LAB

REPORT OF

Modeling and Testing of Digital Systems (VHDL)

(ECPE 22)

A report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
in
ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted By

Rakhee Prajapat (21233)

Submitted To

Mr. Satish Kumar (FACULTY, SoE)

SCHOOL OF ELECTRONICS

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY UNA
HIMACHAL PRADESH

November 2023

INDEX

Experiment – 1

Aim: Write a program in VHDL for the implementation of basic logic gates using behavioural modelling.

Software Used: Xilinx Vivado 2020.1

Truth Table:

1. **AND Gate**

Input		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

3. **OR Gate**

Input		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

2. **NOR Gate**

Input		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

4. **NAND Gate**

Input	Input	Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Program:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity exp_1 is
Port ( A, B : in STD_LOGIC;
O1, O2, O3, O4, O5, O6, O7 : out STD_LOGIC);
end exp_1;

architecture Behavioral of exp_1 is
begin
process(A,B)
```

```
begin
if(A='1' and B='1') then
O1<='1';
else
O1 <='0';
end if;
end process;

process(A , B)
if(A='0' and B='0') then
O2<='0';
else
O2 <='1';
end if;
end process;
process(A , B)
if( A='0' and B='0') then
O3<= '0';
elsif(A= '1' and B= '1') then
O3<='0';
else
O3<='1';
end if;
end process;
process(A , B)
if(A='0' and B='0') then
O4<='1';
else
O4 <='0';
end if;
end process;
process(A , B)
if(A='1' and B='1') then
O5<='0';
else
O5 <='1';
```

```
end if;
end process;
process(A , B)
if( A='0' and B='0') then
O6<= '1';
elsif(A= '1' and B= '1') then
O6<='1';
else
O6<='0';
end if;
end process;
process(A , B)
if(A='1' or B='1') then
O7<='0';
else
O7 <='1';
end if;
end process;
end Behavioral;
```

RTL Schematics:

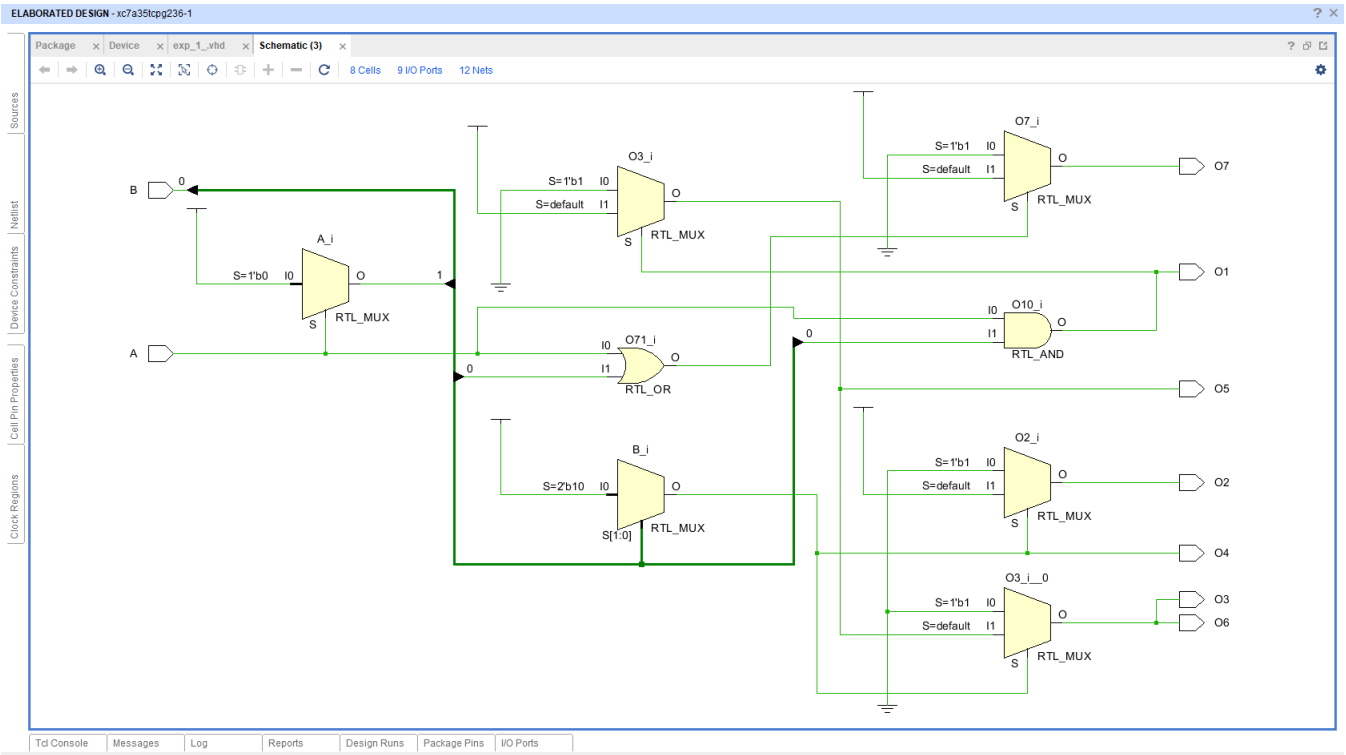


Figure 1 – RTL Schematic Diagram

Waveform:

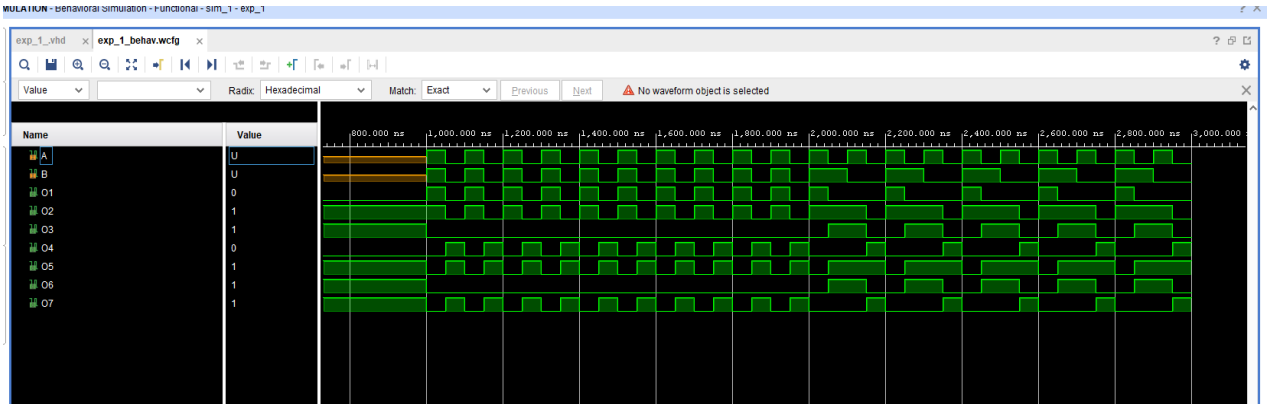


Figure 2 – Waveform Screenshot

Learning Outcome: Hence, all the logic gates are implemented in VHDL and their comprehensive overview of the functioning of all the basic gates and got to know their working style in behavioral modeling in vhdl.

Experiment – 2

Aim: Write a program in VHDL for the implementation of half-adder and full-adder.

Software Used: Xilinx Vivado 2020.1

Truth Table:

Half-Adder:

Truth Table			
Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Full-Adder

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Program:

```
Entity half_adder is
        Port( a, b : in STD_LOGIC;
              s, c : out STD_LOGIC);
end half_adder;
architecture behavior of half_adder is
  begin
    process (a, b)
    begin
      if a = '1' then
        s <= not b;
        c <= b;
      else
        sum <= b;
      carry_out <= '0';
      end if;
    end process;
end behavior;
```

RTL Schematics:

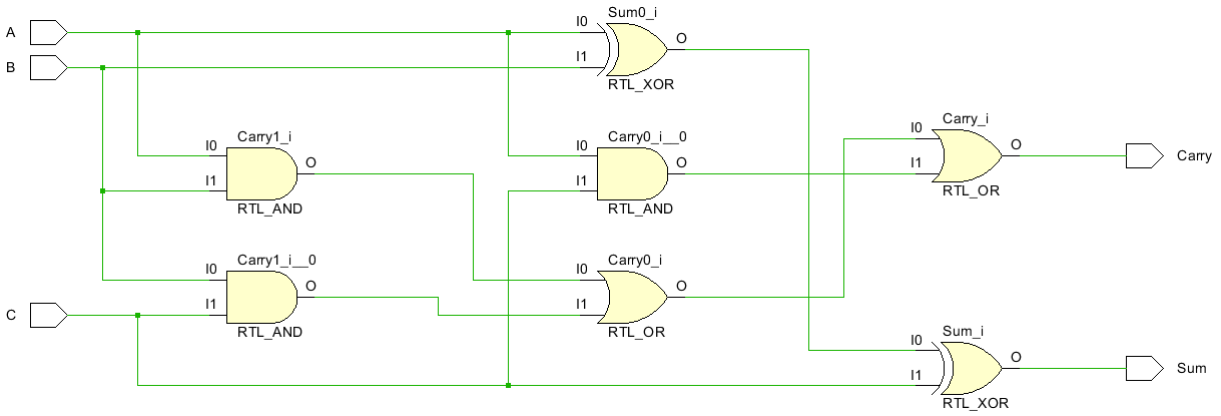


Figure 1 – Schematics Screenshot

Waveform:

Figure 2 – Waveform Screenshot

Program:

```
begin
process(a,b,c)
begin
if(a = '0' and b = '0' and c = '0')
then
sum <= '0';
carry <= '0';
elsif((a = '0'and b = '0' and c = '1') or (a = '0' and b = '1' and c = '0') or (a = '1' and b = '0' and c = '0'))
then
sum <= '1';
carry <= '0';
```

```
elsif((a = '1'and b = '1' and c = '1'))
then
sum <= '1';
carry <= '1';
else
sum <= '0';


carry <= '1';
end if;
end process;
```

RTL Schematics:

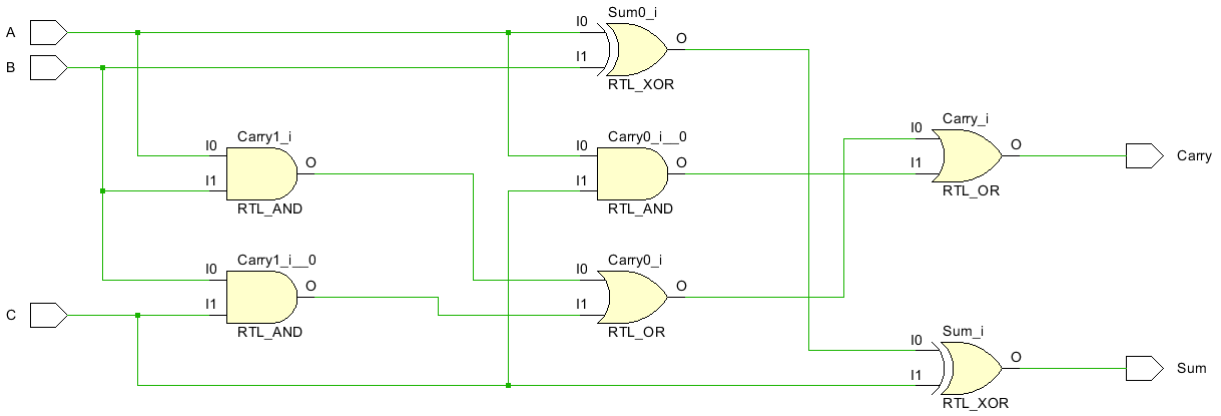


Figure 1 – Schematics Screenshot

Waveform:

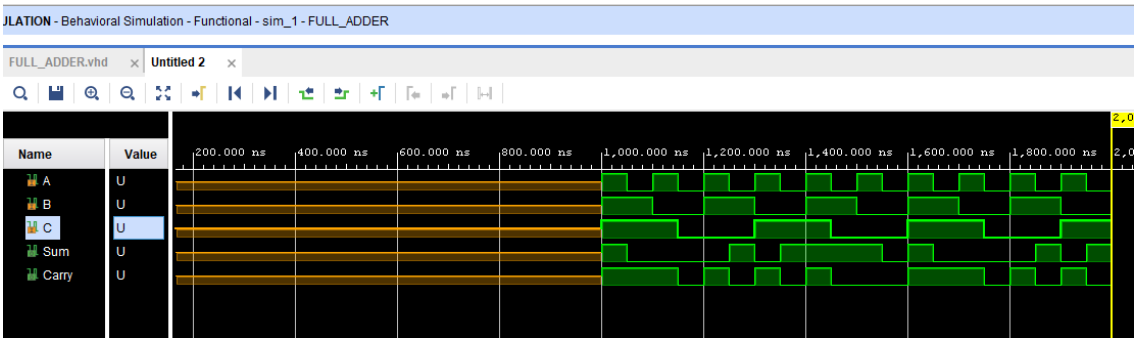


Figure 3 – Waveform Screenshot

Learning Outcome: Hence, half adder and full-Adder implemented in VHDL and its functionalities are verified. Also, we got to know how half adder and full-Adder works.

Experiment – 3

Aim: Write a program in VHDL for the implementation of Half-subtractor and full subtractor.

Software Used: Xilinx Vivado 2020.1

Truth Table:

Half-subtractor:

Inputs		Outputs	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Full-subtractor:

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Program:

```
begin
process(a,b,c)
begin
if(a = '0' and b = '0' and c = '0')
then
sum <= '0';
carry <= '0';
elsif((a = '0'and b = '0' and c = '1') or (a = '0' and b = '1' and c = '0') or (a = '1' and b = '0' and c
= '0'))
then
sum <= '1';
carry <= '0';
elsif((a = '1'and b = '1' and c = '1'))
then
sum <= '1';
carry <= '1';
else
```

```
        sum <= '0';

carry <= '1'; end if;

        end process;
```

RTL Schematics:

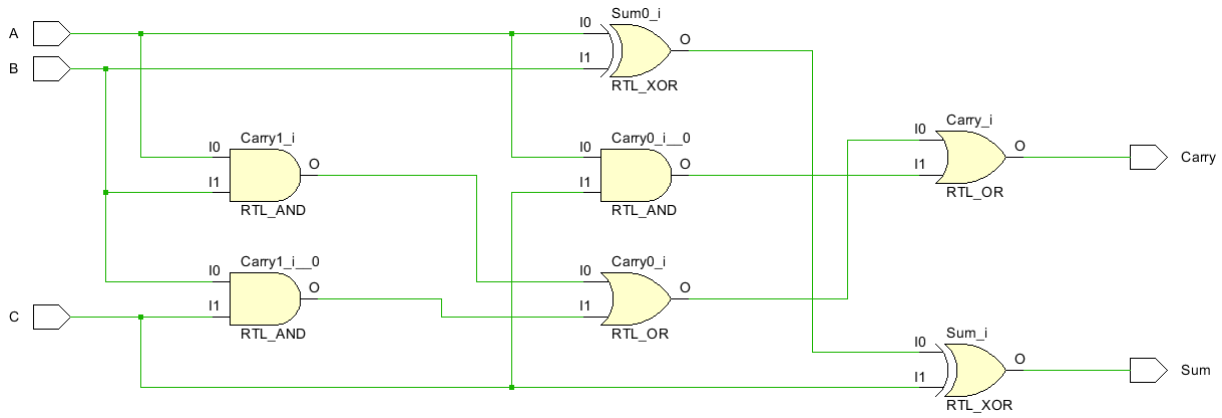


Figure 1 – Schematics Screenshot

Waveform:

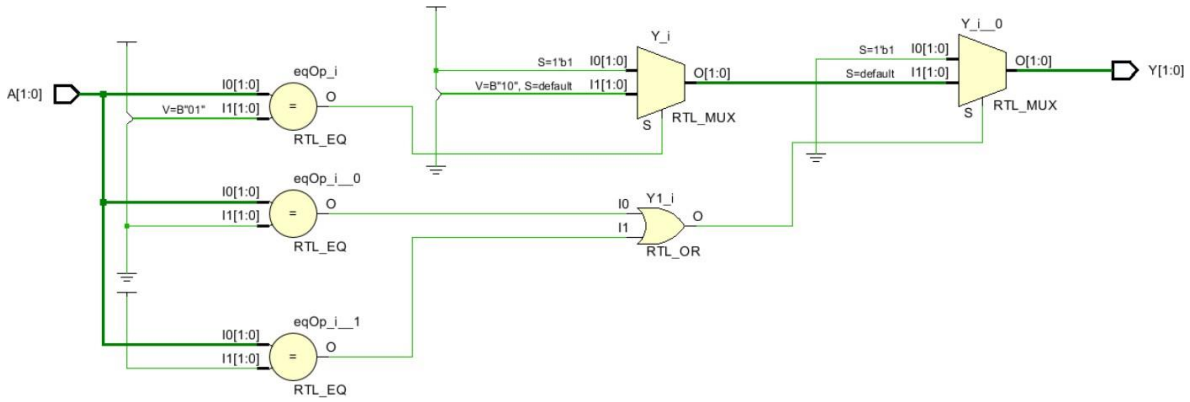
Figure 2 – Waveform Screenshot

Program:

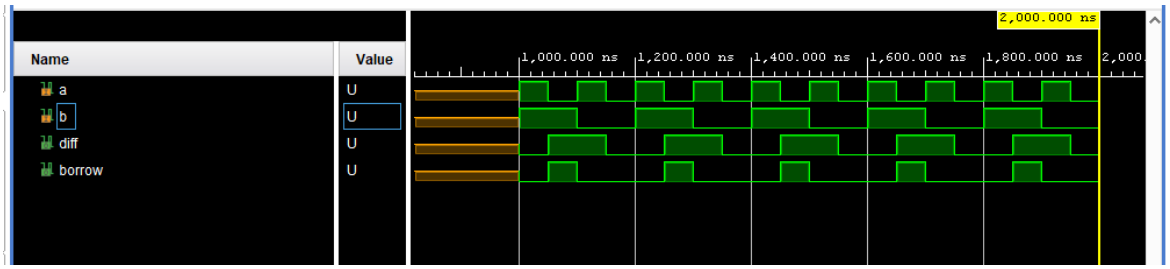
```
entity HALFSUBTRACTOR_BEHAVIORAL_SOURCE is
   Port ( A : in  STD_LOGIC_VECTOR (1 downto 0);
          Y : out  STD_LOGIC_VECTOR (1 downto 0));
end HALFSUBTRACTOR_BEHAVIORAL_SOURCE;
architecture Behavioral of HALFSUBTRACTOR_BEHAVIORAL_SOURCE is
begin
process(A)
begin
```

