Learning Objectives

1. Using the 7-segment displays on the Basys3 board
2. Converting from unsigned binary to BCD

Work You Need To Do

- **Part 1**: Write some modules in Verilog.
  - BCDToLED: Decode a binary coded decimal to the 7-segments needed to display it.
  - DecimalDigitDecoder: Decode a binary number into its decimal digits.
    * Comparator: Determine if a binary number is greater than decimal ten.
    * CircuitA: Subtract 10_{10} from a binary input that is ten or greater.

- **Part 2**: Use these designs to program the FPGA in Vivado.

You have the option of either working in pairs or individually during the lab. **If you work in pairs, only one of you should upload to git, and please make sure to fill the partners.txt file.**

- Carefully go through this handout and make sure you read and follow all the steps.
- Perform the following to get the **FPGALab1** directory

  `git fetch honors_release`
  `git merge honors_release/FPGALab1`

- The contents of the directory are listed below.
  - **src** directory: Contains the XDC file and test benches. Use this folder for uploading your Verilog code.

- **Don’t forget to show the instructor a demo of your design before you leave!**
Introduction

Figure 1 above shows the Basys3 board. Notice the four 7-segment displays in the lower left corner; we will be using one of them in today’s lab. Our goal in this lab is to display the 2-digit decimal equivalent of a 4-bit binary number using the right-most 7-segment display (for the one’s decimal place) and an LED (for the ten’s decimal place). Our input will be coming from 4 switches. The handout will guide you through the different steps required to achieve our goal. The lab will be split into two parts:

• **Part 1** will involve writing the Verilog code for some modules.

• **Part 2** will involve implementing the design on Vivado, joining the modules you wrote together, and downloading onto the FPGA.

In order to achieve our goal we will design the following modules:

1. A “Binary Code Decimal (BCD) \(^1\) to 7-Segment Display Decoder” module that will be used to display the decimal equivalent of a BCD number (0-9) on a 7-segment display.

2. A “Decimal Digit Decoder” module that should convert a 4-bit binary number into its 2-digit decimal equivalent by using a 5-bit output where the most significant bit (MSB) represents the ten’s decimal place and the lower 4 bits are the BCD representation of the one’s decimal place.

3. A top-level module that will utilize the two modules to display the 2-digit decimal equivalent of a 4-bit binary number on a 7-segment display and an LED.

**Example:** We start with the binary number: 1011\(_2\) (11\(_{10}\)), we generate from it the output 10001 where the most significant bit 1, corresponding to the ten’s decimal place, will drive an LED and the lower bits 0001, corresponding to the one’s decimal place (1), will be represented on the 7-segment display using the BCD to 7-segment display decoder.

---

\(^1\)For more information on Binary Coded Decimal refer to the following link: [https://en.wikipedia.org/wiki/Binary-coded_decimal#Basics](https://en.wikipedia.org/wiki/Binary-coded_decimal#Basics)
Part 1 Preparing the Verilog Modules

1-1 BCD to 7-segment Display Decoder

A 7-segment display consists of seven segments, labeled a to g which can be used to display a character (see figure on the right). If we want to display a binary coded decimal (BCD) using a 4-bit input, a BCD to 7-segment decoder is required.

1-1.1 Design a BCD to 7-segment display decoder module that has the following input and output ports:

- **Input:** \( x[3:0] \): a 4-bit binary number between 0 and 9.
- **Outputs:**
  - \( \text{seg}[6:0] \): a 7-bit output that will be used to drive the 7-segment display, where \( \text{seg}[0] \) corresponds to ‘a’ in the truth table, and \( \text{seg}[6] \) corresponds to ‘g’ in the truth table.
  - \( \text{an}[3:0] \): a 4-bit output that controls which display is being used. \( \text{an}[0] \) corresponds to the right-most display. Driving a display with 0 will turn it on, 1 will turn it off.

You already wrote the equations for this in your Karnaugh map homework. Now we will put those expressions into a Verilog module.

