



California State University, Bakersfield
Computer & Electrical Engineering & Computer Science
ECE 322: Digital Design with VHDL
Laboratory 3

Rational: The purpose of this lab is to introduce the student to the following key concepts and ideas.

1. Using VHDL and the compiler built into Quartus II as a Logic Synthesis tool to replace schematic capture.

Abstract:

In lab 2, you learnt how to design a simple logic system (a BCD to 7-Segment decoder) the hard way, by translating the requirements for a logic system into a truth table and then optimizing the Boolean expressions prior to drawing the schematic diagram using basic gates, simulating and downloading to the DE2.

For relatively simple problems such as the BCD to **7-segment LED** display driver, this was not *too* onerous a task, however, when digital systems become more complex, designing circuits out of individual gates is tedious, time consuming and very error prone.

In this part of the lab we are going to use VHDL to **synthesize** a logic system from simple **textual descriptions** of the requirements of the system, i.e. its **behavior**. We shall then compile the code and download to the DE2 board.

In other words, we are going to compare the manual process of designing and producing digital circuits using truth tables, Boolean equations and basic logic gates **vs.** that of using a **VHDL** compiler to **synthesize** the (*optimized*) design from our description of the circuit behavior.

To simplify the **comparison**, the problem will start off being the same as the lab 2 namely the BCD-to-7 segment display driver problem but, just to make it slightly more interesting, we will enhance it later on in the lab to implement a **Hex-to-7** segment display able to display the digits **0-9** and **A-F**.

In addition, we will see how easy it is to create an arithmetic unit in VHDL

What do I have to demonstrate?

You have to demonstrate Parts D, E and F below (which implies that you have done Parts A, B and C), running on the DE2 boards and also show your VHDL code and schematics (where relevant) to the TA.

Part A:

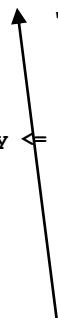
Step 1: Create a new [project](#) in Quartus II called [BCD](#) (since this is the name of the top level entity/circuit in the project – see code below)

Step 2: Create a new [VHDL file](#) (menu File->new) and [save it](#) as [BCD.vhd](#). Describe the entity (real-world input and output signals as buses) plus an architecture looking like this. You can cut and paste the text below into your VHDL file.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY BCD IS
    PORT(
        Inputs : in std_logic_vector(3 downto 0) ; -- inputs A,B,C,D
        Display : out std_logic_vector(6 downto 0) -- outputs a,b,c,d,e,f,g
    );
END;

