Seven-Segment Display

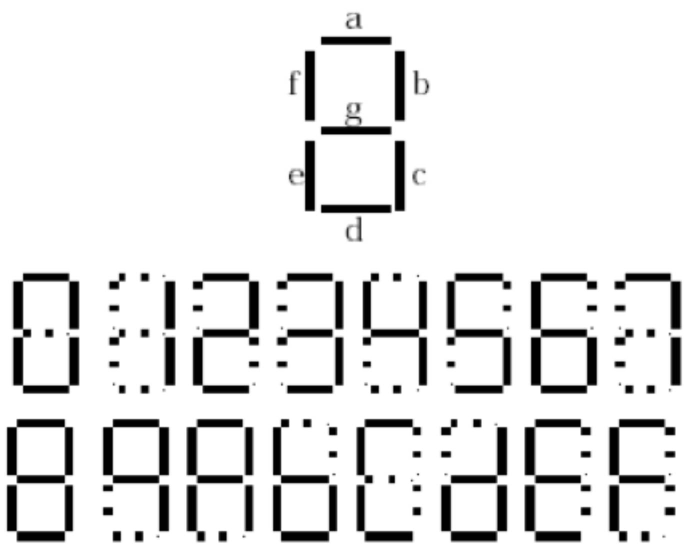
ENEE 245: Digital Circuits and Systems Laboratory
Lab 8

Objectives

The objectives of this laboratory are the following:

- To become familiar with the seven-segment displays on the Nexys2 board
- To design a circuit using decoders and multiplexers that drives the seven-segment displays on the Nexys2 board
- To implement a 4-digit hex-to-7-segment decoder on the Nexys2 FPGA prototyping board

Seven-segment displays are commonly used as alphanumeric displays by logic and computer systems. A seven segment display is an arrangement of 7 LEDs (see below) that can be used to show any hex number between 0000 and 1111 by illuminating combinations of these LEDs. For example, the red digits on a digital clock use 2-segment LED displays. 7-segment displays come in two flavors: common anode and common cathode. A common anode 7-segment display has all of the anodes tied together while a common cathode 7-segment display has all the cathodes tied together.

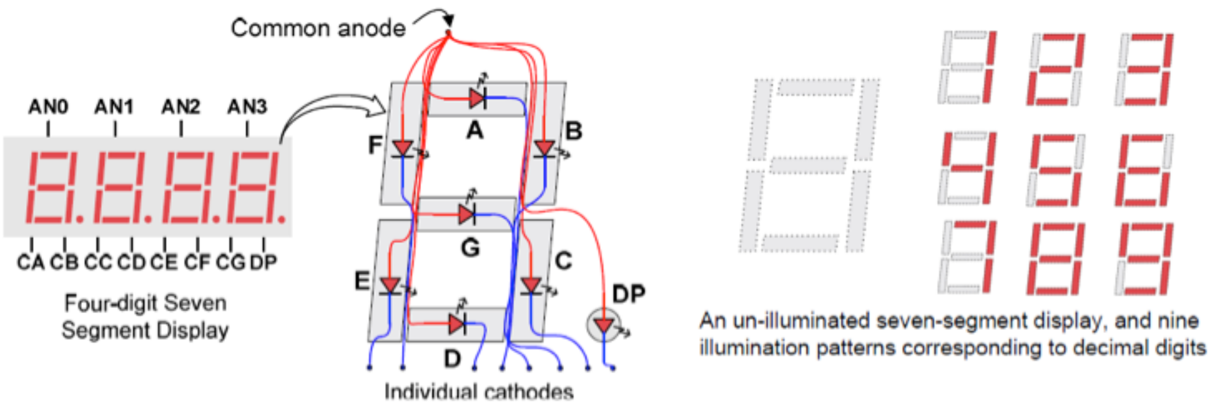


A 7-segment display contains seven light emitting diodes (LEDs)

