

BIRZEIT UNIVERSITY

Department of Electrical & Computer Engineering
ENCS3310 - Advanced Digital Systems Design

COURSE PROJECT: Fibonacci/Prime Up/Down Counter

Prepared by: Mohamad Milhem

ID number: 1211053

Instructor: Dr. Abdallatif Abuissa

Date: July 2, 2023

Abstract

The aim of this project is to build a solid base in the process of designing and simulating hardware circuits. The objective is to design a special counter that can count the first 11 numbers of two different sequences, the prime numbers sequence, and the Fibonacci sequence, both in up and down manners. The counter was designed using T-Flip Flops and combination logic and was tested using a proper testbench. Also, the circuit was generated and optimized based on truth tables, execution tables, k-maps, and close observations. The simulations are written using Verilog HDL, compiled and synthesized by a software tool named "Active HDL Student Edition".

Contents

1	Introduction and Background	1
1.1	Combinational Logic	1
1.1.1	2x1 Multiplexer	1
1.2	Sequential Logic	2
1.2.1	T-Filp Flop	2
1.2.2	4-bit synchronous Counter	3
2	Design and Optimization	4
2.1	Requirements	4
2.2	Design Process	5
2.2.1	Native Approach	5
2.2.2	Optimized Approach	6
3	Implementation and Results	14
3.1	T-Flip Flop	14
3.2	MUX2x1	14
3.3	Modified 4-bit counter UP/Down (MOD 11 counter)	15
3.4	Fibonacci converter	16
3.5	Prime converter	17
3.6	Fibonacci/Prime Up/Down Counter	17
3.7	Circuit TestBench	18
3.8	Results	19
4	Conclusion and Future works	21

1 Introduction and Background

1.1 Combinational Logic

Combinational Logic (time-independent logic) is the type of logic that is implemented by Boolean circuits, where the output depends on the values of the current input only.

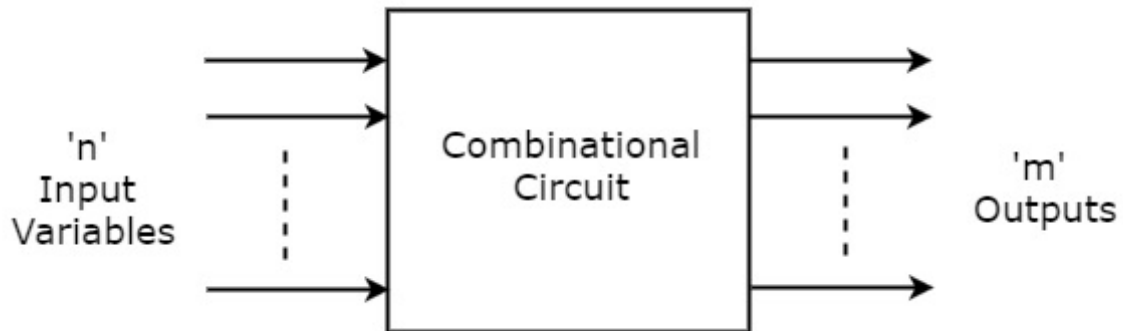


Figure 1.1: Combinational Circuit Diagram.

1.1.1 2x1 Multiplexer

The multiplexer or MUX is a digital switch or a data selector. It is a Combinational Logic circuit that has several input lines, one output line, and several selection lines. Depending on the selection lines, a particular input is routed onto the output line.

