PHYS 234 Lab3

Ethan Yang

Part A. Prime Number Detector

The goal is to determine if a 4-digit binary number is a prime number with 2 different approaches: first with if statements, and second with a case statement.

To accomplish this, first we will translate the 4-digit binary number into decimal and assign it to the sum variable. Since the maximum number a 4-digit binary number can store is 15, we only need to check the prime numbers smaller than 15, that includes: 2, 3, 5, 7, 11, and 13.

Next we implement if statements to check if our sum equals any of the prime numbers above. If yes, we would assign 1 to the output variable and thus our LED would light up, and vice versa.

Case statements work in a similar way. We start by initializing our output e to 0. Then, using a case statement for sum, we define cases for each prime number less than 15. Whenever sum matches one of these cases, we set e to 1 and the LED will light up.

Verilog Code for if-statement approach:

```
`timescale 1ns / 1ps
module eelab3( input a, input b, input c, input d, output reg e );
wire [3:0] sum;
assign sum = a+b*2+c*4+d*8;
always @ (*)
      if (sum==2)
            e = 1;
      else if (sum==3)
            e = 1;
      else if (sum==5)
            e = 1;
      else if (sum==7)
            e = 1;
      else if (sum==11)
            e = 1;
      else if (sum==13)
            e = 1;
      else
            e = 0;

endmodule
```

Verilog Code for case-statement approach:

```
`timescale 1ns / 1ps
module eelab3( input a, input b, input c, input d, output reg e );
wire [3:0] sum;
assign sum = a+b*2+c*4+d*8;
always @ (*)
      begin
            e = 0;
            case (sum)
                  2:e = 1;
                  3:e = 1;
                  5:e = 1;
                  7:e = 1;
                  11:e = 1;
                  13:e = 1;
            endcase
      end
endmodule
```

The results can be represented using the table below. It shows every combination of the 4-digit binary number, its corresponding decimal number, and whether or not it is prime.

A	B	C	D	Decimal	Prime
0	0	0	0	0	0
1	0	0	0	8	0
0	1	0	0	4	0
0	0	1	0	2	1
0	0	0	1	1	0
1	1	0	0	12	0
1	0	0	1	9	0
1	0	1	0	10	0
0	0	1	1	3	1
0	1	1	0	6	0

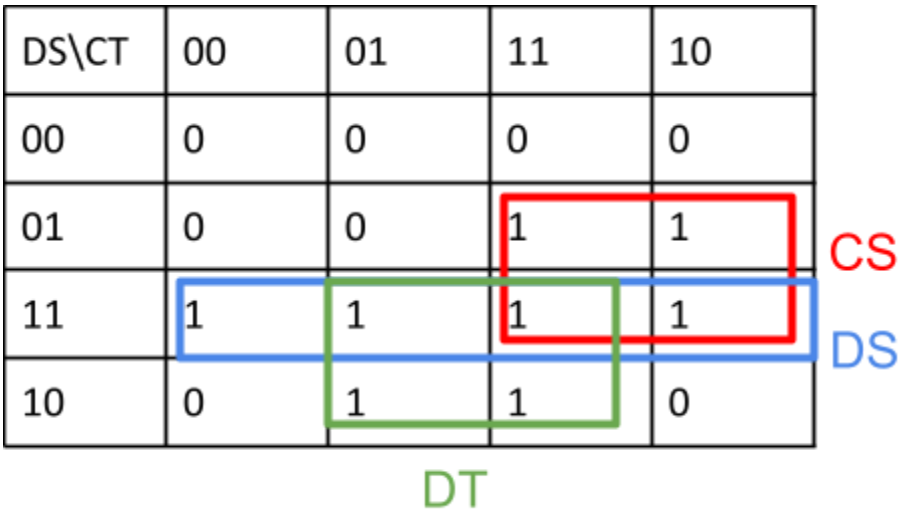
0	1	0	1	5	1
1	1	1	0	14	0
0	1	1	1	7	1
1	0	1	1	11	1
1	1	0	1	13	1
1	1	1	1	15	0

