Lab 5 – 7-Segment Display

CS 2052 Computer Architecture

Dept. of Computer Science and Engineering, University of Moratuwa

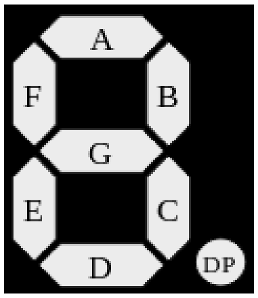
Learning Outcomes

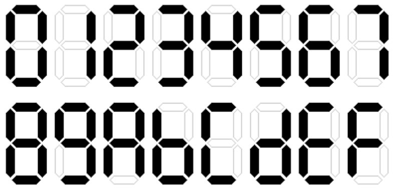
In this lab, we will design a 7-segment display for our 4-bit Arithmetic Unit developed in previous lab. After completing the lab, you will be able to:

- design and develop a lookup table using Read Only Memory (ROM)
- design and develop a 7-segment display using the output from the lookup table
- verify and demonstrate their functionality via simulation and on the development board

Introduction

7-segment display is a form of electronic device for displaying decimal numerals. It is an alternative to the more complex displays like dot matrix displays in LCD and LED screens. BASYS 3 has 4, 7-segment displays. Following figure shows the 7 segments, the decimal point on a 7-segment display, and a possible representation of hexagonal numbers (numbers between 0 and 15).





Depending on the number we want to display, we can switch on or off the desired segment(s). Which segment you may want to switch on or off also depends on the specific 7-segment implementation, where it may light up depending on an active high (common cathode circuit) or active low input (common anode circuit).

In this lab, we will display the output of our 4-bit arithmetic unit (developed in Lab 6) as a hexadecimal number using a 7-segment display. Depending on the 2, 4-bit input numbers, the arithmetic unit will produce a 4-bit sum, a carry, and zero flag. While in previous labs we indicated the resulting sum using a set of LEDs, in this lab we will use a 7-segment display to show the output of 4-bit sum from the RCA as a hexadecimal number. Therefore, we need to decide which segments to light up depending on the sum produced by the RCA. One way to do this is to write logic equations for inputs to each of the segments (using k-maps). Instead, we will use a lookup table to map the 4-bit sum to the 7-segments on the display. Such a lookup table can be built using a ROM.

Building the Circuits

Step 1: Using the BASYS 3 Reference Manual find out whether the BASYS 3 board uses a common cathode circuit or a common anode circuit (refer Sec. 8.1, pg. 15).

Complete the following table to find out the mapping between the 4-bit sum and the corresponding 7-segment code. Some lines are filled as examples.

Output from RCA					**Segments to Switch On**						
S_3	**S_2**	**S_1**	**S_0**	**Hex. Value**	**A**	**B**	**C**	**D**	**E**	**F**	**G**
0	0	0	0	0	*0*	*0*	*0*	*0*	*0*	*0*	*1*
0	0	0	1	1	*1*	*0*	*0*	*1*	*1*	*1*	*1*
0	0	1	0	2							
0	0	1	1	3							
0	1	0	0	4							
0	1	0	1	5							
0	1	1	0	6							
0	1	1	1	7							
1	0	0	0	8							
1	0	0	1	9							
1	0	1	0	A	*0*	*0*	*0*	*1*	*0*	*0*	*0*
1	0	1	1	B							
1	1	0	0	C							
1	1	0	1	D							
1	1	1	0	E							
1	1	1	1	F							

Step 2: Building 4-bit arithmetic unit symbol.

Create a new project in Xilinx Vivado and name it as **Lab 7**.

Import all relevant VHDL files from previous lab (e.g., AU and all its low-level entities and slow clock). Make sure to copy these files to your new project.

Step 3: Building Lookup Table.