```
if (A = "00" or A = "11") then
Y<="00";
else if (A = "01") then
Y<="11";
else
Y<="10";
end if;
end if;
end process;
end Behavioral;
```

RTL Schematics:



Waveform:



Learning Outcome: Hence, full-subtractor and half-subtractor implemented in VHDL and its functionalities are verified. Also, gave us insight of working of full-subtractor and half-subtractor.

Experiment – 4

Aim: Implementation of multiplexers

Software Used: Xilinx Vivado 2020.1

Truth Table:

Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Process:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity mux_41_33 is

   Port ( i0 : in STD_LOGIC;

          i1 : in STD_LOGIC;

          i2 : in STD_LOGIC;

          i3 : in STD_LOGIC;

          y : out STD_LOGIC;

          sel : std_logic_vector(1 downto 0));

end mux_41_33;

architecture Behavioral of mux_41_33 is

begin

process(i0,i1,i2,i3,sel)

begin

case sel is

when "00" => y<=i0;

when "01" => y<=i1;

when "10" => y<=i2;

when others => y<=i3;

end case;

end process;
```

end Behavioral;

RTL Schematics:

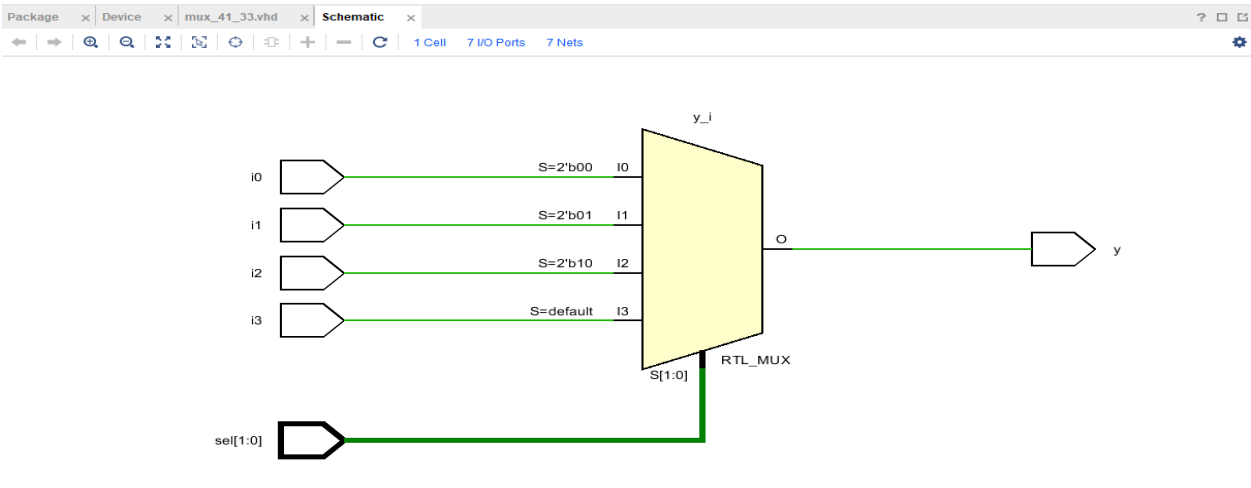


Figure 1 – Schematic Diagram of 4x1 Mux

Waveform:

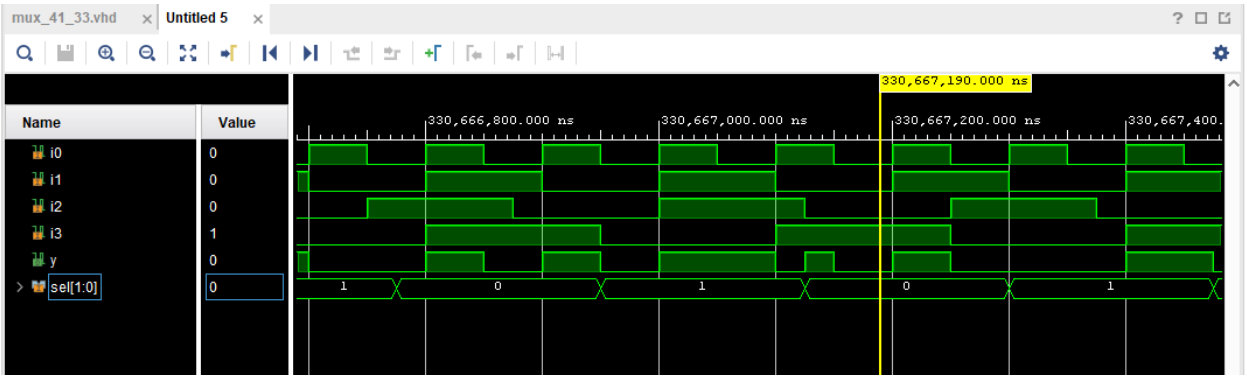
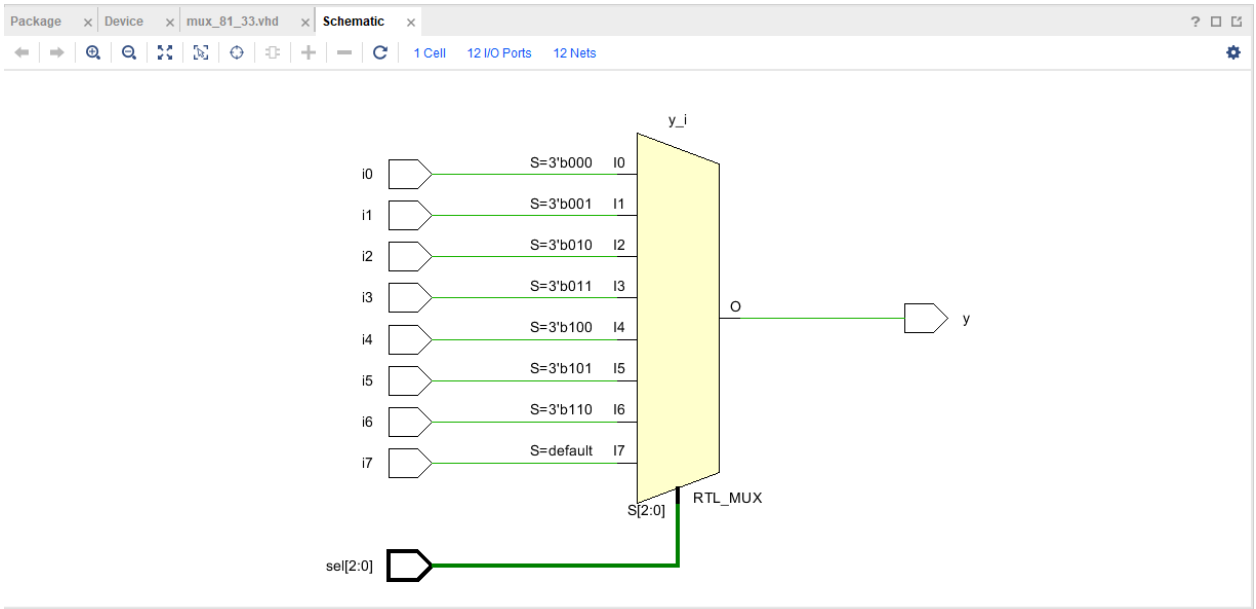
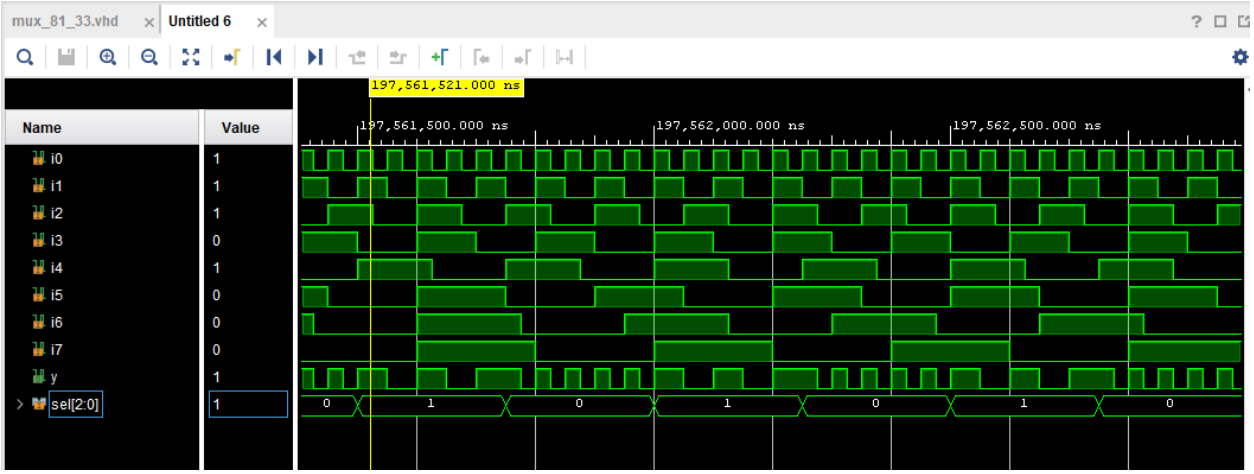


Figure 2 – Waveform of 4x1 Mux

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mux_81_33 is
   Port ( i0 : in STD_LOGIC;
          i1 : in STD_LOGIC;
          i2 : in STD_LOGIC;
          i3 : in STD_LOGIC;
          i4 : in STD_LOGIC;
          i5 : in STD_LOGIC;
          i6 : in STD_LOGIC;
```

```
        i7 : in STD_LOGIC;
        y : out STD_LOGIC;
        sel: std_logic_vector(2 downto 0));
end mux_81_33;

architecture Behavioral of mux_81_33 is

begin
process(i0,i1,i2,i3,i4,i5,i6,i7,sel)
begin
case sel is
when "000" => y<=i0;
when "001" => y<=i1;
when "010" => y<=i2;
when "011" => y<=i3;
when "100" => y<=i4;
when "101" => y<=i5;
when "110" => y<=i6;
when others => y<=i7;
end case;
end process;

end Behavioral;
```





Learning Outcome: Implementation of 4x1 and 8:1 mux is successful in Viavdo, Results were also verified.

Experiment – 5

Aim: Implementation of encoders

Software Used: Xilinx Vivado 2020.1

Truth Table:

A7	A6	A5	A4	A3	A2	A1	A0	Y2	Y1	Y0
0	0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Table 1 : truth table

Program:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity encoder8_3 is
    port(
        din : in STD_LOGIC_VECTOR(7 downto 0);
        dout : out STD_LOGIC_VECTOR(2 downto 0)
        );
end encoder8_3;

architecture encoder8_3_arc of encoder8_3 is
begin
dout <= "000"
when (din="10000000")
else  "001"
when (din="01000000")
else "010"
when (din="00100000")
else "011"
when (din="00010000")
```

```
else "100"
when (din="00001000")
else "101"
when (din="00000100")
else "110"
when (din="00000010")
else "111";

end encoder8_3_arc;
```

RTL Schematics:

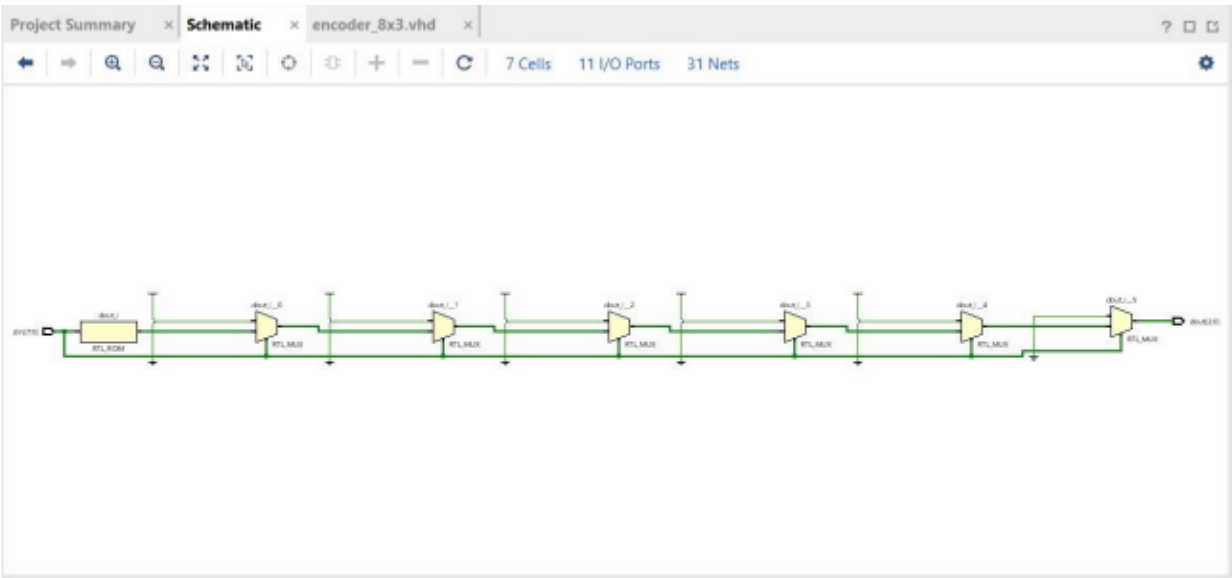


Fig.1: - encoder(8x3)

Figure 1 – RTL Schematic

Waveform:

Figure 2 – Waveform

Learning Outcome: Encoders are used in devices that need to operate in high speed and with high accuracy. The method of controlling the motor rotation by detecting the motor rotation speed and rotation angle using an encoder is called feedback control (closed loop method).

Experiment – 6

Aim: Implementation of decoders with behavioural modelling.

Software Used: Xilinx Vivado 2020.1

Truth Table:

3:8 decoder

a	b	c	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

2:4 decoder:

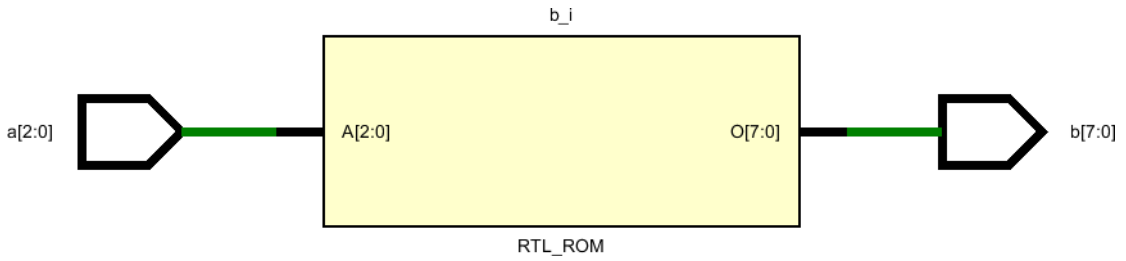
a	b	D3	D2	D1	D0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Program:

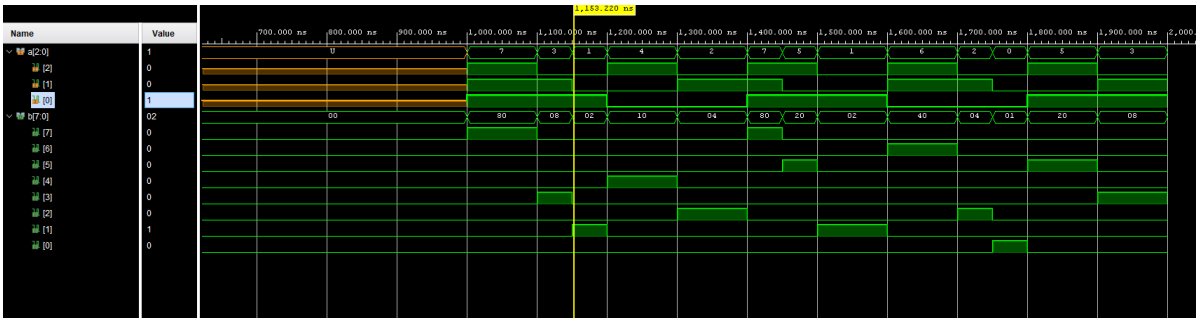
```
library IEEE;
use ieee.std_logic_1164.all;
entity decoder_3x8 is
port(a : in std_logic_vector(2 down to 0)
        d : out std_logic_vector(7 down to 0)
);
end decoder_3x8 ;
architecture behoavioral decoder_3x8 is
begin
process(a)
begin
case a is
when “000” => d <= “00000001”;
when “001” => d <= “00000010”;
```

```
when “010” => d <= “00000100”;
when “011” => d <= “00001000”;
when “100” => d <= “00010000”;
when “101” => d <= “00100000”;
when “110” => d <= “01000000”;
when “111” => d <= “10000000”;
end case;
end process;
end behavioral;
```

