EE25266 – ASIC/FPGA Chip Design

Electrical Engineering Department
Sharif University of Technology

Homework #0 – Spring 2017

Introduction to Computer-Aided Design Software, the Board and Simple Logic, and Control Logic

1. Introduction

The purpose of this assignment is to introduce you to the software tools and hardware that are used in the labs for this course. The main software tool is the Altera Quartus II Computer Aided Design (CAD) system. You will need to install that software on your computer before you can start this lab. The software will be distributed to you in class.

You will be learning to design hardware that will go into a kind of programmable logic chip call a Field-Programmable Gate Array, or FPGA. The FPGA chip is mounted on a board called the Altera DE2 Development and Education board, pictured below (Fig. 1). This board will be used for the first four assignments and possibly the final project of this course.

The board contains many useful features for learning about logic circuits, including simple input and output mechanisms like switches and lights, and more complicated features like audio and video devices. This assignment will use only the switches and lights that are provided on the bottom edge of the board, as illustrated below, but other assignments will utilize more advanced features. A detailed description of the board can be found on the course website at: http://ee.sharif.edu/~asic

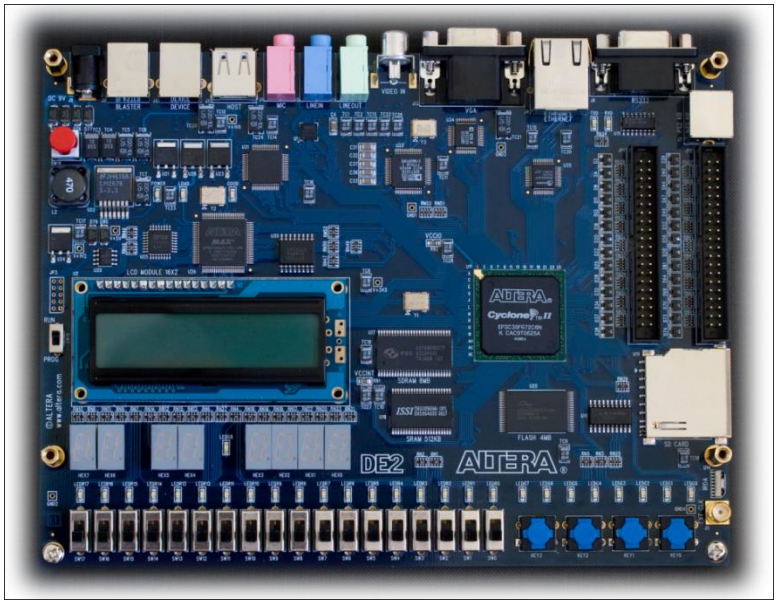


Fig. 1. The Altera DE2 Development and Education board.

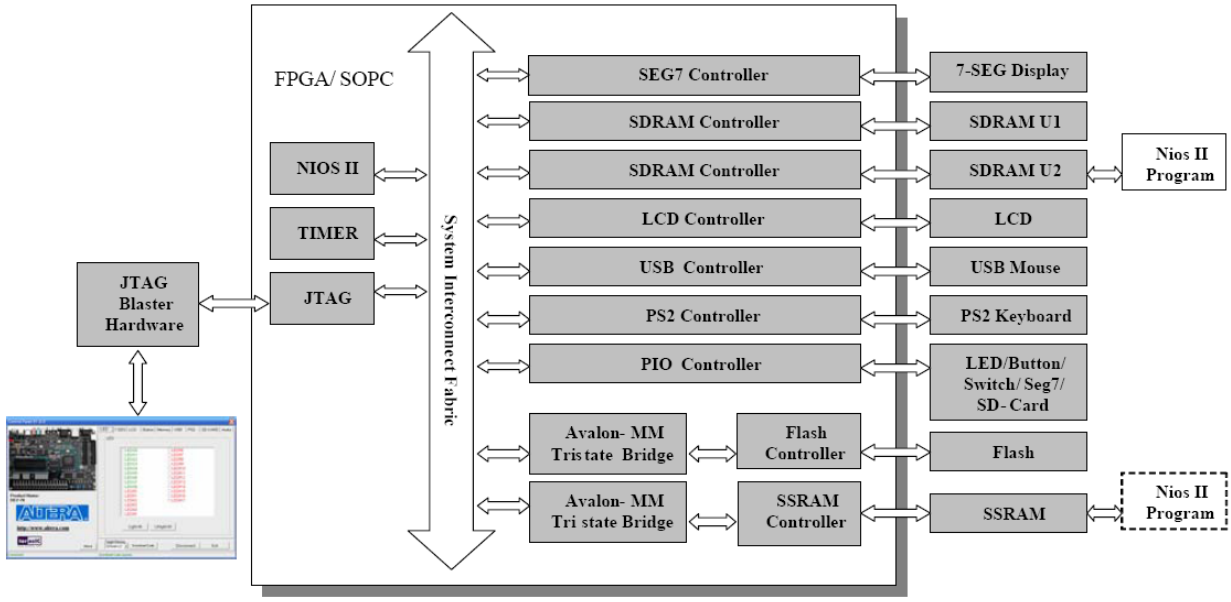


Fig. 2. The block diagram of the DE2-70 Control Panel.

