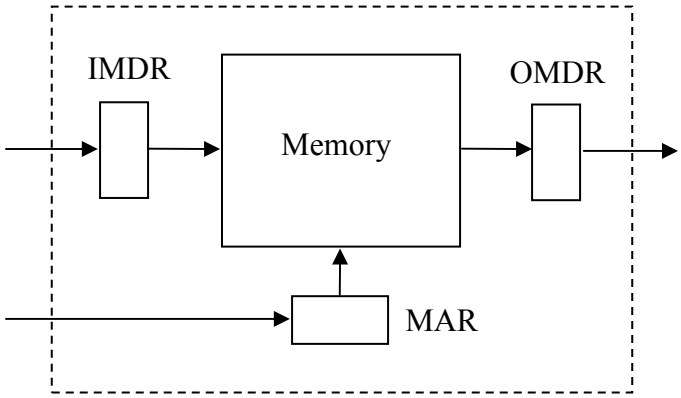

Figure 4: A Memory Organization

2.4 Internal Data Conversion

The emphasis of using BCD for input and output interface poses the needs of binary-to-decimal and decimal-to-binary data conversion. A multiple-digit decimal integer may be converted into its equivalent binary value by combining successive multiplication of ten and addition. Since multiplication is a complex operation in digital design, we suggested students to perform ten additions as a substitution for the operation of multiplying by ten. For a fractional decimal number, the students were advised to derive the closest binary number for each decimal digit based on the placement of binary points and the number of fractional bits. These decisions teach students how to take practical considerations into account. Instead of using division, the same approach accompanied by a ceiling comparison is an efficient method for converting a binary number into its decimal equivalent.

2.5 Logic Implementation Options

There are often a number of ways for implementing a mathematical function. For example, a multiplier can be implemented as a combinational circuit or as a sequential circuit. A combinational circuit implementation for a multiplier is generally straightforward. However, it may suffer from excessive delay and area requirements. A sequential implementation may be more complex. However, it often turns out to be an appropriate choice for reducing circuit area and delay.

Given a sixteen-bit binary number, let the leftmost bit be used for the sign bit, the next seven bits be used for the integer part, and the rightmost eight bits be used to represent the fractional part. In other words, the binary point is placed between the position between the eighth and ninth bits. The data paths and control flow of a sequential multiplier is depicted in Figure 5 and Figure 6, respectively. The organization happens to be a great selection for capturing decimal data in the range of -99.99 to + 99.99. Again, this design shows students how to make an appropriate

decision for the number of bits allocated to the integer and fractional parts as well as where to place the binary point.

