KOTAKONDA ABITHA RAO
194238 - B

Structural Verilog Modelling

The structurae modelling style is the lowest level of abstraction obtained using logic gates.
In this we, assemble several blocks of code and allow the introduction of heirarchy in a design.
The connections between components are specified within component instantiation statements. These statements specify an instance of a component occuring within another component or the circuit. Each component instantiation statement is labelled with an identifier.
The module body provides an internal view, it describes the behaviour or structure of the component.

Abitha

KOTAKONDA ABITHA RAO
194238-B

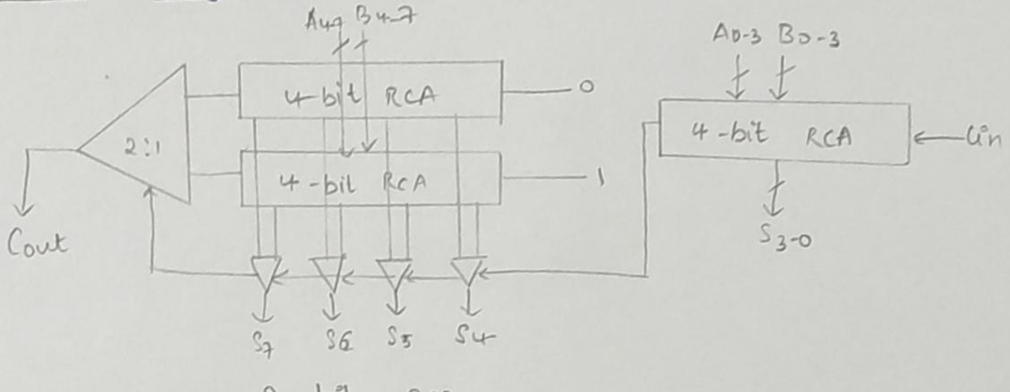
2d] Aim :-
To develop structural verilog model for an 8-bit Carry Select Adder (CSA) using a 4-bit Ripple Carry Adder (RCA)

Software/Tools Used:-
Xilinx Vivado 2019.1

Theory:-
Carry Select Adder is a parallel adder which can add 2 n-bit numbers at a time.
Generally a CSA consists of Ripple Carry Adders and Multiplexers as shown. The advantages of a CSA are: its efficiency over a CLA. The addition of 2 bits is carried out with the help of 2 adders considering both cases, when carry is 0 and when carry is 1. After the calculation is done, correct sum and the correct carry is selected with the help of a MUX.
Here we require 3 4-bit RCA's and 1 2:1 MUX.

KOTAKONDA ABITHA RAO
194238-B

Block Diagram:-



8-bit RCA

Code + Testbench:-

```
module full_adder (a, b, cin, sum, carry);
input a, b, cin;
output sum, carry;

xor (sum, a, b, cin);
and (w1, a, b);
xor (w2, a, b);
and (w3, w2, cin);
or (carry, w3, w1);

endmodule


module RCA_4bit (A, B, CIN, SUM, CARRY);
input [3:0] A;
input [3:0] B;
input CIN;
output [3:0] SUM;
output CARRY;
```

