بسم الله الرحمن الرحيم

Computer Systems Engineering Department

Advanced Digital

ENCS3310

Project report

Special counter

Instruction: Dr. Abdullatif Abu Issa

Student Name & ID:

Nafe Abubaker, 1200047

Table of Contents

Theory

This report presents the design and implementation of a special counter circuit capable of counting two different sequences: the Prime Numbers Sequence and the Fibonacci Sequence. The counter can operate in both up and down modes, depending on the input values. The design utilizes T-Flip Flops and combination logic to achieve the desired functionality. A testbench is developed to verify the correctness of the design, and the results are analyzed. This report provides a detailed explanation of the design philosophy, presents the results obtained, and concludes with future work suggestions

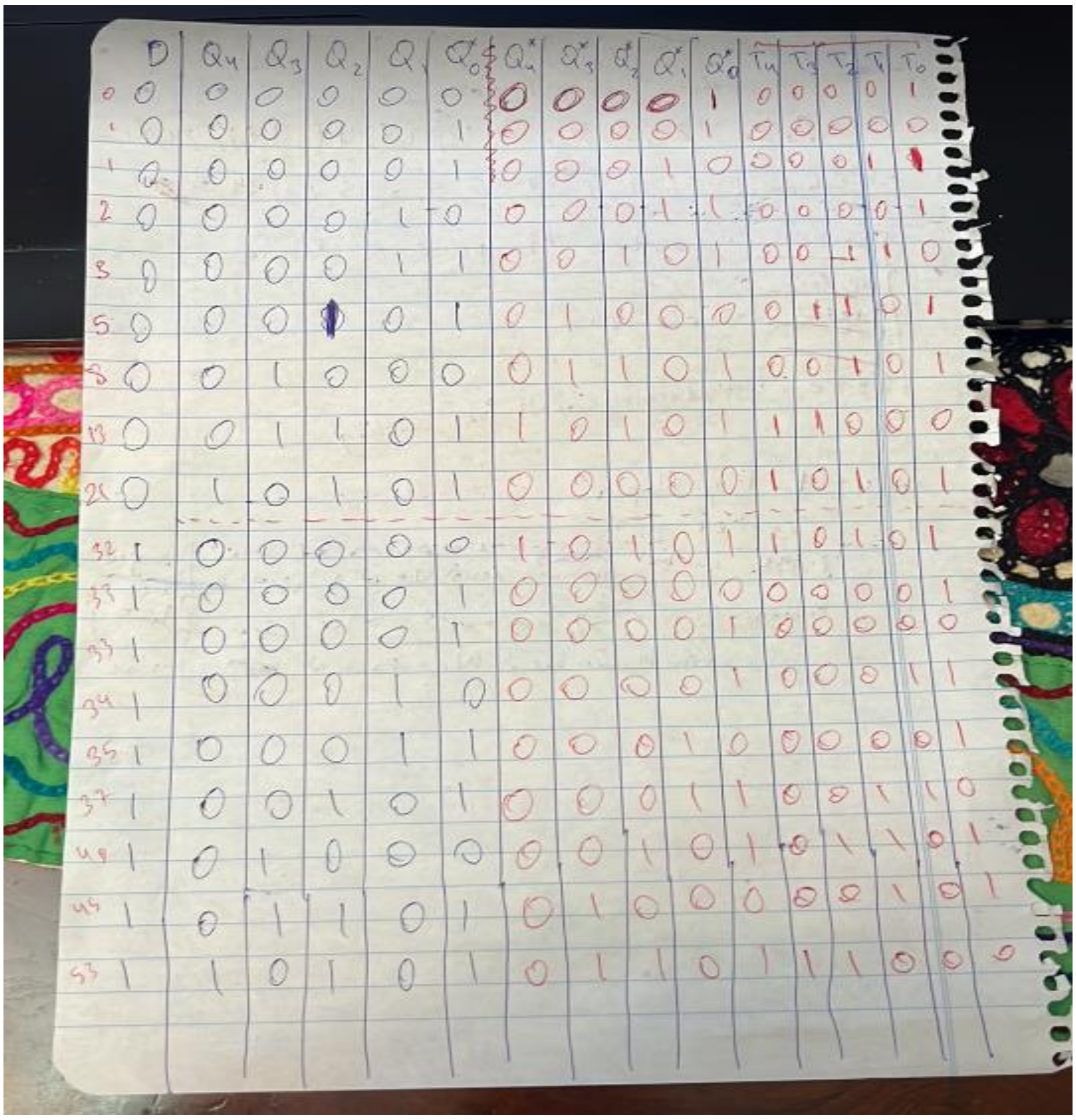
Design Philosophy

Prime counter

At first, I noticed that we want to create a counter for first 11 numbers for both Fibonacci and prime numbers.