Note:

The blue boxes identified by the word "DE2" are the parts that should be done on-line on the board. The rest can be done at home.

Important Notes:

Note that there are two types of board in the lab (DE2 and DE2-70). So the FPGA and the corresponding pin assignment will be different for them. Please consider the following notes in all of the assignments:

1- **Use 'DE2_pin_assignments.csv' for the pin assignment of the DE2 board and use 'DE2_70_pin_assignments.csv' for the pin assignment of the DE2-70 board.**
2- **A '*starter kit*' is provided for each computer assignment, which can be used for both of the boards without any change. Because the names of the useful pins are changed in the above '.csv' files for both of the boards. Note that always the port names of your design should be the same with the content of the corresponding .csv file, which you added to your design.**
3- **Set the part name in your design as follows:**
   - **For the DE2 board: EP2C35F672C6**
   - **For the DE2-70 board: EP2C70F896C6**

Part I:

For this part you need to do three important tutorials indicated in the following. Make sure you perform every single step specified in them as they are the foundation of all you will do in the rest of this course. Save them in three subfolders in a folder called Part I to deliver their soft copy to the TA.

Tutorial 1. Do the tutorial called *Using Quartus II CAD Software*, which is in fact the Appendix B of the book Fundamentals of Digital Logic with Verilog Design, 2nd Edition. This tutorial describes the basics of how Quartus II helps a designer describe circuits and check them for correctness. It shows two ways of creating circuits: either using schematic capture, in which you "draw" a picture of the circuit, or describing a circuit in language form. Please do the same circuit in both ways so that you understand the schematic form and the textual form describing exactly the same thing. We will use the textual form often because it is far more powerful and quicker.

Tutorial 2. Do the first part of the tutorial called *Implementing Circuits in Altera Devices* (Appendix C.1).
Tutorial 3. Do the tutorial *Physical Implementation in an FPGA* (Appendix D).

Part II:

Note:
To simplify your work for the other parts, a *starter kit* is provided. The starter kit is a ZIP archive containing a Quartus II project that you will need for each part of the lab. Unzip the archive into a working directory called Homework_0.

Fig. 3. Inside the starter kit for the Homework_0

Create a simple circuit to connect four switches to four lights on the Altera board, by extending the following Verilog code:

```
// Simple module that connects the SW switches to the LEDR lights
module Part2 (Switch 1, Switch 2, Switch 3, Switch 4, Light 1, Light 2, Light 3, Light 4);
input Switch 1, Switch 2, Switch 3, Switch 4; // toggle switches
output Light 1, Light 2, Light 3, Light 4; // lights

// Your code goes here

Endmodule
```

Fig. 4. Verilog code for Part II.

On the board the FPGA chip that your designs will be programmed into has hardwired connections between the pins of the FPGA chip and the switches and lights on the board. Note that these lights and switches are connected to circuits that allow them to generate 1s and 0s as inputs to your FPGA circuit (for the switches) and to turn on and off in response to digital 1s and 0s that are outputs from your circuit.
To use switches and lights you have to tell Quartus II which of the input/output signals in your Verilog code should be connected to which pins on the FPGA chip and which are connected to the switches or lights. The procedure for doing this is called *pin assignment* and was covered in the tutorial 3 of Part I above. Table 1 below indicates which pins (which are referred to by names such as PIN N25) of the FPGA are connected to which switches and lights on the board. There are 18 total switches and lights on the board, but the table only lists 4 of each needed for this part.

NOTE:

Please only connect the SW0…SW3 to LEDR0…LEDR3 on the Altera board .Do NOT try to connect the other switches and lights.

Table 1. Pin assignment table for lights and switches in Part II.

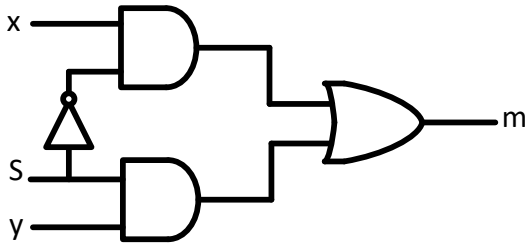
Function	Altera FPGA Cyclone II Pin on the DE2
Switch_1	PIN_N25
Switch_2	PIN_N26
Switch_3	PIN_P25
Switch_4	PIN_AE14
Light_1	PIN_AE23
Light_2	PIN_AF23
Light_3	PIN_AB21
Light_4	PIN_AC22

Do the following steps to download and test the circuit in Part II:

1. Open in Quartus II (using the command File > Open Project) the project named *Part2.qpf* in the *Part2* subdirectory to begin your work.
2. Create a Verilog module named *Part2* for the code in Fig. 4 and include it in your project. Make sure to complete the code by adding the assignment statements for the lights.
3. Use Quartus II to make the pin assignments shown in Table 1 as described in tutorial 3. Compile the project.

