Birzeit University
Faculty of Engineering and Technology
Department of Electrical and Computer Engineering

Project
Advanced Digital System Design

Prepared by
Name: Hasan Qarmash
ID: 1210611

Supervised by
Dr.Abdallatif Abuissa

Birziet-2023

Abstract

The aim of this project to be familiar with Verilog HDL and Understanding the working mechanism of the counters, specifically the prime number system and the Fibonacci system

Contents

List of Figures

1 project course

In this project, we will build a structural circuit using T-Flip Flops and combination logic, that can count either Prime numbers or Fibonacci depends on a value of an input. Also, it can count them either up or down depends on the value of another input. The counter should count the first 11 numbers of any sequence Figure 1: Plan project as shown in Figure 1 , in two different stages by using Aldec Active-HDL student Edition to write Verilog codes,

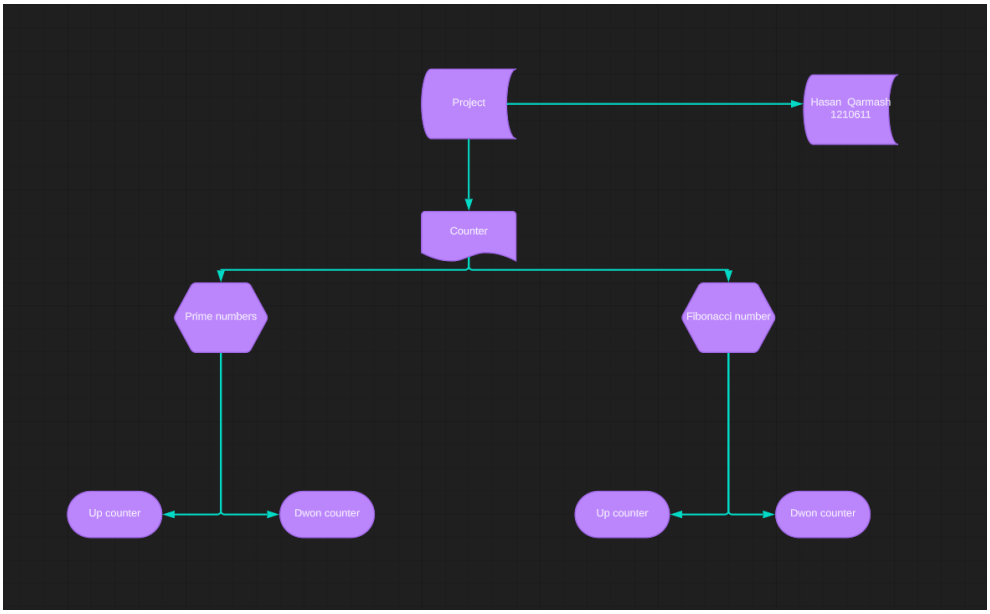


Figure 1: Plan project

- **prime number UP:2,3,5,7,11,13,17,19,23,29,31**
- **prime number Down:31,29,23,19,17,13,11,7,5,5,3,2**
- **Fibonacci number UP:0,1,1,2,3,5,8,13,21,34,55**
- **Fibonacci number Down:55,34,21,13,8,5,3,2,1,1,0**

1.1 My way of thinking

I divided the counters into four types in order to facilitate the design process, especially since in Fibonacci system there is 6-bit and with the entry of ascending and descending it becomes 7 and with the entry the system determines if it is Fibonacci or prime it becomes I have 8 and this is outside the framework of my ability to represent. so The types is:
prime number UP
prime number Down
Fibonacci number UP
Fibonacci number Down

1.1.1 T-FlipFlop

A T-flip flop, also known as a toggle flip flop, is a type of sequential logic circuit that can store and toggle its state based on the input signal. It has one input, denoted as T, and two outputs: Q (the current state) and Q' (the complement of the current state). The behavior of a T-flip flop is determined by its characteristic table, which specifies the output states based on the current input and the previous state
Hint: The T-flip flop active highe That mean when reset =1 result Q=0;

```
134 module T_ff(t,clk,reset,q);
135
136 input t,clk,reset;
137 output q;
138 reg q;
139 always @(posedge clk or posedge reset) begin
140 if(reset) begin
141 q=1'b0;
142 end
143 else
144     begin
145 q<=q^t;
146 end
147 end
148 endmodule
149
```

Figure 2: code T-FLIPFLOP

1.1.2 prime number UP

A prime number is a natural number greater than 1 that cannot be divided evenly by any other natural number except for 1 and itself. In other words, a prime number has exactly two distinct positive divisors: 1 and the number itself.

