EET 325
Microprocessors

Drexel University
Engineering Technology

Lab #1 – A Multiplexed, Seven-Segment Hex Decoder in Verilog

Objectives
The objective of this lab exercise is to demonstrate how arbitrary combinational lookup circuits can be specified in Verilog. Such a circuit will be used to translate input from four logic switches into signals to drive a multiplexed 4-digit, 7-segment display using persistence of vision.

Introduction
Seven-segment displays are useful for all kinds of electronics. They're typically used for displaying numerical data in decimal format, but can also be used to display hexadecimal and a few other letters (they do h and L okay, but struggle with X and K.)
Seven bits are used to drive each one – eight, if you include the decimal point. FPGAs do great at translation tasks like this (four bits input to seven-segment-display outputs), since all of this is essentially just seven lookup tables.
For displays with N digits, you might think 7xN control lines are needed, but this isn't the case. Human eyes integrate light intensity over a short time interval, so if four digits are shown repetitively in quick succession, they will look like a single four-digit display that is always illuminated. (I recommend refresh rates well north of 100Hz, for this. Try 1kHz to start.)

Setup:
Open ICEStudio on your computer. Plug your TinyFPGA BX into the breadboard, if it isn't already.

Exercise 1: Wiring and testing the logic switches and 7-segment display

In order to have some data to present to our decoder, we will need to make up logic switches. Connect four logic switches as shown below. (Use a 1kΩ pull-down resistor on the center pin, and connect the right pin to the 3.3V rail.

(These are SPDT – Single-Pole, Double-Throw – switches, which connect the center pin to either the left or right pins. We will use them as SPST switches, connecting or disconnecting the rightmost two pins. In this configuration, we use the center pin as the output; it will be pulled to Ground via a 1kΩ resistor. When the switch is moved to the right, the center pin will be connected to the 3V3 source, making the output pin high.

Create four of these; the left one will be the Most Significant Bit (MSB; the 8s' place).

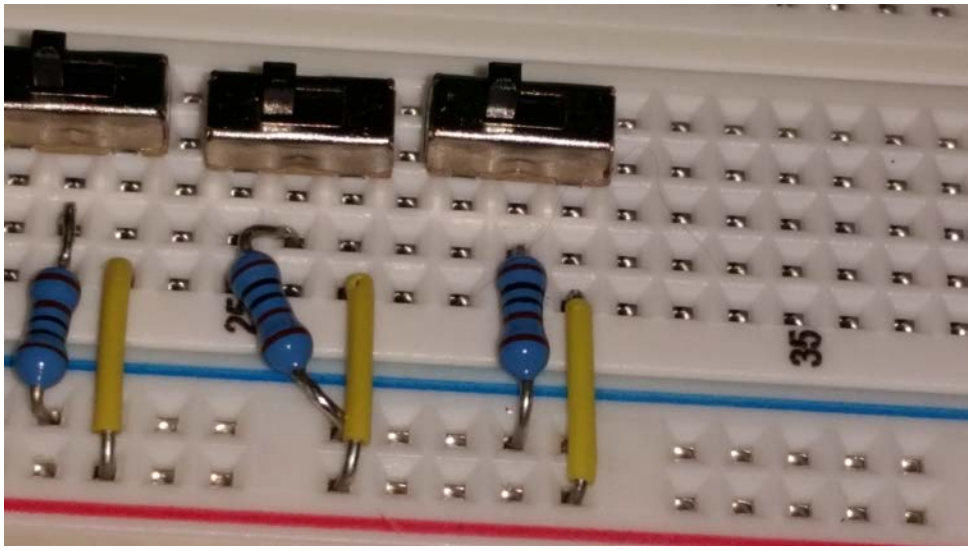


Figure 1. Logic switches. The center pin is output; it is pulled to ground by a 1kΩ (or so) resistor. When the switch is to the right, logic high is connected.

Connect the 3.3V pin on the FPGA to the breadboard positive rail. Connect GND to the negative. Place the 7-segment display on the board (relatively near the FPGA) with the decimal point down. Leave at least two free rows on the top and bottom of the display. (See Figure 2 below.)

Figuring out the LED pinout
Connect a wire to 3.3V via a 330Ω resistor (you can use 1kΩ if you want), and connect another wire to Ground. (Don't touch them together – but if you did, it should put 3.3V across the resistor.)

NOTE: We are using multiplexed, multi-digit displays; these have one cathode (ground) per digit.

Use this pair of wires to check which pins are which:
- If you try a pin combination and nothing lights up, try again.
- If something lights up:
    - You know the pin connected to Ground is the cathode **for that digit.**
    - You know the pin connected to 3.3V via a resistor is the anode **for that segment**.
    - Write this down and keep going until you find all twelve pins (4 cathode, 8 anode).

