Design and Implementation of MUX, Full Subtractor, and Synchronous Counter using Spartan 3e and Xilinx

by

Liza Sarkar

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

National Institute of Technology, Jamshedpur-831014

(November 2023)

TABLE OF CONTENTS

		Page Nos.
1.	DEMUX	3-10
2.	Full Subtractor	4 -17
3.	Synchronous Counter	18-22
4.	Conclusion	23

DEMUX

Aim :

To design 1:2 and 1:4 Demux using verilog code.

Apparatus Required :

Xilinx Vivado and Spartan

Theory :

De-Multiplexer is a combinational circuit that performs the reverse operation of Multiplexer. It has single input, ‘n’ selection lines and a maximum of 2^n outputs. The input will be connected to one of these outputs based on the values of selection lines.

Since there are ‘n’ selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination can select only one output. De-Multiplexer is also called Demux .

I. **1 : 2 Demux :** 1:2 Demux has one select line and 2 output lines . The signal on the select line helps to switch the input to one of the two outputs. The figure below shows the block diagram of a 1-to-2 demultiplexer with additional enable input.

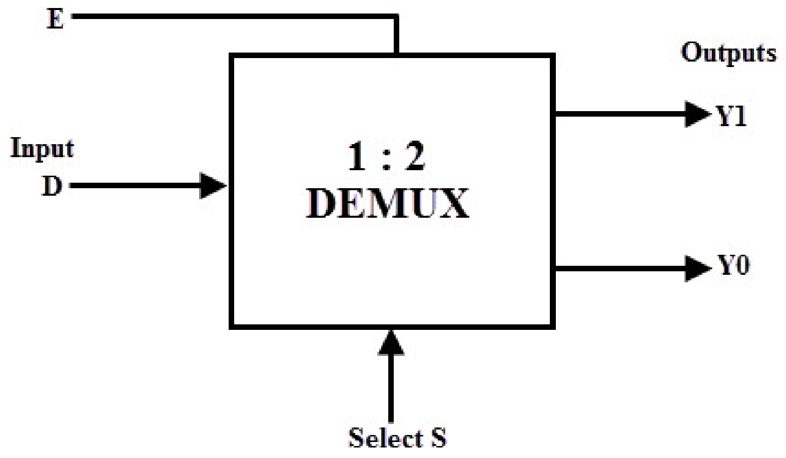
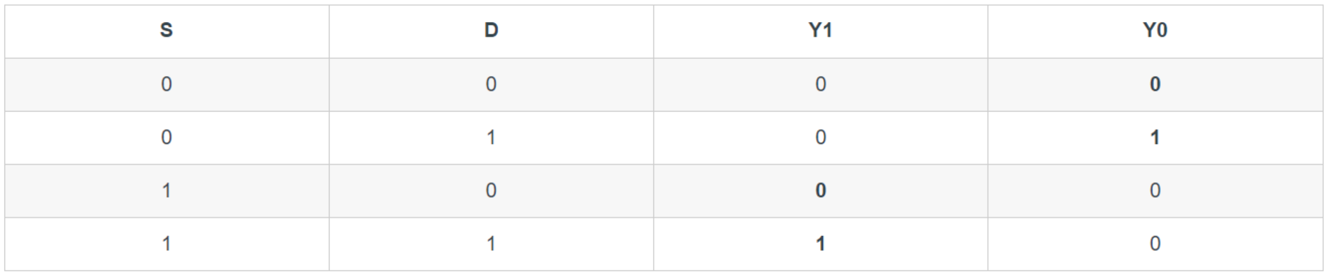


Fig 1 : Block Diagram of 1 : 2 Demux



S	D	Y1	Y0
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

Fig 2 : Truth Table of 1 : 2 Demux

II. **1 : 4 Demux :** A 1-to-4 demultiplexer has a single input (D), two selection lines (S1 and S0) and four outputs (Y0 to Y3). The input data goes to any one of the four outputs at a given time for a particular combination of select lines.This demultiplexer is also called a 2-to-4 Demultiplexer, which means that it has two select lines and 4 output lines. The block diagram of a 1:4 DEMUX is shown below.

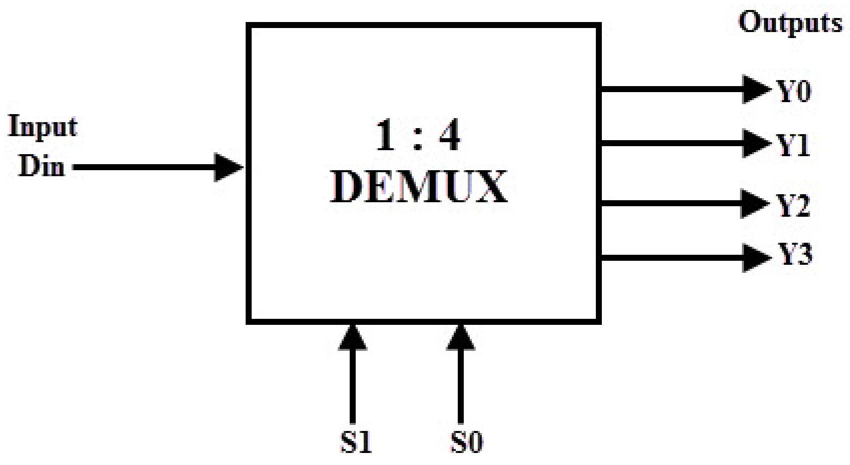
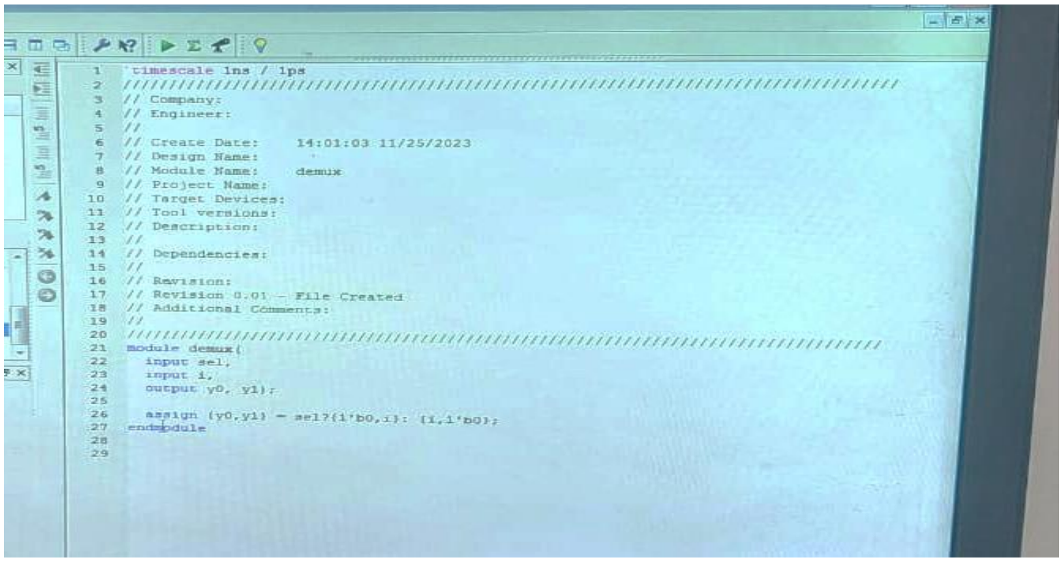


