LABORATORY MANUAL

CE/CZ1005 : Digital Logic

[Location: Hardware Laboratory 3, N4-B1a-05]

Experiment 3 :
Combinational Logic Design
with Structural Verilog

2017/2018

COMPUTER ENGINEERING COURSE
COMPUTER SCIENCE COURSE

SCHOOL OF COMPUTER SCIENCE & ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

COMBINATIONAL LOGIC DESIGN WITH STRUCTURAL VERILOG

1. OBJECTIVES

1.1 To obtain minimum-cost SOP Boolean expressions using Karnaugh Maps.

1.2 To explore logic circuit design with the Xilinx FPGA design tool chain.

1.3 To design a 7-segment decoder with structural Verilog statements.

1.4 To selectively display a digit using a push button and multiplexers.

1.5 To implement and test the designed circuit on the Xilinx FPGA.

2. LABORATORY

This experiment is conducted at **Hardware Laboratory 3**, N4-B1a-05.

3. EQUIPMENT AND SOFTWARE

Personal computer
Xilinx FPGA tool chain (ISE, ISIM, iMPACT)
Xilinx University Programme (XUP) Spartan 6 ATLYS development board
Digilent PmodSSD Peripheral Module Board (7-segment display)

4. INTRODUCTION

4.1 In experiments 1 and 2, you have implemented logic circuits using integrated circuits mounted on a breadboard. That is a quick and low-cost method to build and test simple circuits; but it can become messy for more complex circuits. In this experiment, you will implement a 7-segment decoder circuit on the Xilinx FPGA (Field-programmable Gate Array), which comprises logic components that can be programmed to implement the desired logic [Ref 8.4].

Figure 1 shows the diagram of a 7-segment decoder and display. The segments on the display unit are labeled a, b, c, d, e, f and g. By lighting up the correct segments (e.g. a, b and c in Figure 1), different numeric digits (e.g. "7" in Figure 1) can be displayed. Table 1 lists the hexadecimal digits that can be displayed by controlling the 4-bit input x[3:0].

In this experiment, you will be using the Digilent PmodSSD two-digit 7-segment display unit [Ref 8.5].

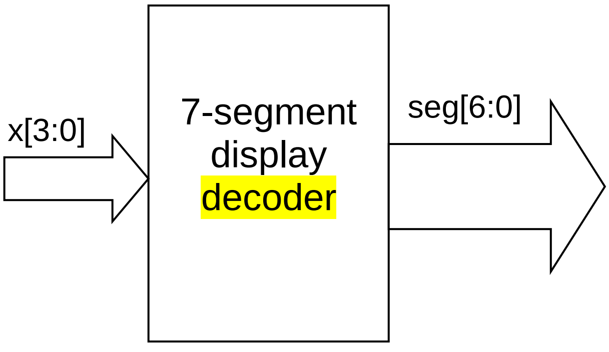


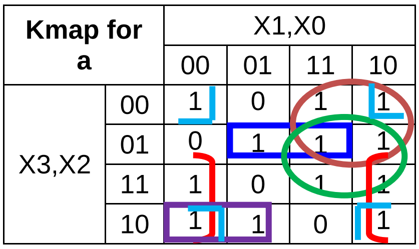
Figure 1: 7-segment decoder and display (e.g. displaying the digit "7")

4.2 Fill in the remaining values in Table 1 to achieve the desired digit display. After that, draw a Karnaugh map for each of the 7 outputs: a, b, c, d, …, and g; obtain the minimized Sum-of-Product (SOP) expression for each output.

For example, the minimized SOP expression for a is:

a = **X3'X2X0** + **X2X1** + **X3'X1** + **X3X0'** + **X3X2'X1'** + **X2'X0'**

where X3, X2, X1, X0 are the 4 bits of the hexadecimal digit (X3 is msb)



Kmap for a		X1,X0			
		00	01	11	10
X3,X2	00	1	0	1	1
	01	0	1	1	1
	11	1	0	1	1
	10	1	1	0	1

Note that in this case a total of 6 loops are required to minimize the SOP: 4 of them cover 4 minterms each, and 2 of them cover 2 minterms each.

