## 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

|  |  |  |
| --- | --- | --- |
| **Sr.No.** | **Title** | **Signature** |
| 1 | Implementation of logic gates |  |
| 2 | Write a program in VHDL for the implementation of half-adder and full adder. |  |
| 3 | Write a program in VHDL for the implementation of half-subtractor and full-  subtractor. |  |
| 4 | Implementation of multiplexers |  |
| 5 | Implementation of encoders |  |
| 6 | Implementation of decoders |  |
| 7 | Implementation of SR Flip Flop |  |
| 8 | Implementation ofJK Flip Flop. |  |
| 9 | Implementation of SISO, SIPO, PIPO and PISO registers using behavioral model. |  |
| 10 | Implementation of 4-bit UP and DOWN counter using behavioral model. |  |
| 11 | Implementation of Johnson and Ring counter using behavioral model. |  |
| 12 | Implementation of Half Adder and Full Adder using data flow model |  |
| 13 | Implementation of Half subtractor and Full subtractor using dataflow model. |  |
| 14 | Implementation of 4X1 multiplexer using dataflow model. |  |
| 15 | Implementation of 1X4 demultiplexer using dataflow model. |  |
| 16 | Implementation of 4x2 Encoder using dataflow model. |  |
| 17 | Implementation of Half Adder and Full Adder using structural model. |  |
| 18 | Implementation of Half Subtractor and Full Subtractor using structural model. |  |
| 19 | Implementation of 4X1 multiplexer using structural model. |  |
| 20 | Write a program in VHDL for the implementation of 4x16 decoder using 2x4 mux structural modeling. |  |
| 21 | Write a program in VHDL for the implementation of SR Flip-flop structural modeling. |  |
| 22 | Write a program in VHDL for the implementation of and gate using test bench for structural modeling. |  |
| 23 | Write a program in VHDL for the implementation of and Siso register using generate test bench for structural modeling. |  |
| 24 | Write a program in VHDL for the implementation of and Mod 10 counter using generate test bench for structural |  |
| 25 | Perform FPGA burning of and gate with Basys3. |  |
| 26 | Perform FPGA burning of Half Adder with Basys3. |  |

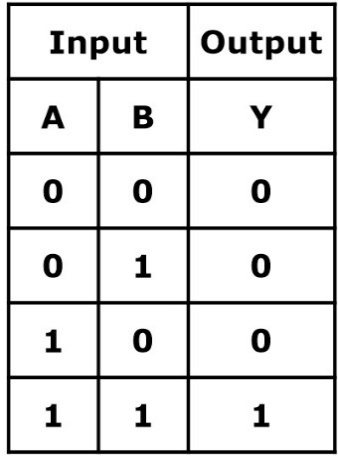
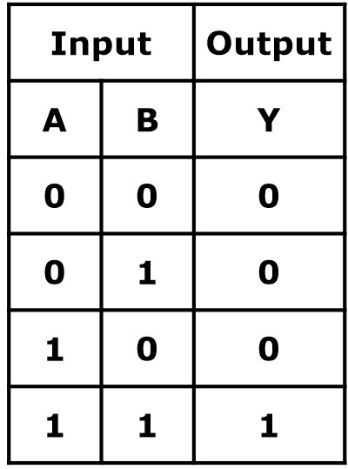
### Experiment – 1

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

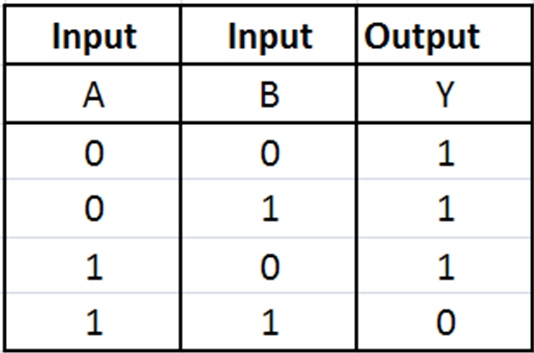
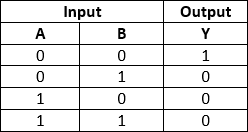
**Software Used:** Xilinx Vivado 2020.1