Note down which pins are the input pins for segments A-G and the decimal point (DP), as well as which pins are the cathode (ground) pins for which segments. You will need this later.

LED segment notation:
(Segment A is at the top; B,C,D,E,F go clockwise around the outside from there. G is the crossbar.)
Include a pinout diagram for this part in your report. (*Laying the ground work like this on a part you or other engineers in your company plan to use can save a lot of time!)*

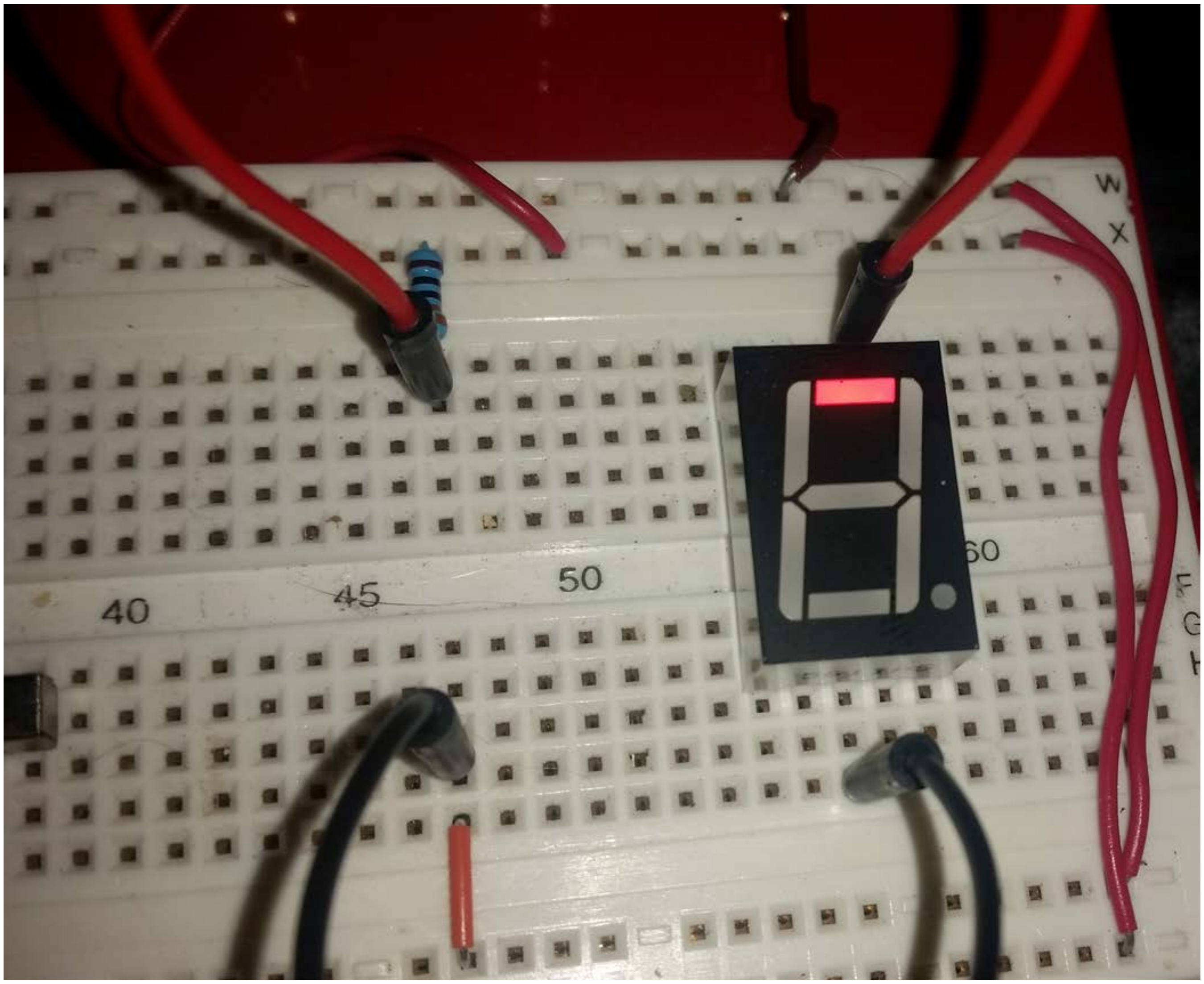


Figure 2. Finding the connections to the seven-segment display.
(Segment A is lit here.) Make sure you use a series resistor, as shown!

Once you figure out which pins are which on the display, wire them (through 330Ω series resistors) to seven different pins on the TinyFPGA BX. (Note down which pins you choose.)

Tip: If you leave the TinyFPGA BX powered up but in the bootloader (breathing LED) state, it might weakly illuminate the LED segments as you connect them, letting you see the order of connection. (Mine did.)