Part III:

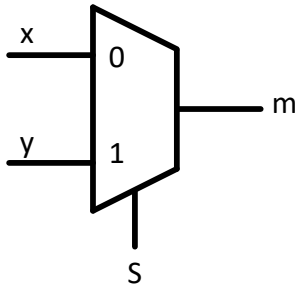
Fig. 5a shows a sum-of-products circuit that implements a 2-to-1 multiplexer (MUX) of inputs x and y select input s and output m. If s = 0 the multiplexer's output m is equal to the input x, and if s = 1 the output is equal to y. Part b of the figure gives a truth table for this multiplexer, and part c shows the schematic circuit symbol.



a) Circuit

S	m
0	x
1	y

b) Truth Table



c) Symbol

Fig. 5. A 2-to-1 Multiplexer.

The multiplexer can be described by the following Verilog statement:

```
assign m = (~s & x) | (s & y);
```

You are to design a circuit, using Verilog, which is a more complex version of a multiplexer. Rather than select between two signals, your circuit is to select between two sets of *eight* signals, as illustrated in Fig. 8a. This circuit has two eight-bit inputs, X and Y , and produces the eight-bit output M. If s = 0 then M = X, while if s = 1 then M = Y . We refer to this circuit as an eight-bit wide 2-to-1 multiplexer. It has the circuit symbol shown in Fig. 5b, in which X, Y , and M are depicted as eight-bit wires.

Do the following steps to download and test the circuit in Part III:

1. Open the project named *Part3.qpf* in the *Part3* subdirectory to begin your work.
2. Include your Verilog file for the eight-bit wide 2-to-1 multiplexer in your project. Use switch SW17 on the board as the s input, switches SW7−0 as the X input and SW15−8 as the Y input. Connect the output M to the green lights on the board, called LEDG7−0.

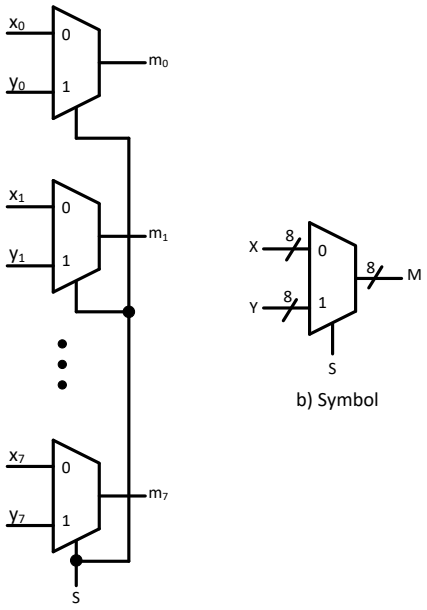


Fig. 6. An eight-bit wide 2-to-1 multiplexer.

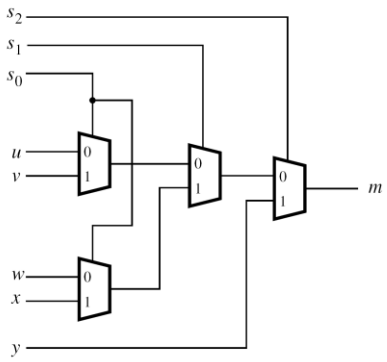
3. Include in your project the required pin assignments for the board using the pin assignment file as described above. As discussed in Parts I and II, these assignments ensure that the inputs declared in your Verilog code will use the pins on the Cyclone II FPGA that are connected to the *SW* switches and LEDRs, and the outputs of your Verilog code will use the FPGA pins connected to the LEDG lights. Compile the project.

NOTE:

Always Connect the inputs to the LEDRs in Part III and Part IV.

Part IV:

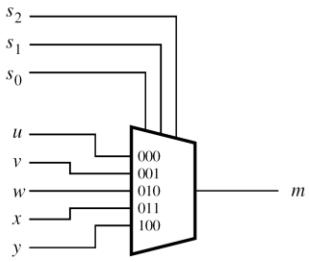
In Fig. 6 we showed a 2-to-1 multiplexer that selects between the two inputs *x* and *y*. For this part consider a circuit in which the output *m* has to be selected from *five* inputs u, v, w, x, and y. Part a of Fig. 7 shows how we can build the required 5-to-1 multiplexer by using four 2-to-1 multiplexers. The circuit uses a 3-bit select input {s2,s1,s0} and implements the truth table shown in Fig. 7b. A circuit symbol for this multiplexer is given in Fig. 7c. Recall from Fig. 6 that an eight-bit wide 2-to-1 multiplexer can be built by using eight 2-to-1 multiplexers. Fig. 8 applies this concept to define a three-bit wide 5-to-1 multiplexer. It contains three instances of the 5-to-1multiplexer circuit in Fig. 7.



a) Circuit

s_2	s_1	s_0	m
0	0	0	u
0	0	1	v
0	1	0	w
0	1	1	x
1	0	0	y
1	0	1	y
1	1	0	y
1	1	1	y

b) Truth table



c) Symbol

Fig. 7. A 5-to-1 MUX.