ARCHITECTURE Behavioural OF BCD IS
BEGIN
    PROCESS(Inputs)          -- process sensitive to changes in these signals
    BEGIN
        -- describe how the values of ABCD in range {0000 - 1001}, i.e. 0 - 9
        -- map to abcdefg using if-else statements etc
        if(Inputs = "0000") then
            Display <= "      ";          -- fill in the required 7 bit value
        elsif(Inputs = "0001") then
            Display <= "      ";          -- fill in the required 7 bit value
        .
        .
        .
        else
            Display <= "      ";          -- fill in the required 7 bit value
        end if ;
    END PROCESS ;              -- end of the process
END;
```



Now fill in the missing statements inside the ‘PROCESS’ to describe how the inputs map to outputs, using if...elsif...else statements. Your truth tables from Lab 2 will help.

Step 3: Make sure you can compile the project successfully.

Step 4: Create a logic symbol for the BCD decoder (click on menu [file->create/update->create symbol files for current file](#)). Now create a [new block diagram file](#) and place the new symbol for your BCD decoder onto it. Save the block diagram file as [BCDto7SegDecoder.bdf](#)

Step 5: Make the above new block diagram file the [top level entity](#) in your design (in the [project navigator window](#), click on the [files](#) tab and then [right click](#) on your new block diagram file and select '[Set as top level entity](#)')

Step 6: Create pins for the symbol (right click on your logic symbol on the block diagram file and select '[Generate pins for symbol ports](#)').

Step 7: Recompile to check for errors.

Step 8: Using the Quartus II pin planner tool, assign the inputs above to four slider switches on the DE2 board (use the DE2 user manual to look up the pins). Take care that you map them correctly, for example, is Inputs (0) above in the entity declaration the '[A](#)' input or the '[D](#)' input? Likewise is the signal Display (6) the segment '[a](#)' output, or the segment '[g](#)' output. Finally does a logic 0 or logic 1 applied to the LED in the 7-segment display to light up?

Step 9: Recompile.

Step 10: Download the [.sof](#) file to the DE2 board and check operation.

Step 11: Add extra statements to the VHDL PROCESS block above to decode the binary values [1010](#), [1011](#), [1100](#), [1101](#), [1110](#), [1111](#) and thus display the letter [A](#), [B](#), [C](#), [D](#), [E](#), [F](#) on the 7 segment display (you'll have to figure out which segments need to be turned on to create [A](#), [B](#), [C](#), [D](#), [E](#), [F](#) on the display. Note you can use upper or lower case [A](#), [B](#), [C](#), [D](#), [E](#), [F](#)). Don't forget to recompile!!!!

How easy was this compared to truth tables, Boolean Algebra/Karnaugh Maps and complex gate level schematic capture?

Part B:

Mirroring the steps we did in Part A above, start a [new Quartus project](#) and design in VHDL a [priority encoder](#) with 16 inputs (that you will eventually connect to the 16 slider switches on the DE2 board). The circuit should output a 2 digit BCD code corresponding to the number of the switch with the highest priority that is active, e.g. if switches 5 and 16 are active (logic 1), then the output from your priority encoder circuit should be (in binary) [00010110](#) i.e. 16 in BCD. **Hint:** use if-elsif-else type statements similar to Part A above to map inputs to outputs. If no switches are active, the output should be [0000 0000](#).

Make sure you know which way around the switches are connected, is the switch in the 'up' position generating a logic '1' or a logic '0' ?

Add a further single bit output called '[Active](#)' that is '0' when [none](#) of the 16 slider switches are active, and '1' when at least one of them is active.

Create a new logic symbol for the Priority Encoder and paste this into a new block diagram, (a file in Quartus. Make the new block diagram file your “[top level entity](#)” as you did in Part A above.

- Generate pins for the new Priority Encoder symbol (as you did in Part A) to represent the real-world inputs and outputs.
- Compile to check for errors.
- Use the pin planner to map the inputs to 16 slider switches and outputs to the 8 Green LEDs. The ‘Active’ output can be mapped to a single red LED.

Compile and download the ‘.sof’ file to the DE2 board and verify the operation by changing the switches and checking that the appropriate LEDs light up.

Part C:

Start a new Quartus project, as we did in Part A and Part B above and create a new VHDL file to describe the operation of a 4-1 multiplexer with four 4 bit inputs called ‘A’, ‘B’, ‘C’ and ‘D’, and a single 4 bit output ‘X’. Your circuit should also include a two bit channel select input ‘S’.

1. The 4 bit inputs ‘A-D’ can be mapped in the Quartus [pin planner](#) to the 16 slider switches on the DE2 board SW0-SW15. Make sure you know which way around the switches are connected, is the switch in the ‘up’ position generating a logic ‘1’ or a logic ‘0’ ?
2. The output X can be mapped in the pin planner to any of the 4 individual LEDs on the DE2 board. (The single LEDs turn ‘on’ when logic 1 is presented to them, unlike the 7segment displays).
3. The input ‘S’ can be mapped in the pin planner to **two of the 4 push buttons** on the DE2 board, these push buttons produce a logic 0 when pressed.

Create a symbol for the multiplexer and paste this into a new schematic diagram, (a block diagram file in Quartus). Make the new block diagram file your “[top level entity](#)” as you did in Part A and B above.

- Generate pins for the new multiplexer symbols (as you did in Part A/B) to represent the real-world inputs and outputs.
- Use the pin planner to map the inputs and outputs to switches, buttons and LEDs.
- Compile to check for errors.

Compile and download the ‘.sof’ file to the DE2 board and verify that by changing ‘S’, the output ‘X’ is set to the 4 bit value of either ‘A’, ‘B’, ‘C’ or ‘D’ as defined by the switches, thus implementing a 2 channel, 4 bit multiplexer.

Part D

- Create a new VHDL “**hexto7Segment**” file as part of the **same project** that you created in Part C above and paste the VHDL code from the file we wrote in Part A (the **hexto7segment** decoder) into it.
- Create the logic symbol for the **hexto7Segment** file and paste the symbol onto the top level schematic file that you already have in this project (the one with the multiplexer on it).
- Connect the 4 bit output ‘X’ from your multiplexer to the 4 bit input of your hex to 7 segment decoder.
- Generate pins for the output of the 7 segment decoder to represent the real world outputs, recompile to check for errors then using the pin planner map these to one of the 7 segment displays on the DE2 board, e.g. Hex0.
- Recompile, download and verify that the BCD display now displays in Hex, either of the ‘A-D’ switch settings when you press the 2 push buttons that select the input channels of the mux.

Part E

Using the project you already have from Part D above create a new VHDL file, plus a symbol for it (*which you can paste onto the existing schematic diagram from Part D above*) to compare four 4 bit unsigned inputs ‘A-D’ (which come from the slider switches and also go into the multiplexor) and turn on one or more of the 3 spare Red LED’s on the DE2 according to the following comparisons. Simulate and Test the operation on the DE2 board. (don’t forget to set up the pins in the pin planner)

$A = B = C = D$	(turn on Led 1, off otherwise).
$A > B$ and $A > C$ and $A = D$	(turn on Led 2, off otherwise). A
$\leq B$ or $(A < C$ and $A \neq D)$	(turn on Led 3, off otherwise).

Part F

Starting a **new project**, design a circuit in VHDL to perform simple arithmetic/Boolean operations on two 8 bit **unsigned** inputs ‘A’ and ‘B’ (set using 16 slider switches on DE2 board). The result (including any carry/borrow and overflow output in the case of addition/subtraction) should be displayed on red or green LEDs, you choose which.

$A + B$	Add A to B
$A - B$	Subtract B from A
A and B	AND each bit of A with corresponding bit in B to produce 8 bit result A
or B	OR each bit of A with corresponding bit in B to produce 8 bit result

The operation is defined by the two remaining slider switches not allocated to **A** or **B**.

Note that to produce the carry/borrow output during addition/subtraction, it will be necessary to make **A** and **B** temporarily into 9 bit operands using a leading 0 in the m.s.b. position (investigate the '&' operand in VHDL to concatenate or merge signals in a bus. We can then add the 9 bit operands using a 9 bit adder and the m.s.b. of the 9 bit result will be the value of the carry/borrow).