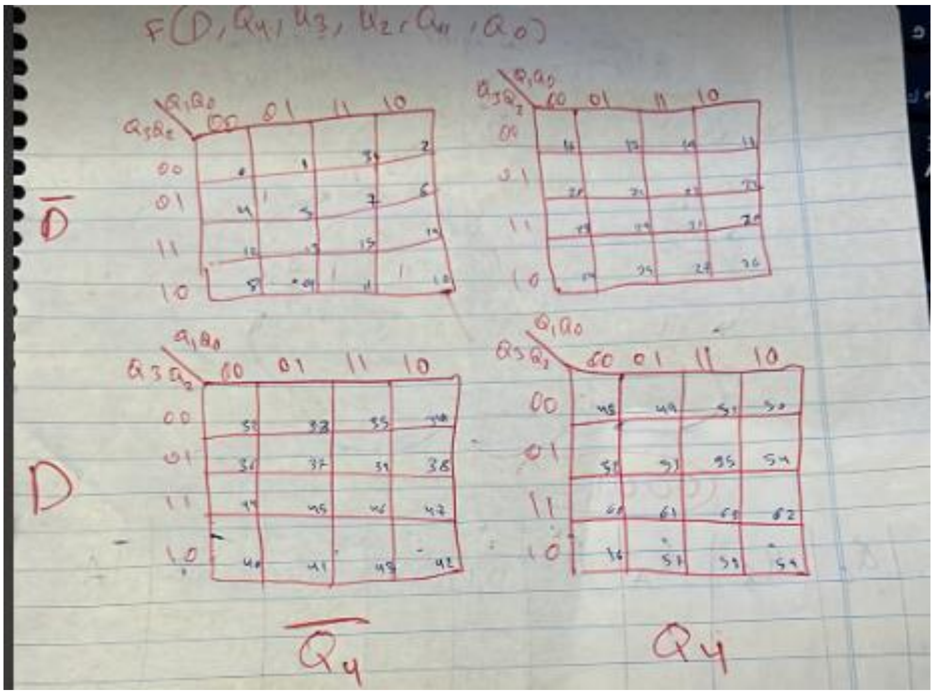
So I headed out and worked with the sequential simulation by hand.
First, I started with the prime number, I draw the truth table and fill in the values that will be considered in the prime number counter, which are in sequence: 2,3,5,7,11,13,17,19,23,29,31.
and since the biggest number is 31, I figured out that I must use 5 bit counter.

Then I added an extra bit to represent the direction that we are going to work with(up/down). And came up with this truth table:



	D	Q_4	Q_3	Q_2	Q_1	Q_0	Q_4^*	Q_3^*	Q_2^*	Q_1^*	Q_0^*	T_4	T_3	T_2	T_1	T_0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	1
2	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	1
3	0	0	0	0	1	1	0	0	1	0	1	0	0	1	1	0
5	0	0	0	1	0	1	0	1	0	0	0	0	1	1	0	1
8	0	0	1	0	0	0	0	1	1	0	1	0	0	1	0	1
13	0	0	1	1	0	1	1	0	1	0	1	1	1	0	0	0
21	0	1	0	1	0	1	0	0	0	0	0	1	0	1	0	1
32	1	0	0	0	0	0	1	0	1	0	1	1	0	1	0	1
33	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
33	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
34	1	0	0	0	1	0	0	0	0	0	1	0	0	0	1	1
35	1	0	0	0	1	1	0	0	0	1	0	0	0	0	0	1
37	1	0	0	1	0	1	0	0	0	1	1	0	0	1	1	0
40	1	0	1	0	0	0	0	0	1	0	1	0	1	1	0	1
45	1	0	1	1	0	1	0	1	0	0	0	0	0	1	0	1
53	1	1	0	1	0	1	0	1	1	0	1	1	1	0	0	0

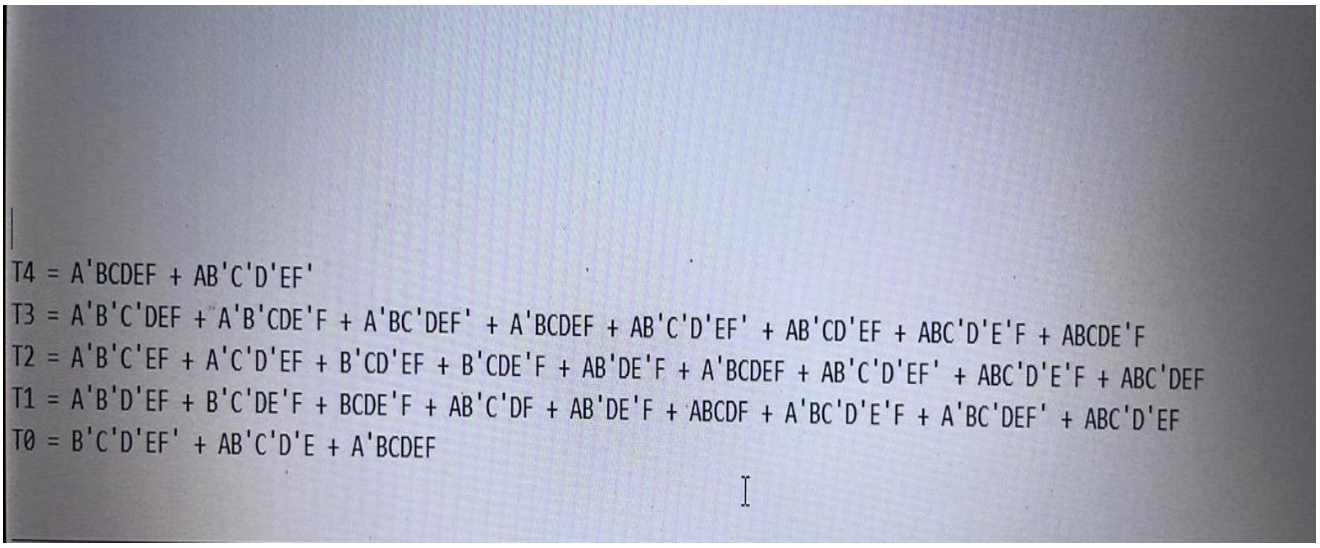
And after evaluating the truth table, I had two choices, the first one is to figure out the equations using K-Map by hand as shown in the next figure.



