ELEC-4200

Digital System Design

FROM: Jacob Howard

TO: Prof. Ujjwal Guin

DUE DATE: 2/4/21

Lab 2

Introduction

The goal of this lab was to learn various representations and methods for converting numbers from one representation into another. The goal was to teach us how to define numbers in various radix, design combinatorial circuits that would convert data represented in one radix into another, design circuits to perform simple addition, and learn how to increase addition speed. The main goal of this lab was to design addition circuits and decoders that display the added numbers.

Task 1

In Task 1, we were asked to define a 4-bit number to display on a 7 segment display. This circuit is called a 4-to-7 bit decoder. The main objective of Task 1 was to develop a way of displaying a number (0-9) on a 7-segment led using 4 switches assigned to the 4-bit number we created. This task was slightly challenging at first to figure out how to code, but once I developed a working code, the rest of the tasks were simple. The truth table for a 4-to-7 bit decoder can be shown in *Figure 1* below as well as the code used to develop the circuit in *Code 1* (Please note the input and output values in the truth table may not be the exact same name as used in the code).

Binary Inputs				Decoder Outputs							7 Segment Display Outputs
D	C	B	A	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	1	0	1	1	9

Figure 1

```
module Task1(
    input [3:0] a,
    output reg [6:0] z,
    output [7:0] AN
    );
    assign AN = 8'b11111110;
    always @(a)
        begin
        case(a)
         4'b0000 : begin z = 7'b0000001; end
         4'b0001 : begin z = 7'b1001111; end
         4'b0010 : begin z = 7'b0010010; end
         4'b0011 : begin z = 7'b0000110; end
         4'b0100 : begin z = 7'b1001100; end
         4'b0101 : begin z = 7'b0100100; end
         4'b0110 : begin z = 7'b0100000; end
         4'b0111 : begin z = 7'b0001111; end
         4'b1000 : begin z = 7'b0000000; end
         4'b1001 : begin z = 7'b0000100; end
         default : begin z = 7'b1111111; end
    endcase
    end
endmodule
```

Code 1

Task 2

In Task 2, we were asked to design a circuit that converts a 4-bit binary number into a 2-digit decimal equivalent. The number could range from 0-15. We had the 7-segment display show values 0-9 and once the value went over 9 (10-15), we had an LED represent a 1, the tens place and had the 7-segment display represent the ones place. We were also given a testbench file to verify the logic. I successfully developed a code that showed the correct values for the switches. The truth table for the circuit can be shown below in Figure 2 and the code can be found below in Code 2. A representation of the circuit we designed can be viewed in *Figure 3*.

v[3:0]	z	m[3:0]
0000	0	0000
0001	0	0001
0010	0	0010
0011	0	0011
0100	0	0100
0101	0	0101
0110	0	0110
0111	0	0111
1000	0	1000
1001	0	1001
1010	1	0000
1011	1	0001
1100	1	0010
1101	1	0011
1110	1	0100
1111	1	0101

Figure 2

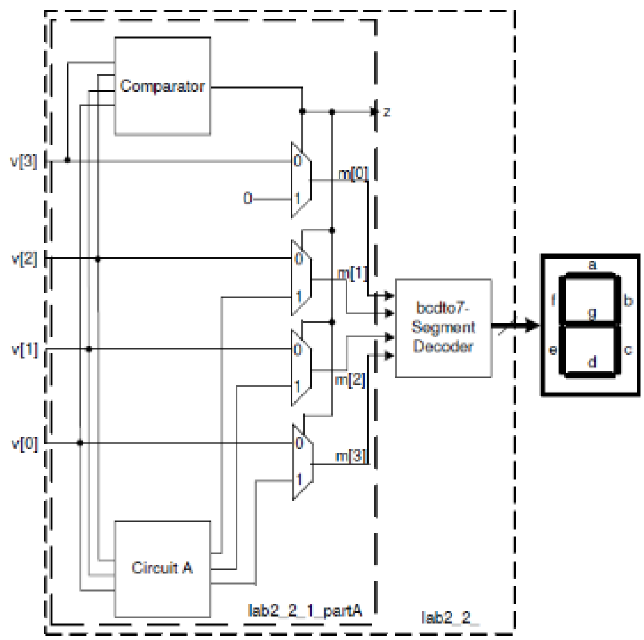
Comparator
z
v[3]
0
m[0]
v[2]
m[1]
bcdto7-
Segment
Decoder
a
f
b
g
e
c
d
v[1]
m[2]
v[0]
m[3]
Circuit A
lab2_2_1_partA
lab2_2_