**Truth Table:**

1. **AND Gate 3. OR Gate**



1. **NOR Gate 4. NAND Gate**



**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:**

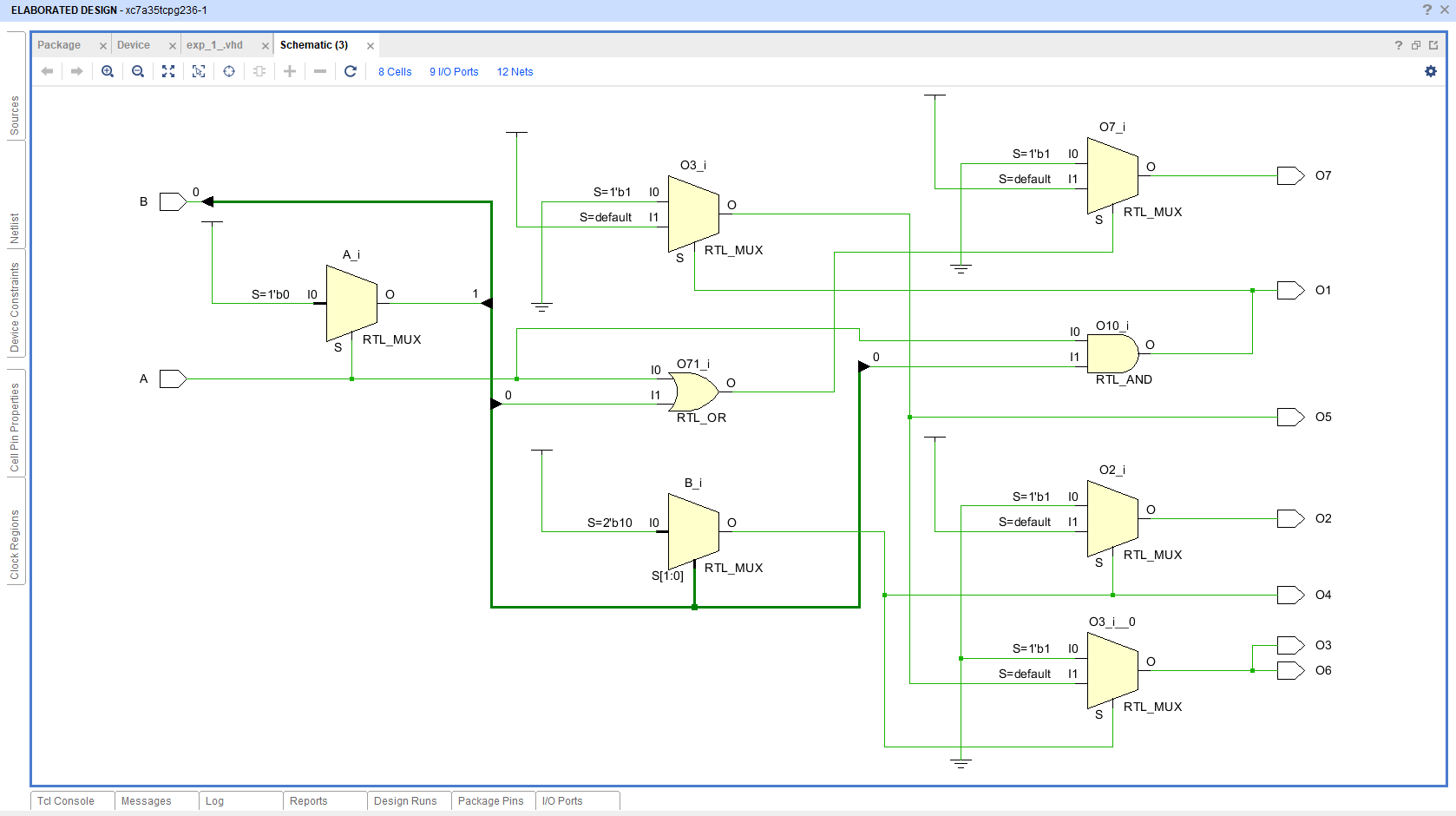


Figure 1 – RTL Schematic Diagram