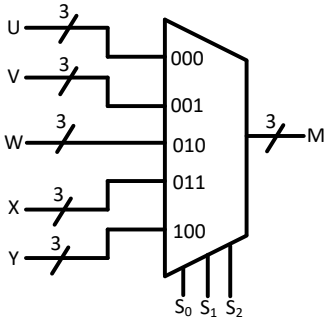


Fig. 8. A three-bit wide 5-to-1 multiplexer.

Do the following steps to download and test the circuit in Part IV:

1. Open the project named *Part4.qpf* in the *Part4* subdirectory to begin your work.
2. Create a Verilog module for the three-bit wide 5-to-1 multiplexer. Connect its select inputs to switches SW17−15, and use the remaining 15 switches on the Altera board (SW14−0) to provide the five 3-bit inputs U through Y. Connect the output M to the green lights *LEDG*2−0.
3. Include in your project the required pin assignments for the board. Compile the project.

Part V - Design of a 7-Segment Decoder Circuit

Fig. 9 shows a 7-segment character display controlled by a logic circuit. These displays are common on digital watches and various electrical devices. This circuit is called a 7-segment *decoder* and it has a three-bit input {c_2, c_1, c_0} (which you will connect to switches on the board), and 7 outputs that turn on or off the 7 different segments in the character display. The three inputs present a code that are translated by the circuit into 7 outputs to create a particular character by lighting up some of the lights on the display.

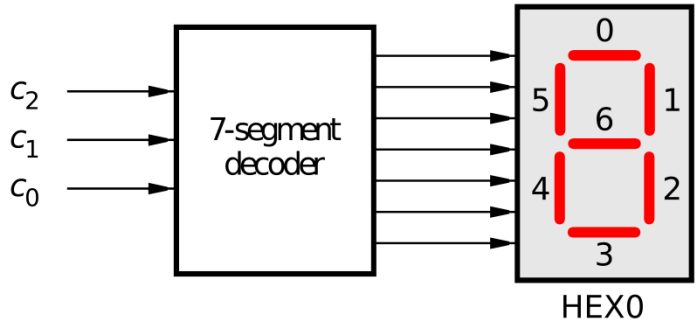


Fig. 9. A 7-segment decoder.

You are to design a circuit that is able to decode and display *the last 5 digits of your student number*. An example mapping is given in Table 2, which lists the characters that should be displayed for each value of {c_2, c_1, c_0}. To keep the design simple, you only need to decode five numbers, represented using the codes {c_2, c_1, c_0} = 000, 001, 010, 011, and 100. The fifth code should produce a blank character with all of the lights off. Since we don't care what is displayed for the remaining code values (110 and 111) you can be treat these as don't care values. The seven segments in the display are identified by the numbers 0 to 6 shown in Fig. 9. Each segment is lit up by driving it to the logic value 0 (which is a little opposite of what you might expect). You must implement a separate logic function that controls each segment in the display. Use a Karnaugh map to determine the minimal (optimal) sum-of-products expressions for each of these 7 outputs.

Create a Verilog module for your 7-segment decoder circuit. In your Verilog code, use only simple assign statements to specify each of the sum-of-products expressions generated from your Karnaugh maps. In your Verilog code assign the {c_2, c_1, c_0} inputs to switches SW2−0 on the board, and assign the outputs of the decoder to the HEX0 display on the board. The segments in this display are called HEX00, HEX01, . . ., HEX06, corresponding to Fig. 9 using the pin assignment file provided in the previous assignment.

Note that in the pin assignments file the HEX0 segments are declared as an array. To use the same names in your Verilog code, your should declare the seven-bit output port as

```
output [0:6] HEX0;
```

By using this 7-bit array, the names of the seven segments in your code will match the names that are used for these segments in the pin assignment file.

Table 2. Character codes (for the case where your student number ends in ’12345’)

$c_2c_1c_0$	Character
000	1 (replace with last digit of your student number)
001	2 (replace with 2nd last digit of your student number)
010	3 (replace with 3rd last digit of your student number)
011	4 (replace with 4th last digit of your student number)
100	5 (replace with 5th last digit of your student number)
101	‘blank’
110	‘Don’t Care’
111	‘Don’t Care’

Part VI - Selecting the Character to be Displayed

Consider the circuit shown in Fig. 10. It uses a three-bit wide 5-to-1 multiplexer to enable the selection of five characters that are displayed on a 7-segment display. Using the 7-segment decoder from Part V of Assignment, this circuit can display any of the five digits and ‘blank’. The character codes are set according to Table 2 of Assignment by using the switches SW14–0, and a specific character is selected for display by setting the switches SW17–15.