Here are the first 11 prime numbers:

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

```
// Truth table for prime number UP
    //Present state          Next state                    T-Flip Flop
//  Q4  Q3  Q2  Q1  Q0     Q4  Q3  Q2  Q1  Q0        T4  T3  T2  T1  T0
//  0   0   0   1   0      0   0   0   1   1   0     0   0   0   1
//  0   0   0   1   1      0   0   1   0   1   0     0   1   1   0
//  0   0   1   0   1      0   0   1   1   1   0     0   0   1   0
//  0   0   1   1   1      0   1   0   1   1   0     1   1   0   0
//  0   1   0   1   1      0   1   1   0   1   0     0   1   1   0
//  0   1   1   0   1      1   0   0   0   1   1     1   1   0   0
//  1   0   0   0   1      1   0   0   1   1   0     0   0   1   0
//  1   0   0   1   1      1   0   1   1   1   0     0   1   0   0
//  1   0   1   1   1      1   1   1   0   1   0     1   0   1   0
//  1   1   1   0   1      1   1   1   1   1   0     0   0   1   0
//  1   1   1   1   1      0   0   0   1   0   1     1   1   0   1
```

Figure 3: Truth Table prime up counter

```
157
158 module PrimeUP(clk,Q,reset);
159     input clk,reset;
160     output [4:0]Q;
161     wire [4:0]D;
162     assign D[0]=~Q[0]|(Q[3] & Q[2] & Q[1]);
163     assign D[1]=(~Q[3]& ~Q[1])|(Q[4] & ~Q[1]) |(~Q[4] & ~Q[2] & Q[0])|(Q[4] & ~Q[3] & Q[2]);
164     assign D[2]=(~Q[4]& Q[3])|(Q[3] & Q[1]) |(~Q[4] & Q[1] & Q[0]) | (~Q[2] & Q[1] & Q[0]);
165     assign D[3]=(Q[2]& Q[1])|(~Q[4] & Q[3] & ~Q[1]);
166     assign D[4]=(~Q[4] & Q[3] & ~Q[1])|(Q[3] & Q[2] & Q[1]);
167
168     T_ff U0(D[0],clk,reset,Q[0]);
169     T_ff U1(D[1],clk,reset,Q[1]);
170     T_ff U2(D[2],clk,reset,Q[2]);
171     T_ff U3(D[3],clk,reset,Q[3]);
172     T_ff U4(D[4],clk,reset,Q[4]);
173
174 endmodule
```

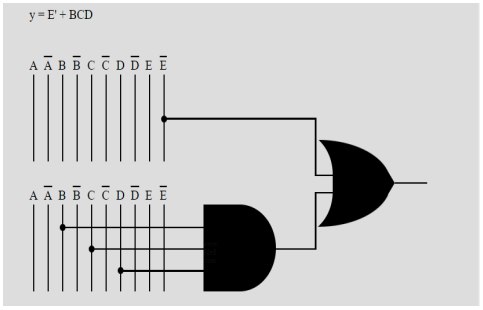
Figure 4: code prime up

```
module testPrimeNumberUP;
    reg clk,reset;
    wire [4:0]Q;
    PrimeUP  R0(clk,Q,reset);
    initial
        begin
        reset = 1'b1;clk=1;
       $monitor("%b%b%b%b%b",Q[4],Q[3],Q[2],Q[1],Q[0]);
      #10ns reset = 0;
   repeat (53)
      #10ns clk = ~clk;
        end
endmodule
```

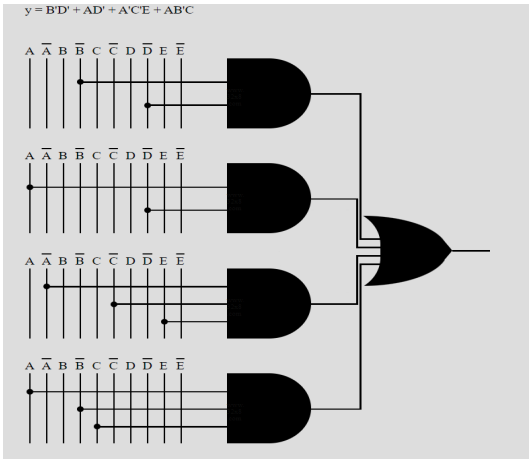
Figure 5: code Test bench

when i do k-map the result is as follows

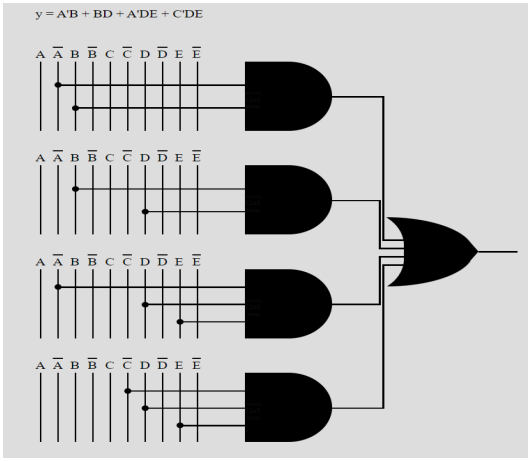
Hint:I have created k-map and circuit in a dedicated site, so if you want to check the site, press the number [1]
y=T
A= Q[4]
B=Q[3]
C= Q[2]
D= Q[1]
E= Q[0]
for This picture