Seven-Segment Displays on the Nexys2 Board

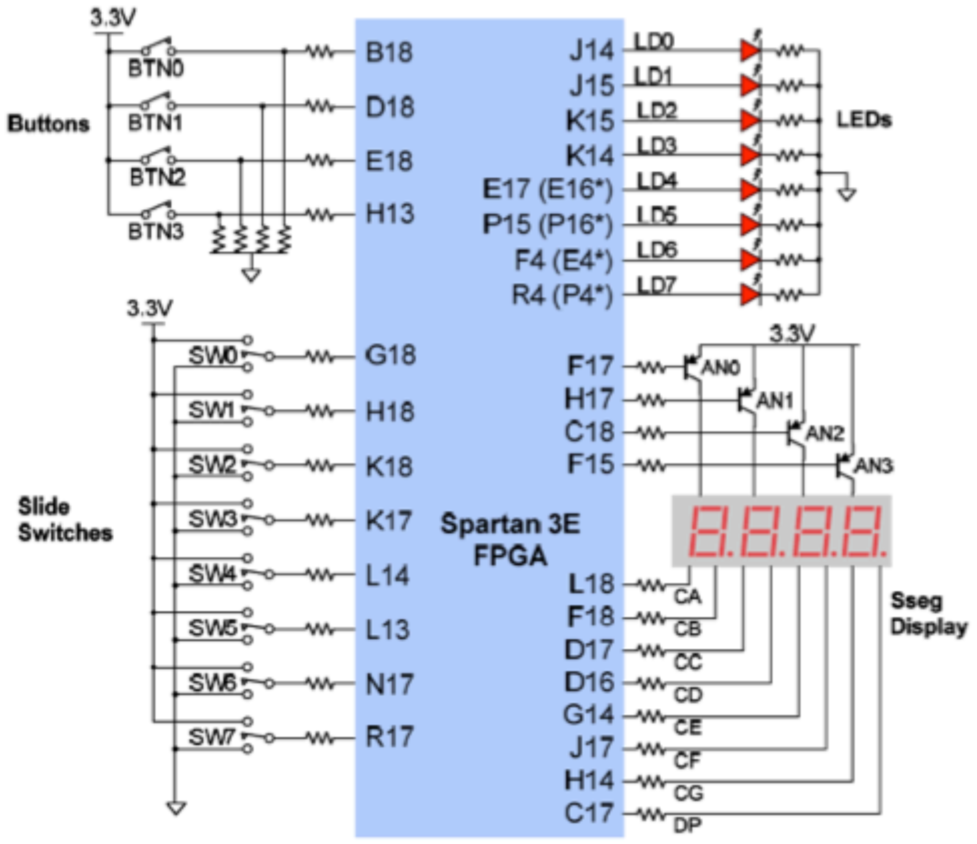
The Nexys2 board has four 7-segment displays. Each seven-segment display consists of seven LED bars and a single LED round (for the decimal point), as shown in the figure below. You can reference the Digilent Nexys2 Board Reference Manual for more information:

http://www.digilentinc.com/Data/Products/NEXYS2/Nexys2_rm.pdf


Common anode
AN0 AN1 AN2 AN3
CA CB CC CD CE CF CG DP
Four-digit Seven Segment Display
A
B
C
D
E
F
G
DP
Individual cathodes
An un-illuminated seven-segment display, and nine illumination patterns corresponding to decimal digits

Nexys2 seven-segment displays

The Nexys2 board uses the common anode method for its displays. This means that all the anodes are tied together and connected through a pnp transistor to +3.3V, as shown in Figure 7.3. A different FPGA output pin is connected through a 100Ω current-limiting resistor to each of the cathodes, a – g, plus the decimal point. A control signal of 0 will turn on an LED segment and a signal of 1 will turn it off. A hex-to-7-segment decoder takes a 4-bit input (a Hex digit) and generates the corresponding 8-bit pattern to light the appropriate LED segments in the display.