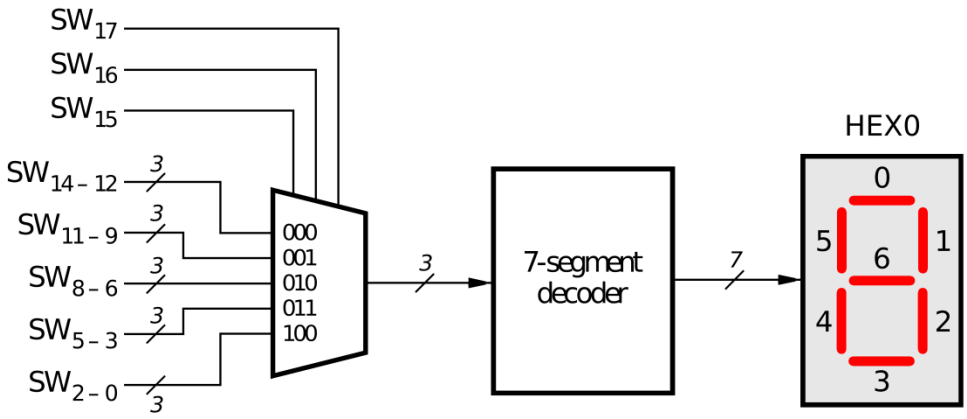


Fig. 10. A circuit that can select and display one of five characters

To build this circuit you can reuse two smaller designs already created: the three-bit 5-to-1 multiplexer from Part IV of Assignment and the 7-segment decoder from Part V of Assignment. Your task is to create a new *top-level* design that implements the circuit in Fig. 10 using the multiplexer and decoder as subcircuits. You are to design this top-level circuit twice, using two different approaches:

1. Create the top-level design using the Quartus II schematic capture tool. In this case you use Quartus II to create a schematic symbol for the multiplexer and decoder circuits, and then include these symbols in the top-level schematic, as described in Tutorial 1 of Assignment by doing the following steps:

a) The project for this part is provided in the starter kit. Open the project named *hierarchy_schematic.qpf* in the *Part6/hierarchy_schematic* subdirectory to begin your work.
b) In the Part6 folder you will find the Verilog files *mux_3bit_5to1.v* and *char_7seg.v*. Edit these files and put in the code you previously wrote to implement the three-bit 5-to-1 multiplexer and the 7-segment decoder, respectively.
c) The next step is to use Quartus II to create a *symbol file* for each of the above Verilog files. The symbol file allows the corresponding Verilog module to be used as a subcircuit in your top-level schematic. The procedure for generating a symbol file and including it in a schematic was shown in section B.5 of the Tutorial 1.
d) In Quartus II use the command **File > New** to create a new Block Diagram file and draw the circuit in Figure 10. Insert the symbols for the subcircuits created above, insert input and output ports, and draw the wiring connections shown in Figure 10.
e) In your schematic diagram you will need to create vector ports to attach to the switches and 7-segment displays. You do this in Quartus II by specifying the name of a signal as a vector of the form NAME[X..Y]. For example, in the schematic capture tool SW2−0 is designated using the input pin name SW[2..0] and HEX00−6 is denoted using an output port named HEX0[0..6].
f) Include the required pin assignments for the board switches and 7-segment display. Compile the project.
g) Simulate the compiled circuit using a timing simulation in the Quartus II Simulator.

2. Create the top-level design by writing complete Verilog code for the circuit in Fig. 10. An outline of this code, which shows how to include the multiplexer and decoder subcircuits in the top-level Verilog module, is given in Fig. 11. Complete this section by performing the following steps:
   a) The project for this part is provided in the starter kit. Open the project named *hierarchy_verilog* in the *Part6/hierarchy_verilog* subdirectory to begin your work.
   b) Use the modified codes in the previous section, i.e., *mux_3bit_5to1.v* and *char_7seg.v*.
   c) Use the File > New command to create a new Verilog file named *hierarchy_verilog.v*. Type your top-level Verilog code, following the style of code shown in Fig. 11, into this file.
   d) Include the required pin assignments for the board switches and 7-segment display. Compile the project.
   e) Simulate the compiled circuit using a timing simulation in the Quartus II Simulator.

```
module hierarchy_verilog (SW, HEX0);
input [17:0] SW; // toggle switches
output [0:6] HEX0; // 7-seg displays
wire [2:0] M;
    mux_3bit_5to1 M0 (SW[17:15], SW[14:12], SW[11:9], SW[8:6], SW[5:3], SW[2:0], M);
    char_7seg H0 (M, HEX0);
endmodule

// implements a 3-bit wide 5-to-1 multiplexer
module mux_3bit_5to1 (S, U, V, W, X, Y, M);
input [2:0] S, U, V, W, X, Y;
output [2:0] M;

. . . code not shown

endmodule
```

```
// implements a 7-segment decoder for each character
module char_7seg (C, Display);
input [2:0] C; // input code
output [0:6] Display; // output 7-seg code

. . . code not shown

endmodule
```

Fig. 11. Verilog code for the circuit in Fig. 2.