Fig 3 : Block Diagram Of 1 : 4 Demux

S1	S0	D	Y3	Y2	Y1	Y0
0	0	0	0	0	0	**0**
0	0	1	0	0	0	**1**
0	1	0	0	0	**0**	0
0	1	1	0	0	**1**	0
1	0	0	0	**0**	0	0
1	0	1	0	**1**	0	0
1	1	0	**0**	0	0	0
1	1	1	**1**	0	0	0

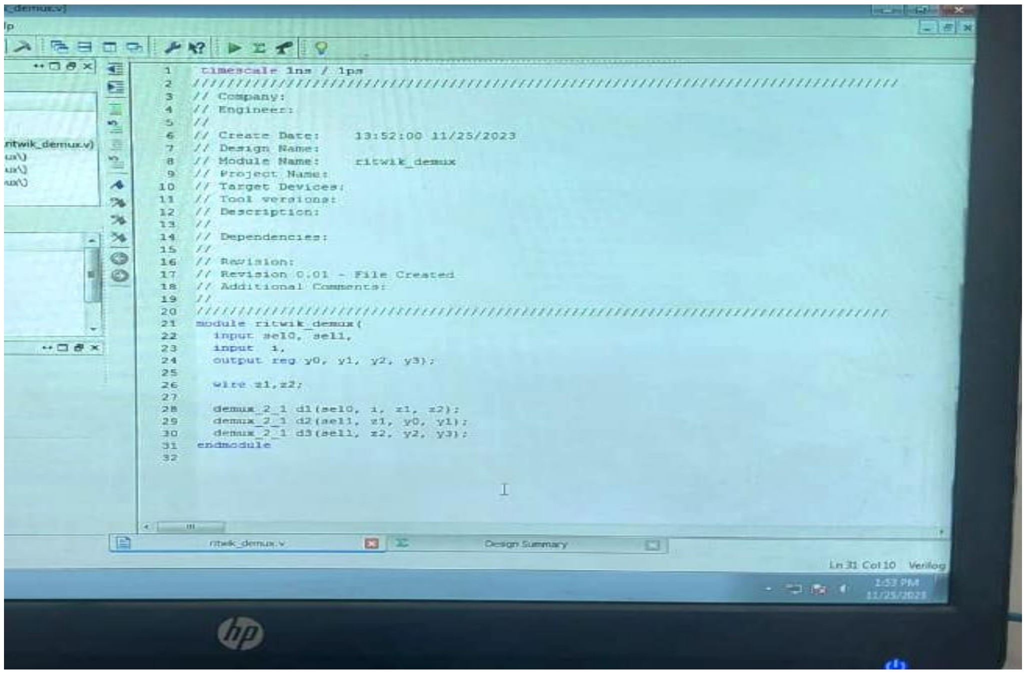
Fig 4: Truth Table Of 1:4 Demux

1:2 Demux Verilog Code :



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    14:01:03 11/25/2023
// Design Name:
// Module Name:    demux
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module demux(
  input sel,
  input i,
  output y0, y1);

  assign {y0,y1} = sel?{1'b0,i}: {i,1'b0};
endmodule
```

1 : 2 Demux Testbench :



```
timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    13:52:00 11/25/2023
// Design Name:
// Module Name:    ritwik_demux
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module ritwik_demux(
  input sel0, sel1,
  input  i,
  output reg y0, y1, y2, y3);

  wire z1,z2;

  demux_2_1 d1(sel0, i, z1, z2);
  demux_2_1 d2(sel1, z1, y0, y1);
  demux_2_1 d3(sel1, z2, y2, y3);
endmodule
```

1 : 4 Verilog Code :

```
module demux_1_4(
  input [1:0] sel,
  input  i,
  output reg y0,y1,y2,y3);

  always @(*) begin
    case(sel)
      2'h0: {y0,y1,y2,y3} = {i,3'b0};
      2'h1: {y0,y1,y2,y3} = {1'b0,i,2'b0};
      2'h2: {y0,y1,y2,y3} = {2'b0,i,1'b0};
      2'h3: {y0,y1,y2,y3} = {3'b0,i};
      default: $display("Invalid sel input");
    endcase
  end
endmodule
```

1:4 Testbench :

```
module tb;
  reg [1:0] sel;
  reg i;
  wire y0,y1,y2,y3;

  demux_1_4 demux(sel, i, y0, y1, y2, y3);

