Faculty of Engineering and Technology

Electrical and Computer Engineering Department

Advanced Digital Design ENCS3310

Project Report

Sondos Shahin 1200166

Instructor: Dr. Abdallatif Abuissa

Section 2

Table of Contents

Table of figures

Introduction

The aim of this project is to design and build a special counter structurally using T flip flops and other combinational logic. This counter counts the first 11 number of both the prime numbers and Fibonacci sequence, based on a specific input. It also counts the both sequences in up and down depending on another input.

The first 11 prime numbers are:

2,3,5,7,11,13,17,19,23,29,31

The first 11 number in Fibonacci sequence are:

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

Design philosophy

This circuit is basically controlled by the clock, reset and enable inputs. This counter has 4 tasks to do, count prime numbers up, prime numbers down, Fibonacci sequence up and Fibonacci sequence down. As a result, a 4x1 mux is needed in this design as it has 2 selection lines to determine the wanted count method.

The largest number we need to reach in this counter is 55, which require 6 binary bits to represent it. So, 6 4x1 mux and 6 t flip flops are needed to complete the design.

Every mux is controlled by the enable (synchronous input), whereas the flip flops are controlled by the edge of both the clock and reset inputs(asynchronous).

For each case, a state table was written and the equations of the flip flops were found.

4x1 mux with enable

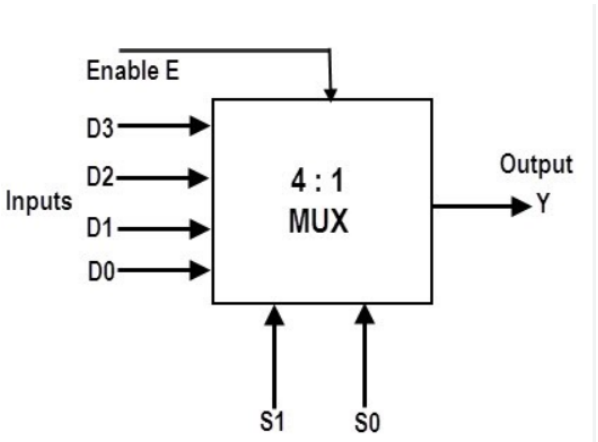

Figure 1- 4x1 mux block

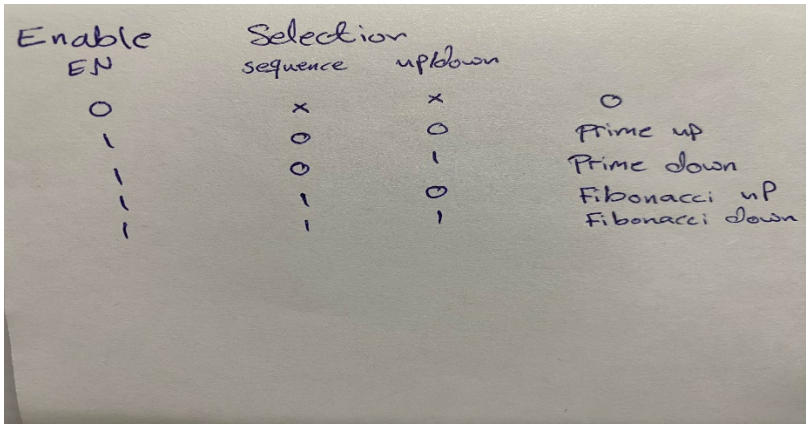

Figure 2- 4x1 mux state table

```
module mux4x1 (in0, in1, in2, in3, en, s0, s1, out);
    input in0, in1, in2, in3, en, s0, s1;
    output reg out;
    always @ (in0 or in1 or in2 or in3 or en or s0 or s1 or out)  begin
        if (en == 1) begin
            case ({s0,s1})
                2'b00: out = in0;
                2'b01: out = in1;
                2'b10: out = in2;
                2'b11: out = in3;
                default: out = 1'b0;
            endcase
        end
        else
            out = 1'b0;
        end
endmodule
```

Figure 3- 4x1 mux code

T- flip flop

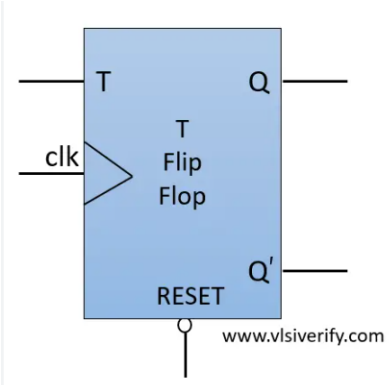

Figure 4- t flip flop block

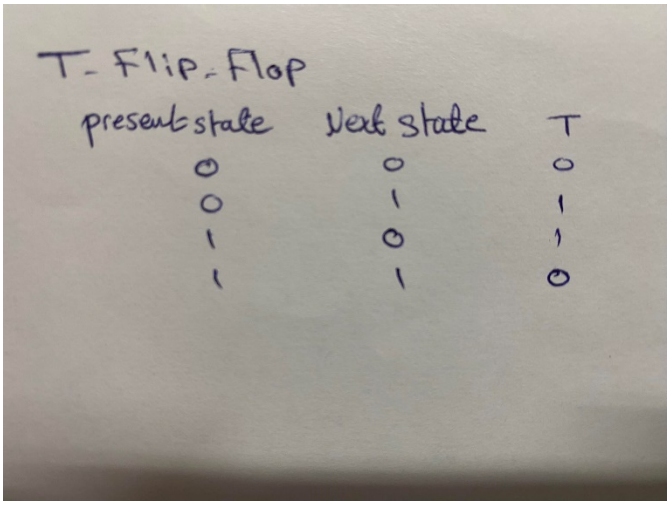

Figure 5 - t flip flop state table

```
module tff (Q, T, clk, rst);    //t flip flop
    output reg Q;
    input T, clk, rst;
    always @ (posedge clk or negedge rst)
        if (~rst)
            Q = 1'b0;
        else
            Q = Q ^ T;
endmodule
```