Note that for the anode output, we want to enable the right-most display. The **an output is the common anode, which means external power is provided to all displays together and to enable a specific one, you have to ground its anode.**

```verilog
module BCDToLED(
    input [3:0] x, // binary input
    output [6:0] seg, // segments
    output [3:0] an // display specific anodes
);

    assign seg[_] =
    assign seg[_] =
    assign seg[_] =
    assign seg[_] =
    assign seg[_] =
    assign seg[_] =
    assign seg[_] =

    assign an[3:0] = 4'b_ _ _ _ ;

endmodule
```
1-2 Binary to Decimal Decoder

Now we will build the Decimal Digit Decoder in parts.

1-2.1 First, let's build a Comparator module that determines if a binary input is $10_{10}$ (decimal ten) or greater. This requires one expression that we can build using a Karnaugh map. We have the following input and output:

- Input: $v$: a 4-bit binary input that is any value between 0 and 15.
- Output: $z$: a 1-bit boolean output that is 1 when the input was ten or greater.

Find an expression so that $z$ outputs 1 when $v$’s binary value is decimal ten or greater.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Expression:

```verilog
module Comparator(
    input [3:0] v, // binary input
    output z       // outputs 1 if v was decimal ten or greater
);

assign z =

endmodule
```
1-2.1 Now we will build a CircuitA module that subtracts out the decimal ten from the binary input, so we can get the binary value of the first decimal digit (e.g. if we had 14, we want CircuitA to give us 4 in binary). $m[0]$ and $m[3]$ are done for you.

- Input: $v$: a 3-bit binary input. We will assume the 4th bit was 1.
- Output: $m$: a 4-bit boolean output that is a value between 0 and 5.

<table>
<thead>
<tr>
<th>$v[3:0]$</th>
<th>$m[3:0]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1010</td>
<td>0000</td>
</tr>
<tr>
<td>1011</td>
<td>0001</td>
</tr>
<tr>
<td>1100</td>
<td>0010</td>
</tr>
<tr>
<td>1101</td>
<td>0011</td>
</tr>
<tr>
<td>1110</td>
<td>0100</td>
</tr>
<tr>
<td>1111</td>
<td>0101</td>
</tr>
</tbody>
</table>

Karnaugh Map for $m[1]$

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Karnaugh Map for $m[2]$

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

module CircuitA(
    input [2:0] v,
    output [3:0] m
);

assign m[0] = v[0]; // Notice the LSB is the same for v and m
assign m[1] =
assign m[2] =
assign m[3] = 0; // Notice the MSB is 0 for all m
endmodule
1-3 Decimal Digit Decoder

Now we can put this all together into a Decimal Digit Decoder. The module will take a 4-bit binary number as input and generate a 5-bit output \( \{ z, m[3:0] \} \), where the most significant bit represents the ten’s digit of the decimal equivalent of the input, and the lower 4 bits represent the BCD representation of the one’s digit of that same number.

![Decimal Digit Decoder Schematic and Truth Table](image)

**Figure 2. Decimal Digit Decoder Schematic and Truth Table**

1-3.1 Put the pieces you’ve designed together to create the Decimal Digit Decoder module with the following ports: *We will use the mux library from class for this.*

- **Input:** \( v[3:0] \): a 4-bit binary number
- **Outputs:**
  - \( z \): a 1-bit output representing the ten’s digit of the decimal equivalent of the input
  - \( m[3:0] \): a 4-bit BCD representation of the decimal’s right-most digit (0-9).
module DecimalDigitDecoder(
    input [3:0] v,
    output z,
    output [3:0] m
);

// Declare any internal wires here.
wire

// module Comparator(input [3:0] v, output z);
Comparator

// module CircuitA(input [2:0] v, output [3:0] m);
CircuitA

mux2v

endmodule
Part 2 Using Vivado and the Basys3 Board

Important Reminders

- To open Vivado: open a terminal window then type in the following commands:

```plaintext
module load xilinx
vivado
```

- To avoid having to load the Xilinx module every time, you can add `module load xilinx` to your `~/.bashrc` file and it will be applied automatically whenever you log on. This way you can open vivado by typing `vivado` in the terminal window.