KOTAKONDA ABITHA RAO
194238-B

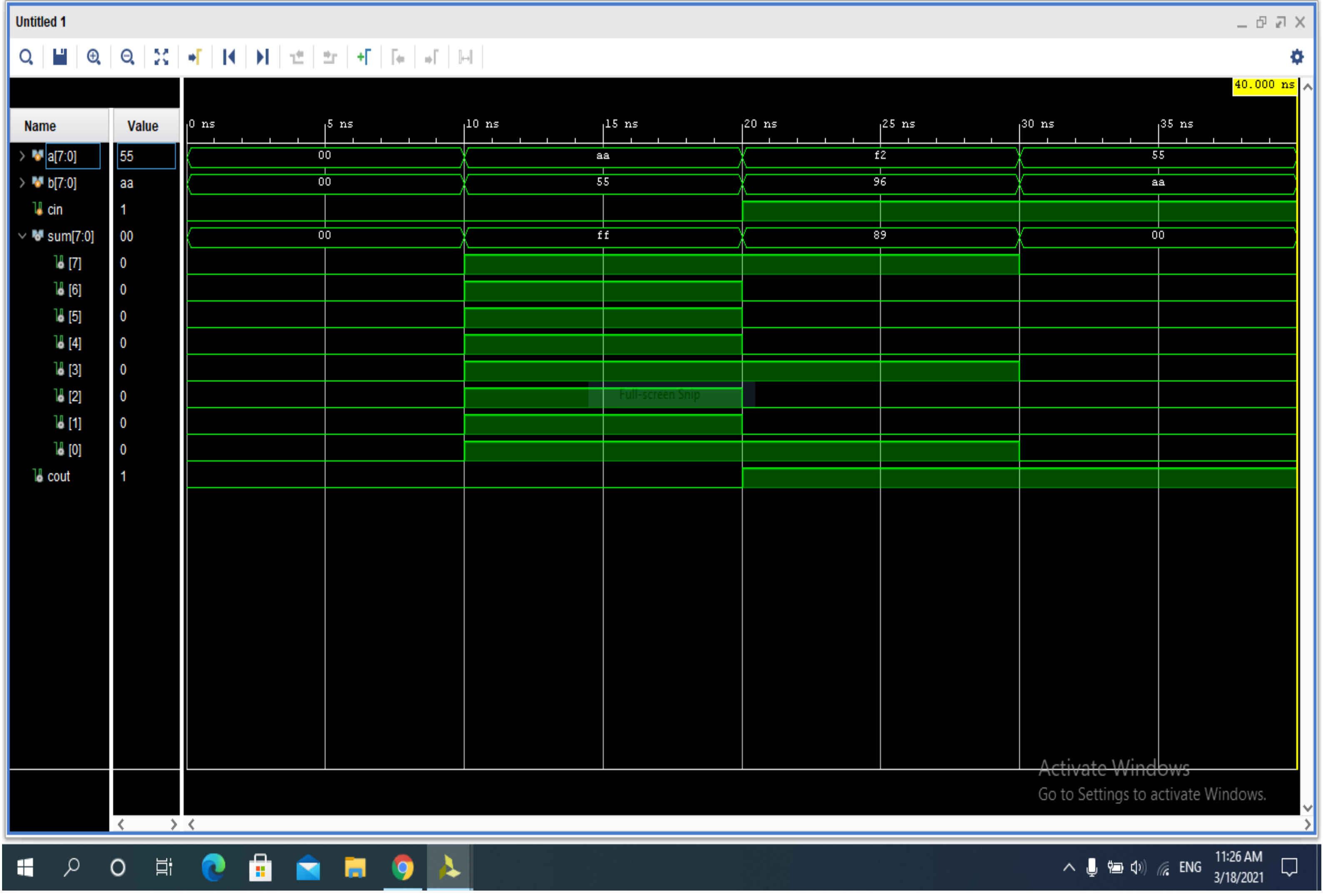
```
wire  W1, W2, W3;
full_adder   FA1 (A[0], B[0], CIN, SUM[0], W1);
full_adder   FA2 ( A[1], B[1], W1, SUM[1], W2);
full_adder   FA3  (A[2], B[2], W2, SUM[2], W3);
full_adder   FA4  (A[3], B[3], W3, SUM[3], CARRY);

endmodule


module MUX (a,b, sel, out);
input a,b;
input sel;
output out;
wire W1, W2

and ( W1 , ~sel, a);
and (W2, sel, b);
or (out, W1, W2);

endmodule


module  csa_8bit (a, b, cin, sum, cout);
input  [7:0] a, b;
input cin;
output [7:0] sum;
output cout;

wire c;
wire [3:0] S1, S2;
wire c1, c2;
```

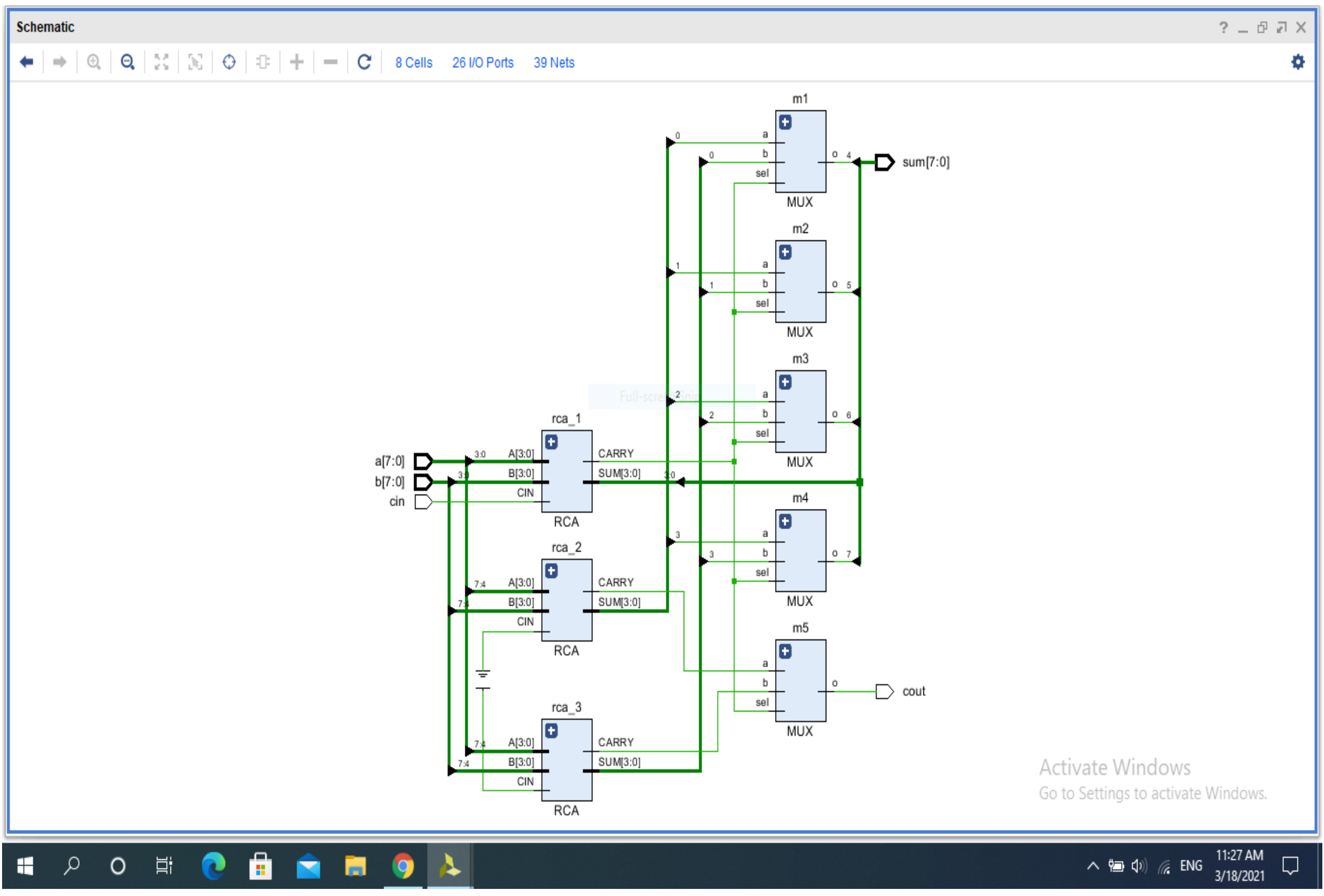
KOTAKONDA ABITHA RAO
194238-B

```
RCA_4bit  rca_1 (.A(a[3:0]), .B(b[3:0]), .CIN(cin),
                .SUM(sum[3:0]), .CARRY(c));
RCA_4bit  rca_2 (.A(a[7:4]), .B(b[7:4]), .CIN(1'b0),
                .SUM(s1[3:0]), .CARRY(c1));

RCA_4bit  rca_3 (.A(a[7:4]), .B(b[7:4]), .CIN(1'b1),
                .SUM(s2[3:0]), .CARRY(c2));

MUX  m1 (.a(s1[0]), .b(s2[0]), .sel(c), .out(sum[4]));
MUX  m2 (.a(s1[1]), .b(s2[1]), .sel(c), .out(sum[5]));
MUX  m3 (.a(s1[2]), .b(s2[2]), .sel(c), .out(sum[6]));
MUX  m4 (.a(s1[3]), .b(s2[3]), .sel(c), .out(sum[7]));
MUX  m5 (.a(c1), .b(c2), .sel(c), .out(cout));

endmodule
```