4.3 Obtain the minimized SOP expressions for each of the remaining segments: b to g. You will need to obtain these 6 logic expressions before the experiment.

You may make use of an online Kmap tool [Ref 8.7] to verify your expressions but make sure you are able to construct Kmaps on your own.

4.4 You will design the 7-segment decoder using simple structural Verilog HDL (Hardware Description Language) statements to describe logic expressions. The essential information on structural Verilog is included in the Appendix.

For example, the expression in 4.2 can be written as follows in Verilog syntax:

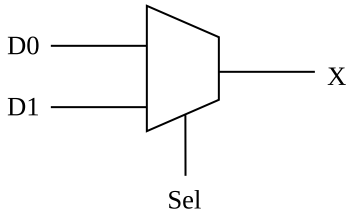
```
assign a = ~x[3]&x[2]&x[0] | x[2]&x[1] | ~x[3]&x[1] | x[3]&~x[0] | x[3]&~x[2]&~x[1] |
~x[2]&~x[0];
```

Table 1: Truth table of a 7-segment display decoder

Inputs x[3:0]	**Display**	**Outputs** seg[0:6]						
Hexadecimal digits (binary)		**Segments (1: on, 0: off)**						
		a	b	c	d	e	f	g
0 (0000)	0	1	1	1	1	1	1	0
1 (0001)	1	0	1	1	0	0	0	0
2 (0010)	2	1	1	0	1	1	0	1
3 (0011)	3	1	1	1	1	0	0	1
4 (0100)	4	0	1	1	0	0	1	1
5 (0101)	5	1	0	1	1	0	1	1
6 (0110)	6	1	0	1	1	1	1	1
7 (0111)	7	1	1	1	0	0	0	0
8 (1000)	8	1	1	1	1	1	1	1
9 (1001)	9	1	1	1	1	0	1	1
A (1010)	A	1	1	1	0	1	1	1
b (1011)	b	0	0	1	1	1	1	1
C (1100)	C	1	0	0	1	1	1	0
d (1101)	d	0	1	1	1	1	0	1
E (1110)	E	1	0	0	1	1	1	1
F (1111)	F	1	0	0	0	1	1	1

Note: seg[0] is segment a, seg[1] is segment b, …, seg[6] is segment g
In digital circuits, a multi-bit signal e.g. x[3:0] is also known as a bus.

4.5 In this experiment, you may also make use of a 2-input multiplexer. Figure 2 shows the logic symbol and truth table. By controlling the logic signal Sel, either the input D0 or the input D1 is directed to the output X at any time.



Sel	Output X
0	D0
1	D1

Figure 2: Logic symbol and truth table of a 2-input multiplexer

4.6 Following a set of procedure, you will design the decoder using the Boolean expressions obtained from 4.3, implement it on the FPGA board, generate the program file and download it to the FPGA board for testing.

4.7 You may contrast this method of logic circuit design and implementation with the breadboard approach used in experiments 1 and 2.

5. EXPERIMENT

5.1 SET UP DESIGN ENVIRONMENT

You may refer to steps 1 to 3 of the *lab3_ise_guide.pdf* for a pictorial guide.

5.1.1 After logging in to the PC with your student account, go to the CECZ1005 folder. Double click the ISE Design Suite 14.4 shortcut to start **Xilinx ISE Project Navigator**.

5.1.2 Create a new project (**File -> New Project**). This opens the ***New Project Wizard***.

5.1.3 In the ***Create New Project*** window, enter a project name (**Lab3**). The lab technicians will let you know the location of the working directory. At the bottom of the window, select the "Top level source type:" as **HDL**. Click **Next**. (HDL = hardware description language)

5.1.4 In the Project Settings window, ensure the correct Family, Device and Package below are selected. Set the "Preferred Language" as **Verilog.** Click **Next.** Examine the ***Project Summary*****,** then click ***Finish*****.**

