Digital
System Design Lab Report - 3

—

By Anshuman Mishra

Roll No : 194110

ECE 2nd Year

Section : A

Software : Xilinx Vivado Version 2020.2

2.a) 16:1 MUX using 4:1 MUX

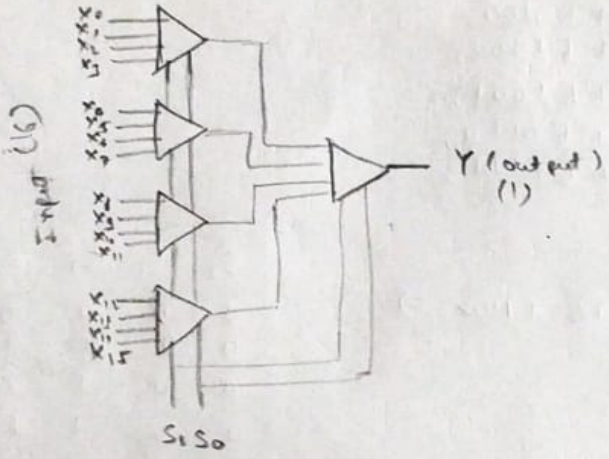
Anushkaran Mishra
19110 ECE Section: A

DSD Lab Report – 3
(a)

Aim: To develop structural verilog Model for 16:1 MUX using 4:1 MUX

Software used: Vivado.

Theory: MUX is a combinational ckt. that has max. 2^n data inputs where n is the number of selection lines. A typical example being 4:1 MUX. The higher order multiplexers can be designed using lower order mux's. for instance 16:1 using 4:1 we in total require 5 MUX. 4 MUX serve input purpose whilst 1 MUX for outputs.



Application – Data Buses, memory, ALU, Communication System

Code:

```
module mux41 (input S0,S1,input [3:0]X, output Y);
    assign Y = (~S0 & ~S1 & X[0]) | (~S0 & S1 & X[2]) | (S0 & ~S1 & X[1]) |
               (S0 & S1 & X[3]);
endmodule
module MUX161 (input [3:0]S, input [15:0]X, output Y);
    wire [3:0] Z;
    mux41 m1 (S[0],S[1], X[3:0], Z[0]),
          m2 (S[0],S[1], X[7:4], Z[1]),
          m3 (S[0],S[1], X[11:8], Z[2]),
          m4 (S[0],S[1], X[15:12], Z[3]);
```

Anshuman Mishra
19110

Section A

```
    mux41 m5(s[2], s[3], z, Y);
endmodule
```

Testbench:

```
module tb;
    reg [15:0] D; reg [3:0] S;
    wire Y;
    MUX16_1 M(.D(D), .S(S), .Y(Y));
    initial begin
        D = 16'b0000111111000011;
        S = 4'b0000;
        #10 S = 4'b0100;
        #10 S = 4'b1000;
        #10 S = 4'b1001;
        #10 S = 4'b0110;
        #10 S = 4'b1111;
    end
endmodule
```