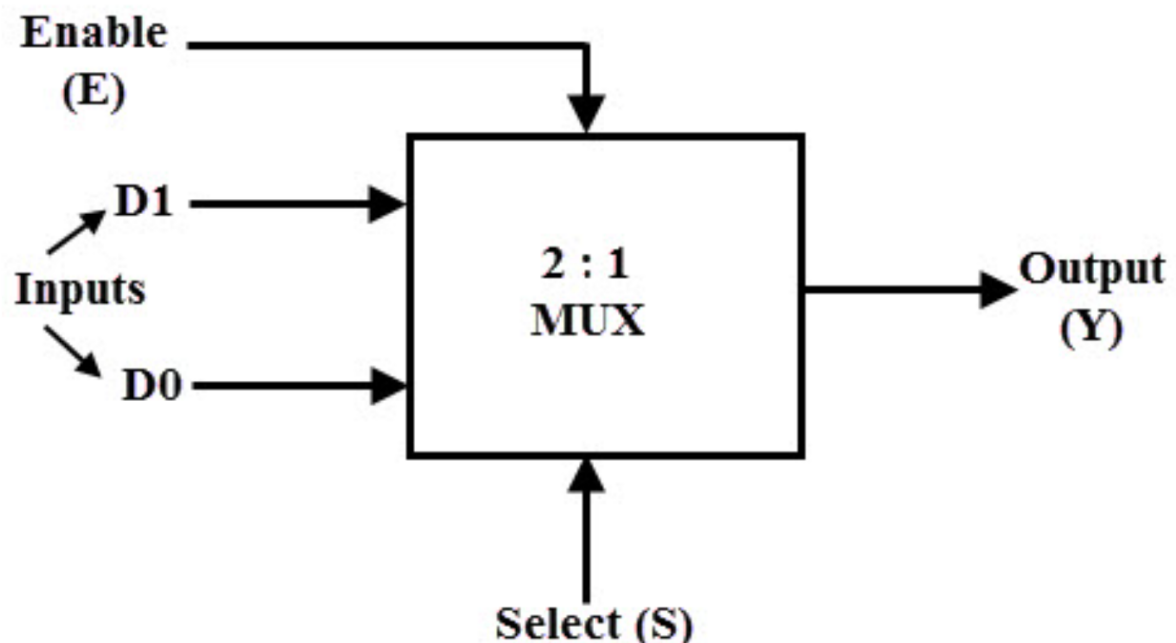


Figure 1.2: MUX2x1 Diagram.

E	D0	D1	S	Y
0	X	X	X	None
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Table 1: Truth table of Mux2x1 with enable input.

1.2 Sequential Logic

Sequential Logic (time-dependent logic) is the type of logic that is implemented by Boolean circuits, where the output depends on the values of the current inputs but also on the previous inputs and the internal state of the circuit. This internal state is maintained by memory elements called flip-flops that store and retain information, allowing the circuit to remember the process data over time.

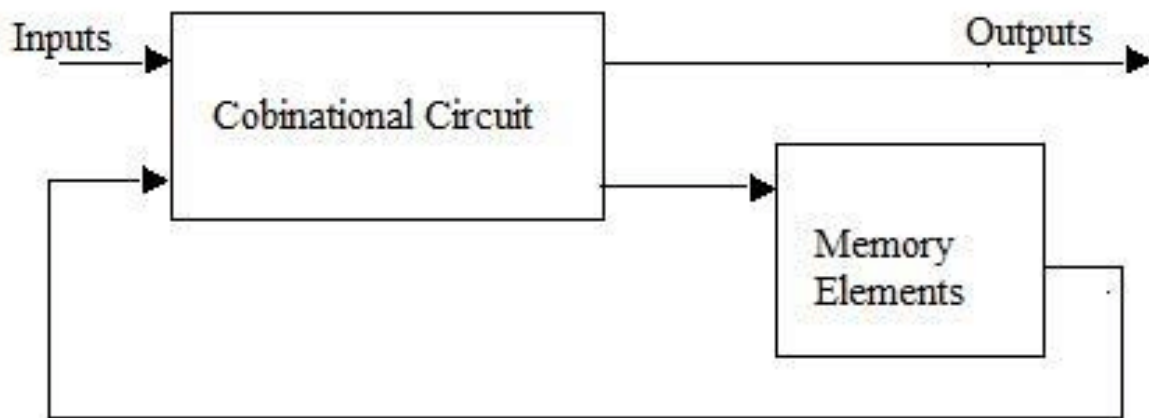


Figure 1.3: Sequential Circuit Diagram.

1.2.1 T-Flip Flop

A T flip-flop, also known as a toggle flip-flop, is a type of sequential logic circuit widely used in digital electronics. The T flip flop has a single input T and two outputs, Q and \bar{Q} . The

input represents the toggle control. When a clock signal triggers the flip-flop, the state of the Q output is inverted if the T input was high.