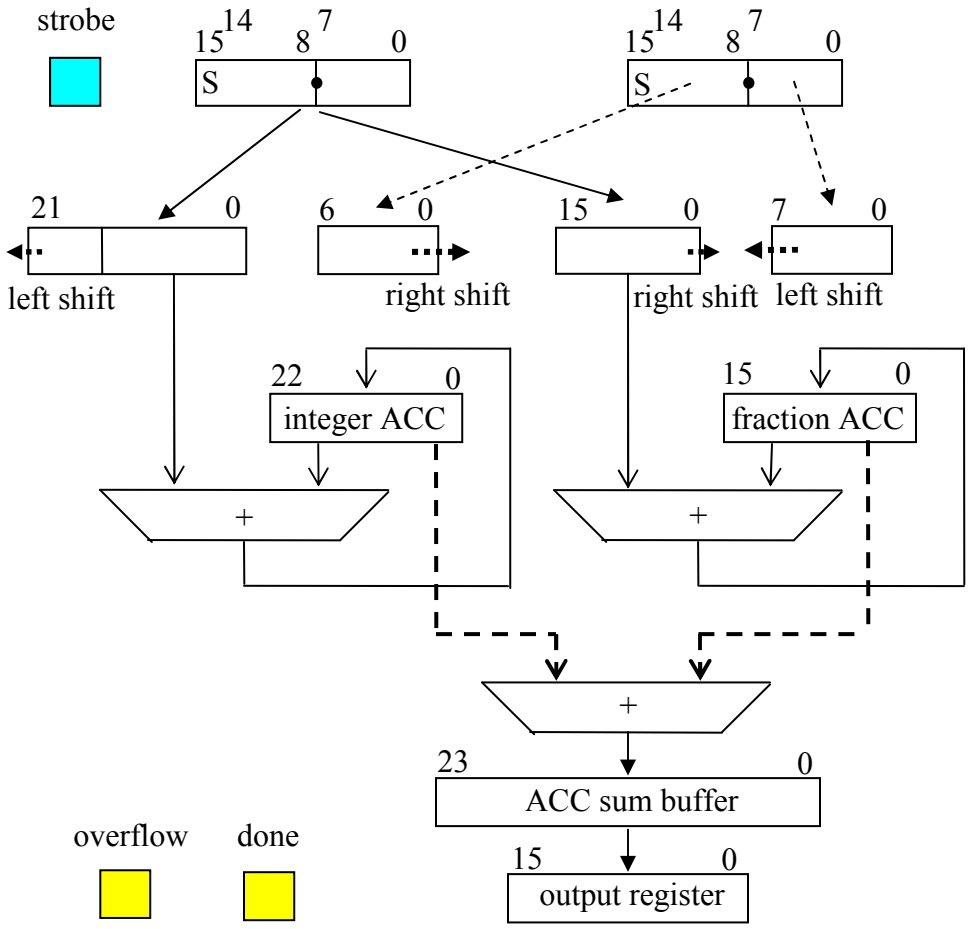

Figure 5: Data Paths of A Sequential Multiplier

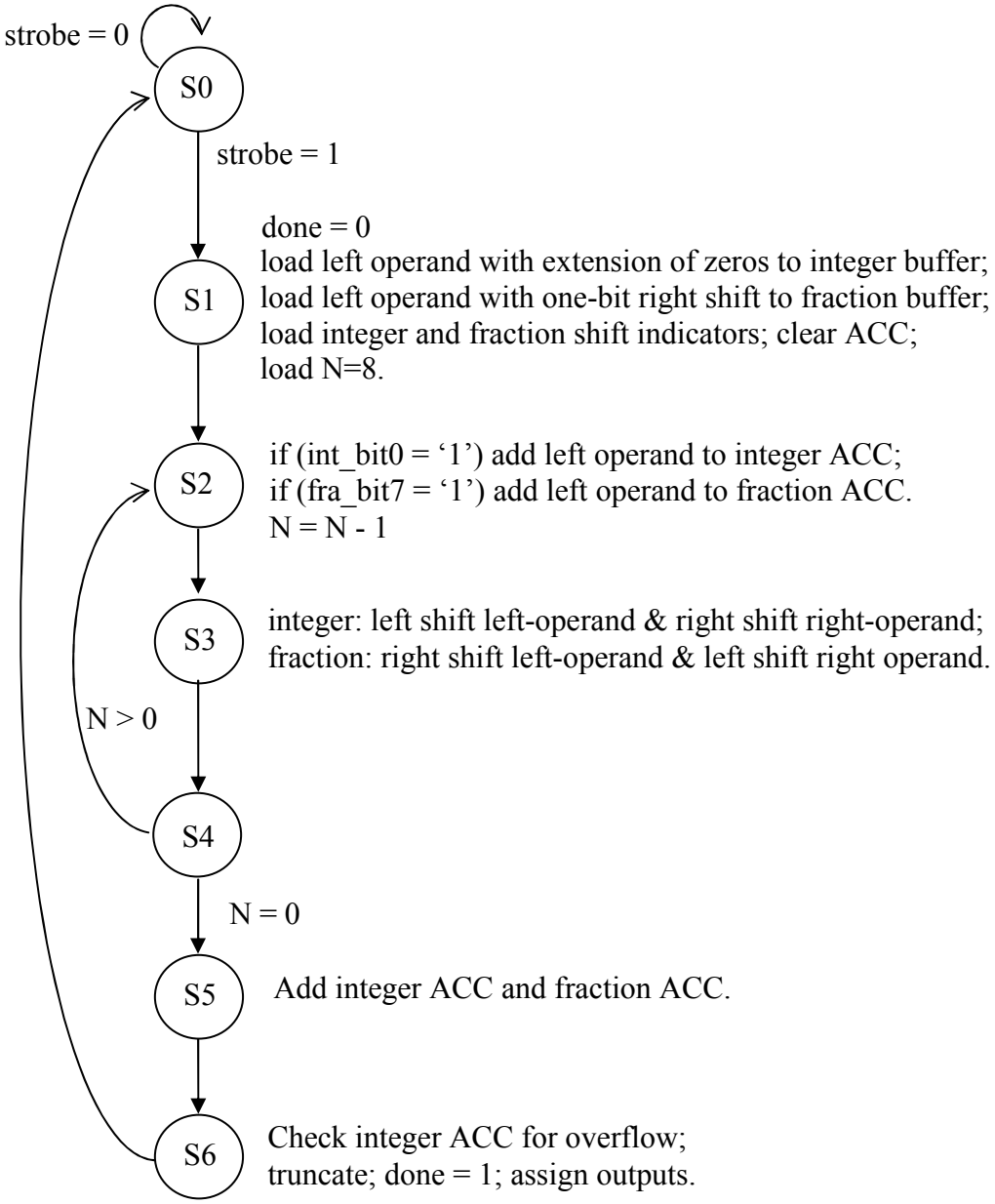

Figure 6: Control Flow of A Multiplier

2.6 Clock Generation

By now, it is very clear that the clock signals used by various parts of a digital design must be efficiently scheduled. Indeed, the generation of periodically synchronized clock signals with required frequencies is essential to the design of these digital laboratories. Students were instructed to use a counter to derive the needed clock signals from an external clock source. They also studied the impact of the duty cycle of a clock signal on a digital design. If it is necessary, a buffering flip-flop may be used to produce a clock signal with 50% duty cycle.

3. Laboratory Assignments for Practicing Efficient Resource Allocation

The methodologies described Section 2 were taught in an "Advanced Digital Design" course at Bucknell University. The course was comprised of three hours of lecture and laboratories, respectively, on a weekly basis. The lectures covered the VHDL[2], advanced topics in logic optimization, and high-level synthesis. Logic optimization focused on Quine-McCluskey method[5,6], multiple-level logic optimization[3], and technology mapping for standard-cell implementation[4]. High-level synthesis addressed the issues of scheduling, clique-partitioning for data-memory synthesis[7], control synthesis, and behavioral transformations for design space exploration[8]. The methodologies of micro-architectural modeling[10] and efficient clocking schedule[9] were emphasized for the design of complex digital circuits. The laboratories included several basic VHDL assignments and complex projects for learning subsystem design and efficient system integration.

Occasionally, some students were overwhelmed by the complexity of digital system design. To alleviate the problem, several simple VHDL assignments were defined for students to practice the essential design considerations detailed in Section 2. These VHDL assignments turned out to be very instrumental for preparing the students to complete other complex laboratory assignments. This section discusses how these design considerations were embedded in the laboratory assignments.

First, a Hamming code generator and a Hamming code receiver were assigned to students for them to learn how to specify a combinational circuit in VHDL. The two VHDL programs required students to consider all the input combinations for logic completeness. Some of the lessons that students were able to derive from this simple assignment are listed below:

- The importance of avoiding the specification of a multiply-defined logic function.
- The specification and applications of parity functions.
- The difference between logic minimization and logic partitioning.

The second assignment was about the design of a clock signals generator. Students practiced how to produce clock signals of different frequencies. These clock signals were derived from the same source so that some of these clock signals could be periodically synchronized. The students were requested to analyze the duty cycle of each clock signal using an oscilloscope. Also, they studied how to generate a clock signal with 50% duty cycle.