Note: Both simulations (of the schematic-based and Verilog-based projects) can be done using the same input waveforms and should produce identical outputs.

Part VII - Displaying and Rotating a Sequence

In this part you will reuse your previous designs, and extend the code from Part VI so that it uses **five** 7-segment displays, rather than just one. You will need to use five instances (copies) of the subcircuits from Part VI. The purpose of your circuit is to display a sequence on the five displays, and be able to manually rotate this sequence in a circular fashion across the displays when the switches SW17−15 are toggled. As an example, if the displayed sequence is 12345, then your circuit should produce the output patterns illustrated in Table 3. The sequence must be the last five digits of your student number.

Table 3. Manually rotating the sequence 12345 on five displays.

SW_{17} SW_{16} SW_{15}	HEX4	HEX3	HEX2	HEX1	HEX0
000	1	2	3	4	5
001	2	3	4	5	1
010	3	4	5	1	2
011	4	5	1	2	3
100	5	1	2	3	4

Perform the following steps.

1. The project for this part is provided in the starter kit. Open the project named *Part7* in the *Part7* subdirectory to begin your work.

2. Copy the Verilog files *mux_3bit_5to1.v* and *char_7seg.v* from Part VI into the project directory for Part VII. Use the Quartus II command Project > Add/Remove Files in Project to add these files to the *Part7* project.

3. Create a new Verilog file called *Part7.v* and write the code to instantiate the required subcircuits. Connect the switches SW17−15, in the same order, to the select inputs of each of the five instances of the three-bit wide 5-to-1 multiplexers. Also connect SW14−0 to each instance of the multiplexers as required to produce the patterns of characters shown in Table 3. Each multiplexer will share the same set of inputs (SW14−0), but the inputs will connect to each multiplexer in a different manner. It is up to you to arrange them properly such that you can manually “rotate” your numbers. Following this, you must connect the

outputs of the five multipexers to the 7-segment displays HEX4, HEX3, HEX2, HEX1, and HEX0 on the board.

4. Include the required pin assignments for the board switches and 7-segment displays. Compile the project.

Part VIII

Extend your design from Part VII so that is uses all eight 7-segment displays on the board. Your circuit should be able to display the last five digits of your student number on the eight displays, and manually rotate the displayed sequence when the switches SW17−15 are toggled. If the displayed sequence is 12345, then your circuit should produce the patterns shown in Table 4.

Table 4. Rotating the sequence 12345 on eight displays.

SW_{17} SW_{16} SW_{15}	HEX7	HEX6	HEX5	HEX4	HEX3	HEX2	HEX1	HEX0
000				1	2	3	4	5
001			1	2	3	4	5	
010		1	2	3	4	5		
011	1	2	3	4	5			
100	2	3	4	5				1
101	3	4	5				1	2
110	4	5				1	2	3
111	5				1	2	3	4

Perform the following steps:

1. The project for this part is provided in the starter kit. Open the project named *Part8* in the *Part8* subdirectory to begin your work.

2. Write the required Verilog code for this part of the exercise and include these Verilog modules in the Quartus II project. Connect the switches SW17−15 to the select inputs of each instance of the multiplexers in your circuit. Also connect SW14−0 to each instance of the multiplexers as required to produce the patterns of characters shown in Table 4. Connect the outputs of your multiplexers to the 7-segment displays HEX7, . . ., HEX0.

3. Include the required pin assignments for the board switches and 7-segment displays. Compile the project.

Part IX :

All FPGAs include flip-flops that are available for implementing a user's circuit. We will show how to make use of these flip-flops in the other assignment. But first we will show how storage elements can be created in an FPGA without using its dedicated flip-flops. Fig. 12 depicts a gated D latch circuit. Two styles of Verilog code that can be used to describe this circuit are given in Fig. 13. Part a of the figure specifies the latch by instantiating logic gates, and part b uses logic expressions to create the same circuit. If this latch is implemented in an FPGA that has 4-input lookup tables (LUTs), then only one lookup table is needed.

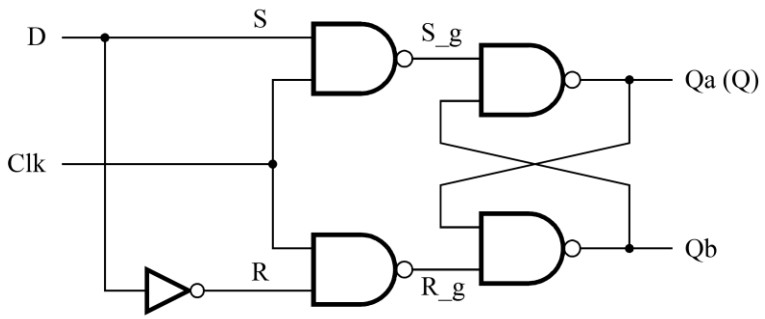


Fig. 12. A gated D latch circuit.

```
// A gated D latch
module Part9(Clk, D, Q);
  input Clk, D;
  output Q;
  wire R_g, S_g, Qa, Qb /* synthesis keep */ ;
  nand (R_g, ~D, Clk);
  nand (S_g, D, Clk);
  nand (Qa, S_g, Qb);
  nand (Qb, R_g, Qa);
  assign Q = Qa;
endmodule
```