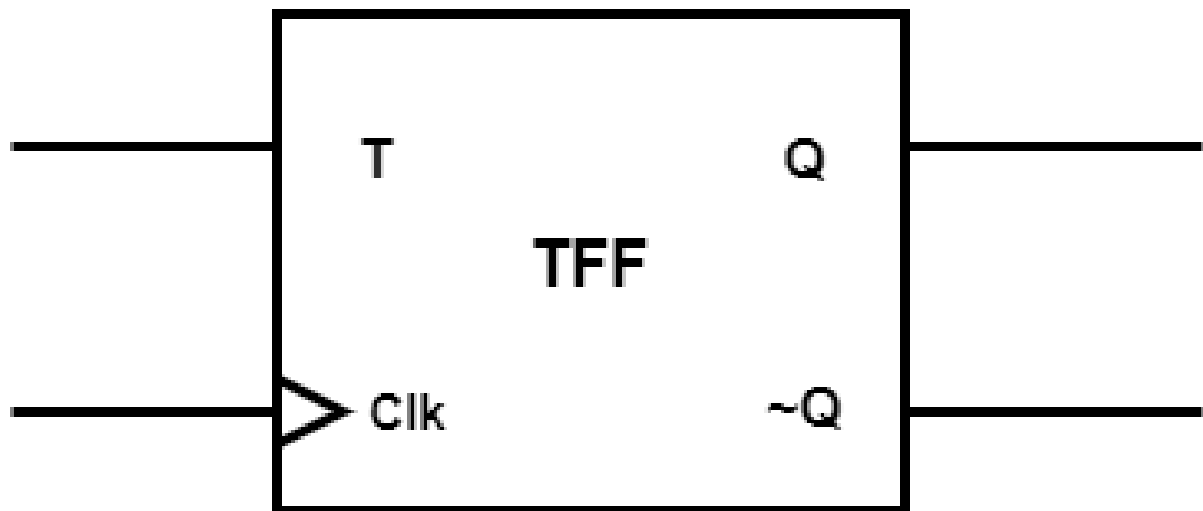


Figure 1.4: T Flip Flop Diagram.





Clk	T	Q(t)	Q(t+1)
	0	0	0
	0	1	1
	1	0	1
	1	1	0

Table 2: T Flip Flop state table.

1.2.2 4-bit synchronous Counter

4-bit counters are sequential logic circuits widely used in digital electronics. They are designed to count from 0 to 15 in binary representation, encompassing all possible combinations of four bits. The counter consists of four flip-flops, each representing one bit of the counter.

The outputs of the 4-bit counter represent the current count in binary format. Each output corresponds to one bit of the count, allowing external components or circuits to observe or utilize the counter's value.

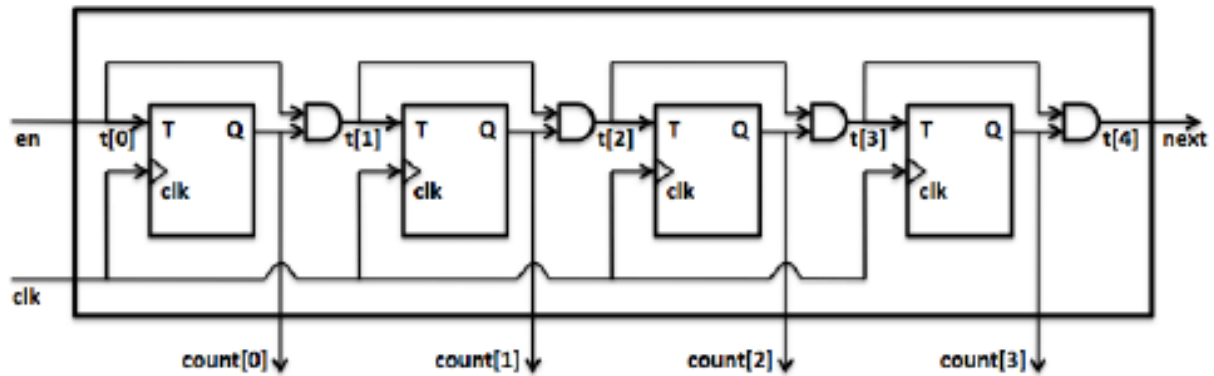


Figure 1.5: 4-bit synchronous counter using T flip flops.

2 Design and Optimization

2.1 Requirements

The requirements of this project are to design a counter that can count two different sequences, the Prime numbers sequence, and the Fibonacci sequence, both in up and down manners.

This counter should be built using T Flip Flops and combinational logic. The counter should count either Prime or Fibonacci depending on a value of an input. Also, it can count them either up or down depending on the value of another input. The counter should count the first 11 terms of both sequences.

The circuit will include reset input (asynchronous) to reset the circuit. Also, enable input (synchronous) to enable/disable the circuit from counting.

Requirements:

1. Counter that can count two different sequences (Prime and Fibonacci).
 - Require input bit (fibPri).
2. Counter should count in both up and down manners.
 - Require input bit (UD).
3. The storage process in the counter should be done using T-Flip flops.
4. The counter should count the first 11 terms of both sequences.
5. Asynchronous Reset input
 - Require input bit (rst).
6. synchronous Enable input

- Require input bit (en).

2.2 Design Process

2.2.1 Native Approach

By writing the first 11 terms of both sequences to analyze our next steps.

sequences	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th
Prime	2	3	5	7	11	13	17	19	23	29	31
Fibonacci	0	1	1	2	3	5	8	13	21	34	55

Table 3: First 11 terms of Prime sequence and Fibonacci sequence.

We should come up with the following observations:

1. The biggest number we have to represent is 55 so we need a maximum of 6 flip flips to represent this number (We will see if you can optimize this)
2. To simplify the implementation (avoiding writing functions and tables of 8 inputs) we can separate the two sequences into 4 circuits (Fib-up, Fib-down, Prime-up, and Prime-down) and choose one based on the input values.
3. Each bit (there are 6 of them) in each state (there are 44 of them if we consider point 2) is a 6 input function, which means each bit needs a 6-bit optimization table to get the expression that will change this bit from it is previous state to the next state based on inputs.