Family: Spartan6

Device: XC6SLX45

Package: CSG324

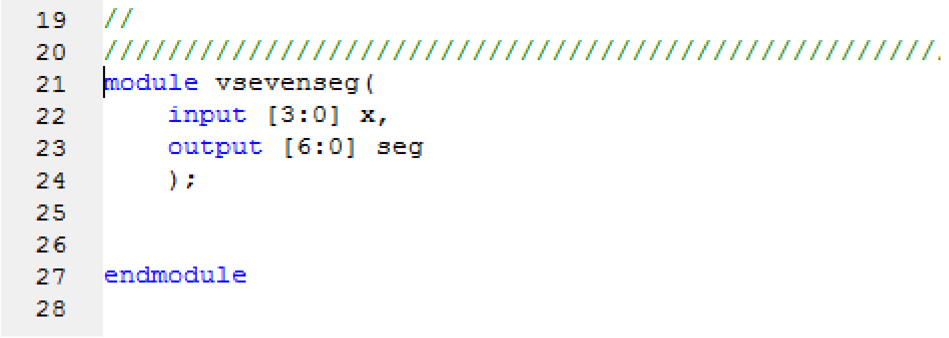
5.2 DESIGN THE 7-SEGMENT DECODER MODULE WITH STRUCTURAL VERILOG STATEMENTS

You may refer to steps 4 to 6 of the *lab3_ise_guide.pdf* for a pictorial guide.

5.2.1 Create a new Verilog source file (**Project->New Source**), this opens the ***New Source Wizard.*** Select **Verilog Module**, enter a filename (**vsevenseg**). Use the default location set earlier in 5.1.3. Ensure the "add to project" checkbox is selected. Click **Next**.

Use the ***Define Module*** interface to define the inputs and outputs of the circuit. The circuit shall have a 4-bit input x (x[3:0], i.e. MSB=3, LSB=0), and a 6-bit output seg (seg[6:0], i.e. MSB=6, LSB=0). Check the "Bus" box since the signals are multi-bit. **Note that Verilog syntax is case-sensitive.**

5.2.2 Click ***Next*** and ***Finish***. You will see a Verilog file *vsevenseg.v* created with the specified inputs and outputs as shown in Figure 3.



```
//
///////////////////////////////////////////////////.
module vsevenseg(
    input [3:0] x,
    output [6:0] seg
    );


endmodule

```

Figure 3: vsevenseg.v with inputs and outputs specified

5.2.3 Download the file *seg.v* from NTULearn to your PC. Open the file with Notepad, copy ALL the content and paste it into the file *vsevenseg.v* (insert between line 25 and line 26 of the code in Figure 3) and click **SAVE**. Examine the file *vsevenseg.v* carefully. Compare the logic expressions of segments a, b and e in the file and the ones you have obtained in 4.3.
Pause and think: Are they the same? If not, why?
Refer to the Appendix for the Verilog logic operators.

5.2.4 Using the logic expressions that you have obtained in 4.3, key in the Verilog expressions of segments c, d, f and g into the file *vsevenseg.v.* Follow the same syntax as the given expressions and don't forget the semi colon at the end of each line. No statement must be inserted after **endmodule**. Click **SAVE**. Leave the file open as you will need to edit it in 5.4

5.3 MAP INPUTS/OUTPUTS AND IMPLEMENT CIRCUIT ON FPGA

You may refer to steps 7 to 11 of the *lab3_ise_guide.pdf* for a pictorial guide.

5.3.1 After the decoder circuit is designed in 5.2.4, you are ready to realize it as a physical circuit using the FPGA. You will be using the Xilinx University Programme ATLYS FPGA development board as the target platform [Ref 8.4].

To see the effect of your circuit's outputs, you need to assign the outputs to the seven-segment display unit. Table 2 shows the required interface connections between the FPGA board and the Digilent PmodSSD Peripheral Module Board, as well as the logic outputs of the *vsevenseg* module designed in 5.2. The physical connection between the FPGA and the display module has already been made for you using two short cables. Do not alter or remove them.