RTL Schematics:



Waveform:

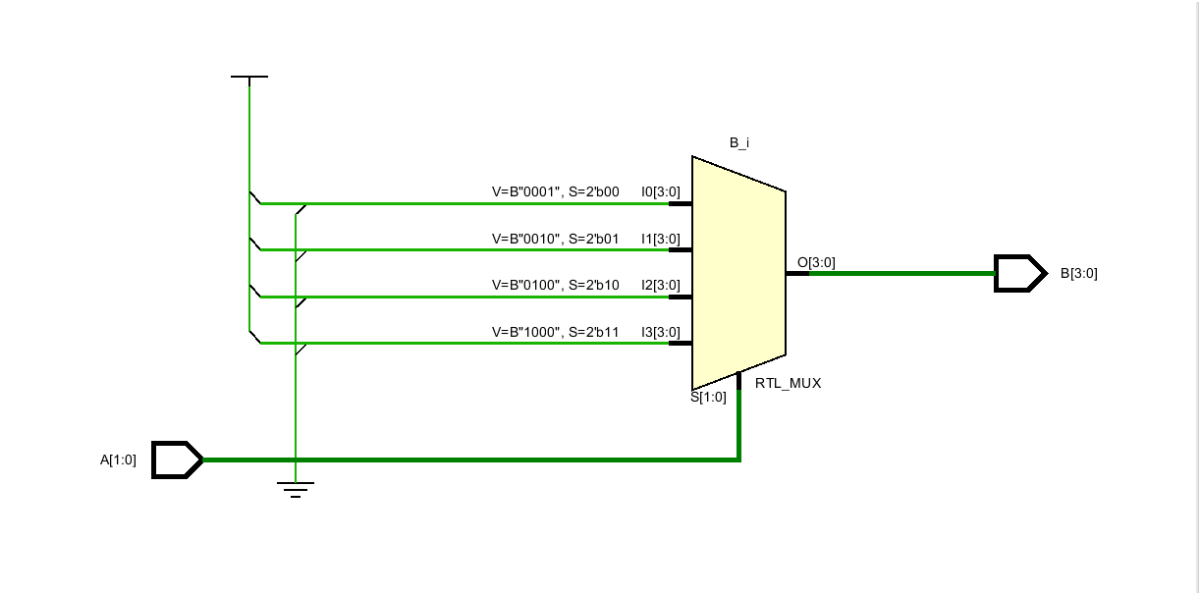


Program:

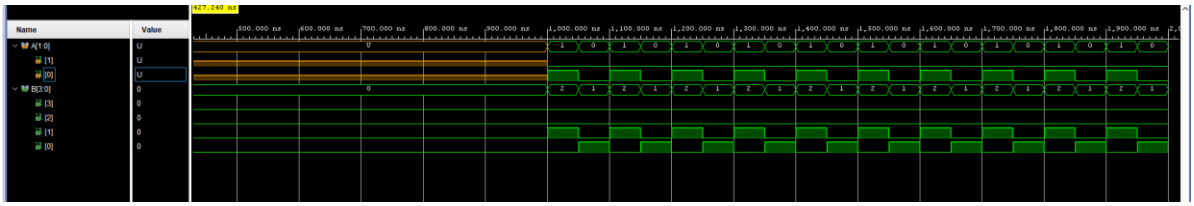
```
library IEEE;
using ieee.std_logic_1164.all;
entity decoder_2x4 is
port(a : in std_logic_vector(1 down to 0);
        d : out std_logic_vector(3 down to 0));
end decoder_2x4;
```

```
architecture behavioral of decoder_2x4 is
begin
process (a)
begin
case a is
when “00” => d =< “0001”;
when “01” => d =< “0010”;
when “10” => d =< “0100”;
when “11” => d =< “1000”;
when others => d <= “0000”;
end case;
end process;
end behavioral;
```

RTL Schematics:



Waveform:



Learning Outcome: Mentioned decoder are implemented in VHDL and their properties are verified.

Experiment – 7

Aim: Perform SR flip flop with behavioural modelling.

Software Used: Xilinx Vivado 2020.1

Truth Table:

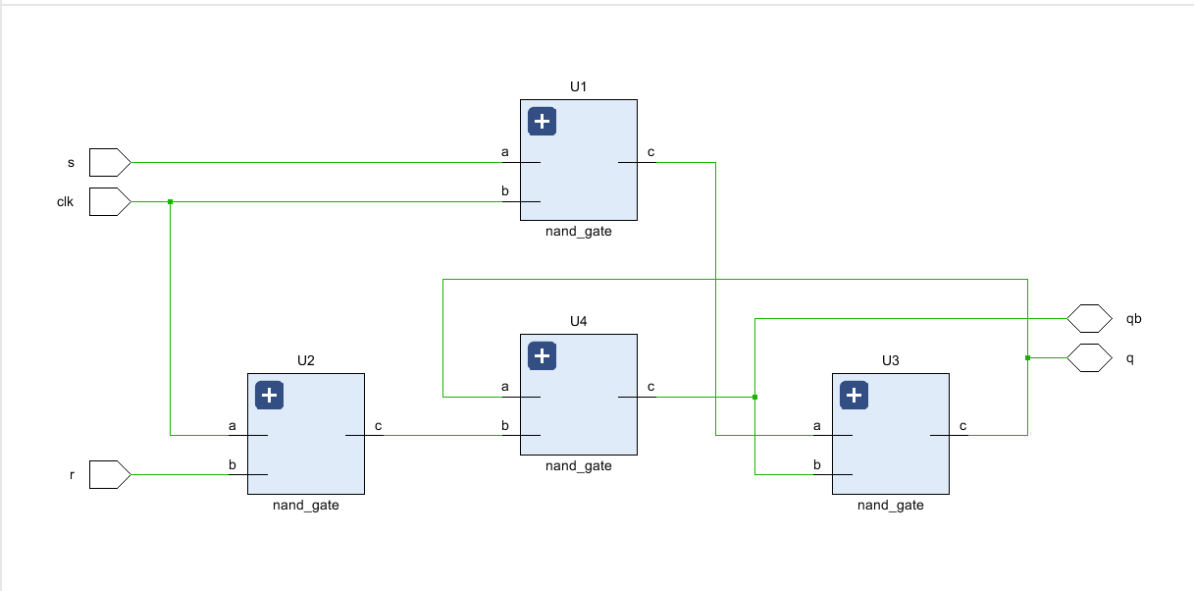
CLK	S	R	Q	Q'
0	x	x	Qprv	Q'prv
1	0	0	Qprv	Q'prv
1	0	1	0	1
1	1	0	1	0
1	1	1	–	–

Program:

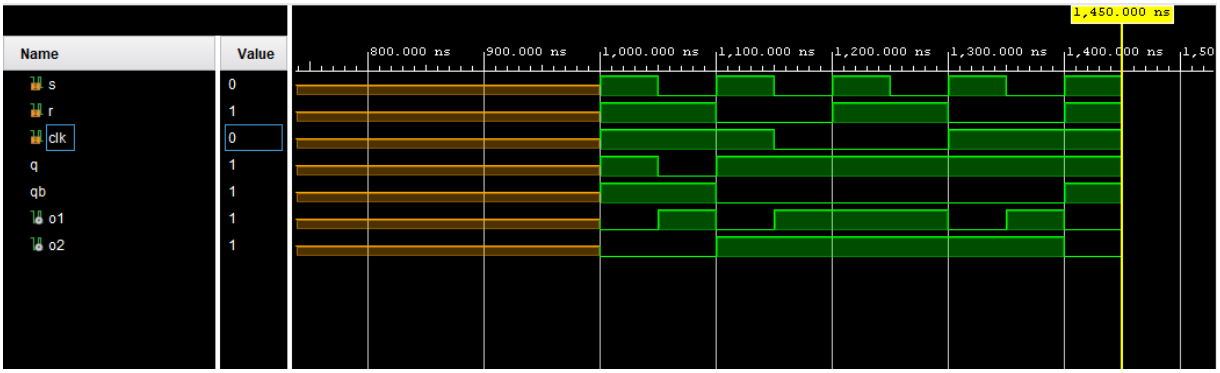
```
entity SR_FLIPFLOP_SOURCE is
   Port ( S,R,RST,CLK : in  STD_LOGIC;
          Q,Qb : out STD_LOGIC);
end SR_FLIPFLOP_SOURCE;
architecture Behavioral of SR_FLIPFLOP_SOURCE is
begin
process (S,R,RST,CLK)
begin
if (RST = '1') then
Q <= '0';
elsif (RISING_EDGE(CLK))then
if (S /= R) then
Q <= S;
Qb <= R;
elsif (S = '1' AND R = '1') then
Q <= 'Z';
Qb <= 'Z';
end if;
end if;
end process;
```

end Behavioral;

RTL Schematics:



Waveform:



Learning Outcome: In this experiment, SR flip flop was successfully implemented in Vivado and results are varified.

Experiment – 8

Aim: Write a program to implement JK flip-flop in VHDL.

Software Used: Xilinx Vivado 2020.1

Truth Table:

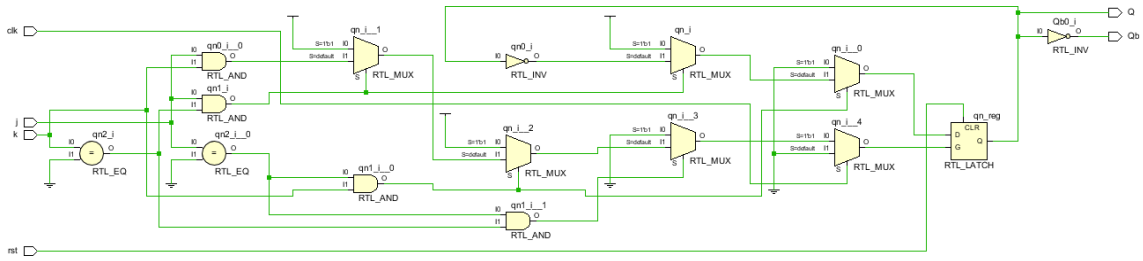
Clock	J	K	Q_{n+1}	State
0	x	x	Q_n	
1	0	0	Q_n	Hold
1	0	1	0	Reset
1	1	1	1	Set
1	1	1	$\overline{Q_n}$	Toggle

Program:

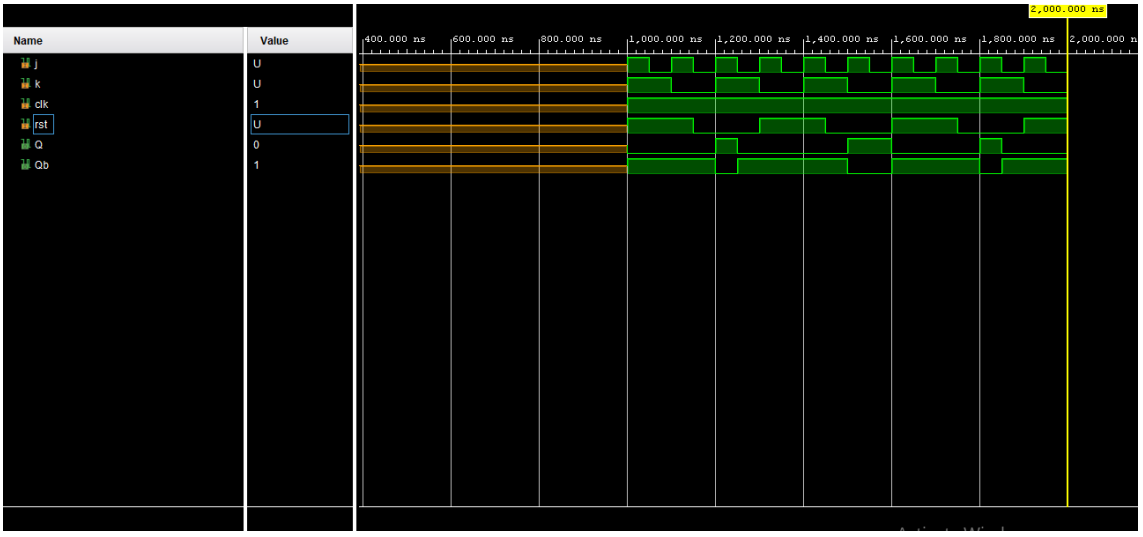
```
entity JK_FF is
        port( J, K, clk, rst : in std_logic;
                Q, Qbar : out std_logic);
end JK_FF;
architecture behavioral of JK_FF is
begin
process(clk, rst)
variable qn : std_logic;
begin
if(rst = '1')then
qn := '0';
elsif(clk'event and clk = '1')then
if(J='0' and K='0')then
qn := qn;
elsif(J='0' and K='1')then
qn := '0';
elsif(J='1' and K='0')then
qn := '1';
```

```
elsif(J='1' and K='1')then
qn := not qn;
else
null;
end if;
else
null;
end if;
Q <= qn;
Qbar <= not qn;
end process;
end behavioral;
```

RTL Schematics:



Waveform:



Conclusion: JK flip-flop have been implemented in VHDL and its properties are verified.

Experiment-09

Aim: Implementation of SISO, SIPO, PIPO and PISO registers using behavioral model.
Software Required: Xilinx Vivado 2020.1
Theory:

SISO: allows serial input (one bit after the other through a single data line) and produces a serial output is known as a Serial-In Serial-Out shift register. Since there is only one output, the data leaves the shift register one bit at a time in a serial pattern.

SIPO: The shift register, which allows serial input (one bit after the other through a single data line) and produces a parallel output, is known as the Serial-In Parallel-Out shift register.

PIPO: The shift register, which allows parallel input (data is given separately to each flip flop and in a simultaneous manner) and also produces a parallel output is known as Parallel-In parallel-Out shift register.

PISO: The shift register, which allows parallel input (data is given separately to each flip flop and in a simultaneous manner) and produces a serial output is known as a Parallel-In Serial-Out shift register.

Truth Table:

SISO:

CLK	Q_3	Q_2	Q_1	Q_0
Initially	0	0	0	0
1st falling edge	1	0	0	0
2nd falling edge	1	1	0	0
3rd falling edge	1	1	1	0
4th falling edge	1	1	1	1

SIPO:

CLK Pulse	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	0	1	0
4	1	1	0	1

PIPO

CLK Pulse	QA	QB	QC
0	0	0	0
1	1	1	0

PISO

CLK Pulse	Q_A	Q_B	Q_C	Q_D (Data Out
0	0	0	0	0
1	1	1	0	1
2	0	1	1	0
3	0	0	1	1
4	0	0	0	1

Coding:

SISO

```
library IEEE; use IEEE.STD LOGIC 1164. ALL;
entity SISO_21209 in
Port (a: in STD LOGIC vector (0 to 3); elk in STD LOGIC:
b: out std logic_vector (0 to 3)): end SISO_21209;
architecture Behavioral of SISO_21209 is
begin
process(a,clk)
begin
if (clk'l' and clk'EVENT) then if a="0000" then b<="0000"; elsif a="1000" then b<="1000": elsif a="1100" then
b<="1100"; Jelaif a="1110" then b<="1110"; else b<="1111";
end if: end if;
end process; end Behavioral;
```

SIPO

```
library ieee;
use ieee.std_logic_1164.all;
entity sipo is
port(
clk, clear: in std_logic;
Input_Data: in std_logic;
Q: out std logic_vector(3 downto 0) ); end sipo;
architecture arch of sipo is
begin
process (clk) begin
if clear = '1' then Q<= "0000";
elsif (CLK'event and CLK='1') then Q(3 downto 1) <= Q(2 downto 0); Q(0) <= Input_Data;
end if;
end process;
end arch;
```

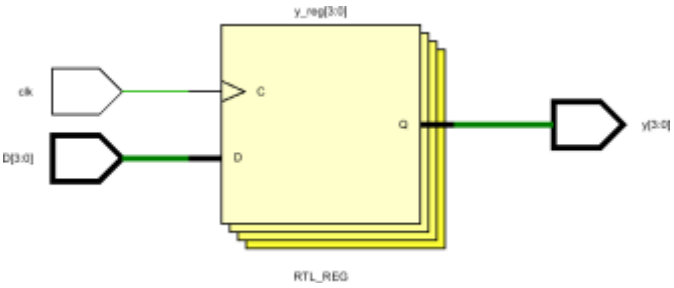
PIPO

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity pipo_21209 is
port(
     clk: in std_logic;
     D: in std_logic_vector(3 downto 0);
     y: out std_logic_vector(3 downto 0));
end pipo_21209;

architecture Behavioral of pipo_21209 is

begin
process(clk)
begin
if (clk='1' and clk'event)then
y<=D;
end if;
end process;
end behavioral;
```

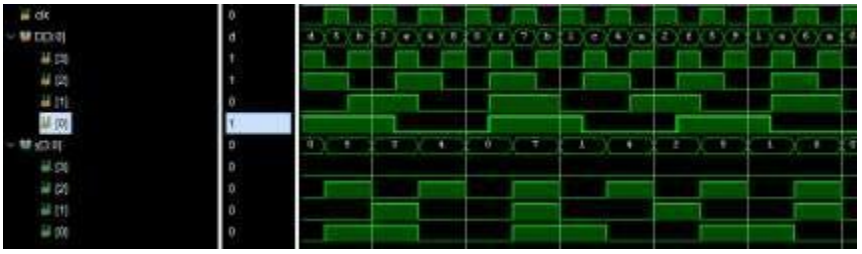
PISO

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity parallel_in_serial_out is
    port(
       clk : in STD_LOGIC;
       reset : in STD_LOGIC;
       load : in STD_LOGIC;
       din : in STD_LOGIC_VECTOR(3 downto 0);
       dout : out STD_LOGIC
       );
end parallel_in_serial_out;


architecture piso_arc of parallel_in_serial_out is
begin

   piso : process (clk,reset,load,din) is
   variable temp : std_logic_vector (din'range);
   begin
      if (reset='1') then
         temp := (others=>'0');
      elsif (load='1') then
         temp := din ;
      elsif (rising_edge (clk)) then
         dout <= temp(3);
         temp := temp(2 downto 0) & '0';
      end if;
   end process piso;

end piso_arc;
```

Schematic Result:



RTL Result



Learning Outcome:
Understood the basic concepts of VHDL design.
Familiarity with the Vivado design environment and user interface.
Create and simulate simple RTL designs using VHDL and was able to simulate them.