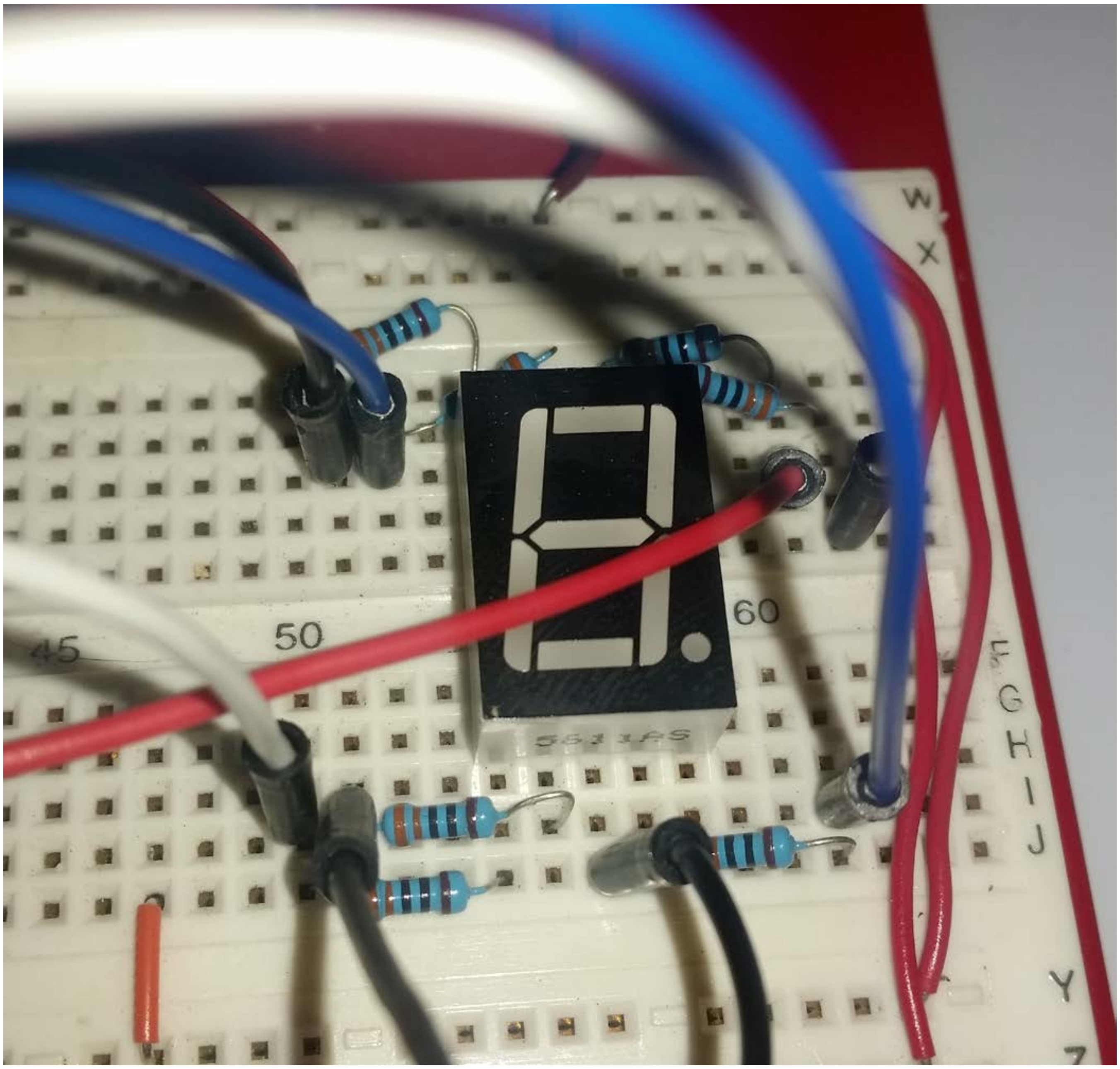


Figure 3. Wiring the seven-segment display

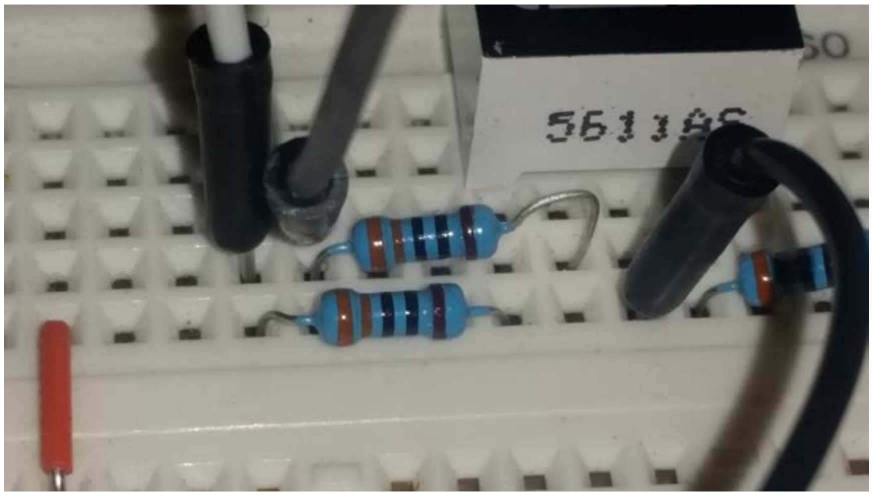


Figure 4. Detail of series current-limiting resistors connected to LED:
(Signals from FPGA go to the resistors; the other end of the resistor goes to the LED.)
I’m using 330Ω resistors here. You can use 1kΩ, but don’t go *lower* than 330.

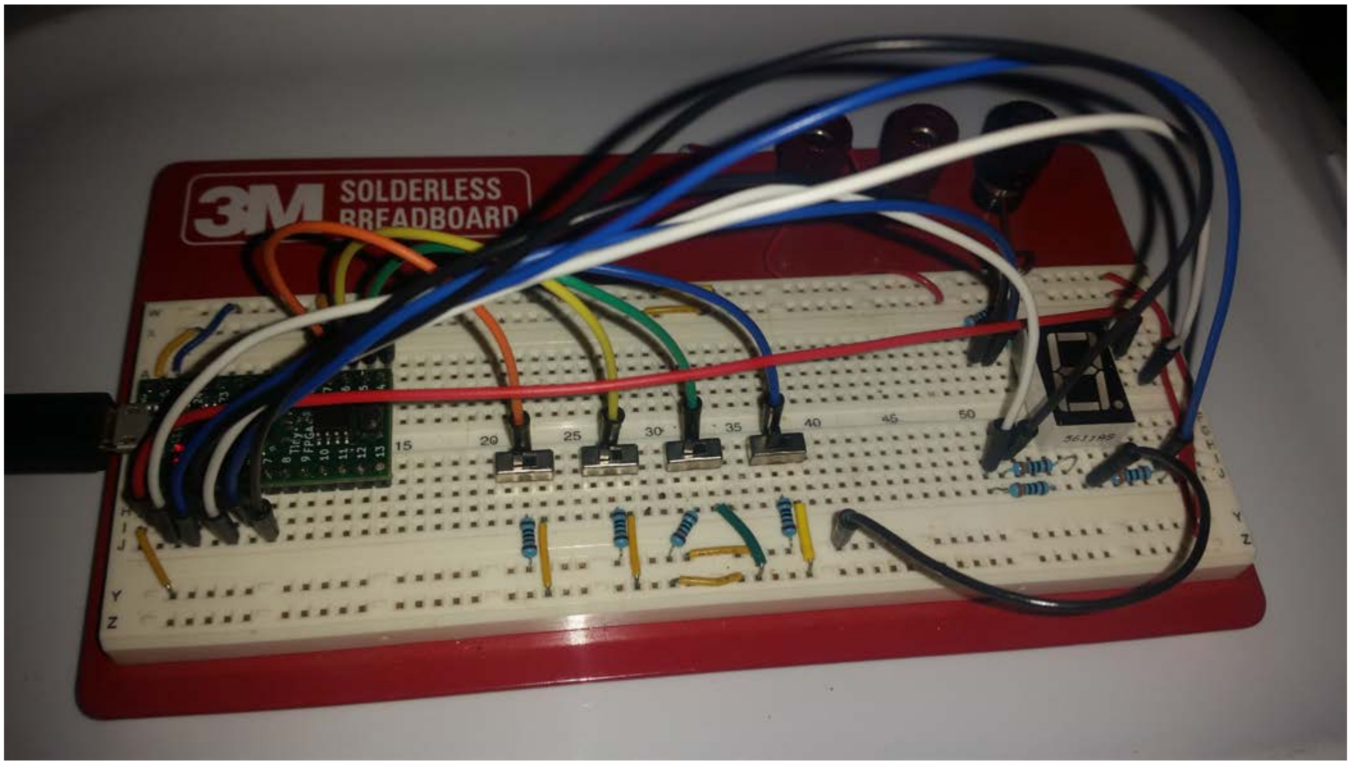
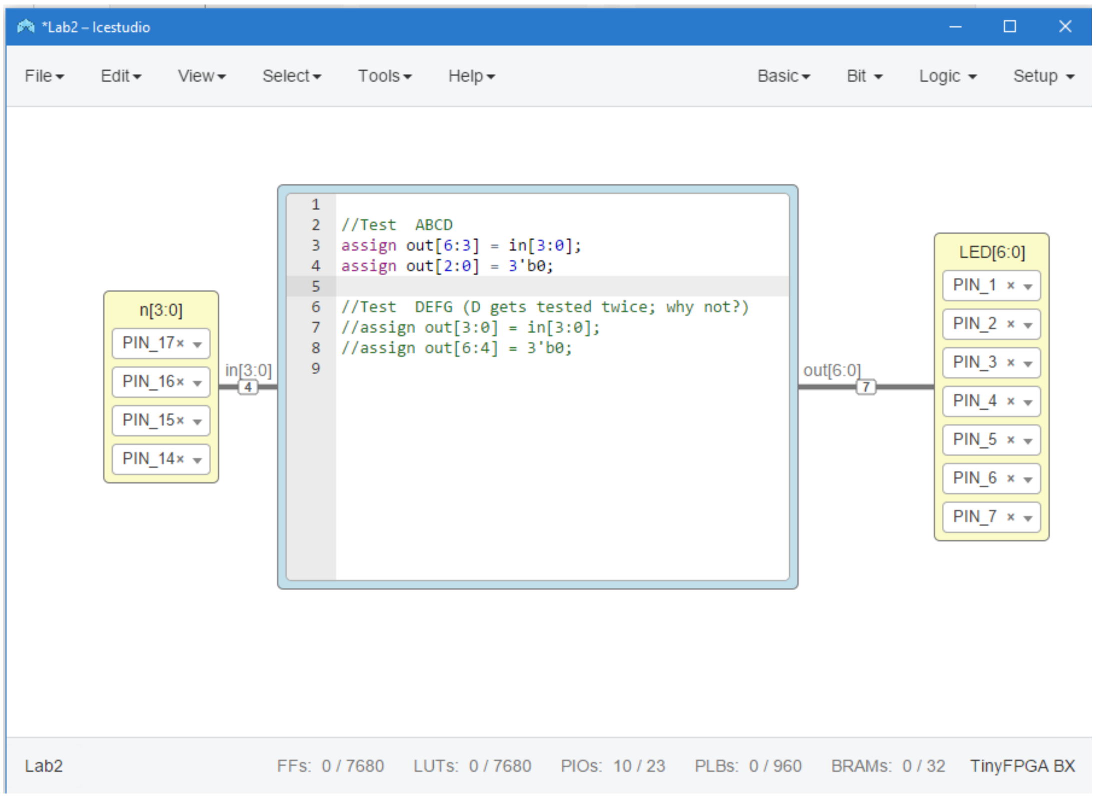