Table 2: Connections from circuit on FPGA to 7-segment display module

	Logic outputs of *vsevenseg*	seg[6]	seg[5]	seg[4]	seg[3]	seg[2]	seg[1]	seg[0]	tglout
UCF mapping	**FPGA pin id**	V4	T9	V9	N5	P6	R3	T3	T4
Cable connection	**Display module**	AG	AF	AE	AD	AC	AB	AA	C

5.3.2 Download the user constraints file (UCF) *sevenseg.ucf* from NTULearn on to the PC and copy it into the Lab3 working directory specified earlier in 5.1.3. Add the UCF file to the project using **Project->Add Source**.

5.3.3 Double click on the added UCF file to open it up and note the given mappings of the logic inputs/outputs from *vsevenseg.v* to the FPGA board. The inputs x[3:0] are mapped to the switches SW3, SW2, SW1 and SW0 shown on Figure 4. Edit the UCF file and add in the remaining mappings shown in Table 2 (except for tglout which is not used in this part). Note that for every I/O (i.e. input/output) pin, two statements are required:
NET "wire_name**" LOC =** pin_id**;**
NET "wire_name**" IOSTANDARD = LVCMOS25;**

5.3.4 **SAVE** the UCF file after all the output mappings have been added in. Leave the file open as you will need to edit it in 5.4.

5.3.5 At the ISE Project Navigator application, in the **Design:View:** window, select the **Implementation** button. Make sure that it is the correct Family, Device and Package specified in 5.1.4. Click to select *vsevenseg.v*. In the **Processes** window, double click **Generate Programming File**. It will take several minutes to complete the bit file generation. If there is error in the process, look at the error messages and fix them (usually in *vsevenseg.v* or *sevenseg.ucf* file) before proceeding further.

5.3.6 Turn on the main power supply to the FPGA board. Push the power switch on the board and a green LED next to it will light up. In the CECZ1005 folder, double click the Adept shortcut icon, browse to the *vsevenseg.bit* file, click **Program**.

5.3.7 Test the circuit by changing the switches. Are all the 16 digits displayed correctly? If not, check your logic expressions in *vsevenseg.v* and the mappings in the *sevenseg.ucf* file.