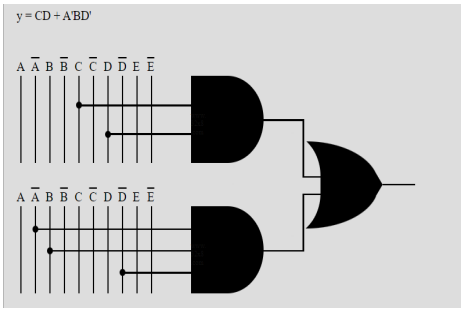
(a) Circuit T0 in prime up



(b) Circuit T1 in prime up



(c) Circuit T2 in prime up



(d) Circuit T3 in prime up

Figure 6: Result For K-map and Circuit For prime up counter

```
16 //T0={2,31}
17 //T1={3,5,11,17,23,29}
18 //T2={3,7,11,13,19,31}
19 //T3={7,13,23,31}
20 //T4={13,31}
21
22 //T0=Q0' +Q3Q2Q1
23 //T1=Q3'Q1'+Q4Q1'+Q4'Q2'Q0+Q4Q3'Q2
24 //T2=Q4'Q3+Q3Q1+Q4'Q1Q0+Q2'Q1Q0
25 //T3=Q2Q1+Q4'Q3Q1'
26 //T4=Q4'Q3Q1'+Q3Q2Q1
27
```

Figure 7: SOP & characteristic equation For counter prime up

```
# KERNEL: 00010
# KERNEL: 00011
# KERNEL: 00101
# KERNEL: 00111
# KERNEL: 01011
# KERNEL: 01101
# KERNEL: 10001
# KERNEL: 10011
# KERNEL: 10111
# KERNEL: 11101
# KERNEL: 11111
# KERNEL: 00010
```

Figure 8: run test bench result For counter prime up

1.2 Prime Down

Here are the first 11 prime numbers:
31,29,23,19,17,13,11,7,5,3,2

1.2.1 truth table prime Down

```
// Truth table for prime number Down
    //Present state          Next state                    T-Flip Flop
//  Q4  Q3  Q2  Q1  Q0      Q4  Q3  Q2  Q1  Q0        T4  T3  T2  T1  T0
//  0   0   0   1   0       1   1   1   1   1         1   1   1   0   1
//  0   0   0   1   1       0   0   0   1   0         0   0   0   0   1
//  0   0   1   0   1       0   0   0   1   1         0   0   1   1   0
//  0   0   1   1   1       0   0   1   0   1         0   0   0   1   0
//  0   1   0   1   1       0   0   1   1   1         0   1   1   0   0
//  0   1   1   0   1       0   1   0   1   1         0   0   1   1   0
//  1   0   0   0   1       0   1   1   0   1         1   1   1   0   0
//  1   0   0   1   1       1   0   0   0   1         0   0   0   1   0
//  1   0   1   1   1       1   0   0   1   1         0   0   1   0   0
//  1   1   1   0   1       1   0   1   1   1         0   1   0   1   0
//  1   1   1   1   1       1   1   1   0   1         0   0   0   1   0
```

Figure 9: Truth Table prime down counter

1.2.2 Code prime down & Test bench

```
module primeDwon(clk,Q,reset);
input clk,reset;
output [4:0]Q;
wire [4:0]D;

    assign D[0]=(~Q[4] & ~Q[3] & ~Q[2]);
    assign D[1]=(~Q[4] & Q[2]) |(Q[3] & Q[2]) |(Q[4] & ~Q[2] & Q[1]);
    assign D[2]=(~Q[0])|(~Q[4]& ~Q[1])|(~Q[2] & ~Q[1]) |(~Q[4] & Q[3]) | (Q[4] & ~Q[3] & Q[2] & Q[1]);
    assign D[3]=(~Q[0])|(~Q[2]& ~Q[1])|(Q[3]& ~Q[2])|(Q[4] & Q[3] & ~Q[1]);
    assign D[4]=(~Q[0])|(~Q[2] & ~Q[1]);


    T_ff U0(D[0],clk,reset,Q[0]);
    T_ff U1(D[1],clk,reset,Q[1]);
    T_ff U2(D[2],clk,reset,Q[2]);
    T_ff U3(D[3],clk,reset,Q[3]);
    T_ff U4(D[4],clk,reset,Q[4]);
endmodule
```

Figure 10: code prime down

```
204 module testPrimeNumberDwon;
205     reg clk,reset;
206     wire [4:0]Q;
207     primeDwon R1(clk,Q,reset);
208     initial
209         begin
210         reset = 1'b1;clk=1;
211        $monitor("%b%b%b%b%b",Q[4],Q[3],Q[2],Q[1],Q[0]);
212       #10ns reset = 0;
213    repeat (53)
214       #10ns clk = ~clk;
215         end
216 endmodule
```

Figure 11: code Test bench

```
62  //T0={2,3}
63  //T1={5,7,13,19,29,31}
64  //T2={2,5,11,13,17,23}
65  //T3={2,11,17,29}
66  //T4={2,17}
67
68  //T0=Q4'Q3'Q2'
69  //T1= Q4'Q2+Q3Q2+Q4Q2'Q1
70  //T2= Q0'+Q4'Q1'+Q2'Q1'+Q3'Q3+Q4Q3'Q2Q1
71  //T3=Q0'+Q2'Q1'+Q3Q2'+Q4Q3Q1'
72  //T4=Q0'+Q2'Q1'
73
74
```