One of the 4 state tables we will get is the following:

Consider FibPri = 0 (Fibonacci), and UD = 0 (counting up).

Present state						Next state					
2^5	2^4	2^3	2^2	2^1	2^0	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1	0	0	0	0	0	1
0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	1	0	0	0	1	0	1
0	0	0	1	0	1	0	0	1	0	0	0
0	0	1	0	0	0	0	0	1	1	0	1
0	0	1	1	0	1	0	1	0	1	0	1
0	1	0	1	0	1	1	0	0	0	1	0
1	0	0	0	1	0	1	1	0	1	1	1

Table 4: State table of Counting Fibonacci up.

In addition to the complexity shown in the observations of the design of such a circuit, we have the case of the second and the third row where the first 1 in Fibonacci should transfer us to the second 1 and the second 1 should transfer to 2, in such case we have the same input for both cases which will require another indicator to find out which "1" is the present state now.

This additional indicator will add more complexity to the circuit and therefore, more cost and more power consumption.

2.2.2 Optimized Approach

We can avoid all this complexity by applying a smart trick inspired by a technique named "Hashing".

let's consider the same problem again but this time, but this time with applying the hashing technique.

sequences	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th
Prime	2	3	5	7	11	13	17	19	23	29	31
Fibonacci	0	1	1	2	3	5	8	13	21	34	55
Counter value	0	1	2	3	4	5	6	7	8	9	10

Table 5: First 11 terms of Prime sequence and Fibonacci sequence with the counter value.

Since we aim to count the first 11 terms in both sequences we can apply the following technique.

1. Build MOD 11 counter or modify a 4-bit counter to count to 10.
2. Take the counter output and map it to the corresponding term in the specified sequence.
3. Set the output to the corresponding term of the specified sequence.

By doing this we will have the following observation which almost solves all the problems we faced considering the native approach.

1. In order to build MOD 11 counter we only need 4 Flip Flops (instead of 6 in the native approach) which means a 33.3% reduction in memory.
2. The complexity is reduced significantly, each bit in the corresponding term in any sequence can be computed as an output of 4 inputs, which can be easily done and optimized using a 2x2 K map.
3. We can modify the counter to count up and down and add the Asynchronous Reset and the synchronous enable easily, and this will be reflected on the output sequence.
4. We can switch between the two sequences freely without resetting the counter, the counter will continue to count as it does, but the mapped value that appears at the output will change based on the FibPri input line.

Design Details

Count in Decimal	Count in binary				Fibonacci						Prime				
	C_3	C_2	C_1	C_0	F_5	F_4	F_3	F_2	F_1	F_0	P_4	P_3	P_2	P_1	P_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	0	0	0	0	1	0	0	0	1	1
2	0	0	1	0	0	0	0	0	0	1	0	0	1	0	1
3	0	0	1	1	0	0	0	0	1	0	0	0	1	1	1
4	0	1	0	0	0	0	0	0	1	1	0	1	0	1	1
5	0	1	0	1	0	0	0	1	0	1	0	1	1	0	1
6	0	1	1	0	0	0	1	0	0	0	1	0	0	0	1
7	0	1	1	1	0	0	1	1	0	1	1	0	0	1	1
8	1	0	0	0	0	1	0	1	0	1	1	0	1	1	1
9	1	0	0	1	1	0	0	0	1	0	1	1	1	0	1
10	1	0	1	0	1	1	0	1	1	1	1	1	1	1	1

Table 6: Using Counter Output as input for the Fibonacci and Prime Circuits.

We can break the previous table into small tables by considering each bit of the output alone.

Starting with Fibonacci bits we can represent each bit with a 2x2 k map of the count bits as inputs.

Fibonacci

		C_1C_0			
		00	01	11	10
C_3C_2	00	0	1	0	1
	01	1	1	1	0
	11	X	X	X	X
	10	1	0	X	1

Figure 2.1: K map For F_0 .

The result is (F_0 least significant bit of Fibonacci)

$$F_0 = C_0 C'_3 + C_1 C'_2 + C_1 C_3 + C'_1 C_2 C'_3 + C'_0 C'_2 C_3$$

		$C_1 C_0$			
		00	01	11	10
$C_3 C_2$	00	0	0	1	0
	01	1	0	0	0
	11	X	X	X	X
	10	0	1	X	1

Figure 2.2: K map For F_1 .

The result is (F_1)

$$F_1 = C_0 C_2 + C_0 C_3 + C'_1 C_2 C_3 + C_1 C'_2 C'_3$$

		$C_1 C_0$			
		00	01	11	10
$C_3 C_2$	00	0	0	0	0
	01	0	1	1	0
	11	X	X	X	X
	10	1	0	X	1

Figure 2.3: K map For F_2 .