Experiment-10

Objective: Implementation of 4-bit UP and DOWN counter using behavioral model.
Software Required: Xilinx Vivado 2020.1
Theory:

Counter- In digital logic and computing, a Counter is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal. Counters are used in digital electronics for counting purpose, they can count specific event happening in the circuit. For example, in UP counter a counter increases count for every rising edge of clock. Not only counting, a counter can follow the certain sequence based on our design like any random sequence 0,1 ,3,2 They can also be designed with the help of flip flops. Flip-flops. It is a group of flip-flops with a clock signal applied.

Synchronous counter has one global clock which drives each flip flop so output changes in parallel. The one advantage of synchronous counter over asynchronous counter is, it can operate on higher frequency than asynchronous counter as it does not have cumulative delay because of same clock is given to each flip flop.

Truth Table:

Rst	CLK	O3	O2	O1	O0
1	↑	0	0	0	0
0	↑	0	0	0	1
0	↑	0	0	1	0
0	↑	0	0	1	1
0	↑	0	1	0	0
0	↑	0	1	0	1
0	↑	0	1	1	0
0	↑	0	1	1	1
0	↑	1	0	0	0
0	↑	1	0	0	1
0	↑	1	0	1	0
0	↑	1	0	1	1
0	↑	1	1	0	0
0	↑	1	1	0	1
0	↑	1	1	1	0
0	↑	1	1	1	1
0	↑	0	0	0	0

Coding:

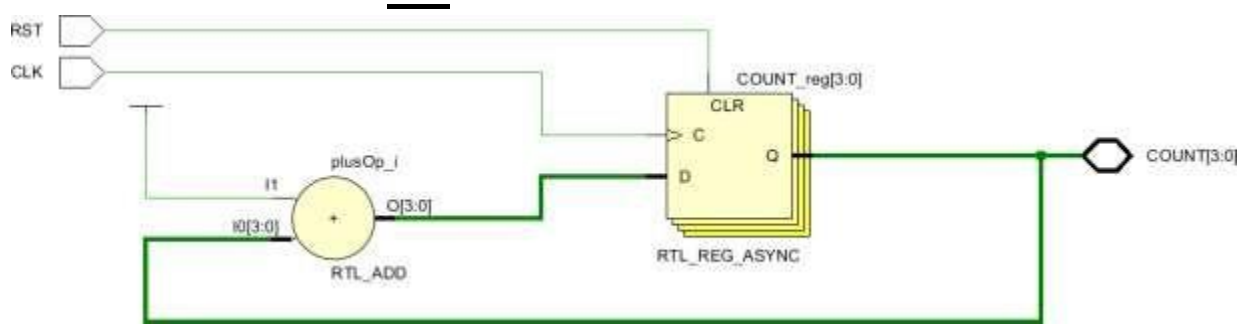
UP

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity up_counter_21209 is
    Port ( CLK,RST : in  STD_LOGIC;
           COUNT : inout  STD_LOGIC_VECTOR (3 downto 0));
end up_counter_21209;

architecture Behavioral of up_counter_21209 is

begin
process (CLK,RST)
begin

if (RST = '1')then
COUNT <= "0000";
elsif(rising_edge(CLK))then
COUNT <= COUNT+1;

end if;
end process;
end Behavioral;
```

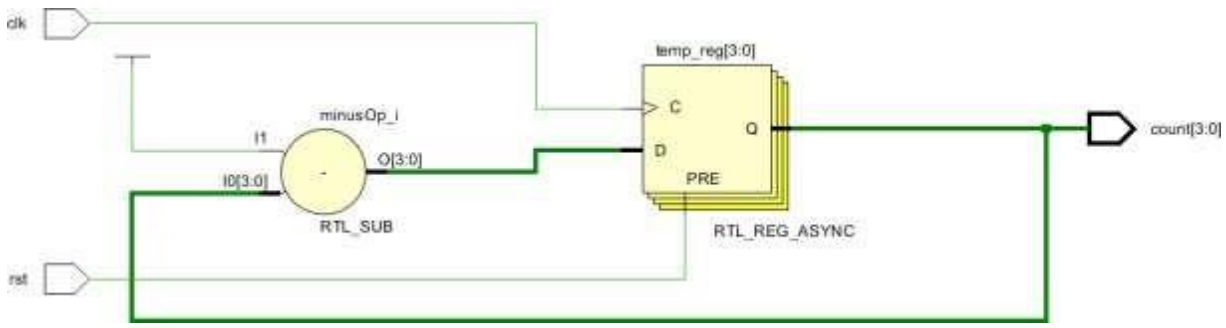
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity down_counter_21209 is
    Port ( clk,rst : in  STD_LOGIC;
           count : out  STD_LOGIC_VECTOR (3 downto 0));
end down_counter_21209 ;

architecture Behavioral of down_counter_21209  is

signal temp:std_logic_vector(3 downto 0);
begin
process(clk,rst)
begin

if(rst='1')then
temp<="1111";
elsif(rising_edge(clk))then
temp<=temp-1;
end if;
end process;

count<=temp;
end Behavioral;
```

Schematic Result:

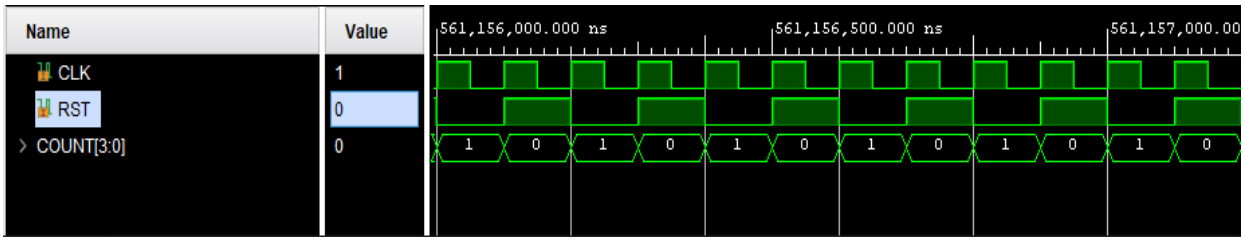
UP



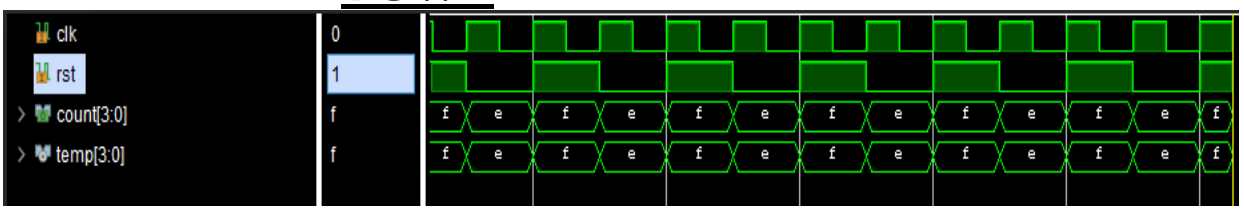
DOWN



RTL Result UP



DOWN



Learning Outcome:

Understood the basic concepts of VHDL design.
Familiarity with the Vivado design environment and user interface.
Create and simulate simple RTL designs using VHDL and was able to simulate them.

Experiment-11

Aim: Implementation of Johnson and Ring counter using behavioral model.
Software Required: Xilinx Vivado 2020.1
Theory:

Counter- In digital logic and computing, a Counter is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal.

In Johnson counter, the complemented output of last flip flop is connected to input of first flip flop and to implement n-bit Johnson counter we require n flip-flop. It is one of the most important type of shift register counter. It is formed by the feedback of the output to its own input. Johnson counter is a ring with an inversion.

A ring counter is a typical application of the Shift register. The ring counter is almost the same as the shift counter. The only change is that the output of the last flip-flop is connected to the input of the first flip-flop in the case of the ring counter but in the case of the shift register it is taken as output.

I. Truth Table:

JOHNSON COUNTER

Clock	Q_0	Q_1	Q_2	Q_3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

RING COUNTER

CLK	D_1	D_2	D_3	D_4	D_5	D_6
1	1	0	0	0	0	0
2	0	1	0	0	0	0
3	0	0	1	0	0	0
4	0	0	0	1	0	0
5	0	0	0	0	1	0
6	0	0	0	0	0	1

Coding:

Johnson Counter

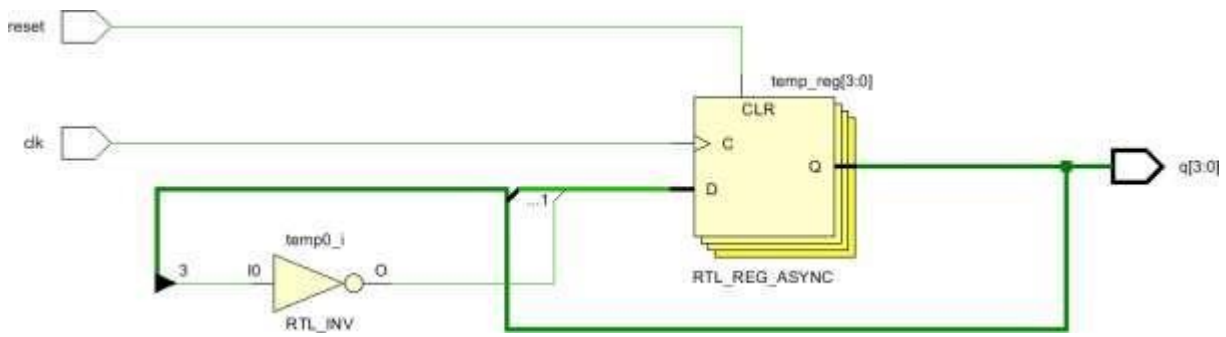
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Johnson_21209 is
Port ( clk : in STD_LOGIC;
reset : in STD_LOGIC;
q : out STD_LOGIC_VECTOR (3 downto 0));
end Johnson_21209;

architecture Behavioral of Johnson_21209 is
signal temp: std_logic_vector(3 downto 0):= "0000";
begin
process(clk,reset)
begin
if reset = '1' then
temp <= "0000";
elsif Rising_edge(clk) then
temp(1) <= temp(0);
temp(2) <= temp(1);
temp(3) <= temp(2);
temp(0) <= not temp(3);
end if;
end process;
q <= temp;
end Behavioral;
```

Ring Counter

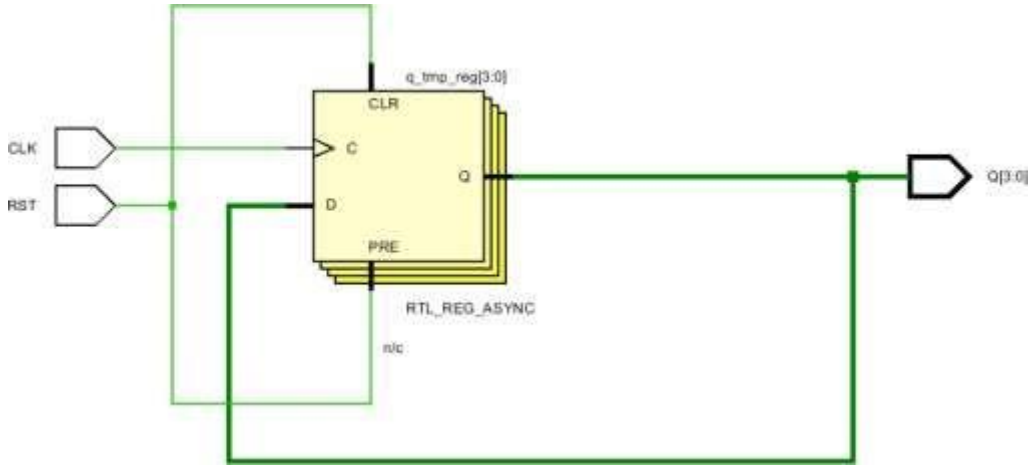
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Ring_counter is
    Port ( CLK : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           Q : out  STD_LOGIC_VECTOR (3 downto 0));
end Ring_counter;

architecture Behavioral of Ring_counter is
signal q_tmp: std_logic_vector(3 downto 0):= "0000";
begin
process(CLK,RST)
begin
if RST = '1' then
    q_tmp <= "0001";
elsif Rising_edge(CLK) then
    q_tmp(1) <= q_tmp(0);
    q_tmp(2) <= q_tmp(1);
    q_tmp(3) <= q_tmp(2);
    q_tmp(0) <= q_tmp(3);
end if;
end process;
Q <= q_tmp;
end Behavioral;
```

Schematic Result:

JOHNSON

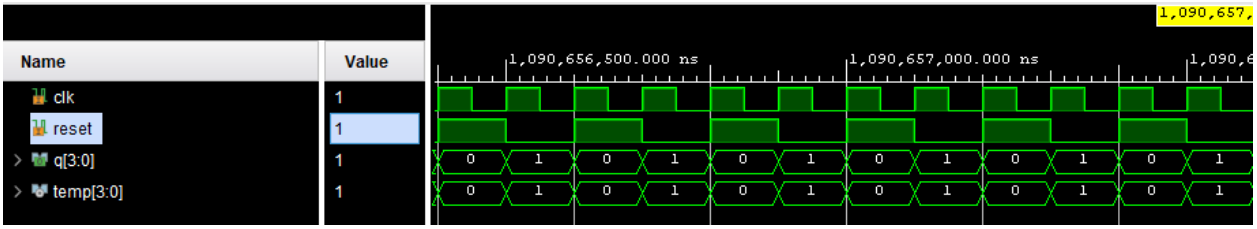


RING

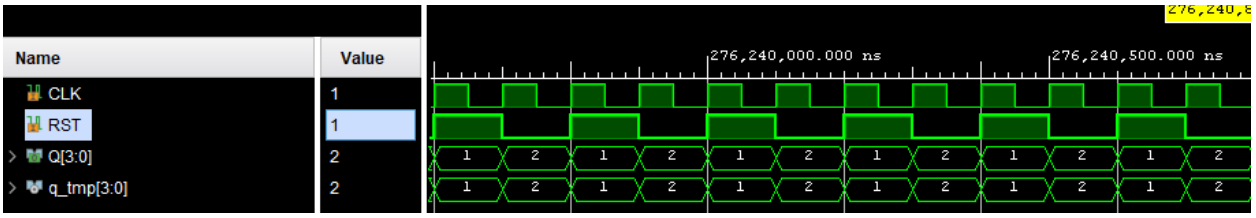


II.RTL Result

JOHNSON



RING COUNTER



Learning Outcome:

Understood the basic concepts of VHDL design.
Familiarity with the Vivado design environment and user interface.
Create and simulate simple RTL designs using VHDL and was able to simulate them.

Experiment – 12

Aim: Implementation of Half Adder and Full Adder using data flow model.

