Lab 2: Decoders and Multiplexers

Introduction

Decoders and multiplexers are important combinational circuits in many digital logic designs. Decoders convert n inputs to a maximum of unique 2^n outputs. A special case is the binary coded decimal (BCD) to 7-segment decoder, where a four-bit decimal digit (represented in BCD) is decoded into the corresponding seven-segment code, which is then used as an input to the physical seven-segment display (Figure 1). In this lab an understanding of both multiplexers and how to wire a DIP switch will be fostered. These skills will be used in tandem to create a multiplexer circuit and controlled by external input stimuli.

A simple computer has several main blocks, e.g.:

- Arithmetic Logic Unit (ALU): Performs arithmetic operations on numbers.
- Memory: Where the program is stored.
- Multiplexers: Select which piece of information to be passed on.
- Decoders: Determine, based on the input, whether to read from memory or input/output lines.
- Computer Control Unit(CCU): Outputs the control signals that direct the operation of the rest of the computer.

Even though we are not building a computer, this information give you some perspective on the different components that you will be building and what they may be used for.

In this lab we will focus on the multiplexer that chooses either a reset address (`Rst_Addr`), program counter (`PC`), memory address register (`MAR`), or index register X (`IRX`). These signals are used to determine the information required to enter the arithmetic logic unit (ALU) component of the computer.

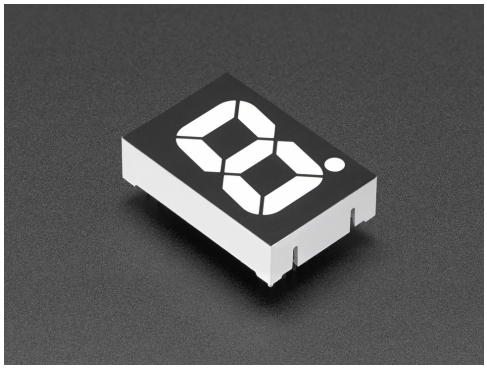


Figure 1: 7-Segment Display

1 Prelab

1.1. Read the lab and the material provided in the supplement.

1.2. Place two of the 7-segment displays and the Spartan 7 FPGA on your breadboard. How should the display be connected to the FPGA? What connections need to be made? Are resistors needed for current management? (**Note:** Consult Spartan pinout diagram in Figure 4) Make the necessary connections and wire up 2 seven segment displays on your breadboard.

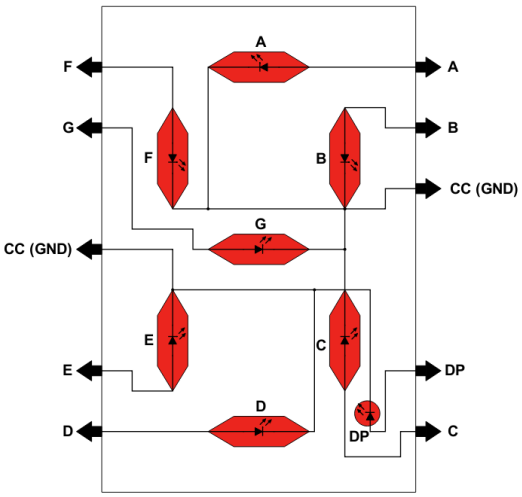


Figure 2: 7-Segment Display Diagram

1.3. Fill in the truth table for the BCD to 7-segment decoder shown in Table 1.
-Ex. If the input is **0011**, LEDs A, B, C, D, and G should be on, while LEDs F and E should be off ("1" is LED on, and "0" is LED off). See Figure 2 for reference.

For inputs `0xA` through `0xF`, naturally they don't correspond to any number in the decimal range, therefore output the corresponding hex value instead, i.e., for `0xA` the display should show the letter A. (all segments but D)

1.4. Design the multiplexer shown in Figure 5 with `Addr_Sel` as the select signal, and `Rst_Addr` (we will use address $0xFF$), `PC`, `MAR`, and `IRX` as 8-bit input signals.

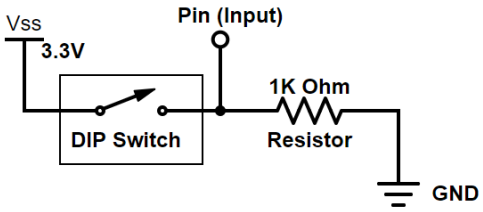
1.5. Design a Verilog program to implement the decoder from Table 1.