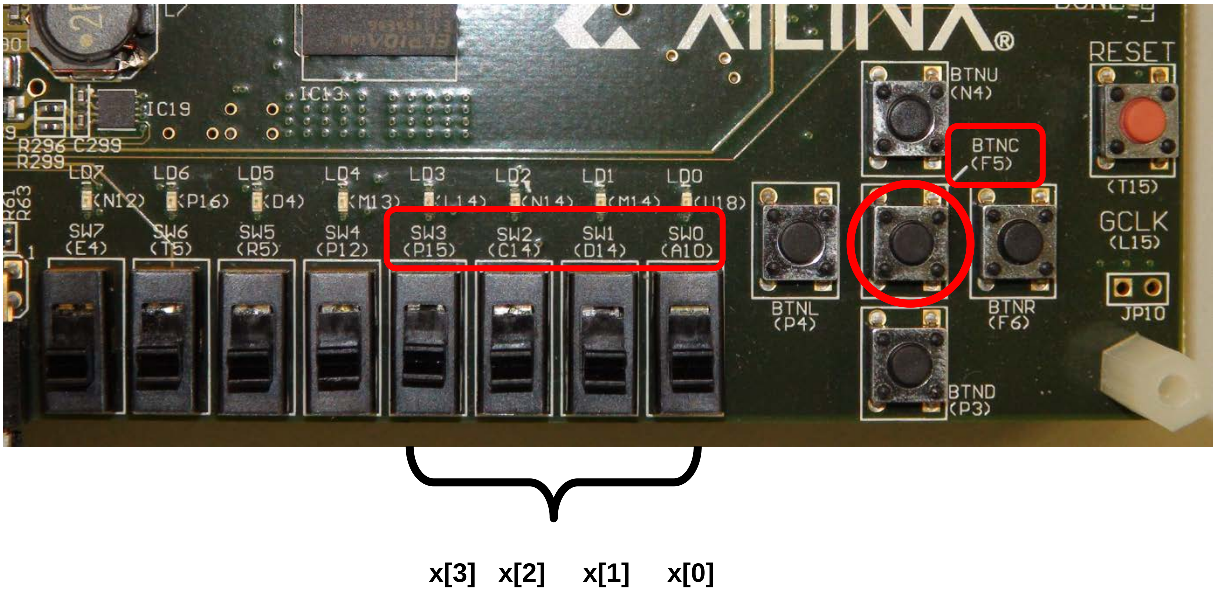


Figure 4: Switches and pushbuttons on the Xilinx FPGA board

5.4 MODIFY CIRCUIT TO DISPLAY EITHER LEFT OR RIGHT DIGIT

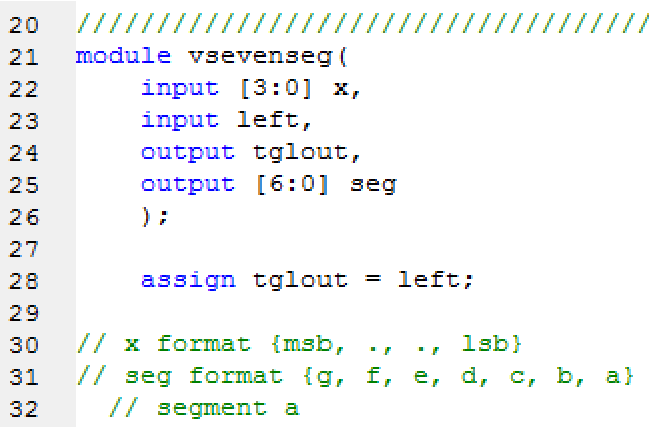
5.4.1 In section 5.3, you notice that only the digit on the right lights up. On the 7-segment display module, the input C (see Table 2 rightmost column) is used to select either the left or the right digit display to light up.

Side note: If the logic level at input C is switched between 0 and 1 rapidly (say 60 Hz), both digits will blink so rapidly that they appear to be lighted up at the same time. You will explore this in lab experiment 4.

5.4.2 You will now add an input to the circuit designed in 5.2.4 to control which digit (left or right) to light up. Edit the file *vsevenseg.v*. Add an input left and an output tglout. Assign the logic level of tglout to be that of left. Figure 5 shows the additional code inserted at line numbers 23, 24 and 28. Note that unlike x[3:0] and seg[6:0], both left and tglout are single-bit signals.

5.4.3 Before the modified circuit can be implemented, you must specify the connections to be made to left and tglout. Edit the UCF to achieve this. Insert additional statements into *sevenseg.ucf* such that left is mapped to pushbutton F5 (BTNC on Figure 4), and tglout is mapped to T4 (see Table 2 rightmost column). Refer to 5.3.3 for the correct syntax. **Save all**.

5.4.4 Follow the same steps in 5.3.5 and 5.3.6 to generate the new bit file and program it on to the FPGA.



```
20 ///////////////////////////////////////
21 module vsevenseg(
22     input [3:0] x,
23     input left,
24     output tglout,
25     output [6:0] seg
26     );
27
28     assign tglout = left;
29
30 // x format {msb, ., ., lsb}
31 // seg format {g, f, e, d, c, b, a}
32   // segment a
```

Figure 5: Additional code inserted at line numbers 23, 24 and 28 for 5.4.2

5.4.5 How do you select the left or right digit to be displayed by using the pushbutton BTNC? Note that when the pushbutton is held down, it generates a logic 1 signal, otherwise it is logic 0. i.e. it is an active high output.
Pause and think: will it be more or less appropriate to name the input as *right* instead of *left*?

5.4.6 Pause and think: If you have inserted the following Verilog statement instead at line number 28 of Figure 5, in what way do you think the circuit behavior will be different?

```
assign tglout = ~left;
```

5.5 OPTIONAL: MODIFY CIRCUIT TO DISPLAY A DIFFERENT VALUE ON THE LEFT AND RIGHT DIGIT

Complete this part only if time permits and you have understood the earlier parts of this experiment.