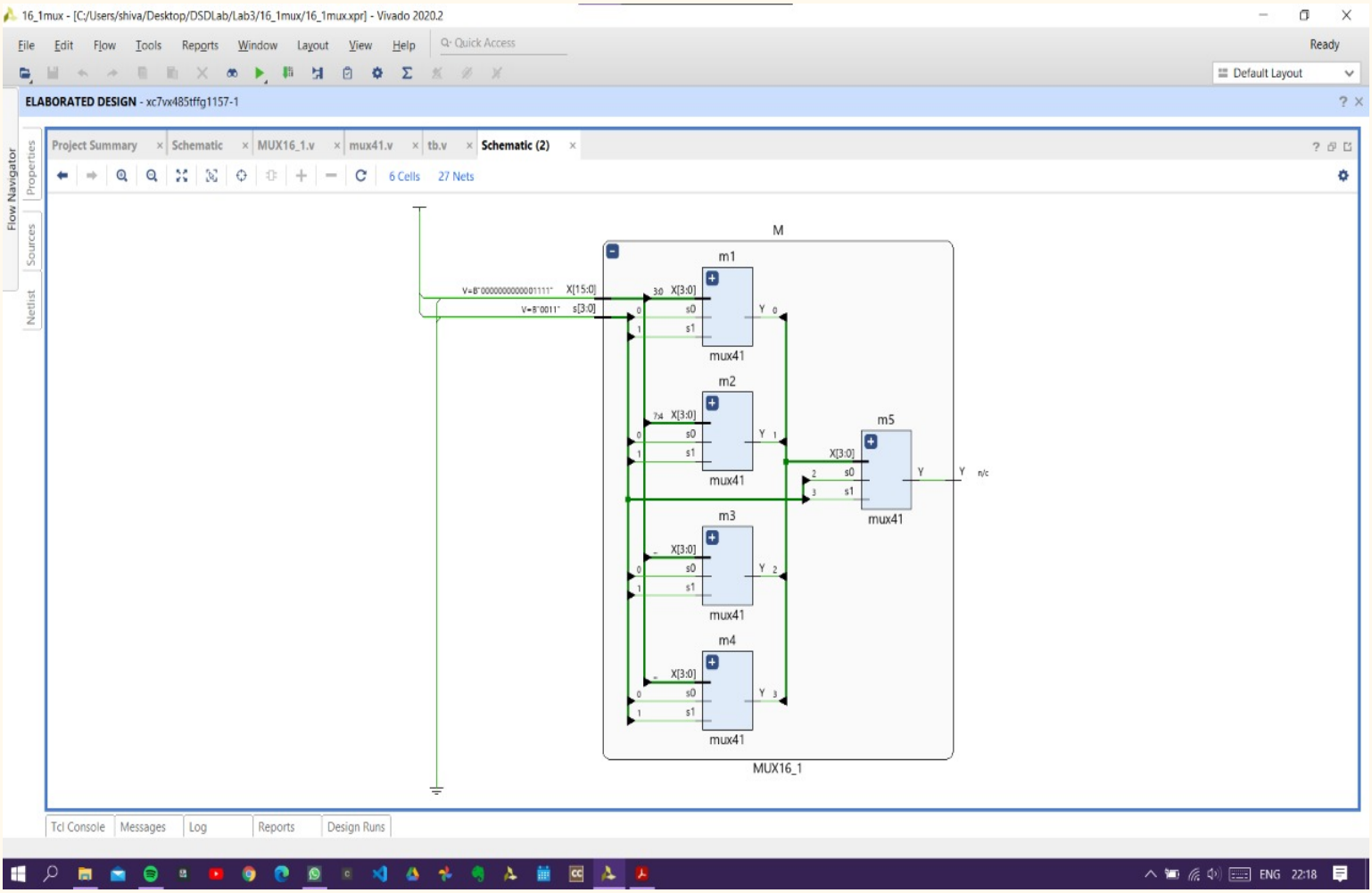
Result: Truth Table for 16:1 MUX ⇒

S_3	S_2	S_1	S_0	Y
0	0	0	0	x[0]
0	0	0	1	x[1]
0	0	1	0	x[2]
0	0	1	1	x[3]
0	1	0	0	x[4]
0	1	0	1	x[5]
0	1	1	0	x[6]
0	1	1	1	x[7]
1	0	0	0	x[8]
1	0	0	1	x[9]
1	0	1	0	x[10]
1	0	1	1	x[11]
1	1	0	0	x[12]
1	1	0	1	x[13]
1	1	1	0	x[14]
1	1	1	1	x[15]

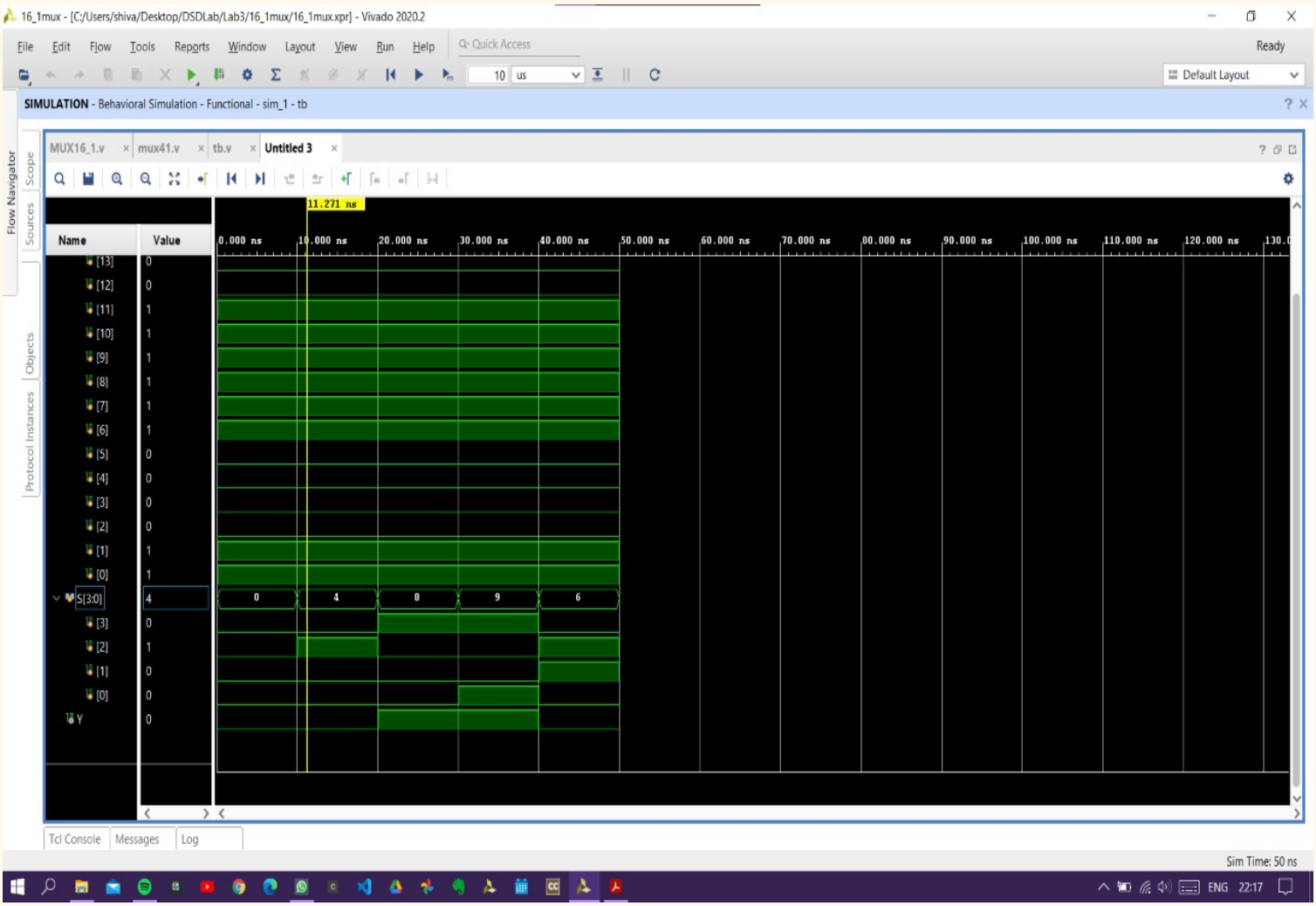
Conclusion: 16:1 Mux was designed using 4:1 MUX. following structural design style.