  initial begin
    $monitor("sel = %b, i = %b -> y0 = %0b, y1 = %0b ,y2 = %0b, y3 = %0b", sel,i, y0,y1,y2,y3);
    sel=2'b00; i=0; #1;
    sel=2'b00; i=1; #1;
    sel=2'b01; i=0; #1;
    sel=2'b01; i=1; #1;
    sel=2'b10; i=0; #1;
    sel=2'b10; i=1; #1;
    sel=2'b11; i=0; #1;
    sel=2'b11; i=1; #1;
  end
endmodule
```

Output :

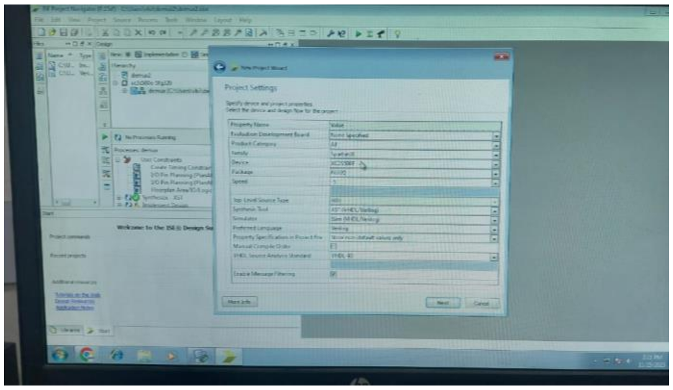


Fig 5 : Creating File

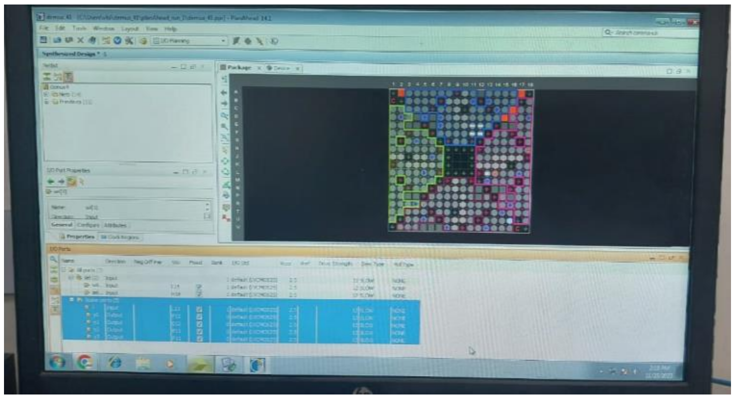


Fig 6 : FPGA Architecture of 1:2 Demux

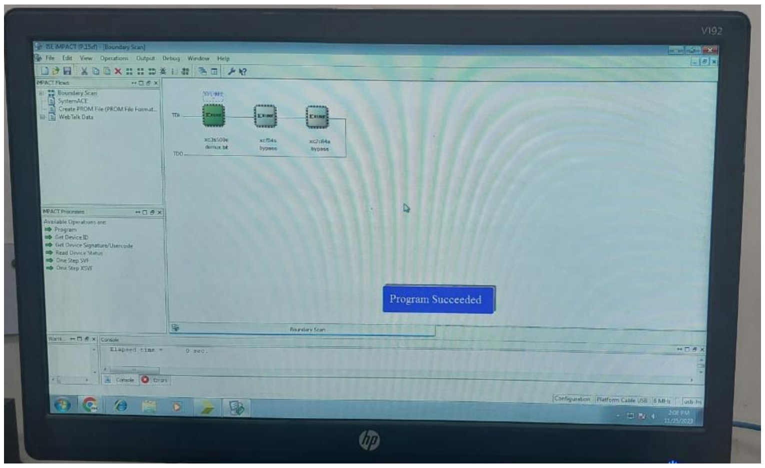
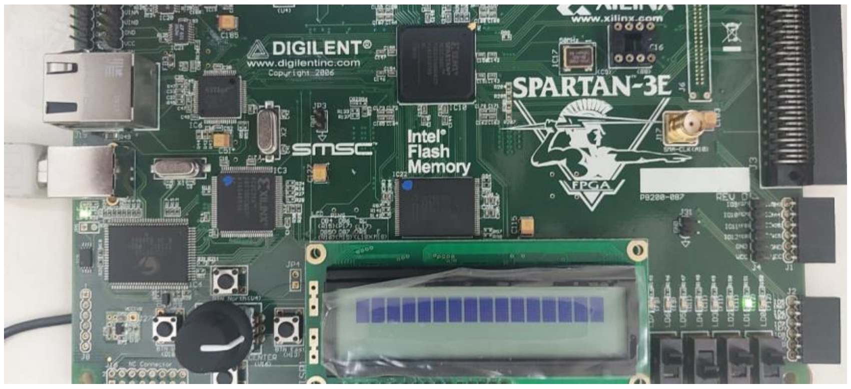


Fig 7 : Uploading of Program



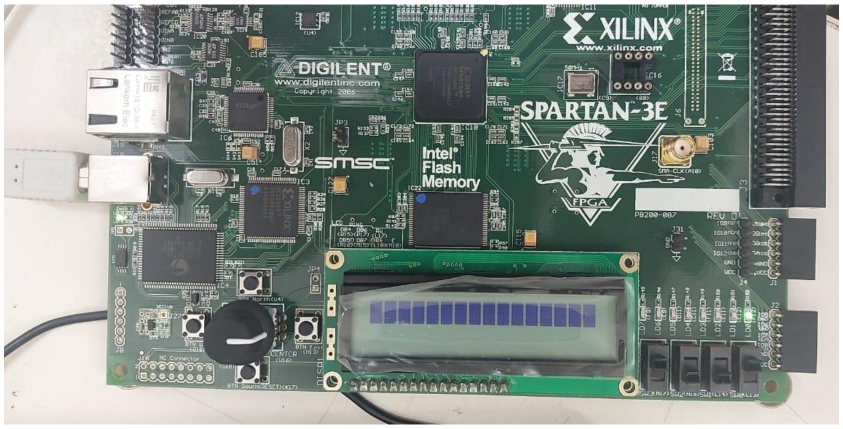
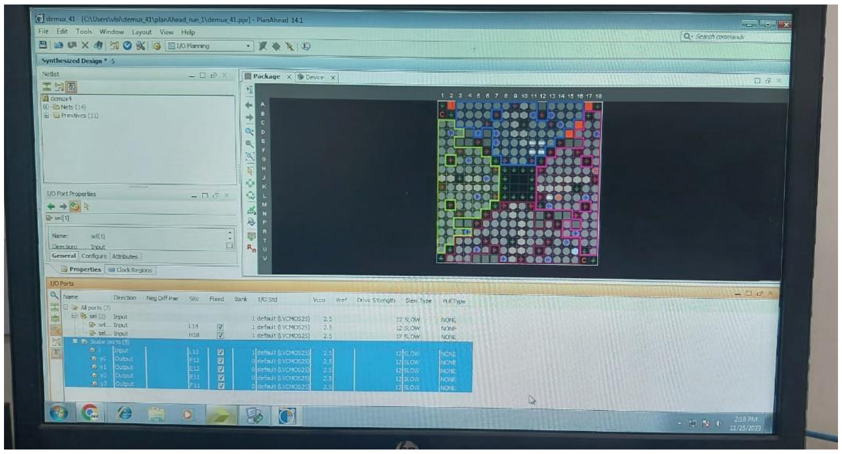


Fig 8 : Successful Implementation of 1: 2 Demux on SPARTAN-3E kit



I/O Port Properties

sel[1]

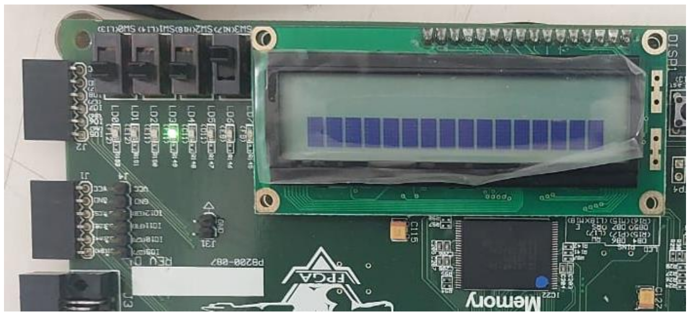
Name: sel[1]

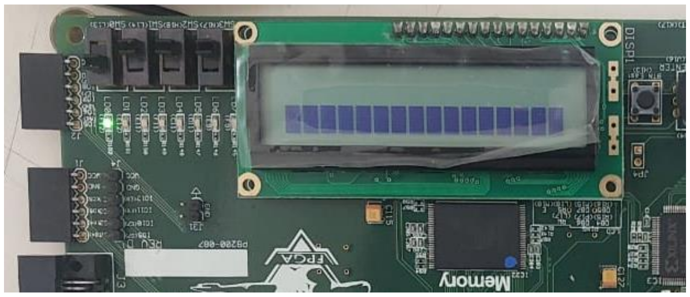
