



**Faculty of Engineering & Technology Electrical &
Computer Engineering Department**

ADVANCED DIGITAL SYSTEMS DESIGN

Structural counter using T-Flip Flops

Instructor: Abdallatif Abuissa

Prepared by: Qossay Rida 1211553

Section: One

Date: 2023/6/12

Abstract

In this project, the objective is to design a structural circuit that can count either prime numbers or Fibonacci numbers based on a given input. Additionally, the circuit should be able to count in either the upward or downward direction, depending on another input. The counter will be designed to count the first 11 numbers of the selected sequence. The circuit will utilize T Flip Flop and combination logic to implement the functionality.

Through the previous information, it becomes clear to us that the counter that we want to build will contain five input clocks: PorF (to determine counting prime or fibonacci), UorD (to determine counting up or down), asynchronous reset and synchronous enable, and one output (6 bits) because this counter will have a 6 T Flip Flop to count the first 11 numbers from two sequences.

To verify the correctness of the design, a test bench will be created. The test bench will simulate different scenarios and inputs to ensure that the circuit functions properly. Any errors or discrepancies in the circuit's behavior will be identified and printed as output during the testing phase.

Table of Contents

Abstract.....	2
Theory	4
Analysis project.....	4
Before starting building	5
Build counter.....	7
Test the counter.....	11
.....	16
.....	16
Another Solution.....	19
Conclusion.....	20
Appendix I	21

Table of Figure

Figure 1: T flip flop use in counter	5
Figure 2: solution not used.	6
Figure 3: Prime Fibonacci counter	10
Figure 4: Automatic Test Bench.....	11
Figure 5: Another solution	19

Table of Tables

Table 1: Prime State	4
Table 2: Fibonacci State.....	4
Table 3: State Table for prime up	7
Table 4: State Table for prime down	7
Table 5: State Table for Fibonacci up	8
Table 6: State Table for Fibonacci down	8

Theory

Analysis project

The counter we will build will count two different sequences in addition to each sequence counting up and counting down; hence, we will determine the two sequences.

As we mentioned, the counter counts the first 11 numbers from each sequence.

First Sequence (counting prime number):

2 <-> 3 <-> 5 <-> 7 <-> 11 <-> 13 <-> 17 <-> 19 <-> 23 <-> 29 <-> 31

Second Sequence (counting Fibonacci number):

0 <-> 1 <-> 1 <-> 2 <-> 3 <-> 5 <-> 8 <-> 13 <-> 21 <-> 35 <-> 55

The biggest number from two sequences is 55, which means the number of T flip flops we need equals the number of digits for 55 in binary (100010); hence, the number of flip flops will be six.

Then the state for this counter will be:

Prime State					
Q6	Q5	Q4	Q3	Q2	Q1
0	0	0	0	1	0
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	1
0	0	1	0	1	1
0	0	1	1	0	1
0	1	0	0	0	1
0	1	0	0	1	1
0	1	0	1	1	1
0	1	1	1	0	1
0	1	1	1	1	1

Table 2: Prime State

Fibonacci State					
Q6	Q5	Q4	Q3	Q2	Q1
0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	0	1	1
0	0	0	1	0	1
0	0	1	0	0	0
0	0	1	1	0	1
0	1	0	1	0	1
1	0	0	0	1	0
1	1	0	1	1	1

Table 2: Fibonacci State.

Before starting building

Before starting to build the counter, we must identify T flip flop that will be used

T flip flop, also known as a toggle flip-flop, is a type of digital circuit that can store a single bit of information. It is called a T flip flop because it has a single input called T (Toggle) that changes the state of the flip-flop when a signal is applied to it.

When the counter run appear initial state in T flip flops depends on inputs of counter, because that we add additional input for T flip flop called initial state.

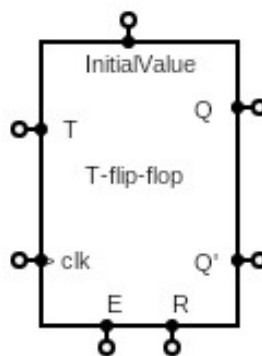


Figure 1: T flip flop use in counter

There is another solution that I did not use:

The initial state for the counter when counting Fibonacci up is 0 (000000) and when counting prime up is 2 (000010). The difference between the two initial states in this case is in the second bit (second T flip flop). To solve this problem, if the counter is counting prime up, don't reset the counter, and if the value of the second flip flop doesn't equal 1, set it to 1. The circuit below shows how to apply this solution (note: this counter has an active low reset).

This solution isn't used because if the counter counts Fibonacci down or prime down, the initial state will be (55) or (31) and this solution can't do this, unlike the first solution.

Two solutions are correct but I choose first solution

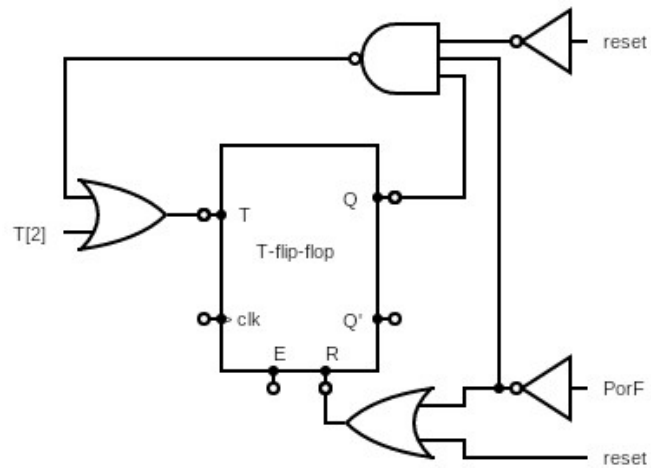


Figure 2: solution not used.

Hence the implementation for T flip flop used is:

```

module TFF (input T,clk,reset,initialState,enable,output reg Q);
    always @(posedge clk or negedge reset ) begin
        if (enable)
            if (~reset)
                Q=initialState;
            else
                Q=Q^T;
        end
    end
endmodule

```

Now we need to find each equation that connects to each flip flop, and we need a control bit to determine if the counter is counting prime or Fibonacci. this bit called (PorF) if equal to 0 counting prim else counting Fibonacci and need another bit to determine if the counter is counting up or down this bit called (UorD) if equal to 0 counting up else counting down

In next step we will build the counter using state table.

