

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

In this lab you will make 4 test circuits, the 4-digit 7-segment decoder, and demonstration circuit using the decoder. The test circuits are simply used to test the `LF_clock_source` and the `hex2LED_decoder`, as well as the built-in mux and the decoder. From these components you will build the 4-digit 7-segment decoder circuit that you will use in the next two labs. The final circuit is a working demonstration of the 4-digit 7-segment display decoder module. This circuit will accept four 4-bit binary numbers (or 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 Nexys board has four 7-segment displays.

### **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 your name and section,
- 2) (80 pts total) a print out of the 6 schematic diagrams (from the schematic editor),
  - 20 pts (5 pts ea) parts A-D
  - 50 pts part E
  - 10 pts part F
- 3) (20 pts total) a print of the pin assignment file (\*.ucf) [click on “edit constraints (text)” which will bring up the \*.ucf file in a text editor from which it can be printed]
  - 12 pts (3 pts ea) parts A-D
  - 8 pts part 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%.

#### **Your instructor has a working model you can inspect.**

#### **New in this lab are:**

- working with the 7-segment LED display
- using multiplexers (muxes)
- using decoders
- working with hexadecimal (base 16) and binary (base 2) numbers
- incorporating an external source into the project (`LF_clock_source.vhd`, `hex2led_decoder.vhd`, and your 4-digit 7-segment decoder module)

These instructions give you a general guide to completing the lab assignment. However, some details of the logic are left for you to discover. 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 requires 7 separate wires for each digit in the display and a separate “decoder” for each digit. (See appendix 1.1 in your text and the reference manual for the Nexys board). 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 A-G. The jargon is to say that the segments are “bussed together”. (See the Digilab board schematic.) Thus writing to segment A writes the same “data” to all four digits of the display (CA on the Nexys board). Which digit is active is controlled by AN0, AN1, AN2, and AN3.

An LED has two terminals. In our display the anodes are connected together. In 7-segment-LED jargon this is referred to as “common anode”. To light the display, the anode is connected to ground. We can use the common anode to “address” the active digit. In the Digilab board the active digit is controlled by AN0–AN3, the switch transistors on the common anode of each digit. 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. We could make them from individual logic gates, but we will just use the existing modules supplied with the schematic capture program (m4\_1e).

**The Decoder** (different from the hex2LED decoder) Decoders are described in detail in your text ch 12.3. We could make them from individual logic gates, but we will just use the existing modules supplied with the schematic capture program (d2\_4e).

**The hex2led\_decoder.** The decoder has 4 inputs and 7 outputs. Each of the output terminals is labeled  $a$  through  $g$  and corresponds to the standard labeling of 7-segment displays. Starting at the top with  $a$  and proceeding clockwise around the outside to  $f$ . Segment  $g$  is in the middle. Thankfully the standard designation is also followed in the Digilab board schematic. The corresponding terminals are CA through CG. 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 combinations of the 4 inputs. Notice that they are labeled bit0, bit1, bit2, and bit3. It is helpful and convenient to think of these as digits in a 4-bit binary number. The ones place is bit0 ( $2^0$ ) and is referred to as the *least significant bit* or LSB. The *most significant bit* or MSB in this group of four is bit3, the eights place ( $2^3$ ). To write this as a four digit binary number the order would be bit3 bit2 bit1 bit0. 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)

hex2led_decoder I/O Map							
bit3	bit2	bit2	bit0	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 project using a module we have not used before, it is advisable to first test it in a simple circuit. This is good programming and circuit practice. Trouble shooting is easier in a simple circuit!

**Part A) Testing the low frequency clock source.**

**Background.** The Nexys board has a system clock which is selectable at 100MHz/50MHz/25MHz by jumper 4 (JP4). For our application this is too fast. The “LF\_clock\_source.vhd” module has four low frequency clock outputs. These outputs are simply the input (clock) divided the clock by  $10^8$ ,  $10^7$ ,  $10^6$ , and  $10^5$ . For a 100 MHz input clock, these output clock frequencies are 1 Hz, 10 Hz, 100 Hz, and 1 kHz, respectively.