5.5.1 In 5.4, you are able to display the same hexadecimal value either on the left or the right digit of the 7-segment display module. Here you will display two different values on the two digits, one digit at a time.

5.5.2 To achieve this, you will make use of 2-input multiplexers (see Figure 2). You will learn more about them in this course.

5.5.3 Figure 6 shows the block diagram of the modified circuit (bounded within the blue dashes). The circuit now accepts two 4-bit inputs: a[3:0] for left display value and b[3:0] for right display value. The multiplexer will direct input a[3:0] to x[3:0] when left=1, and input b[3:0] to x[3:0] when left=0. There is no change to the 7-segment decoder circuit itself which was designed in 5.2.

Note on Figure 6 that the mappings of switches to circuit inputs and 7-segment display to circuit outputs are achieved by the UCF file.

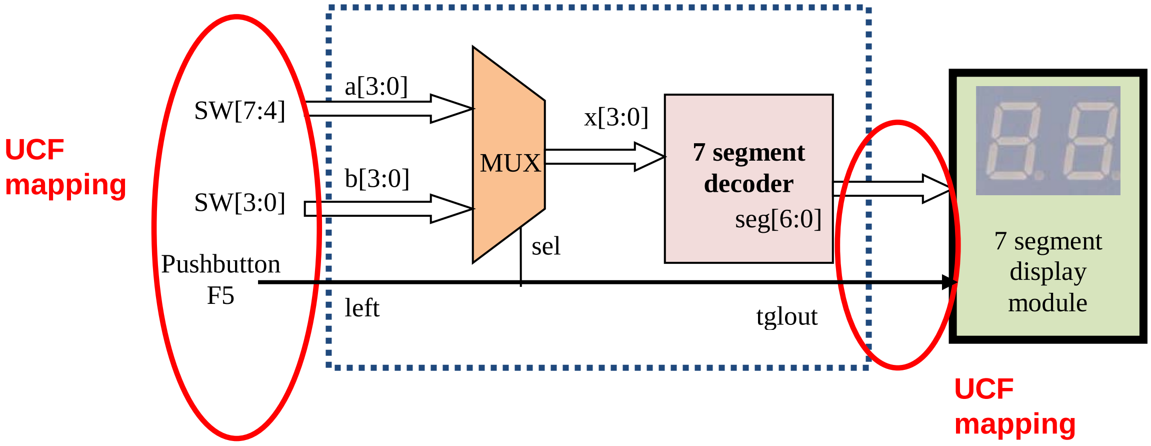
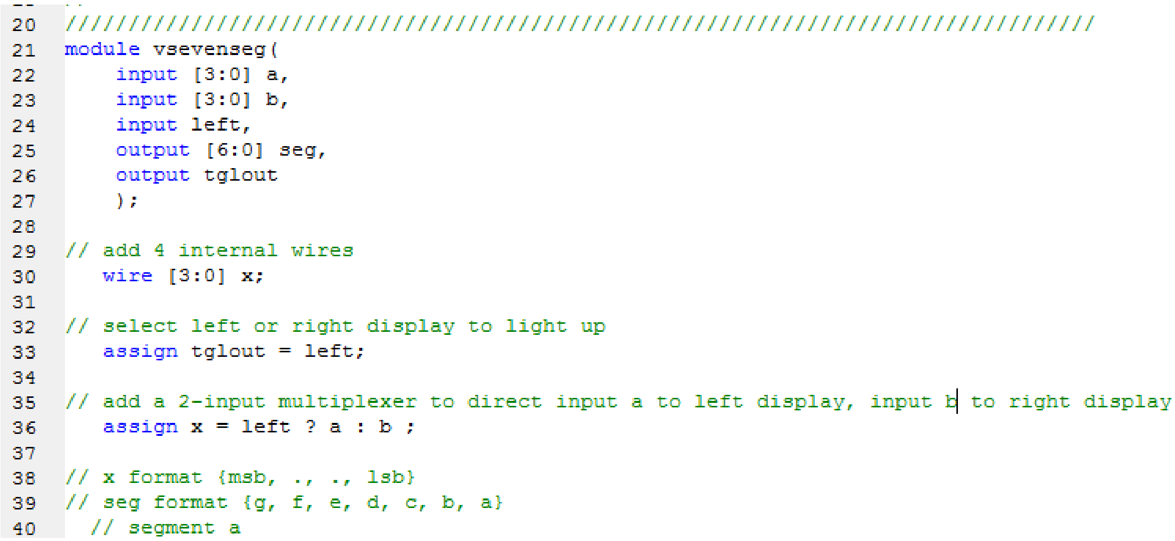