Build counter

Now we need to find each equation that connects to each flip flop using this state tables

D value	input		Present State						Next State						Flip-Flop input					
	PorF	UorD	Q6	Q5	Q4	Q3	Q2	Q1	Q6	Q5	Q4	Q3	Q2	Q1	T6	T5	T4	T3	T2	T1
2	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	1
3	0	0	0	0	0	0	1	1	0	0	0	1	0	1	0	0	0	1	1	0
5	0	0	0	0	0	1	0	1	0	0	0	1	1	1	0	0	0	0	1	0
7	0	0	0	0	0	1	1	1	0	0	1	0	1	1	0	0	1	1	0	0
11	0	0	0	0	1	0	1	1	0	0	1	1	0	1	0	0	0	1	1	0
13	0	0	0	0	1	1	0	1	0	1	0	0	0	1	0	1	1	1	0	0
17	0	0	0	1	0	0	0	1	0	1	0	0	1	1	0	0	0	0	1	0
19	0	0	0	1	0	0	1	1	0	1	0	1	1	1	0	0	0	1	0	0
23	0	0	0	1	0	1	1	1	0	1	1	1	0	1	0	0	1	0	1	0
29	0	0	0	1	1	1	0	1	0	1	1	1	1	1	0	0	0	0	1	0
31	0	0	0	1	1	1	1	1	0	0	0	0	1	0	0	1	1	1	0	1

Table 3: State Table for prime up

In this table, when counter-counting primes up, show the present state and what the next state is, then find the value of T when this state happened.

D value	input		Present State						Next State						Flip-Flop input					
	PorF	UorD	Q6	Q5	Q4	Q3	Q2	Q1	Q6	Q5	Q4	Q3	Q2	Q1	T6	T5	T4	T3	T2	T1
95	0	1	0	1	1	1	1	1	0	1	1	1	0	1	0	0	0	0	1	0
93	0	1	0	1	1	1	0	1	0	1	0	1	1	1	0	0	1	0	1	0
87	0	1	0	1	0	1	1	1	0	1	0	0	1	1	0	0	0	1	0	0
83	0	1	0	1	0	0	1	1	0	1	0	0	0	1	0	0	0	0	1	0
81	0	1	0	1	0	0	0	1	0	0	1	1	0	1	0	1	1	1	0	0
77	0	1	0	0	1	1	0	1	0	0	1	0	1	1	0	0	0	1	1	0
75	0	1	0	0	1	0	1	1	0	0	0	1	1	1	0	0	1	1	0	0
71	0	1	0	0	0	1	1	1	0	0	0	1	0	1	0	0	0	0	1	0
69	0	1	0	0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1	0
67	0	1	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	1
66	0	1	0	0	0	0	1	0	0	1	1	1	1	1	0	1	1	1	0	1

Table 4: State Table for prime down

In this table, when counter-counting primes down, show the present state and what the next state is, then find the value of T when this state happened.

D value	input		Present State						Next State						Flip-Flop input					
	PorF	UorD	Q6	Q5	Q4	Q3	Q2	Q1	Q6	Q5	Q4	Q3	Q2	Q1	T6	T5	T4	T3	T2	T1
128	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
129	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	1
130	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	1
131	1	0	0	0	0	0	1	1	0	0	0	1	0	1	0	0	0	1	1	0
133	1	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	1	1	0	1
136	1	0	0	0	1	0	0	0	0	0	1	1	0	1	0	0	0	1	0	1
141	1	0	0	0	1	1	0	1	0	1	0	1	0	1	0	1	1	0	0	0
149	1	0	0	1	0	1	0	1	1	0	0	0	1	0	1	1	0	1	1	1
162	1	0	1	0	0	0	1	0	1	1	0	1	1	1	0	1	0	1	0	1
183	1	0	1	1	0	1	1	1	0	0	0	0	0	0	1	1	0	1	1	1

Table 5: State Table for Fibonacci up

In this table, when counter-counting Fibonacci up, show the present state and what the next state is, then find the value of T when this state happened.

D value	input		Present State						Next State						Flip-Flop input					
	PorF	UorD	Q6	Q5	Q4	Q3	Q2	Q1	Q6	Q5	Q4	Q3	Q2	Q1	T6	T5	T4	T3	T2	T1
247	1	1	1	1	0	1	1	1	1	0	0	0	1	0	0	1	0	1	0	1
226	1	1	1	0	0	0	1	0	0	1	0	1	0	1	1	1	0	1	1	1
213	1	1	0	1	0	1	0	1	0	0	1	1	0	1	0	1	1	0	0	0
205	1	1	0	0	1	1	0	1	0	0	1	0	0	0	0	0	0	1	0	1
200	1	1	0	0	1	0	0	0	0	0	0	1	0	1	0	0	1	1	0	1
197	1	1	0	0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1	0
195	1	1	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	1
194	1	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	1
193	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
192	1	1	0	0	0	0	0	0	1	1	0	1	1	1	1	1	0	1	1	1

Table 6: State Table for Fibonacci down

In this table, when counter-counting Fibonacci down, show the present state and what the next state is, then find the value of T when this state happened.

Hence:

$T_1(\text{PorF}, \text{UorD}, Q_6, Q_5, Q_4, Q_3, Q_2, Q_1) = \sum(2, 31, 66, 67, 128, 129, 130, 133, 136, 149, 162, 183, 192, 193, 194, 195, 200, 205, 226, 247)$

$T_2(\text{PorF}, \text{UorD}, Q_6, Q_5, Q_4, Q_3, Q_2, Q_1) = \sum(3, 5, 11, 17, 23, 29, 69, 71, 77, 83, 93, 95, 129, 131, 149, 183, 192, 194, 197, 226)$

$$T3(\text{PorF}, \text{UorD}, Q6, Q5, Q4, Q3, Q2, Q1) = \sum(3, 7, 11, 13, 19, 31, 66, 69, 75, 77, 81, 87, 131, 133, 136, 149, 162, 183, 192, 197, 200, 205, 226, 247)$$

$$T4(\text{PorF}, \text{UorD}, Q6, Q5, Q4, Q3, Q2, Q1) = \sum(7, 13, 23, 31, 66, 75, 81, 93, 133, 141, 200, 213)$$

$$T5(\text{PorF}, \text{UorD}, Q6, Q5, Q4, Q3, Q2, Q1) = \sum(13, 31, 66, 81, 141, 149, 162, 183, 192, 213, 226, 247)$$

$$T6(\text{PorF}, \text{UorD}, Q6, Q5, Q4, Q3, Q2, Q1) = \sum(149, 183, 192, 226)$$