**Waveform:**

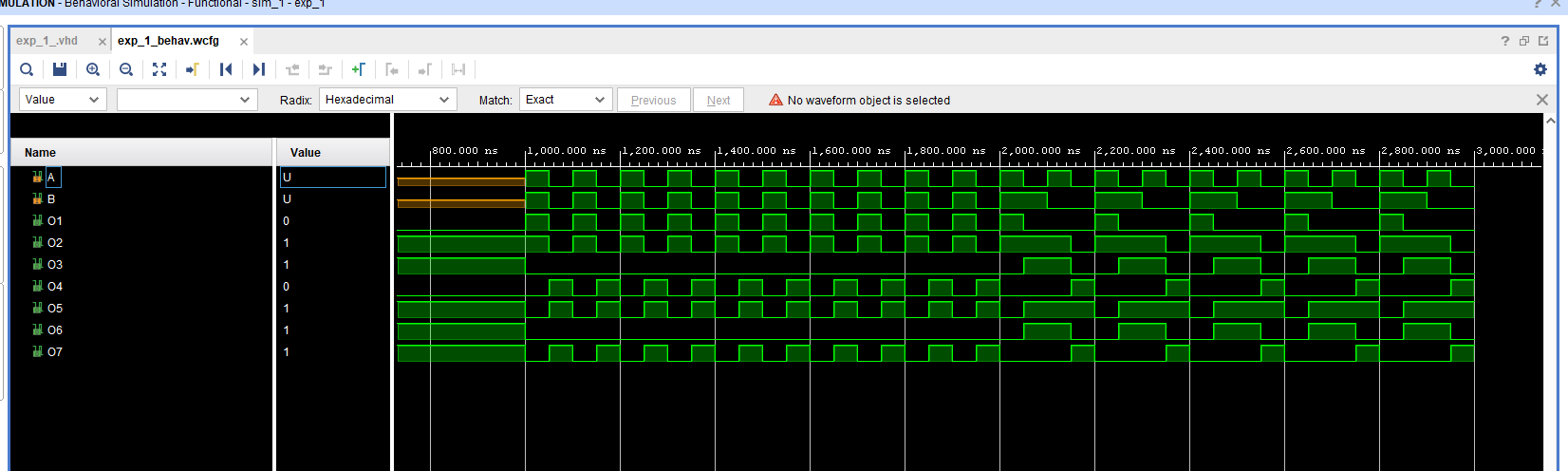


Figure 2 – Waveform Screenshot

**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.

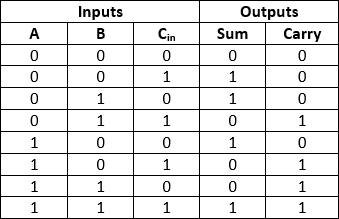
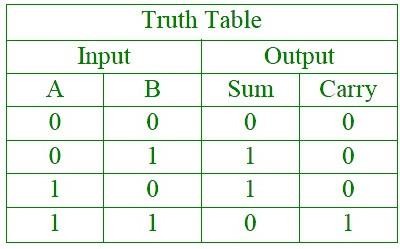
### Experiment – 2