Software Used: Xilinx Vivado 2020.1

Truth Table:

Half-Adder:

Truth Table			
Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

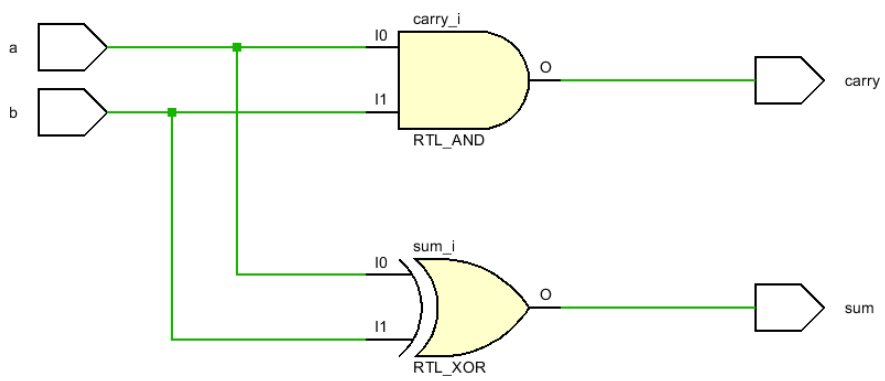
Full-Adder:

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Program:

```
entity H_adder is
port(
a,b : IN std_logic;
sum,carry : OUT std_logic);
end H_adder;
architecture dataflow of H_adder is
begin
sum <= a xor b;
carry <= a and b;
end dataflow;
```

RTL Schematics:

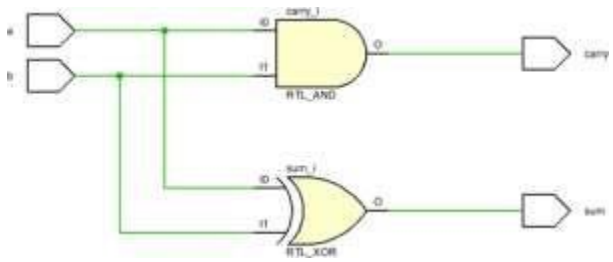


Waveform:

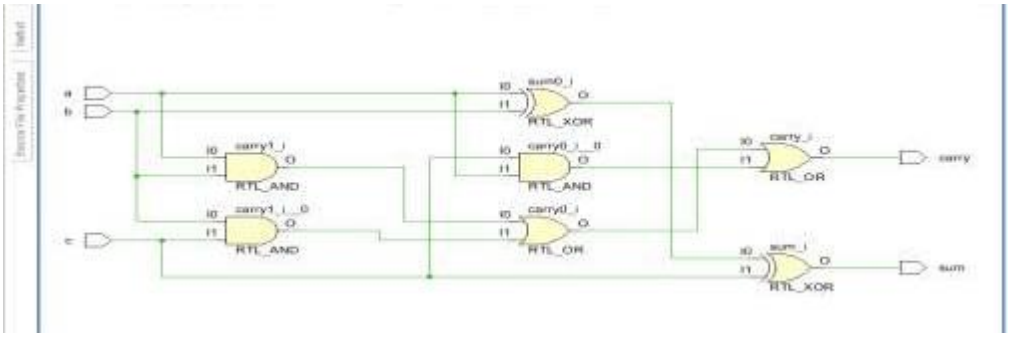
Full Adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity FULLADDER is
   Port ( A, B, C : in STD_LOGIC;

   Sum, carry : out  STD_LOGIC);
end FULLADDER ;
architecture  dataflow of FULLADDE  is

begin

Sum<= A xor B xor C;

carry<= (A and B) or (B and C) or (C and A);

end dataflow;
```

Schematic Result: Half Adder

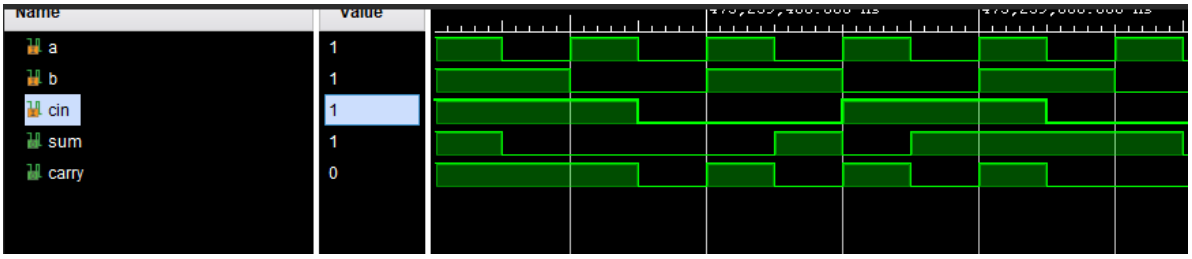


Full Adder



RTL Result

Full Adder



Learning Outcome:

Understood the basic concepts of VHDL design.
Familiarity with the Vivado design environment and user interface.
Create and simulate simple RTL designs using VHDL and was able to simulate them.

Experiment – 13

Aim: Write a program in VHDL for the implementation of Half subtractor and Full subtractor using dataflow model.

Software Used: Xilinx Vivado 2020.1

Truth Table:

Half Subtractor:

A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Full Subtractor:

A	B	B_{in}	Diff	Borrow
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

Program:

Half-Subtractor

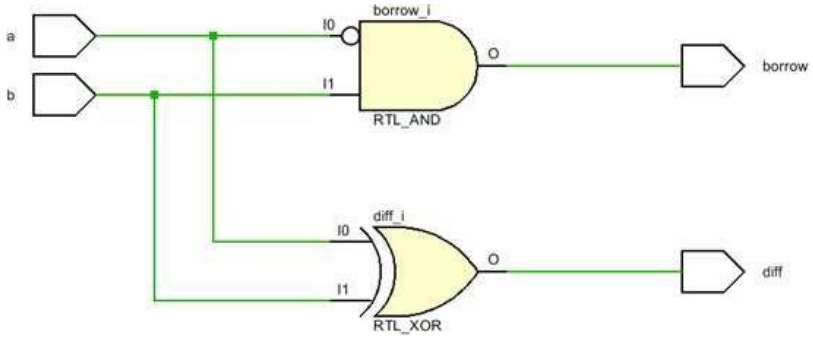
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity half_sub is
port( A, B : in std_logic;
DIFF, Borrow : out std_logic);
end entity;
architecture dataflow of half_sub is
begin
DIFF <= A xor B;
Borrow <= (not A) and B;
end architecture;
```

Full Subtractor

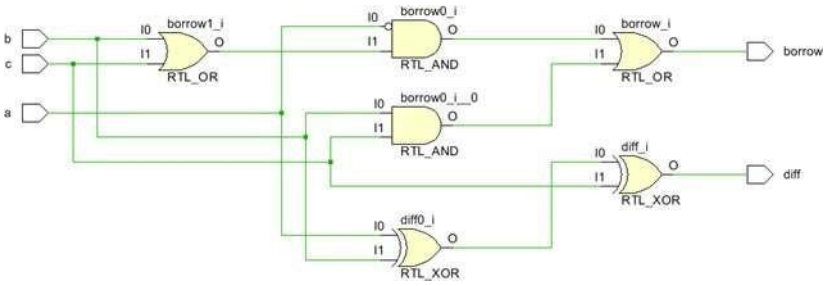
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity full_sub is
port( A, B, C : in std_logic;
DIFF, Borrow : out std_logic);
end entity;
architecture dataflow of full_sub is
begin
DIFF <= (A xor B) xor C;
Borrow <= ((not A) and (B or C)) or (B and C);
end dataflow;
```

RTL Schematics Half

Subtractor

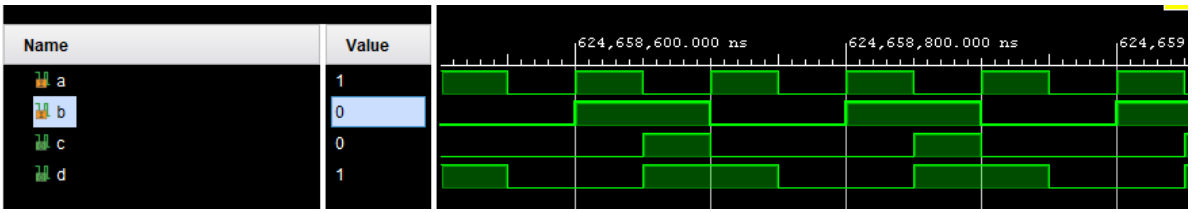


Full Subtractor

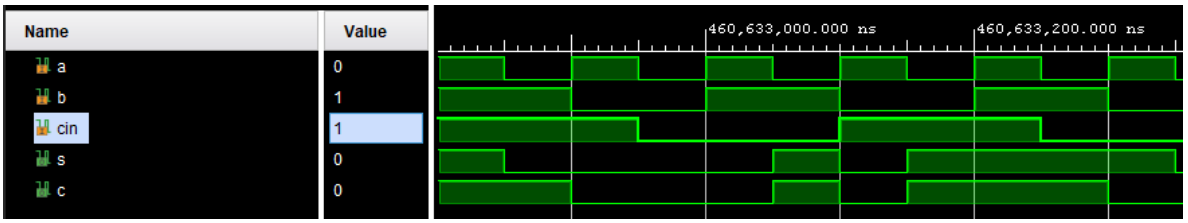


RTL Result

Half Subtractor



Full Subtractor



Learning Outcome:

Understood the basic concepts of VHDL design.
Familiarity with the Vivado design environment and user interface.
Create and simulate simple RTL designs using VHDL and was able to simulate them.

Experiment – 14

Aim: Write a program in VHDL for the implementation of multiplexer using dataflow model.

Software Used: Xilinx Vivado.

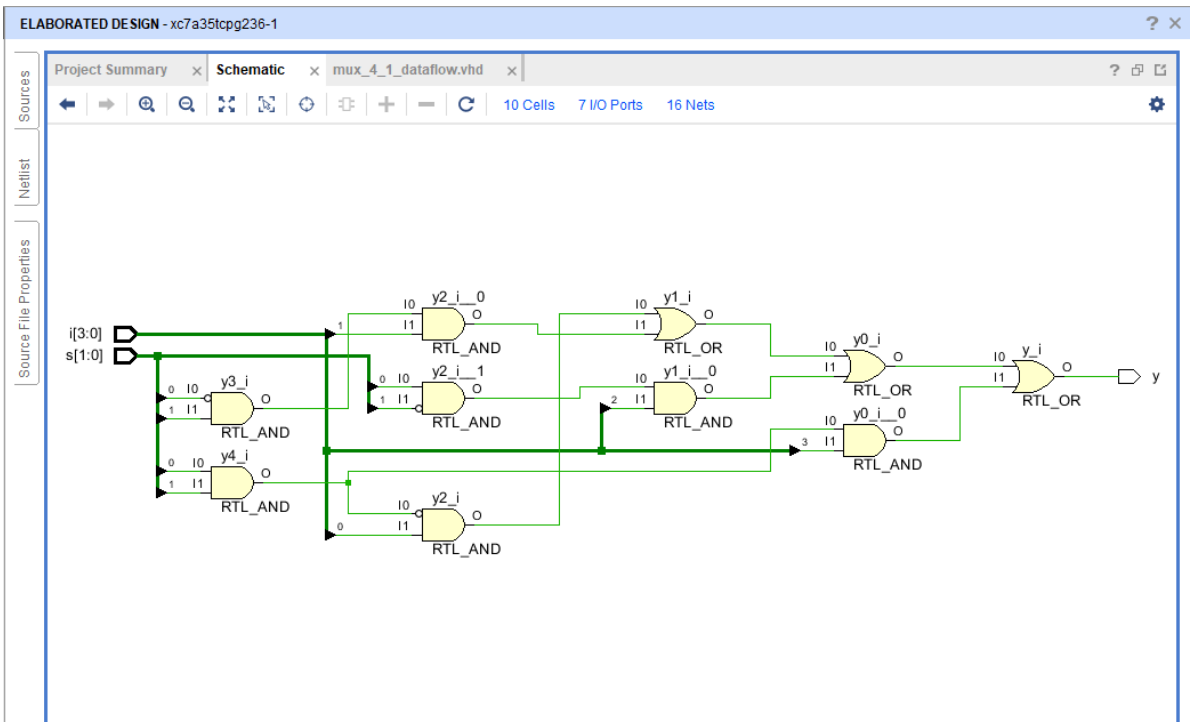
Truth Table:

Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

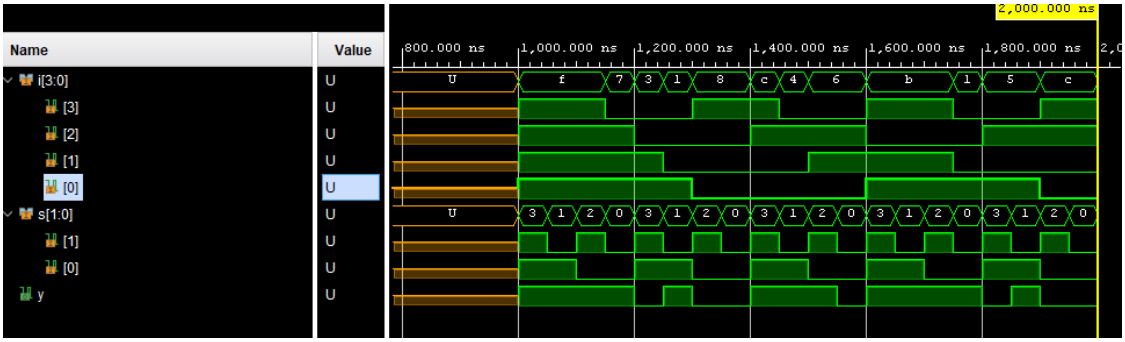
Program:

```
entity mux_4_1_dataflow is
    Port ( i : in STD_LOGIC_vector(3 downto 0);
           s : in STD_LOGIC_vector(1 downto 0);
           y : out STD_LOGIC);
end mux_4_1_dataflow;

architecture Behavioral of mux_4_1_dataflow is

begin
y<= ((not(s(0)and s(1))) and i(0)) or ((((not(s(0))) and s(1)))and i(1)) or ((s(0)and (not s(1))) and i(2)) or ((s(0) and s(1))

end Behavioral;
```

RTL Schematics:



Waveform:



Learning Outcome: Understood the basic concepts of VHDL design.
Familiarity with the Vivado design environment and user interface.
Create and simulate simple RTL designs using VHDL and was able to simulate them.

Experiment – 15

Aim: Write a program in VHDL for the implementation of 1x4 demultiplexer using dataflow model.

Software Used: Xilinx Vivado.

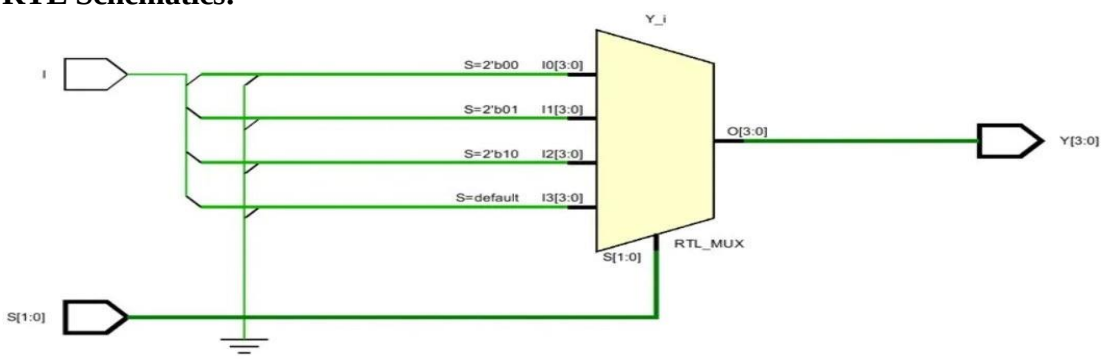
Truth Table:

Inputs			Outputs			
E	S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	X	X	0	0	0	0
1	0	0	I	0	0	0
1	0	1	0	I	0	0
1	1	0	0	0	I	0
1	1	1	0	0	0	I

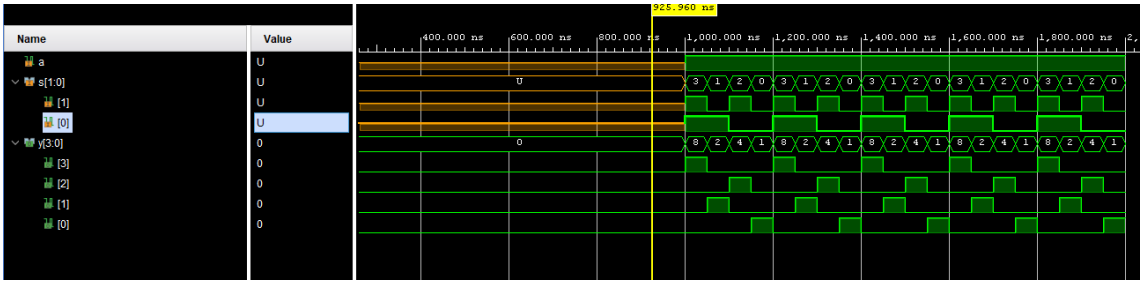
Program:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity DEMUX is
   Port ( I : in STD_LOGIC;
        S1, S0 : in STD_LOGIG;
        Y0,Y1,Y2,Y3 : out STD_LOGIC);

end DEMUX;
architecture dataflow of DEMUX is
begin
Y0<= (not S0) and (not S1) and I;
Y1<= (S0) and (not S1) and I;
Y2<= (not S0) and (S1) and I;
Y3<= (S0) and (S1) and I;
end dataflow;
```

RTL Schematics:



Waveform:



Learning Outcome: Understood the basic concepts of VHDL design.
Familiarity with the Vivado design environment and user interface.
Create and simulate simple RTL designs using VHDL and was able to simulate them.

Experiment-16

Aim: Implementation of 4x2 Encoder using dataflow model.

Software Required: Xilinx Vivado 2020.1

Theory: A 4x2 encoder is a digital circuit that takes four input lines and encodes them into a 2-bit binary output, where the output represents the position of the active input. It detects the first active input and encodes its position in binary form.