Figure 6- t flip flop code

Circuit design

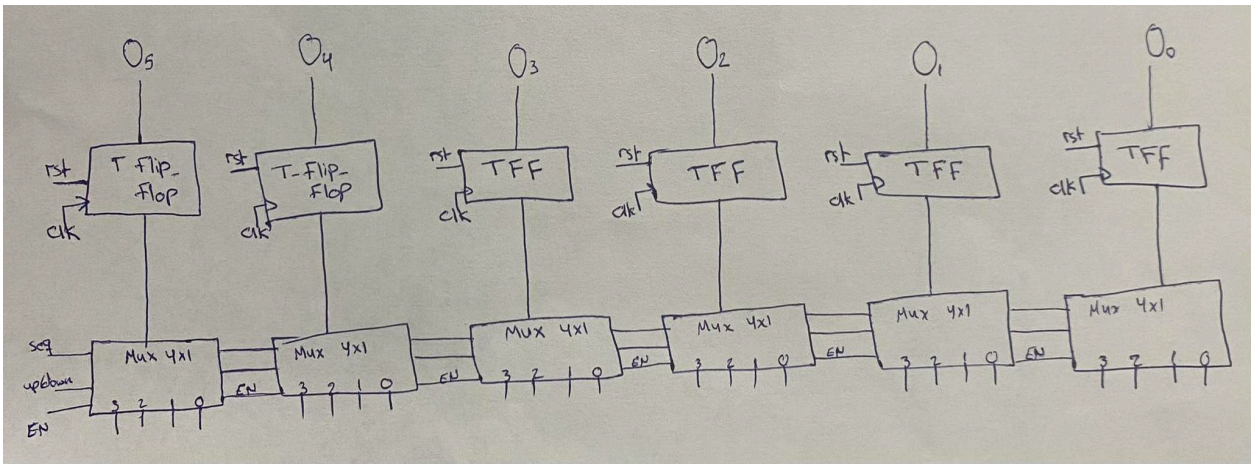

Figure 7- circuit design

Counting prime numbers up

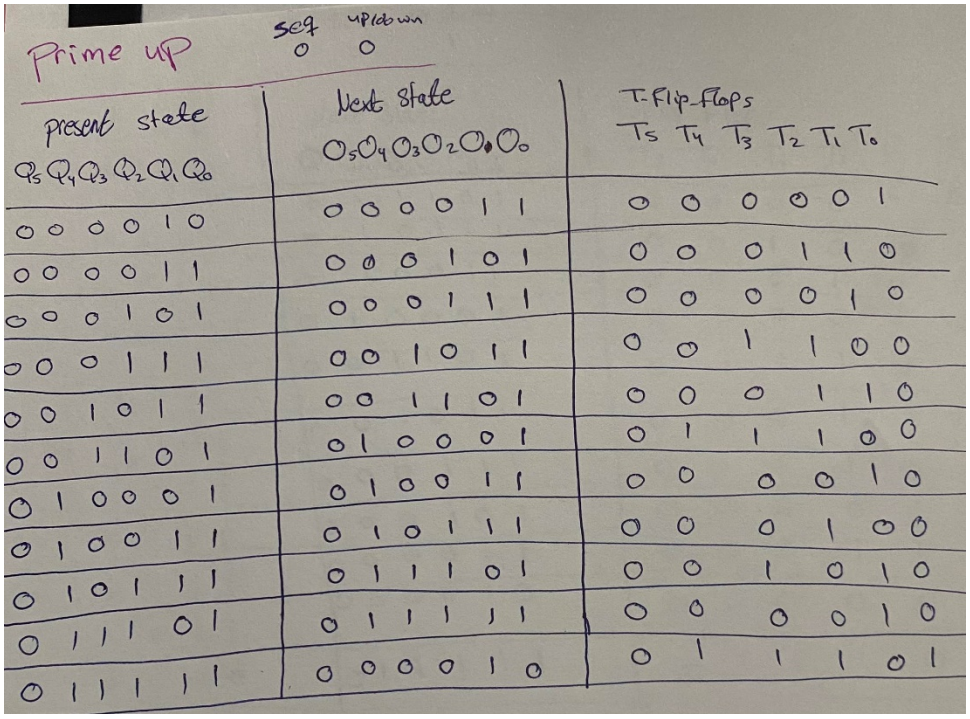

Figure 8- prime up state table

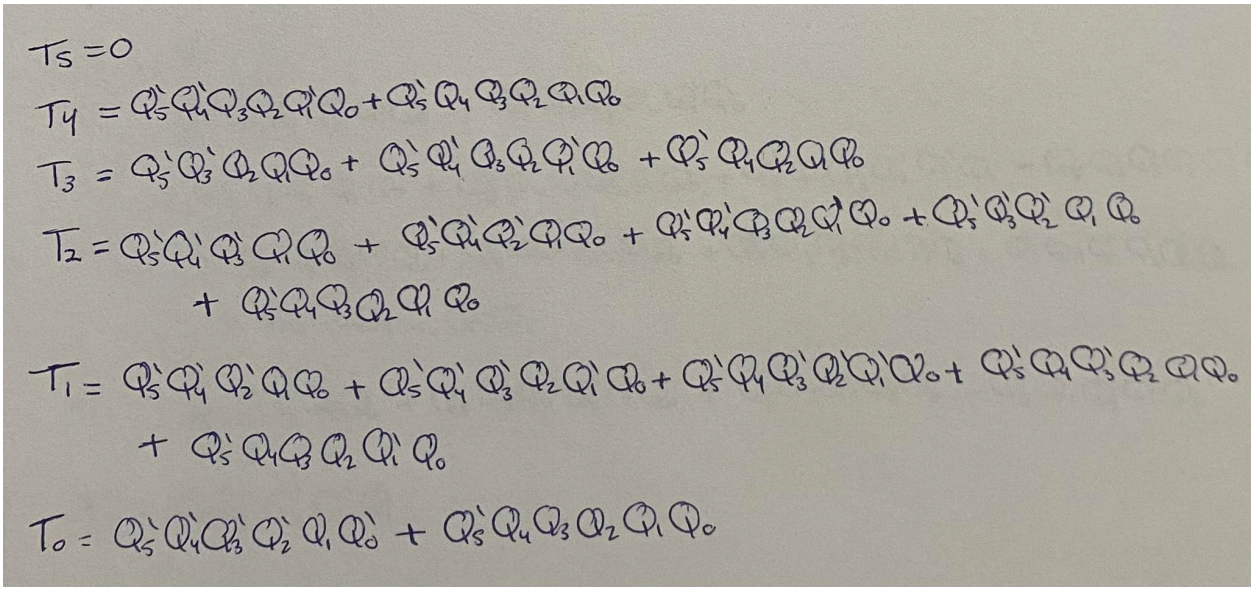

Figure 9- prime up equations

Counting prime numbers down

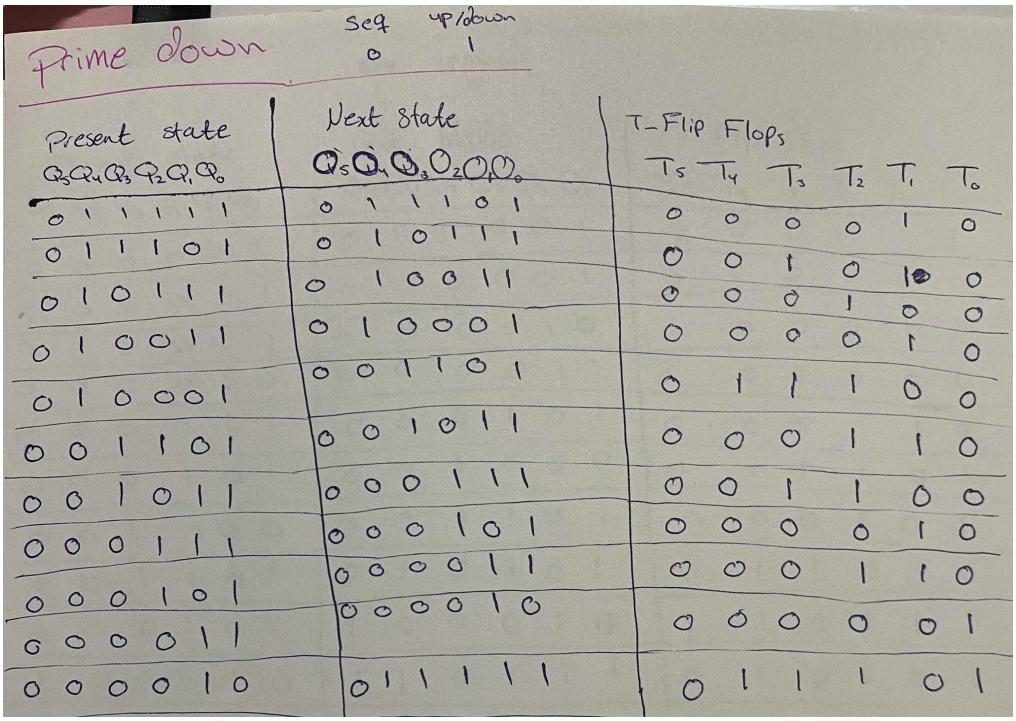

Figure 10- prime down state table

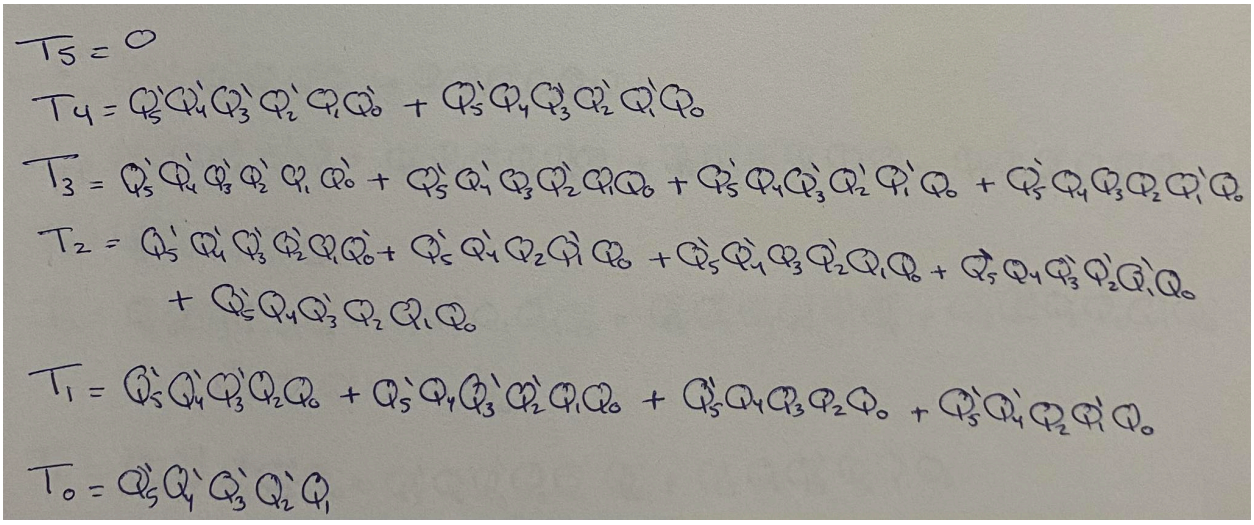

Figure 11 - prime numbers equations

Counting Fibonacci sequence up

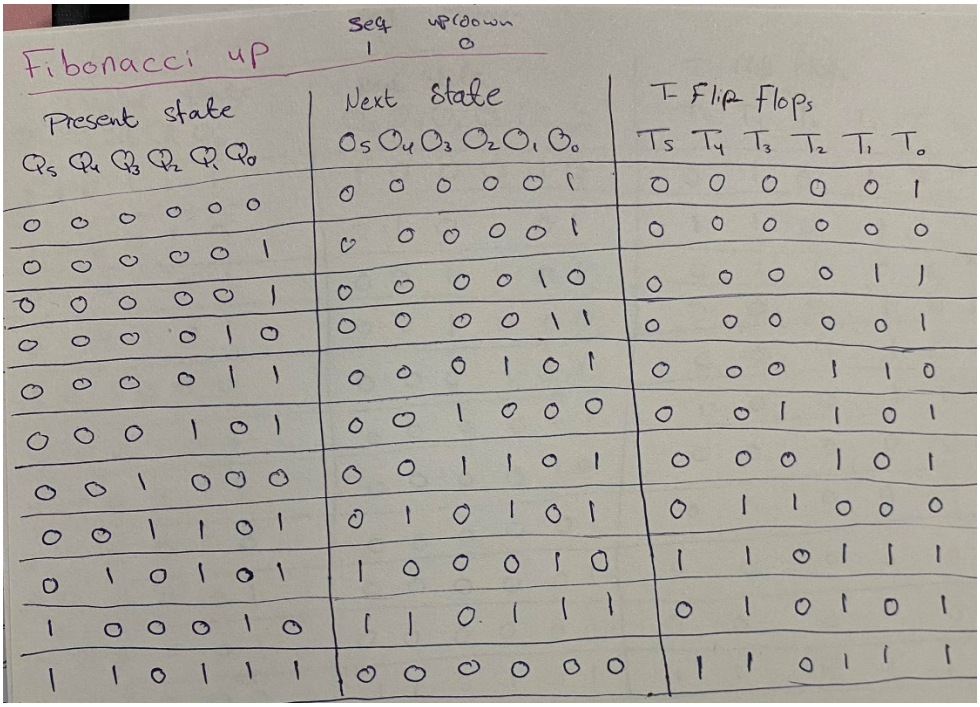

Figure 12- Fibonacci up state table

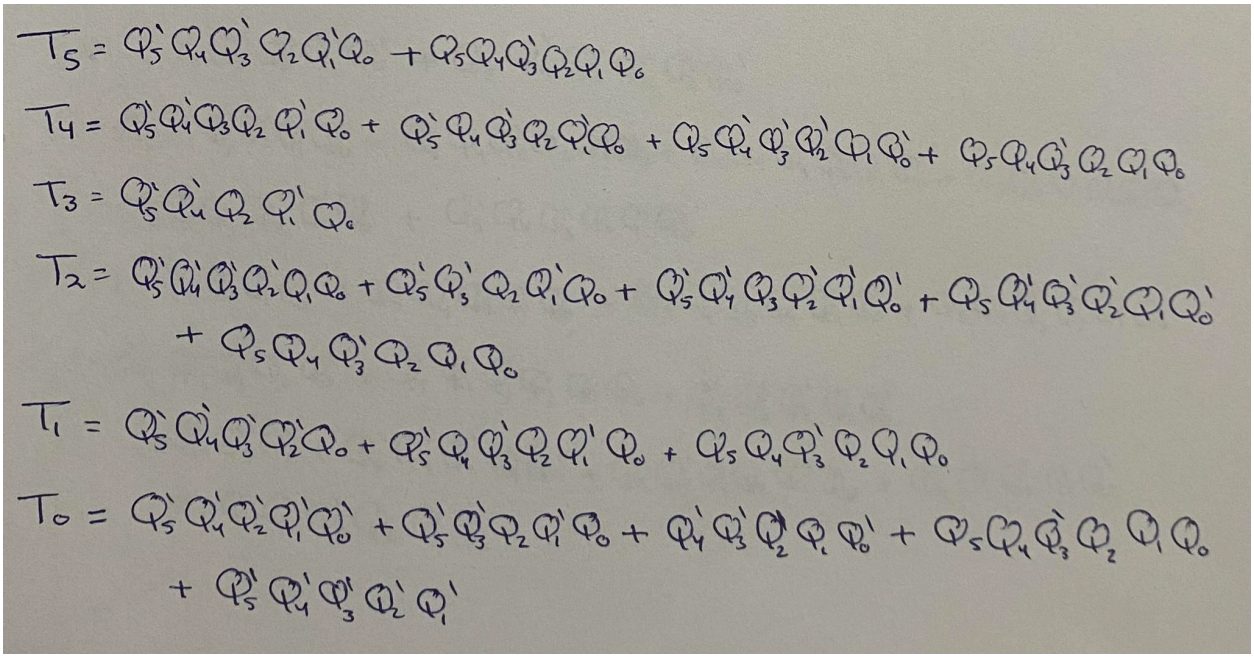

