International Journal of Information and Communication Technology Education
Volume 17 • Issue 2 • April-June 2021

Why Are Hardware Description Languages Important for Hardware Design Courses?

Raghad Obeidat, Yarmouk University, Jordan
Hussein Alzoubi, Yarmouk University, Jordan
https://orcid.org/0000-0002-4469-5786

ABSTRACT

Curricula in computer engineering, computer science, and other related fields include several courses about hardware design. Examples of these courses are digital logic design, computer architecture, microprocessors, computer interfacing, hardware design, embedded systems, switching theorem, and others. In order for the students to realize the concepts taught in such courses, practical track should be reinforced along with the theoretical track. Many universities offer to their students labs in which they can practice hardware design. However, students need more than that: they need tools that enable them to design, model, simulate, synthesize, and implement hardware designs. Although high-level programming languages like Java and C++ could be an option, it might be a tedious task to use them for this mission. Fortunately, hardware-description languages (HDLs) have been specifically devised for this purpose. This paper shows some of the great features of HDLs and compare using them with using C++ for illustrating digital concepts through salient examples.

KEYWORDS

Decoder, Digital Design, Hardware Design, Hardware-Description Language (HDL), Memory, Multiplexer, VHDL

INTRODUCTION

Digital circuits are important components in the modern civilization. Embedded systems can be found in every aspect of our today's life. Therefore, it is not uncommon to find many courses that are focused on the hardware design in the curricula of majors like computer engineering and computer science. Digital logic design is an example of such courses. Digital logic design is usually an introductory undergraduate course where students study in their first or second years. Digital devices represent totally new subjects for freshmen and sophomores. After finishing the course, some students cannot differentiate between the devices they learned about in that course. Most universities follow digital logic design with a laboratory; students can design and connect their own digital circuits using well-known integrated circuits (ICs) like decoders, multiplexers, adders, priority encoder, etc.

While learning about the hardware aspects of computer systems can be an easy task for some students, it can be challenging for many others. Students are required to develop their knowledge by building complex circuits in different technological aspects. To this end, hardware-description

DOI: 10.4018/IJICTE.2021040101

languages (HDLs) represent a suitable and very helpful tool in the digital field. Abundant numbers of textbooks are available nowadays for both students and instructors. The main purpose of learning and teaching HDLs is to deepen understanding of hardware components used in hardware design.

In this paper, we emphasize the importance of hardware-description languages from a different perspective, where we enrich our discussion using three common examples of digital devices, namely the decoder, multiplexer, and random-access memory (RAM). We first use these three examples to illustrate how ordinary high-level programming languages can be used to model such logical devices. The deficiencies of these ordinary high-level programming languages will be obvious when compared to hardware-description languages that were specifically created to model digital circuits. Moreover, different commercial software packages are available for use by students. These packages have many attractive features that enable students get profound understanding of digital hardware design, open their minds for creativity, and facilitate implementing practical useful projects complex enough to be manufactured as commercial products.

This paper is organized as follows: next make a quick literature review on the subject. Then, we talk about the three examples that we will use in our discussion in this paper, the decoder, multiplexer, and memory. Next, we provide a brief overview of Very-High-Speed Integrated Circuits Hardware Description Language (VHDL). The next section provides a comparison between the use of general-purpose high-level languages and hardware description languages for the purpose of hardware design using our three examples. A discussion of the rich features of the hardware description languages is provided next. Finally, we conclude our paper.

LITERATURE REVIEW

The importance of incorporating hardware description languages in teaching the students of computer science and engineering the course of digital logic design and similar courses has always been emphasised by teachers and educators of the filed.

The state-of-the-art reveals several works addressing the subject in depth. For example, (Huang et al., 1997), talk about a pilot VHDL teaching resources that have been suggested to cope with the availability of VHDL tools and class educational resources that agree with the computer architecture course syllabus. In (Etxebarria et al., 2001), authors built a workbench board integrating VHDL that aids in reinforcing the theoretical knowledge of students on digital design, practise design and simulation, in addition to observe the actual behaviour of programmable logic devices (PLDs).

In (Amaral et al., 2005), the authors discuss how the state-of-the art digital logic design tools can be very useful for the students of computer science. In (Boluda et al., 2006), the authors showed how to improve active in-class engagement of students in addition to leveraging self-learning of electronic systems design, in general, and programmable logic devices (PLDs) in specific. The paper included pictures of the ALTERA UP1 and MEPUA1 boards through power point lectures available on the Internet.

In (Abidin et al., 2010), authors describe how the VHDL can be incorporated into the course of digital logic to reinforce the student's theoretical concepts. This should target undergraduate students majoring in electrical engineering in general and computer, electronics, communication, as well as instrumentation, in particular. The example given in the paper is about the design of a simple digital circuit consisting of two AND gates connected to an OR gate. The VHDL code was simulated using Xilinx ISE Design simulator and synthesizer with the obtained simulation results.

EXAMPLES

Digital systems are constructed basically using digital circuits. Digital devices like decoders, multiplexers, counters … etc. are used to build such circuits. In this section, we take three common examples: decoder, multiplexer, and memory (RAM) and illustrate them in detail.

Table 1. The active-low 2-to-4 decoder truth table (Mano, 2007)

E	I_1	I_0	O_0	O_1	O_2	O_3
1	×	×	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Decoder

The decoder is a basic component used in many digital circuits. It is a combinational circuit that decodes its inputs to a unique output. The decoder has *n* input lines and 2*n* output lines. That is, *n*-bit input has 2^n different output combinations. Therefore, each combination on input lines produces a unique combination of the 2^n possible outputs. The 2×4 decoder is the smallest practical unit. This decoder typically uses an enable input signal, which is usually active low that allows decoding inputs to an output. The 2×4 decoder has two inputs (I1 and I0) and four outputs (O_3, O_2, O_1, and O_0. With active-low enable, the outputs are: $O_3 = \overline{\bar{E} \cdot I_1 \cdot I_0}$, $O_2 = \overline{\bar{E} \cdot I_1 \cdot \bar{I}_0}$, $O_1 = \overline{\bar{E} \cdot \bar{I}_1 \cdot I_0}$, and $O_0 = \overline{\bar{E} \cdot \bar{I}_1 \cdot \bar{I}_0}$. Table 1 shows how the decoder works (Mano, 2007).

Multiplexer

The multiplexer is a selective device that has 2^n inputs and one output. It has *n* selection lines which select one input to be passed to the output of the multiplexer. A 2-to-1 multiplexer is the smallest practical multiplexer, which has two inputs (I1 and I0), one selection line (s), and one output (O), where $O = \bar{s} \cdot I_0 + s \cdot I_1$ (Mano, 2007).

