

Lab 17: Building a 4-Digit 7-Segment LED Decoder

In this lab you will make 5 test circuits in addition to the 4-digit 7-segment decoder. The test circuits will explore the behavior of the nib2led_decoder and the LF_clock_source, as well as develop VHDL code for the toggle flip-flop, mux, and decoder. You will use the 4-digit 7-segment decoder as a component in the next two labs. The final circuit is a working demonstration of the 4-digit 7-segment display decoder module. The final circuit will use as modules (lower level VHDL code blocks) the nib2led_decoder, LF_clock_source, mux, decoder, and toggle flip-flop. This circuit will accept one 16-bit binary number and display it as four hexadecimal digits on the 7-segment LED. The term “decoder” is applied because it converts the binary input to the proper combination of outputs to form characters on the 7-segment display that have meaning to us humans. Note the Basys3 board has four 7-segment displays and 16 slide switches.

Be sure to save your circuits, you will need them in the next two labs!

Submission of the lab consists of:

- 1) (0pts) a cover sheet with you name and section
- 2) (82 pts total) a print out of the 6 sets of VHDL programs (all modules included)
 - 20 pts (10 pts ea) parts A-B
 - 45 pts (15 pts ea) part C-E
 - 17 pts part F
- 3) (18 pts total) a print out of the constraints files (*.xdc) [Delete the unused sections before printing.]
 - 18 pts (3 pts ea) parts A-F
- 4) Demonstrate the working project (part F only) on the Digilab board to your instructor.

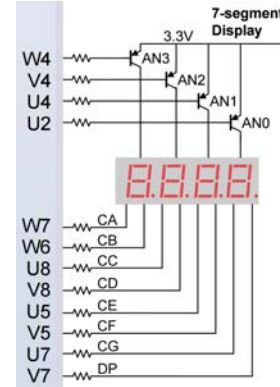
Failure to demonstrate your project will cause your grade for the above to be discounted by 50%. A circuit that does not meet the required specifications will result in your lab being discounted by a minimum of 10%.

New in this lab are:

- working with the 7-segment LED display
- using toggle flip-flops
- using multiplexers (muxes)
- using decoders
- working with hexadecimal (base 16) and binary (base 2) numbers
- incorporating existing sources into the project (LF_clock_source.vhd, nib2led_decoder.vhd)

These instructions give you a general guide to completing the lab assignment. However some details are left for you to discover, such as implementing the toggle-flip flop, mux, and decoder in VHDL and how to make all the components work correctly with your digilab board. If your first try doesn't work the way you expect, observe the behavior of your circuit. Hypothesize why it is doing what it is doing. Make a plan to fix it and implement the plan. Good designs are iterative.

The 4-Digit 7-Segment LED Display. We might at first expect that each segment of each digit has its own separate connection (wire) to control it. In practice this is not done because it would require 7 separate wires for each digit in the display and a separate “decoder” for each digit. (See chapter 13.5.7 in your text and the reference manual for the Basys3 board.) In practice, the digits are “multiplexed” by connecting together the corresponding segments of the digits. This forms a buss, a set of wires that connect the segments. The buss is comprised of 7 wires, one for each segment CA–CG (C for cathode). In jargon we say that the segments are “bussed together”. (See the Digilab board schematic.) Thus writing to segment CA writes the same “data” to all four digits of the display. Which digit is active is controlled by AN0, AN1, AN2, and AN3 (A for anode).



An LED has two terminals. In our display the anodes of the segments in each digit are connected together, so there is one anode connection for each digit. In 7-segment-LED jargon this is referred to as “common anode”. We use the common anode to “address” the active digit. In the Digilab board the active digit is controlled by the pnp transistors on AN0–AN3. To activate one digit, the corresponding base resistor is connected to ground. The economy of hardware is obvious. Without multiplexing, an n -segment display would require $7n$ wires and n decoders. With multiplexing, an n segment display requires $7+n$ wires and one decoder.* The economy is obvious even with a four digit display.

The Multiplexer (mux). A multiplexer is a digitally controlled switch. Multiplexers are described in detail in your text ch 12.3. You will write VHDL code for a 4-way mux that accepts a 16-bit array input and selects from the input array one of four 4-bit sub arrays (four nibbles) as output, controlled by an input 2-bit selector array.

The Decoder (different from the nib2led decoder). Decoders (also called demultiplexers) are described in detail in your text ch 12.3. You will write VHDL code for a 4-line decoder that outputs a 4-bit array, only one element of the array is true. Which element is true is controlled by an input 2-bit selector array.