- The code was written in vivado. and was thoroughly tested against various inputs.
- RTL schematic was generated using Vivado's tool.

RTL Schematic:



Waveform


16_1mux - [C:/Users/shiva/Desktop/DSDLab/Lab3/16_1mux/16_1mux.xpr] - Vivado 2020.2
File Edit Flow Tools Reports Window Layout View Run Help
Quick Access
Ready
Default Layout
SIMULATION - Behavioral Simulation - Functional - sim_1 - tb
MUX16_1.v
mux41.v
tb.v
Untitled 3
11.271 ns
Name
Value
S[3:0]
Y
Tcl Console
Messages
Log
Sim Time: 50 ns
ENG 22:17

b) Ripple Carry Adder

Anuthamam Mishra
19110

ECE – 2-2 · Section-A

(b)

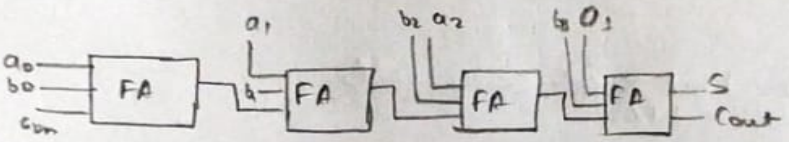
Aim: To design 4 bit ripple carry adder using Full adder in Verilog

Software: Xilinx Vivado (2020.2).

Theory:

Full adder is the adder which adds three inputs and produces two outputs (sum and carry). The three inputs namely (A, B, Cin i.e. 2 bits to be added and carry from previous stage, For initial state Cin = 0).

4 bit Ripple Carry adder: is an adder by cascading 4 Full adders such that output of one is input to other. Process of summation and circuit diagram below develop more intuition



a_3 a_2 a_1 a_0 c_{in}
b_3 b_2 b_1 b_0
S_3 S_2 S_1 S_0

remember $S_i = a_i \oplus b_i \oplus C_{in}$

$C_i = A \cdot B + (A \oplus b) C_{in}$

Code:

```
module FA (input a, b, Cin, output s, cout);
    assign s = a ^ b ^ Cin,
           cout = (a & b) | [(a ^ b) & Cin];
endmodule,

module Ripple_Adder ( input [3:0] a, b, input cin, output [3:0] S, Cout);
    wire [3:0] sum;
    FA A1 (a[0], b[0], cin, S[0], C[0]);
    FA A1(a[0], b[0], cin, S[0], C[0]),
       A2(a[1], b[1], C[0], S[1], C[1]),
       A3 (a[2], b[2], C[1], S[2], C[2]),
       A4 (a[3], b[3], C[2], S[3], C[3]);

endmodule
```