Memory

Memory is an essential component in any computer system. It is used to save digital data. Memory is simply a set of registers, which are small digital devices used to store digital data. The Register is a sequential circuit constructed with a number of flip-flops. D-type edge triggered flip-flop is usually used in memory since it passes the input to the output when the rising (or falling) edge of a clock occurs.

A set of flip-flops in a certain register stores data in binary representation; a stream of zeros and ones. The two fundamental operations performed on memory are memory read and memory write. Reading data from a memory location means activating a desired register, passing the data stored on it to the output. On the other hand, writing data to a memory location means passing data from the input and activating the desired register to store the data in. Thus, memory is externally controlled via two control signals: memory read control signal (MEMR) enables reading from a memory location, and memory write signal (MEMW) enables writing to a memory location (Gaonkar, 2002). Moreover, memory has a special specification "size" which represents its capacity. The size of a memory uses the notation $2^m \times n$ where *m* represents the number of address lines and *n* the number of data lines. 2^m represents the total number of locations in that memory (Gaonkar, 2002). For example, a memory with size of 256 × 8 has 8 address lines and 8 data lines (Gaonkar, 2002).

A fabricated memory chip has a certain number of pins depending on its size. These pins include the pins of address lines, the pins of data lines, and the pins of control signals: clock, MEMR, MEMW, and possibly others (Gaonkar, 2002). Address lines on address pins are directed to an internal decoder

that decodes the input bits and generates an enable control signal to activate the desired register, passing data in or out that register (Gaonkar, 2002).

VHDL

Students can utilize Hardware Description Languages (HDLs) in the hardware design courses. HDLs complement working with ICs and wires in the lab. The Hardware description language is a programming language that is specifically created to work with digital circuits, which can be modeled and simulated. Various software packages make it possible to monitor signals the same way as using oscilloscopes. VHDL is an example HDL. "V" is for VHSIC, which is Very High Speed Integrated Circuit (Thomas and Moorby, 1995). VHDL is widely used and very common academically.

VHDL provides various means for the programmer who can describe the design of digital systems in two levels of abstraction. The first level is the early conceptual stage with the system's behavioral construction, and the second level is the later stage with the structural construction. VHDL provides Computer-Aided-Design (CAD) tools to aid both levels in building a digital system. The first stage is used for simulating the behavior of all components in a certain design, the other stage is to collect these components and describe the connections among them. Most designs to be simulated mix both levels together. Some simple designs may not need to mix the levels; behavioral level will be fair enough (Thomas and Moorby, 1995).

Digital circuits constructed with VHDL can be implemented using reconfigurable integrated circuits, like the Field Programmable Gate Array (FPGA). Field programmable means that it can be programmed in field. It also can be reprogrammed many times by the programmer, different from other Integrated Circuits (ICs) that are configured by manufacturers. A VHDL program that has no errors on simulation is converted to a binary file to be the new configuration for an FPGA (Thomas and Moorby, 1995).

The most widely known FPGA vendors are Altera and Xilinx, both develop FPGAs to be faster and easier to use. They build special kits for users. Kits include other devices that facilitate the use of FPGAs. For example, some input/output ports, memory, displays, etc. Altera offers different FPGAs with different specifications. These specifications are designed for different usage requirements. Altera products include Cyclone series, Arria series, and Stratix series. The Cyclone series are designed to serve low power and low cost designs. Arria series are used for low power and intermediate cost but it is used for more serial data transfer, it has more transceivers, almost 10G which enables the user to maximize system bandwidth. Stratix series are named also high-end FPGAs that are high-density and high-performance with enabled maximum system bandwidth (FPGA Altera). Xilinx products are different but serve the same purposes. Xilinx products include Artix-7 family, which has the lowest cost and lowest power, Kintex-7 family which has the industry's suitable price, and Virtex-7 family, which has the specification of higher system performance with two a million logic-cells FPGA. In addition to these, EasyPath-7 family has identical functionality and timing to Virtex-7 but with 35% cost reduction. Moreover, Virtex-6, Spartan-6, and EasyPath-6 are common with the specifications of lower power, lower cost and high performance (FPGA Xilinx).

HLL vs. HDL

In this section, we provide simple programs to simulate the three devices: decoder, multiplexer, and memory. First, we show how students can use general-purpose high-level programming languages to simulate these digital devices. Specifically, we provide simple programs written in C++. We then provide the VHDL substitute that students can use throughout their course of study.

The C++ programs shown below use console applications for simplicity. Figure 1 (a) shows the C++ code to simulate the 2-to-4 decoder. A sample run of the program is provided in Figure 1 (b).

Figure 2 (a) shows the C++ code to simulate the 2-to-1 Multiplexer. A sample run of the program is provided in Figure 2 (b).

Figure 1. C++ Decoder simulation (Console Application)

```
#include "stdafx.h"
#include <iostream>

using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    int i1,i0,d0,d1,d2,d3;
    cout << "\n\n enter the input lines i1 then i0 "<< endl;
    cin >> i1 >> i0;

    if (i1==0)
        {
            if (i0==0)
            {
                d0=1; d1=0; d2=0; d3=0;
            }
            else
            {
                d0=0; d1=1; d2=0; d3=0;
            }
        }
    else
        {
            if (i0==0)
            {
                d0=0; d1=0; d2=1; d3=0;
            }
            else
            {
                d0=0; d1=0; d2=0; d3=1;
            }

        }
    cout << " the outputs are: (arranged from d3 in left to d0 in right): " << d3 << d2 << d1 << d0 << endl;
    return 0;
}
```

(a) C++ code to simulate the 2-to-4 decoder

```
 enter the input lines i1 then i0
0
1
 the outputs are: (arranged from d3 in left to d0 in right): 0010
```

(b) A sample run of the 2-to-4 decoder program

The C++ code for implementing memory is shown in Figure 3, where the two operations read and write are considered.

The C++ sample programs above for implementing decoders, multiplexers, and memories illustrate that high-level programming languages can be used to simulate digital devices. The discussion given below emphasizes that using a specialized hardware description language is a much better alternative.

VHDL provides to students simple, yet efficient programming environment. In VHDL, all combinations can be shown in a single waveform. Timing diagrams illustrate the behavior of the programmed element, showing the results according to the changes on inputs. Students can set all inputs combinations and watch the result after running for once. Also, it lets student to observe some timing elements such as cumulative delays from gates for large designs. The small timing periods help in observing such changes.