After apply each function in K-map the result will be:

$$T1 = (\sim Q[1]) \mid (\text{UorD} \& \sim Q[5] \& \sim Q[4] \& \sim Q[3]) \mid (\text{PorF} \& \text{UorD} \& Q[4]) \mid (\sim \text{UorD} \& Q[4] \& Q[3] \& Q[2]) \mid (\text{PorF} \& \sim \text{UorD} \& \sim Q[4] \& \sim Q[2]) \mid (\text{PorF} \& Q[3] \& Q[2]);$$

$$T2 = (\sim \text{UorD} \& \sim Q[5] \& \sim Q[3] \& Q[1]) \mid (\sim \text{UorD} \& Q[5] \& \sim Q[4] \& Q[3]) \mid (\text{UorD} \& Q[5] \& \sim Q[3] \& Q[2]) \mid (\text{UorD} \& \sim Q[5] \& \sim Q[4] \& Q[3]) \mid (\sim \text{PorF} \& \sim \text{UorD} \& \sim Q[4] \& \sim Q[2]) \mid (\sim \text{PorF} \& Q[5] \& Q[3] \& \sim Q[2]) \mid (\sim \text{PorF} \& \text{UorD} \& Q[4] \& Q[3]) \mid (\text{PorF} \& \text{UorD} \& \sim Q[4] \& \sim Q[1]);$$

$$T3 = (\sim \text{PorF} \& \sim Q[5] \& Q[4]) \mid (\sim \text{UorD} \& Q[4] \& Q[2]) \mid (\sim \text{PorF} \& \text{UorD} \& \sim Q[1]) \mid (Q[6]) \mid (\text{UorD} \& \sim Q[2] \& \sim Q[1]) \mid (\sim \text{UorD} \& \sim Q[5] \& Q[2] \& Q[1]) \mid (\sim \text{UorD} \& \sim Q[3] \& Q[2] \& Q[1]) \mid (\sim \text{PorF} \& \text{UorD} \& \sim Q[4] \& \sim Q[2]) \mid (\sim \text{PorF} \& \text{UorD} \& Q[5] \& \sim Q[4] \& Q[3]) \mid (\text{PorF} \& \sim \text{UorD} \& \sim Q[4] \& Q[3]) \mid (Q[4] \& \sim Q[3]) \mid (\text{UorD} \& \sim Q[5] \& Q[3] \& \sim Q[2]);$$

$$T4 = (\sim \text{PorF} \& \text{UorD} \& \sim Q[4] \& \sim Q[1]) \mid (\text{UorD} \& Q[5] \& \sim Q[2]) \mid (\text{PorF} \& \sim \text{UorD} \& \sim Q[5] \& Q[3]) \mid (\sim \text{PorF} \& \sim \text{UorD} \& Q[3] \& Q[2]) \mid (\sim \text{PorF} \& \sim \text{UorD} \& \sim Q[5] \& Q[4] \& \sim Q[2]) \mid (\text{UorD} \& \sim Q[5] \& Q[4] \& Q[2] \& Q[1]) \mid (\text{UorD} \& Q[4] \& \sim Q[3] \& \sim Q[2]);$$

$$T5 = (\sim \text{PorF} \& \text{UorD} \& \sim Q[4] \& \sim Q[1]) \mid (Q[6]) \mid (\text{UorD} \& \sim Q[4] \& \sim Q[2] \& \sim Q[1]) \mid (\sim \text{UorD} \& \sim Q[5] \& Q[4] \& \sim Q[2] \& Q[1]) \mid (\sim \text{UorD} \& Q[4] \& Q[3] \& Q[2]) \mid (\sim \text{PorF} \& \text{UorD} \& \sim Q[3] \& \sim Q[2]) \mid (Q[5] \& \sim Q[4] \& Q[3] \& \sim Q[2]);$$

$$T6 = (\text{UorD} \& \sim Q[4] \& \sim Q[2] \& \sim Q[1]) \mid (\text{PorF} \& \sim \text{UorD} \& Q[5] \& Q[3]) \mid (\text{UorD} \& Q[6] \& \sim Q[5]);$$

The Verilog code appended in Appendix I

-> when the present state for counter equal 1 disable the counter for one clock because the Fibonacci sequence counting 0 then 1 then again 1, but the state table go to next state directly when posedge clock come then disable the counter for one clock