Direction: Input

I/O Ports

Name	Direction	Neg Diff Pair	Site	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type
All ports (7)											
sel (2)	Input				1	default (LVCMOS25)	2.5		12	SLOW	NONE
sel...	Input		L14	✓	1	default (LVCMOS25)	2.5		12	SLOW	NONE
sel...	Input		H18	✓	1	default (LVCMOS25)	2.5		12	SLOW	NONE
Scalar ports (5)											
i	Input		L13	✓	1	default (LVCMOS25)	2.5		12	SLOW	NONE
y0	Output		F12	✓	0	default (LVCMOS25)	2.5		12	SLOW	NONE
y1	Output		E12	✓	0	default (LVCMOS25)	2.5		12	SLOW	NONE
y2	Output		E11	✓	0	default (LVCMOS25)	2.5		12	SLOW	NONE
y3	Output		F11	✓	0	default (LVCMOS25)	2.5		12	SLOW	NONE

Fig 11 : Input and output pin selection of 1:4
Demux





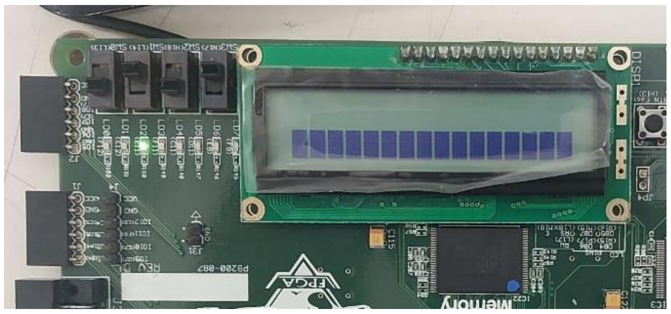
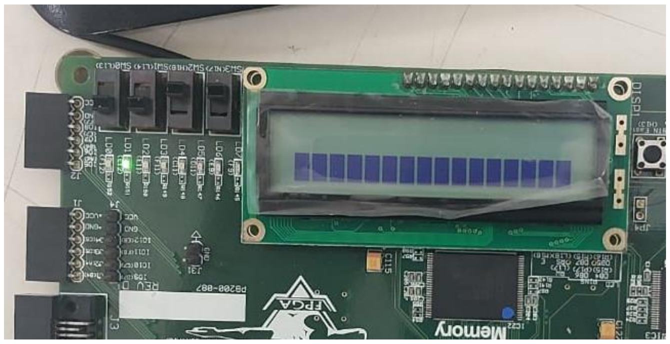



Fig 12 : Successful implementation of 1: 4 Demux

Result :