Figure 2. The 2×1 Multiplexer Simulation

```
#include "stdafx.h"
#include <iostream>

using namespace std;


int _tmain(int argc, _TCHAR* argv[])
{
    int i0,i1,s,o;
    cout << "enter the selection line"<< endl;
    cin >> s;
    cout << "enter i0 and i1" << endl;
    cin >> i0 >> i1;
    if (s==0)
        {
        o=i0;
        cout << " the output o is equal to i0=" << o;
        }
    else
        {
        o=i1;
        cout << " the output o is equal to i1= "<< o;
        }

    return 0;
}
```

(a) C++ code to simulate the 2-to-4 decoder

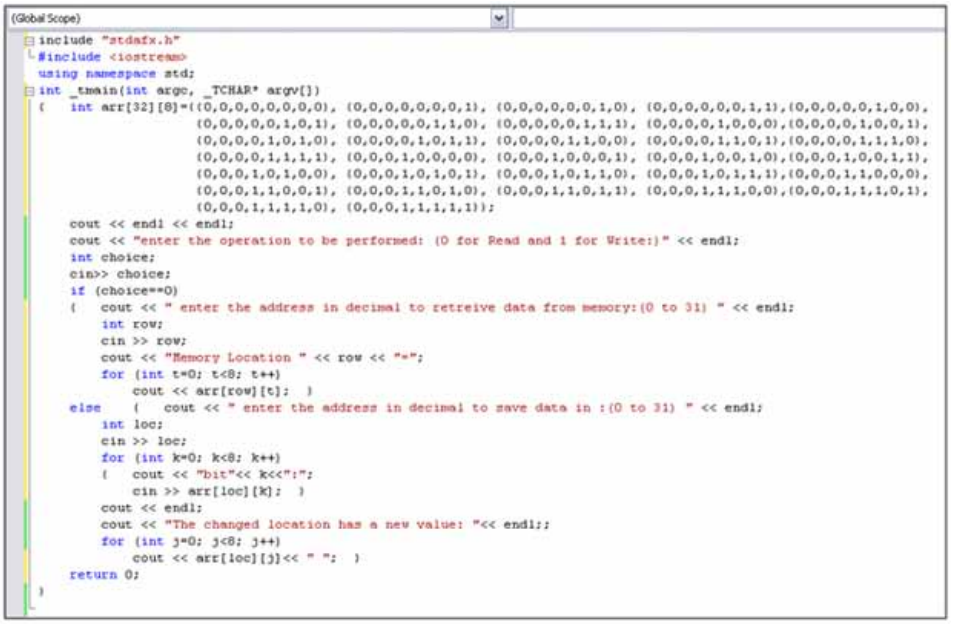
```
enter the selection line
0
enter i0 and i1
1
0
 the output o is equal to i0=1_
```

(b) A sample run of the 2-to-4 decoder program

Figure 4 (a) shows the VHDL code of the 2-to-4 decoder and Figure 4 (b) shows the simulation in a waveform type. In Figure 4 (b), the variables used in the program are listed on left. The variable "e" implements the enable for the decoder. "i0" and "i1" are the inputs of the decoder; they select the output line to be activated. The grouped output "d" represents the four output lines from the decoder "d3", "d2", "d1" and "d0", consecutively. When input "e" equals to logic 0, the decoder is disabled and the four outputs are equal to logic 0; this is shown in the simulation during the time interval from 0 ns to 400 ns. The period between 400 ns and 800 ns the decoder is enabled and outputs are changed due to changes on inputs. Initially, both "i0" and "i1" are logic 0, therefore output "d0" is activated. Next, "i0" is changed to logic 1 and as a result the decoder's outputs are changed accordingly, that is "d0" becomes 0 and "d1" goes to 1. After 800 ns, the decoder is deactivated, which disables the output again.

Figure 5 shows the VHDL implementation and simulation of the 2-to-1 multiplexer. The implemented multiplexer has an enable line "e". It also has one selection line "s" and two input lines "i0" and "i1". These three inputs will control the output "d", which will have the value of input "i0" if s = 0, and the value of input "i1" if s = 1. As shown in Figure 5 (b), during the first 400 ns the multiplexer is disabled. At 400 ns, it is enabled and the output "d" gets a value. At time interval from

Figure 3. C++ Memory Implementation

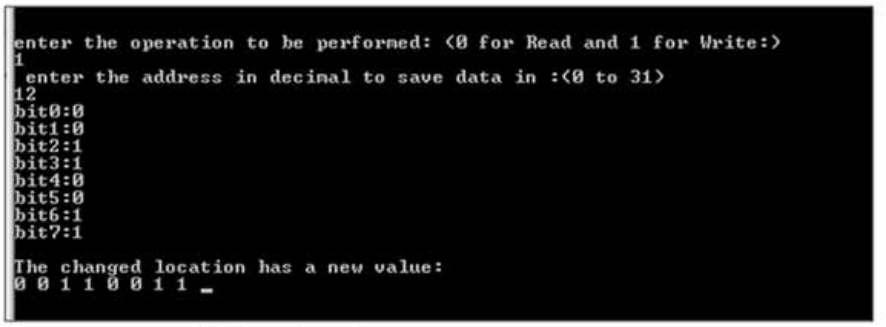


```
(Global Scope)
#include "stdafx.h"
#include <iostream>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{   int arr[32][8]={{0,0,0,0,0,0,0,0}, {0,0,0,0,0,0,0,1}, {0,0,0,0,0,0,1,0}, {0,0,0,0,0,0,1,1},{0,0,0,0,0,1,0,0},
                    {0,0,0,0,0,1,0,1}, {0,0,0,0,0,1,1,0}, {0,0,0,0,0,1,1,1}, {0,0,0,0,1,0,0,0},{0,0,0,0,1,0,0,1},
                    {0,0,0,0,1,0,1,0}, {0,0,0,0,1,0,1,1}, {0,0,0,0,1,1,0,0}, {0,0,0,0,1,1,0,1},{0,0,0,0,1,1,1,0},
                    {0,0,0,0,1,1,1,1}, {0,0,0,1,0,0,0,0}, {0,0,0,1,0,0,0,1}, {0,0,0,1,0,0,1,0},{0,0,0,1,0,0,1,1},
                    {0,0,0,1,0,1,0,0}, {0,0,0,1,0,1,0,1}, {0,0,0,1,0,1,1,0}, {0,0,0,1,0,1,1,1},{0,0,0,1,1,0,0,0},
                    {0,0,0,1,1,0,0,1}, {0,0,0,1,1,0,1,0}, {0,0,0,1,1,0,1,1}, {0,0,0,1,1,1,0,0},{0,0,0,1,1,1,0,1},
                    {0,0,0,1,1,1,1,0}, {0,0,0,1,1,1,1,1}};
    cout << endl << endl;
    cout << "enter the operation to be performed: (0 for Read and 1 for Write:)" << endl;
    int choice;
    cin>> choice;
    if (choice==0)
    {   cout << " enter the address in decimal to retreive data from memory:(0 to 31) " << endl;
        int row;
        cin >> row;
        cout << "Memory Location " << row << "=";
        for (int t=0; t<8; t++)
            cout << arr[row][t];  }
    else    {   cout << " enter the address in decimal to save data in :(0 to 31) " << endl;
        int loc;
        cin >> loc;
        for (int k=0; k<8; k++)
        {   cout << "bit"<< k<<":";
            cin >> arr[loc][k];  }
        cout << endl;
        cout << "The changed location has a new value: "<< endl;;
        for (int j=0; j<8; j++)
            cout << arr[loc][j]<< " ";  }
    return 0;
}
```