The third project involved reading five BCD numbers, storing them into a memory, and reading memory words one by one and displaying each on a seven-segment LED. This project allowed

students to practice the specification of memories, input capture, and output display. The assignment also forced students to address the implications of five, instead of eight, memory words. They studied the methodologies of micro-architectural modeling and efficient scheduling of clock signals as well. These methodologies enable students to produce a working design for any computer algorithms that can be described by a high-level procedural language.

In addition to these three basic VHDL assignments, three system projects were assigned to students. These three projects included a motion guide[9], an alarm clock, and a discrete cosine transform[1,10]. The alarm clock allowed students to compare direct output of time data using 42 pins and the scheme of applying parallel-to-serial, serial-to-parallel, high-speed data transfer and buffering techniques for output management. The discrete cosine transform provided them an opportunity to study number systems, digital organization for numeric operations, internal data conversion techniques, and the multiplication scheme described in Subsection 2.5.

The three basic VHDL assignments are included in this section for those readers who are interested in additional details.

3.1 A Hamming Code Transmitter

Given a four-bit code, a Hamming code contains four additional parity bits to support single-bit error correction and two-bit error detection. Assume that all four parity bits are of even parity. Write a VHDL program which will take a four-bit binary codeword as its input and produce an eight-bit Hamming codeword as its output such that the new codeword may support the purpose of single-bit error correction and two-bit error detection. Assume that the eight bits of a Hamming codeword are indexed by 1 to 8 from right to left; the parity bits are indexed by 1, 2, 4, and 8. Use four DIP switches on the FPGA board as the input devices and eight LED bars on the FPGA board as the output display.

Let the four DIP switches be denoted by *dipsw(1)*, *dipsw(2)*, *dipsw(3)*, and *dipsw(4)*. Also, assume that the eight bits of the resultant Hamming codeword are represented by *P8*, *H7*, *H6*, *H5*, *P4*, *H3*, *P2*, and *P1*. The four data bits *dipsw(1)*, *dipsw(2)*, *dipsw(3)*, and *dipsw(4)* correspond to the bits *H3*, *H5*, *H6*, and *H7*, respectively, in the Hamming codeword.

Below are four equations defining the four parity bits of a Hamming codeword for your reference.

P1 = dipsw(1) xor dipsw(2) xor dipsw(4);
P2 = dipsw(1) xor dipsw(3) xor dipsw(4);
P4 = dipsw(2) xor dipsw(3) xor dipsw(4);
P8 = dipsw(1) xor dipsw(2) xor dipsw(3) xor dipsw(4) xor P1 xor P2 xor P4;

3.2 A Hamming Code Receiver

Assume that even parity is used for all four parity bits of an eight-bit Hamming code. Write a VHDL program, which takes an eight-bit Hamming codeword as its input and shows whether an error is contained in the codeword. Use an LED segment to show that a single-bit error is

detected; the eight-segment LED should display the corrected codeword. Use a separate LED segment to show that a two-bit error is detected. Both LED segments should be off if no error is detected.

3.3 A Clocking Signals Generator

Clock signals generate the heartbeats for a synchronous sequential circuit. In other words, a synchronous sequential circuit needs clock signals to drive its operation. For example, a finite state machine, in addition to input signals, requires a clock signal to trigger state transition.

An oscillator is often used in a digital system to generate a clock signal of stable frequency. In each FPGA board that we use, a 100 mega hertz clock signal is available from an integrated-circuit oscillator in the FPGA board. Inside a digital design, a counter is often used to generate clock signals of lower frequencies. In this assignment, write a VHDL description of a counter to derive clock signals of the following frequencies.

- 1 mega hertz.
- 10 kilo hertz.
- 1 kilo hertz.
- 10 hertz.
- 1 hertz.

Use an oscilloscope to display the clock signals that you generate. Examine each clock signal. Does it have a 50% duty cycle? If it does not, how would you generate a clock signal with 50% duty cycle?