<u>Result:-</u>

We have instantiated and integrated lower level modules to design / attain higher - level functionality. We used RCA, MUX, to construct a Carry select Adder (8 bits).


Untitled 1
40.000 ns
Name
Value
0 ns
5 ns
10 ns
15 ns
20 ns
25 ns
30 ns
35 ns
a[7:0]
55
00
aa
f2
55
b[7:0]
aa
00
55
96
aa
cin
1
sum[7:0]
00
00
ff
89
00
[7]
0
[6]
0
[5]
0
[4]
0
[3]
0
[2]
0
[1]
0
[0]
0
cout
1
Activate Windows
Go to Settings to activate Windows.
ENG
11:26 AM
3/18/2021


Schematic
8 Cells
26 I/O Ports
39 Nets
m1
m2
m3
m4
m5
a
b
sel
o
MUX
sum[7:0]
cout
a[7:0]
b[7:0]
cin
rca_1
rca_2
rca_3
A[3:0]
B[3:0]
CIN
CARRY
SUM[3:0]
RCA
Activate Windows
Go to Settings to activate Windows.
ENG
11:27 AM
3/18/2021

KOTAKONDA ABITHA RAO

194238-B

2e] Aim:-

To develop a structural verilog model for a 16-bit adder by cascading an 8-bit Kogge stone Adder/Ripple Carry Adder.

Software/Tools used:-

Xilinx Vivado 2019.1

Theory:-

Adder is one of the most commonly used digital feature in the sketch of a digital built-in circuit. Carry is an important part of an adder. To construct a 16-bit adder, we can cascade 2 - 8-bit adders. Here we cascade an 8-bit Kogge-Stone Adder to a Ripple Carry Adder (8 bit). The carry-out of the first stage will be the carry-in of the second stage. By cascading the modules using the basics of structural verilog and primitives, we are able to add two 16-bit numbers and obtain the sum and carry. We can test the working of our verilog code by giving random 16-bit numbers in the testbench.

KOTAKONDA ABITHA RAO
194238 - B

Code + Testbench:-

```
module   KSA (A, B, S, Cin, Cout);
input  [7:0] A, B;
input  Cin;
output  [7:0] S;
output  Cout;
wire  [7:0] G1, P1, P2, G2, G3, P3, G4, P4, G5, P5, C;

assign  G1 = A & B;
assign  P1 = A ^ B;

assign  G2[0] = G1[0];
assign  P2[0] = P1[0]

assign  G2[1] = G1[1] | (P1[1] & G1[0]);
assign  P2[1] = P1[1] & P1[0];

assign  G2[2] = G1[2] | (P1[2] & G1[1]);
assign  P2[2] = P1[2] & P1[1];

assign  G2[3] = G1[3] | (P1[3] & G1[2]);
assign  P2[3] = P1[3] & P1[2];

assign  G2[4] = G1[4] | (P1[4] & G1[3]);
assign  P2[4] = P1[4] & P1[3];

assign  G2[5] = G1[5] | (P1[5] & G1[4]);
assign  P2[5] = P1[5] & P1[4];

assign  G2[6] = G1[6] | (P1[6] & G1[5]);
assign  P2[6] = P1[6] & P1[5];
```