Fig. 13a. Instantiating logic gates for the D latch.

```
// A gated D latch
module Part9(Clk, D, Q);
  input Clk, D;
  output Q;

  wire R_g, S_g, Qa, Qb /* synthesis keep */;

  assign R_g = ~(~D & Clk);
  assign S_g = ~(D & Clk);
  assign Qa = ~(S_g & Qb);
  assign Qb = ~(R_g & Qa);

  assign Q = Qa;
endmodule
```

Fig. 13b. Specifying the D latch by using logic expressions

Although the latch can be correctly realized in one 4-input LUT, this implementation does not allow its internal signals, such as R_g and S_g, to be observed, because they are not provided as outputs from the LUT. To preserve these internal signals in the implemented circuit, it is necessary to include a *compiler*

directive in the code. In Fig. 13 the directive /*synthesis keep*/ is included to instruct the Quartus II compiler to use separate logic elements for each of the signals R_g, S_g,Qa, and Qb. Compiling the code produces the circuit with four 4-LUTs, one for each assign statement.

Perform the following steps.

1. The project for this part is provided in the starter kit. Open the project named *Part9* in the *Part9* subdirectory.

2. Generate a Verilog file with the code in either part a or b of Fig. 13 (both versions of the code should produce the same circuit) and include it in the project.

3. Select as the target chip and compile the code. You may wish to use the Quartus II RTL Viewer tool to examine the gate-level circuit produced from the code, and the Technology Map Viewer tool to verify that the latch is implemented with four LUTs. These tools are found under the menu Tools > Netlist Viewers.

4. Remove the /*synthesis keep*/ construct from the code and compile it again and report the difference.

5. Create a VectorWaveform File (.vwf) which specifies the inputs and outputs of the circuit. You need to show it to the TA. Draw waveforms for the D input and use the Quartus II Simulator to produce the corresponding waveforms for *R_g*, *S_g*, *Qa*, and *Qb*. Verify that the latch works as expected using both functional and timing simulation.

6. Create a new Quartus II project which will be used for implementation of the gated D latch on the board. This project should consist of a top-level module that contains the appropriate input and output ports (pins) for the board. Instantiate your latch in this top-level module. Use switch SW0 to drive the *D* input of the latch, and use SW1 as the Clk input. Connect the Q output to LEDR0.

7. Recompile your project and simulate the compiled circuit.

Part X:

Fig. 14 shows the circuit for a master-slave D flip-flop.

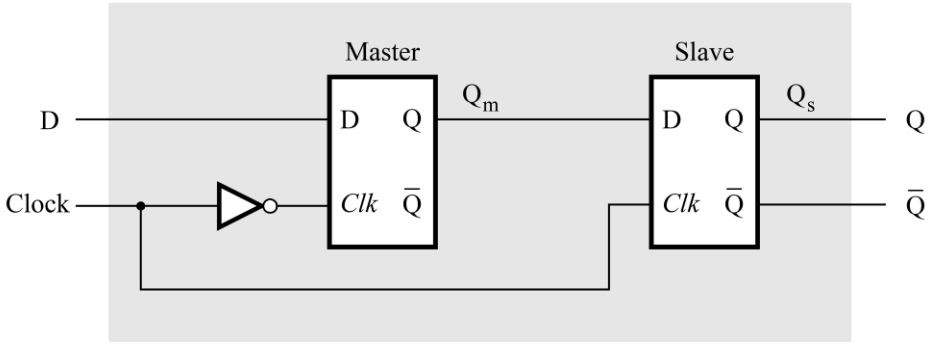


Fig. 14. Circuit for a master-slave D flip-flop.

Perform the following steps:

1. The project for this part is provided in the starter kit. Open the project named *Part10* in the *Part10* subdirectory to begin your work.
2. Generate a Verilog file that instantiates two copies of your gated D latch module from Part IX to implement the master-slave flip-flop.

3. Include in your project the appropriate input and output ports for the Altera board. Use switch SW0 to drive the D input of the flip-flop, and use *SW*1 as the *Clock* input. Connect the Q output to LEDR0. Compile your project.

4. You may wish to use the Quartus Technology Map Viewer to examine the D flip-flop circuit, using the command Tools > Netlist Viewer > Technology Map Viewer.

5. Use simulation to verify the correct operation of your circuit.

Part XI:

Fig. 15 shows a circuit with three different storage elements: a gated D latch, a positive-edge triggered D flip-flop, and a negative-edge triggered D flip-flop.