Nexys2 I/O devices and circuits

The table below shows output cathode values for each segment a – g needed to display all hex values from 0 – F.

x	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
A	0	0	0	1	0	0	0
B	1	1	0	0	0	0	0
C	0	1	1	0	0	0	1
D	1	0	0	0	0	1	0
E	0	1	1	0	0	0	0
F	0	1	1	1	0	0	0

```
1 = off

0 = on
```

Hex-to-7-Segment Decoder: Logic Equations

To display hexadecimal digits on a 7-segment display, we need to design a hex-to-7-segment decoder (called *hex7seg*), whose input is a 4-bit number (x[3:0]), and outputs are the 7-segment values *a – g* given by the truth table above. We can make a Karnaugh map for each segment and then develop logic equations for the segments *a – g*. For example, a logic equation for the segment *e* is

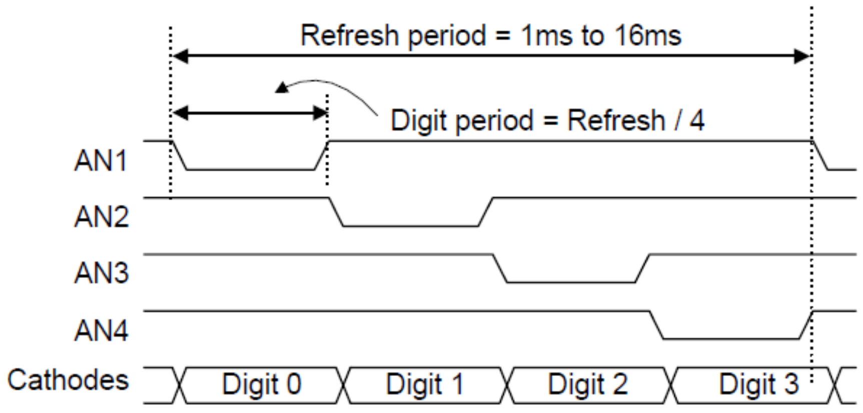
```
e = ~x[3] & x[0] | ~x[3] & x[2] & ~x[1] | ~x[2] & ~x[1] & x[0]
```

You can similarly develop equations for the other six segments and then write the Verilog program for the 7-segment decoder.

Multiplexing 4 Hex-to-7-Segment Displays

As described in the Nexys2 User's Guide, the Nexys2 board designers saved FPGA pins by wiring the four seven-segment displays to the same set of control lines. The user can display four separate characters "simultaneously" by time-multiplexing the seven-segment display control lines at a fast enough rate so that the human eye views all four of the displays as ON and displaying the correct value. Each digit is illuminated just one-quarter of the time, but because the eye cannot perceive the darkening of a digit before it is illuminated again, the digit appears continuously illuminated. If the "refresh" rate is slowed to about 45 hertz, the display will start flickering.