(a) C++ implementation of a 32×8 memory segment

```
enter the operation to be performed: (0 for Read and 1 for Write:)
0
 enter the address in decimal to retreive data from memory:(0 to 31)
25
Memory Location 25=00011001
```

(b) A sample run for reading from memory operation



```
enter the operation to be performed: (0 for Read and 1 for Write:)
1
 enter the address in decimal to save data in :(0 to 31)
12
bit0:0
bit1:0
bit2:1
bit3:1
bit4:0
bit5:0
bit6:1
bit7:1

The changed location has a new value:
0 0 1 1 0 0 1 1 _
```

(c) A sample run for writing into memory operation

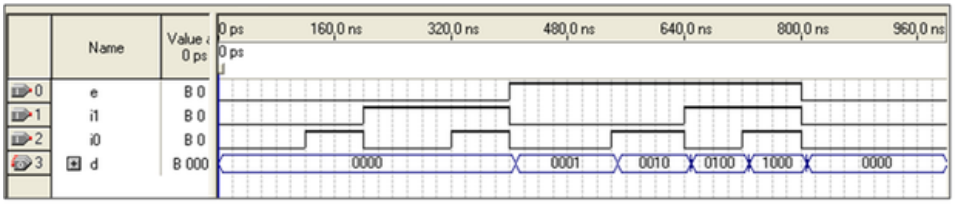
400 ns to 650 ns, the selection line "s" has logic 0 value, which means passing the value of input "i0" to "d". During 650 ns to 900 ns time interval, "s" is inverted and as a result "i1" is passed to the output.

A simple implementation of memory of size 256 bytes with eight address lines is shown in Figure 6 (a) (Short, 2009). Figure 6 (b) shows the waveforms obtained from the simulation. The code of Figure 6 (a) defines several ports that will be used in dealing with memory. These include the address lines "address", the data lines "datain" and "dataout", the control signal "we", which is an abbreviation to (write enable), and the "clock" input to synchronize events. Figure 6 (b) shows

Figure 4. VHDL Decoder Simulation

```
4
5   entity decoder is
6   port ( i0,i1,e: in bit;
7            d0,d1,d2,d3: out bit);
8   end decoder;
9
10  architecture behave of decoder is
11  begin
12    d0 <= (e and (not(i1) and not(i0)));
13    d1 <= (e and (not(i1) and i0));
14    d2 <= (e and (i1 and not(i0)));
15    d3 <= (e and (i1 and i0));
16
17    end behave;
18
```

(a) VHDL 2-to-4 Decoder code

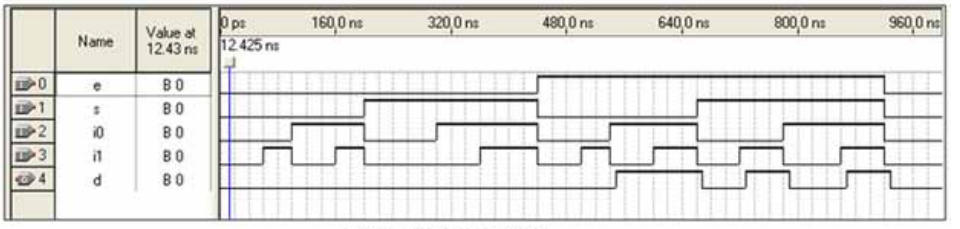


(b) Simulation using waveforms

Figure 5. VHDL Multiplexer Simulation

```
3   entity mux is
4   port(   i1,i0 : in bit;
5           e,s : in bit;
6           d : out bit);
7   end mux;
8
9
10  architecture multiplexer of mux is
11  begin
12    d <= e and ((not(s) and i0) or (s and i1));
13    end multiplexer;
```

(a)VHDL 2-to-1 Multiplexer code

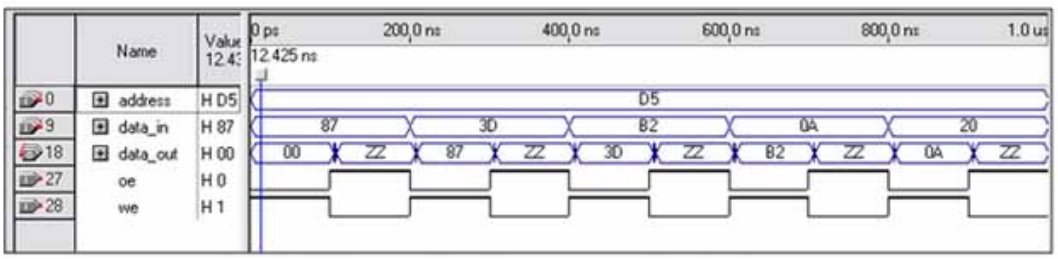


(b)Simulation waveform

Figure 6. VHDL Memory Simulation

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity memory is
port (  oe, we : in bit;
        address : in std_logic_vector (7 downto 0);
        data_in : in std_logic_vector (7 downto 0);
        data_out : out std_logic_vector (7 downto 0));
end memory;

architecture behavioral of memory is
    type mem_array is array (0 to 2**(address'length)-1) of std_logic_vector (7 downto 0);
    signal mem_s : mem_array;

begin
        write: process (we, data_in, address)
        begin
            if we='0' then
                mem_s(to_integer(unsigned(address))) <= data_in;
            end if;
        end process;