3.4 Input Capture and Output Display of Decimal Numbers

In this VHDL assignment, you will first write a VHDL description to display a decimal number using a 7-segment LED. Then, you will experiment with three methods of capturing an input decimal number. In each case, a captured number will be displayed by a 7-segment LED. We assume that a decimal number may consist of one to five decimal digits.

Each design for capturing a decimal number will consist of four system modes including idle, input capture, output display, and global reset; use two DIP switches to identify the active system mode: 01 for input capture, 11 for output display, 10 for global reset and 00 for idle mode, respectively.

3.4.1 Display of A BCD Number

Use four DIP switches to define a BCD number. Display a BCD number using a seven-segment LED display. If a number between 10 and 15 is entered, all the LED segments should be turned off. Write a VHDL program of a digital circuit which performs the aforementioned function.

3.4.2 Generating Decimal Digits Using DIP Switches

Use four DIP switches to define a decimal digit between zero and nine. Then, use a push-button to capture the decimal digit identified by the four DIP switches. If a number between 10 and 15 is entered, the number should be ignored. If a decimal number consists of multiple BCD digits, use the following procedure to capture the input number. Starting from the leftmost digit, capture one BCD digit at a time by pressing a push-button switch once. The process of an input capturing process is terminated by switching the system mode of the circuit. When the circuit is set to the output mode, a push-button is used to initiate the output display. Starting from the leftmost digit, display one digit at a time. Each digit is displayed for a short period of time. The LED segments should be turned off for half a second between the displaying of two digits. After the rightmost digit is displayed, the LED segments are turned off. This process can be repeated by pressing the push-button again.

3.4.3 Capturing Decimal Digits Using an Externally-Controlled Counter

In this case, you will use a push-button to trigger the generation and display of a decimal digit. A counter is used to store a BCD number. Starting from zero, the value stored in the counter is incremented by one each time a push-button is pressed. A push-button press while nine is being displayed resets the counter to zero. This decimal number generation process can be repeated. Use a second push-button to capture a displayed decimal digit. The input capture is deactivated when the system mode of input capturing is terminated. Similar to the previous design, the output display is activated by a push-button while the system is set to the output mode.

3.4.4 Decimal Digits Generation Using a Free-Running BCD Counter

Use a free-running BCD counter to generate a decimal digit between zero and nine at a low speed. A push-button press will capture the digit being displayed. The input capturing process can be terminated by changing the system mode. Similar to the previous cases, the output display is initiated by a push-button while the system is set to the output mode.

3.4.5 Additional Notes

Figure 7 depicts a framework for students to develop a complex laboratory. The clocks generator and user-friendly interface circuits are used in the test phase. After a design is verified to be functional, the real input and output can be attached to the application circuit. Numerous lessons can be derived from the development of the utility design. The following list includes a few of the lessons:

- Micro-architectural modeling technique.
- Memory structures.
- Intellectual property (IP) module reuse.
- Stored program control and embedded systems.
- Rapid prototyping.