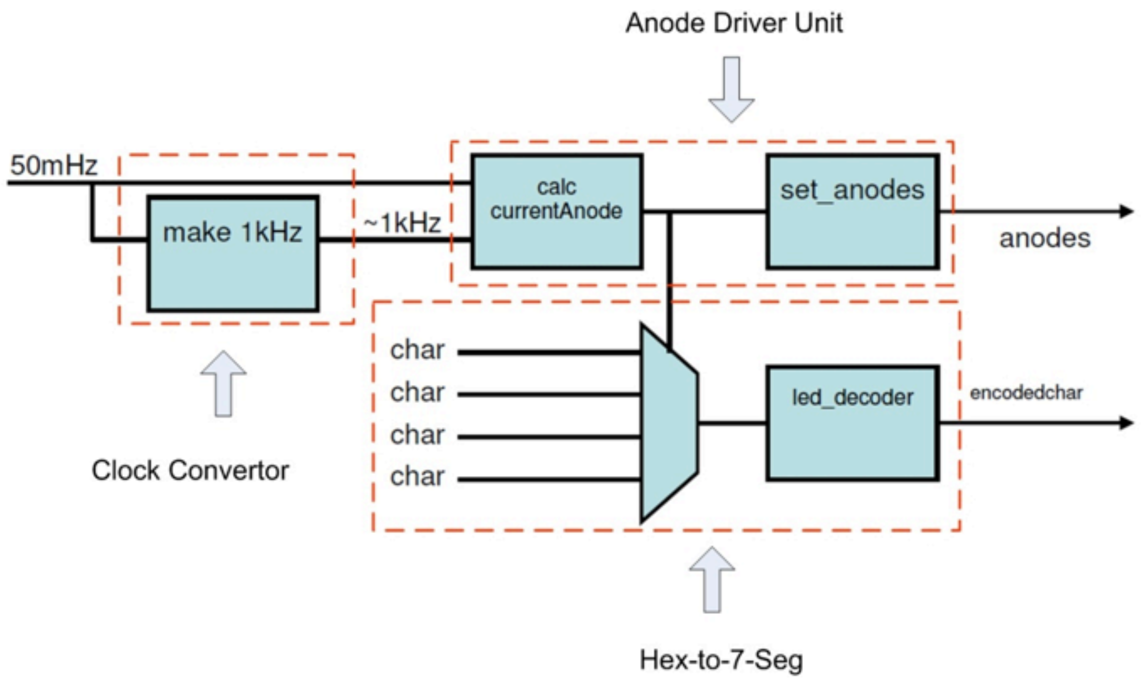
For each of the four digits to appear bright and continuously illuminated, all four digits should be driven at least once every 16 ms, for a refresh frequency of 60 Hz. In a 60 Hz refresh scheme, the entire display would be refreshed once every 16 ms, and each digit would be illuminated for ¼ of the refresh cycle, or 4 ms. The controller must drive the cathodes with the correct pattern when the corresponding anode signal is driven. To illustrate the process, if AN0 is asserted while CB and CC are asserted, then a "1" will be displayed in digit position 1 (leftmost of the four displays). Then, if AN1 is asserted while CA, CB and CC are asserted, then a "7" will be displayed in digit position 2. If AN0 along with CB and CC are driven for 4 ms, and then A1 along with CA, CB, and CC are driven for 4 ms in an endless succession, the display will show "17" in the first two digits. The figure below shows an example timing diagram for a four-digit controller.



Seven-segment display timing diagram

Counters and Clock Dividers

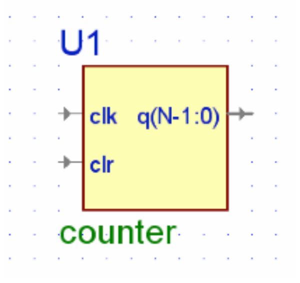
Our 4-digit seven-segment controller will take a clock and four characters (4-bit each) as inputs, and will write the seven-segment control signals as well as the four anode signals to display all four characters simultaneously. The figure below shows a block diagram of a possible implementation of this controller.



A block diagram of a possible implementation of the seg7_driver

Clock Converter

To sequence through the display anodes, you will need to have a clock in your design. The Nexys-2 board has an onboard 50 MHz clock. This 50 MHz clock signal is a square wave with a period of 20 ns. This clock signal is too fast for this application, so you will need to "down convert" that clock for this lab. For example, you can down convert the 50 MHz clock to 1 KHz. can be used to create an N-bit counter, whose block diagram is shown in the figure below.