Part B. Follow the Yellow Brick Road

The goal of the circuit is to get the majority of the votes from Dorothy, the Scarecrow, the Tin Woodman, and the Cowardly Lion on whether or not to travel to Oz. The twist is, Dorothy's vote determines whose vote doesn't get counted. If she votes 1, the Cowardly Lion's vote doesn't count; if she votes 0, the Tin Woodman's vote doesn't count. We will construct the circuit using three methods.

Method 1: Using K-Map to minimize the circuit

K-Map
A K-map is a method to simplify boolean expressions that results in a more efficient circuit by reducing the number of gates and variables. Below is a k-map for our voting example.



We can group the largest possible groups of 1's that are 4x2, 4x1, 2x2, 2x1, or 1x1 and determine their corresponding circuit by eliminating common variables that change. As a result, the red group produces the circuit "CS", the blue group produces "DS", and the green group produces "DT". By adding/or-ing them together, we can get our finalized circuit of

“y=CS+DS+DT”, which is simplified to “y=S(C+D)+DT”. The circuit can be programmed into our FPGA using the verilog code below:

```
`timescale 1ns / 1ps
module eelab3( input d, input c, input t, input s, output e );
assign e = s&(c|d)|(d&t);
endmodule
```

Method 2: Use if statements based on Dorothy’s vote to determine sum of the corrected 3 votes

From the K-map, we can identify the cases where the voting output is 1. We can compute the decimal sum of them. These entries are: 0111 (7 in decimal), 1001 (9), 1010 (10), 1011 (11), 1101 (13), 1110 (14), and 1111 (15). Using an if-else-statement, we can check whether our sum matches any of these values. If it does, we assign our output e to 1, and the LED lights on.

Verilog Code:

```
`timescale 1ns / 1ps
module eelab3( input d, input c, input t, input s, output reg e );
wire [3:0] sum;
assign sum = d+c*2+t*4+s*8;
always @ (*)
      if (sum==7)
            e = 1;
      else if (sum==9)
            e = 1;
      else if (sum==10)
            e = 1;
      else if (sum==11)
            e = 1;
      else if (sum==13)
            e = 1;
      else if (sum==14)
            e = 1;
      else if (sum==15)
            e = 1;
      else
            e = 0;
endmodule
```

Method 3: Assign sum to the 4 votes, then use if statements

We can approach the problem with a different logic. First, we will tally the number of votes equal to 1. Using an if-statement, we check Dorothy's vote: if she voted 1, Cowardly Lion's vote is excluded from the tally; if she voted 0, Tin Woodman's vote is excluded from the tally. Lastly, we compare the tally to determine whether the majority voted 1, and assign 1 or 0 accordingly.

Verilog Code:

```
`timescale 1ns / 1ps
module eelab3( input d, input c, input t, input s, output reg e );
wire [3:0] sum;
assign sum = d+c+t+s;
always @ (*)
      if (d == 1)
            sum = sum - c;
      else
            sum = sum - t;
      if (sum >= 3)
             e = 1;
       else
            e = 0;
endmodule
```

Results can be shown using the table below. The letters correspond to the voters' initials, and the 4-digit binary number is also shown in decimal specifically for method 2.

S	T	C	D	Decimal	Voting Results
0	0	0	0	0	0
1	0	0	0	8	0
0	1	0	0	4	0
0	0	1	0	2	0
0	0	0	1	1	0
1	1	0	0	12	0
1	0	0	1	9	1
1	0	1	0	10	1

0	0	1	1	3	0
0	1	1	0	6	0
0	1	0	1	5	1
1	1	1	0	14	1
0	1	1	1	7	1
1	0	1	1	11	1
1	1	0	1	13	1
1	1	1	1	15	1