The nib2led_decoder. The decoder has a 4-bit array input and a 7-bit array output. Each element of the output array corresponds one segment of the 7-segment display. Thankfully the elements of the output array are in the same format expected for the Basys3_master.xdc file. Note that these control the segments of ALL FOUR 7-segment displays. Within each digit, all segments share a common anode that is connected to +3.3 V through a transistor “switch”. The pins connected to AN0–AN3 control these switches.

There are 16 possible values for the 4-bit input array. It is helpful and convenient to think of the array as a 4-bit binary number, a nibble. A nibble half a byte, 4 bits versus 8 bits. The ones place is nib0 (2^0) and is referred to as the *least significant bit* or LSB. The *most significant bit* or MSB in this group of four is nib3, the eights place (2^3). To write this as a four-digit binary number the order would be nib3, nib2, nib1, nib0. This is also the way arrays are treated in VHDL. To save space and make the number more readable, groups of 4-bits are represented by a single base-16 digit (hexadecimal or hex for short). A single hexadecimal digit has a decimal value of 0-15, which in hex is 0-F. (see the table)

nib2led_decoder I/O Map							
nib(3)	nib(2)	nib(1)	nib(0)	binary	hexadecimal	decimal	LED
0	0	0	0	0000	0	0	0
0	0	0	1	0001	1	1	1
0	0	1	0	0010	2	2	2
0	0	1	1	0011	3	3	3
0	1	0	0	0100	4	4	4
0	1	0	1	0101	5	5	5
0	1	1	0	0110	6	6	6
0	1	1	1	0111	7	7	7
1	0	0	0	1000	8	8	8
1	0	0	1	1001	9	9	9
1	0	1	0	1010	A	10	A
1	0	1	1	1011	B	11	b
1	1	0	0	1100	C	12	C
1	1	0	1	1101	D	13	d
1	1	1	0	1110	E	14	E
1	1	1	1	1111	F	15	F

TESTING TESTING TESTING, 1, 2, 3, 4. Before we embark on a any project using components we are developing, or have not used before, it is advisable to first test each in a simple design. This is good programming and design practice. Trouble shooting is easier in a simpler code!

Part A) Testing the display and the nib2led decoder.

Summary. Create a new project “seven_seg_test” and top level VHDL code using nib2led_decoder as a module. Use the four slide switches sw(0-3) as inputs for the nib2led decoder and seg(0-6) as outputs to the 7-segment display. **Assign the switches so that sw3 is the MSB and sw0 is the LSB.** This way the pattern on the switches will look just like the binary numbers in the table above. Connect an(0-3) to the push buttons btnR, btnD, btnL, and btnU. Connect the decimal point dp to btnC.

Through this test code, you will determine the correct control logic to turn on only one digit at a time and show the display pattern corresponding to the nibble provided by the slide switches. In our projects we want the decimal point turned off. You will need to determine the logic to do that as well.

Test your circuit. Try all 16 possible combinations of the slide switches. btnR will activate the right most digit (digit 0), btnD, second digit (digit 1), and so on. The symbols displayed should correspond to the table shown for the nib2LED_decoder. It is important that the bits are in the correct order so that sw3 is the MSB and sw0 is the LSB. When no button is pressed, no digits should light. The decimal point should be completely dark.

Part B) Testing the low frequency clock source.

Background. The Basys3 board has a 100 MHz system clock, which is way too fast for our application. The “LF_clock_source.vhd” module takes the 100 MHz system clock as an input and divides it by 10^5 , 10^6 , 10^7 , and 10^8 , providing those results as a 4-bit array output of lower frequency clocks. These output clock frequencies are approximately 1 kHz, 100 Hz, 10 Hz, and 1 Hz, respectively.

Summary. Create a new project “LF_clock_test”. Add LF_clock_source to the project as a module. Create top-level VHDL to use the LF_clock_source. Connect the inputs to the system clock and the four outputs to led(0-3). **Assign the LEDs in order of increasing speed, led0 to 1 Hz through led3 to 1 kHz.**

Test your circuit: led0 should flash at approximately 1 Hz frequency, led1 should be ten times faster. If you wave the board in the dark, the flashing of led2 and led3 should be evident as dashed trails due to persistence of your retina. Save this program, you will find it useful later.

Part C) Implementing and testing the toggle flip flop (TFF).

Background. In the final step of this project we will use the TFF to create the MSB of the selector array used in the mux and decoder. If the LSB is the 100 Hz clock, we can use the TFF to divide that clock frequency by 2. When combined, they produce a selector array that continuously cycles 00, 01, 10, 11, In VHDL a TFF can be implemented in several ways. I encourage you to look at examples in the *language template* in the Vivado editor. Many examples can also be found on the internet. You will need to know enough VHDL to adapt the examples to your project.