Or to use a website that can do that for me. And due to the lack of time, I chose the second option. So, I headed to this website that can solve out and figure the equations faster than me.

http://www.32x8.com/var5.html

And got the following equations.



```
T4 = A'BCDEF + AB'C'D'EF'
T3 = A'B'C'DEF + A'B'CDE'F + A'BC'DEF' + A'BCDEF + AB'C'D'EF' + AB'CD'EF + ABC'D'E'F + ABCDE'F
T2 = A'B'C'EF + A'C'D'EF + B'CD'EF + B'CDE'F + AB'DE'F + A'BCDEF + AB'C'D'EF' + ABC'D'E'F + ABC'DEF
T1 = A'B'D'EF + B'C'DE'F + BCDE'F + AB'C'DF + AB'DE'F + ABCDF + A'BC'D'E'F + A'BC'DEF' + ABC'D'EF
T0 = B'C'D'EF' + AB'C'D'E + A'BCDEF
```

A,B,C,D,E,F represent **direction, count[4], count[3], count[2], count[1], count[0]** respectively.

Fibonacci Counter

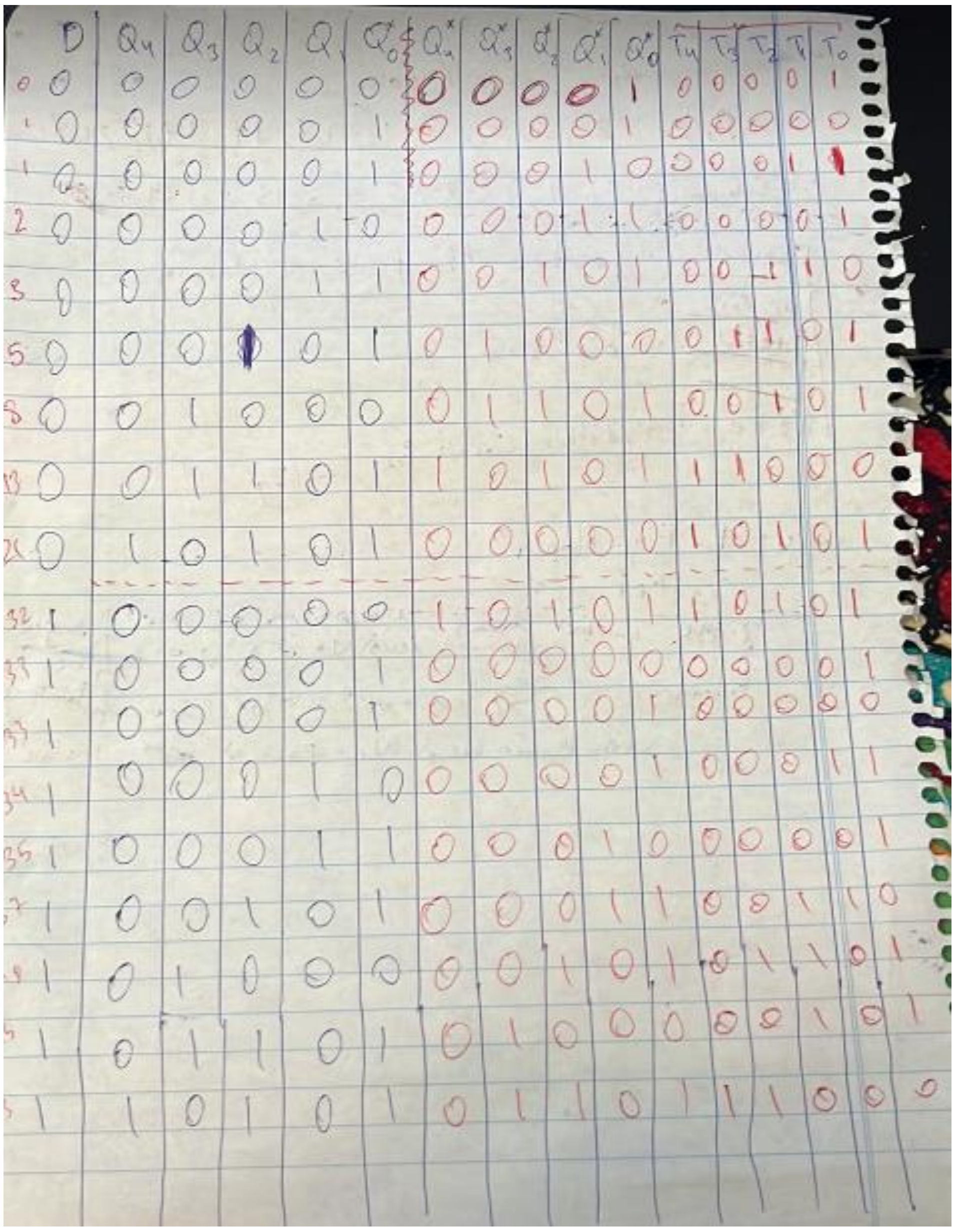
The same steps used in Prime counter are used here in Fibonacci Counter.