Anshuman Mishra
19110
ECE – Section A

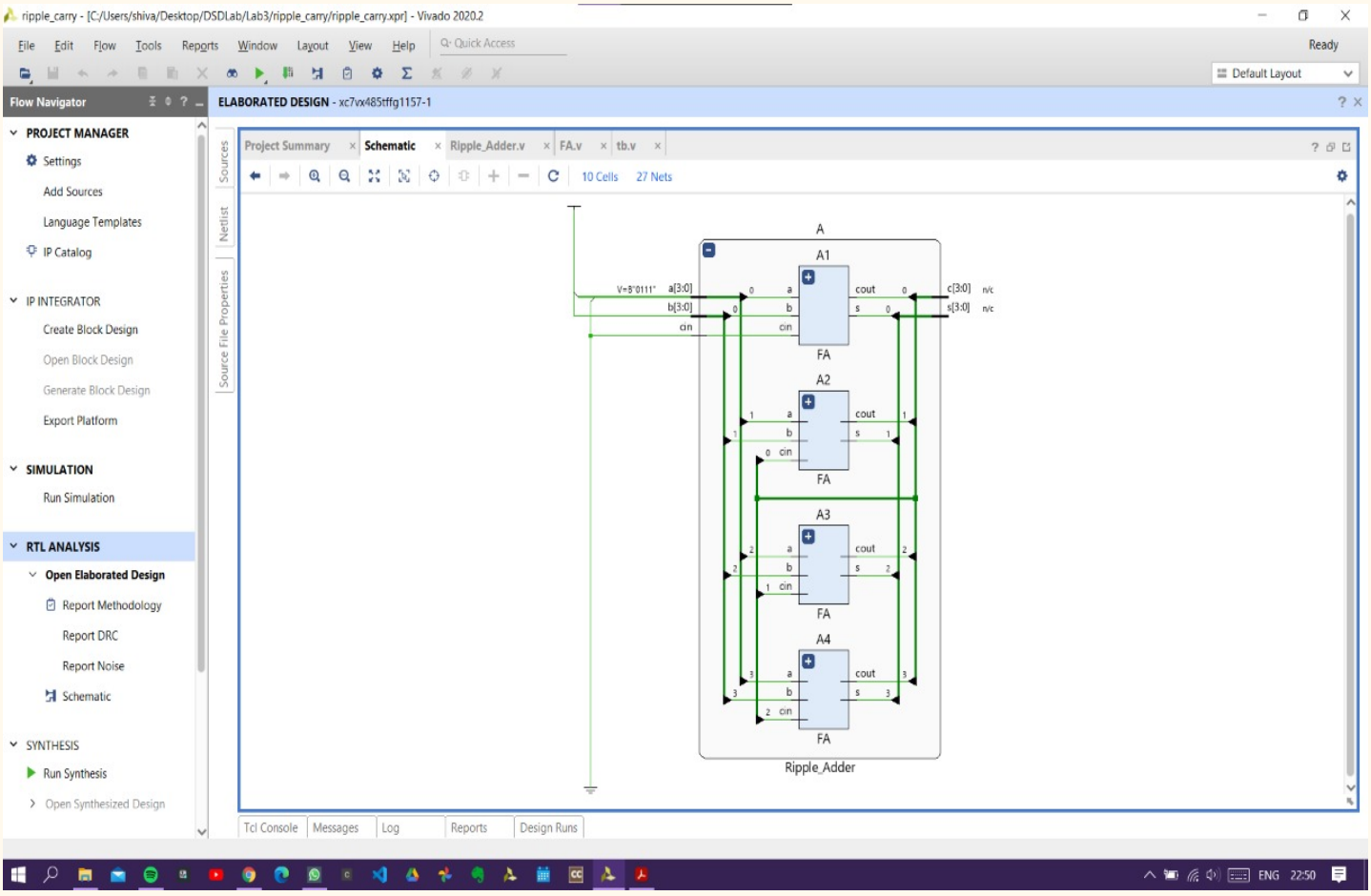
```
module tb;
    reg [3:0] a, b, s, c;
    wire cin = 1'b0;  // carry = 0 for initialization
    Ripple_Adder A( .a(a), .b(b), .cin(cin), .s(s), .c(c) );
    initial begin
        a = 4'd7; b = 4'd5;
        #10 a = 4'd1; b = 4'd9;
        #10 a = 4'd15; b = 4'd7;
        #10 a = 4'd0; b = 4'd1;
        #10 a = 4'd11; b = 4'd11;
        #10 a = 4'd7; b = 4'd15;
    end
endmodule.
```