A lookup table is used to find the mapping between inputs S_3 - S_0 and outputs A - G in the above table. As the mapping will not change with time, we can save it in a ROM. Another alternative is to build a combinational circuit to convert 4-bit input to 7-bit output. However, with the objective of learning how to use ROMs, we will use a ROM-based lookup table. To further simplify we will use an asynchronous ROM where its operation is not controlled by a clock.

Create a new VHDL file and name it as **LUT_16_7** (which stands for lookup table). Set **address** and a 4-bit input bus and **data** and a 7-bit output bus.

Add the following VHDL code after architecture definition line while completing missing lines.

```
type rom_type is array (0 to 15) of std_logic_vector(6 downto 0);

     signal sevenSegment_ROM : rom_type := (
                              "1000000", -- 0
                              ...
                              "0001000", -- a
                              ...
                              "0001110"  -- f
          );
```

As we have 16 output combinations to represent 0 to F in hexadecimal, we need to instruct the ROM from which location/address we need to send the output to 7-segment display. As there are 16 memory locations in ROM, we need a 4-bits to address each location uniquely. Here keyword `rom_type` indicates the type of memory we want is a ROM. We define the ROM as an array of 16 elements using `array (0 to 15)`, where `std_logic_vector(6 downto 0)` indicates each array location contains 7-bits. Then we define the content of each array position starting from index 0 up to index 15.

Add the following line after begin keyword to set the data output based on the given address.

```
data <= sevenSegment_ROM(to_integer(unsigned(address)));
```

Our ROM is created using `integer` data type, whereas address is of type `std_logic_vector`. Therefore, to read the memory location, address value must be changed to `integer` type. However, we can directly cast `std_logic_vector` to an `integer`; therefore, we first convert it to `unsigned` and then converted to `integer`. For this we need to import numeric library using following statement (add it after the line that says `use IEEE.STD_LOGIC_1164.ALL;`)

```
use ieee.numeric_std.all;
```

Test the functionality of the LUT by simulating the circuit. At least try four distinct input cases based on the binary representation of your index no.

Step 4: Build High-Level Circuit.

Create a new VHDL file and save it as **AU_7_seg**.

Now add the 4-bit **AU** (from Lab 6) and **LUT_16_7** to the schematic. Connect their pins such that the output from RCA in AU is fed to both the LEDs and look up table.

Label the inputs to AU as **A**, **Clk**, and **RegSel**. Label the outputs as **S_LED**, **S-7Seg**, **Carry**, and **Zero**.

Step 5: Connecting inputs and outputs.

Input and output pins are connected only to the top-level design **AU_7_seg** (so make sure to set the top-level design). Connect switches **SW0**-**SW3** as **A** inputs and **SW15** as **RegSel** input. Connect outputs **S_7Seg** to **CA**-**CG** (on 7-segment), **S_LED** to LEDs **LD0**-**LD3**, **Carry** to **LD14, Zero** to **LD15**. Use BASYS 3's internal clock for **Clk**.

Step 6: Test on BASYS 3.

Generate the programming file (i.e., bitstream) and load it to the BASYS 3 board.

Change the switches on the BASYS 3 and verify the functionality of your circuit (check the output of 7-segment display and LED).

You may realize that though you plan to show the output using only a single 7-segment, all 4 7-segments will light up. Using the BASYS 3 manual, find out how to show your results only on the right most 7-segment display.

Demonstrate the circuit to the instructor and get the Lab Completion Log singed.

Step 10: Lab Report

You need to submit a report for this lab. Your report should include the following:

- Student name and index number. Do not attach a separate front page
- State the assigned lab task in a few sentences
- Filled up table with hexadecimal codes
- All VHDL files (label figures as Annex 1, Annex 2, ...). No need to include HA, FA, and RCA.
- Timing diagram
- Briefly describe how can you show your results using only a single 7-segment
- Conclusions from the lab

Submit the lab report at the beginning of the next lab.

Prepared By

- Dilum Bandara, PhD – Mar 06, 2014.
- Updated on Oct 25, 2018.