Figure 6: Block diagram of circuit to display different input values

5.5.4 Edit the *vsevenseg.v* file to modify the input and insert the multiplexer. Refer to Figure 7 for the necessary changes at line numbers 22, 23, 30 and 36.



```
20  /////////////////////////////////////////////////////////////////////////////////
21  module vsevenseg(
22      input [3:0] a,
23      input [3:0] b,
24      input left,
25      output [6:0] seg,
26      output tglout
27      );
28
29  // add 4 internal wires
30     wire [3:0] x;
31
32  // select left or right display to light up
33     assign tglout = left;
34
35  // add a 2-input multiplexer to direct input a to left display, input b to right display
36     assign x = left ? a : b ;
37
38  // x format {msb, ., ., lsb}
39  // seg format {g, f, e, d, c, b, a}
40    // segment a
```

Figure 7: Changes to *vsevenseg.v* at line numbers 22, 23, 30 and 36

5.5.5 Open *sevenseg.ucf* to edit it. Write statements to map input a[3:0] to switches SW7, SW6, SW5, SW4 (Table 3 gives the pin id for each switch – they are also visible on the FPGA board above each switch). Rename x[3:0] in the UCF to b[3:0]. **Save all.**

Table 3: pin id of 4 switches on the left

Circuit input	a[3] (msb)	a[2]	a[1]	a[0] (lsb)
Switch name	SW7	SW6	SW5	SW4
pin id	E4	T5	R5	P12

5.5.6 Follow the steps in 5.3.5 and 5.3.6 to generate the new bit file and program it on to the FPGA.

5.5.7 Set switches SW[7:4] and SW[3:0] to two different 4-bit values. Observe the values displayed on the left and right digit by pressing and releasing the button F5.

Pause and think: How would you modify the circuit such that the left and right values are swapped? i.e. the value set by the 4 switches SW[7:4] gets displayed on the right, while the value set by the 4 switches SW[3:0] gets displayed on the left?

6. PREPARATION AND OBSERVATIONS

Maintain a proper record of your preparation (e.g. Kmaps, derivations of logic expressions, sketches of logic circuit diagrams, etc.) and observations throughout the experiment. You will be able to refer to this record during the lab quiz.

7. LAB ASSESSMENT AND QUIZ

You will need to demonstrate the completion of section 5.3 and 5.4 to the lab instructor.

There will be a 10-minute written lab quiz at the end of this experiment. The quiz will include (but not limited to) materials covered in this experiment.

8. REFERENCES

8.1 Laboratory materials, Digital Logic Couse Site, NTULearn.
8.2 Digital Design Principles and Practices, Ed. 4, John F Wakerly, Prentice Hall, 2007.
8.3 Fundamentals of Digital Logic with Verilog Design, 2nd Ed., by Stephen Brown and Zvonko Vranesic, McGraw Hill, 2008
8.4 Xilinx Atlys_FPGA_DevBoard documentation.
8.5 Digilent PmodSSD Peripheral Module Board Reference Manual.
8.6 http://en.wikipedia.org/wiki/Seven-segment_display
8.7 http://www.32x8.com/ (online Kmap tool)

Appendix

Note on Xilinx ISE:

For .v, .sch and .ucf files, it is not sufficient to place them in the same working directory as the design. They need to be added to the design using **Project->Add Source.** Similarly, these files can be **removed** from the design when they are not needed. Removing the files from a design will not delete the files from the working directory.