KOTAKONDA ABITHA RAO
194238 - B

```
assign  G2[7]= G1[7] | (P1[7] & G1[6]);
assign  P2[7]= P1[7] & P1[6];

assign  G3[0]= G2[0];
assign  P3[0]= P2[0];

assign  G3[1] = G2[1];
assign  P3[1] = P2[1];

assign  G3[2] = G2[2] | (P2[2] & G2[0]);
assign  P3[2] = P2[2] & P2[0];

assign  G3[3] = G2[3] | (P2[3] & G2[1]);
assign  P3[3]= P2[3] & P2[1];

assign  G3[4] = G2[4] | (P2[4] & G2[2]);
assign  P3[4] = P2[4] & P2[2];

assign  G3[5] = G2[5] | (P2[5] & G2[3]);
assign  P3[5] = P2[5] & P2[3];

assign  G3[6] = G2[6] | (P2[6] & G2[4]);
assign  P3[6] = P2[6] & P2[4];

assign  G3[7] = G2[7] | (P2[7] & G2[5]);
assign  P3[7] = P2[7] & P2[5];

assign  G4[0] = G3[0];
assign  P4[0] = P3[0];

assign  G4[1] = G3[1];
assign  P4[1] = P3[1]

assign  G4[2] = G3[2];
assign  P4[2] = P3[2];
```

Abitha

KOTAKONDA ABITHA RAO
194238-B

```
assign  G4[3]= G3[3]|(P3[3]& G3[0]);
assign  P4[3]= P3[3]& P3[0];

assign  G4[4]= G3[4]|(P3[4]& G3[1]);
assign  P4[4]= P3[4]& P3[1];

assign  G4[5] = G3[5]|(P3[5]& G3[2]);
assign  P4[5]= P3[5]& P3[2];

assign  G4[6]= G3[6]|(P3[6]& G3[3]);
assign  P4[6] = P3[6]& P3[3];

assign  G4[7] = G3[7]|(P3[7]& G3[4]);
assign  P4[7]= P3[7]& P3[4];

assign  G5[0] = G4[0];
assign  P5[0]= P4[0];

assign  G5[1] = G4[1];
assign  P5[1]= P4[1];

assign  G5[2] = G4[2];
assign  P5[2] = P4[2];

assign  G5[3] = G5[3];
assign  P5[3] = P4[3];

assign  G5[4] = G4[4]|(P4[4]& G3[0]);
assign  P5[4] = P4[4]& P4[0];

assign  G5[5]= G4[5]|(P4[5]& G3[1]);
assign  P5[5] = P4[5]& P4[1];

assign  G5[6] = G4[6]|(P4[6]& G3[2]);
assign  P5[6]= P4[6]& P4[2];
```

Abitha

```
assign  G5[7] = G4[7] | (P4[7] & G3[3]);
assign  P5[7] = P4[7] & P4[3];

assign  C=G5;
assign  S[0]= P1[0] ^ Cin;
assign  S[1]= P1[1] ^ C[0];
assign  S[2]= P1[2] ^ C[1];
assign  S[3] = P1 [3] ^ C[2];
assign  S[4] = P1[4] ^ C[3];
assign  S[5] = P1 [5] ^ C[4];
assign  S [6] = P1 [6] ^C[5];
assign  S [7] = P1[7] ^ C [6];
assign  Cout = C[7];

endmodule


module  KSA_tb();
reg  [7:0] a,b;
reg  c_in;
wire  [7:0] sum;
wire  c_out;

KSA dut (.A(a), .B(b), .Cin (c_in), .S(sum), .Cout
          (c-out));

initial begin
 a = 8'b 10100101 ; b= 8'b 11110100; c_in =0;
end

endmodule
```