Figure 12: SOP & characteristic equation For counter prime Down

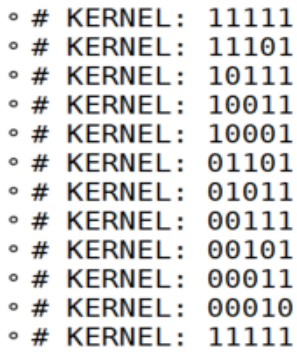
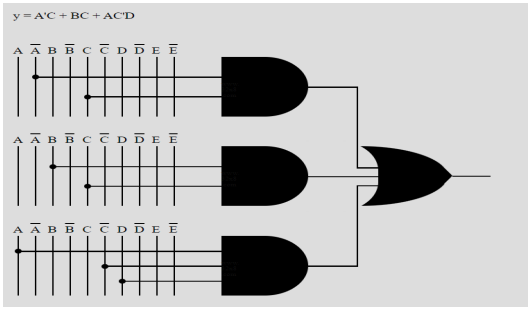
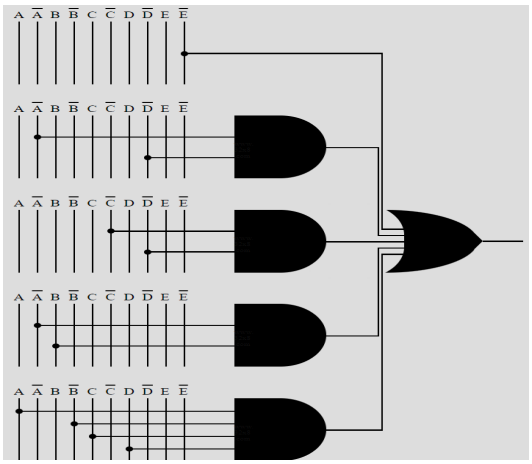


Figure 13: run test bench result For counter prime Down

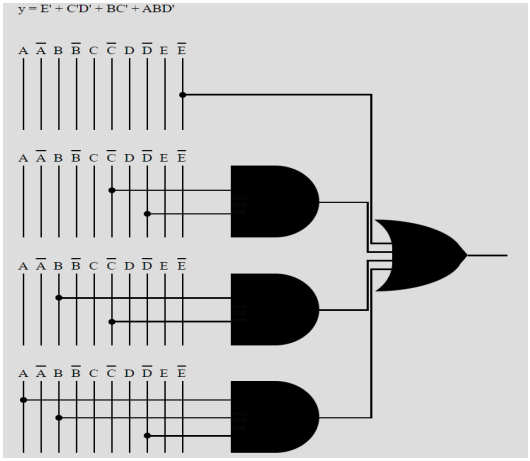
Hint:I have created k-map and circuit in a dedicated site, so if you want to check the site, press the number [1]
y=T
A= Q[4]
B=Q[3]
C= Q[2]
D= Q[1]
E= Q[0]
for This picture



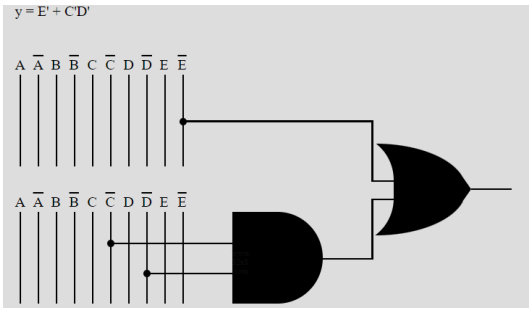
(a) Circuit T1 in prime down



(b) Circuit T2 in prime down



(c) Circuit T3 in prime down



(d) Circuit T4 in prime down

Figure 14: Result For K-map and Circuit For prime Down counter

1.3 Fibonacci number UP

Fibonacci numbers are a sequence of numbers in which each number is the sum of the two preceding ones. The sequence starts with 0 and 1, and the subsequent numbers are generated by adding the previous two numbers together.

Here are the first few Fibonacci numbers:

$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...$

The Fibonacci sequence can be defined recursively by the following formula:

$F(n) = F(n-1) + F(n-2)$

where $F(0) = 0$ and $F(1) = 1$.

Fibonacci numbers have many interesting properties and can be found in various areas of mathematics and nature. They often appear in patterns in biological systems,

art, and even financial markets.

1.3.1 Truth table Fibonacci UP

```
// Truth table for Fibb number UP
   //Present state          Next state              T-Flip Flop
// Q5   Q4  Q3  Q2  Q1  Q0   Q5 Q4  Q3  Q2  Q1  Q0     T5  T4  T3  T2  T1  T0
// 0    0   0   0   0   0    0  0   0   0   0   1      0   0   0   0   0   1   s0
// 0    0   0   0   0   1    0  0   0   0   1   0      0   0   0   0   1   1   s1
// 0    0   0   0   0   1    0  0   0   0   1   0      0   0   0   0   1   1   sR ---------------- Error
// 0    0   0   0   1   0    0  0   0   0   1   1      0   0   0   0   0   1   s2
// 0    0   0   0   1   1    0  0   0   1   0   1      0   0   0   1   1   0   s3
// 0    0   0   1   0   1    0  0   1   0   0   0      0   0   1   1   0   1   s5
// 0    0   1   0   0   0    0  0   1   1   0   1      0   0   0   1   0   1   s8
// 0    0   1   1   0   1    0  1   0   1   0   1      0   1   1   0   0   0   s13
// 0    1   0   1   0   1    1  0   0   0   1   0      1   1   0   1   1   1   s21
// 1    0   0   0   1   0    1  1   0   1   1   1      0   1   0   1   0   1   s34
// 1    1   0   1   1   1    0  0   0   0   0   0      1   1   0   1   1   1   s55
```