        read: process (oe, address, mem_s)
        begin
            if oe='0' then
                data_out <= mem_s(to_integer(unsigned(address)));
            else
                data_out <=(others => 'Z');
            end if;
        end process;
end behavioral;
```

(a) VHDL Memory Code [6]



(b) Simulation waveform for memory read and write

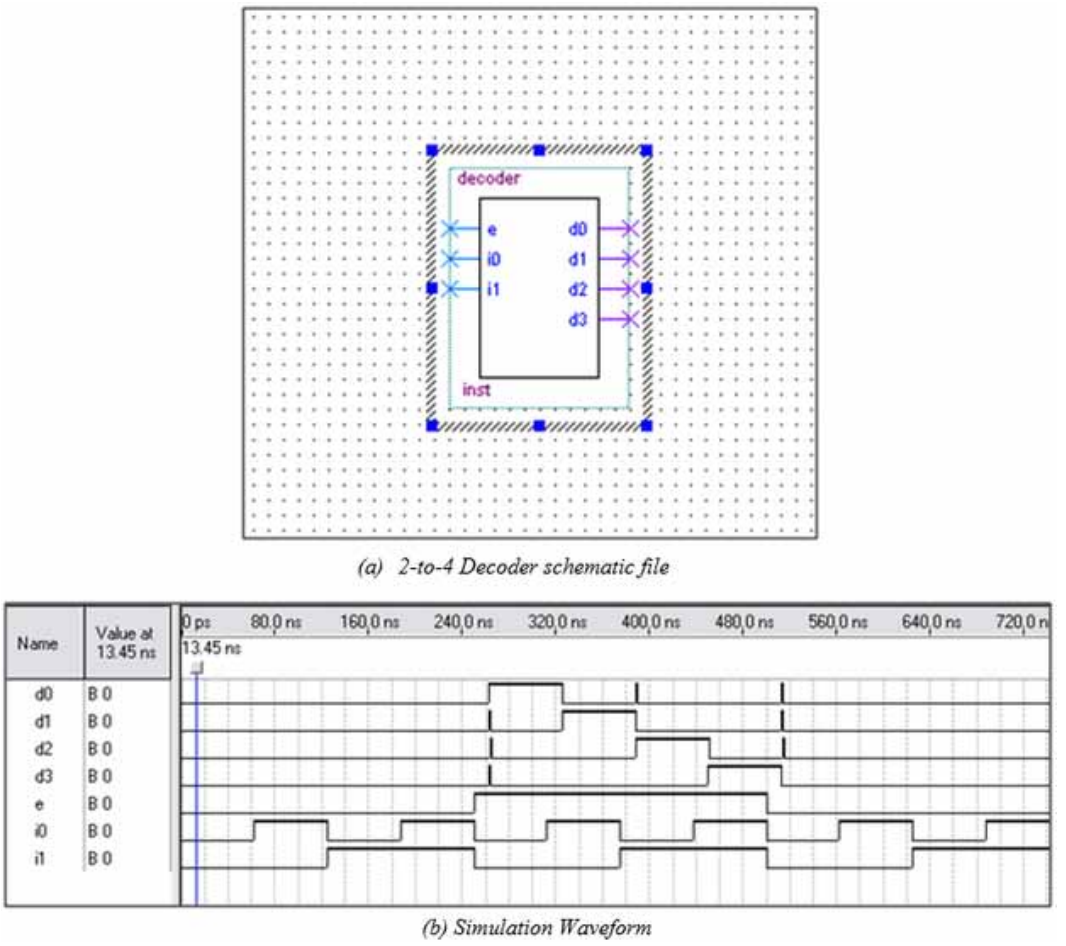
the control signals "we" and "oe", when one of them is the inverse of the other signal which means that when a read operation is activated, the write operation is deactivated. During the time interval from 0ns to 200ns, two operations are considered memory read and memory write. From time 0 to approximately 100 ns, a read operation is activated using the "oe" control signal, that is reading from memory location addressed by the address on the address lines (D5HEX) and data stored is reflected on "dataout" output port. After that time, the process of writing into a memory location is activated and the data (87HEX) on "datain" is stored on the same memory location that is read before. Figure 6 (b) also illustrates consecutive read and write operations using the same memory location, time period from 300 to 400 ns the data (3DHEX) is stored in the memory location and after that, at 400 ns the same data is read and appeared on dataout port.

Students can also implement digital devices schematically. Then, they can connect various components using wires through simulation. Another option is to convert the VHDL code into a

schematic diagram. This can be achieved by choosing this conversion option from menus. The result is a schematic file for the programmed digital element; students can force inputs to all input pins and test outputs using waveforms. For the studied examples, the figures below show the schematic diagrams (schematic files) for each digital element, where every file can be tested using waveforms. Students can utilize this simulation technique through understanding of the fabrication process, where any digital system is designed, tested, and then fabricated. The decoder implemented by the code of Figure 3 (a) is converted into a schematic file shown in Figure 6. In Figure 7 (a) the defined inputs "e", "i0", and "i1" are shown as input pins of a chip, while the four outputs "d0","d1", "d2", and "d3" are shown as output pins. Figure 7 (b) shows the simulation waveform. In Figure 7 (b) during the time interval 0 ns to 250 ns, the designed decoder is inactive, and the four outputs have low voltage. During the time segment from 260 ns to 320 ns, the decoder is activated with e = 1 and inputs i1i0 = 00, the output d0 goes high and the other outputs stayed low.

The Multiplexer conversion to schematic is shown in Figure 8. The programmed architecture is converted into a chip that behaves as programmed with input pins "i1", "i0", "e", and "s" and output pin "d" as shown in Figure 8 (a). Figure 8 (b) illustrates the behavior of this 2-to-1 multiplexer with enable control signal. The multiplexer is disabled when "e" input control signal is low (logic 0) and