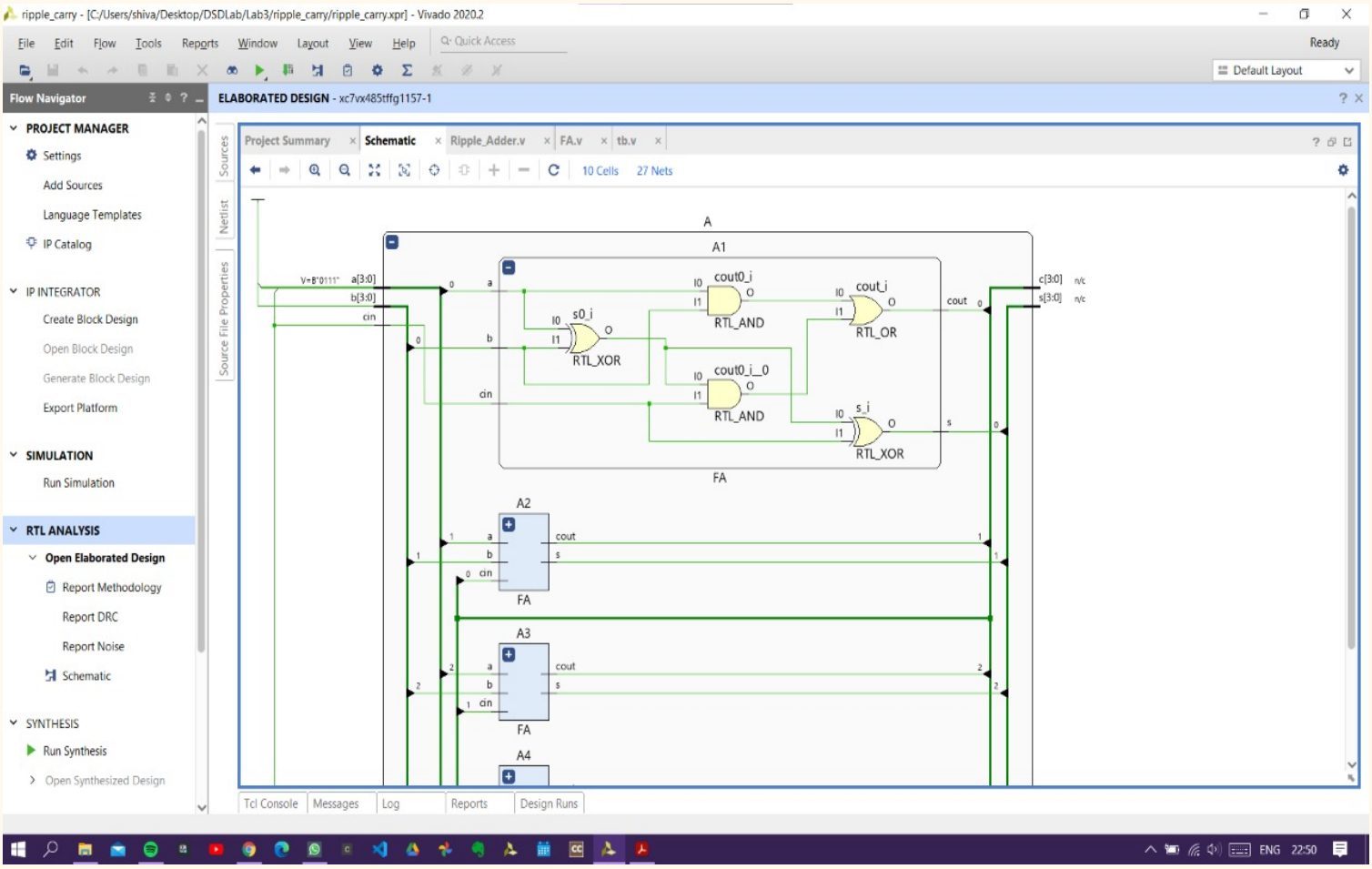
~~Result: Truth table for 4 bit ripple carry adder (using 4 FA) :~~

Conclusion:

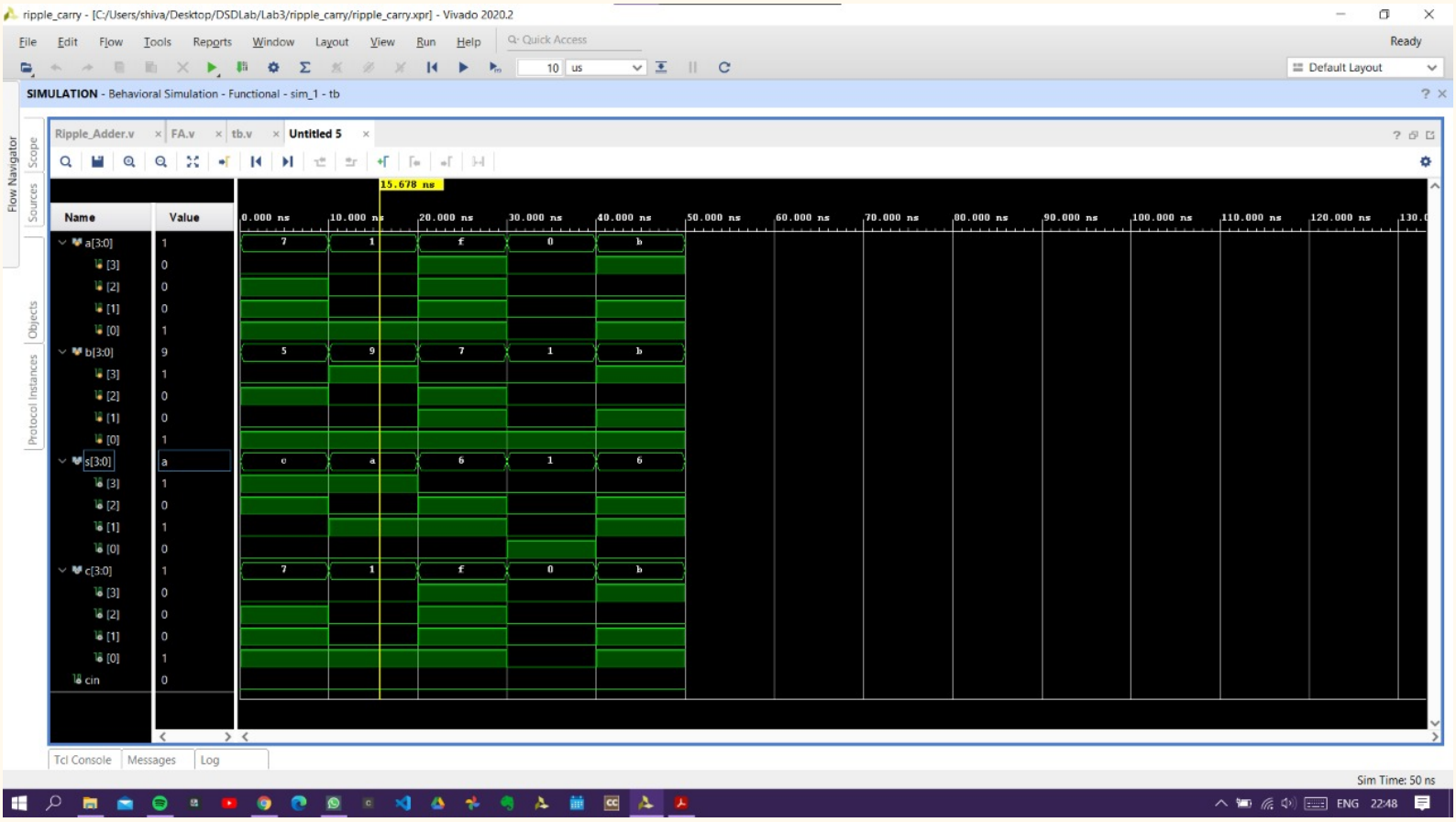
- 4 bit ripple carry adder was designed using 4 Full adders. in Verilog structural modelling style
- Code was thoroughly tested against various testcases and RTL schematic was generated