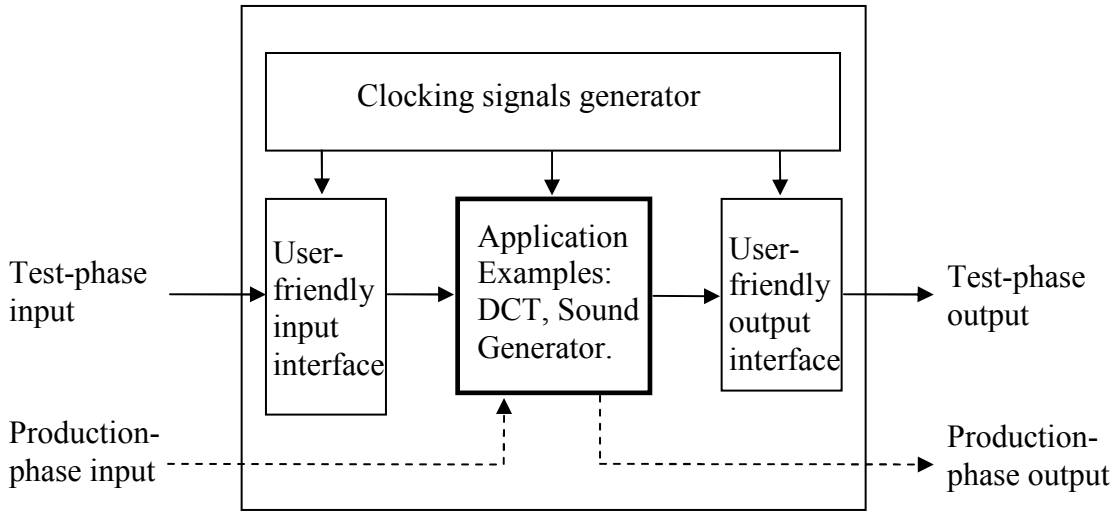

Figure 7: A Laboratory Framework

4. Conclusion

In this paper we describe a number of design considerations for efficient resource allocation for FPGA demo-board based digital laboratories. Students may use these design considerations as a recipe for handling various design requirements. Also, we present several laboratories for students to practice these methodologies. Some students defined their own projects using the framework described in Subsection 3.4. For example, one project extended the assignment to include a sound generator so that the design was able to support input of musical scales into a memory block and replay of stored music.

Most importantly, students learned how to apply micro-architectural modeling to hardware implementation of computer algorithms as well as efficient event and clock scheduling for seamless system integration.

References

1. M. F. Aburdene, J. Zheng, and R. J. Kozick, Computation of Discrete Cosine Transform Using Clenshaw's Recurrence Formula, IEEE Signal Processing Letters, Vol. 2, No. 8,pp. 155-156, August 1995.

2. P. J. Ashenden, The Designer's Guide to VHDL, 2002, Morgan Kaufmann Publishers, San Francisco, California 94104.

3. R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli and A. R. Wang, MIS: A Multiple-Level Interactive Logic Optimization System, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Pages 1062-1081, CAD-6, 6, November 1987.

4. K. Keutzer, DAGON: Technology Binding and Local Optimization by DAG Matching, Proceedings of the 24th Design Automation Conference, 1987.

5. E. J. McCluskey, Minimization of Boolean Functions, The Bell System Technical Journal, Vol. XXXV, No. 6, pp. 1417-1444, November 1956.

6. W. V. Quine, A Way to Simplify Truth Functions, The American Mathematical Monthly 62, pp. 627-631, November 1955.

7. C. J. Tseng and D. P. Siewiorek, "Automated Synthesis of Data Paths in Digital Systems," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. CAD-5, No. 3, pp. 379-395, July 1986.

8. C. J. Tseng, Behavioral Transformation for Pipeline Synthesis, Proceedings of the Custom Integrated Circuit Conference, June 1991.

9. C. J. Tseng, "Clocking Schedule and Writing VHDL Programs for Synthesis," Proceedings of The 2004 ASEE Annual Conference & Exposition, Session 1532, June 2004.

10. C. J. Tseng and M. F. Aburdene, "Digital Signal Processing and Digital System Design Using Discrete Cosine Transform," Proceedings of The 2005 IEEE International Conference on Acoustics, Speech, and Signal Processing, Philadelphia, March 2005.

11. J. F. Wakerly, Digital Design Principles and Practices, Prentice Hall.

12. Xess Corporation, XSA Board V1.1, V1.2 User Manual, Apex, North Carolina 27502, 2002.

13. Xess Corporation, Introduction to WebPACK 5.1 for FPGAs, Apex, North Carolina 27502, 2002.

14. Xilinx, Xilinx Synthesis Technology (XST) User Guide, California, 2002.

15. Xilinx, Spartan-3™ Development Board, California, 2004.

16. Xilinx, ML-310 Virtex-II Pro™ Development Platform, California, 2004.

Biography

CHIA-JENG TSENG is an Assistant Professor in the Department of Electrical Engineering at Bucknell University. His current research interests focus on speech processing and digital system design methodologies.