Abitha

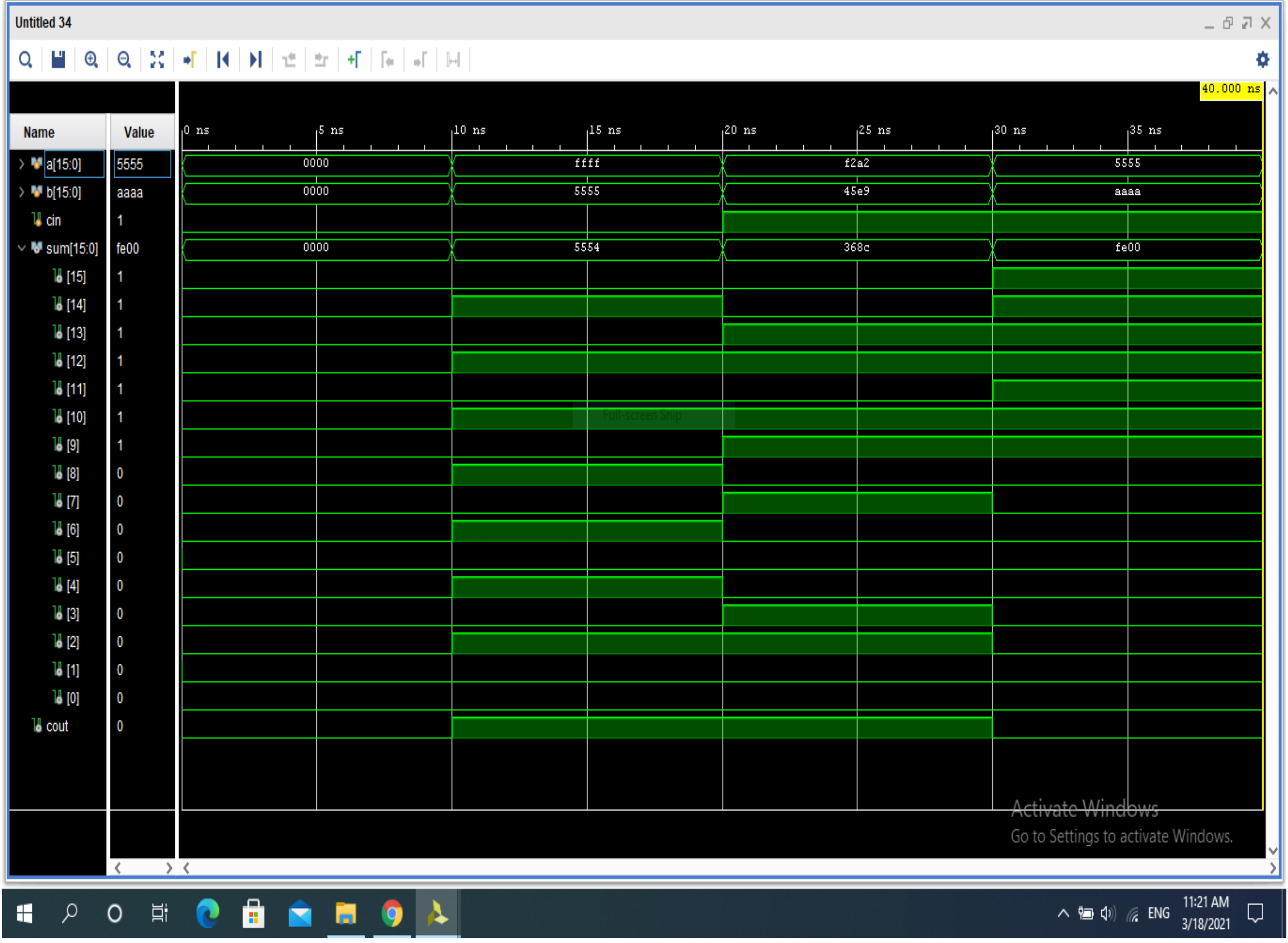
KOTAKONDA ABITHA RAO
194238 - B

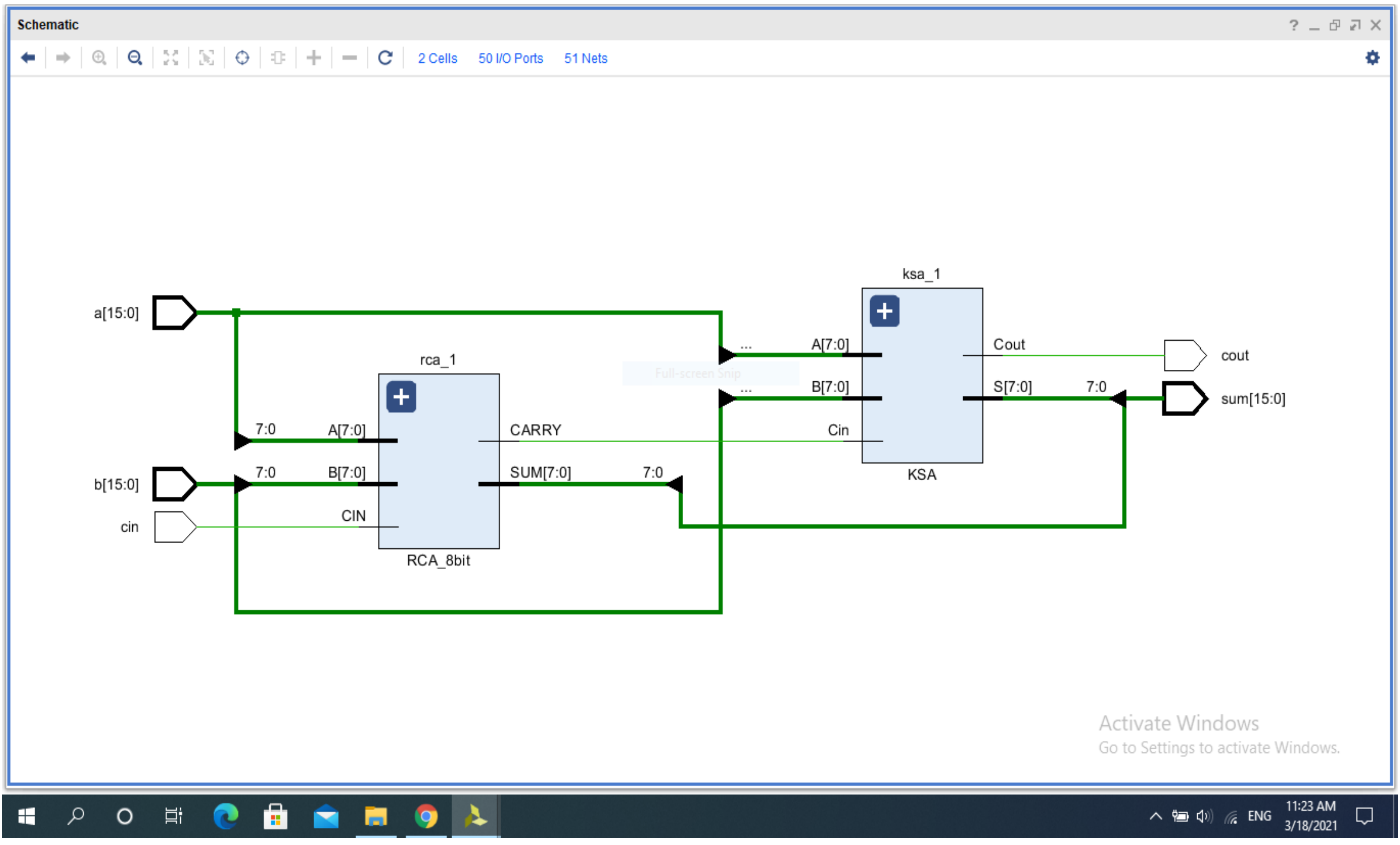
```
module RCA_8bit (A,B, CIN , SUM , CARRY) ;

input [7:0]A;
input [7:0]B;
input CIN;
output [7:0]SUM;
output CARRY;
wire W1;

RCA_4bit RCA1 (A[3:0],B[3:0], CIN, SUM[3:0], W1);
RCA_4bit RCA2 (A[7:4],B[7:4],W1, SUM[7:4], CARRY);

endmodule

module cascade_16bit (a, b, cin, sum, cout);

input [15:0] a,b;
input cin;
output [15:0] sum;
output cout;

wire c;

RCA_8bit rca-1(.A (a[7:0]) , .B(b[7:0]) , .CIN(cin)
               .SUM (sum[7:0]), . CARRY (c));

KSA    ksa-1 (.A (a[15:8]), . B(b[15:8]), .s(sum[15:8])
              .Cin(c),  . Cout (cout));

endmodule
```