1:2 and 1:4 Demux was obtained as well as its implementation on Spartan was done successfully.

Full Subtractor

Aim:

To design a Full Subtractor using verilog code.

Apparatus:

Xilinx Vivado and Spartan

Theory:

A full subtractor is a combinational circuit that performs binary subtraction of three bits: minuend, subtrahend, and borrow. It has three inputs (A, B, and Borrow In) and two outputs (Difference and Borrow Out). The logic is more complex than a half subtractor, as it considers borrowing from the previous stage.

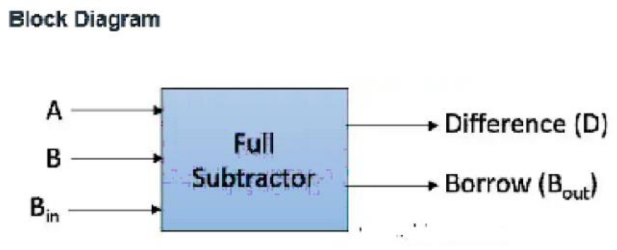
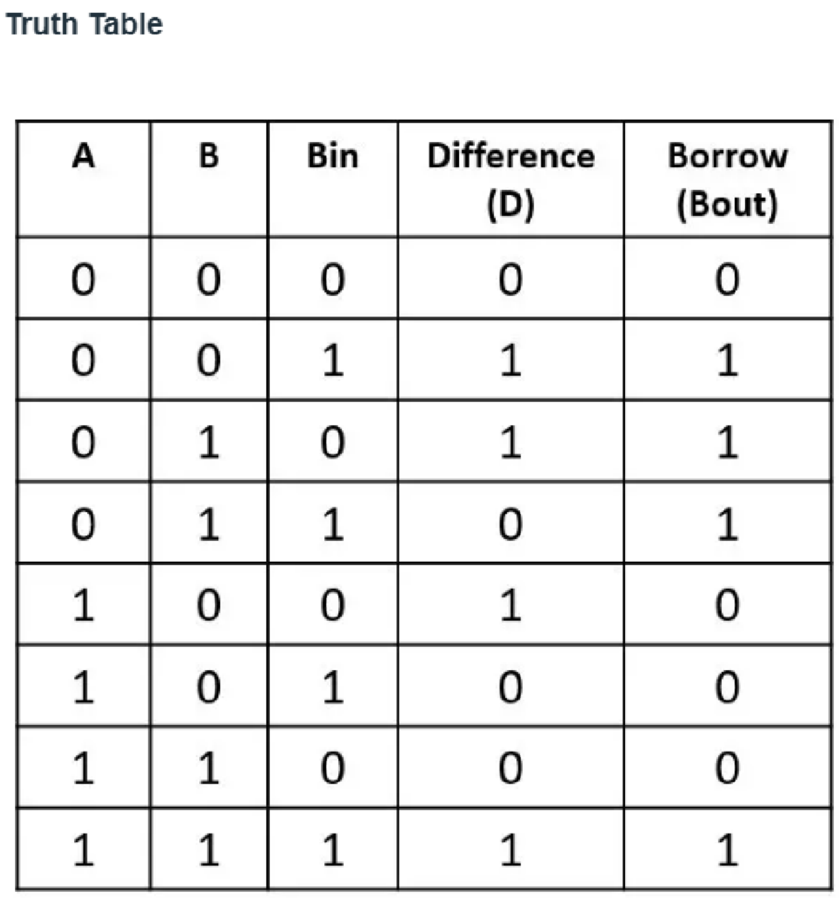
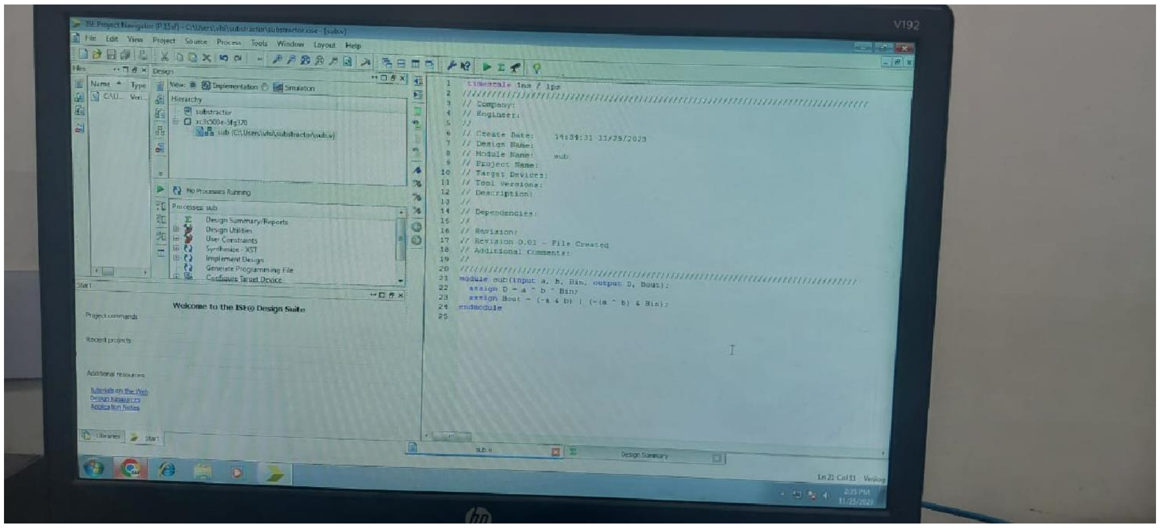


Fig 1: *Block Diagram of Full Subtractor*


Truth Table

A	B	Bin	Difference (D)	Borrow (Bout)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Full Subtractor Verilog Code :

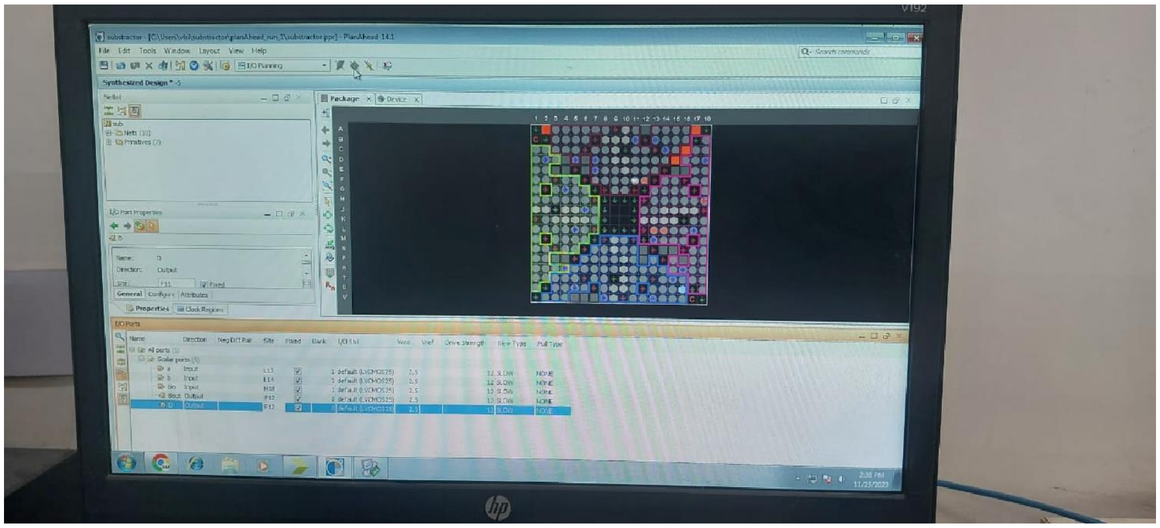