Figure 13- Fibonacci up equations

Counting Fibonacci sequence down

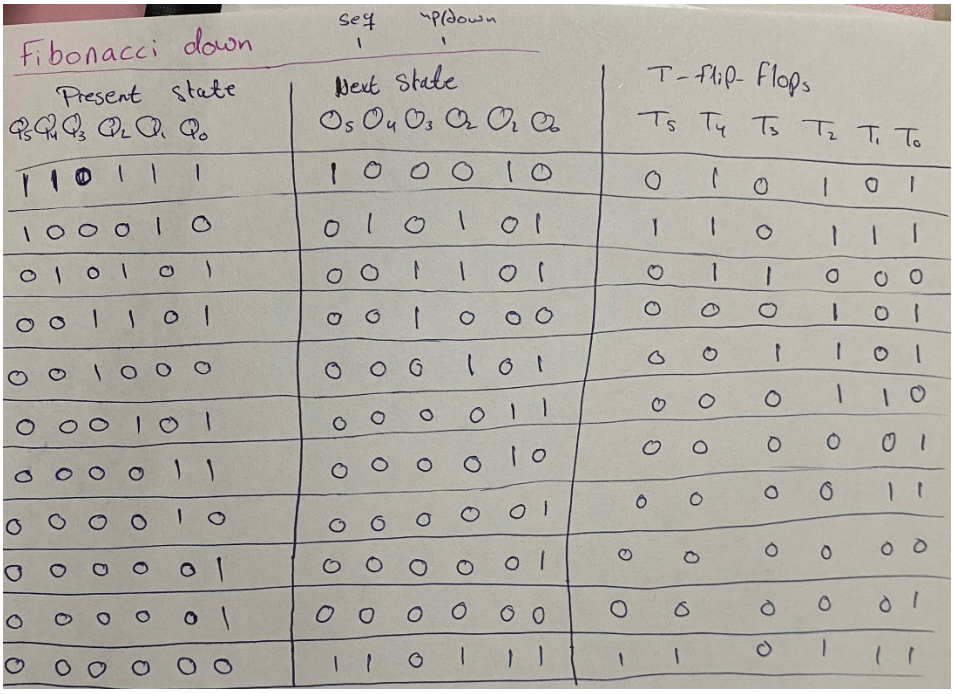

Figure 14-Fibonacci down state table

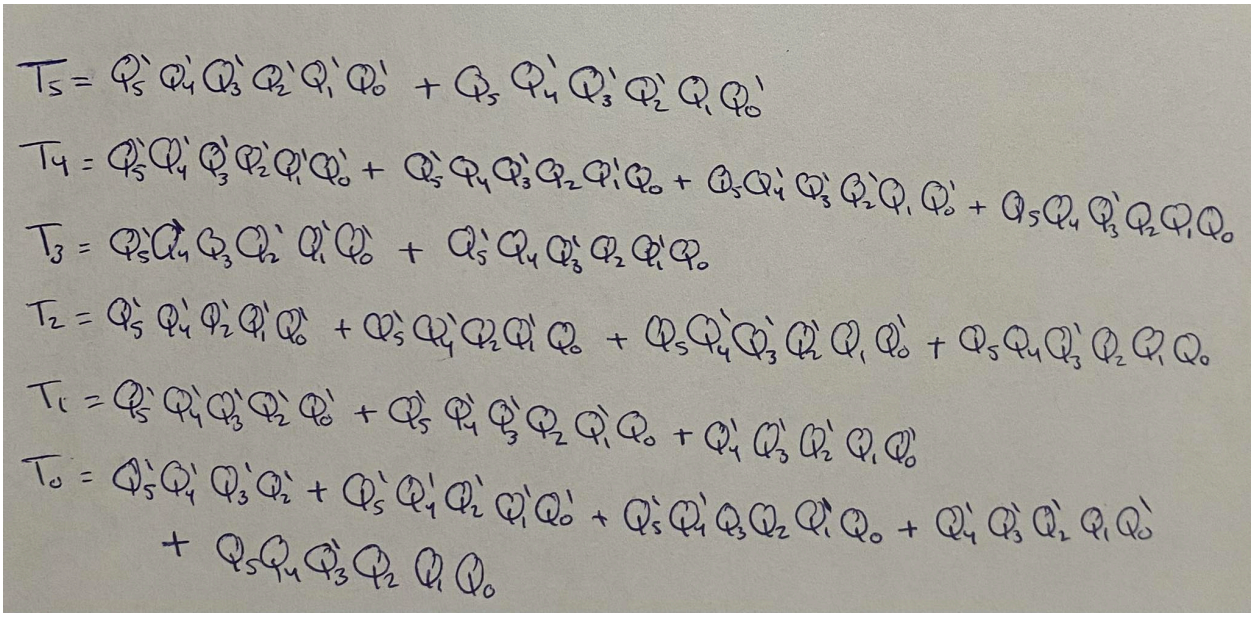

Figure 15- Fibonacci down equations

Code and results

Mux & flip flop

```
module tff (Q, T, clk, rst);    //t flip flop
    output reg Q;
    input T, clk, rst;
    always @ (posedge clk or negedge rst)
        if (~rst)
            Q = 1'b0;
        else
            Q = Q ^ T;
endmodule


module mux4x1 (in0, in1, in2, in3, en, s0, s1, out);
    input in0, in1, in2, in3, en, s0, s1;
    output reg out;