KOTAKONDA ABITHA RAO
194238-B

```
module cascade_16bit_tb();
reg [15:0] a,b;
reg cin;
wire [15:0] sum;
wire cout;

cascade_16bit dut (.a(a), .b(b), .cin(cin), .sum(sum),
                     .cout(cout));

initial begin
a=16'b0000000000000000; b=16'b0000000000000000;
cin=0;
#10
a=16'b1111111111111111 ; b=16'0101010101010101;
cin=0;
#10
a=16'b1111 0010101 00010; b=16'b01000101111 01001;
cin=1;
#10
a=16'b0101010101010101 ; b=16'b1010101010101010;
cin=1;
#10
$stop;
end

endmodule
```

Result :-
Here we observe that instantiation of lower level modules for the construction of higher level module is indeed a powerful concept in HDL.


Untitled 34
Name
Value
a[15:0] 5555
b[15:0] aaaa
cin 1
sum[15:0] fe00
[15] 1
[14] 1
[13] 1
[12] 1
[11] 1
[10] 1
[9] 1
[8] 0
[7] 0
[6] 0
[5] 0
[4] 0
[3] 0
[2] 0
[1] 0
[0] 0
cout 0
40.000 ns
0 ns
5 ns
10 ns
15 ns
20 ns
25 ns
30 ns
35 ns
0000
ffff
f2a2
5555
0000
5555
45e9
aaaa
0000
5554
368c
fe00


Schematic
2 Cells
50 I/O Ports
51 Nets
a[15:0]
b[15:0]
cin
rca_1
A[7:0]
B[7:0]
CIN
CARRY
SUM[7:0]
RCA_8bit
7:0
7:0
7:0
ksa_1
A[7:0]
B[7:0]
Cin
Cout
S[7:0]
KSA
7:0
cout
sum[15:0]

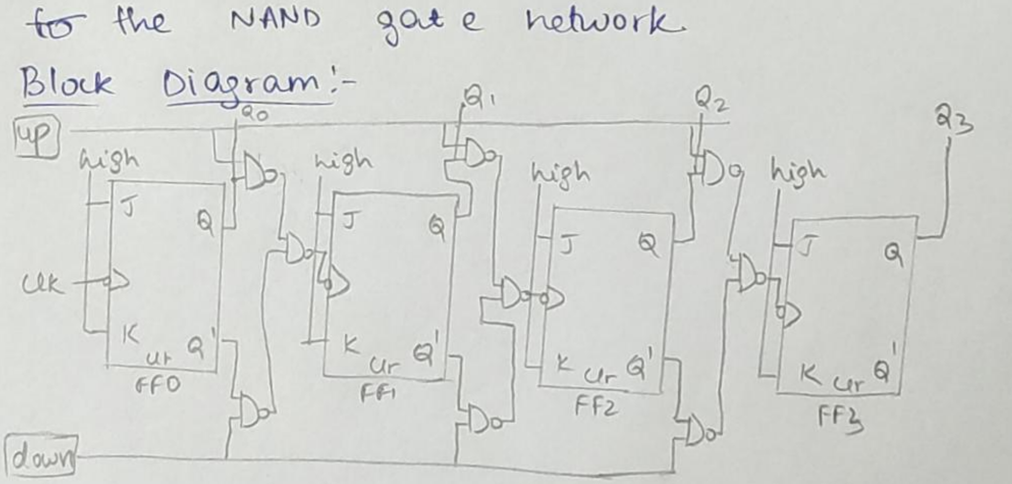
KOTAKONDA ABITHA RAO
194238-B

2f) Aim:-
To develop a 4 bit up-down counter asynchronous verilog model.