RTL Schematic




ripple_carry - [C:/Users/shiva/Desktop/DSDLab/Lab3/ripple_carry/ripple_carry.xpr] - Vivado 2020.2
File Edit Flow Tools Reports Window Layout View Help
Quick Access
Ready
Default Layout
Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
Export Platform
SIMULATION
Run Simulation
RTL ANALYSIS
Open Elaborated Design
Report Methodology
Report DRC
Report Noise
Schematic
SYNTHESIS
Run Synthesis
Open Synthesized Design
ELABORATED DESIGN - xc7vx485tffg1157-1
Project Summary
Schematic
Ripple_Adder.v
FA.v
tb.v
10 Cells
27 Nets
A
A1
V=8'0111'
a[3:0]
b[3:0]
cin
s0_i
RTL_XOR
cout0_i
RTL_AND
cout_i
RTL_OR
cout0_i_0
RTL_AND
s_i
RTL_XOR
cout
c[3:0]
s[3:0]
FA
A2
FA
A3
FA
A4
Tcl Console
Messages
Log
Reports
Design Runs
ENG
22:50

Waveform:


ripple_carry - [C:/Users/shiva/Desktop/DSDLab/Lab3/ripple_carry/ripple_carry.xpr] - Vivado 2020.2
File Edit Flow Tools Reports Window Layout View Run Help
Ready
Default Layout
SIMULATION - Behavioral Simulation - Functional - sim_1 - tb
Ripple_Adder.v FA.v tb.v Untitled 5
15.678 ns
Name
Value
a[3:0]
b[3:0]
s[3:0]
c[3:0]
cin
0.000 ns
10.000 ns
20.000 ns
30.000 ns
40.000 ns
50.000 ns
60.000 ns
70.000 ns
80.000 ns
90.000 ns
100.000 ns
110.000 ns
120.000 ns
Tcl Console
Messages
Log
Sim Time: 50 ns

c) 8-bit adder using RCA

Aruhuman Mishra
19911o

FCC - Sec A

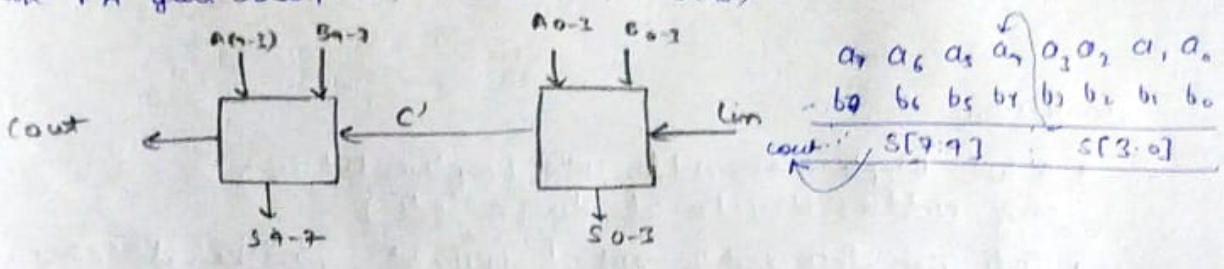
(C)

Aim: 8 bit adder using 4 bit ripple carry adder (Verilog structural modelling style).