Figure 5. Logic switches and 7-segment LED
(I used 1-7 as A-G and 17-14 as the logic inputs.
Use whatever pins you like – but write down which is which.)

NOTE: The four-digit display has four cathodes – one for each digit.
You will need to locate all four.

Create a sketch in ICEStudio to wire the four logic switches to four of the LED segments as a test. (We have four switches and seven segments to test, so we will have to do this in two stages…)



```

//Test  ABCD
assign out[6:3] = in[3:0];
assign out[2:0] = 3'b0;

//Test  DEFG (D gets tested twice; why not?)
//assign out[3:0] = in[3:0];
//assign out[6:4] = 3'b0;

```

Figure 6. Very simple ICEstudio sketch to test inputs and outputs.
(Ground all four cathodes, for now.)

CHECKPOINT: Each switch controls one segment; you know which wire controls which segment. Switch the comments to the other code segment, to make sure you test all seven segments. (I.E. test ABCD as-is, then move the comments to the other piece of code to test DEFG. Remember to reset your board before uploading the new configuration.)

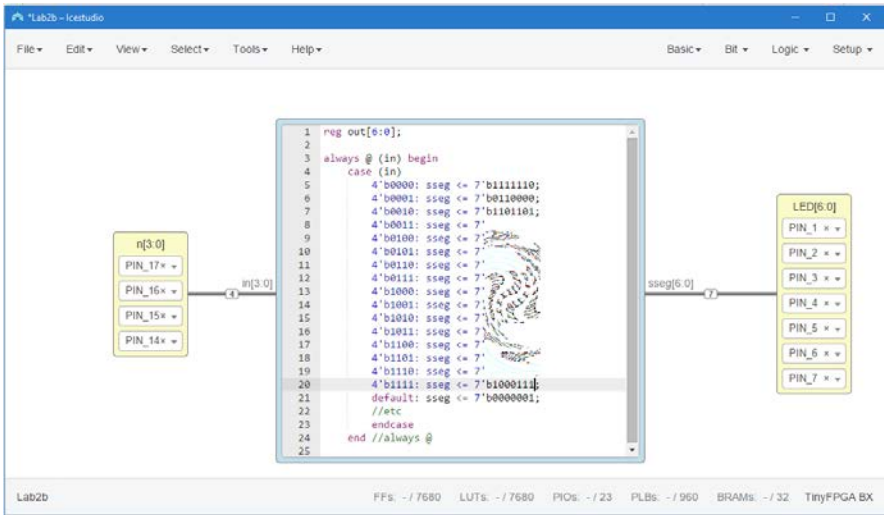
Exercise 2: The Seven-Segment Display: Binary To Digits

Next, we create the seven-segment lookup code to turn a four-bit binary number into a digit.