The result is (F_2)

$$F_2 = C_0 C'_3 + C_1 C_3$$

		C_1C_0			
		00	01	11	10
C_3C_2	00	0	0	0	0
	01	0	0	1	1
	11	X	X	X	X
	10	0	0	X	0

Figure 2.4: K map For F_3 .

The result is (F_3)

$$F_3 = C_1C_2$$

		C_1C_0			
		00	01	11	10
C_3C_2	00	0	0	0	0
	01	0	0	0	0
	11	X	X	X	X
	10	1	0	X	1

Figure 2.5: K map For F_4 .

The result is (F_4)

$$F_4 = C_0C'_3$$

		C_1C_0			
		00	01	11	10
C_3C_2	00	0	0	0	0
	01	0	0	0	0
	11	X	X	X	X
	10	0	1	X	1

Figure 2.6: K map For F_5 .

The result is (F_5)

$$F_5 = C_0C_2 + C_0C_3$$

Prime

		C_1C_0			
		00	01	11	10
C_3C_2	00	0	1	1	1
	01	1	1	1	1
	11	X	X	X	X
	10	1	1	X	1

Figure 2.7: K map For P_0 .

The result is (P_0)

$$P_0 = C_0 + C_1 + C_2 + C_3$$

		C_1C_0			
		00	01	11	10
C_3C_2	00	1	1	1	0
	01	1	0	1	0
	11	X	X	X	X
	10	1	0	X	1

Figure 2.8: K map For P_1 .

The result is (P_1)

$$P_1 = C_2C_3 + C_2'C_3' + C_0C_2 + C_0'C_1C_3$$

		C_1C_0			
		00	01	11	10
C_3C_2	00	0	0	1	1
	01	0	1	0	0
	11	X	X	X	X
	10	1	1	X	1

Figure 2.9: K map For P_2 .

The result is (P_2)

$$P_2 = C_0 + C_1'C_2 + C_1C_2'C_3$$

		C_1C_0			
		00	01	11	10
C_3C_2	00	0	0	0	0
	01	1	1	0	0
	11	X	X	X	X
	10	0	1	X	1

Figure 2.10: K map For P_3 .

The result is (P_3)

$$P_3 = C_0C_3 + C_0C_2 + C_1C_2'$$

		C_1C_0			
		00	01	11	10
C_3C_2	00	0	0	0	0
	01	0	0	1	1
	11	X	X	X	X
	10	1	1	X	1

Figure 2.11: K map For P_4 .

The result is (P_4)

$$P_4 = C_0 + C_1C_2$$

Using these equations we can implement the required circuit easily as shown in the next section.

3 Implementation and Results

3.1 T-Flip Flop

Our task is to build the special counter using T Flip Flops so we start by building the simulation module for the T Flip Flop.

```
1 module TFF(Q, T, clk, rst);
2
3     input clk, T, rst;
4     output reg Q;
5
6     always @ (posedge clk, negedge rst)
7     begin
8         if (rst == 0)
9             Q = 1'b0;
10        else
11            Q = Q ^ T;
12
13    end
14 endmodule
```

The previous Verilog code describes the T Flip Flop, it has 3 inputs Clk, T, and rst. The Clk line will be connected to the Clock line which will allow the storage element to apply the changes to the stored value when the positive edge of the clock impulse occurs. The T line will represent the data-in line. The rst line will represent the reset signal which will reset the stored value to 0 on the negative edge of the rest signal.

The output Q is the stored value in the T Flip Flop and it is computed by the characteristic equation of the T Flip Flop given by:

$$Q(t+1) = T \oplus Q(t)$$

3.2 MUX2x1

```
1 module MUX2x1(A, B, S, Y);
2     input [5:0] A,B;
3     input S;
4     output reg [5:0] Y;
5
6     always @ (A, B, S)
7     begin
8         if (S == 0)
9             Y = A;
10        else
11            Y = B;
12
13    end
14 endmodule
```

As we know and as mentioned in the background and theory section, the mux2x1 will choose one of two input signals to display on the output line based on the selection signal. Here,

we have two inputs of width 5 which represent the corresponding term of the count in both sequences (Fibonacci and Prime), one of them will be passed to the output bus based on the FibPri input from the user.