Figure 15: Truth Table Fibonacci UP counter

1.3.2 Code Fibonacci UP & Test bench

```
module FibbUP(clk,Q,reset);
input clk,reset;
output [5:0]Q;
wire [5:0]D;
    assign D[0]=~Q[0]|(Q[4])|(~Q[3] & ~Q[1]);
    assign D[1]=(Q[4])|(~Q[2]& Q[0]);
    assign D[2]=(Q[5])|(Q[1]& Q[0])|(~Q[3] & Q[2]) | (Q[3] & ~Q[2]);
    assign D[3]=(~Q[4] & Q[2]);
    assign D[4]=(Q[4]) | (Q[5]) | (Q[3] & Q[0]);
    assign D[5]=Q[4];
    T_ff U0(D[0],clk,reset,Q[0]);
    T_ff U1(D[1],clk,reset,Q[1]);
    T_ff U2(D[2],clk,reset,Q[2]);
    T_ff U3(D[3],clk,reset,Q[3]);
    T_ff U4(D[4],clk,reset,Q[4]);
    T_ff U5(D[5],clk,reset,Q[5]);
endmodule
```

Figure 16: code Fibonacci UP

```
300 module testFibbUP;
301     reg clk,reset;
302     wire [5:0]Q;
303     FibbUP  U(clk,Q,reset);
304     initial
305         begin
306         reset = 1'b1;clk=1;
307        $monitor("%b%b%b%b%b%b",Q[5],Q[4],Q[3],Q[2],Q[1],Q[0]);
308        #10ns reset = 0;
309    repeat (100)
310       #10ns clk = ~clk;
311         end
312 endmodule
313
```

Figure 17: code Test bench

1.3.3 Circuits and Summation SOP

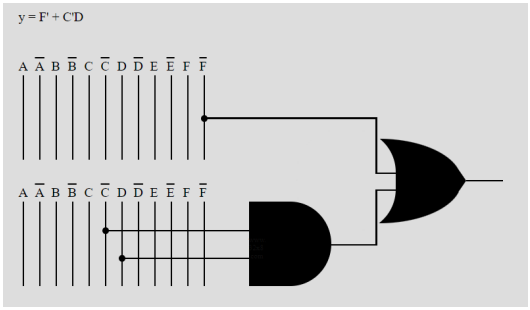
I did not photograph all the circuits because they are simple with the caveat on the same note that I wrote earlier which was I have created k-map and circuit in a dedicated site, so if you want to check the site, press the number [1]
Hint:I considered case 1-1 non-existent in Fibonacci Consecutive excuse me

```
92  //T0={0,1,2,5,8,21,34,55}
93  //T1={1,3,21,55}
94  //T2={3,5,8,21,34,55}
95  //T3={5,13}
96  //T4={13,21,34,55}
97  //T5={21,55}
98
99  //T0=Q0'+Q3'Q2
100 //T1=Q4+Q1Q0
101 //T2=Q5+Q1Q0+Q3'Q2+Q3Q2'
102 //T3=Q4'Q2
103 //T4=Q4+Q5+Q3Q0
104 //T5=Q4
```

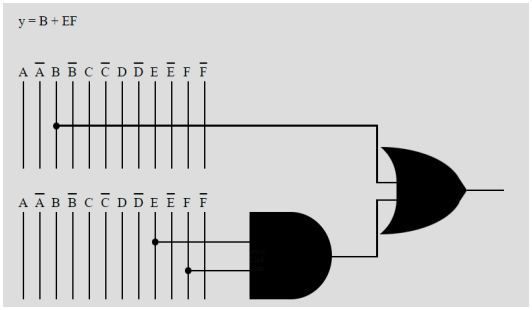
(a) SOP & characteristic equation

```
# KERNEL: 000000
# KERNEL: 000001
# KERNEL: 000010
# KERNEL: 000011
# KERNEL: 000101
# KERNEL: 001000
# KERNEL: 001101
# KERNEL: 010101
# KERNEL: 100010
# KERNEL: 110111
# KERNEL: 000000
```