Verilog bitwise operators

Operator	**Logic**	**Examples**	**Meaning**
~	NOT	~A	NOT A
&	AND	A&B	A AND B
|	OR	A|B	A OR B

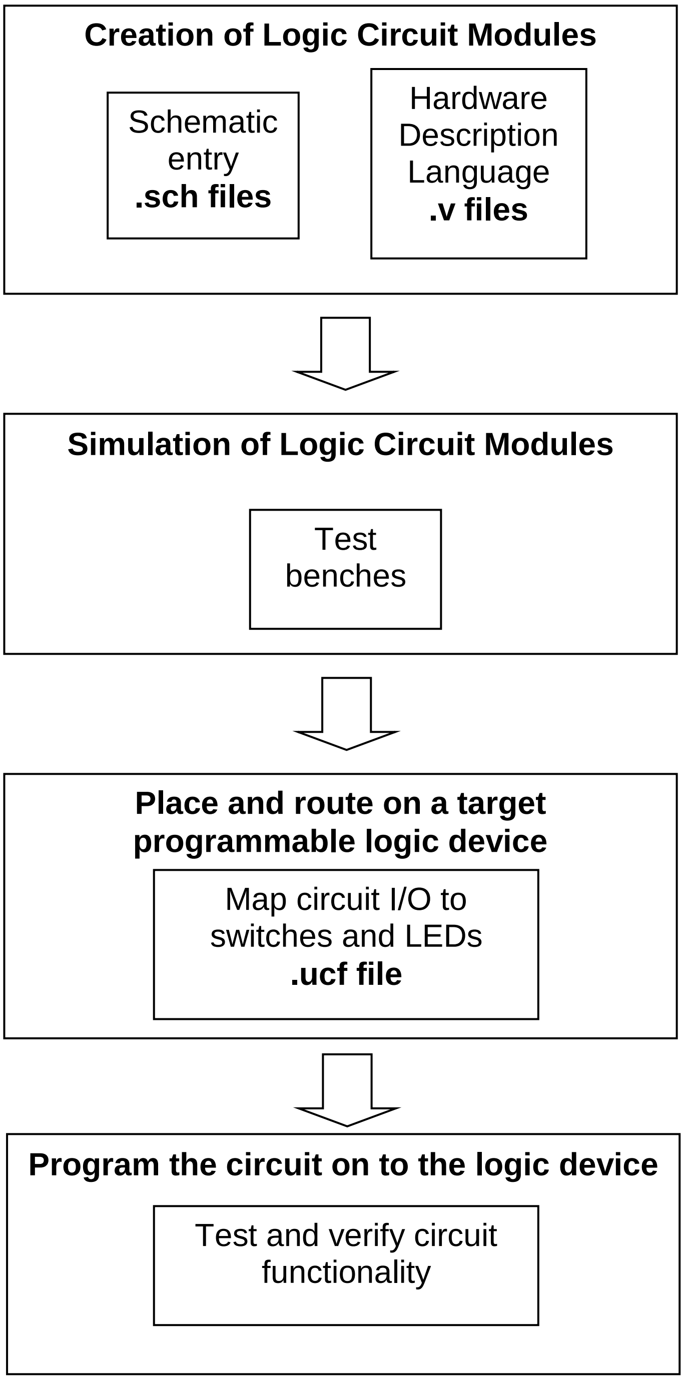
Note: usual order of precedence applies

A typical Structural Verilog module declaration

```
module module_name (
        input input_name1,
        input input_name2,
        input input_name3,
        etc.,
        output output_name1,
        output output_name2,
        etc.);

        assign output_name1 = logic expression;

        assign output_name2 = logic expression;

        … etc.

Endmodule
```

Note: each **assign** statement must end with a semi colon (;)

A Typical Computer Aided Logic Design Process


Creation of Logic Circuit Modules
Schematic entry **.sch files**
Hardware Description Language **.v files**
Simulation of Logic Circuit Modules
Test benches
Place and route on a target programmable logic device
Map circuit I/O to switches and LEDs **.ucf file**
Program the circuit on to the logic device
Test and verify circuit functionality