3.3 Modified 4-bit counter UP/Down (MOD 11 counter)

```

1 module Mod11UpDownCounter(clk, M, rst, count);
2     input clk, M, rst;
3     output wire [3:0] count;
4     wire [2:0] up, down;
5     wire [2:0] Tin;
6
7
8     TFF T0(count[0], 1, clk, rst
9     & (~(count[0] & count[1] & ~count[2] & count[3]))
10    & (~(count[0] & count[1] & count[2] & count[3])));
11
12    and up0(up[0], ~M, count[0]);
13    and down0(down[0], M, ~count[0]);
14    or Tin0(Tin[0], up[0], down[0]);
15
16    TFF T1(count[1], Tin[0], clk, rst
17    & (~(count[0] & count[1] & ~count[2] & count[3])));
18
19    and up1(up[1], up[0], count[1]);
20    and down1(down[1], down[0], ~count[1]);
21    or Tin1(Tin[1], up[1], down[1]);
22
23    TFF T2(count[2], Tin[1], clk, rst
24    & (~(count[0] & count[1] & ~count[2] & count[3]))
25    & (~(count[0] & count[1] & count[2] & count[3])));
26
27    and up2(up[2], up[1], count[2]);
28    and down2(down[2], down[1], ~count[2]);
29    or Tin2(Tin[2], up[2], down[2]);
30
31    TFF T3(count[3], Tin[2], clk, rst
32    & (~(count[0] & count[1] & ~count[2] & count[3])));
33
34
35 endmodule

```

This module is simulating a 4-bit up/down counter with a reset line. The design is shown in the next figure.

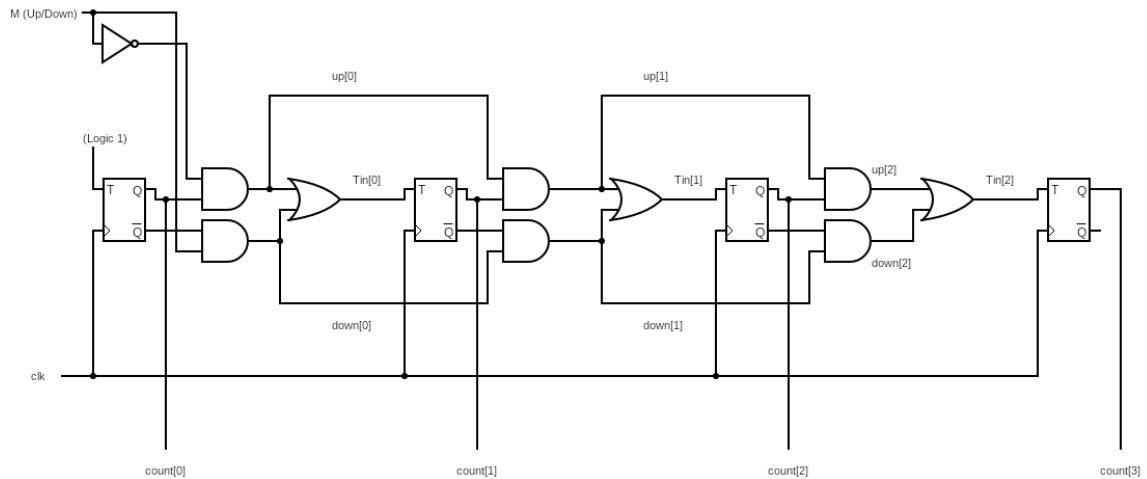


Figure 3.1: Up/down 4-bit counter Diagram.

The reset line is modified to reset the counter when the reset input occurs or after the counter reaches a value of 10.

3.4 Fibonacci converter

```

1 module fib(index, value);
2     input [3:0] index;
3     output [5:0] value;
4
5     assign value[0] = index[3]&(~index[0]) | index[2]&(~index[1])
6     | index[2]&index[0] | (~index[2] & index[1] & ~index[0])
7     | (~index[3] & ~index[1] & index[0]);
8     assign value[1] = index[3]&index[1] | index[3]&index[0]
9     | (~index[2] & index[1] & index[0])
10    | (index[2] & ~index[1] & ~index[0]);
11    assign value[2] = index[3]&(~index[0]) | index[2]&index[0];
12    assign value[3] = index[2]&index[1];
13    assign value[4] = index[3]&(~index[0]);
14    assign value[5] = index[3]&index[1] | index[3]&index[0];
15
16 endmodule

```

Here in this module, we use the equations we came up with from the design section to convert the current value of the counter (0 to 10) to the corresponding value in the Fibonacci sequence (0 to 55).

3.5 Prime converter

```
1 module prime(index, value);
2     input [3:0] index;
3     output [5:0] value;
4
5
6     assign value[0] = index[3] | index[2] | index[1] | index[0];
7     assign value[1] = index[1]&index[0] | (~index[1])&(~index[0])
8     | index[3]&index[1]
9     | (~index[3])&(~index[2])&index[0];
10    assign value[2] = index[3] | (~index[2])&index[1]
11    | index[2]&(~index[1])&index[0];
12    assign value[3] = index[3]&index[0]
13    | index[3]&index[1] | index[2]&(~index[1]);
14    assign value[4] = index[3] | index[2]&index[1];
15    assign value[5] = 0;
16
17 endmodule
```

similar to the previous subsection, in this module, we use the equations we came up with from the design section to convert the current value of the counter (0 to 10) to the corresponding value in the Prime sequence (2 to 31) and that is why we do not need the 5th line of the output.