(b) Result Of RUN testbench



(c) Circuit T0 in Fibonacci UP



(d) Circuit T1 in Fibonacci UP

Figure 18: Circuits and Summation SOP For Fibonacci UPcounter

1.4 Fibonacci number Down

Here are the first 11 Fibonacci number Down:
55,34,21,13,8,5,3,2,1,1,0

1.4.1 Truth table Fibonacci Down

```
93  // Truth table for Fibb number Dwon
94      //Present state              Next state                  T-Flip Flop
95  // Q5   Q4  Q3  Q2  Q1  Q0    Q5 Q4  Q3  Q2  Q1  Q0          T5   T4  T3  T2  T1  T0
96  // 0    0   0   0   0   0     1  1   0   1   1   1           1    1   0   1   1   1   s0
97  // 0    0   0   0   0   1     0  0   0   0   0   0           0    0   0   0   0   1   s1
98  // 0    0   0   0   0   1     0  0   0   0   1   1           0    0   0   0   1   1   sR ---
99  // 0    0   0   0   1   0     0  0   0   0   0   1           0    0   0   0   1   1   s2
100 // 0    0   0   0   1   1     0  0   0   0   1   0           0    0   0   0   0   1   s3
101 // 0    0   0   1   0   1     0  0   0   0   1   1           0    0   0   1   1   0   s5
102 // 0    0   1   0   0   0     0  0   0   1   0   1           0    0   1   1   0   1   s8
103 // 0    0   1   1   0   1     0  0   1   0   0   0           0    0   0   1   0   1   s13
104 // 0    1   0   1   0   1     0  0   1   1   0   1           0    1   1   0   0   0   s21
105 // 1    0   0   0   1   0     0  1   0   1   0   1           1    1   0   1   1   1   s34
106 // 1    1   0   1   1   1     1  0   0   0   1   0           0    1   0   1   0   1   s55
107
```

Figure 19: Truth Table Fibonacci Down counter

1.4.2 Code Fibonacci Down & Test bench

```
303 module FibbDown(clk,Q,reset);
304 input clk,reset;
305 output [5:0]Q;
306 wire [5:0]D;
307       assign D[0]=~Q[2]|| Q[0] | Q[3];
308       assign D[1]=(~Q[3] & ~Q[0])|(~Q[4]& ~Q[3] & Q[2]);
309       assign D[2]=(Q[5])|(~Q[1]& ~Q[0])|(~Q[3] & Q[2]);
310       assign D[3]=(Q[3] & ~Q[2])|(~Q[5] & Q[4]);
311       assign D[4]=(Q[4]) | (Q[5]) | (~Q[3] & ~Q[1] & ~Q[0]);
312       assign D[5]=(Q[5] & ~Q[4]) |(~Q[3] & ~Q[1] & ~Q[0]);
313       T_ff U0(D[0],clk,reset,Q[0]);
314       T_ff U1(D[1],clk,reset,Q[1]);
315       T_ff U2(D[2],clk,reset,Q[2]);
316       T_ff U3(D[3],clk,reset,Q[3]);
317       T_ff U4(D[4],clk,reset,Q[4]);
318       T_ff U5(D[5],clk,reset,Q[5]);
319 endmodule
```

Figure 20: code Fibonacci Down

```
module testFibbDown;
    reg clk,reset;
    wire [5:0]Q;
    FibbDown U(clk,Q,reset);
    initial
        begin
        reset = 1'b1;clk=1;
       $monitor("%b%b%b%b%b%b",Q[5],Q[4],Q[3],Q[2],Q[1],Q[0]);
     #10ns reset = 0;
  repeat (100)
     #10ns clk = ~clk;
       end
endmodule
```

Figure 21: code Test bench

1.4.3 Circuits and Summation SOP

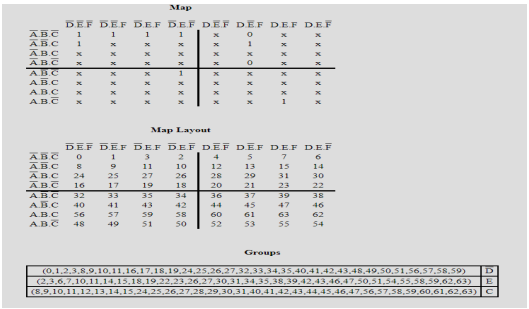
I did not photograph all the circuits because they are simple with the caveat on the same note that I wrote earlier which was I have created k-map and circuit in a dedicated site, so if you want to check the site, press the number [1]
Hint:I considered case 1-1 non-existent in Fibonacci Consecutive excuse me

```
109 //T0={0,1,2,3,8,13,34,55}
110 //T1={0,2,5,34}
111 //T2={0,5,8,13,34,55}
112 //T3={8,21}
113 //T4={0,21,34,55}
114 //T5={0,34}
115
116 //T0=Q2'+Q0+Q3
117 //T1=Q3'Q0'+Q4'Q2'Q3'
118 //T3=Q3Q2'+Q5'Q4
119 //T4=Q4+Q5+Q3'Q2'Q0'
120 //T5=Q5Q4'+Q3'Q2'Q0'
```

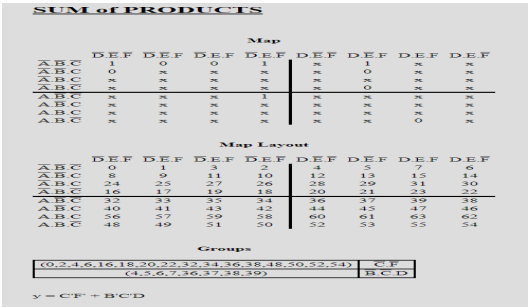
(a) SOP & characteristic equation

```
# KERNEL:  110111
# KERNEL:  100010
# KERNEL:  010101
# KERNEL:  001000
# KERNEL:  000101
# KERNEL:  000010
# KERNEL:  000001
# KERNEL:  000000
# KERNEL:  110111
# KERNEL:  100010
# KERNEL:  010101
# KERNEL:  001000
# KERNEL:  000101
```

(b) Result Of RUN testbench



(c) Circuit T0 in Fibonacci UP



(d) Circuit T1 in Fibonacci down

Figure 22: Circuits and Summation SOP For Fibonacci DOWN counter

1.5 All System

This system collects the four counters and makes the selection based on the input
prime number UP
prime number Down
Fibonacci number UP
Fibonacci number Down