You should already have four inputs and seven outputs in your sketch. We just need to replace the contents of our code block. (The connection names shown in the figures vary, but don’t really matter.)

For the code, we need to create seven Boolean assign statements that translate the four bits of the input number into the seven bits of the character we want to display.
For example, input 0000 (zero) would produce output 1111110, with all digits except the center bar on.
Input 0001 (one) would produce 0110000 (just segments B and C on.)
We can do this in one big case statement, where we assign all of the bits at once for each case:

```
always @ (num) begin
        case (num)
                4’b0000: assign sseg <= 7’b1111110;
                4’b0001: assign sseg <= 7’b0110000;
                //etc
                //etc through 4’b1111 (16 cases total)
                endcase
        end //always @
```



```
reg out[6:0];

always @ (in) begin
    case (in)
        4'b0000: sseg <= 7'b1111110;
        4'b0001: sseg <= 7'b0110000;
        4'b0010: sseg <= 7'b1101101;
        4'b0011: sseg <= 7'
        4'b0100: sseg <= 7'
        4'b0101: sseg <= 7'
        4'b0110: sseg <= 7'
        4'b0111: sseg <= 7'
        4'b1000: sseg <= 7'
        4'b1001: sseg <= 7'
        4'b1010: sseg <= 7'
        4'b1011: sseg <= 7'
        4'b1100: sseg <= 7'
        4'b1101: sseg <= 7'
        4'b1110: sseg <= 7'
        4'b1111: sseg <= 7'b1000111;
        default: sseg <= 7'b0000001;
        //etc
        endcase
    end //always @
```

Figure 7. Verilog code to implement a seven-segment decoder. (Figure out the constants for “3” through “E” yourself, based on the wiring scheme you are using for the seven segments. Remember to use lowercase “b” and lowercase “d”.)

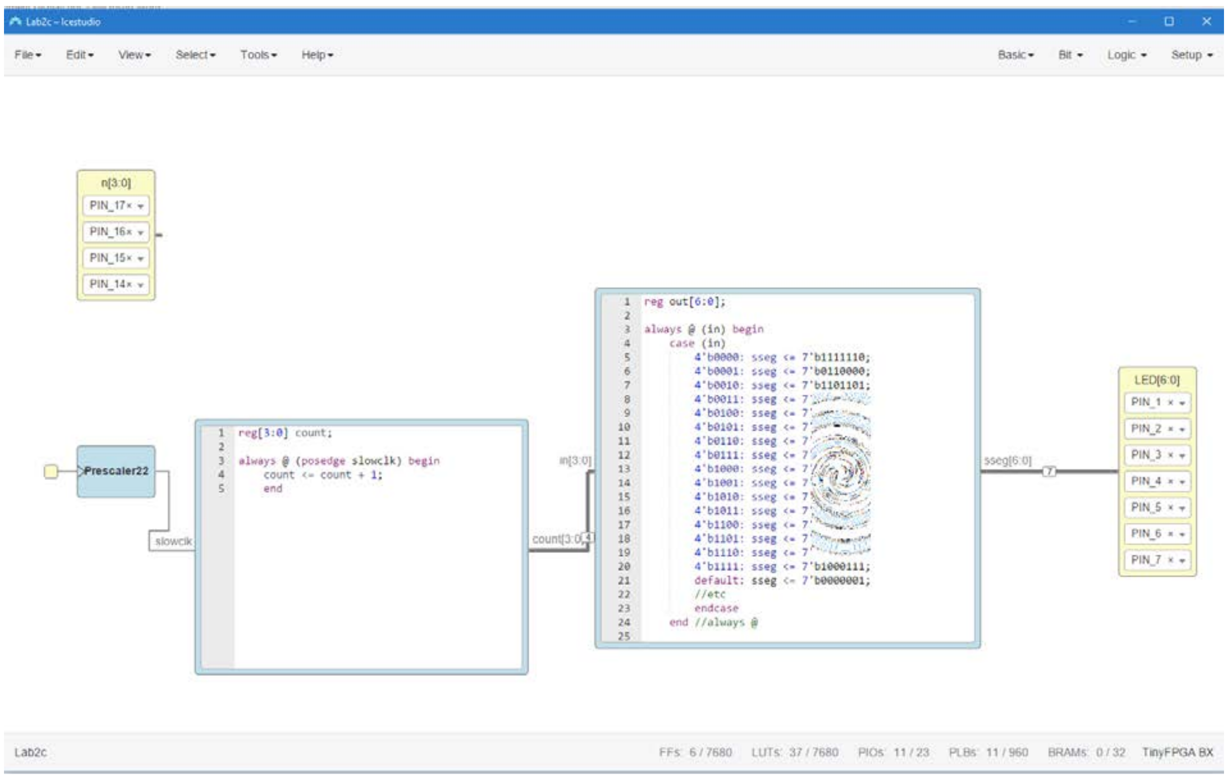
CHECKPOINT: The logic switches control the LED digit as the 8s, 4s, 2s, and 1s place; you can enter any 4-digit binary number and see it displayed as a hex digit.
***Check all combinations.* (It’s possible to do this with just 15 switch flips, if you plan ahead.)**