Figure 3

```
module Task2(
    input [3:0] v,
    output reg z,
    output reg [6:0] seg,
    output [7:0] AN
        );
    assign AN = 8'b11111110;
    always @(v)
            begin
            case(v)
             4'b0000 : begin seg = 7'b0000001; z = 1'b0; end
             4'b0001 : begin seg = 7'b1001111; z = 1'b0; end
             4'b0010 : begin seg = 7'b0010010; z = 1'b0; end
             4'b0011 : begin seg = 7'b0000110; z = 1'b0; end
             4'b0100 : begin seg = 7'b1001100; z = 1'b0; end
             4'b0101 : begin seg = 7'b0100100; z = 1'b0; end
             4'b0110 : begin seg = 7'b0100000; z = 1'b0; end
             4'b0111 : begin seg = 7'b0001111; z = 1'b0; end
             4'b1000 : begin seg = 7'b0000000; z = 1'b0; end
             4'b1001 : begin seg = 7'b0000100; z = 1'b0; end

             4'b1010 : begin seg = 7'b0000001; z = 1'b1; end
             4'b1011 : begin seg = 7'b1001111; z = 1'b1; end
             4'b1100 : begin seg = 7'b0010010; z = 1'b1; end
             4'b1101 : begin seg = 7'b0000110; z = 1'b1; end
             4'b1110 : begin seg = 7'b1001100; z = 1'b1; end
             4'b1111 : begin seg = 7'b0100100; z = 1'b1; end

             default : begin seg = 7'b1111111; z = 1'b0; end
        endcase
        end

endmodule
```

Code 2

Task 3

In Task 3, we were asked to model a 2-out-of-5 binary code and display a 4-bit binary coded decimal input number on 5 LEDs. We were specifically told to use dataflow modeling in our code. I better understanding of a 2-out-of-5 binary code can be viewed by the truth table below in Figure 4 (Note: the truth table shows more than just the 2-out-of-5 decoder truth table).

Decimal Digits	BCD (8-4-2-1)	6-3-1-1	Excess-3	2-out-of-5	Gray code
0	0000	0000	0011	00011	0000
1	0001	0001	0100	00101	0001
2	0010	0011	0101	00110	0011
3	0011	0100	0110	01001	0010
4	0100	0101	0111	01010	0110
5	0101	0111	1000	01100	1110
6	0110	1000	1001	10001	1010
7	0111	1001	1010	10010	1011
8	1000	1011	1011	10100	1001
9	1001	1100	1100	11000	1000

Figure 4

```
module Task3(
    input [3:0] a,
    output [4:0] z
    );
    //truth tables into code
    assign
z[0]=(~a[3])&(~a[2])&(~a[1])+(~a[3])&(~a[2])&a[0]+a[2]&a[1]&(~a[0]);
        assign
z[1]=(~a[3])&(~a[2])&(~a[0])+(~a[3])&(~a[2])&(~a[0])+a[2]&a[1]&a[0];
        assign
z[2]=~(a[3])&(~a[1])&a[0]+(~a[2])&a[1]&(~a[0])+a[3]&(~a[0]);
        assign z[3]=(~a[2])&a[1]&a[0]+a[2]&(~a[1])+a[3]&a[0];
        assign z[4]=a[2]&a[1]&(~a[0])+a[3];
endmodule
```

Code 3

Task 4

Task 4 had 2 parts. In part 1, we were asked to create a 1-bit full adder with 3 inputs (a, b, and cin) and 2 outputs (s and cout). Variables a and b are the 1-bit inputs, cin is a carry-in 1-bit input, and s is the sum, with cout being carry out. We were given a testbench file to confirm the truth table but this code was not directly put onto the board for physical testing. A truth table of a 1-bit adder can be seen below in *Figure 5.*

In part 2, we were to use the knowledge and code from part 1 to create a 4-bit ripple carry adder with 3 inputs (a, b, and cin, all being 4-bits) and 2 outputs (cout, and s, also being 4-bit outputs each). We were asked to assign switch 4 through switch 7 on the board to a and switch 0 through switch 3 to b, also adding cin to switch 15. LED 0 through LED 3 were assigned to s (the sum) and cout (carry out) to LED 15. Once we had designed a code, we were asked to write the code onto the board for testing. My code worked correctly and was verified with the truth table in *Figure 6* below. The code for Part 1 can be seen in *Code 4* and Part 2 can be seen in *Code 5*.