Software Used:-
Xilinx Vivado 2019.1

Theory:-
By adding up the ideas of UP and DOWN counters, we can design an asynchronous up/down counter. It can count either ways, up to down or down to up, based on the clock signal input.
The up/down counter is slower than up counter or a down counter, because the addition propagation delay will get added ~~to~~ for the NAND gate network.

Block Diagram:-



UP-DOWN counter

KOTAKONDA ABITHA RAO

194238-B

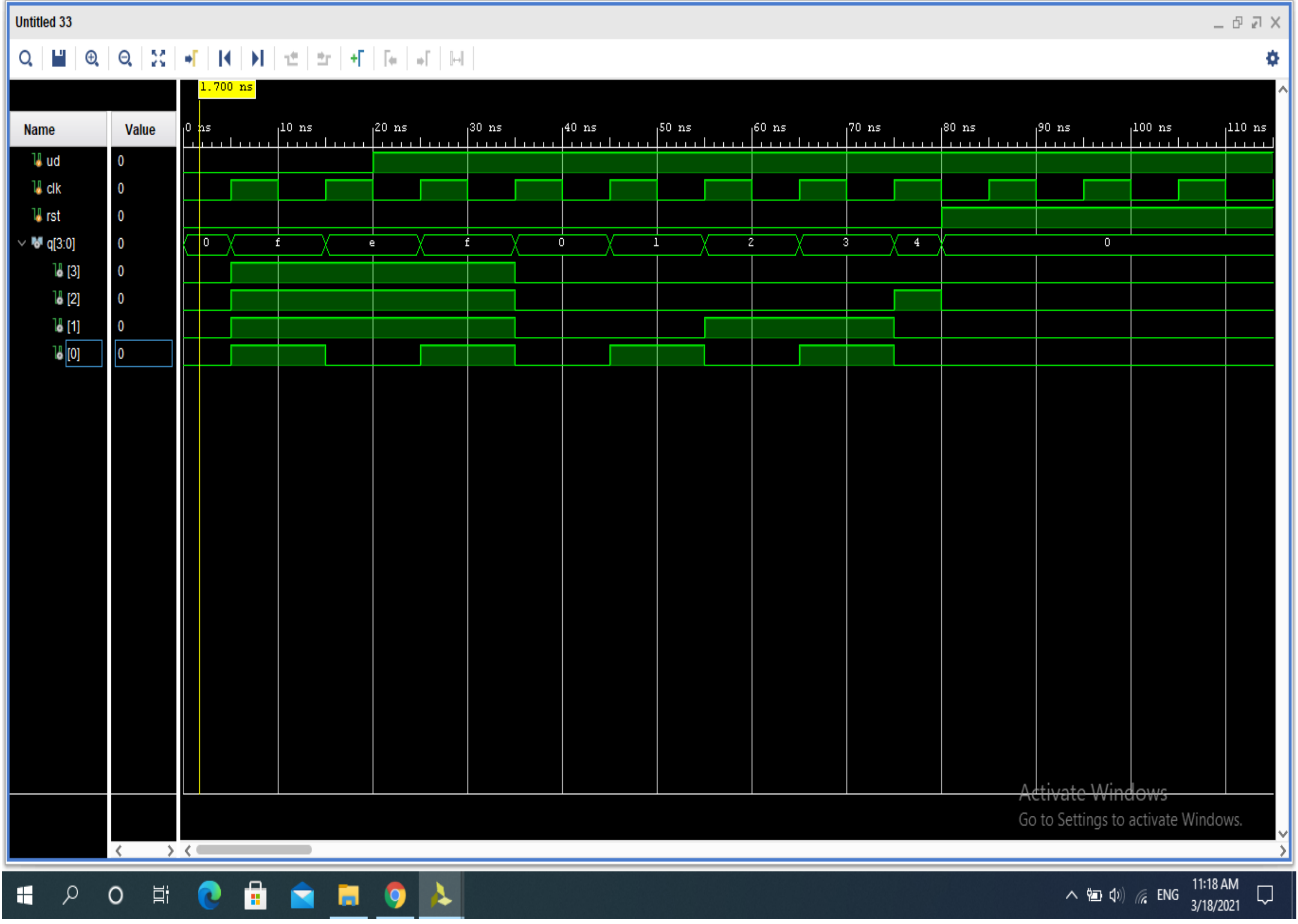
Code + Test bench:-

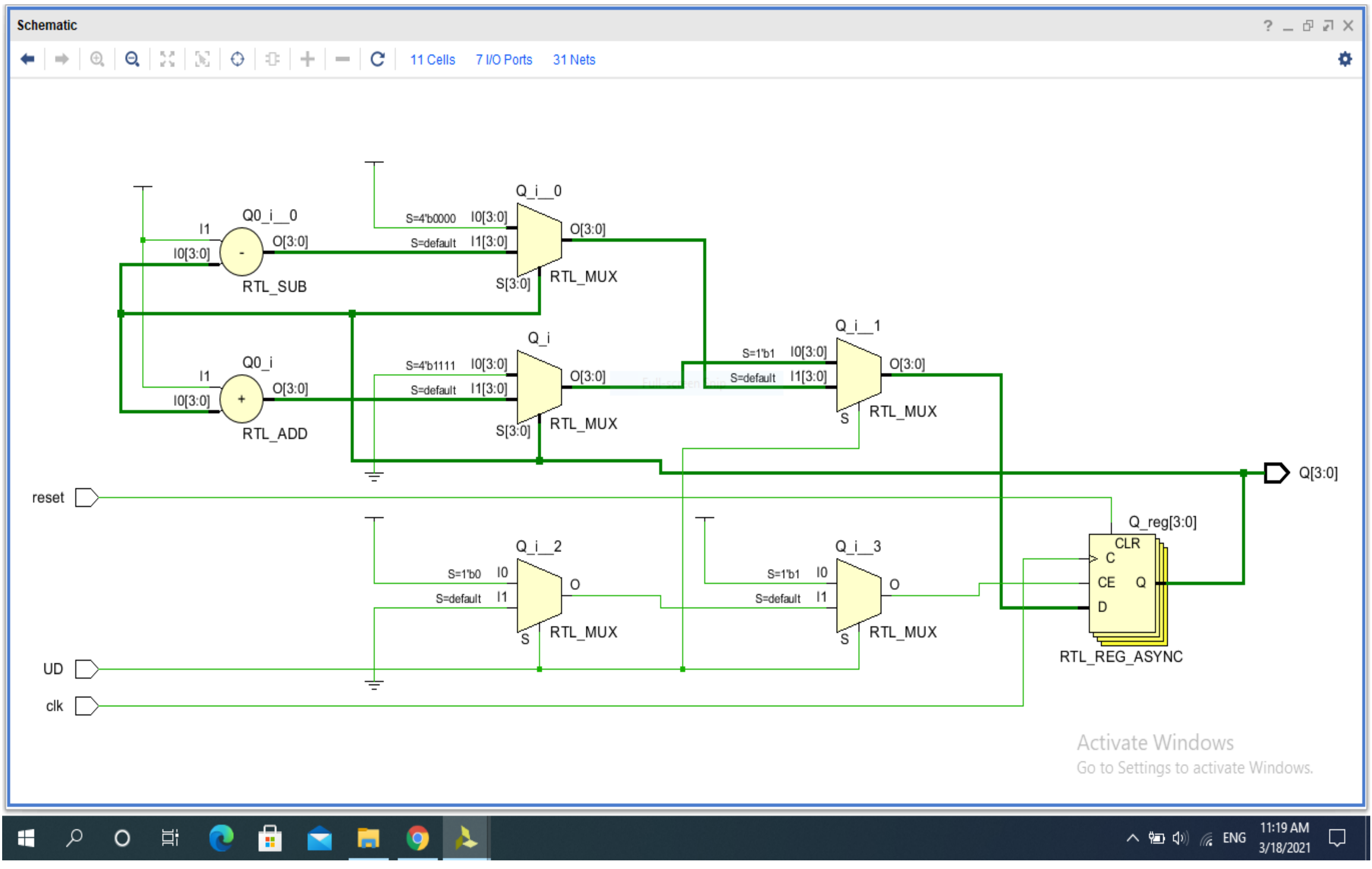
```
module  counter_4bit (UD, clk, reset, Q);
input  UD;
input clk;
input reset;
output [3:0] Q;
reg [3:0]  Q = 4'b0000;

always @ (posedge clk  or posedge  reset)
if (reset ==1)
   Q <= 4'b0000;
else
   if (UD==1)
       if (Q == 4'b1111)
           Q <= 4'b0000;
       else
           Q <= 4'b0001 + Q;

   else if (UD==0)
       if (Q = 4'b0000);
           Q <= 4'b1111;
       else
           Q <= Q - 4'b0001 ;

endmodule


module  counter_4bit_tb ();
reg ud, clk, rst;
wire [3:0] q;
```