    always @ (in0 or in1 or in2 or in3 or en or s0 or s1 or out)  begin
        if (en == 1) begin
            case ({s0,s1})
                2'b00: out = in0;
                2'b01: out = in1;
                2'b10: out = in2;
                2'b11: out = in3;
                default: out = 1'b0;
            endcase
        end
        else
            out = 1'b0;
        end
endmodule

```

Figure 16- mux & tff code

System

```
module sys (q5,q4,q3,q2,q1,q0,clk,rst,en,seq,ud,o5,o4,o3,o2,o1,o0);
    input q5,q4,q3,q2,q1,q0,clk,rst,en,seq,ud;
    output o5,o4,o3,o2,o1,o0;
    wire t5,t4,t3,t2,t1,t0;

    //pu -> prime up
    //pd -> prime down
    //fu -> fibonacci up
    //fd -> fibonacci down

    //mux5
    //bit 5 - prime up & down = 0
    wire outfu51, outfu52,outfu5 , outfd5, outfd51, outfd52;
    and fu51 (outfu51,~q5, q4, ~q3, q2, ~q1, q0);
    and fu52 (outfu52, q5, q4, ~q3, q2, q1, q0);
    or fu5 (outfu5, outfu51, outfu52);   //bit 5 - fibonacci up

    and fd51 (outfd1,~q5, ~q4, ~q3, ~q2, ~q1, ~q0);
    and fd52 (outfd2, q5, ~q4, ~q3, ~q2, q1, ~q0);
    or fd5 (outfd5, outfd51, outfd52);   //bit 5 - fibonacci down

    mux4x1 mux5 (1'b0, 1'b0, outfu5, outfd5, en,seq,ud, t5);