Inputs			Outputs	
A	B	C – IN	Sum	C - Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 5

A_1	A_2	A_3	A_4	B_4	B_3	B_2	B_1	S_4	S_3	S_2	S_1	Carry
0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	1
1	0	1	0	1	0	1	0	0	1	0	0	1
1	1	0	0	1	1	0	0	1	0	0	0	1
1	1	1	0	1	1	1	0	1	1	0	0	1
1	1	1	1	1	1	1	1	1	1	1	0	1

Figure 6

```
module Task4(
    input a,
    input b,
    input cin,
    output sum,
    output cout
    );
    assign sum = a ^ b ^ cin;
    assign cout = a & b | a & cin | b & cin;

endmodule
```

Code 4

```
module Task4_4BitRippleCarryAdder(
    input [3:0] a,
    input [3:0] b,
    input cin,
    output [3:0] sum,
    output cout
    );
    wire cout1;
    wire cout2;
    wire cout3;

    //first adder
    assign sum[0] = a[0] ^ b[0] ^ cin;
    assign cout1 = a[0]&b[0] | a[0]&cin |
b[0]&cin;

    //seccond adder
    assign sum[1] = a[1] ^ b[1] ^ cout1;
    assign cout2 = a[1]&b[1] | a[1]&cout1 |
b[1]&cout1;

    //third adder
    assign sum[2] = a[2] ^ b[2] ^ cout2;
    assign cout3 = a[2]&b[2] | a[2]&cout2 |
b[0]&cout2;

    //fourth adder
    assign sum[3] = a[3] ^ b[3] ^ cout3;
    assign cout = a[3]&b[3] | a[3]&cout3 |
b[3]&cout3;

endmodule
```

Code 5

Task 5

In Task 5we were asked to modify our design of Task 4's ripple carry adder as a 4-bit input to a 4-to-7 bit decoder. We were to perform addition and generate the results in the BCD, displaying the number on LED0 and the 7-segment display. We used switch 7 through switch 0 as the input switches, switch 15 being the carry-in input switch. My designed code performed as intended and can be seen in *Code 6* below. The code cannot fit on one single page, so it is shown below on to pages.

```
module Task5(
input [3:0] a,
   input [3:0] b,
   input cin,
   output [3:0] sum,
   output cout,
   output reg z,
   output reg [6:0] seg,
   output [7:0] AN
   );

   ///////////////////////////////4-Bit Adder/////////////////////////////////
   wire cout1;
   wire cout2;
   wire cout3;

   //first adder
   assign sum[0] = a[0] ^ b[0] ^ cin;
   assign cout1 = a[0]&b[0] | a[0]&cin | b[0]&cin;

   //seccond adder
   assign sum[1] = a[1] ^ b[1] ^ cout1;
   assign cout2 = a[1]&b[1] | a[1]&cout1 | b[1]&cout1;

   //third adder
   assign sum[2] = a[2] ^ b[2] ^ cout2;
   assign cout3 = a[2]&b[2] | a[2]&cout2 | b[0]&cout2;

   //fourth adder
   assign sum[3] = a[3] ^ b[3] ^ cout3;
   assign cout = a[3]&b[3] | a[3]&cout3 | b[3]&cout3;

   /////////////////////7-Segment Display with Carry LED/////////////////

   assign AN = 8'b11111110;
   always @(sum & z & cout)
          begin
          if(cout == 0)
          case(sum)
           4'b0000 : begin seg = 7'b0000001; z = 1'b0; end //0
           4'b0001 : begin seg = 7'b1001111; z = 1'b0; end //1
           4'b0010 : begin seg = 7'b0010010; z = 1'b0; end //2
           4'b0011 : begin seg = 7'b0000110; z = 1'b0; end //3
           4'b0100 : begin seg = 7'b1001100; z = 1'b0; end //4
           4'b0101 : begin seg = 7'b0100100; z = 1'b0; end //5
           4'b0110 : begin seg = 7'b0100000; z = 1'b0; end //6
           4'b0111 : begin seg = 7'b0001111; z = 1'b0; end //7
           4'b1000 : begin seg = 7'b0000000; z = 1'b0; end //8
           4'b1001 : begin seg = 7'b0000100; z = 1'b0; end //9

           //2 Digit LED
           4'b1010 : begin seg = 7'b0000001; z = 1'b1; end //10
           4'b1011 : begin seg = 7'b1001111; z = 1'b1; end //11
           4'b1100 : begin seg = 7'b0010010; z = 1'b1; end //12
           4'b1101 : begin seg = 7'b0000110; z = 1'b1; end //13
           4'b1110 : begin seg = 7'b1001100; z = 1'b1; end //14
```

```
            4'b1111 : begin seg = 7'b0100100; z = 1'b1; end //15

            default : begin seg = 7'b1111111; z = 1'b0; end //default
case

        endcase
        else
        case(sum)
        4'b0000 : begin seg = 7'b0100000; z = 1'b1; end //16
        4'b0001 : begin seg = 7'b0001111; z = 1'b0; end //17
        4'b0010 : begin seg = 7'b0000000; z = 1'b0; end //18

        default : begin seg = 7'b1111111; z = 1'b0; end //default
case
        endcase
        end

endmodule
```