Summary. Create a new project “TFF_test” and top level source to use your TFF. Assign sw(0) to the TFF toggle input T, btnR to input clk, and led(1) to the TFF output Q. Also assign btnR to led(0).

Test your circuit. Pressing btnR should cause led0 to light. If sw0 is on, then pressing btnR will cause led1 to toggle (led1 will turn on if it was off and turn off if it was on). If sw0 is off, the toggling of led1 will not occur.

Part D) Implementing and testing the mux.

Background. A mux is simply a switch. Here we have a 16-bit array of data that we need to break into 4 nibbles so it can be displayed. In VHDL a mux can be implemented in several ways. I encourage you to look at examples in the *language template* in the Vivado editor. Many examples can also be found on the internet. You will need to know enough VHDL to adapt the examples to your project.

Summary. Create a new project “mux_test” and top level source to use your mux. Assign sw(0-15) to the mux input I (16-bit array) and led(0-3) to the mux output O (4-bit array). Assign the 2-bit selector array S to btnR and btnL.

Test your circuit. The selector array (btnR and btnL) is a 2-bit binary number (00 to 11) that addresses which nibble of the inputs (sw0-15) appear at the output, led(0-3).

Part E) Implementing and testing the decoder.

Background. We will use the decoder to select which digit should be illuminated based on the 2-bit selector array. In VHDL a decoder can be implemented in several ways. I encourage you to look at examples in the *language template* in the Vivado editor. Many examples can also be found on the internet. You will need to know enough VHDL to adapt the examples to your project.

Summary. Create a new project “decoder_test” and top level module to use your decoder. Assign sw(0-1) to the decoder input S (2-bit array) and led(0-3) to the decoder output O (4-bit array).

Test your circuit. The selector array sw(0-1) is a 2-bit binary number (0-3) that addresses which of the element of the O output array is “1”.

Part F) Building the 4-Digit 7-Segment LED Decoder.

THIS CIRCUIT WILL TO BE USED AS A COMPONENT IN LATER LABS.

Summary. Create a new project “four_dig_seven_seg_decoder” and top level VHDL code using nib2led_decoder, LF_clock_source, and the VHDL components you created (mux, decoder, TFF) as a modules. The inputs will be a 16-bit array (sw) and the clock (100 MHz system clock). The outputs will be a 7-bit array (seg) corresponding to the seven led segments, the decimal point (dp), and the 4-bit array (an) corresponding to the digits.

To test the code, use the 16 slide switches sw(0-15) as inputs. **Assign the switches so that sw15 is the MSB and sw0 is the LSB.** This way the pattern on the switches will

look just like the binary numbers in the table. Connect the outputs to the 4-digit 7-segment display.

Make certain that your decoder is working correctly. The four digits displayed should follow the specifications in the following table.

The binary input to your demonstration circuit should be arranged so that **sw15 is the MSB and sw0 is the LSB.** sw(12-15) will be the 4 bits that control the left most digit and sw(0-3) will be the 4 bits that control the right most digit. This way the pattern on the switches will look just like the binary numbers in the table below. Your finished program must be able to reproduce the table below and all numbers in between. (The table below is not complete, a complete table has 65536 entries.)

4-Digit 7-Segment Decoder					
Digit 3	Digit 2	Digit 1	Digit 0	binary	four hexadecimal digits
bit(3-0)	bit(3-0)	bit(3-0)	bit(3-0)		
sw(15-12)	sw(11-8)	sw(7-4)	sw(3-0)		
0000	0000	0000	0000	0000 0000 0000 0000	0000
0001	0000	0000	0001	0001 0000 0000 0001	1001
0000	0010	0000	0100	0000 0010 0000 0100	0204
1110	0111	1010	0011	1110 0111 1010 0011	E7A3
0101	1011	0110	1101	0101 1011 0110 1101	5B6D
1100	1000	1001	1111	1100 1000 1001 1111	C89F
1001	1111	0011	0101	1001 1111 0011 0101	9F35
1111	1111	1111	1111	1111 1111 1111 1111	FFFF

Footnote from 4-Digit 7-Segment LED Display section.

* In reality the number of address lines needed at the processor can be further reduced using a dedicated multiplexer to address active digits. While the display physically needs one address wire per digit, a dedicated multiplexer requires as input, the number of lines necessary to represent the number of output lines as a binary number. Thus 8 different digits can be uniquely addresses using 3 address lines to the multiplexer. The trade off is that only one digit can be turned on at a time. All other possibilities would be sacrificed.