Then the circuit will be as picture

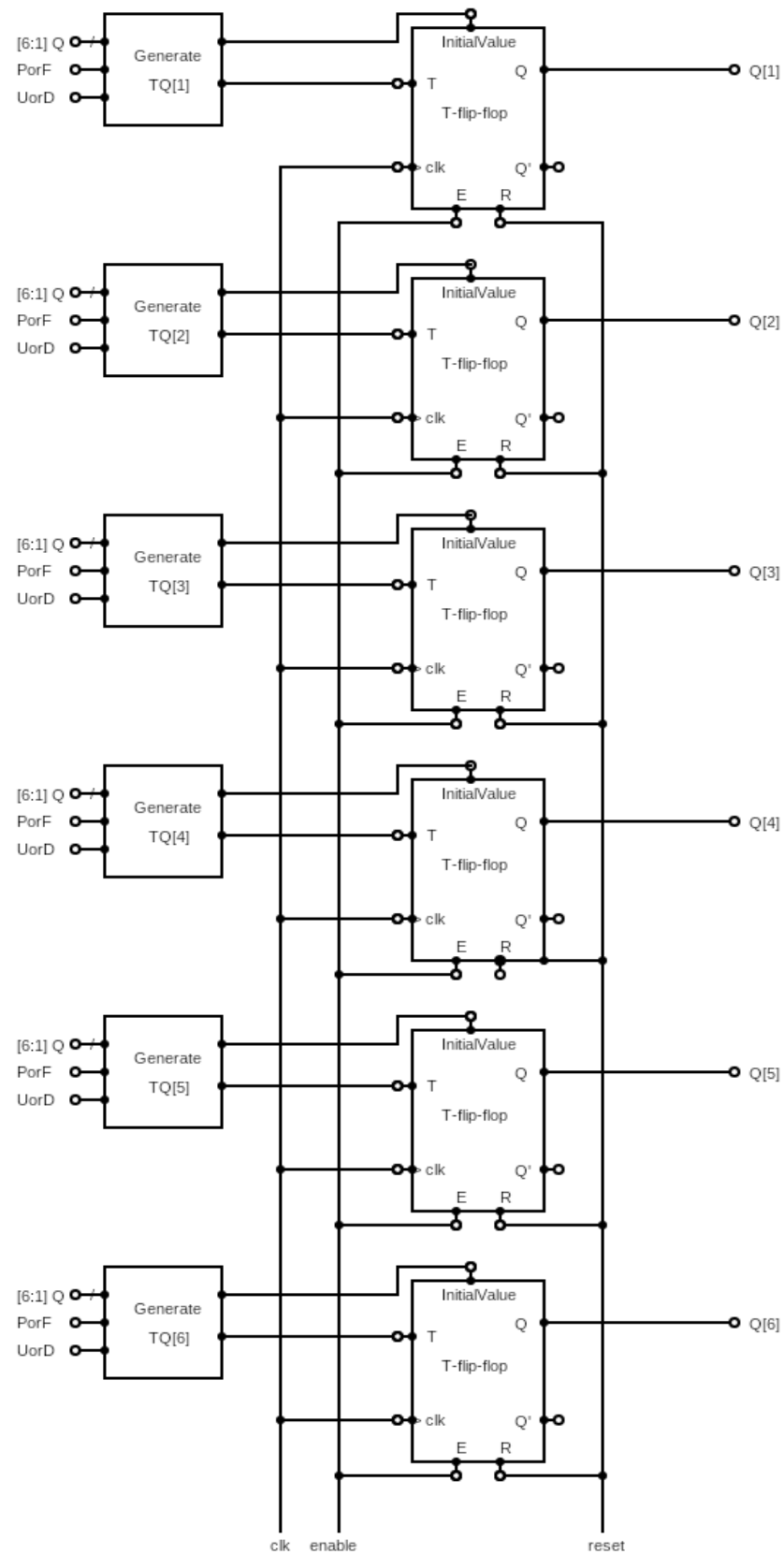


Figure 3: Prime Fibonacci counter

Test the counter

After building the counter, we now generate an automatic tester for this counter.

To generate automatic tester we need a three module

First module(Generate Tests): This module serves as a test module. It has five outputs (clock, PorF, UorD, reset, enable) and an additional 6-bit output ER. This module was responsible to generating input for the counter and calculating the expected output for this input.

Second module(Test Analyser): This module is responsible for analyzing the outputs of the test. It takes two input real result and expected result. Inside the always block, it compares the values of real result and expected result. If they are equal, it displays "True" using the \$display system task. Otherwise, it displays "False".

Third module (Test Bench): This module represents the overall test bench that connects the other modules. It creates wires clk, PorF, UorD, reset, enable, Q, and Q1 to connect the modules. It instantiates the Generate Tests module on the respective wires. It also instantiates the counter module and connects its inputs (clock, PorF, UorD, reset, and enable) to the corresponding wires. Finally, it instantiates the Test Analyzer module and connects its inputs (Q,Q1) to the respective wires, this module shown in Figure4 .

The Verilog code for test bench appended in Appendix I

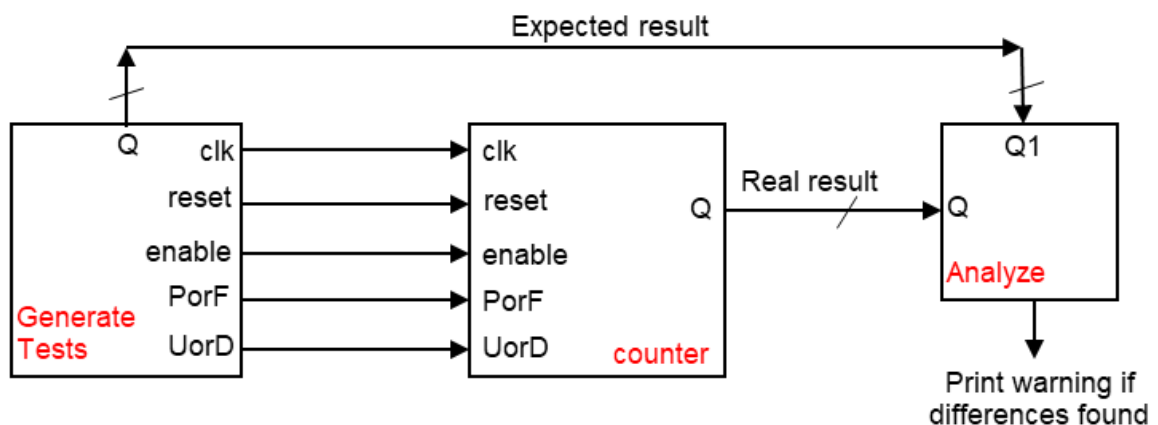


Figure 4: Automatic Test Bench

Result for test:

Test1: Show all sequences

In this test, we will test the counter on the four sequences, after which it will make sure that it counts the four sequences correctly.

As it becomes clear to us from the results attached on the next two pages, that the expected result matches the real result, and therefore the counter works as required.

Test bench will apply:

```
initial begin
    clk = 0;

    UorD = 0; → Case1: prime up
    PorF = 0;

    reset = 0 ;
    enable = 1 ;
    #10ns reset=1;

    #490ns UorD=1; → Case2: prime down

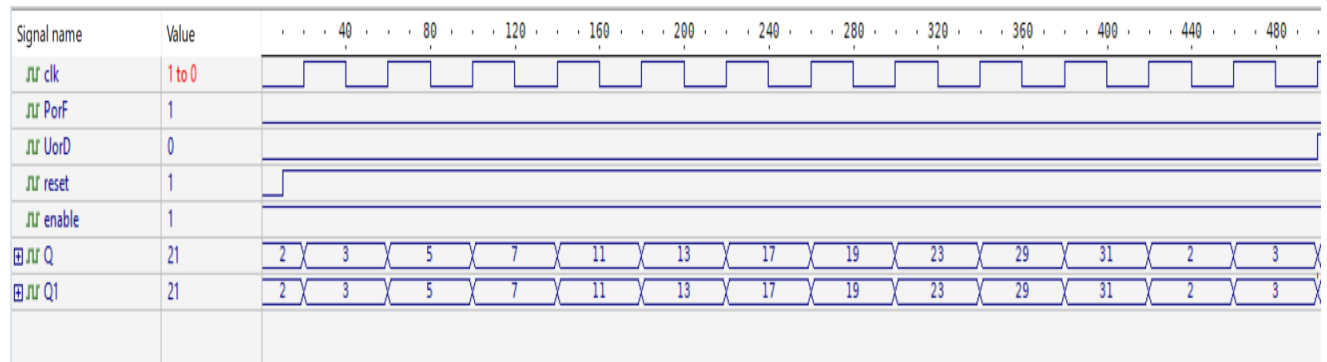
    #480ns PorF=1; → Case3: fibonacci down

    #480ns UorD=0; → Case4: fibonacci up

end

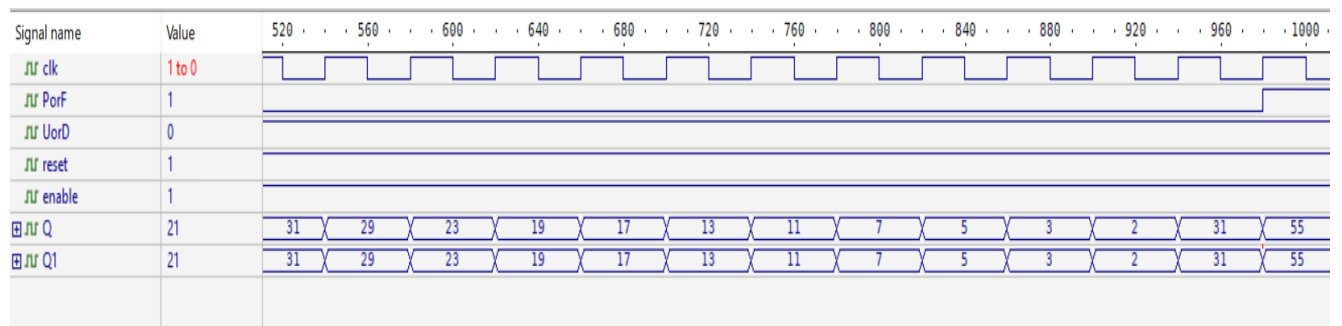
initial begin
    repeat(200)
        #20ns clk=~clk ;
end
```