2 Lab

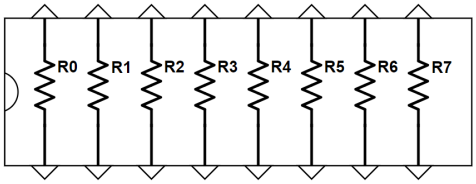
2.1. Set the protoboard's variable positive voltage (+) to 3.5V using a voltmeter. **(Do NOT use more than 4 V.)** Verify that you are using the correct power source (row) for the lab, as using 5V or more will damage the Spartan 7 input pins.

2.2. Connect the protoboard's ground voltage to the Spartan 7 **GND** pin. Be sure to use the Protoboard 3.5V as Vss, and Do NOT connect the Spartan 7 Vu pin to Vss. **Why do we keep Vu and Vss Separate?**

2.3. Place a block of 8 DIP switches on your breadboard. Wire each pin according to the circuit show in Figure 3a. Use the through-hole 1KΩ resistors, not the DIP package 220Ω resistors from your lab kit, shown in Figure 3b. These DIP Resistors can be used to limit the current through LEDs (e.g. 7-segment displays). **Note:** adding these resistors is not necessary using Spartan 7 board, due to the built-in pin resistors. Each pin is already wired with a 240Ω resistor, limiting output current to around 20mA.



(a) DIP Switch Circuit



(b) DIP 220Ω Resistor (4116R) Layout

Figure 3

2.4. Normally, connecting an LED to Vu and Gnd will result in the LED burning out. **Why does this happen?**

2.5. In order to use the Spartan 7 I/O (In/Out) pins, you will need to assign the desired pins to correspond to your code's variables in Vivado. Familiarize yourself with the orientation and numbering convention of the board, shown in Figure 4. Consult Tables 3 and 4 for S7 features as well as pin numbering and names.

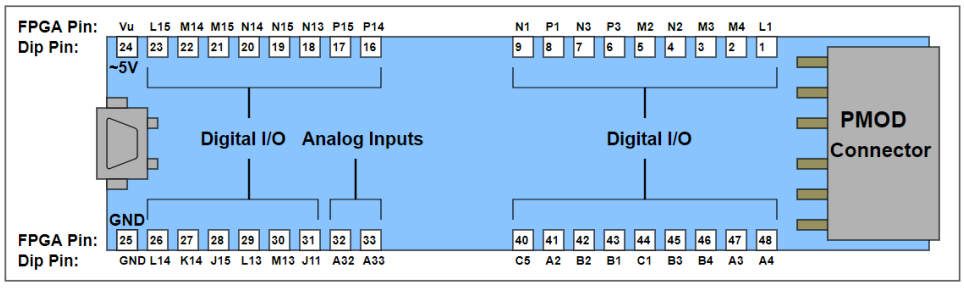


Figure 4: CMOD-Spartan 7 Board Oreintation and Pin Numbering

Table 1: Truth Table for Hexadecimal to 7-Segment Decoder

Digit	Binary	A	B	C	D	E	F	G
0	0000							
1	0001							
2	0010							
3	0011	1	1	1	1	0	0	1
4	0100							
5	0101							
6	0110							
7	0111							
8	1000							
9	1001							
A	1010							
B	1011							
C	1100							
D	1101							
E	1110							
F	1111							

2.6. Now that the hardware is setup, design the binary coded decimal (BCD) to 7-segment decoder and test it using different inputs from the dip switches.

2.7. Implement the multiplexer program that you made in the Prelab, as shown in Figure 5. To test the multiplexer we need to hard code in Verilog `Rst_Addr` to `0xFF`, `PC` to `0x0A`, and `MAR` to `0x10`. Connect `IRX` to the 8 DIP switches, and `Mem_Sel` to the 2 push-button switches on the board.

2.8. Connect the output address from the multiplexer to the two seven-segment LED decoders, and display the selected output address.

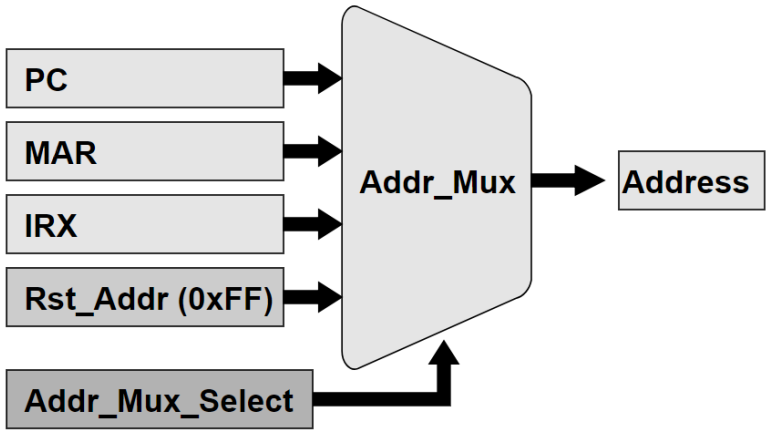


Figure 5: Simple Multiplexer (Mux)

3 Supplement: Verilog (2)

3.1 Verilog Logic Levels

Within Verilog there exist four logic levels, listed in Table 2.