Software used: Xilinx Vivado 2020.2

Theory:

8 bit adder is an adder that 2 numbers of 8 bits and generate carry. It uses 2, 4 bit ripple carry adders as subroutine (which in turn use FA full adder as sub-sub routine).



a

Code:

```
module Adder( input [7:0] a, [illegible] b, input cin, output [7:0] S, C );
                                                    output [8:0] Sum
    reg [7:0] a,b; wire [7:0] S,C;
    wire cin = 1'b
    Ripple_Adder R1( a[3:0], b[3:0], cin, S[3:0], C[3:0]);
    R2 ( a[7:4], b[7:4], C[3], S[7:4], C[7:4]);
    assign Sum = {C[7], S[7:0]};
endmodule

module tb;
    reg [7:0] a, b; wire [7:0] S, C; wire [8:0] sum;
    wire cin = 1'b0;
    Ripp Adder A(a, b, cin, S, C, sum);
    initial begin
        a = 8'd17; b = 8'd 12;
        #5 a = 8'd70; b = 8'd 52;
```

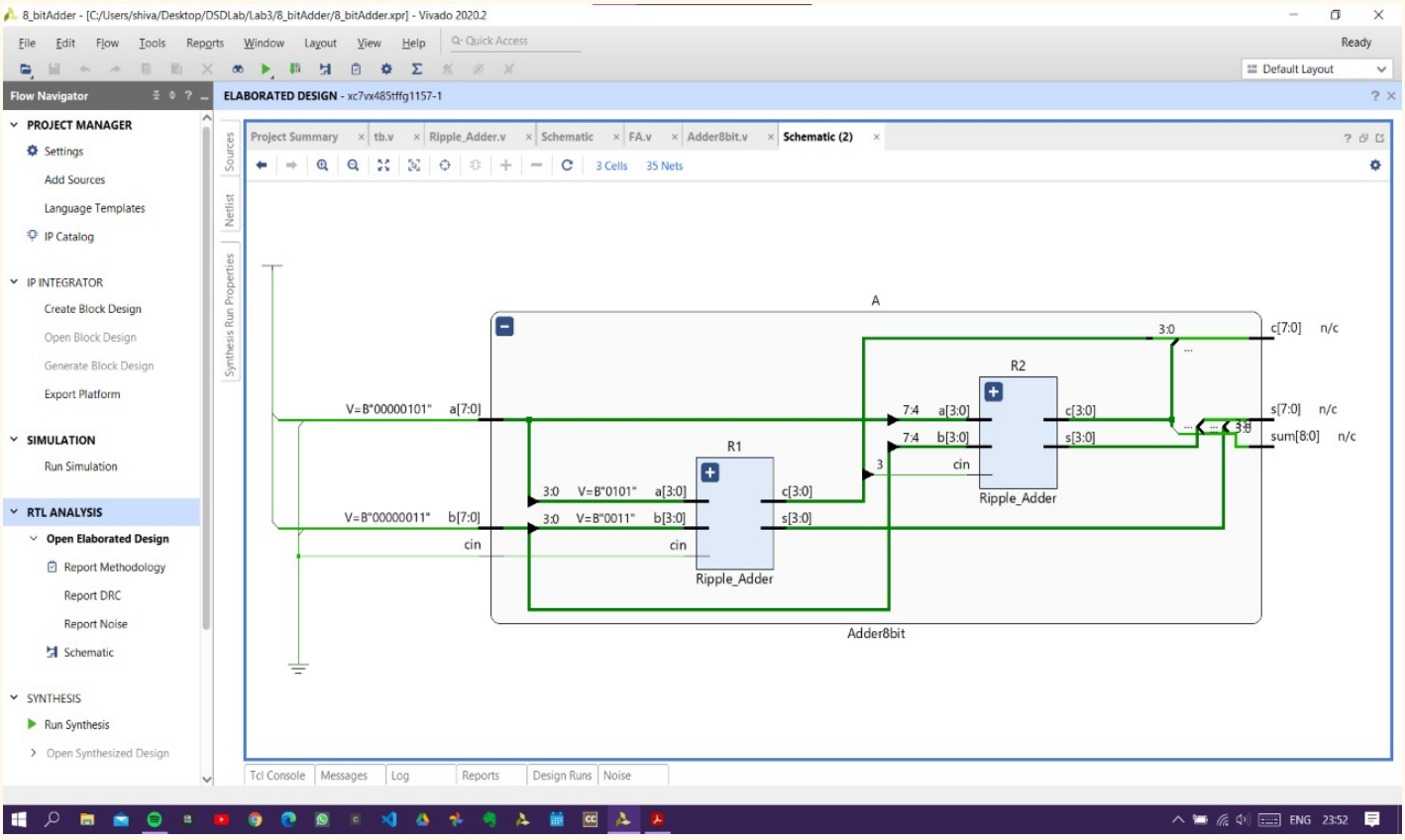
Arshuman Mishra
ECE Sec. A
19111o

```
#5  a= 8'd 120 ;  b= 8'd 58;
#5  a = 8'd 79;  b= 8'd 97;
#5  a= 8'd 61 ;  b= 8'd 31;
#5  a= 8'd 5;    b= 8'd 3;
end
endmodule
```