Case1: prime up



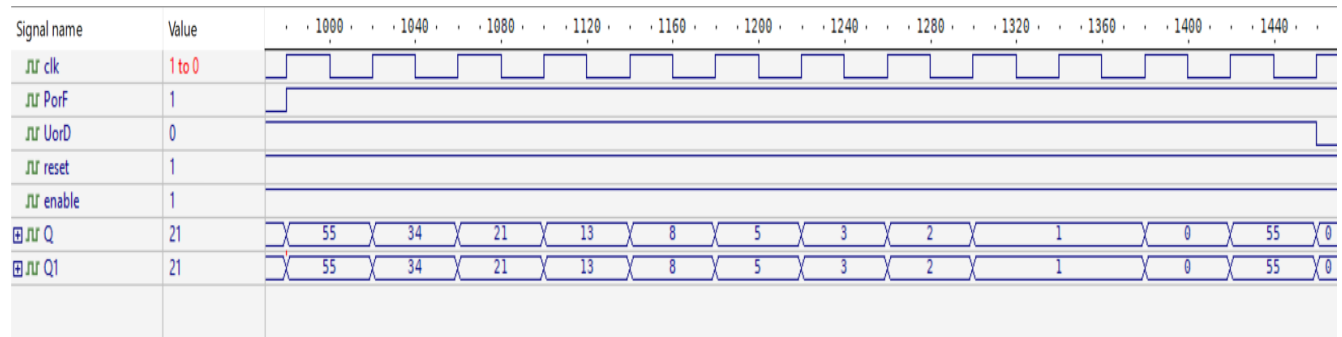
20000	RealResult=	2	ExpectedResult=	2	True
60000	RealResult=	3	ExpectedResult=	3	True
100000	RealResult=	5	ExpectedResult=	5	True
140000	RealResult=	7	ExpectedResult=	7	True
180000	RealResult=	11	ExpectedResult=	11	True
220000	RealResult=	13	ExpectedResult=	13	True
260000	RealResult=	17	ExpectedResult=	17	True
300000	RealResult=	19	ExpectedResult=	19	True
340000	RealResult=	23	ExpectedResult=	23	True
380000	RealResult=	29	ExpectedResult=	29	True
420000	RealResult=	31	ExpectedResult=	31	True
460000	RealResult=	2	ExpectedResult=	2	True
500000	RealResult=	3	ExpectedResult=	3	True

Case2: prime down



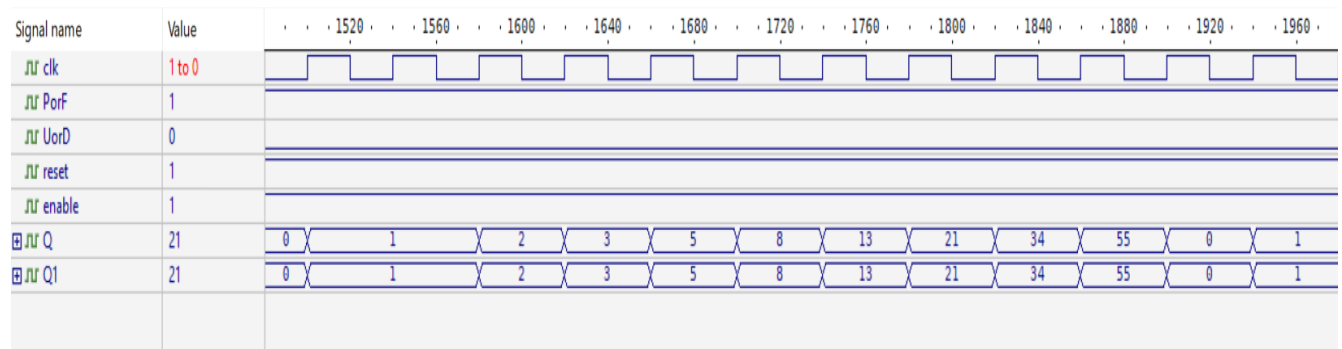
540000	RealResult=	31	ExpectedResult=	31	True
580000	RealResult=	29	ExpectedResult=	29	True
620000	RealResult=	23	ExpectedResult=	23	True
660000	RealResult=	19	ExpectedResult=	19	True
700000	RealResult=	17	ExpectedResult=	17	True
740000	RealResult=	13	ExpectedResult=	13	True
780000	RealResult=	11	ExpectedResult=	11	True
820000	RealResult=	7	ExpectedResult=	7	True
860000	RealResult=	5	ExpectedResult=	5	True
900000	RealResult=	3	ExpectedResult=	3	True
940000	RealResult=	2	ExpectedResult=	2	True
980000	RealResult=	31	ExpectedResult=	31	True

Case3: fibonacci down



1020000	RealResult=	55	ExpectedResult=	55	True
1060000	RealResult=	34	ExpectedResult=	34	True
1100000	RealResult=	21	ExpectedResult=	21	True
1140000	RealResult=	13	ExpectedResult=	13	True
1180000	RealResult=	8	ExpectedResult=	8	True
1220000	RealResult=	5	ExpectedResult=	5	True
1260000	RealResult=	3	ExpectedResult=	3	True
1300000	RealResult=	2	ExpectedResult=	2	True
1340000	RealResult=	1	ExpectedResult=	1	True
1380000	RealResult=	1	ExpectedResult=	1	True
1420000	RealResult=	0	ExpectedResult=	0	True
1460000	RealResult=	55	ExpectedResult=	55	True