1.5.1 type Mood

if Mood=2'b00 Then **prime number Down**
if Mood=2'b01 Then **prime number UP**
if Mood=2'b10 Then **Fibonacci number UP**
if Mood=2'b11 Then **Fibonacci number Down**

1.5.2 Output System

I made four outputs and they are as follows
outputPrimeUP
outputPrimeDwon
outputFibbUP

outputFibbDown
For example if it is Mood =2'b00 it means that the counter has build is prime down between other counter is reset(equal zero)

1.5.3 Code

```
module SystemAll(clk,outputPrimeUP,outputPrimeDwon,outputFibbUP,outputFibbDown,Mood);
 input clk;
 input [1:0]Mood;
output [4:0]outputPrimeUP,outputPrimeDwon;
output [5:0]outputFibbUP,outputFibbDown;
wire [3:0]yourChoice;
//prime down
assign yourChoice[0]= (Mood==2'b00)?1'b0:1'b1;
//prime  UP
assign yourChoice[1]= (Mood==2'b01)?1'b0:1'b1;
//Fibonacci Down
assign yourChoice[2]= (Mood==2'b10)?1'b0:1'b1;
//Fibonacci UP
assign yourChoice[3]= (Mood==2'b11)?1'b0:1'b1;
PrimeUP  R0(clk,outputPrimeUP,yourChoice[0]) ;
primeDwon R1(clk,outputPrimeDwon,yourChoice[1]);
FibbUP   R2(clk,outputFibbUP,yourChoice[2]);
FibbDown R3(clk,outputFibbDown,yourChoice[3]);
endmodule
```

Figure 23: code all system

1.5.4 Test bench System

```
module testAllSystem;
    reg clk;
    reg [1:0]Mood;
    wire [4:0]outputPrimeUP,outputPrimeDwon;
    wire [5:0]outputFibbUP,outputFibbDown;
    SystemAll S(clk,outputPrimeUP,outputPrimeDwon,outputFibbUP,outputFibbDown,Mood);
initial
    begin
         Mood=2'b00;
        clk=1;
         $monitor("Up prime:%b%b%b%b%b",outputPrimeUP[4],outputPrimeUP[3],outputPrimeUP[2],outputPrimeUP[1],outputPrimeUP[0]);
         $monitor("Down  prime:%b%b%b%b%b",outputPrimeDwon[4],outputPrimeDwon[3],outputPrimeDwon[2],outputPrimeDwon[1],outputPrimeDwon[0]);
         $monitor("up Fibb:%b%b%b%b%b",outputFibbUP[5],outputFibbUP[4],outputFibbUP[3],outputFibbUP[2],outputFibbUP[1],outputFibbUP[0]);
         $monitor("Down Fibb:%b%b%b%b%b",outputFibbDown[5],outputFibbDown[4],outputFibbDown[3],outputFibbDown[2],outputFibbDown[1],outputFibbDown[0]
         repeat(60)
         begin
         #10ns clk=~clk;
         end
          #10ns Mood=2'b01;
         repeat(60)
         begin
         #10ns clk=~clk;
         end
          #10ns Mood=2'b10;
```

Figure 24: code Test bench for all system

```
241             repeat(60)
242             begin
243             #10ns clk=~clk;
244             end
245              #10ns Mood=2'b01;
246             repeat(60)
247             begin
248             #10ns clk=~clk;
249             end
250              #10ns Mood=2'b10;
251             repeat(60)
252             begin
253             #10ns clk=~clk;
254             end
255             #10ns Mood=2'b11;
256             repeat(60)
257             begin
258             #10ns clk=~clk;
259             end
260           end
261   endmodule
```

Figure 25: code Test bench for all system part two

run # Simulation has been initialized run KERNEL: Down prime:xxxxx
KERNEL: Up prime:00000
KERNEL: up Fibb:000000
KERNEL: Down Fibb:000000
KERNEL: Down prime:00000
KERNEL: Up prime:00011
KERNEL: Up prime:00101
KERNEL: Up prime:00111
KERNEL: Up prime:01011
KERNEL: Up prime:01101
KERNEL: Up prime:10001
KERNEL: Up prime:10011
KERNEL: Up prime:00000
KERNEL: Down prime:11101
KERNEL: Down prime:10111
KERNEL: Down prime:10011
KERNEL: Down prime:10001
KERNEL: Down prime:01101
KERNEL: Down prime:01011
KERNEL: Down prime:00111
KERNEL: Down prime:00101
KERNEL: Down prime:00000
KERNEL: up Fibb:000001
KERNEL: up Fibb:000010
KERNEL: up Fibb:000011
KERNEL: up Fibb:000101
KERNEL: up Fibb:001000
KERNEL: up Fibb:001101
KERNEL: up Fibb:010101
KERNEL: up Fibb:000000
KERNEL: Down Fibb:110111
KERNEL: Down Fibb:100010
KERNEL: Down Fibb:010101
KERNEL: Down Fibb:001000
KERNEL: Down Fibb:000101

KERNEL: Down Fibb:000010
KERNEL: Down Fibb:000001
KERNEL: Down Fibb:000000

KERNEL: Simulation has finished. There are no more test vectors to simulate

We note that if we were, for example, in state up Down prime:00000 and came after clk and Mood become 2'b10 then this makes Next State equal up Fibb:000001.