**Summary.** Create a new source schematic in your project “LF\_clock\_test”. Add LF\_clock\_source to the project. Connect the inputs to the system clock and the four outputs to LD0-LD3. **Assign the LEDs in order of increasing speed, LD3 to 1 Hz through LD0 to 1 kHz.** .

**Adding LF\_clock\_source to you project.** The following procedure will add the LF\_clock\_source symbol to the library of symbols in the schematic capture utility. Copy the file “LF\_clock\_source.vhd” into your project directory. You can download it from the course web site or copy it from the lab directory on the DVD. With the “LF\_clock\_test.sch” highlighted in the “sources in project” window right click and select “add sources” and select “LF\_clock\_source.vhd”. It will appear in the “sources in project” window as “LF\_clock\_source-behavioral (LF\_clock\_source.vhd)”. Now highlight it. From the “Processes in source” window, double click on “create Schematic Symbol”. (See the updated Digilab Introduction with new information on working with modules.)

Insert the LF\_clock\_source symbol into the “LF\_clock\_test” schematic. The LF\_clock\_source has 1 inputs and 4 outputs. Connect the circuit as described in the summary above.

Test your circuit. With JP4 set to 100 MHz, LD3 should flash with an approximately 1 Hz frequency, LD2 should be ten times faster. If you wave the board in the dark, the flashing of LD1 and LD0 should be evident as dashed trails due to persistence of your retina. The flashing of LD1 will also be apparent with JP4 set to 25 MHz. Save this program, you will find it useful later.

**Part B) Testing the display and the decoder.**

**Summary.** Create a new source schematic in your project “LED\_test”. Add hex2LED\_decoder to the project. Connect the four inputs to the four slide switches SW0-SW3. **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 the outputs of hex2LED\_decoder to the 7-segment buss CA-CG. Connect AN0-AN3 to BTN0-BTN3 (through buffers or other component). [NOTE: Input terminals

cannot be connected directly to output terminals!!] Make sure the decimal point is off at all times.

The 7-segment display on the Nexys board also has a decimal point (CDP). You must also force this too be off, otherwise it will be dimly illuminated when the digit is activated. You can do this by connecting it to a fixed logic level, in the symbols category, select “general”. Two symbols are useful in our circumstance, “pullup” and pulldown”.

**Adding hex2LED\_decoder to you project.** The following procedure will add the hex2LED\_decoder symbol to the library of symbols in the schematic capture utility. Copy the file “hex2led\_decoder.vhd” into your project directory. You can download it from the course web site or copy it from the lab directory on the DVD. With the “LED\_test.sch” highlighted in the “sources in project” window right click and select “add sources” and select “hex2led\_decoder.vhd”. It will appear in the “sources in project” window as “hex2led\_decoder-behavioral (hex2led\_decoder.vhd)”. Now highlight it. From the “Processes in source” window, double click on “create Schematic Symbol”. (See the updated Digilab Introduction with new information on working with modules.)

Insert the hex2led\_decoder symbol into the “LED\_test” schematic. The decoder has 4 inputs and 7 outputs.

Test your circuit. Try all 16 possible combinations of the slide switches. BTN0 will activate the right most digit (digit 0), BTN1, second digit (digit 1), and so on. The symbols displayed should correspond to the table shown for the hex2LED\_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 C) Testing the mux.**

**Background.** You are encouraged to read the description of the component. One way to open this information is as follows. First insert the component into your schematic. Right-clicking on the symbol opens the context menu, select “symbol” then “symbol information”.

**Summary.** Create a new source schematic in your project “mux\_test”. Add “m4\_1e”. Assign SW0-SW3 to the mux inputs D0-D3 and LD0-LD3, respectively. Assign the SW4-SW5 to S0-S1 and LD4-LD5, respectively. Connect SW6 to E and LD6. Connect the mux output to LD7.

Test your circuit. The select lines, S0 and S1, form a 2-bit binary number (0-3) that addresses which of the inputs (D0-D3) appear at the output. The output is enabled by E.

### **Part D) Testing the decoder.**

**Background.** You are encouraged to read the description of the component. One way to open this information is as follows. First insert the component into your schematic.

Right-clicking on the symbol opens the context menu, select “symbol” then “symbol information”.