Case4: fibonacci up



1500000	RealResult=	0	ExpectedResult=	0	True
1540000	RealResult=	1	ExpectedResult=	1	True
1580000	RealResult=	1	ExpectedResult=	1	True
1620000	RealResult=	2	ExpectedResult=	2	True
1660000	RealResult=	3	ExpectedResult=	3	True
1700000	RealResult=	5	ExpectedResult=	5	True
1740000	RealResult=	8	ExpectedResult=	8	True
1780000	RealResult=	13	ExpectedResult=	13	True
1820000	RealResult=	21	ExpectedResult=	21	True
1860000	RealResult=	34	ExpectedResult=	34	True
1900000	RealResult=	55	ExpectedResult=	55	True
1940000	RealResult=	0	ExpectedResult=	0	True
1980000	RealResult=	1	ExpectedResult=	1	True
2020000	RealResult=	1	ExpectedResult=	1	True

Test2: Check enable & reset

In this test, we will test enable counter and reset counter, after which it will make sure that it counts the four sequences correctly.

As it becomes clear to us from the results attached on the next two pages, that the expected result matches the real result, and therefore the counter works as required.

As we mentioned earlier:

Reset input (asynchronous) to reset the circuit

Enable input (synchronous) to enable/disable the circuit

Test bench will apply:

```
initial begin
```

```
    clk = 0;  
    UorD = 0 ;  
    PorF = 0;  
    reset = 0 ;  
    enable = 1 ;  
    #10ns reset=1 ;
```

```
    #200ns enable=0;  
    #200ns enable=1;
```

→ Case1: Disable/Enable counter

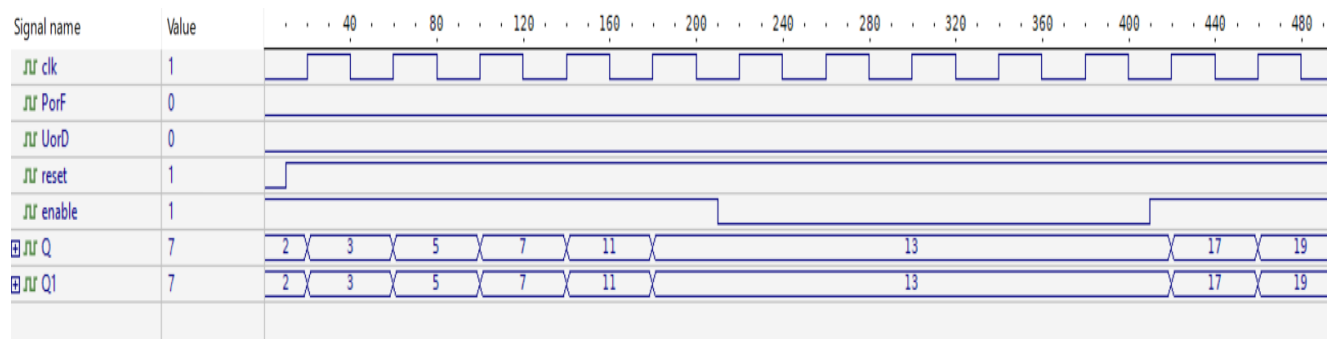
```
    #100ns reset=0;  
    #10ns reset=1;
```

→ Case2: Reset counter

```
end
```

```
initial begin  
    repeat(100)  
        #20ns clk=~clk ;  
end
```

Case1: Disable/Enable counter



20000	RealResult= 2	ExpectedResult= 2	True
60000	RealResult= 3	ExpectedResult= 3	True
100000	RealResult= 5	ExpectedResult= 5	True
140000	RealResult= 7	ExpectedResult= 7	True
180000	RealResult= 11	ExpectedResult= 11	True
220000	RealResult= 13	ExpectedResult= 13	True
260000	RealResult= 13	ExpectedResult= 13	True
300000	RealResult= 13	ExpectedResult= 13	True
340000	RealResult= 13	ExpectedResult= 13	True
380000	RealResult= 13	ExpectedResult= 13	True
420000	RealResult= 13	ExpectedResult= 13	True
460000	RealResult= 17	ExpectedResult= 17	True
500000	RealResult= 19	ExpectedResult= 19	True
540000	RealResult= 2	ExpectedResult= 2	True

Case2: Reset counter



300000	RealResult= 13	ExpectedResult= 13	True
340000	RealResult= 13	ExpectedResult= 13	True
380000	RealResult= 13	ExpectedResult= 13	True
420000	RealResult= 13	ExpectedResult= 13	True
460000	RealResult= 17	ExpectedResult= 17	True
500000	RealResult= 19	ExpectedResult= 19	True
540000	RealResult= 2	ExpectedResult= 2	True
580000	RealResult= 3	ExpectedResult= 3	True
620000	RealResult= 5	ExpectedResult= 5	True
660000	RealResult= 7	ExpectedResult= 7	True

Test3: Check some corner cases

In this test, we will test corner cases like change UorD and PorF at the same moment, after which it will make sure that it counts the four sequences correctly.

As it becomes clear to us from the results attached on the next two pages, that the expected result matches the real result, and therefore the counter works as required.

Test bench will apply:

```
initial begin
```

```
    clk = 0;  
    UorD = 0 ;  
    PorF = 0;  
    reset = 0 ;  
    enable = 1 ;  
    #10ns reset=1 ;
```

```
    #50ns UorD = 1;  
           PorF = 1;  
  
    #60ns PorF = 0;  
  
    #100ns UorD = 0;  
           PorF = 1;
```

→ Chang all cases

```
end
```

```
initial begin  
    repeat(100)  
        #20ns clk=~clk ;  
end
```

Chang all cases



20000	RealResult=	2	ExpectedResult=	2	True
60000	RealResult=	3	ExpectedResult=	3	True
100000	RealResult=	55	ExpectedResult=	55	True
140000	RealResult=	31	ExpectedResult=	31	True
180000	RealResult=	29	ExpectedResult=	29	True
220000	RealResult=	23	ExpectedResult=	23	True
260000	RealResult=	0	ExpectedResult=	0	True
300000	RealResult=	1	ExpectedResult=	1	True
340000	RealResult=	1	ExpectedResult=	1	True
380000	RealResult=	2	ExpectedResult=	2	True
420000	RealResult=	3	ExpectedResult=	3	True
460000	RealResult=	5	ExpectedResult=	5	True
500000	RealResult=	8	ExpectedResult=	8	True
540000	RealResult=	13	ExpectedResult=	13	True
580000	RealResult=	21	ExpectedResult=	21	True
620000	RealResult=	34	ExpectedResult=	34	True
660000	RealResult=	55	ExpectedResult=	55	True
700000	RealResult=	0	ExpectedResult=	0	True
740000	RealResult=	1	ExpectedResult=	1	True
780000	RealResult=	1	ExpectedResult=	1	True

There may be some mismatches between the enumerator that outputs the actual result and the enumerator that outputs the expected result

This is because there is no perfect match between the two enumerations, But each individual result is correct due to the lack of specific and accurate instructions in building this counter

Another Solution

We can build a regular counter that counts from zero to 11 and then returns to zero again, then we build a combinational circuit to convert the numbers from 0 to 11 to another output according to the sequence we have.