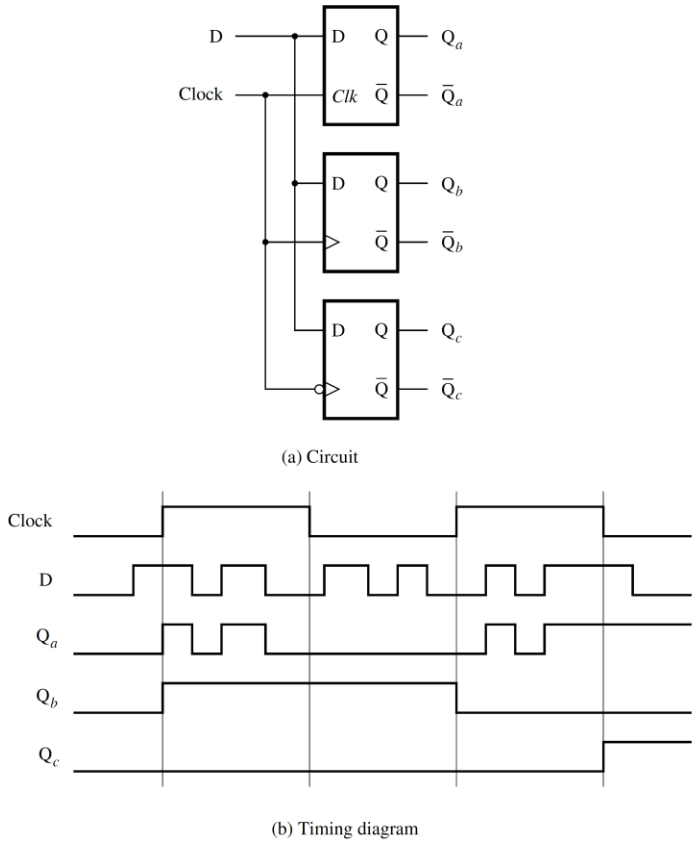


Fig. 15. Circuit and waveforms for Part XI.

Implement and simulate the circuit in Fig. 15 by using the Quartus II software as follows:

1. The project for this part is provided in the starter kit. Open the project named *Part11* in the *Part11* subdirectory to begin your work.

2. Write a Verilog file that instantiates the three storage elements. For this part you should no longer include the /* synthesis keep */ directive in your Verilog code, because you will not be describing the exact structure of flip-flops like you did in Part X. Instead, you should use a style of Verilog code that will allow the Verilog compiler (in the Quartus II software) to automatically *instantiate* flip-flops that are provided as part of the FPGA logic elements. Such Verilog code is often called *behavioral* code, because it specifies a desired circuit behavior rather than an exact circuit structure. As an example, Fig. 16 gives a behavioral style of Verilog code that specifies the gated D latch in Fig. 13. This latch can be implemented in one 4-input lookup table. Use a similar style of code to specify the flip-flops in Fig. 15.

3. Compile your project.

4. You may wish to use the Quartus Technology Map Viewer to examine the compiled circuit, by using the command Tools > Netlist Viewer > Technology Map Viewer.

5. Create a Vector Waveform File (.vwf) which specifies the inputs and outputs of the circuit. You need to show it to the TA. Draw the inputs D and *Clock* as indicated in Fig. 15. Use functional simulation to obtain the three output signals. Observe the different behavior of the three storage elements.

```
module D_latch (D, Clk, Q);
input D, Clk;
output reg Q;
always @ (D, Clk)
if (Clk)
   Q = D;
endmodule
```

Fig. 16. A behavioral style of Verilog code that specifies a gated D latch.

Part XII:

In this part you will need to work in base 16, also known as hexadecimal, or hex for short. There are sixteen digits in hexadecimal, the usual 0-9 and then the letters A through F. It is fairly easy to translate from base 2 to base 16 because each hexadecimal digit corresponds to exactly 4 bits in binary. A sixteen bit number translates into exactly 4 hexadecimal digits.

The goal for the circuit in this part is to display the hexadecimal value of a 16-bit number, A, on the four 7-segment displays, HEX7 − 4, and a different hexadecimal value of a 16-bit number B on the four 7-segment displays, HEX3 − 0. The values of A and B are to be input to the circuit through switches SW15−0, one value at a time. This is to be done by first setting the switches to the value of A and then setting the switches to the value of B; therefore, the value of A must be stored in the circuit. You will need a 16-bit register (which should be clocked by *KEY*1) to store the value of A once it has been set using SW15−0. Please note that the hex decoder is provided in the starter kit (i.e., hex_digits.v).

Perform the following steps:

1. The project for this part is provided in the starter kit. Open the project named *Part12* in the *Part12* subdirectory to begin your work.

2. Write a Verilog file that provides the necessary functionality. Use *KEY*0 as an active-low asynchronous reset, and use *KEY*1 as a clock input.

3. Include the Verilog file in your project and compile the circuit.

4. Assign the pins on the FPGA to connect to the switches and 7-segment displays, as you have done in previous parts of this exercise.