1.6 Test Generator & System Analyzer & Verification

This section is one of the most important sections, and I would have liked to do it, but time did not help me, and I will only explain it

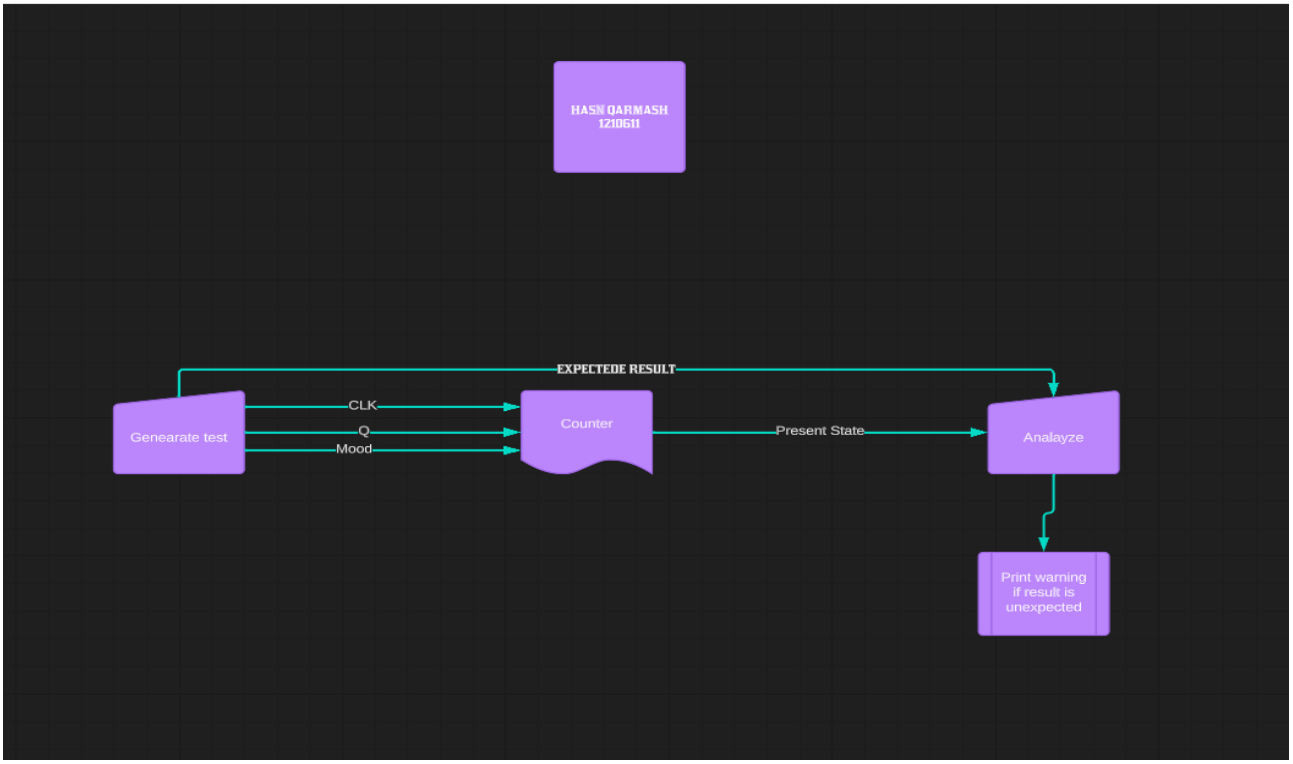


Figure 26: explain

A test generator code in Verilog is a piece of code that is designed to provide test stimuli or inputs in order to test a design or module. The design under test (DUT) inputs are subjected to a set of instructions or patterns that are used to check the functioning of the DUT or to trigger certain test cases.
A test generator code's main function is to offer a methodical, automated mechanism to create input patterns so that the DUT is thoroughly tested. By putting the design through multiple test situations, it helps to confirm its accuracy and resilience.

2 Conclusion

In this project, we encountered a lot of obstacles, some of which we solved and others not, as one of these problems was the number of inputs 8, and this number I cannot deal with, especially since I chose the k-map method, so I divided the counters into four, and this makes it possible for me to do k-map and then complete the rest The steps in the design, and I encountered another problem, which is that while I was doing the testBench process, the results that appeared were illogical, which made me repeat some steps in the design, and this consumed a lot of time, while the problem was not writing testBench correctly, one of the problems also, which is when I type code"all system" is not able to determine which outputs I can start, especially since some of the counters contain 5-bit and some of them contain 6-bit.The solution was to make four outputs for each counter, and in the event that one of them works, the value of the remainder will be zero. One of the problems that I could not solve is the Fibonacci sequence 0 - 1 - 1, which is how to make it repeat the same number twice. I thought a lot and searched for you, I could not find any results.
by the end of this The report My understanding of counters has deepened much more as I had a weakness in this subject and I reviewed the digital systems course, I hope you have been pleased while reading the report.
With Best Wishes:) .

References

[1] *K-map solver*. URL: http://www.32x8.com/var6.html.