Implementation for Counter:

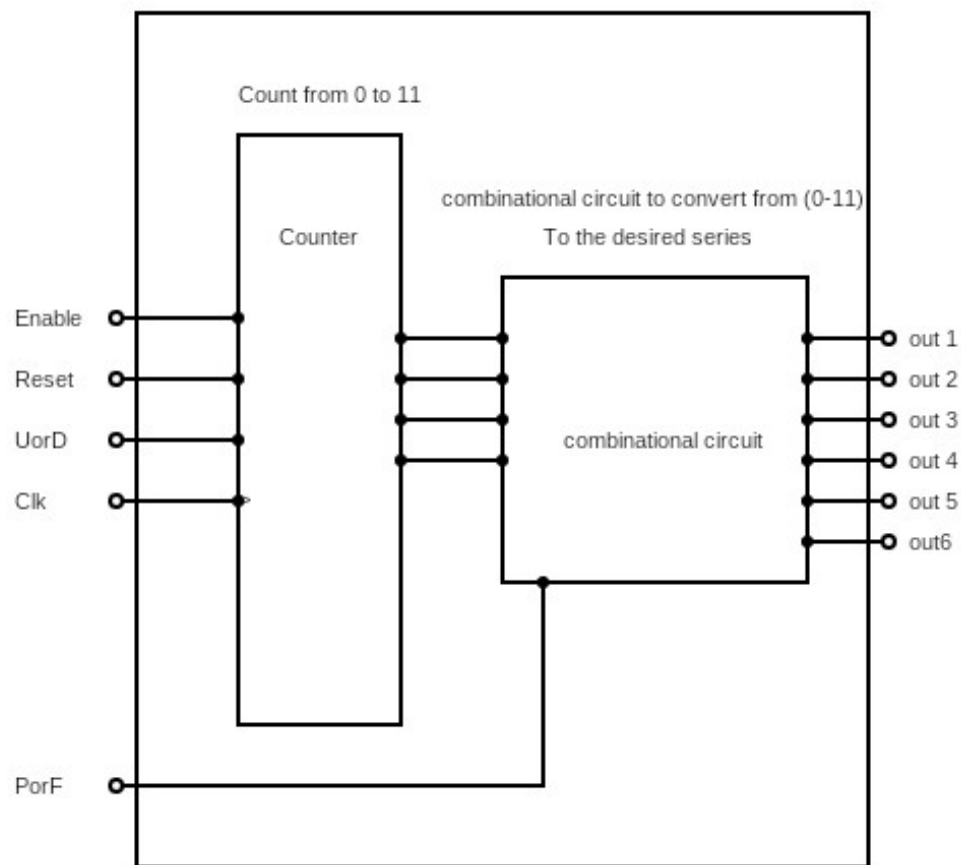


Figure 5: Another solution

Conclusion

In conclusion, the designed structural circuit utilizing T-Flip Flops and combination logic successfully implements a counter capable of counting either prime numbers or Fibonacci numbers based on the value of an input. Additionally, it allows the counting direction to be controlled by another input. The counter accurately counts the first 11 numbers of the selected sequence, and the reset input provides a means to reset the circuit to its initial state. The enable input enables or disables the counting functionality of the circuit.

The designed circuit has been thoroughly tested using a test bench to ensure its proper functioning. The test bench validates the circuit by verifying the correct sequence of numbers being generated and comparing it against the expected output. Any discrepancies or errors are reported during the testing process, allowing for prompt identification and correction.

Future Works:

While the current design successfully achieves the desired functionality, there are several potential areas for improvement and future work. First, the circuit could be optimized for better performance and efficiency. This could involve reducing the number of gates used, streamlining the combinational logic, or exploring alternative circuit architectures.

Furthermore, additional features could be incorporated into the circuit to enhance its versatility and usability. For instance, the ability to count other mathematical sequences or patterns could be added, providing a broader range of applications. Moreover, the circuit could be extended to accommodate larger sequence lengths or incorporate more complex counting algorithms.

Another avenue for future work is the integration of the circuit into a larger system or application. This could involve interfacing with external devices or integrating it into a microcontroller or FPGA-based system. The circuit's inputs and outputs could be connected to other modules or components to enable more comprehensive functionality and interaction with the surrounding system.

Overall, the designed circuit serves as a solid foundation for counting prime numbers or Fibonacci numbers and can be further improved upon to meet specific requirements or explore additional possibilities in the realm of sequence counting circuits.