I draw the truth table and fill in the values that will be considered in the Fibonacci number counter, which are in sequence:

0,1,1,2,3,5,8,13,21,34,55

And since the biggest number Is 55, I noticed that I must use 6 bits, unfortunately I had no such time to work with 6 bits so I worked with 5 bits that can count up to 21.

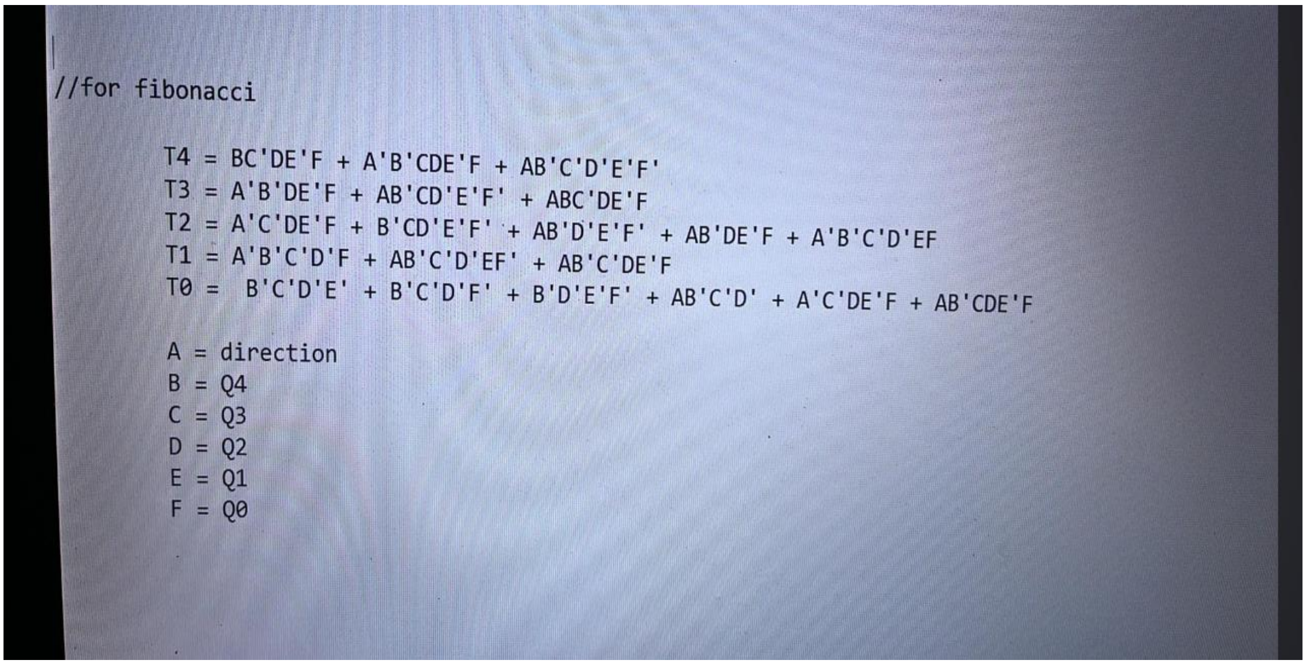
I also added the direction bit to the truth table and got the following:



	D	Q_4	Q_3	Q_2	Q_1	Q_0	Q_4^*	Q_3^*	Q_2^*	Q_1^*	Q_0^*	T_4	T_3	T_2	T_1	T_0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	1
2	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	1
3	0	0	0	0	1	1	0	0	1	0	1	0	0	1	1	0
5	0	0	0	1	0	1	0	1	0	0	0	0	1	1	0	1
8	0	0	1	0	0	0	0	1	1	0	1	0	0	1	0	1
13	0	0	1	1	0	1	1	0	1	0	1	1	1	0	0	0
21	0	1	0	1	0	1	0	0	0	0	0	1	0	1	0	1
32	1	0	0	0	0	0	1	0	1	0	1	1	0	1	0	1
33	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
33	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
34	1	0	0	0	1	0	0	0	0	0	1	0	0	0	1	1
35	1	0	0	0	1	1	0	0	0	1	0	0	0	0	0	1
37	1	0	0	1	0	1	0	0	0	1	1	0	0	1	1	0
9	1	0	1	0	0	0	0	0	1	0	1	0	1	1	0	1
5	1	0	1	1	0	1	0	1	0	0	0	0	0	1	0	1
5	1	1	0	1	0	1	0	1	1	0	1	1	1	0	0	0

Then I headed to the same link mentioned previously to get faster equations.