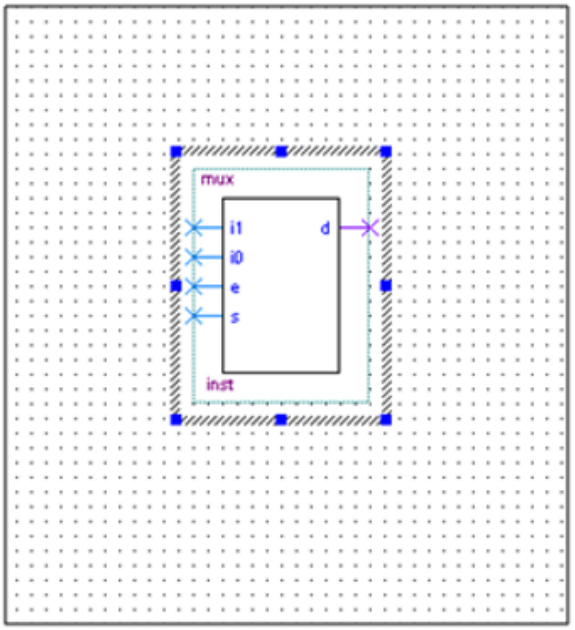
Figure 7. Decoder converted to schematic file



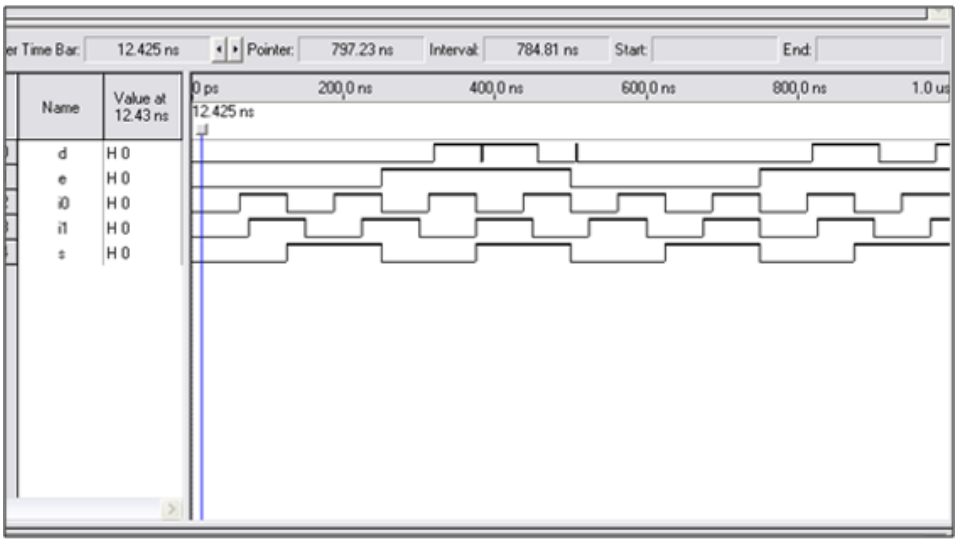
(a) 2-to-4 Decoder schematic file

(b) Simulation Waveform

Figure 8. 2-to-1 Multiplexer schematic simulation



(a) Schematic file of the 2-to-1 MUX

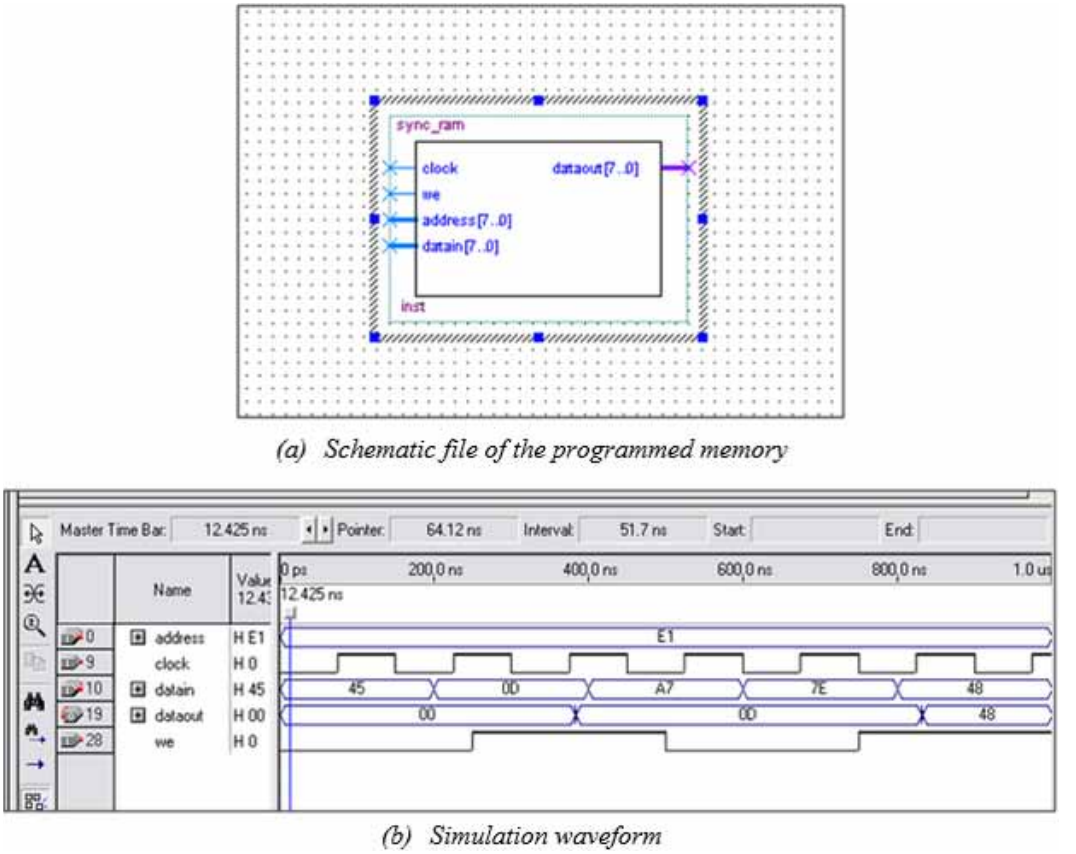


(b) Simulation waveform

enabled when high (logic 1). The time period from 250 ns to 375 ns, the multiplexer is enabled and the selection line chooses input “i0” to be passed to output “d”.

Memory conversion to schematic is shown in Figure 9, where Figure 9 (a) shows the programmed memory chip and Figure 9 (b) shows the behavior waveform. The output port “dataout” carries the data saved into memory at every positive edge transition of the clock and with logic high on the “we” control signal, which enables the memory to save data on “datain” input port. The programmed memory saved the value “0DHEX” at location “E1HEX”, while “we” is high and at a clock positive

Figure 9. Memory schematic simulation



(a) Schematic file of the programmed memory

(b) Simulation waveform

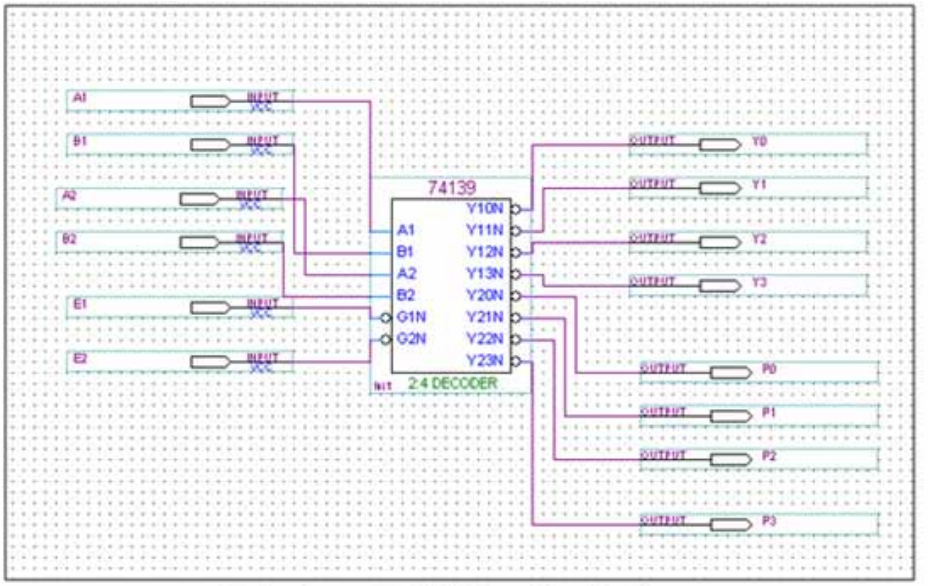
edge. The output port holds that data until a new writing happens at the same location with a new value "48 HEX", when the output port changes and holds the new data saved in that location.

Dealing With Real ICS

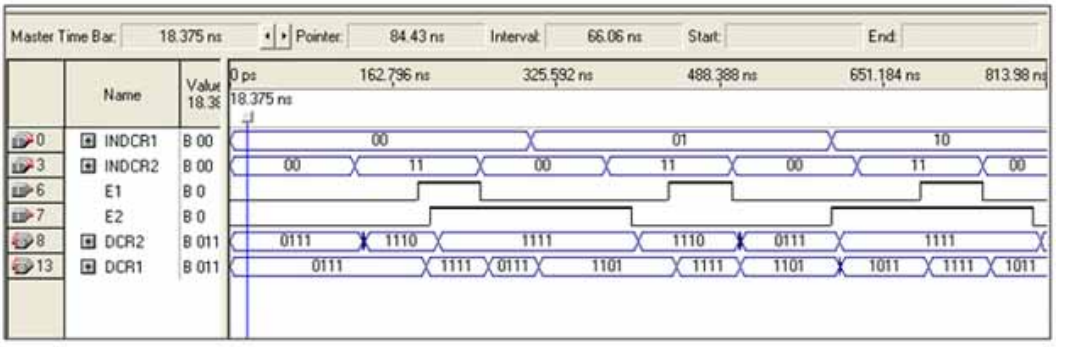
Max+Plus II is a toolkit that is installed with the Quartus II software. This tool enables students to deal with real IC chips for various digital components. Figure 10 shows how the 74LS139 IC can be used by students to make simulations and obtain the required waveforms. Figure 10 (a) illustrates that the 74LS139 chip pins are named in reference to their function. In the generated waveforms, another naming system is used to merge inputs (as well as outputs) into groups as shown in Figure 10 (b). INDCR1 of Figure 10 (b) is the input group, which represents both inputs A1 and B1. On the other hand, DCR1 is the output group that represents the four outputs of the first decoder arranged from Y0 on the left most to Y3 on the right most. This naming convention applies for the other. The enable lines E1 and E2 are the two inputs of the 74LS139 IC named G1 and G2 as shown in Figure 10 (a).

Similar to 74LS139, the simulation and waveforms for the quadruple 2-to-1 multiplexers 74LS157 chip are shown in Figure 11. A memory segment can be constructed by choosing the size of the required memory through specifying the number of locations that it will hold, and the number of bits of each location. As an example, the simulation of a 256×8 memory is shown in Figure 12 (b).

Figure 10. Using Max+Plus II to deal with 74LS139



(a) Simulation of 74LS139 Using Max+Plus II



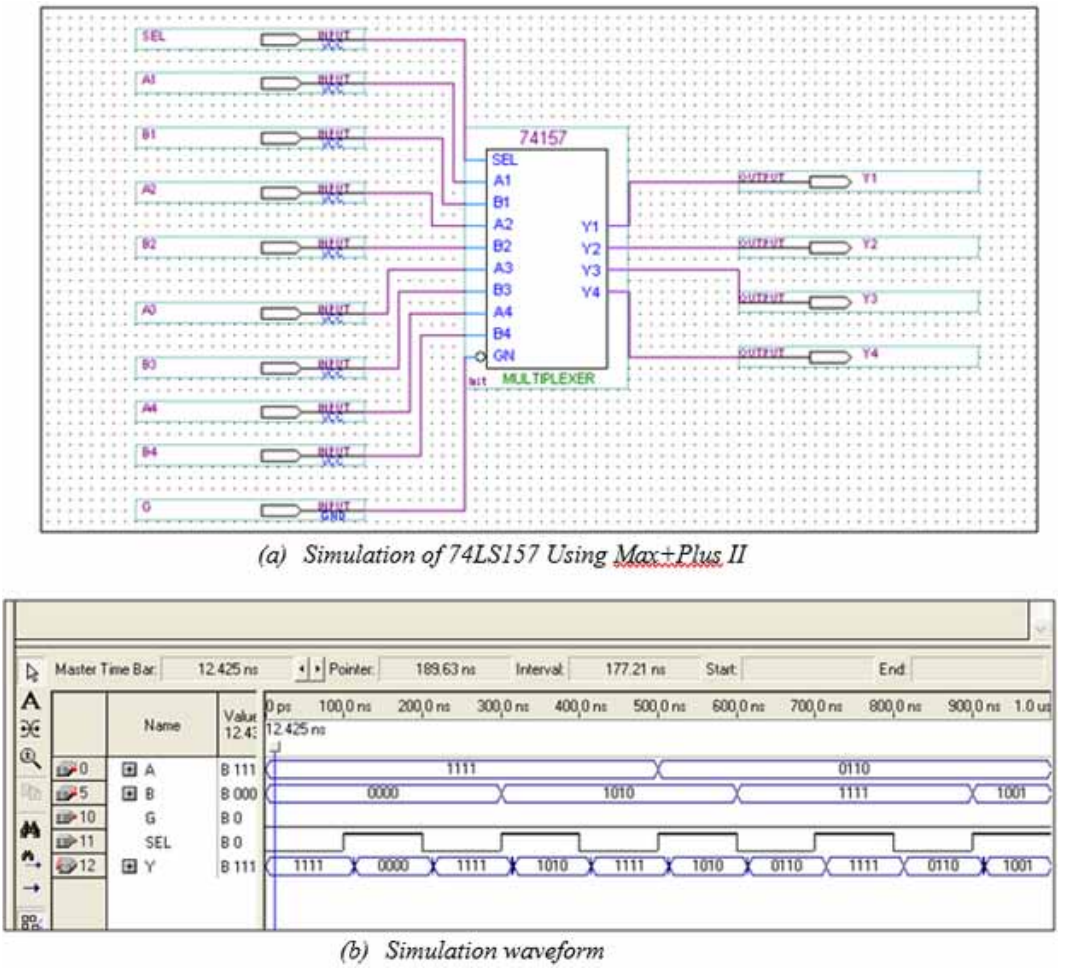
(b) Simulation Waveform