```

```
    //mux4
    wire outpu4,outpu41,outpu42,outpd4, outpd41, outpd42  ,outfu41, outfu42, outfu43, outfu44,outfu4 , outfd4, outfd41, outfd42, outfd43, outfd44;
    and pu41 (outpu41,~q5, ~q4, q3, q2, ~q1, q0);
    and pu42 (outpu42, ~q5, q4, q3, q2, q1, q0);
    or pu4 (outpu4, outpu41, outpu42);   //bit 4 - prime up

    and pd41 (outpd41,~q5, ~q4, ~q3, ~q2, q1, ~q0);
    and pd42 (outpd42, ~q5, q4, ~q3, ~q2, ~q1, q0);
    or pd4 (outpd4, outpd41, outpd42);   //bit 4 - prime down

    and fu41 (outfu41,~q5, ~q4, q3, q2, ~q1, q0);
    and fu42 (outfu42, ~q5, q4, ~q3, q2, ~q1, q0);
    and fu43 (outfu43,q5, ~q4, ~q3, ~q2, q1, ~q0);
    and fu44 (outfu44, q5, q4, ~q3, q2, q1, q0);
    or fu4 (outfu4, outfu41, outfu42, outfu43, outfu44);     //bit 4 - fibonacci up

    and fd41 (outfd41,~q5, ~q4, ~q3, ~q2, ~q1, ~q0);
    and fd42 (outfd42, ~q5, q4, ~q3, q2, ~q1, q0);
    and fd43 (outfd43,q5, ~q4, ~q3, ~q2, q1, ~q0);
    and fd44 (outfd44, q5, q4, ~q3, q2, q1, q0);
    or fd4 (outfd4, outfd41, outfd42, outfd43, outfd44);     //bit 4 - fibonacci down

    mux4x1 mux4 (outpu4, outpd4, outfu4, outfd4, en, seq, ud, t4);
```

```
    //mux3
    wire outpu3,outpu31,outpu32, outpu33,outpd3, outpd31, outpd32, outpd33, outpd34 ,outfu3 , outfd3, outfd31, outfd32;
    and pu31 (outpu31,~q5,~q3,q2,q1,q0);
    and pu32 (outpu32,~q5,~q4,q3,q2,~q1,q0);
    and pu33 (outpu33,~q5,q4,q2,q1,q0);
    or pu3 (outpu3, outpu31, outpu32,outpu33);   //bit 3 - prime up

    and pd31 (outpd31,~q5, ~q4, ~q3, ~q2, q1, ~q0);
    and pd32 (outpd32, ~q5, ~q4, q3, ~q2, q1, q0);
    and pd33 (outpd33,~q5, q4, ~q3, ~q2, ~q1, q0);
    and pd34 (outpd34, ~q5, q4, q3, q2, ~q1, q0);
    or pd3 (outpd3, outpd31, outpd32, outpd33, outpd34);     //bit 3 - prime down

    and fu3 (outfu3,~q5, ~q4, q2, ~q1, q0);          //bit 3 - fibonacci up

    and fd31 (outfd31,~q5, ~q4, q3, ~q2, ~q1, ~q0);
    and fd32 (outfd32, ~q5, q4, ~q3, q2, ~q1, q0);
    or fd3 (outfd3, outfd31, outfd32);   //bit 3 - fibonacci down

    mux4x1 mux3 (outpu3, outpd3, outfu3, outfd3, en, seq, ud, t3);
```

```
    //mux2
    wire outpu2, outpu21, outpu22, outpu23, outpu24,outpu25, outpd2, outpd21, outpd22,outpd23, outpd24, outpd25,
    outfu2, outfu21, outfu22, outfu23, outfu24, outfu25,outfd2, outfd21, outfd22, outfd23, outfd24;

    and pu21 (outpu21,~q5, ~q4, ~q3, q1, q0);
    and pu22 (outpu22, ~q5, ~q4, ~q2, q1, q0);
    and pu23 (outpu23,~q5, ~q4, q3, q2, ~q1, q0);
    and pu24 (outpu24, ~q5, ~q3, ~q2, q1, q0);
    and pu25 (outpu25, ~q5, q4, q3, q2, q1, q0);
    or pu2 (outpu2, outpu21, outpu22, outpu23, outpu24,outpu25);     //bit 2 - prime up

    and pd21 (outpd21,~q5, ~q4, ~q3, ~q2, q1, ~q0);
    and pd22 (outpd22, ~q5, ~q4, q2, ~q1, q0);
    and pd23 (outpd23,~q5, ~q4, q3, ~q2, q1, ~q0);
    and pd24 (outpd24, ~q5, q4, ~q3, ~q2, ~q1, q0);
    and pd25 (outpd25,~q5, q4, ~q3, q2, q1, q0);
    or pd2 (outpd2, outpd21, outpd22, outpd23, outpd24,outpd25);        //bit 2 - prime down

    and fu21 (outfu21,~q5, ~q4, ~q3, ~q2, q1, q0);
    and fu22 (outfu22, ~q5, ~q3, q2, ~q1, q0);
    and fu23 (outfu23, ~q5, ~q4, q3, ~q2, ~q1, ~q0);
    and fu24 (outfu24, q5, ~q4, ~q3, ~q2, q1, ~q0);
    and fu25 (outfu25, q5, q4, ~q3, q2, q1, q0);
    or fu2 (outfu2, outfu21, outfu22, outfu23, outfu24, outfu25);    //bit 2 - fibonacci up

    and fd21 (outfd21,~q5, ~q4, ~q2, ~q1, ~q0);
    and fd22 (outfd22, ~q5, ~q4, q2, ~q1, q0);
    and fd23 (outfd23,q5, ~q4, ~q3, ~q2, q1, ~q0);
    and fd24 (outfd24, q5, q4, ~q3, q2, q1, q0);
    or fd2 (outfd2, outfd21, outfd22, outfd23, outfd24);     //bit 2 - fibonacci down

    mux4x1 mux2 (outpu2, outpd2, outfu2, outfd2, en, seq, ud, t2);
```

```
    //mux1
    wire outpu1, outpu11, outpu12, outpu13, outpu14,outpu15, outpd1, outpd11, outpd12, outpd13, outpd14,
    outfu1, outfu11, outfu12, outfu13, outfd1, outfd11, outfd12, outfd13;


    and pu11 (outpu11,~q5, ~q4, ~q2, q1, q0);
    and pu12 (outpu12, ~q5, ~q4, ~q3, q2, ~q1, q0);
    and pu13 (outpu13,~q5, q4, ~q3, ~q2, ~q1, q0);
    and pu14 (outpu14, ~q5, q4, ~q3, q2, q1, q0);
    and pu15 (outpu15, ~q5, q4, q3, q2, ~q1, q0);
    or pu1 (outpu1, outpu11, outpu12, outpu13, outpu14,outpu15);     //bit 1 - prime up

    and pd11 (outpd11,~q5, ~q4, ~q3, q2, q0);
    and pd12 (outpd12, ~q5, q4, ~q3, ~q2, q1, q0);
    and pd13 (outpd13,~q5, q4, q3, q2, q0);
    and pd14 (outpd14, ~q5, ~q4, q2, ~q1, q0);
    or pd1 (outpd1, outpd11, outpd12, outpd13, outpd14);        //bit 1 - prime down

    and fu11 (outfu11,~q5, ~q4, ~q3, ~q2, q0);
    and fu12 (outfu12, ~q5, q4, ~q3, q2, ~q1, q0);
    and fu13 (outfu13, q5, q4, ~q3, q2, q1, q0);
    or fu1 (outfu1, outfu11, outfu12, outfu13);  //bit 2 - fibonacci up

    and fd11 (outfd11,~q5, ~q4, ~q3, ~q2, ~q0);
    and fd12 (outfd12, ~q5, ~q4, ~q3, q2, ~q1, q0);
    and fd13 (outfd13, ~q4, ~q3, ~q2, q1, ~q0);
    or fd1 (outfd1, outfd11, outfd12, outfd13);  //bit 1 - fibonacci down

    mux4x1 mux1 (outpu1, outpd1, outfu1, outfd1, en, seq, ud, t1);
```

```
    //mux0
    wire outpu0, outpu01, outpu02, outpd0, outfu0, outfu01, outfu02, outfu03, outfu04, outfu05,
     outfd0, outfd01, outfd02, outfd03, outfd04, outfd05;


    and pu01 (outpu01,~q5, ~q4, ~q3, ~q2, q1, ~q0);
    and pu02 (outpu02, ~q5, q4, q3, q2, q1, q0);
    or pu0 (outpu0, outpu01, outpu02);   //bit 0 - prime up

    and pd0 (outpd0,~q5, ~q4, ~q3, ~q2, q1); //bit 0 - prime down



    and fu01 (outfu01,~q5, ~q4, ~q2, ~q1, ~q0);
    and fu02 (outfu02, ~q5, ~q3, q2, ~q1, q0);
    and fu03 (outfu03, ~q4, ~q3, ~q2, q1, ~q0);
    and fu04 (outfu04, q5, q4, ~q3, q2, q1, q0);
    and fu05 (outfu05, ~q5, ~q4, ~q3, ~q2, ~q1);
    or fu0 (outfu0, outfu01, outfu02, outfu03, outfu04, outfu05);    //bit 0 - fibonacci up

    and fd01 (outfd01,~q5, ~q4, ~q3, ~q2);
    and fd02 (outfd02, ~q5, ~q4, ~q2, ~q1, ~q0);
    and fd03 (outfd03, ~q5, ~q4, q3, q2, ~q1, q0);
    and fd04 (outfd04, ~q4, ~q3, ~q2, q1, ~q0);
    and fd05 (outfd05, q5, q4, ~q3, q2, q1, q0);
    or fd0 (outfd0, outfd01, outfd02, outfd03, outfd04, outfd05);    //bit 0 - fibonacci down

    mux4x1 mux0 (outpu0, outpd0, outfu0, outfd0, en, seq, ud, t0);
```

```


      //connect the output of every mux with the input of a t flip flop
      tff tf5 (o5,t5, clk, rst);
      tff tf4 (o4,t4, clk, rst);
      tff tf3 (o3,t3, clk, rst);
      tff tf2 (o2,t2, clk, rst);
      tff tf1 (o1,t1, clk, rst);
      tff tf0 (o0,t0, clk, rst);
endmodule