And I got the following:

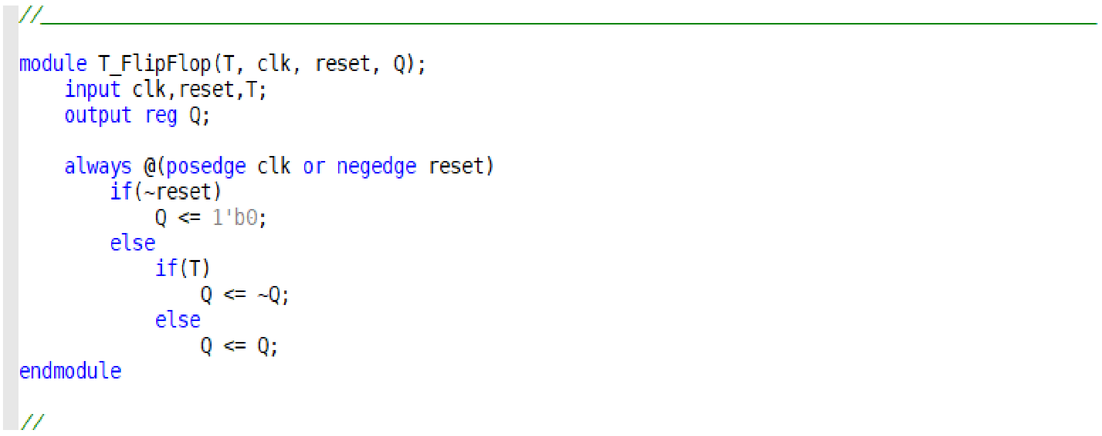


```
//for fibonacci

        T4 = BC'DE'F + A'B'CDE'F + AB'C'D'E'F'
        T3 = A'B'DE'F + AB'CD'E'F' + ABC'DE'F
        T2 = A'C'DE'F + B'CD'E'F' + AB'D'E'F' + AB'DE'F + A'B'C'D'EF
        T1 = A'B'C'D'F + AB'C'D'EF' + AB'C'DE'F
        T0 =  B'C'D'E' + B'C'D'F' + B'D'E'F' + AB'C'D' + A'C'DE'F + AB'CDE'F

        A = direction
        B = Q4
        C = Q3
        D = Q2
        E = Q1
        F = Q0
```

A,B,C,D,E,F represent **direction, count[4], count[3], count[2], count[1], count[0]** respectively.

Code

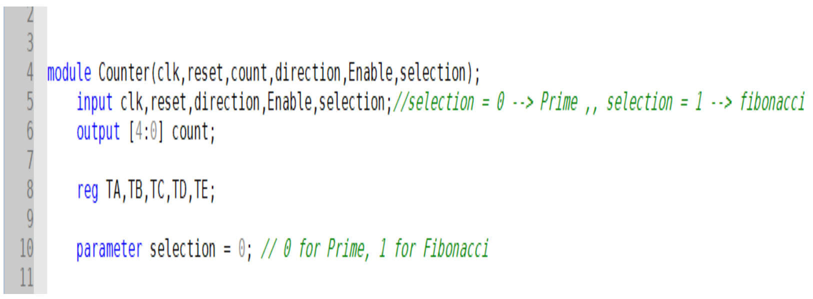
First, I created the Flipflop module



```verilog
//______________________________________________________________________________

module T_FlipFlop(T, clk, reset, Q);
    input clk,reset,T;
    output reg Q;

    always @(posedge clk or negedge reset)
        if(~reset)
            Q <= 1'b0;
        else
            if(T)
                Q <= ~Q;
            else
                Q <= Q;
endmodule

//
```

:

Then, I started with the counter module and initialized the variables, (inputs, outputs, registers, etc.)



```verilog


module Counter(clk,reset,count,direction,Enable,selection);
   input clk,reset,direction,Enable,selection;//selection = 0 --> Prime ,, selection = 1 --> fibonacci
   output [4:0] count;

   reg TA,TB,TC,TD,TE;

   parameter selection = 0; // 0 for Prime, 1 for Fibonacci

```

After this, I used a couple of "if" statements to deal with the select and enable inputs.

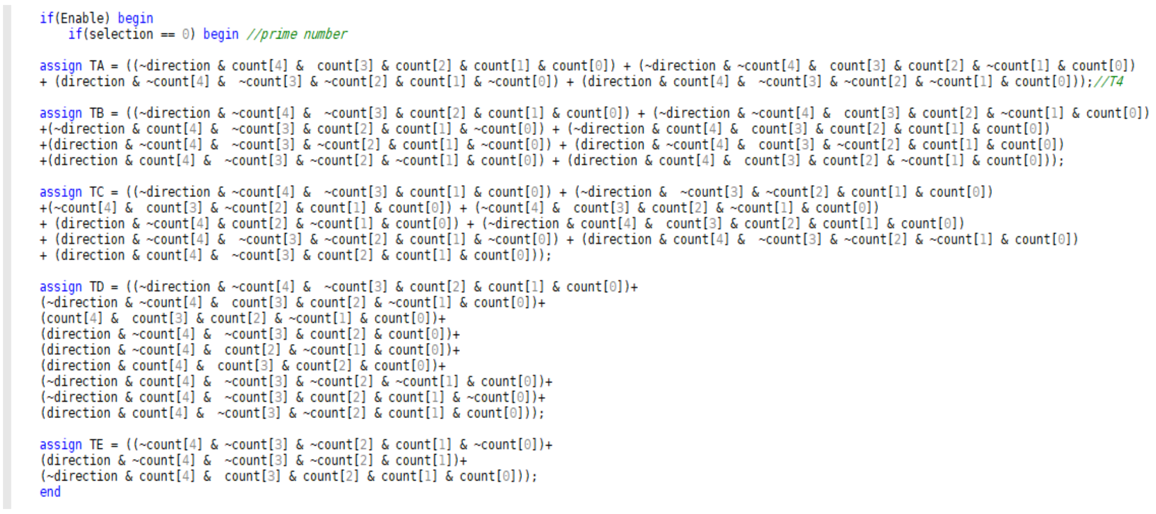
If select = 0, then counter will count prime numbers.