- Command for remote access: `ssh -Y netid@remlnx.ews.illinois.edu`. Remember: X-forwarding should be enabled on your machine and you cannot download to the board remotely.

What You Should Submit

Due at the end of the lab session:

- All the Verilog modules: `BCDToLED.v`, `DecimalDigitDecoder.v`, and `top_level.v`.

- The modified `Basys3_Master.xdc` file

- The `partners.txt` file if you worked in pairs

2-1 Creating a BCD to 7-Segment Display Decoder

The Basys3 board contains a four-digit common anode seven-segment LED display module. Each of the four digits within a module is composed of seven segments arranged in a pattern shown in Figure 3 below, with an LED embedded in each segment. Segment LEDs can be individually illuminated, so any one of 128 patterns can be displayed on a digit by illuminating certain LED segments and leaving the others dark. Of these 128 possible patterns, the ten corresponding to the decimal digits are the most useful.

Figure 3 below shows how the 7-segment display is connected to the Artix-7 on the Basys3. The anodes of the seven LEDs forming each digit are tied together into one “common anode” circuit node, but the LED cathodes remain separate. The common anode signals are available as four “digit enable” input signals to the 4-digit display. The cathodes of similar segments on all four displays are connected into seven circuit nodes labeled CA through CG (so, for example, the four “D” cathodes from the four digits are grouped together into a single circuit node called “CD”). These seven cathode signals are available as inputs to the 4-digit display. This signal connection scheme creates a multiplexed display, where the cathode signals are common to all digits but they can only illuminate the segments of the digit whose corresponding anode signal is asserted.

A scanning display controller circuit can be used to show a four-digit number on this display. This circuit drives the anode signals and corresponding cathode patterns of each digit in a repeating, continuous succession, at an update rate that is faster than the human eye can detect. If the update or “refresh” rate is slowed to around 45 hertz, most people will begin to see the display flicker. You will design and use the scanning circuit in the next lab. For more information on the seven segment display and other I/O, refer to the Basys3 Reference Manual (available on Piazza).
Figure 3. Pin connections for 7-segment display [Reference - Basys3 Reference Manual]
The following steps will guide you through the process of synthesizing a module that uses the BCD
to 7-segment display decoder that you prepared in Part 1.

2-1.1 Open Vivado and create a blank project called lab1. You can create the project in the
FPGALab1 directory like last time.

2-1.2 Add your BCDToLED.v module with 4-bit data input x[3:0], anode enable output signals
an[3:0], and 7-bit output seg[6:0].

   **Hint:** Click the Green Plus button on the Add Sources on the New Project window.
   Click Add File. Select the BCDToLED.v file. Click Next.

2-1.3 Add the board related master XDC file to the project.

   **Hint:** Click the Green Plus button on the Add Constraints on the New Project win-
dow. Click Add File. Select the Basys3_Master.xdc file. Click Next. (You can find the file
under the FPGALab1/src directory on git).

2-1.4 Select Basys3 from Boards. Click Next. Click Finish.

2-1.5 Double click on the source file in the Sources tab and fill out the Verilog of the module. This
is where you should type the code you wrote for the BCDToLED module.

2-1.6 After filling the module code, click on Open Elaborated Design under RTL Analysis in
the Flow Navigator. This will check your Verilog syntax and generate a netlist out
of it. If you have any syntax errors, fix them and try this step again before you continue.
   (Note: You should always perform this step after saving a Verilog source file).

2-1.7 Once you successfully generate the netlist, double click the XDC file in the Sources tab, under
the Constraints folder, to open it. Edit the file to enable sw[0] through sw[3] and rename
them from sw to x. Enable seg[0] through seg[6] and an[0] through an[3]. Notice seg[0]
through seg[6] correspond to CA,CE,CC,CD,CE,CF,CG from the image on the previous page.
By enable, we mean uncomment both lines for each pins we are using.