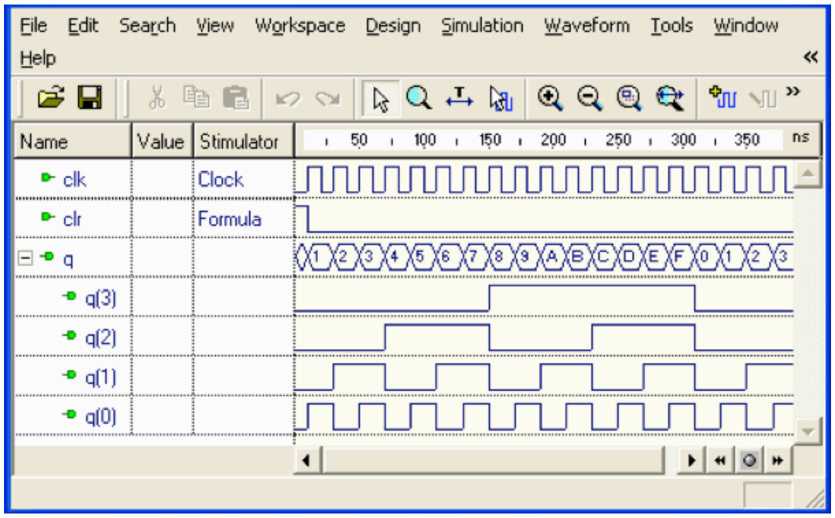
Block Diagram of an N-bit Counter

The following Verilog program can be used to generate this counter. Note that the sensitivity list of the **always** statement contains the phrase **posedge clk** or **posedge clr**. This means that the **if** statement within the **always** block will execute whenever either **clr** or **clk** goes high. If **clr** goes high then the output **q**[**N-1:0**] will go to zero. On the other hand, if **clr** = 0 and **clk** goes high, then the output **q**[**N-1:0**] will be incremented by 1. The default value of the parameter N in this code is 4. A simulation of this 4-bit counter is shown in the figure below the code. Note that this counter counts from 0 to F and then wraps around to 0.

```
// Example N-bit counter

module counter
  #(parameter N = 4)
 (input wire clr ,
  input wire clk ,
  output reg [N-1:0] q
 );

// N-bit counter
always @(posedge clk or posedge clr)
  begin
    if (clr == 1)
      q <= 0;
    else
      q <= q + 1;
  end
endmodule
```


File Edit Search View Workspace Design Simulation Waveform Tools Window
Help
Name
Value
Stimulator
50 100 150 200 250 300 350 ns
clk
Clock
clr
Formula
q
1 2 3 4 5 6 7 8 9 A B C D E F 0 1 2 3
q(3)
q(2)
q(1)
q(0)

Simulation of the 4-bit counter

In these simulation results, note that the output **q**[**0**] is a square wave at half the frequency of the input **clk**. Similarly, the output **q**[**1**] is a square wave at half the frequency of the input **q**[**0**], the output **q**[2] is a square wave at half the frequency of the input **q**[1], and the output **q**[3] is a square wave at half the frequency of the input **q**[2]. Note also that the binary numbers **q**[**3:0**] count from 0000 to 1111.

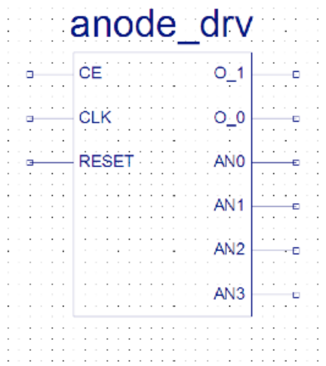
You can instantiate larger counters such as a 24-bit counter, for instance, that would count from 0 to $2^{24} - 1$ by using an instantiation statement such as the following:

```
counter #(.N(24))
  cnt24 (.clr(clr),
         .clk(clk),
         .q(q)
  );
```

Thus, a counter can be used to divide the frequency f of a clock, where the frequency of the output **q**(**i**) is $f_i = f / 2^{i+1}$. This shows one way you can obtain a lower clock frequency by simply using one of the outputs **q**[**i**]. For instance, the output **q**[0] will have a frequency of 25 MHz, the output **q**[17] will have a frequency of 190.73 Hz, and the output **q**[23] will have a frequency of 2.98 Hz.