If select = 1, then the counter will count Fibonacci.

And all of this will execute if Enable = 0.

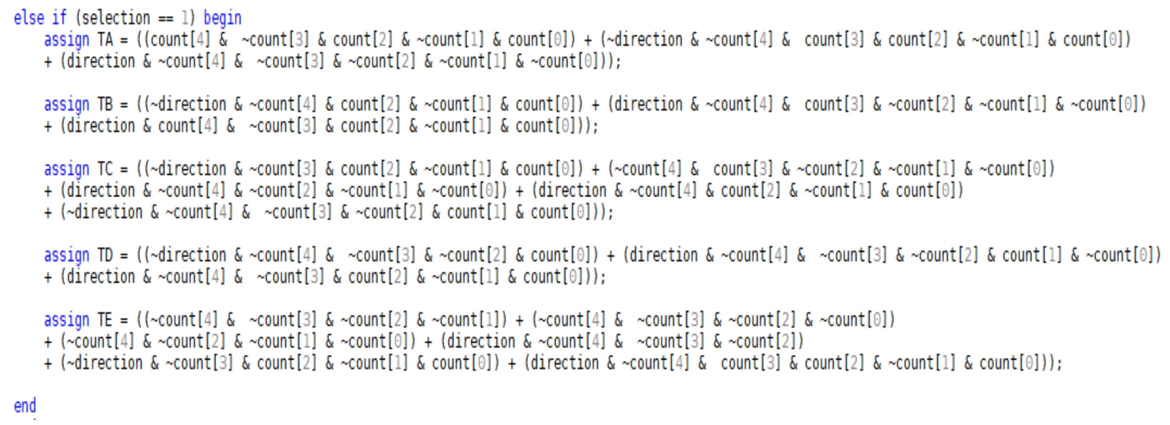
After that I used the registers (TA, TB, TC, TD, TE) to assign the equations into.

And for the prime numbers it came out like this:



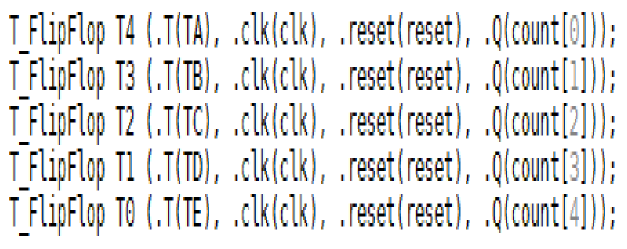
```
if(Enable) begin
    if(selection == 0) begin //prime number

assign TA = ((~direction & count[4] &  count[3] & count[2] & count[1] & count[0]) + (~direction & ~count[4] &  count[3] & count[2] & ~count[1] & count[0])
+ (direction & ~count[4] &  ~count[3] & ~count[2] & count[1] & ~count[0]) + (direction & count[4] &  ~count[3] & ~count[2] & ~count[1] & count[0]));//T4

assign TB = ((~direction & ~count[4] &  ~count[3] & count[2] & count[1] & count[0]) + (~direction & ~count[4] &  count[3] & count[2] & ~count[1] & count[0])
+(~direction & count[4] &  ~count[3] & count[2] & count[1] & ~count[0]) + (~direction & count[4] &  count[3] & count[2] & count[1] & count[0])
+(direction & ~count[4] &  ~count[3] & ~count[2] & count[1] & ~count[0]) + (direction & ~count[4] &  count[3] & ~count[2] & count[1] & count[0])
+(direction & count[4] &  ~count[3] & ~count[2] & ~count[1] & count[0]) + (direction & count[4] &  count[3] & count[2] & ~count[1] & count[0]));

assign TC = ((~direction & ~count[4] &  ~count[3] & count[1] & count[0]) + (~direction &  ~count[3] & ~count[2] & count[1] & count[0])
+(~count[4] &  count[3] & ~count[2] & count[1] & count[0]) + (~count[4] &  count[3] & count[2] & ~count[1] & count[0])
+ (direction & ~count[4] & count[2] & ~count[1] & count[0]) + (~direction & count[4] &  count[3] & count[2] & count[1] & count[0])
+ (direction & ~count[4] &  ~count[3] & ~count[2] & count[1] & ~count[0]) + (direction & count[4] &  ~count[3] & ~count[2] & ~count[1] & count[0])
+ (direction & count[4] &  ~count[3] & count[2] & count[1] & count[0]));

assign TD = ((~direction & ~count[4] &  ~count[3] & count[2] & count[1] & count[0])+
(~direction & ~count[4] &  count[3] & count[2] & ~count[1] & count[0])+
(count[4] &  count[3] & count[2] & ~count[1] & count[0])+
(direction & ~count[4] &  ~count[3] & count[2] & count[0])+
(direction & ~count[4] &  count[2] & ~count[1] & count[0])+
(direction & count[4] &  count[3] & count[2] & count[0])+
(~direction & count[4] &  ~count[3] & ~count[2] & ~count[1] & count[0])+
(~direction & count[4] &  ~count[3] & count[2] & count[1] & ~count[0])+
(direction & count[4] &  ~count[3] & ~count[2] & count[1] & count[0]));

assign TE = ((~count[4] & ~count[3] & ~count[2] & count[1] & ~count[0])+
(direction & ~count[4] &  ~count[3] & ~count[2] & count[1])+
(~direction & count[4] &  count[3] & count[2] & count[1] & count[0]));
end
```