3.6 Fibonacci/Prime Up/Down Counter

```
1 module circuit(en, fibPri ,UD, clk, rst, OUT);
2
3     input en, fibPri, UD, clk, rst;
4     output [5:0] OUT;
5     wire [3:0] index;
6     wire [5:0] Fib, Pri;
7
8
9     Mod11UpDownCounter counter(en&clk, UD, rst, index);
10    fib f(index, Fib);
11    prime p(index, Pri);
12
13    MUX2x1 M(Fib, Pri, fibPri, OUT);
14
15
16
17 endmodule
```

To put everything together, we take the user inputs to this circuit which are en (enable), FibPri (Fibonacci/Prime), UD (Up/Down), clk (Clock), and rst (Reset). Then initialize an instance of the MOD 11 counter which will count from 0 to 10 up or down counting based on the value of UD line. After that, we will convert the output of the MOD 11 counter to the corresponding value of the Fibonacci and Prime sequence. Finally, we will choose one of the two sequences to connect to the output.

3.7 Circuit TestBench

```
1 module circuit_tb;
2
3     reg en, fibPri, UD, clk, rst;
4     reg [5:0] Fibonacci[0:11], Prime[0:11];
5     wire [5:0] Y;
6     integer index;
7
8     circuit C(en , fibPri, UD, clk, rst, Y);
9
10    initial begin
11        $display("CLK_M_RST_Y");
12        $monitor("%b_%b_%b_%d", clk, UD, rst, Y);
13        index = 0;
14
15        // Hardcode the Fibonacci terms
16        Fibonacci[0] = 0          ; Fibonacci[6] = 8;
17        Fibonacci[1] = 1          ; Fibonacci[7] = 13;
18        Fibonacci[2] = 1          ; Fibonacci[8] = 21;
19        Fibonacci[3] = 2          ; Fibonacci[9] = 34;
20        Fibonacci[4] = 3          ; Fibonacci[10] = 55;
21        Fibonacci[5] = 5          ;
22
23        // Hardcode the Prime terms
24        Prime[0] = 2              ; Prime[6] = 17;
25        Prime[1] = 3              ; Prime[7] = 19;
26        Prime[2] = 5              ; Prime[8] = 23;
27        Prime[3] = 7              ; Prime[9] = 29;
28        Prime[4] = 11             ; Prime[10] = 31;
29        Prime[5] = 13             ;
30
31        clk = 0; UD = 0; rst = 1; fibPri = 1; en = 1;
32        #10 rst = 0;
33        #10 rst = 1;
34
35        #500 UD = 1;
36        #500 UD = 0;
37        #500 rst = 0;
38
39        #100 fibPri = 0;
40        rst = 1;
41
42        #500 UD = 1;
43        #500 UD = 0;
44        #500 rst = 0;
45
46        #100
47        $display("All_Tests_Passed!!");
48        $finish;
49
50    end
```

```

51
52     always #10 clk = ~clk;
53
54     always @ (posedge clk, negedge rst)begin
55         if (rst == 0)
56             index = 0;
57         else begin
58             if (UD == 0)
59                 index = index + 1;
60             else
61                 index = index - 1;
62             index = (index+11) % 11;
63         end
64     end
65
66     always @ (posedge clk)
67     begin
68         #5 // to make sure that the changes were applied.
69         if (fibPri == 0 && Fibonacci[index] != Y)begin
70             $display("Error expected: %d Found: %d", Fibonacci[index], Y);
71             $finish ;
72         end
73
74         else if (fibPri == 1 && Prime[index] != Y)begin
75             $display("Error expected: %d Found: %d", Prime[index], Y);
76             $finish ;
77         end
78     end
79
80 endmodule

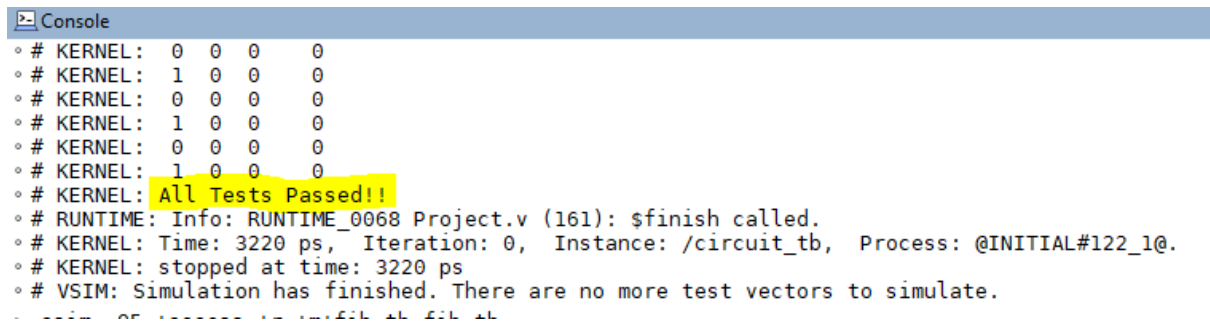
```

To test the implemented circuit, we insatiate the CUT (circuit Under Test) and pass the inputs to it, to keep tracking the answer we hardcoded the values of both sequences in two arrays and define an index that will track the changes of the counter as counting up/down and stop counting. After each change, the value of the output will be compared to the hardcoded value that the index is pointing to it. if all values were right the CUT will pass.