```

Figure 17- system code

Test bench

```
    //test bench
module test () ;
    reg q5,q4,q3,q2,q1,q0,clk,rst,en,seq,ud;
    wire o5,o4,o3,o2,o1,o0;
    sys tsys (q5,q4,q3,q2,q1,q0,clk,rst,en,seq,ud,o5,o4,o3,o2,o1,o0);
    initial begin
        $monitor("time %0d  %b %b %b %b %b %b", $time, o5,o4,o3,o2,o1,o0);
        rst = 0; clk = 0;
        #10 rst = 1;
        repeat (100)
        #20 clk = ~clk;
    end
    initial begin
        en = 1'b0;
        #20 en = 1'b1;
        //test prime up
        seq= 1'b0; ud = 1'b0;
        {q5,q4,q3,q2,q1,q0} = 6'b000000;
        repeat (31)
        #10 {q5,q4,q3,q2,q1,q0} = {q5,q4,q3,q2,q1,q0} + 6'b000001;
         //test prime down
        seq= 1'b0; ud = 1'b1;
        {q5,q4,q3,q2,q1,q0} = 6'b000000;
        repeat (31)
        #10 {q5,q4,q3,q2,q1,q0} = {q5,q4,q3,q2,q1,q0} + 6'b000001;
         //test fibonacci up
        seq= 1'b1; ud = 1'b0;
        {q5,q4,q3,q2,q1,q0} = 6'b000000;
        repeat (63)
        #10 {q5,q4,q3,q2,q1,q0} = {q5,q4,q3,q2,q1,q0} + 6'b000001;
          //test fibonacci down
        seq= 1'b1; ud = 1'b1;
        {q5,q4,q3,q2,q1,q0} = 6'b000000;
        repeat (63)
        #10 {q5,q4,q3,q2,q1,q0} = {q5,q4,q3,q2,q1,q0} + 6'b000001;
    end
endmodule
```