```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    14:34:31 11/25/2023
// Design Name:
// Module Name:    sub
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module sub(input a, b, Bin, output D, Bout);
  assign D = a ^ b ^ Bin;
  assign Bout = (~a & b) | (~(a ^ b) & Bin);
endmodule
```

Corresponding Test Bench Code:

```verilog
module tb_top;
  reg a, b, Bin;
  wire D, Bout;

  full_subtractor fs(a, b, Bin, D, Bout);

initial begin
  $monitor("At time %0t: a=%b b=%b, Bin=%b, difference=%b,
borrow=%b",$time, a,b,Bin,D,Bout);
    a = 0; b = 0; Bin = 0; #1;
    a = 0; b = 0; Bin = 1; #1;
    a = 0; b = 1; Bin = 0; #1;
    a = 0; b = 1; Bin = 1; #1;
    a = 1; b = 0; Bin = 0; #1;
    a = 1; b = 0; Bin = 1; #1;
    a = 1; b = 1; Bin = 0; #1;
    a = 1; b = 1; Bin = 1;
  end
endmodule
```

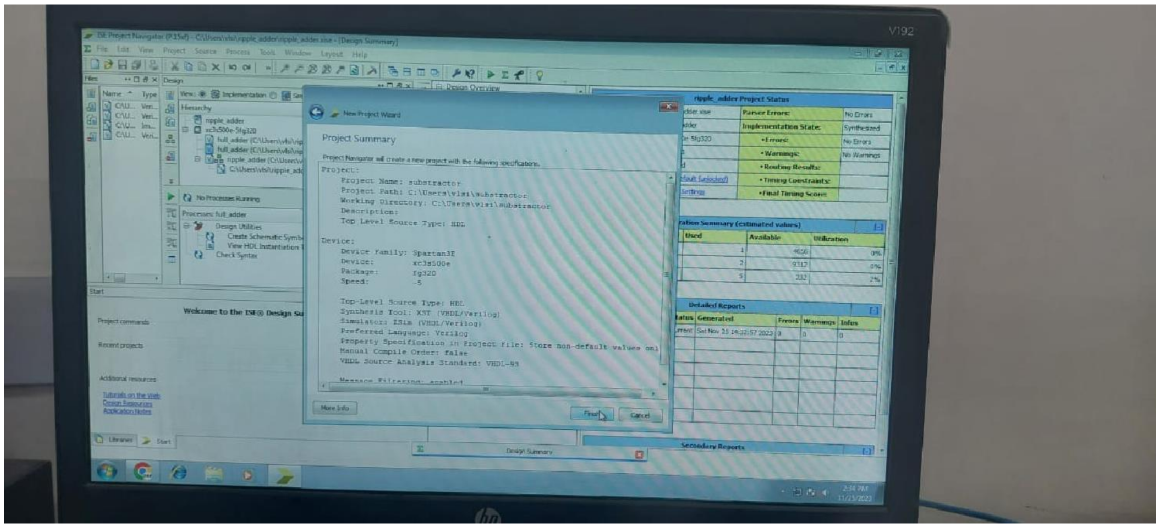
Outputs :



I/O Ports

Name	Direction	Neg Diff Pair	Site	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type
All ports (5)											
Scalar ports (5)											
a	Input		L13	✓	1	default (LVCMOS25)	2.5		12	SLOW	NONE
b	Input		L14	✓	1	default (LVCMOS25)	2.5		12	SLOW	NONE
bin	Input		H18	✓	1	default (LVCMOS25)	2.5		12	SLOW	NONE
Bout	Output		F12	✓	0	default (LVCMOS25)	2.5		12	SLOW	NONE
D	Output		E12	✓	0	default (LVCMOS25)	2.5		12	SLOW	NONE

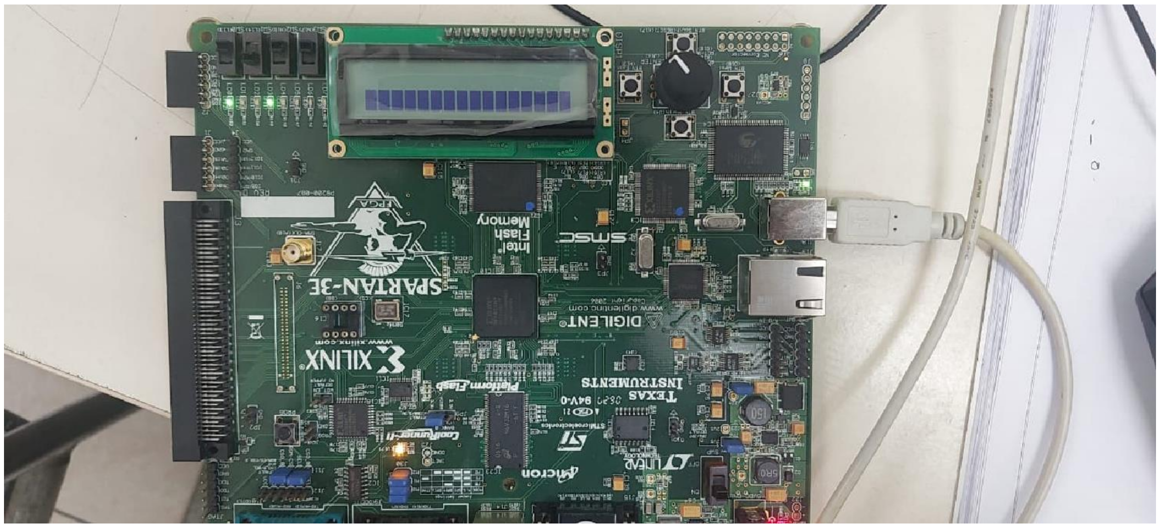
Fig: *FPGA Architecture of Full Subtractor*

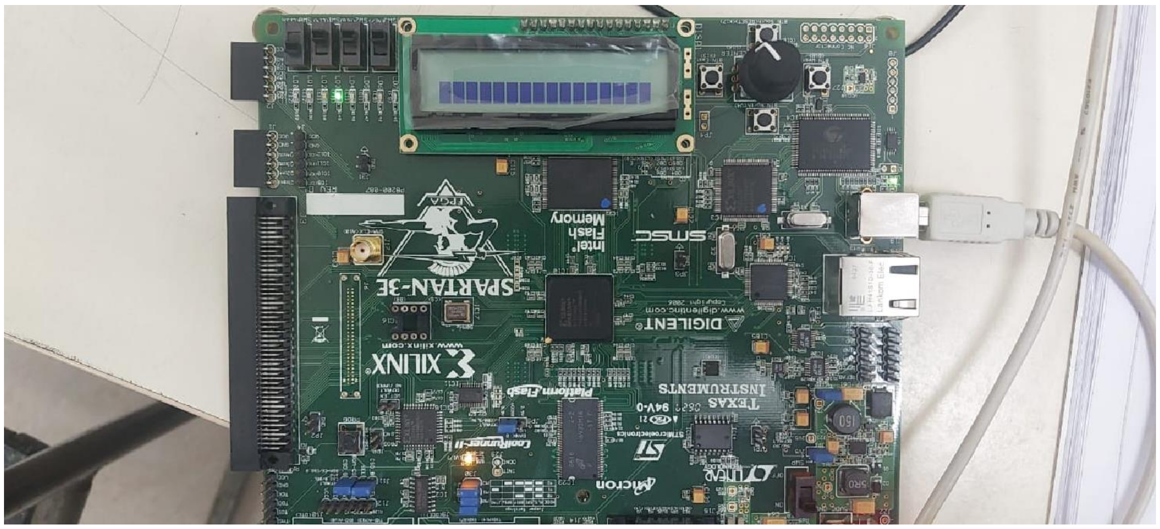


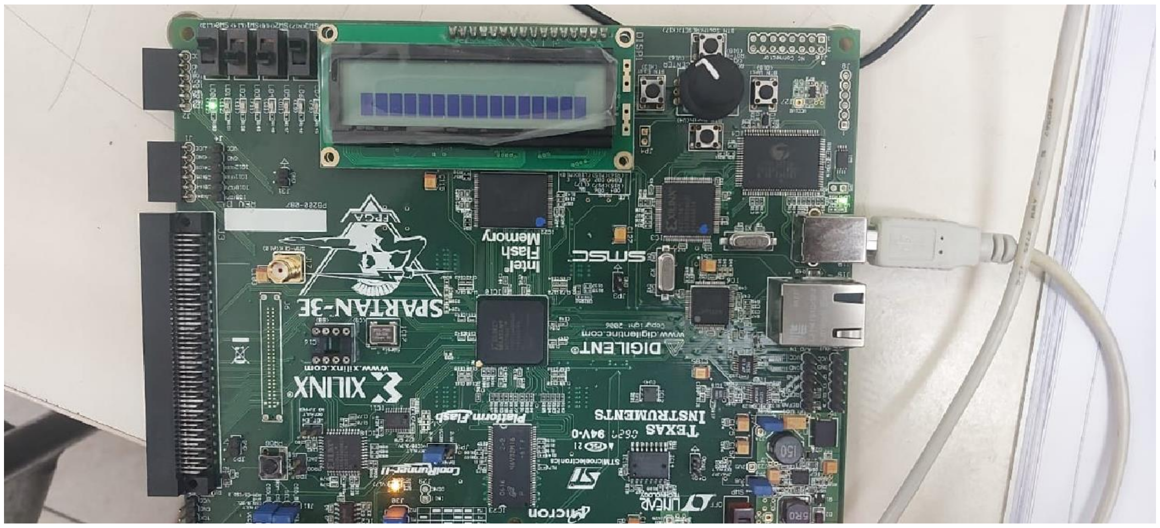
Project Summary

Project Navigator will create a new project with the following specifications.