**Summary.** Create a new source schematic in your project “decoder\_test”. Add “D2\_4e”. Assign SW4-SW5 to the decoder inputs A0-A1 and LD4-LD5, respectively. Assign the SW7 to E and LD7. Connect the decoder outputs D0-D3 to LD0-LD3, respectively.

Test your circuit. The address lines, A0 and A1, form a 2-bit binary number (0-3) that addresses which of the outputs (D0-D3) is active. The output is enabled by E.

### **Part E) Building a 4-Digit 7-Segment LED Decoder.**

**THIS CIRCUIT IS DESIGNED TO BE USED AS A MODULE IN LATER LABS.**

Open a new project. (Do not begin the name of your project or schematic with a number. Hyphens are also not acceptable characters for the name.) Insert the “hex2LED\_decoder” into your project. The 4-Digit 7-Segment LED Decoder will accept four 4-bit binary numbers (or one 16-bit binary number) and display it as four hexadecimal digits on the Nexys 7-segment LED. The strategy is to connect the hex2led\_decoder outputs to segments A-G of your display. One of the 4-bit binary numbers is selected by four 4-input muxes (m4\_1e) and applied to the input of the hex2LED decoder which is then written to your display. The same signal that controls the muxes is used to select the active digit (AN0-AN3) using a decoder (D2\_4e). Clearly only one digit can be selected at a time and each digit should correspond uniquely to one 4-bit number.

Label the data inputs d0b0, d0b1, d0b2, d0b3, d1b0, .... d3b3. This clearly groups the data by hex digits or nibbles (one nibble = one half byte). The ordering should be such that d0b0 is the LSB and d3b3 is the MSB of the 16 bits. The address for the digits is A0 and A1, with A0 the LSB and A1 the MSB, so that (A1, A0) = 0,0 sends d0 to the right most digit, 0,1 sends d1 to the digit second from the right, and so on. Label the outputs of the circuit CA-CG, CDP (7-segments plus the decimal point (forced to be off)) and AN0-AN3 (the 4 anode controls). The circuit has a total of 18 inputs and 12 outputs.

### **Part F) Building a Demonstration circuit for your 4-Digit 7-Segment LED Decoder.**

Open a new project. (Do not begin the name of your project or schematic with a number. Hyphens are also not acceptable characters for the name.) Insert the 4-digit\_7-segment\_decoder, the LF\_clock\_source, and the hex2LED\_decoder into your project.

Assign SW0-SW7 to d0b0-d1b3 and d2b0-d3b3, respectively. This will make the left two digits identical to the right two. Assign the system clock to the input of the LF\_clock\_source and its 1000 Hz output to A0. Assign A1 to BTN0. Assign the outputs of the 4-digit 7-segment decoder to their corresponding functions.

As you are aware, the 4-digit 7-segment LED decoder circuit will accept four 4-bit binary numbers, but only one unique digit can be displayed at one time. To give the appearance

of having multiple digits display simultaneously, the address lines to the decoder can be rapidly and repetitively cycled through all the digits. If this is done fast enough, they appear continuously illuminated to the eye. In this demonstration, we connect A0 to the 1000 Hz clock, so that two digits are “simultaneously” illuminated. BTN0 selects the upper (pressed) or lower (not pressed) bank of digits to display. In Lab 19, we will make use of all 4 digits.

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

The binary input to your demonstration circuit should be arranged so that **SW7 is the MSB and SW0 is the LSB**. SW4-SW7 will be the 4 bits that control the left digit and SW0-SW3 will be the 4 bits that control the right 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. (The table below is not complete, a complete table has 256 entries.)

2-Digit 7-Segment Decoder									
Digit 1				Digit 0				binary	LED1,LED0
bit3	bit2	bit1	bit0	bit3	bit2	bit1	bit0		
SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0		
0	0	0	0	0	0	0	0	0000 0000	00
0	0	0	0	0	0	0	1	0000 0001	01
0	0	0	0	0	0	1	0	0000 0010	02
0	0	0	1	0	0	0	0	0001 0000	10
0	0	1	0	0	0	0	0	0010 0000	20
1	1	0	0	0	1	0	1	1100 0101	C5
1	1	0	0	1	0	1	0	1100 1010	CA
1	1	1	1	1	1	1	1	1111 1111	FF

**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.