Figure 18- test bench code

Waveform

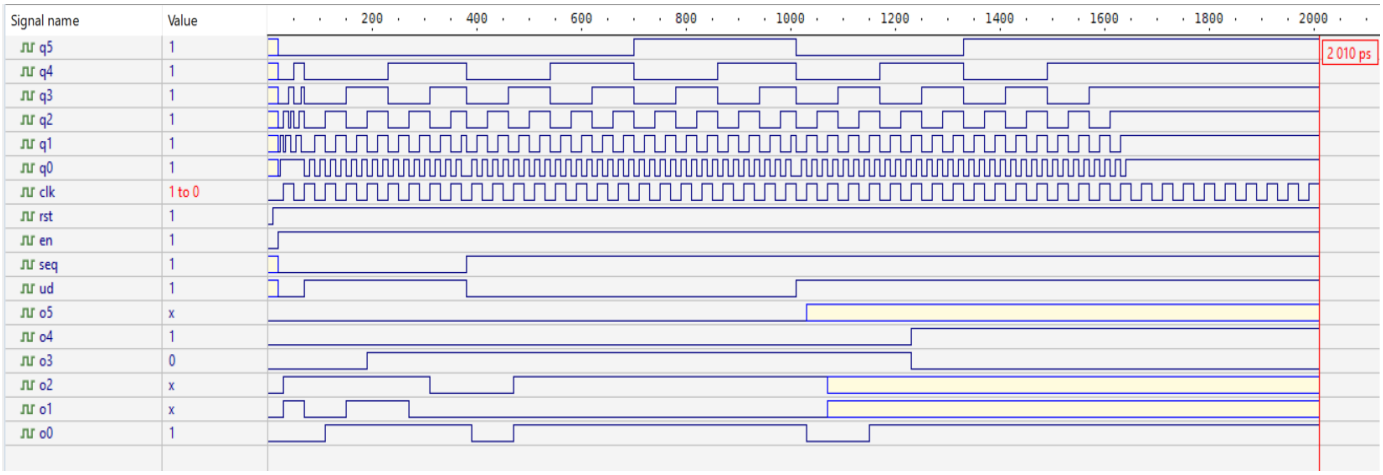

Figure 19-waveform

Testbench 2

```
214     //test bench
215 module test () ;
216     reg q5,q4,q3,q2,q1,q0,clk,rst,en,seq,ud;
217     wire o5,o4,o3,o2,o1,o0;
218     sys tsys (q5,q4,q3,q2,q1,q0,clk,rst,en,seq,ud,o5,o4,o3,o2,o1,o0);
219     initial begin
220         rst = 0; clk = 0;
221         #10 rst = 1;
222         repeat (100)
223         #20 clk = ~clk;
224     end
225     initial begin
226         en = 1'b0;
227         #20 en = 1'b1;
228 
229         //test prime up
230         seq= 1'b0; ud = 1'b0;
231         {q5,q4,q3,q2,q1,q0} = 6'b000000;    //test generator
232         if ({o5,o4,o3,o2,o1,o0} != 6'b000011)  $display ("count failed");       //result analyzer
233         #5 {q5,q4,q3,q2,q1,q0} = 6'b000011;   //test generator
234         if ({o5,o4,o3,o2,o1,o0} != 6'b000101)  $display ("count failed");        //result analyzer
235         #5 {q5,q4,q3,q2,q1,q0} = 6'b000101;   //test generator
236         if ({o5,o4,o3,o2,o1,o0} != 6'b000111)  $display ("count failed");       //result analyzer
237         #5 {q5,q4,q3,q2,q1,q0} = 6'b000111;  //test generator
238         if ({o5,o4,o3,o2,o1,o0} != 6'b001011)  $display ("count failed");       //result analyzer
239         #5 {q5,q4,q3,q2,q1,q0} = 6'b001011;  //test generator
240         if ({o5,o4,o3,o2,o1,o0} != 6'b001101)  $display ("count failed");       //result analyzer
241         #5 {q5,q4,q3,q2,q1,q0} = 6'b001101;   //test generator
242         if ({o5,o4,o3,o2,o1,o0} != 6'b010001)  $display ("count failed");       //result analyzer
243         #5 {q5,q4,q3,q2,q1,q0} = 6'b010001;   //test generator
244         if ({o5,o4,o3,o2,o1,o0} != 6'b010011)  $display ("count failed");       //result analyzer
245         #5 {q5,q4,q3,q2,q1,q0} = 6'b010011;   //test generator
246         if ({o5,o4,o3,o2,o1,o0} != 6'b010111)  $display ("count failed");       //result analyzer
247         #5 {q5,q4,q3,q2,q1,q0} = 6'b010111;   //test generator
248         if ({o5,o4,o3,o2,o1,o0} != 6'b011101)  $display ("count failed");       //result analyzer
249         #5 {q5,q4,q3,q2,q1,q0} = 6'b011101;   //test generator
250         if ({o5,o4,o3,o2,o1,o0} != 6'b011101)  $display ("count failed");       //result analyzer
251         #5 {q5,q4,q3,q2,q1,q0} = 6'b011111;   //test generator
252         if ({o5,o4,o3,o2,o1,o0} != 6'b011111)  $display ("count failed");       //result analyzer
253         end
254 endmodule
255 
```

```
Console
◦ run
◦ # KERNEL: count failed
◦ # KERNEL: count failed
◦ # KERNEL: count failed
◦ # KERNEL: count failed
◦ # KERNEL: count failed
◦ # KERNEL: count failed
◦ # KERNEL: count failed
◦ # KERNEL: count failed
◦ # KERNEL: count failed
◦ # KERNEL: count failed
◦ # KERNEL: count failed
◦ # KERNEL: Simulation
  has finished. There are
  no more test vectors to
  simulate.
◦
◦
◦
>
Console
```

Figure 20-testbench2

Conclusion

As shown in the figures above, it is clear that there is an error in the functionality of the circuit, since the expected output is not the same as the actual output of the circuit.

The problem might be in the equations that I wrote from the state tables, or it might be in the design itself.