Anode Driver Unit

Using the clock signal, you need to develop a system that time-multiplexes among signals AN3-AN0 as depicted earlier. This will require using some sequential logic elements. The figure below shows a block, which provides the desired signals described beneath it.



```
Clk: The Clock for this component can be set to 1 KHz
Reset: Resets this block (on logic 1)
CE: Enables this component (on logic 1)
O_1,O_0: binary representation of the activated output.
AN0-AN3: Active low outputs that will activate selected anode.
```

Anode driving block

During each clock period only one of the outputs (AN3-AN0) is activated and O_1, O_0 corresponds to the active signal 00, 01, 10 or 11. The sequence O_1, O_0 can be generated by a 2-bit counter that counts input clock tics. Thus, the output of a 2-bit counter cycles from 00 to 11, incrementing the count with each clock tic. After O_1O_0 = 11 is generated on the fourth clock cycle, the count will "rollover" back to 00 and continue. To generate the individual anode control signals AN0-AN3, you can use a 2-to-4 decoder with O_1, O_0 as the inputs. However, remember that AN0-AN3 need to be active LOW outputs.

Pre-Lab Preparation

Part 1: Reading

Read pages 4-6 in the Nexys2 Board Reference Manual available at http://www.digilentinc.com/Data/Products/NEXYS2/Nexys2_rm.pdf

Part 2: Hex-to-7-Segment Decoder Using Continuous Assignment

1. Design a hex-to-7-segment decoder, with x[3-0] as input.
2. Define Inputs and Outputs. Draw a block diagram of the circuit showing the inputs and outputs for the circuit.
3. Draw a truth-table that shows the 4 inputs and the 7 outputs.
4. Create Verilog code using continuous assignment statements.
5. Functional Simulation. Perform a functional simulation of the circuit to verify that it is working correctly.
6. Create Symbol. Create a symbol for the hex7seg to use in the graphical editor. This creates a symbol file that is a Graphic File and can be viewed and edited by opening it.

Part 3: Hex-to-7-Segment Decoder Using Case Statement

7. Design a hex-to-7-segment decoder using the Verilog case statement.
8. Perform a functional simulation of the circuit. Paste the results in your prelab report.
9. Create Symbol. Create a symbol for the hex7seg to use in the graphical editor.
10. Bring your Verilog codes in a flash drive.

Part 4: Multiple Digits Display

11. Design a clock converter that converts the 50 MHz clock to a 1 KHz clock.
12. Design an anode driver unit that generates the anode driver signals.
13. Design a 4-digit seven-segment controller.

Pre-Lab Report

In your prelab report, include the following:

- Truth Table, circuit schematic, Verilog program, and simulation results for the hex-to-7-segment decoder using continuous assignments, with all possible values of inputs x[3-0].
- Truth Table, circuit schematic, Verilog program, and simulation results for the hex-to-7-segment decoder with case statement, with all possible values of inputs x[3-0].
- Circuit schematic, Verilog program, and simulation results for the anode signals.

Incorrect or incomplete designs and Verilog programs will not receive full credit. **If you have any problems with Verilog syntax and other pre-lab related issues, please resolve them before coming to the lab. Your TA may not be able to help you with these issues during the lab session.**

In-Lab Procedure

Bring flash drives to store your data.

Ask the TA questions regarding any procedures about which you are uncertain.

Implement the complete design (synthesize, map, and Place & Route) of your two designs using the Xilinx ISE tools. Program the FPGA using the bit-stream file which is generated in the process. For checkoff, you will show the TA the following:

1. Show the two designs working on the board.
2. Demonstrate at least one input pattern you have used in the test bench. Show that your simulation results above match the observed waveforms on the DLA.
3. Demonstrate your 4-digit display to your TA with the clock at 1 KHz.

Post-Lab Report

Write up your code, schematics, and lab procedures. Demonstrate the correctness of your designs through your pre-lab simulations and note any differences between what you simulated and how the circuits behaved in the lab.