KOTAKONDA ABITHA RAO
194238-B

```
counter_4bit dut(.UD(ud), .Q(q), .clk(clk),
                 .reset(rst));

initial
   clk = 1'b0;
always
   #5 clk = ~clk;

initial begin
#0
ud=0; rst=0;
#20;
ud=1; rst=0;
#60.
ud=1; rst=1;
end

endmodule
```

<u>Result:-</u>
We have written the verilog code for a 4-bit up-down counter. To check if it is working fine, we wrote the testbench and observed the waveform.


Untitled 33
1.700 ns
Name
Value
ud 0
clk 0
rst 0
q[3:0] 0
[3] 0
[2] 0
[1] 0
[0] 0
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
90 ns
100 ns
110 ns
0 f e f 0 1 2 3 4 0
Activate Windows
Go to Settings to activate Windows.
11:18 AM
3/18/2021
ENG


Schematic
11 Cells
7 I/O Ports
31 Nets
Q0_i__0
I1
I0[3:0]
O[3:0]
RTL_SUB
Q_i__0
S=4'b0000
S=default
I0[3:0]
I1[3:0]
O[3:0]
S[3:0]
RTL_MUX
Q0_i
RTL_ADD
Q_i
S=4'b1111
RTL_MUX
Q_i__1
S=1'b1
S=default
S
RTL_MUX
Q[3:0]
reset
Q_reg[3:0]
CLR
C
CE
D
Q
RTL_REG_ASYNC
Q_i__2
S=1'b0
I0
I1
O
RTL_MUX
Q_i__3
S=1'b1
RTL_MUX
UD
clk
Activate Windows
Go to Settings to activate Windows.
11:19 AM
3/18/2021
ENG