DISCUSSION

Students who have hardware-design oriented studies can use several ways to simulate digital circuits; one way is by using the general-purpose high-level programming languages. This approach may produce correct simulation results; it however might not be enough to get full understanding for the majority of students. Hardware description languages provide more specific functions and methodologies to program various digital components. Hardware description languages are complemented with software packages from known vendors like Altera and Xilinx. VHDL, in addition to those software packages, enable students to simulate and practice theoretical concepts they learned in digital hardware-design classes.

Through simulation waveforms, students can enable and disable devices and see time delays generated by digital devices. Students can follow either the top-down or the bottom-up design methodologies. Therefore, a student can split his design into different stages and test each stage separately. Finally, the different components can be connected together and tested. This provides better understanding, reduces error, and saves effort and time. Moreover, the fabrication process

Figure 11. Using Max+Plus II to deal with 74LS157



(a) Simulation of 74LS157 Using Max+Plus II

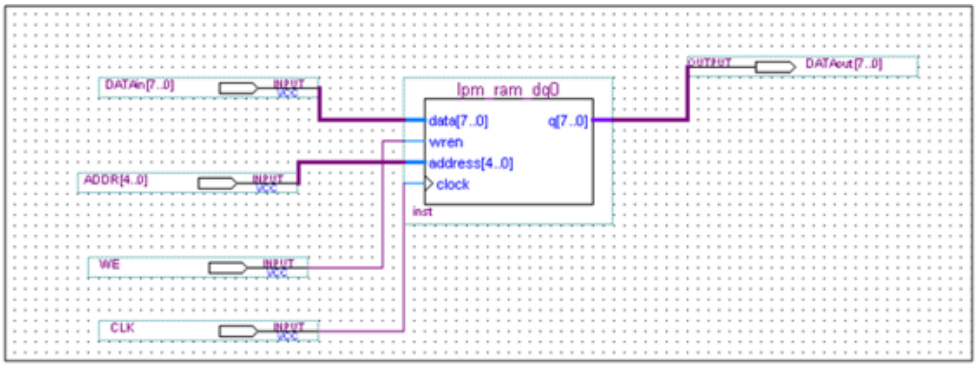
(b) Simulation waveform

will be simpler to imagine. Students can design and implement digital system and then convert to fabricated devices. If errors are encountered, they can be caught and corrected in early stages before fabrication. Software packages like MAX+PLUS II allows the designer to use and connect off-the shelf digital ICs.

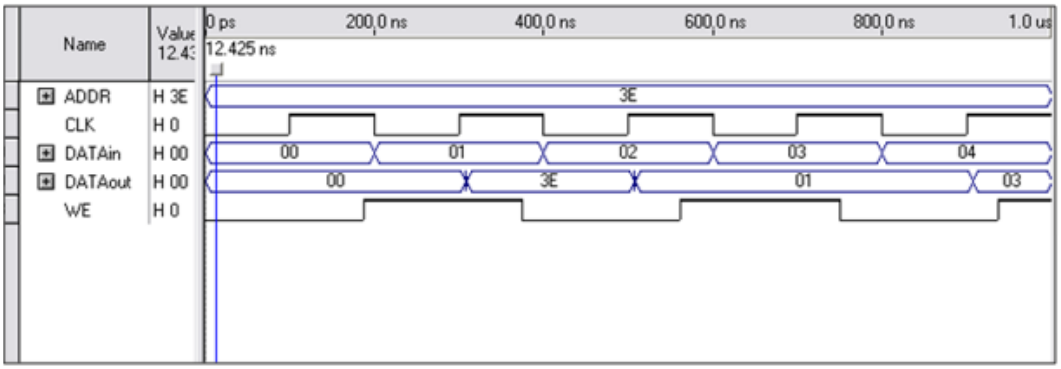
In addition to simulation, students can synthesize their hardware designs and test their implementations on easy-to-use kits like the Altera DE2 development kit. Usually, these kits are provided with other components that enable students get a deeper understanding and practice complex projects. Examples of these components include dip switches, push buttons, seven-segment displays, keypads, stepper motors, USB interfaces, and many others.

Even after completing hardware courses, many students prefer to use hardware description languages in their undergraduate graduation projects. Moreover, graduate students can utilize the extraordinary capabilities of hardware description languages in their thesis or dissertation work. Novel ideas in embedded system design, computer architecture, computer networks and security, and other areas can be easily implemented and tested using hardware description languages.

Figure 12. 256×8 Memory element simulation on Max+Plus II



(a) 256×8 Memory chip implementation on Max+Plus II



(b) Simulation waveform

CONCLUSION