Code 6

Task 6

The last task was pretty simple compared to the previous task. We were asked to create a carry look-ahead adder. This is essentially an adder that looks at the inputs to see if a carry is needed instead of waiting for the added output. This is a faster adder than a regular adder. My design and code worked as intended and was tested on the board. A visual circuit design can be seen in *Figure 7* below and the code for this task is shown in *Code 7*.

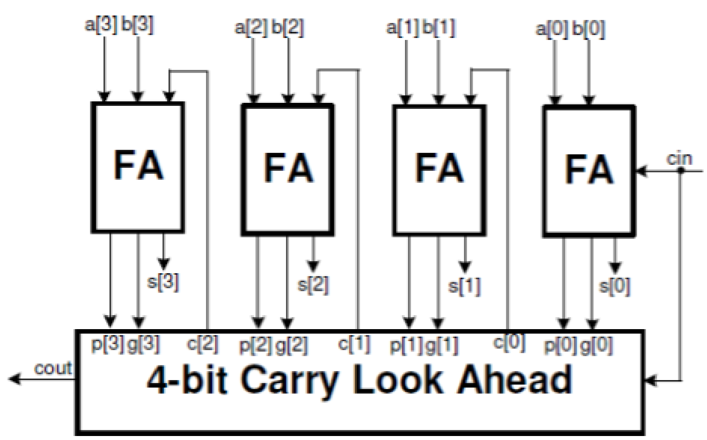

Figure 7

```
module Task6(
input [3:0] a,
    input [3:0] b,
    input cin,
    output [3:0] sum,
    output cout
    );
    wire cout1;
    wire cout2;
    wire cout3;
    wire [3:0] p;
    wire [3:0] g;

    //first adder
    assign p[0] = a[0] | b[0];
    assign g[0] = a[0] & b[0];
    assign sum[0] = a[0] ^ b[0] ^ cin;
    assign cout1 = g[0] | (p[0] & cin);

    //seccond adder
    assign p[1] = a[1] | b[1];
    assign g[1] = a[1] & b[1];
    assign sum[1] = a[1] ^ b[1] ^ cout1;
    assign cout2 = g[1] | (p[1] & cout1);

    //third adder
    assign p[2] = a[2] | b[2];
    assign g[2] = a[2] & b[2];
    assign sum[2] = a[2] ^ b[2] ^ cout2;
    assign cout3 = g[2] | (p[2] & cout2);

    //fourth adder
    assign p[3] = a[3] | b[3];
    assign g[3] = a[3] & b[3];
    assign sum[3] = a[3] ^ b[3] ^ cout3;
    assign cout = g[3] | (p[3] & cout3);
endmodule
```

Code 7

Conclusion

In conclusion, this lab was very helpful in helping us to develop combinational logic circuits. The lab tested our critical thinking knowledge on how to develop some complex circuits into code and taught us some techniques on how to make improved circuits. Overall the lab was very useful for learning how to transform addition circuits into a software design for capable hardware.