3.8 Results

Based on the testbench implemented, the CUT passed all the test vectors generated. The CUT still has to pass other tests written by other engineers to verify its functionality. The testbench displays the result of the test vector but it is too long to add to this report. Instead, we will add the final result that indicates that all tests passed. To see the instantaneous change in the output when the input is changed see the attached text file "Tests".

⁰Symbol _ indicates the number of spaces inside the quotations.

A screenshot of a console window with a blue header bar labeled "Console". The window contains several lines of text output from a simulation. The first six lines are log messages from the "KERNEL" component, each preceded by a bullet point and showing a sequence of four zeros. The seventh line, which is highlighted in yellow, shows a message from the "KERNEL" component stating "All Tests Passed!!". The eighth line is an information message from the "RUNTIME" component. The ninth line is a log message from the "KERNEL" component showing simulation time and iteration details. The tenth line is a log message from the "KERNEL" component stating the simulation has stopped. The eleventh line is a message from the "VSIM" component stating the simulation has finished. The twelfth line is a partially visible line at the bottom of the console.

```
• # KERNEL: 0 0 0 0
• # KERNEL: 1 0 0 0
• # KERNEL: 0 0 0 0
• # KERNEL: 1 0 0 0
• # KERNEL: 0 0 0 0
• # KERNEL: 1 0 0 0
• # KERNEL: All Tests Passed!!
• # RUNTIME: Info: RUNTIME_0068 Project.v (161): $finish called.
• # KERNEL: Time: 3220 ps, Iteration: 0, Instance: /circuit_tb, Process: @INITIAL#122_1@.
• # KERNEL: stopped at time: 3220 ps
• # VSIM: Simulation has finished. There are no more test vectors to simulate.
• # VSIM: Simulation has finished. There are no more test vectors to simulate.
```

Figure 3.2: Console displays all tests passed.

4 Conclusion and Future works

In this project, we design and implemented an 11 terms Fibonacci/Prime Up/Down counter from scratch. Starting with the theory about some components and moving to the classical design process, finding the optimized version of the design and simulating it using Verilog, and finally, testing it using a testbench.

In addition to following the basic steps of designing a digital circuit, creativity is also required, as we have seen the creative choice of the design has a significant impact on the complexity of the implementation. But on the other hand, we should consider the limitations that a creative design may have, in this case, if the counter for example counts the first 11 prime terms and first 15 Fibonacci terms, the design shown won't work without some modifications.

To take into consideration, the problem stated at the bottom of the previous paragraph can be solved by separating the base counter (which counted from 0 to 10 in our case) into two counters, each of them will count to a different value based on the requirements and the idea of mapping the value of the counter to the corresponding term in the sequence will still be applicable and will reduce some of the complexity, the choice of which counter will work is based on the selection line that will select the sequence we want to present on the output bus.

Worth to mention, that all the simulating and codes shown in this report are written and compiled by a software tool named "Active HDL student edition". Figure 3.1 which shows the diagram of the Up/down 4-bit counter is drawn by an online tool called "Circuit Diagram".

List of Figures

1.1	Combinational Circuit Diagram.	1
1.2	MUX2x1 Diagram.	1
1.3	Sequential Circuit Diagram.	2
1.4	T Flip Flop Diagram.	3
1.5	4-bit synchronous counter using T flip flops.	4
2.1	K map For F_0	8
2.2	K map For F_1	9
2.3	K map For F_2	9
2.4	K map For F_3	10
2.5	K map For F_4	10
2.6	K map For F_5	11
2.7	K map For P_0	11
2.8	K map For P_1	12
2.9	K map For P_2	12
2.10	K map For P_3	13
2.11	K map For P_4	13
3.1	Up/down 4-bit counter Diagram.	16
3.2	Console displays all tests passed.	20

List of Tables

1	Truth table of Mux2x1 with enable input.	2
2	T Flip Flop state table.	3
3	First 11 terms of Prime sequence and Fibonacci sequence.	5
4	State table of Counting Fibonacci up.	6
5	First 11 terms of Prime sequence and Fibonacci sequence with the counter value.	6
6	Using Counter Output as input for the Fibonacci and Prime Circuits.	8