In majors like computer engineering, about half the courses in the curriculum consider the hardware design. Many students find it difficult to understand the theoretical concepts taught in these courses. In this paper, we highlight the importance of hardware description languages in teaching hardware design courses. In our discussion, we have used three examples of digital devices, the decoder, multiplexer, and memory to show the rich features that hardware description languages and complementary software packages provide to students and designers. The example programs in this paper are written in VHDL and simulations are performed using Quartus II software of Altera. We have shown that students can design, build, test, implement, and fabricate digital systems step-by-step using these great features.

REFERENCES

Abidin, H. Z., Kassim, M., Othman, K. A., & Samad, M. (2010). *Incorporating VHDL in teaching combinational logic circuit.* 2010 2nd International Congress on Engineering Education (ICEED), Kuala Lumpur, Malaysia.

Amaral, J. N., Berube, P., & Mehta, P. (2005, February). Teaching Digital Design to Computing Science Students in a Single Academic Term. *IEEE Transactions on Education, VOL.*, *48*(1), 127–132. doi:10.1109/TE.2004.837048

Boluda, J. C., Peiró, M. A. M., Torres, M. Á. L., Gironés, R. G., & Palero, R. J. C. (2006, August). An Active Methodology for Teaching Electronic Systems Design. *IEEE Transactions on Education, VOL.*, *49*(3), 355–359. doi:10.1109/TE.2006.879247

Etxebarria, A., Oleagordia, I. J., & Sanchez, M. (2001). *An Educational Environment for VHDL Hardware Description Language Using The Www And Specific*. 31st Annual Frontiers in Education Conference, Reno, NV.

FPGA CPLD and ASIC from Altera. (n.d.). https://www.altera.com/

FPGA, CPLD, and EPP Solutions from Xilinx, Inc. (n.d.). https://www.xilinx.com/

Gaonkar, R. S. (2002). *Microprocessor Architecture, Programming, and Applications with the 8085* (5th ed.). Prentice Hall.

Huang, T. C., Melton, R. W., Bingham, P. R., Alford, C. O., & Ghannadian, F. (1997). The teaching of VHDL in computer architecture. *Proceedings of the 1997 IEEE International Conference on Microelectronic Systems Education (MSE '97)*.

Mano, M. M. (2007). *Digital Design (4th ed.). Pearson Prentice Hall*.

Short, K. L. (2009). *VHDL for Engineers* (1st ed.). Pearson Education Inc.

Thomas, D. E., & Moorby, P. R. (1995). The Verilog® Hardware Description Language. Kluwer Academic Publishers.

Hussein R. Al-Zoubi received his MSE and Ph.D. in Computer Engineering from the University of Alabama in Huntsville, USA in 2004 and 2007, respectively. He received his bachelor degree in Electrical Engineering from Jordan University of Science and Technology. Since 2007, he has been working with the Department of Computer Engineering, Hijjawi Faculty for Engineering Technology, Yarmouk University, Jordan. He is currently a Professor. His research interests include machine vision, pattern recognition, image processing, computer networks and their applications: wireless and wired, security, multimedia, queuing analysis, and high-speed networks.