And for the Fibonacci it came out like this:



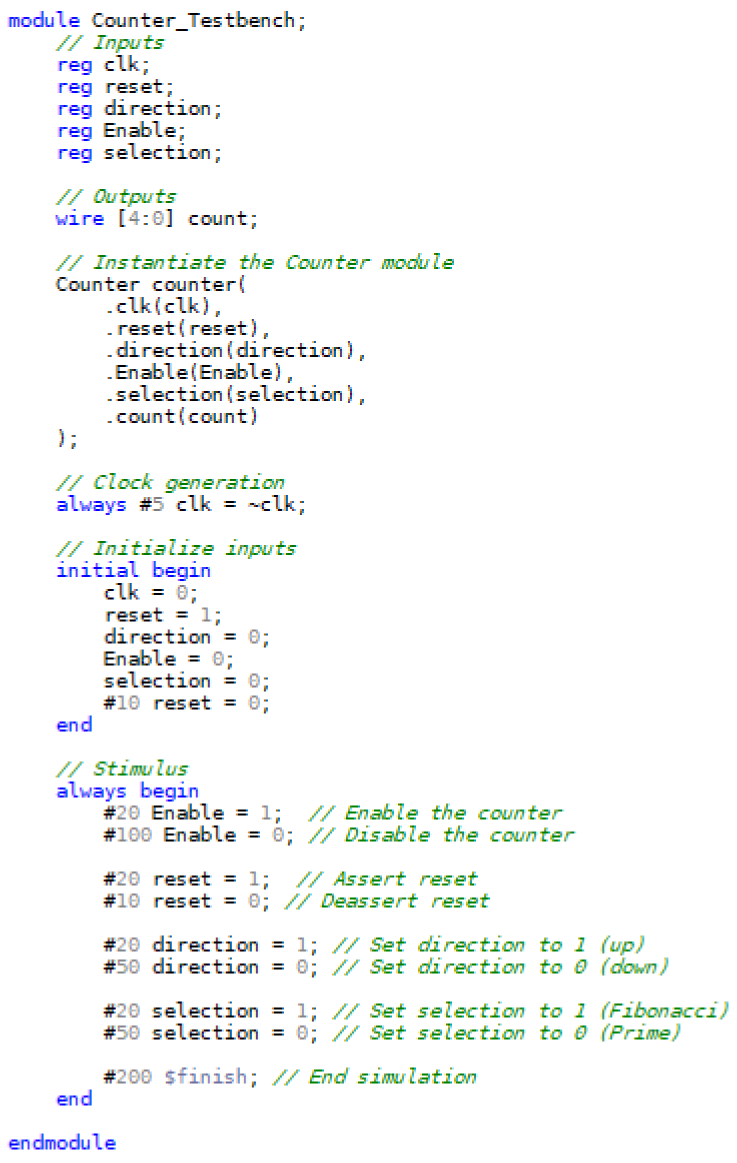
```
else if (selection == 1) begin
    assign TA = ((count[4] &  ~count[3] & count[2] & ~count[1] & count[0]) + (~direction & ~count[4] &  count[3] & count[2] & ~count[1] & count[0])
    + (direction & ~count[4] &  ~count[3] & ~count[2] & ~count[1] & ~count[0]));

    assign TB = ((~direction & ~count[4] & count[2] & ~count[1] & count[0]) + (direction & ~count[4] &  count[3] & ~count[2] & ~count[1] & ~count[0])
    + (direction & count[4] &  ~count[3] & count[2] & ~count[1] & count[0]));

    assign TC = ((~direction & ~count[3] & count[2] & ~count[1] & count[0]) + (~count[4] &  count[3] & ~count[2] & ~count[1] & ~count[0])
    + (direction & ~count[4] & ~count[2] & ~count[1] & ~count[0]) + (direction & ~count[4] & count[2] & ~count[1] & count[0])
    + (~direction & ~count[4] &  ~count[3] & ~count[2] & count[1] & count[0]));

    assign TD = ((~direction & ~count[4] &  ~count[3] & ~count[2] & count[0]) + (direction & ~count[4] &  ~count[3] & ~count[2] & count[1] & ~count[0])
    + (direction & ~count[4] &  ~count[3] & count[2] & ~count[1] & count[0]));

    assign TE = ((~count[4] &  ~count[3] & ~count[2] & ~count[1]) + (~count[4] &  ~count[3] & ~count[2] & ~count[0])
    + (~count[4] & ~count[2] & ~count[1] & ~count[0]) + (direction & ~count[4] &  ~count[3] & ~count[2])
    + (~direction & ~count[3] & count[2] & ~count[1] & count[0]) + (direction & ~count[4] &  count[3] & count[2] & ~count[1] & count[0]));

end
```