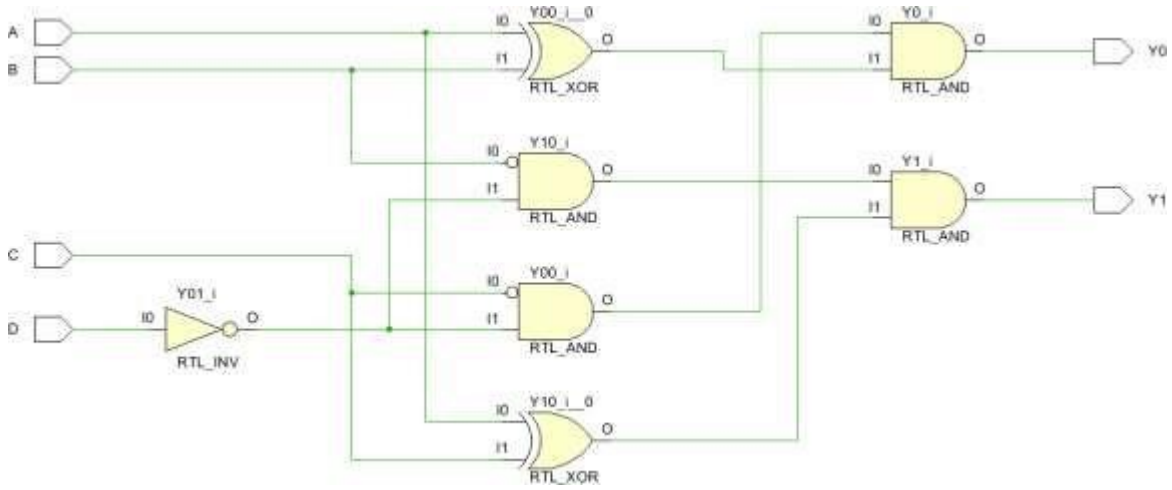
Truth Table:
4x2 Encoder:

I0	I1	I2	I3	Q1	Q0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	1	1	0
1	0	0	0	1	1

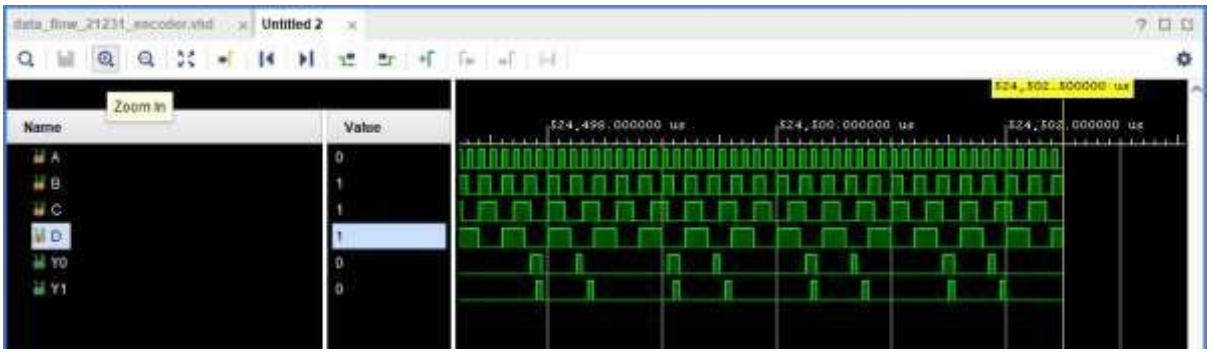
Coding and Output:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity data_flow_21231_encoder is
   Port ( A,B,C,D : in STD_LOGIC;
          Y0,Y1 : out STD_LOGIC);
end data_flow_21231_encoder;

architecture dataflow of data_flow_21231_encoder is

begin
Y0 <= ((not C)and(not D))and(A xor B);
Y1 <= ((not B)and(not D))and(a xor C);

end dataflow;
```

Schematic Result:



RTL Result



Learning Outcome:

- Understood the basic concepts of VHDL design.
- Familiarity with the Vivado design environment and user interface.
- Create and simulate simple RTL designs using VHDL and was able to simulate them.

Experiment – 17

Aim: Write a program in VHDL for the implementation of full adder and half adder using structural modelling.

Software Used: Xilinx Vivado 2020.1

Truth Table:

Full-Adder:

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Half-Adder:

Truth Table			
Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

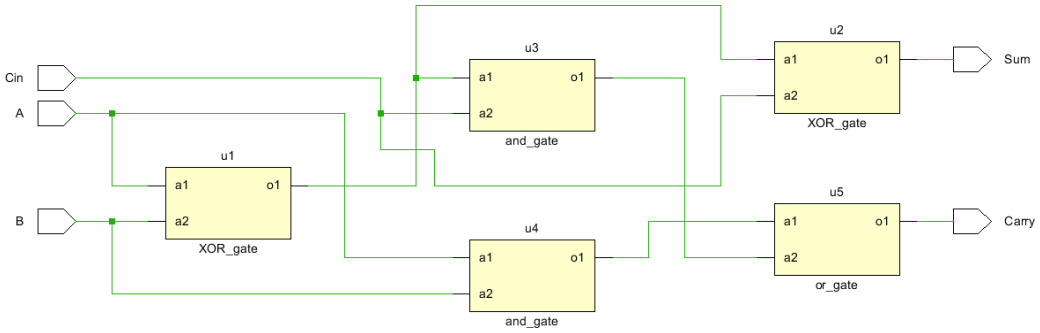
Program:

```
entity FAdder is
Port ( FA, FB, FC : in STD_LOGIC;
        FS, FCA : out STD_LOGIC);
end FAdder;
architecture structural of FAdder is
component HA is
Port ( A,B : in STD_LOGIC;
      S,C : out STD_LOGIC);
end component;
component ORGATE is
Port ( X,Y: i STD_LOGIC;
        Z: out STD_LOGIC);
end component;
SIGNAL S0,S1,S2:STD_LOGIC;
begin
U1:HA PORT MAP(A=>FA,B=>FB,S=>S0,C=>S1);
U2:HA PORT MAP(A=>S0,B=>FC,S=>FS,C=>S2);
```

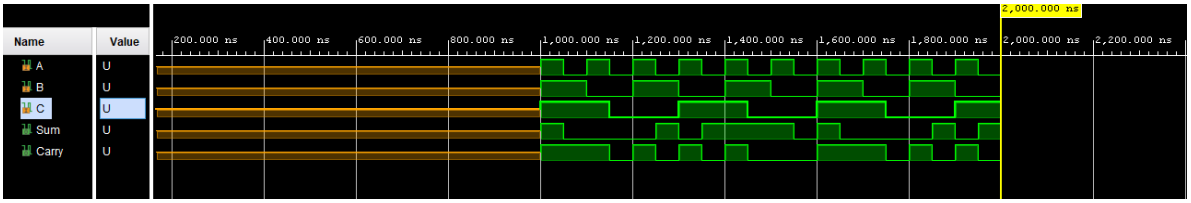
U3:ORGATE PORT MAP(X=>S2,Y=>S1,Z=>FCA);

end structural;

RTL Schematics:



Waveform:

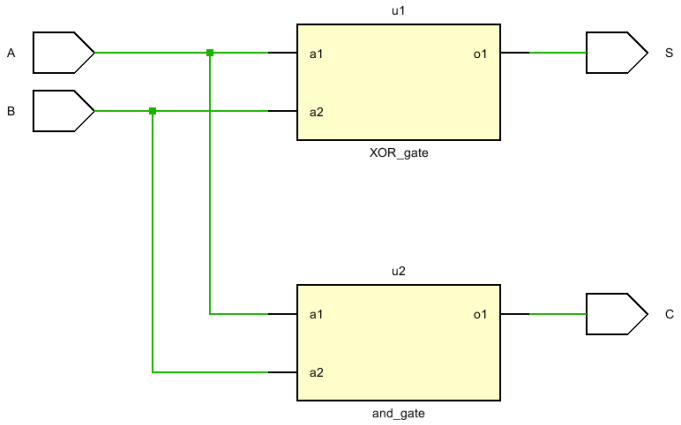


Program:

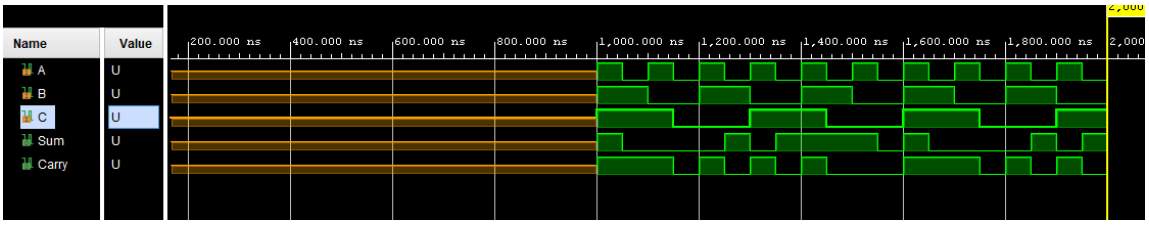
```
entity half_adder is
  port (a, b: in std_logic;
    sum, carry_out: out std_logic);
end half_adder;
architecture structure of half_adder is
  component xor_gate
    port (i1, i2: in std_logic;
      o1: out std_logic);
  end component;
  component and_gate
    port (i1, i2: in std_logic;
      o1: out std_logic);
  end component;
```

```
begin
   u1: xor_gate port map (i1 => a, i2 => b, o1 => sum);
   u2: and_gate port map (i1 => a, i2 => b, o1 => carry_out);
end structure;
```

RTL Schematics:



Waveform:



Learning Outcome: Understood the basic concepts of VHDL design.
Familiarity with the Vivado design environment and user interface.
Create and simulate simple RTL designs using VHDL and was able to simulate them.

Experiment – 18

Aim: Write a program in VHDL for the implementation of full subtractor using structural modelling.

Software Used: Xilinx Vivado 2020.1

Truth Table:

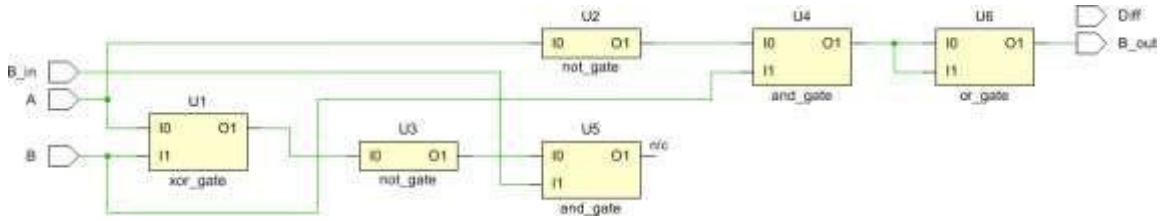
A	B	B_{in}	Diff	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Program:

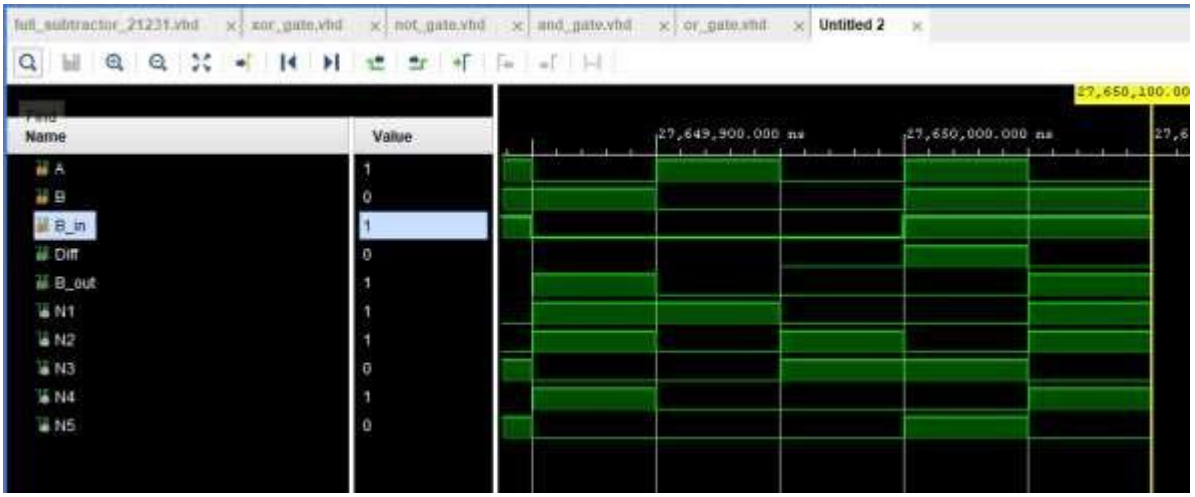
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity
   full_subtractor_21231 is
   Port ( A : in
   STD_LOGIC;
         B : in STD_LOGIC;
         B_in : in STD_LOGIC;
         Diff : out STD_LOGIC;
         B_out : out
         STD_LOGIC);
end full_subtractor_21231;
architecture Structural of full_subtractor_21231 is
component xor_gate
port(I0, I1 : in std_logic; O1 : out
std_logic); end component;
component and_gate
port(I0, I1 : in std_logic; O1 : out
std_logic); end component;
component or_gate
port(I0, I1 : in std_logic; O1 : out
std_logic); end component;
component not_gate
port(I0 :in std_logic; O1 : out
std_logic); end component;
signal N1, N2, N3, N4, N5 : std_logic;
begin
U1 : xor_gate port map(A, B,
N1); U2 : not_gate port
```

map(A, N2); U3 : not_gate
port map(N1, N3);
U4 : and_gate port map(N2, B,
N4); U5 : and_gate port map(N3,
B_in, N5); U6 : or_gate port
map(N4, N4, B_out);

Schematic Result:



RTL Result



Learning Outcome:

- Understood the basic concepts of VHDL design.
- Familiarity with the Vivado design environment and user interface.
- Create and simulate simple RTL designs using VHDL and was able to simulate them.

Experiment – 19

Aim: Write a program in VHDL for the implementation of 4x1 mux using structural modelling.

Software Used: Xilinx Vivado 2020.1

Truth Table:

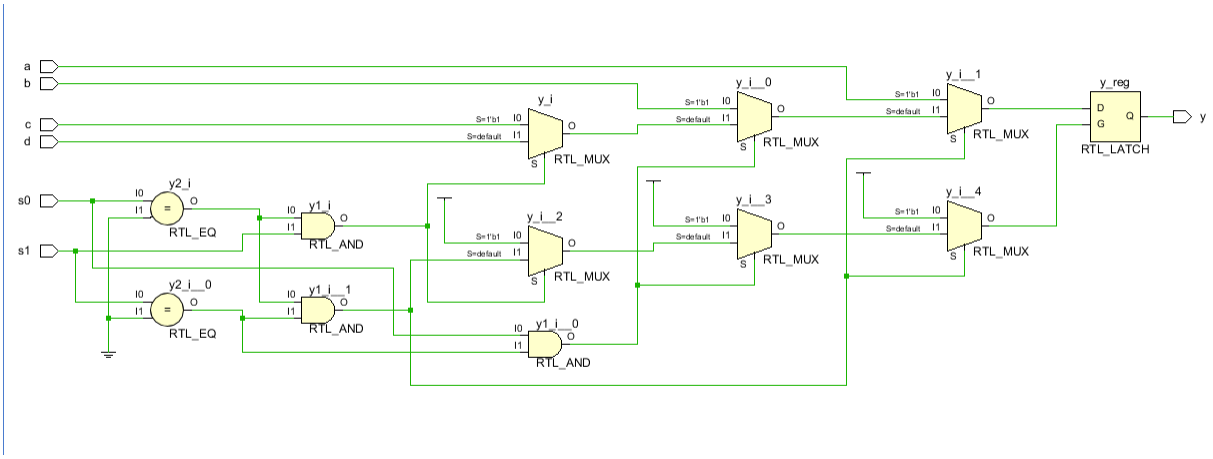
Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Program:

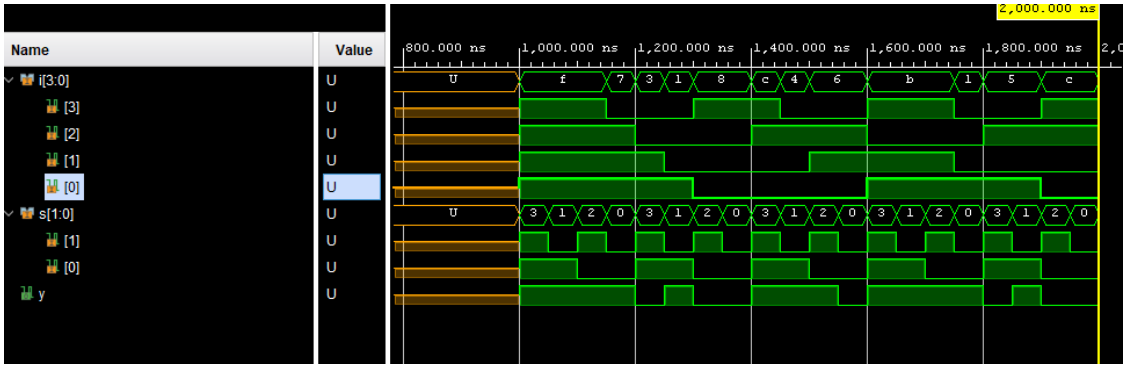
```
entity MUX4_1 is
port ( Sel0,Sel1 : in std_logic;
A, B, C, D : in std_logic;
Y : out std_logic );
end MUX4_1;
architecture structural of MUX4_1 is
component inv
port (pin : in std_logic;
pout :out std_logic)
end component;
component and3
port (a0,a1,a2: in std_logic;
aout:out std_logic);
end component;
component or4
port (r0,r1,r2,r3:in std_logic;
rout:out std_logic);
end component;
signal selbar0,selbar1,t1,t2,t3,t4: std_logic;
```

```
begin
INV0: inv port map (Sel0, selbar1); INV1:
inv port map (Sel1, selbar1);
A1: and3 port map (A, selbar0, selbar1, t1); A2:
and3 port map (B, Sel0, selbar1, t2); A3: and3
port map (C, selbar0, Sel1, t2); A4: and3 port map
(D, Sel0, Sel1, t4);
O1: or4 port map (t1, t2, t3, t4, Y); end
structural;
```

RTL Schematics:



Waveform:

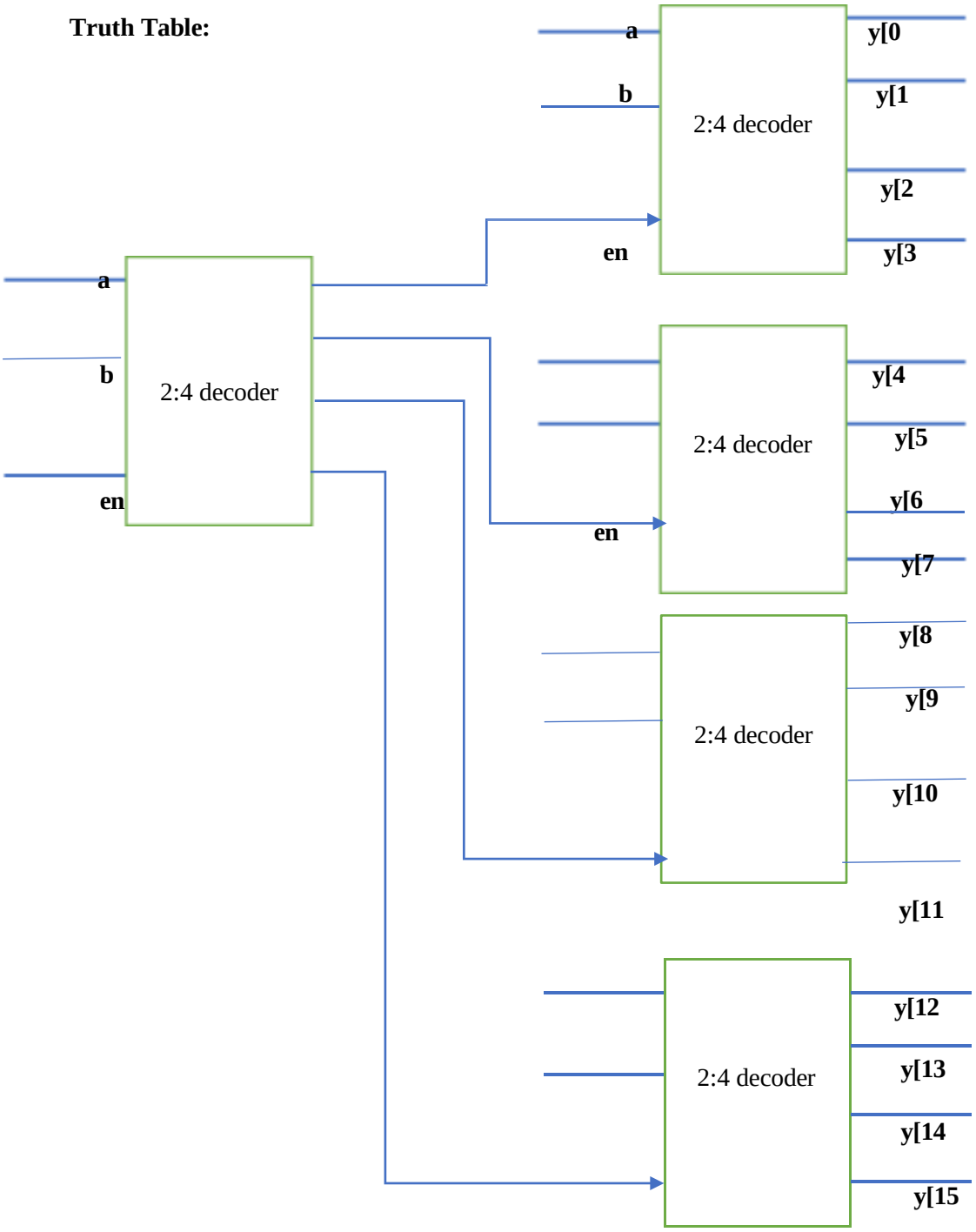


Learning Outcome: Understood the basic concepts of VHDL design.
Familiarity with the Vivado design environment and user interface.
Create and simulate simple RTL designs using VHDL and was able to simulate them.

Experiment - 20

Aim: Write a program in VHDL for the implementation of 4x16 decoder using 2x4 mux structural modelling.
Software Required: Xilinx Vivado 2020.1
Theory: To create a 4x16 decoder using 2x4 multiplexers (mux), you will need to use four 2x4 mux‟s in a cascading arrangement. Each 2x4 mux will have two select lines and four data inputs. The outputs of these mux‟s will form the 16 outputs of the decoder.

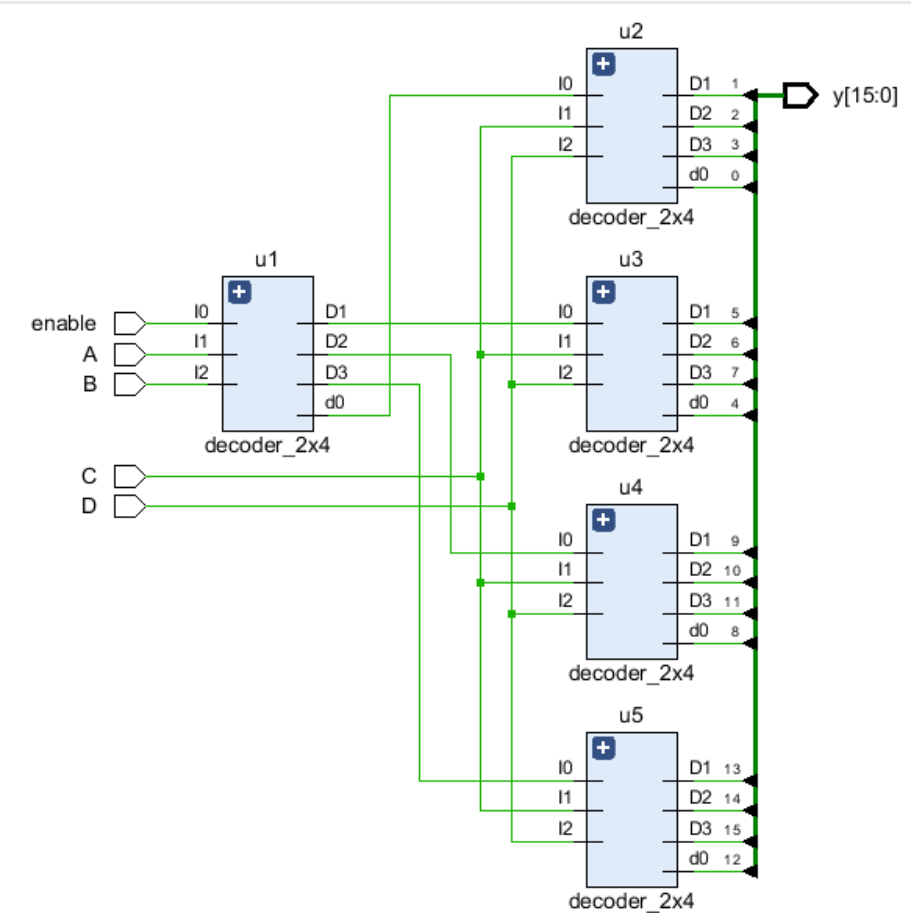
Truth Table:



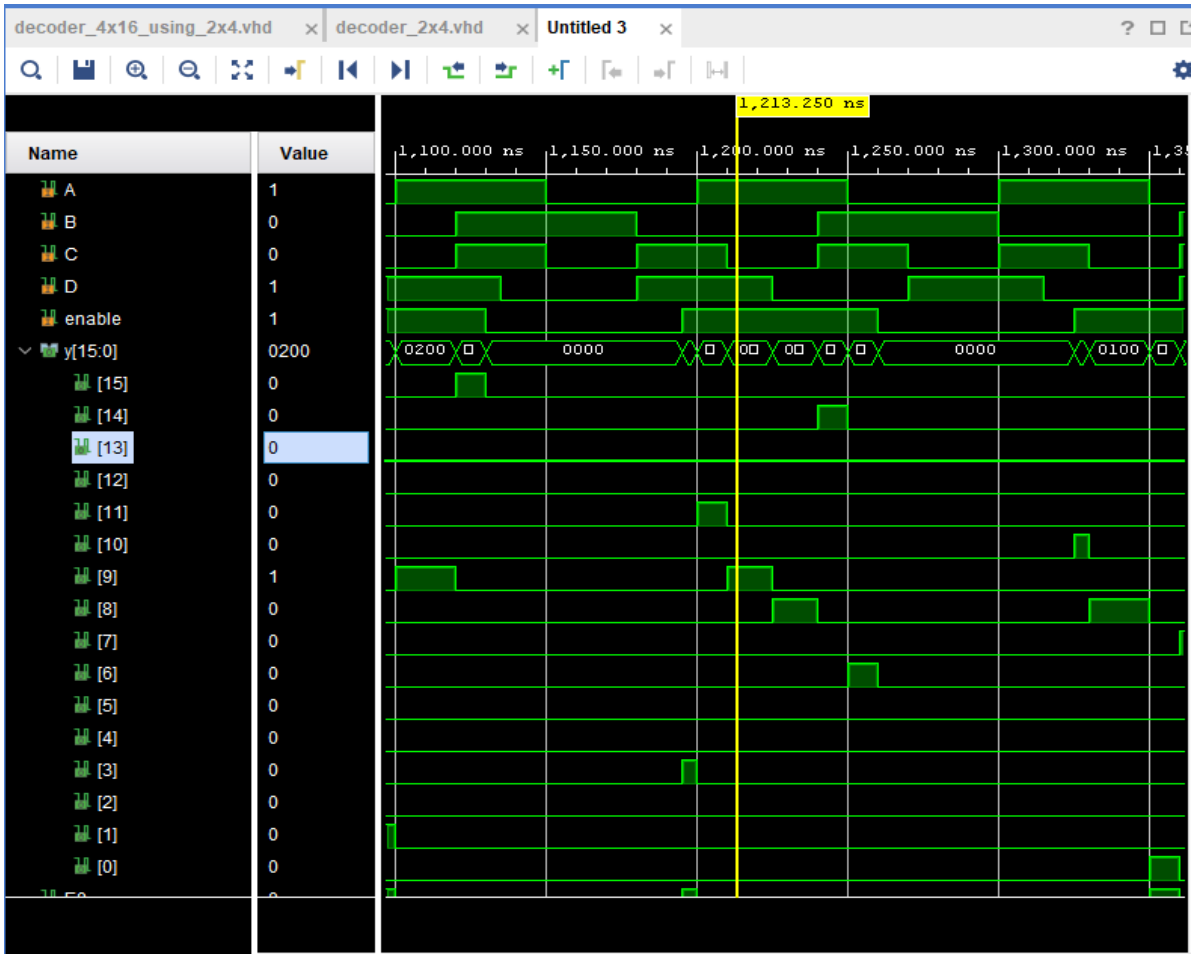
Coding and Output:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity decoder_4x16use_2x4_21231
   is Port ( a,b,c,d,enable : in
   STD_LOGIC;
         Y : out std_logic(15 downto 0));
end decoder_4x16use_2x4_21231;

architecture Structural of
decoder_4x16use_2x4_21231 is component
decoder_2x4
port(I0, I1, I2, I3 : in std_logic; D0, D1, D2, D3 : out
std_logic); end component;
signal N1, N2, N3, N4 :
std_logic; begin
U1 : decoder_2x4 port map(„1", a, b, N1, N2, N3, N4);
U2 : decoder_2x4 port map(N1, c, d, Y(0), Y(1), Y(2), Y(3));
U3 : decoder_2x4 port map(N2, c, d, Y(4), Y(5), Y(6), Y(7));
U4 : decoder_2x4 port map(N3, c, d, Y(8), Y(9), Y(10), Y(11));
U5 : decoder_2x4 port map(N4, c, d, Y(11), Y(12), Y(13),

Y(15)); end Structural;
```

Schematic Result:



RTL Result



Learning Outcome:

- Understood the basic concepts of VHDL design.
- Familiarity with the Vivado design environment and user interface.
- Create and simulate simple RTL designs using VHDL and was able to simulate them.

Experiment - 21

Objective: Write a program in VHDL for the implementation of SR Flipflop structuralmodelling.
Software Required: Xilinx Vivado 2020.1
Theory: A Set-Reset (SR) flip-flop is a simple sequential logic circuit with two inputs (S for Set and R for Reset) and two outputs (Q and Q'). The flip-flop changes its output state based on the inputs, and it can be used for various applications in digital electronics.
Set (S) Input: When the Set input (S) is high (1), it forces the Q output to be set to 1 (Q = 1) regardless of the current state of the flip-flop.
Reset (R) Input: When the Reset input (R) is high (1), it forces the Q output to be reset to 0 (Q = 0) regardless of the current state of the flip-flop.
Q and Q' Outputs: The Q output represents the current state of the flip-flop, and Q' is the complement of Q, meaning it is the inverse of Q. When Q is high (1), Q' is low (0), and vice versa.

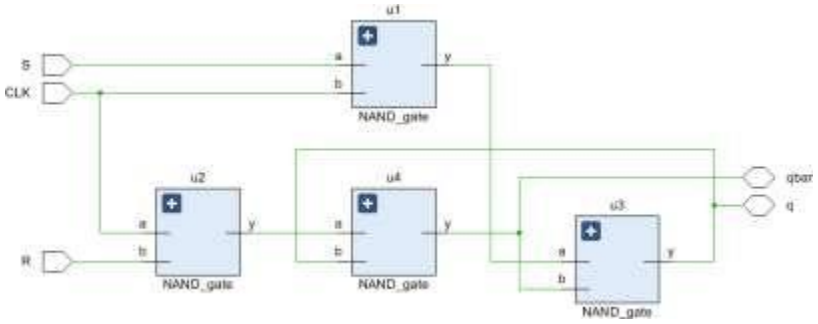
Truth Table:

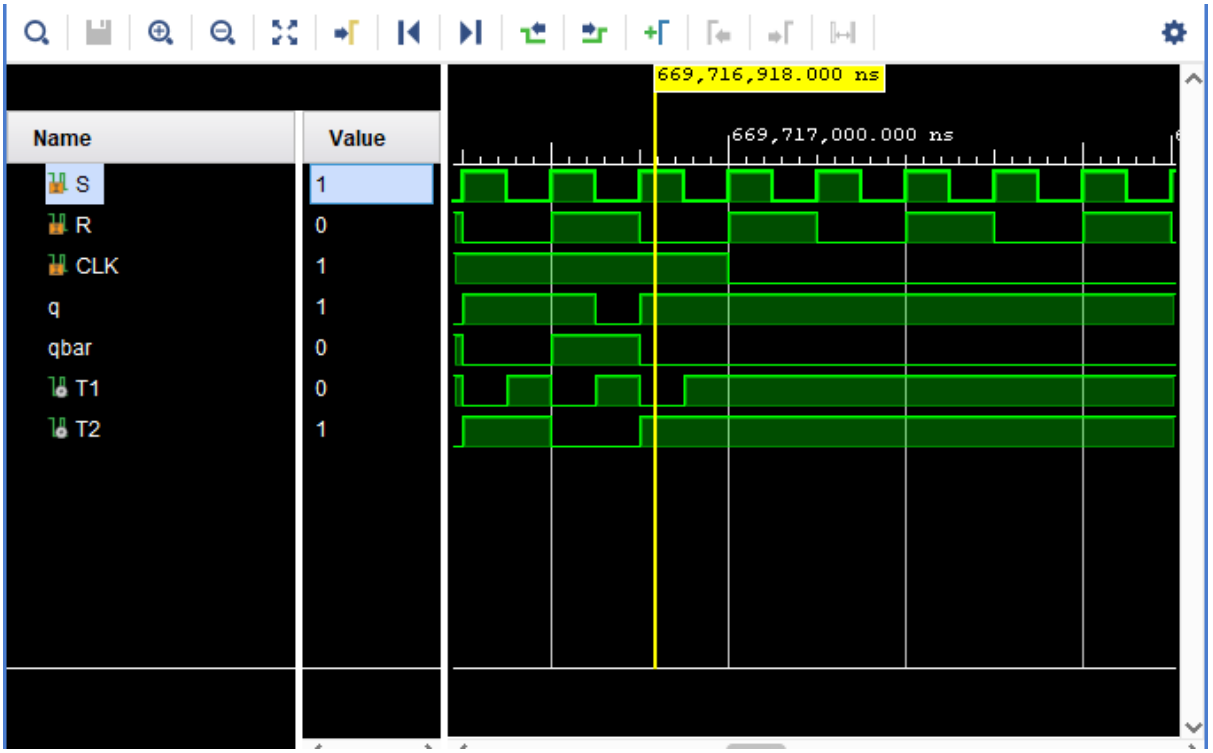
S	R	Q	Q’
0	0	0	Q‟
0	1	0	1
1	0	1	0
1	1	X	X

Coding and Output:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity SR_FF_STR_21231
is Port (S, R, CLK: IN
std_logic; q, qbar: inout
std_logic );
end SR_FF_STR_21231;
architecture structural of SR_FF_STR_21231 is
component NAND_gate
port(a, b:in std_logic; y:OUT
std_logic); end component;
signal T1,T2:
std_logic; begin
ul:NAND_gate port map (S, CLK,
T1) ; u2:NAND_gate port map
(CLK, R, T2) ; u3:NAND_gate port
map (T1, qbar, q) ; u4:NAND_gate
port map (T2, q, qbar) ; end
structural:
```

Schematic Result:





Learning Outcome:

- Understood the basic concepts of VHDL design.
- Familiarity with the Vivado design environment and user interface.
- Create and simulate simple RTL designs using VHDL and was able to simulate them.

Experiment - 22

Aim: Write a program in VHDL for the implementation of and gate using test bench for structural modelling.

Software Required: Xilinx Vivado 2020.1

Theory: A full adder is an extension of the half adder that takes into account not only the two input bits (A and B) but also an additional carry input (Cin) from a previous addition.