   **Note:** The inputs and outputs of BCDToLED.v is determined by the constraint file. We
want to make sure they are compatible.

2-1.8 Add and simulate the testbench (BCDToLED_tb.v) provided in the git FPGALab1/src directory
for 180 ns. Set the radix of x_in as hexadecimal, seg_out and an_out as binary and k
as decimal. Expand the seg_out and an_out signal so that each bit’s waveform is visible.
Confirm your waveform with an instructor.

   **Hint:** You should add the test bench by clicking on the Add Sources button in the
Project Manager section of the Flow Navigator (on the left). Select the Add or create
simulation sources option and click Next. Click the Add Files button (or Green Plus
button > Add Files), this will open a File Browser. Select the appropriate testbench in your
git directory and click OK. Make sure the Copy sources into project checkbox is checked, then
click Finish. Don’t forget to set the simulation time to 180ns in the Simulation
Settings window.
2-1.9 Generate the bitstream, download it to the Basys3 board, and check the functionality. You should be able to show the numbers 0 through 9 on the right-most display.

Have an instructor check that the implemented design is functioning correctly before moving on!

2-2 Creating a 4-bit Binary to 2-Digit Decimal Decoder

The following steps will guide you through the process of adding a module that uses the Decimal Digit decoder that you prepared in Part 1-3. This module will only be simulated, we will not synthesize and download it on the FPGA yet.

2-2.1 Add DecimalDigitDecoder.v to the project.

**Hint:** Click the Add Sources button in the Project Manager. Chose the Add or create design sources option, click Next and then click the Add File button. Select DecimalDigitDecoder.v. Click Next.

2-2.2 Open the verilog source file and write the DecimalDigitDecoder module you wrote previously. Include the modules for CircuitA and Comparator in this file as well. Then add the provided mux_lib.v using the Add Existing Source option.

2-2.3 At this point, DecimalDigitDecoder.v should be the new top level module. (You can verify this by searching for the bolded name in the Sources tab.) If it isn’t, right click on it and click Set as Top. Once it is the top level module, generate the netlist to be able to simulate the module. Make sure you correct any syntax errors before you continue.

2-2.4 Add and simulate the testbench (DecimalDigitDecoder_tb.v) provided in the git FPGALab1/src directory for 180 ns. Set the radix of v_in and k as decimal and m_out and z_out as binary. Expand the m_out signal so that each bit’s waveform is visible.

**Hint:** You should also set the test bench that you wish to simulate as the Top Module in the Simulation Sources. Either that, or place it in a separate Simulation Set.

Show an instructor the schematic or waveform to confirm you have written it correctly.

2-3 Creating the 4-bit Binary to LED Module

We will now create a top-level module which takes as input a 4-bit binary number and generates two outputs: 1) A 1-bit output representing the ten’s digit of the decimal equivalent of the input, and 2) A 7-bit output that will be used to represent the one’s digit of the decimal equivalent of the input on the 7-segment display. Figure 4 below represents a block diagram of the module.

We will not simulate this module.

2-3.1 Create a new source file named top_level.v which takes as input sw[3:0] and outputs led and seg[6:0] and an[3:0]. Fill the module by instantiating instances of the two modules we designed in Parts 2-1 (BCDToLED) and 2-2 (DecimalDigitDecoder) and connecting them as shown in the block diagram then generate the netlist. **Note:** Remember, to set the correct module as top level.
2-3.2 Edit the XDC file to connect the inputs and outputs to the proper pins. This including naming back the switch ports we renamed previously and enabling an LED pin. You can use any LED you want and give it a port name of your choosing.

2-3.3 Generate the bitstream, download it to the Basys3 board, and check the functionality.

   **Have an instructor check that your implemented design is functioning correctly before you leave.**

   **Don’t forget to upload your verilog files to git before you leave!**