Exercise 3: A stopwatch

Instead of driving the seven-segment decoder from logic switches, let's run it from the (divided) clock.

We'll need a counter and a clock divider – but why write the clock divider when we can steal a copy? (Open up the "One LED blink" example and steal the clock divider from there. **Code reuse!**)

Save your new sketch as Lab1_Ex3 or similar.



```
reg[3:0] count;

always @ (posedge slowclk) begin
    count <= count + 1;
    end
```

```
reg out[6:0];

always @ (in) begin
    case (in)
        4'b0000: sseg <= 7'b1111110;
        4'b0001: sseg <= 7'b0110000;
        4'b0010: sseg <= 7'b1101101;
        4'b0011: sseg <= 7'[illegible]
        4'b0100: sseg <= 7'[illegible]
        4'b0101: sseg <= 7'[illegible]
        4'b0110: sseg <= 7'[illegible]
        4'b0111: sseg <= 7'[illegible]
        4'b1000: sseg <= 7'[illegible]
        4'b1001: sseg <= 7'[illegible]
        4'b1010: sseg <= 7'[illegible]
        4'b1011: sseg <= 7'[illegible]
        4'b1100: sseg <= 7'[illegible]
        4'b1101: sseg <= 7'[illegible]
        4'b1110: sseg <= 7'[illegible]
        4'b1111: sseg <= 7'b1000111;
        default: sseg <= 7'b0000001;
        //etc
        endcase
    end //always @
```

Figure 8. Code for a "stopwatch" (automatic 3Hz-or-so hex counter)

CHECKPOINT: Digits count 0 through F and display correctly.

Save your work.

Exercise 4: A multi-digit stopwatch

Finally, we want to create a multi-digit stopwatch / counter. **Save your work as Lab1_Ex4 or similar.**

To do this with only seven segment wires, we will time-multiplex them, displaying each digit by itself very briefly and moving to the next digit. If done quickly enough, this will look to the human eye like all four digits are lit up, since our eyes integrate or low-pass filter the light intensities they receive.
Cycle the digits at a few hundred Hz, and they will look solid.

So, we need to switch between the four displays, so that each can display an independent digit.
This works by lighting up the first display with the first digit for a short period of time (~milliseconds), then the second digit, then the third, then the fourth. Each time we switch to a new digit, we send the correct code for that digit to the seven segment wires.

Do this fast enough (couple hundred Hz or more), and it will look as if all four are constantly on. This is known as the "persistence of vision" effect, and is used in many different types of visual displays, especially with LEDs.

Create a new code block with a one-bit "clk" input, and outputs of digit[3:0] and d[3:0].
This represents a controller block, which selects which digit to display and also decides on the 4-bit binary number for that digit. We will create a 32-bit counter in this block, and display parts of it.