Truth Table:

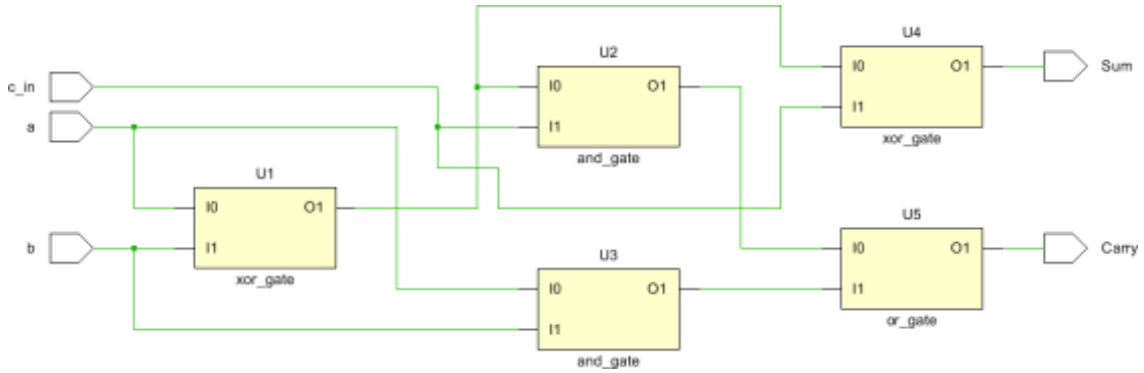
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Coding and Output:

```
library IEEE;
use IEEE.STD LOGIC_1164.ALL;
entity tb_and_gate
is end
tb_and_gate;
architecture Behavioral of
tb_and_gate is component
and_gate_21231_tb is
port (A, B:in
std_logic; C:out
std_logic);
'end component;
signal a: std_logic :=
'0'; signal b: std_logic
:= '0'; signal c:
std_logic; begin
uut: and_gate_21231_tb port map(a=>A, b=>B, c=>C);
:-- stimulus
process;
stim_proc:process
begin
wait for 10 ns;
a <= '1';
b <= '0';
wait for 10 ns;
a <= '0';
b <= '1';
wait for 10 ns;
a <= '0';
b <= '0';
wait for 10 ns;
;a <= '1';
b <= 'l';
wait for 10ns;
end process;
end
```

Behavioral;

Schematic Result:



Learning Outcome:

- Understood the basic concepts of VHDL design.
- Familiarity with the Vivado design environment and user interface.
- Create and simulate simple RTL designs using VHDL and was able to simulate them.

Experiment - 23

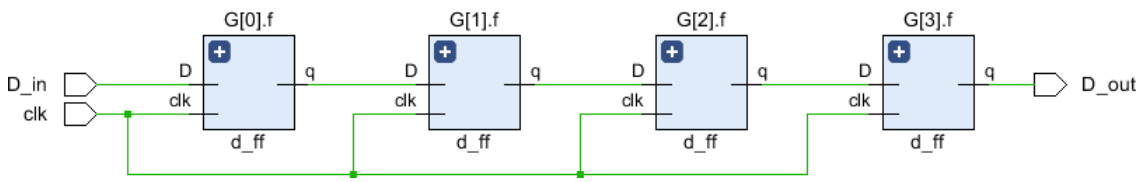
Aim: Write a program in VHDL for the implementation of and Siso register using generate test bench for structural modelling.
Software Required: Xilinx Vivado 2020.1
Theory: A full adder is an extension of the half adder that takes into account not only the two input bits (A and B) but also an additional carry input (Cin) from a previous addition.
Truth Table:

CLK	Q3	Q2	Q1	Q0
Initially	0	0	0	0
1st falling edge	1	0	0	0
2nd falling edge	0	1	0	0
3rd falling edge	1	1	1	0
4th falling edge	1	1	1	1

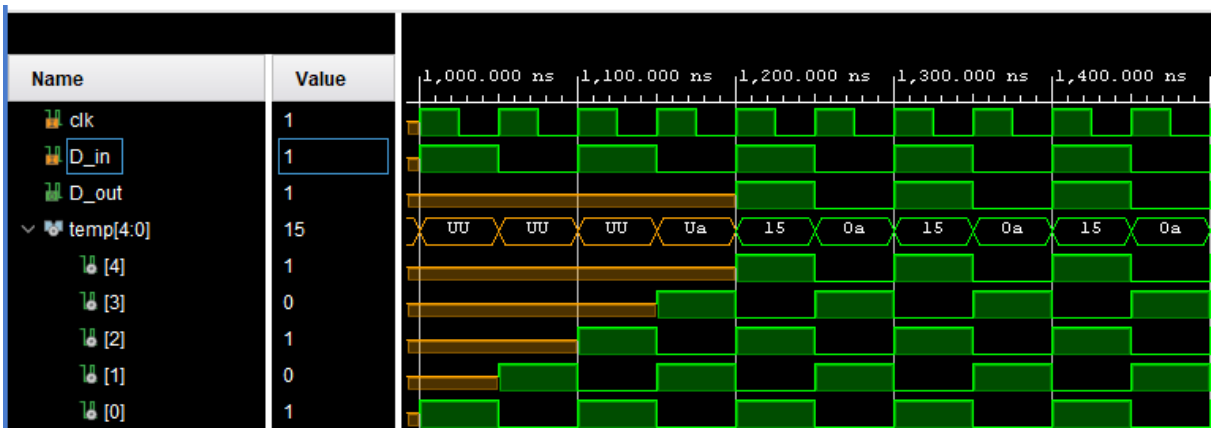
Coding and Output:

```
'library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity siso_generate is
Port (clk, D_in: in
std_logic; D_out: out
std_logic);
end siso_generate;
architecture structural of siso_generate
is 'component d_ff
port (D, clk:in
std_logic; g: out
std_logic);
'end component;
signal temp: std_logic_vector(4 downto 0);
begin
temp(0) <= d_in;
'G:for i in 0 to 3 generate
f: d_ff port map(temp(i), clk, temp
(i+1) ); end generate G;
D_out <= temp
(4); end
structural;
```

Schematic Result:



RTL Result:



Learning Outcome:

- Understood the basic concepts of VHDL design.
- Familiarity with the Vivado design environment and user interface.
- Create and simulate simple RTL designs using VHDL and was able to simulate them.

Experiment - 24

Aim: Write a program in VHDL for the implementation of and Mod 10 counter using generate test bench for structural modelling.
Software Required: Xilinx Vivado 2020.1
Theory: A full adder is an extension of the half adder that takes into account not only the two input bits (A and B) but also an additional carry input (Cin) from a previous addition.
Truth Table:

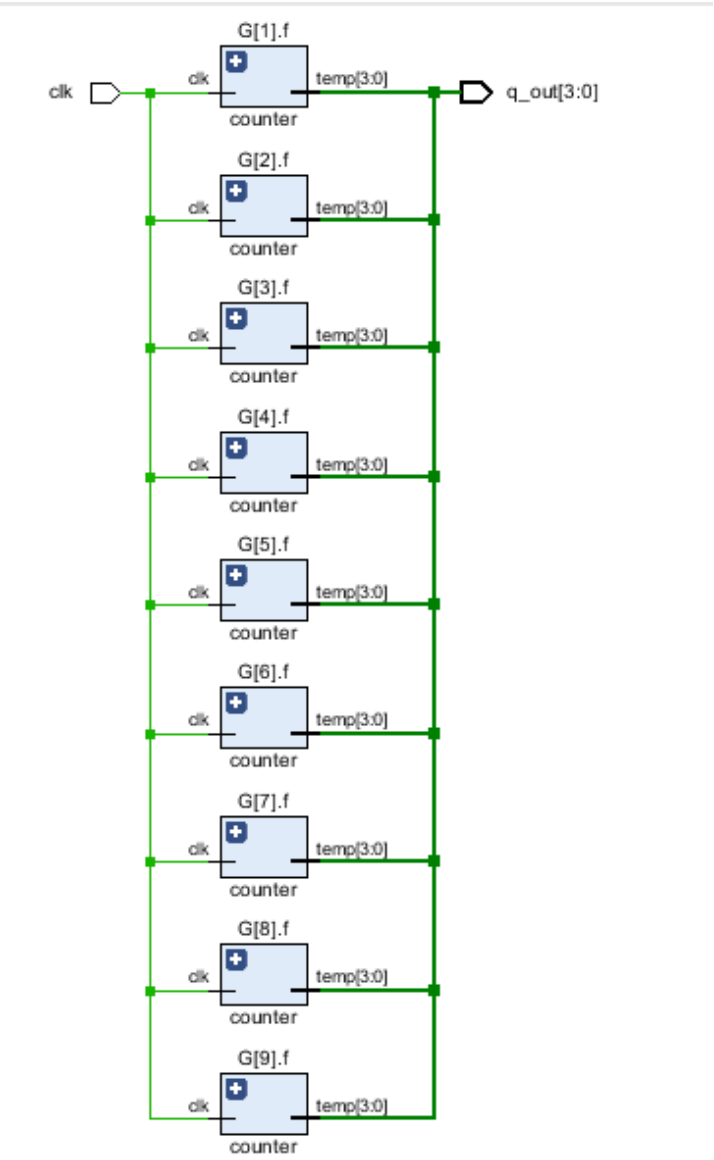
Present State (Q2 Q1 Q0)	Next State (Q2 Q1 Q0)
000	001
001	010
010	011
011	100
100	101
101	110
110	111
111	000

Coding and Output:

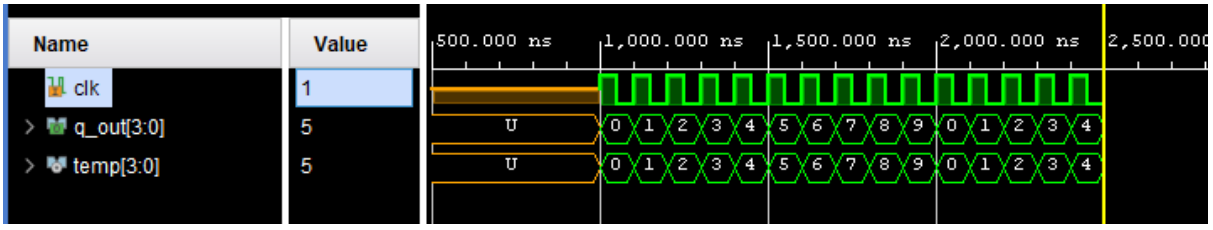
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
| entity
MOD_10_counter_Generate is
Port (clk: std_logic;
q_out:out std_logic_vector (3 downto 0 ));
| end MOD_10_counter_Generate;
architecture Behavioral of
MOD_10_counter_Generate is I component counter
port(clk:in std_logic;
temp: inout std_logic_vector);
| end component;
signal temp: std_logic_vector (3 downto
0); begin
G:for i in 1 to 9 generate
f: counter port map (clk, temp)
; end generate G;
q_out <=
temp; end
Behavioral;
-m.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;
-- Uncomment the following library declaration if
using ... entity counter is
Port (clk: std_logic;
temp: inout std_logic_vector(3 downto
0));
end counter;
```

```
Architecture Behavioral of counter
is begin
process(clk
) begin

if (clk' event and
clk='1') then
if (temp <"1001") then
temp <= conv_std_logic_vector(conv_integer (temp) +1,
4); else
temp <=
"0000"; end if;
end if;
end process;
end
Behavioral;
```

Schematic Result:



RTL Result:



Learning Outcome:

- Understood the basic concepts of VHDL design.
- Familiarity with the Vivado design environment and user interface.
- Create and simulate simple RTL designs using VHDL and was able to simulate them.

Experiment – 25

Aim: Perform FPGA burning of and gate with Basys3.

Software Used: Xilinx Vivado.2020.1

Program:

1. **Design Source File**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity and_gate_basys is
Port (a: in STD_LOGIC;
b: in STD_LOGIC;
c: out STD_LOGIC);
end and gate_basys;
architecture Behavioral of and_gate_basys is
begin
c<= a and b;
end Behavioral;
```

2. **Constrain File**

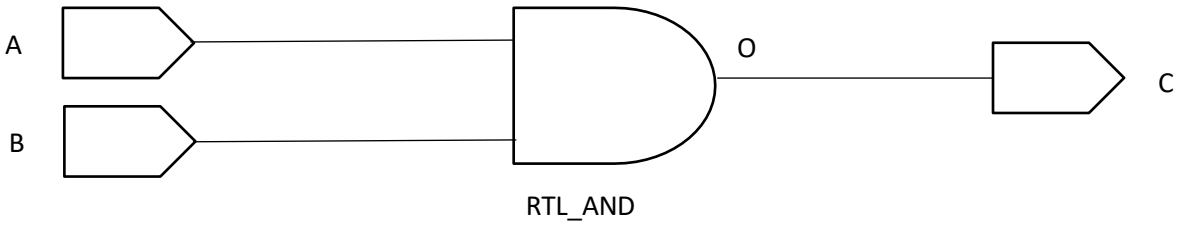
```
set property PACKAGE PIN V17 [get_ports {a}]
set property 1OSTANDARD LVCMOS33 [get ports (a)]
set_property PACKAGE_PIN V16 [get_ports (b)]
set_property IOSTANDARD LVCMOS33 [get_ports (b)]
set property PACKAGE_PIN U16 [get_ports (c)]
set_property IOSTANDARD LVCMOS33 [get_ports (e)]
```

3. **Simulation File**

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity and tb is

Port ();

end and tb;

architecture Behavioral of and tb is

--Component name and entity's name must be same

--ports must be same

component and gate_basys is

Port (A,B:in std_logic;

C: out std_logic);

end component;

--inputs
```

```
signal a: std_logic:= '0';
signal b: std_logic:= '0';
--outputs
signal c: std_logic;
begin
uut: and_gate_basys PORT MAP(a=>A,b=>B,c=>C);
--Stimulus Process
stim_proc:process
begin
wait for 10ns;
a<='1';
b<='0';
wait for 10ns;
a<='0';
b<='1';
wait for 10ns; end process;
a<='0';
b<='0';
wait for 10ns;
a<='1';
b<='1';
wait for 10ns;
end Behavioral;
```

RTL Schematics:



Learning Outcome: Here we can burn and gate on FPGA with Basys3 on Vivado with help of the same basic commands of VHDL.

Experiment – 26

Aim: Perfrom FPGA burning of half adder with Basys3.

Software Used: Xilinx Vivado.2020.1

Program:

1. **Design Source File**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity HA_basys is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           sum : out STD_LOGIC;
           cout : out STD_LOGIC);
end HA_basys;

architecture Dataflow of HA_basys is

begin
sum <= a XOR b;
cout <= a and b;

end Dataflow;
```

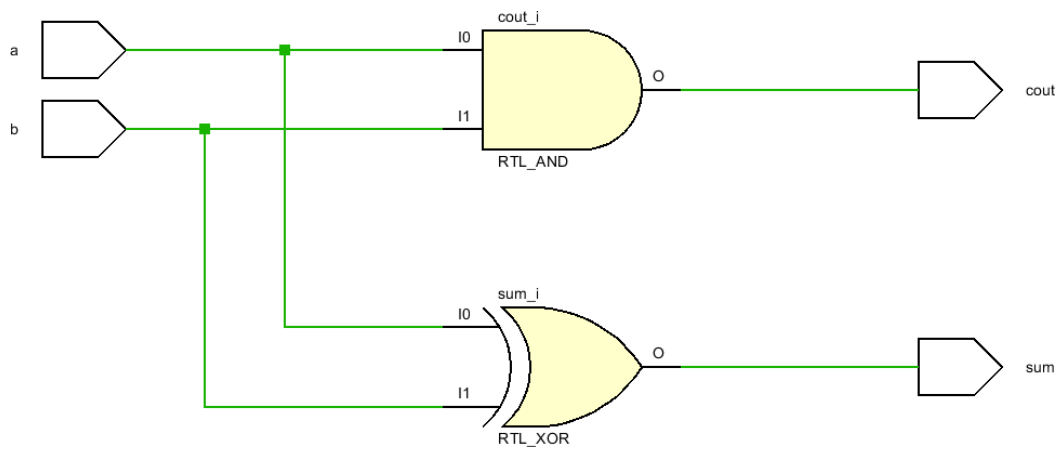
2. **Constraint File**

```
set_property PACKAGE_PIN V17 [get_ports {a}]
set_property IOSTANDARD LVCMOS33 [get_ports {a}]
set_property PACKAGE_PIN V16 [get_ports {b}]
set_property IOSTANDARD LVCMOS33 [get_ports {b}]

set_property PACKAGE_PIN U16 [get_ports {sum}]
set_property IOSTANDARD LVCMOS33 [get_ports {sum}]
set_property PACKAGE_PIN E19 [get_ports {cout}]
set_property IOSTANDARD LVCMOS33 [get_ports {cout}]
```

3. **Simulation File**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity HA_tb is
--  Port ( );
end HA_tb;

architecture Behavioral of HA_tb is
--Component name and entity's name must be same
--ports must be same
 component andgate is
  Port (A,B:in std_logic;
  C: out std_logic );
end component;
--inputs
signal a: std_logic:= '0';
signal b: std_logic:= '0';
--outputs
signal c : std_logic;

begin
uut: andgate PORT MAP(a=>A,b=>B,c=>C);
--Stimulus Process
stim_proc:process
begin
wait for 10ns;
a<='1';
b<='0';
wait for 10ns;
a<='0';
b<='1';
wait for 10ns;
a<='0';
b<='0';
wait for 10ns;
a<='1';
b<='1';
wait for 10ns;
end process;
end Behavioral;
```

RTL Schematics:



Learning Outcome: Understood the basic concepts of SoC design.
Familiarity with the Vivado design environment and user interface.
Create and simulate simple RTL designs using VHDL and was able to simulate them