Table 2: Verilog Logic Levels

Logic	Description
0	Logic Zero; False Condition
1	Logic One; True Condition
X	Unknown Logic Value
Z	High Impedance

3.2 Verilog Always and Reg Keywords

3.2.1. Behavioral modeling uses the keywords *always*.

3.2.2. Target output is a type *reg*. Unlike a wire, *reg* is updated only when a new value is assigned. In other words, it is not continuously updated as wire data types.

3.2.3. *always* may be followed by an event control expression.

3.2.4. *always* is followed by the symbol '@' which is followed by a list of variables. Each time there is a change in those variables, the *always* block is executed.

3.2.5. There is no semicolon at the end of the *always* block.

3.2.6. The list of variables are separated by the logical operator **or**, not the bitwise operator **OR**.

3.2.7. Below in Listing 1 is an example of an always block:

Listing 1: Example of an Always Block

```
1 always @(A or B)
2         //Do Stuff
```

3.3 Verilog if-else Statements

In Verilog coding, if-else statements provide a means for conditional outputs based on the arguments of the if statement. An example of this is shown in Listing 2.

Listing 2: Example of if-else Statement

```
1 output out;
2 input s,A,B;
3 reg out;
4 if(s)
5         out = A; // if s is 1, then out is A
6 else
7         out = B; // else (s =/= 1), then out is B
```

3.4 Verilog case Statements

Case Statements provide an easy way to represent a multi-branch conditional statement.

3.4.1. The first statement that makes a match is executed.

3.4.2. Unspecified bit patterns should be treated using "default" as the keyword.

An example of a case statement is provided in Listing 3.

Listing 3: Four-to-one Line Multiplexer

```
1  module mux_4x1_example(
2          output reg out,
3          input [1:0] s,              // Select, represented by 2 bit vector
4          input in_0, in_1, in_2, in_3);
5          always @(in_0,in_1,in_2,in_3,s)
6                  case(s)                  // case s
7                          2'b00: out <= in_0; // if s is 00 then output is in_0
8                          2'b01: out <= in_1; // if s is 01 then output is in_1
9                          2'b10: out <= in_2; // if s is 10 then output is in_2
10                         2'b11: out <= in_3; // if s is 11 then output is in_3
11                 endcase
12 endmodule
```

Table 3: CMOD Spartan 7 — Feature Assignments

Special	Wire	**FPGA Pin**
Button 0	BTN0	**D2**
Button 1	BTN1	**D1**
12 MHz Clock	FPGA-CLK	**M9**
RGB LED Red	LED0	**F2**
RGB LED Green	LED0	**D3**
RGB LED Blue	LED0	**F1**
LED 1	LED1	**E2**
LED 2	LED2	**K1**
LED 3	LED3	**J1**
LED 4	LED4	**E1**

Table 4: CMOD Spartan 7 — DIP Pin Assignments

DIP Pin	**FPGA Pin**	Wire	Wire	**FPGA Pin**	DIP Pin
1	**L1**	PIO01	PIO48	**A4**	48
2	**M4**	PIO02	PIO47	**A3**	47
3	**M3**	PIO03	PIO46	**B4**	46
4	**N2**	PIO04	PIO45	**B3**	45
5	**M2**	PIO05	PIO44	**C1**	44
6	**P3**	PIO06	PIO43	**B1**	43
7	**N3**	PIO07	PIO42	**B2**	42
8	**P1**	PIO08	PIO41	**A2**	41
9	**N1**	PIO09	PIO40	**C5**	40
-	**-**	-	-	**-**	-
-	**-**	-	-	**-**	-
-	**-**	-	-	**-**	-
-	**-**	-	-	**-**	-
-	**-**	-	-	**-**	-
-	**-**	-	-	**-**	-
16	**P14**	PIO16	AIN33P	**A11**	33
			AIN33N	**A12**	
17	**P15**	PIO17	AIN32P	**A13**	32
			AIN32N	**A14**	
18	**N13**	PIO18	PIO31	**J11**	31
19	**N15**	PIO19	PIO30	**M13**	30
20	**N14**	PIO20	PIO29	**L13**	29
21	**M15**	PIO21	PIO28	**J15**	28
22	**M14**	PIO22	PIO27	**K14**	27
23	**L15**	PIO23	PIO26	**L14**	26
24	**VU**	VU	GND	**GND**	25