```
Project:
    Project Name: substractor
    Project Path: C:\Users\vlsi\substractor
    Working Directory: C:\Users\vlsi\substractor
    Description:
    Top Level Source Type: HDL

Device:
    Device Family: Spartan3E
    Device:        xc3s500e
    Package:       fg320
    Speed:         -5

    Top-Level Source Type: HDL
    Synthesis Tool: XST (VHDL/Verilog)
    Simulator: ISim (VHDL/Verilog)
    Preferred Language: Verilog
    Property Specification in Project File: Store non-default values onl
    Manual Compile Order: false
    VHDL Source Analysis Standard: VHDL-93

    Message Filtering: enabled
```







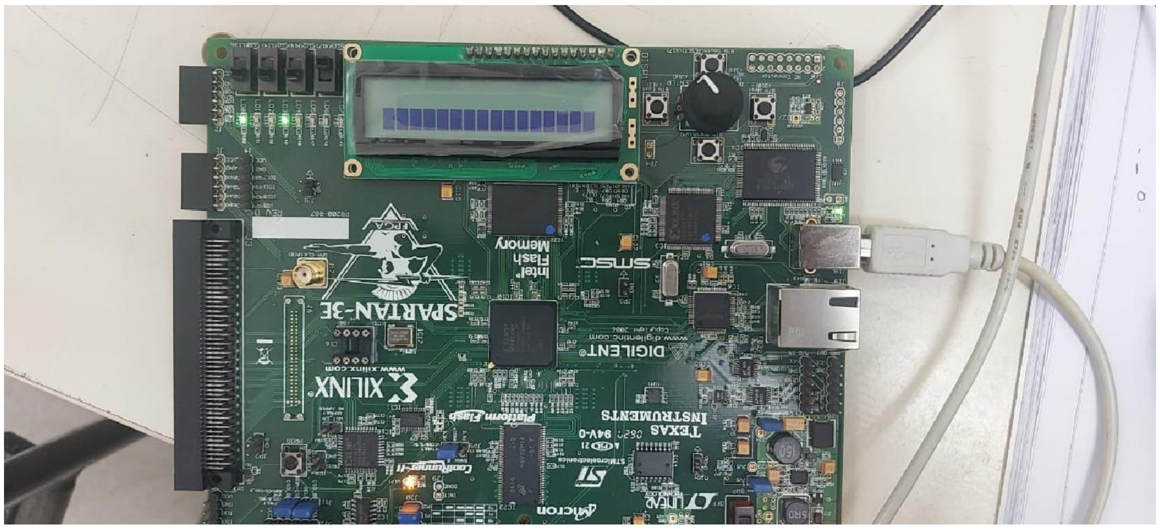


Fig: *Different outputs for various bits of Full Subtractor in Spartan*

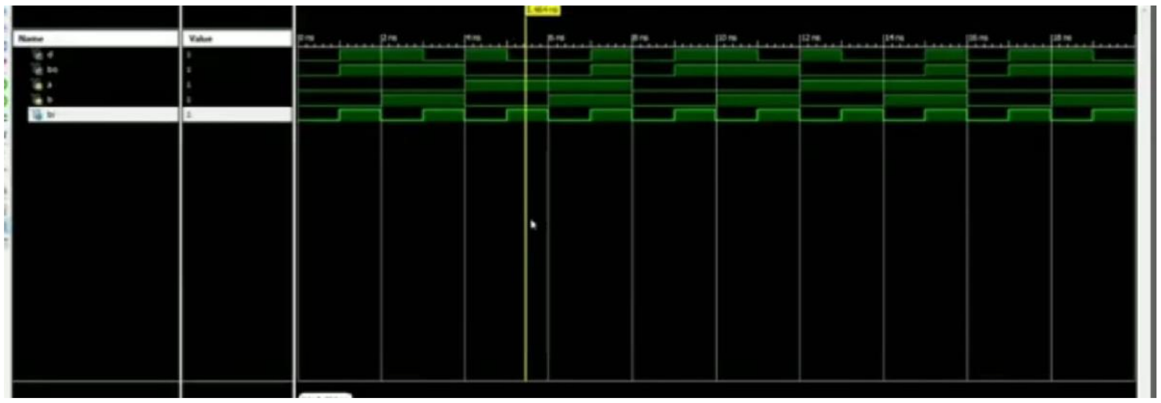


Fig: *Waveform of Full Subtractor*

Result:

Schematic & Timing Diagram of Full Subtractor was obtained as well as its implementation on Spartan was done successfully.

Synchronous Counter

Aim:

To design a Synchronous counter using verilog code.

Apparatus:

Xilinx Vivado and Spartan

Theory:

A synchronous counter is a type of digital circuit composed of flip-flops where all the flip-flops share a common clock signal, ensuring simultaneous state transitions. In the case of a binary synchronous counter, each flip-flop represents a binary bit, and the counter progresses through binary counts with each clock pulse.

Synchronous counters offer predictability and synchronization, making them preferable in various applications such as frequency dividers, time delay circuits, and digital clocks. They can be configured as up or down counters, counting in ascending or descending order, and multiple synchronous counters can be cascaded to extend the counting range.

The design considerations include modulus N counting, clear and preset inputs, and the choice between synchronous and asynchronous counter types based on specific application requirements. Overall, synchronous counters play a fundamental role in digital systems where controlled and coordinated sequential counting is essential.

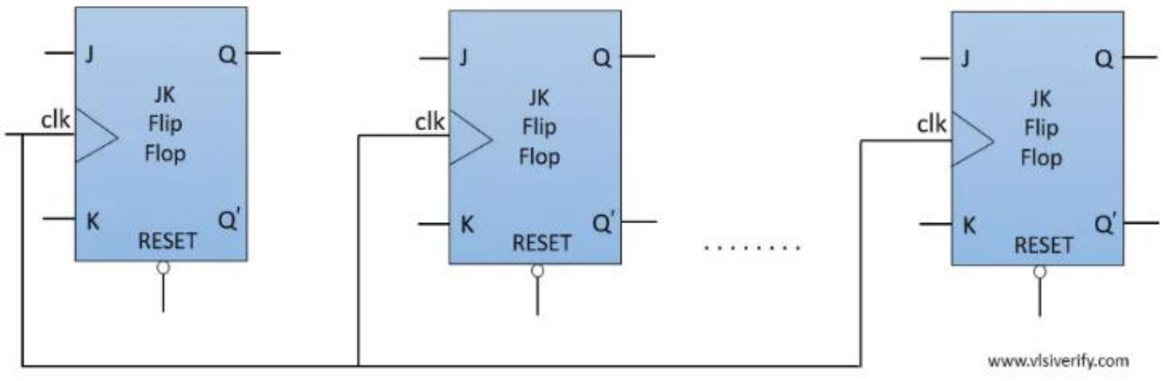


Fig: *Block Diagram of Synchronous Counter*

Verilog for 4 bit Up-Down Synchronous Counter:

```
module synchronous_counter #(parameter SIZE=4)(
  input clk, rst_n,
  input up,
  output reg [3:0] cnt
);

  always@(posedge clk) begin
    if(!rst_n) begin
      cnt <= 4'h0;
    end
    else begin
      if(up) cnt <= cnt + 1'b1;
      else cnt <= cnt - 1'b1;
    end
  end
endmodule
```

Corresponding Test Bench Code:

```
module tb;
  reg clk, rst_n;
  reg up;
  wire [3:0] cnt;
  synchronous_counter(clk, rst_n, up, cnt);
  initial begin
    clk = 0; rst_n = 0;
    up = 1;
    #4; rst_n = 1;
    #80;
    rst_n = 0;
    #4; rst_n = 1; up = 0;
    #50;
    $finish;
  end
  always #2 clk = ~clk;

  initial begin
    $dumpfile("dump.vcd"); $dumpvars;
  end
endmodule
```

Outputs :

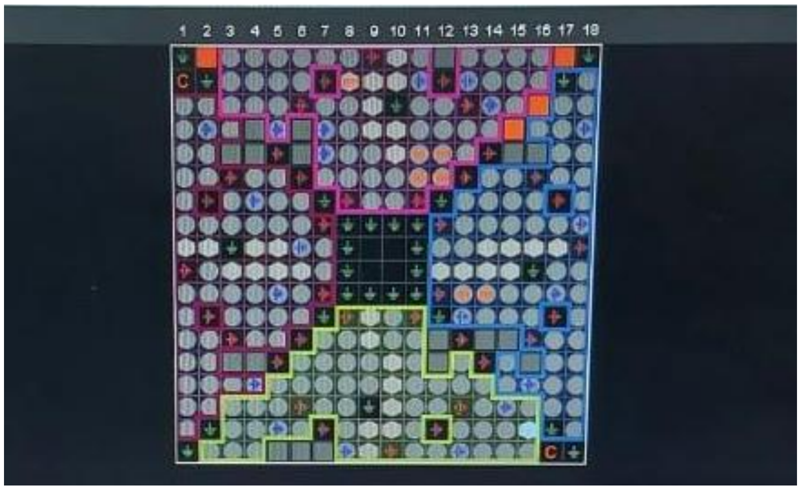
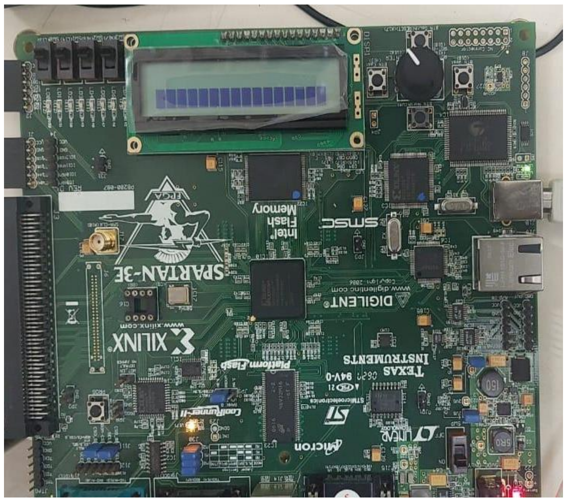
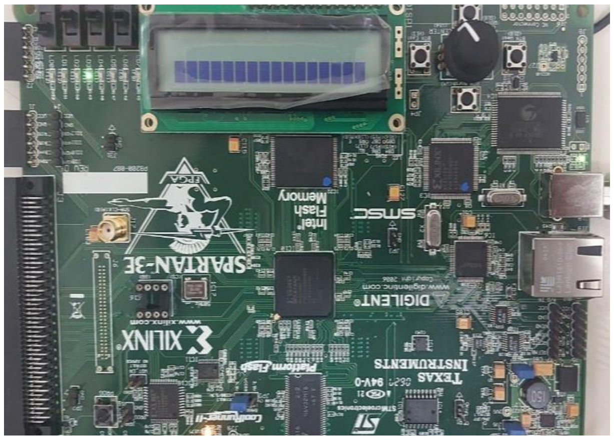


Fig: *FPGA Architecture of Synchronous Counter*





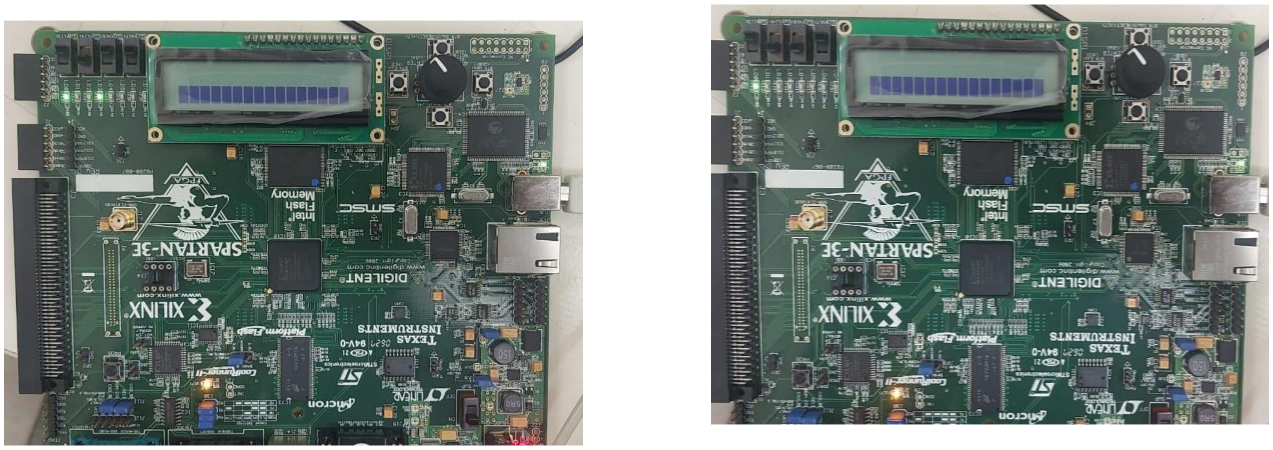


Fig: *Different outputs for various bits of Synchronous Counter in Spartan*

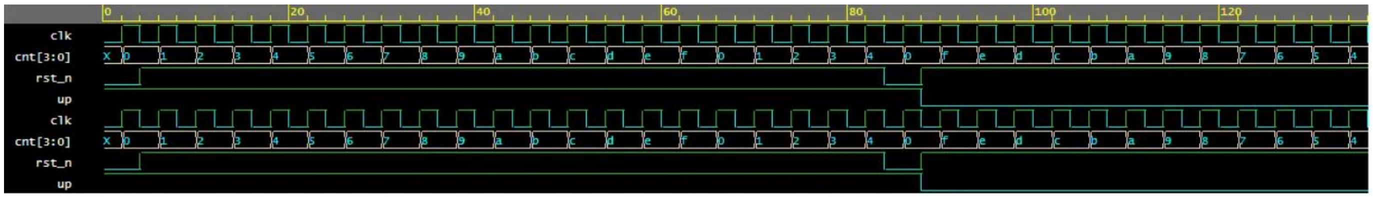


Fig: *Waveform of Synchronous Counter*

Result:

Schematic & Timing Diagram of Synchronous counter was obtained as well as its implementation on Spartan was done successfully.

Conclusion

The experiments conducted on Demux, full subtractor, and synchronous counter using xyline Spartan 3e FPGA board were successful. The results obtained from the experiments were analyzed and compared with the theoretical values. It was observed that the experimental values were in good agreement with the theoretical values. The experiments demonstrated the practical applications of these concepts in real-world scenarios. The knowledge gained from these experiments can be applied in various fields such as automation, control systems, and signal processing