Appendix I



```
module counter (input clk, PorF, UorD, reset, enable, output [6:1]Q);

    wire [6:1]TQ; // input for flip flop
    reg [6:1]T; // initial state of flip flop
    reg w, z;
    // w: work as reset (when happend any change in PorF or UorD)
    // z: work as enable (to diable counter for one clock when present state= 1)


    // input for each flip flop
    assign TQ[1] = (~Q [1]) | (UorD & ~Q [5] & ~Q[4] & ~Q [3]) | (PorF & UorD & Q[4]) | (~UorD & Q[4]
        & Q [3] & Q[2]) | (PorF & ~UorD & ~Q[4] & ~Q[2]) | (PorF & Q [3] & Q[2]);
    assign TQ[2] = (~UorD & ~Q [5] & ~Q [3] & Q [1]) | (~UorD & Q [5] & ~Q[4] & Q [3]) | (UorD & Q
        [5] & ~Q [3] & Q[2]) | (UorD & ~Q [5] & ~Q[4] & Q [3]) | (~PorF & ~UorD & ~Q[4] &
        ~Q[2]) | (~PorF & Q [5] & Q [3] & ~Q[2]) | (~PorF & UorD & Q[4] & Q [3]) | (PorF &
        UorD & ~Q[4] & ~Q [1]);
    assign TQ[3] = (~PorF & ~Q [5] & Q[4]) | (~UorD & Q[4] & Q[2]) | (~PorF & UorD & ~Q [1]) | (Q[6])
        | (UorD & ~Q[2] & ~Q [1]) | (~UorD & ~Q [5] & Q[2] & Q [1]) | (~UorD & ~Q [3] &
        Q[2] & Q [1]) | (~PorF & UorD & ~Q[4] & ~Q[2]) | (~PorF & UorD & Q [5] & ~Q[4] &
        Q [3]) | (PorF & ~UorD & ~Q[4] & Q [3]) | (Q[4] & ~Q [3]) | (UorD & ~Q [5] & Q [3]
        & ~Q[2]);
    assign TQ[4] = (~PorF & UorD & ~Q[4] & ~Q [1]) | (UorD & Q [5] & ~Q[2]) | (PorF & ~UorD & ~Q [5]
        & Q [3]) | (~PorF & ~UorD & Q [3] & Q[2]) | (~PorF & ~UorD & ~Q [5] & Q[4] &
        ~Q[2]) | (UorD & ~Q [5] & Q[4] & Q[2] & Q [1]) | (UorD & Q[4] & ~Q [3] & ~Q[2]);
    assign TQ[5] = (~PorF & UorD & ~Q[4] & ~Q [1]) | (Q[6]) | (UorD & ~Q[4] & ~Q[2] & ~Q [1]) |
        (~UorD & ~Q [5] & Q[4] & ~Q[2] & Q [1]) | (~UorD & Q[4] & Q [3] & Q[2]) | (~PorF
        & UorD & ~Q [3] & ~Q[2]) | (Q [5] & ~Q[4] & Q [3] & ~Q[2]);
    assign TQ[6] = (UorD & ~Q[4] & ~Q[2] & ~Q [1]) | (PorF & ~UorD & Q [5] & Q [3]) | (UorD & Q[6] &
        ~Q [5]);

    // call T flip flop six times
    TFF t1(TQ[1], clk, w, T[1], z, Q[1]);
    TFF t2(TQ[2], clk, w, T[2], z, Q[2]);
    TFF t3(TQ[3], clk, w, T[3], z, Q[3]);
    TFF t4(TQ[4], clk, w, T[4], z, Q[4]);
    TFF t5(TQ[5], clk, w, T[5], z, Q[5]);
    TFF t6(TQ[6], clk, w, T[6], z, Q[6]);

    // this always to control about initial state and the state when reset counter
    always @(negedge reset or PorF or UorD) begin
        w=reset ;
        // choose suitable state
        if (PorF && ~UorD)
            T=0;
        else if (~PorF && ~UorD)
            T=2;
        else if (PorF && UorD)
            T=55;
        else
            T=31;
        w=0;
        #1ns w=1;
    end

    // this always to disable counter for one clock when present state equal 1
    always @(posedge clk or enable) begin
        if (Q==0 && z && UorD==0 && PorF)
            z=0;
        else if (Q==2 && z && UorD && PorF)
            z=0;
        else
            z = enable ;
    end

endmodule
```



```

module testGenerater (clk ,PorF,UorD ,reset , enable,Q) ;
    output reg clk ,PorF,UorD ,reset , enable ;
    output reg [6:1] Q ;

    // generating input for the counter
    initial begin
        clk = 0;
        PorF = 0;
        UorD = 0 ;
        reset = 0 ;
        enable = 1 ;
        #10ns reset=1 ;
        #410ns PorF = 0;
        UorD = 1 ;
        #420ns PorF = 1;
        UorD = 0 ;
        #420ns PorF = 1;
        UorD = 1 ;
    end

    initial begin
        repeat(88)
            #20ns clk=~clk ;
    end

    //calculating the expected output for this input
    reg flag ,flag1;

    // control about state when PorF UorD change
    always @(negedge reset or PorF or UorD) begin
        flag1=0;
        if (PorF && ~UorD)
            Q=0;
        else if (~PorF && ~UorD)
            Q=2;
        else if (PorF && UorD)
            Q=55;
        else
            Q=31;
        #1ns flag1=1;
    end

    // behavioral counter to calculate expected result
    always @(posedge clk ) begin
        if (enable && flag1) // if the counter enable go to next state
            if (PorF==0 && UorD==0) // choose suitable next state
                case (Q)
                    2: Q=3;
                    3: Q=5;
                    5: Q=7;
                    7: Q=11;
                    11: Q=13;
                    13: Q=17;
                    17: Q=19;
                    19: Q=23;
                    23: Q=29;
                    29: Q=31;
                    31: Q=2;
                endcase
            else if (PorF==0 && UorD==1)
                case (Q)
                    2: Q=31;
                    3: Q=2;
                    5: Q=3;
                    7: Q=5;
                    11: Q=7;
                    13: Q=11;
                    17: Q=13;
                    19: Q=17;
                    23: Q=19;
                    29: Q=23;
                    31: Q=29;
                endcase
    end

```

```

else if (PorF==1 && UorD==0 ) begin
    if (Q≠1)
        flag=0;
    case ({Q,flag})
        0: begin // control about state 1 apper 2 times
            Q=1;
            flag=1;
            end
        3: flag=0;
        2: Q=2;
        4: Q=3;
        6: Q=5;
        10: Q=8;
        16: Q=13;
        26: Q=21;
        42: Q=34;
        68: Q=55;
        110: Q=0;
    endcase
end
else if (PorF==1 && UorD==1) begin
    if (Q≠1)
        flag=0;
    case ({Q,flag})
        0: Q=55;
        2: Q=0;
        3: flag=0;
        4: begin // control about state 1 apper 2 times
            Q=1;
            flag=1;
            end
        6: Q=2;
        10: Q=3;
        16: Q=5;
        26: Q=8;
        42: Q=13;
        68: Q=21;
        110: Q=34;
    endcase
end
end
endmodule

```



```

module TFF (input T,clk,reset,initialState,enable,output reg Q);

    always @(posedge clk or negedge reset ) begin
        if (enable)// if enable= 1 go to next state
            if(~reset) // active low reset
                Q=initialState;
            else
                Q=Q^T;

    end

endmodule

```



```
module testanalyser (clk,Q,Q1);
    input clk;
    input [6:1]Q,Q1 ;

    always @(posedge clk) begin
        $write("RealResult= %d ExpectedResult= %d ",Q1,Q); // print results
        if (Q==Q1) // if expected result equal real result print true
            $display("True");
        else
            $display("False");
    end
endmodule
```



```
module testbench ;

    wire clk ,PorF,UorD ,reset , enable;
    wire [6:1]Q ,Q1;

    // Generate vector tese
    testGenerater t(clk ,PorF,UorD ,reset , enable,Q);

    // find real result
    counter t1(clk ,PorF,UorD ,reset , enable,Q1);

    // go to compare
    testanalyser t2(clk,Q , Q1);

endmodule
```