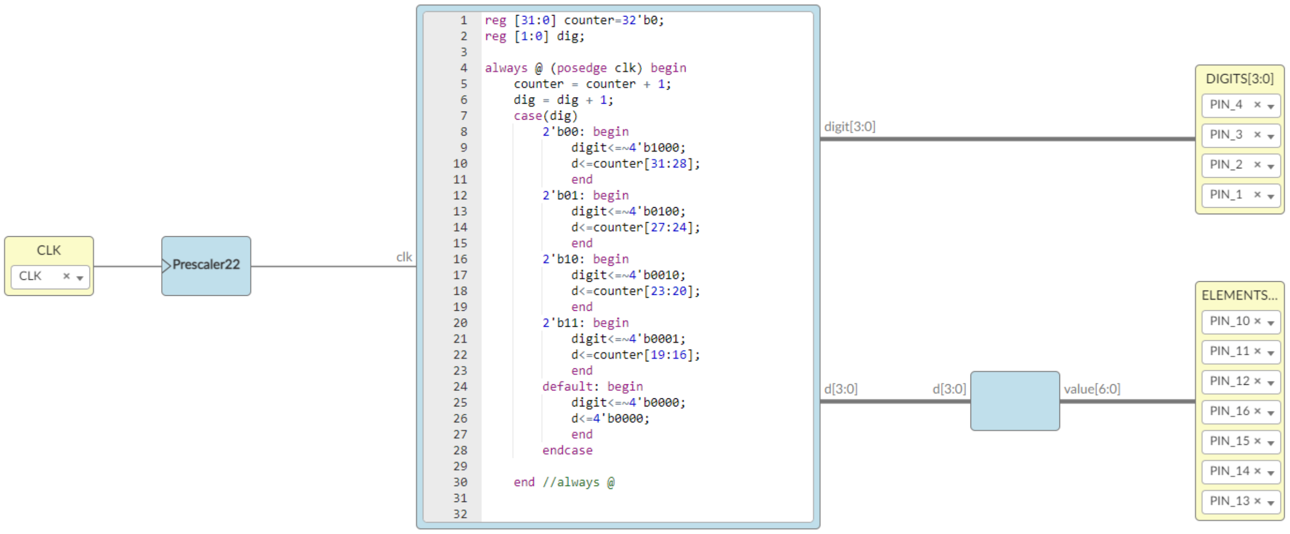
Connect the output d[3:0] of this new block to the input of your old code block.
(Replace the connection from the old counter block.

Next, create an output block: digits[3:0]
This creates four outputs. Assign these to four unused FPGA pins, and connect it to the new digit[3:0] output that you created in the newer (currently empty) code block. **Wire these pins to the cathodes.**

Finally, write the code.
Create a 32-bit register called counter: **reg [31:0] counter=32'b0;**
Create a 2-bit register called dig: **reg [1:0] dig=2'b0;**
Create an always block that (on the rising edge of the clock):

- Increments <counter> by one;
- Increments <dig> by one;
- Uses a case statement to do the following (try to write it yourself; look at Figure 9 if stuck):
    - If <dig> is 00:
        - Set <digit> to 0111; (First digit on / low; others off / high)
        - Set <d> to counter[31:28]; //Highest four bits of register
    - If <dig> is 01:
        - Set <digit> to 1011; (Second digit on / low; others off / high)
        - Set <d> to counter[27:24]; //Second-highest four bits of register
    - If <dig> is 10:
        - Set <digit> to 1101; (Third digit on / low; others off / high)
        - Set <d> to counter[23:20]; //Third-highest four bits of register

- If <dig> is 00:
    - Set <digit> to 1110;   (Fourth digit on / low; others off / high)
    - Set <d> to counter[19:16]; //Fourth-highest four bits of register



```
reg [31:0] counter=32'b0;
reg [1:0] dig;

always @ (posedge clk) begin
    counter = counter + 1;
    dig = dig + 1;
    case(dig)
        2'b00: begin
            digit<=~4'b1000;
            d<=counter[31:28];
            end
        2'b01: begin
            digit<=~4'b0100;
            d<=counter[27:24];
            end
        2'b10: begin
            digit<=~4'b0010;
            d<=counter[23:20];
            end
        2'b11: begin
            digit<=~4'b0001;
            d<=counter[19:16];
            end
        default: begin
            digit<=~4'b0000;
            d<=4'b0000;
            end
        endcase

    end //always @
```

Figure 9. The control code for the display multiplexer.
(The existing seven-segment-decoder code is in the blue box at lower right.)

Reset your board, compile, and test it.

Checkpoint: Numbers count up from 0000-FFFF (or 0000-9999, if doing the decimal option).

PLEASE LEAVE YOUR PROJECT ASSEMBLED; we will build on it later!
(Put your group members' names on a note attached to your project.)
PLEASE SAVE YOUR FILES – they will be very useful later!

Discussion

In your lab report, please make sure you cover the following:

- What equipment and software did you use? (Just a few sentences is fine; the idea is you want other engineers to be able to replicate and build on your work.)

- What did you set out to make the board do? How? Was it successful? If so, how does it work? If not, what do you think went wrong, and what could be done to address this?
(Contact me for help if you're having problems with any lab. ICEStudio is still experimental, so we may have to develop workarounds if we run into bugs.)

- (Make sure to include any code that you create or modify, and note where it goes in the project.)

- Include a pinout of the four-digit, 7-segment LED. Mark pins and segments A through G and the decimal point, as well as the cathodes (common wires) for each digit.

- Include a schematic of your project (four logic switches, including pull-downs; LEDs and series resistors, etc.) Name pins used (you have a lot of freedom here, but make sure your schematic matches what pins you use in your code.) MultiSim could help with schematic creation.

- Without using software to calculate it, how many 4-LUTs (4-input lookup tables) do you think are needed to implement a seven-segment decoder? How many more are needed for the (single-digit) stopwatch? How many for the whole design?
Compare your guesses to what IceStudio calculates.

END OF LAB 1