After the assigning, I used the Flipflop module to figure out the next state of the counter:



```
T_FlipFlop T4 (.T(TA), .clk(clk), .reset(reset), .Q(count[0]));
T_FlipFlop T3 (.T(TB), .clk(clk), .reset(reset), .Q(count[1]));
T_FlipFlop T2 (.T(TC), .clk(clk), .reset(reset), .Q(count[2]));
T_FlipFlop T1 (.T(TD), .clk(clk), .reset(reset), .Q(count[3]));
T_FlipFlop T0 (.T(TE), .clk(clk), .reset(reset), .Q(count[4]));
```

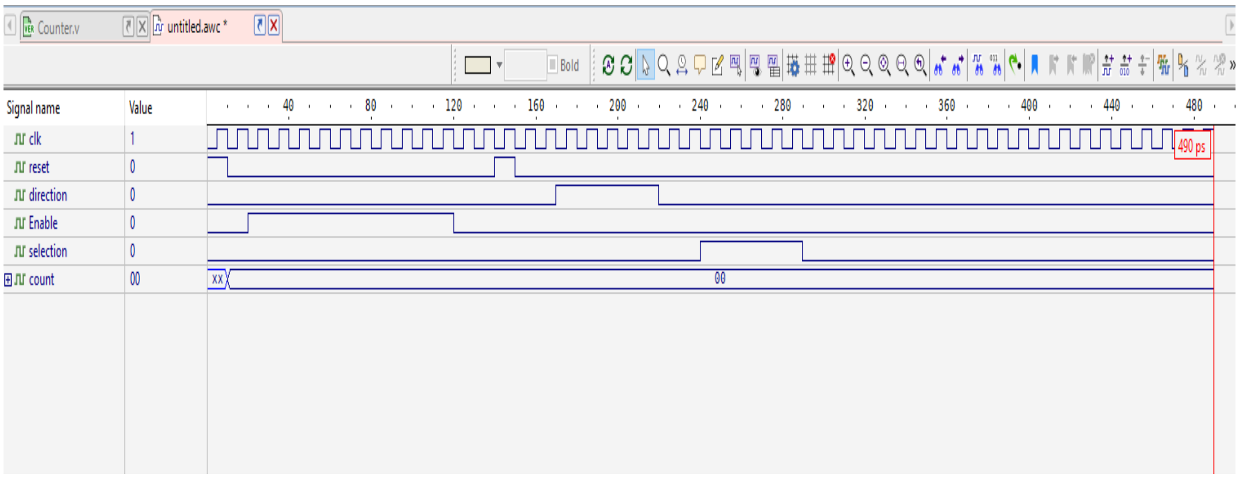
The last module was the testbench which will test the whole circuit.



```
module Counter_Testbench;
    // Inputs
    reg clk;
    reg reset;
    reg direction;
    reg Enable;
    reg selection;

    // Outputs
    wire [4:0] count;

    // Instantiate the Counter module
    Counter counter(
        .clk(clk),
        .reset(reset),
        .direction(direction),
        .Enable(Enable),
        .selection(selection),
        .count(count)
    );

    // Clock generation
    always #5 clk = ~clk;

    // Initialize inputs
    initial begin
        clk = 0;
        reset = 1;
        direction = 0;
        Enable = 0;
        selection = 0;
        #10 reset = 0;
    end

    // Stimulus
    always begin
        #20 Enable = 1;  // Enable the counter
        #100 Enable = 0; // Disable the counter

        #20 reset = 1;  // Assert reset
        #10 reset = 0; // Deassert reset

        #20 direction = 1; // Set direction to 1 (up)
        #50 direction = 0; // Set direction to 0 (down)

        #20 selection = 1; // Set selection to 1 (Fibonacci)
        #50 selection = 0; // Set selection to 0 (Prime)

        #200 $finish; // End simulation
    end

endmodule
```

Results

I created a new waveform file to test the circuit using the testbench and got the following.



After all the coding and designing, the waveform that I got had a problem in which the counter can not exceed the number zero, I tried many and many times in which 2 days were wasted only trying to solve this problem and I could not.

Unfortunately, I came to the deadline of handing this project and I still have some problems with the waveform that I still do not know the real problem to it.

Conclusion and Future Works

In conclusion, this project aimed to design and implement a special counter circuit capable of counting two different sequences: the Prime Numbers Sequence and the Fibonacci Sequence. The counter operates in both up and down modes, depending on the input values, and utilizes T-Flip Flops and combination logic for its functionality. The design philosophy was based on analyzing the requirements for each sequence and determining the appropriate bit size for the counter. Equations were obtained using a website to simplify the design process.

However, during the testing phase, an issue was encountered where the counter could not exceed zero. Despite multiple attempts to solve this problem, the root cause could not be identified within the time I had, leading to an unresolved issue in the waveform.

In future works, further investigation is recommended to identify and address the problem with the waveform. Additionally, it would be beneficial to optimize the counter design and explore alternative approaches to improve its functionality and performance. This could involve refining the equations or exploring different circuit architectures to overcome the limitations encountered in this project.

Overall, this project provided valuable insights into the design and implementation of special counter circuits. Despite the unresolved issue, it served as a learning experience and demonstrated the importance of thorough testing and debugging in complex projects.