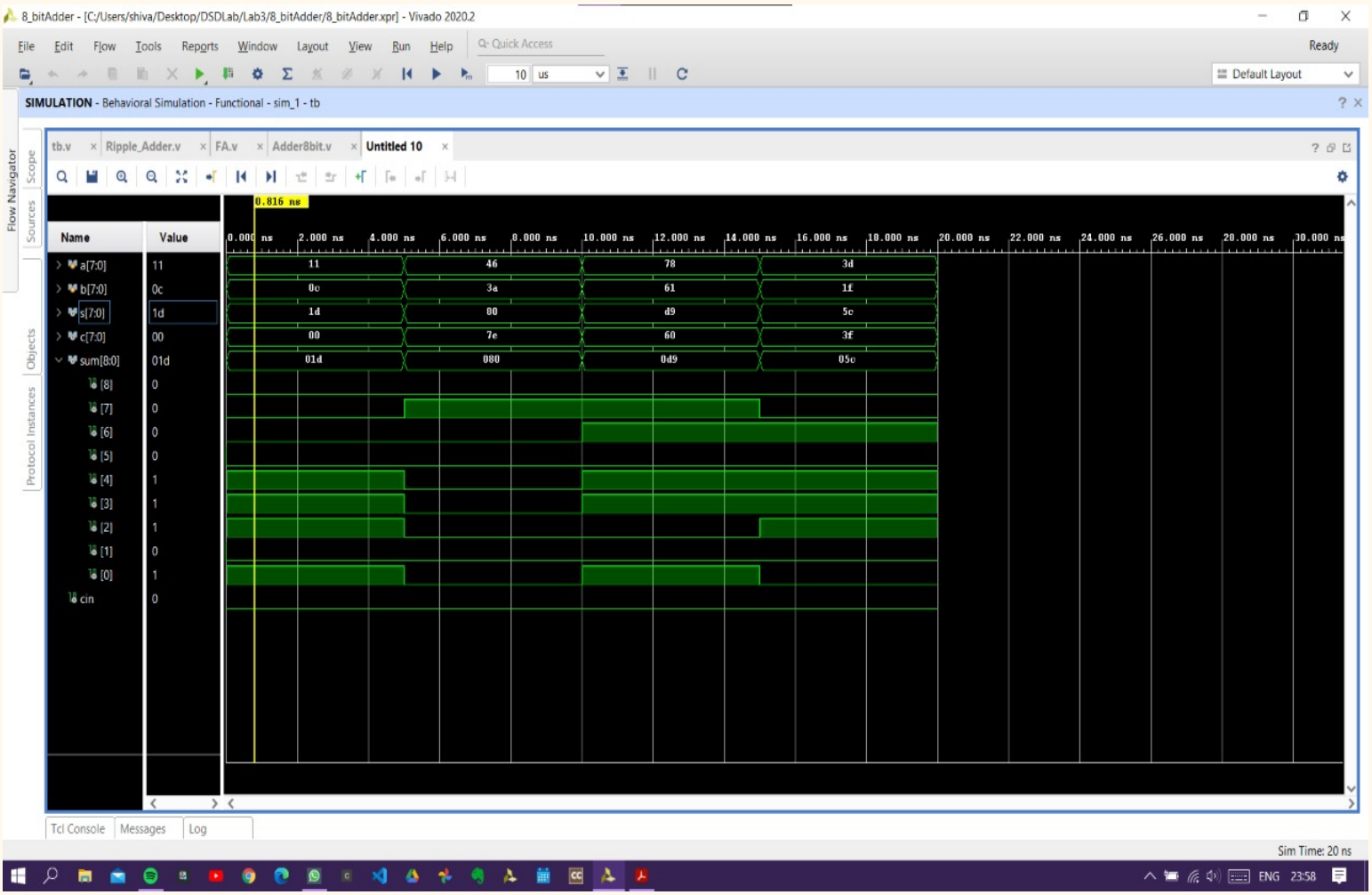
Conclusion:

- 8 bit ~~ripple carry~~ adder was implemented using 2, 4 bit ripple carry adder (structural design style)
- Code was thoroughly ~~tested~~ against various testcases and RTL schematic was generated using Vivado's tool.

RTL Schematic:



Waveform:


8_bitAdder - [C:/Users/shiva/Desktop/DSDLab/Lab3/8_bitAdder/8_bitAdder.xpr] - Vivado 2020.2
File Edit Flow Tools Reports Window Layout View Run Help
Quick Access
Ready
Default Layout
SIMULATION - Behavioral Simulation - Functional - sim_1 - tb
tb.v Ripple_Adder.v FA.v Adder8bit.v Untitled 10
0.816 ns
Name
Value
a[7:0] 11
b[7:0] 0c
s[7:0] 1d
c[7:0] 00
sum[8:0] 01d
[8] 0
[7] 0
[6] 0
[5] 0
[4] 1
[3] 1
[2] 1
[1] 0
[0] 1
cin 0
11 46 78 3d
0c 3a 61 1f
1d 80 d9 5c
00 7e 60 3f
01d 080 0d9 05c
Tcl Console Messages Log
Sim Time: 20 ns
ENG 23:58