**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: Full-Adder**



**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:**

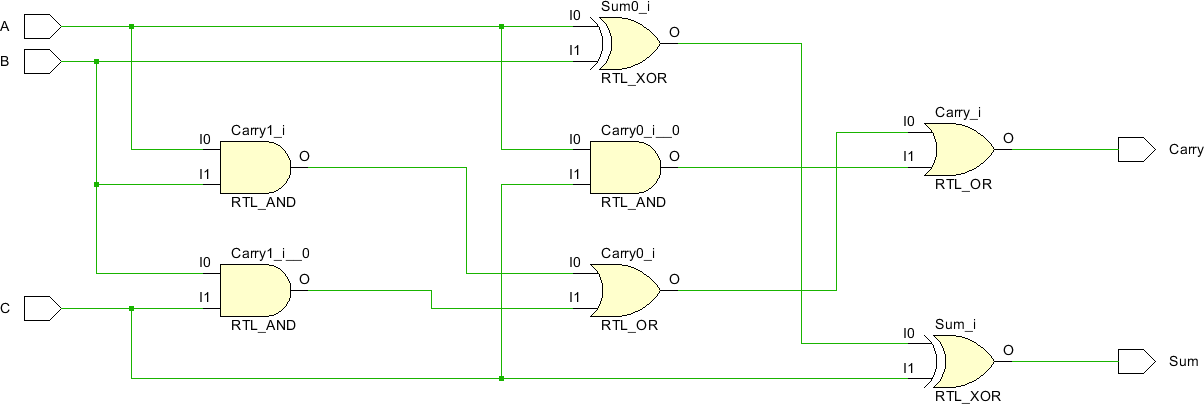


Figure 1 – Schematics Screenshot

**Waveform:**

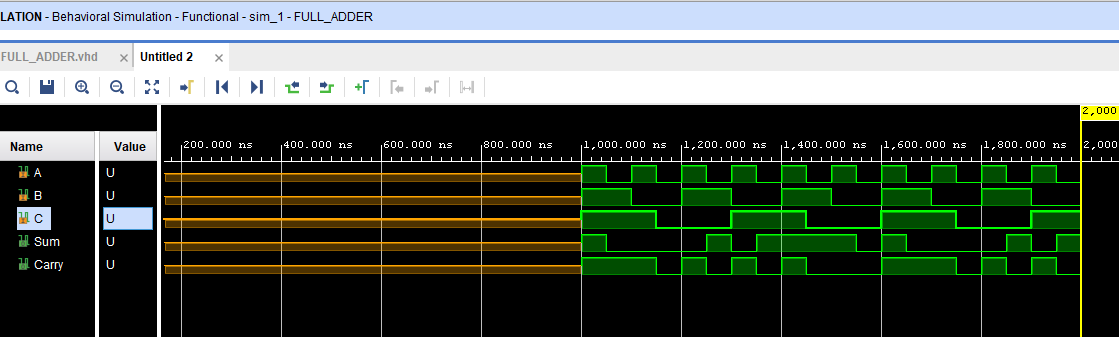


Figure 2 – Waveform Screenshot

**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:**

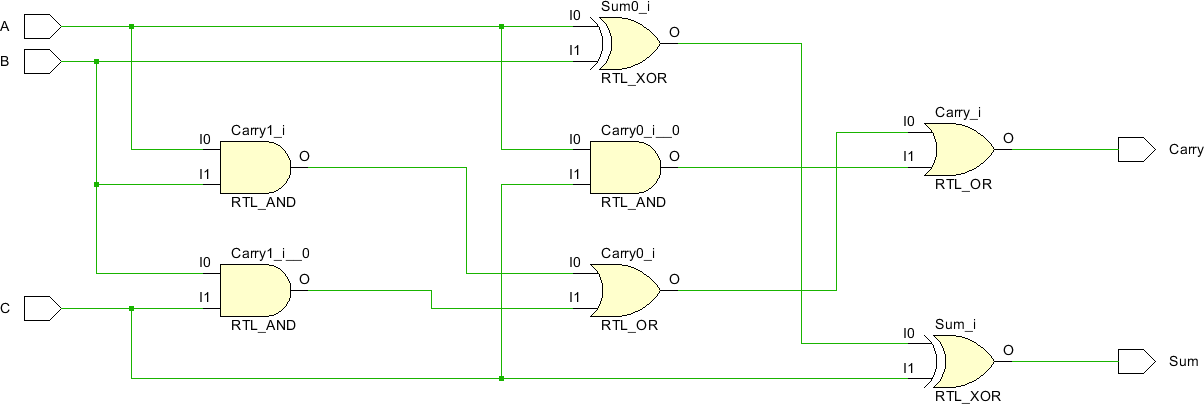


Figure 1 – Schematics Screenshot

**Waveform:**

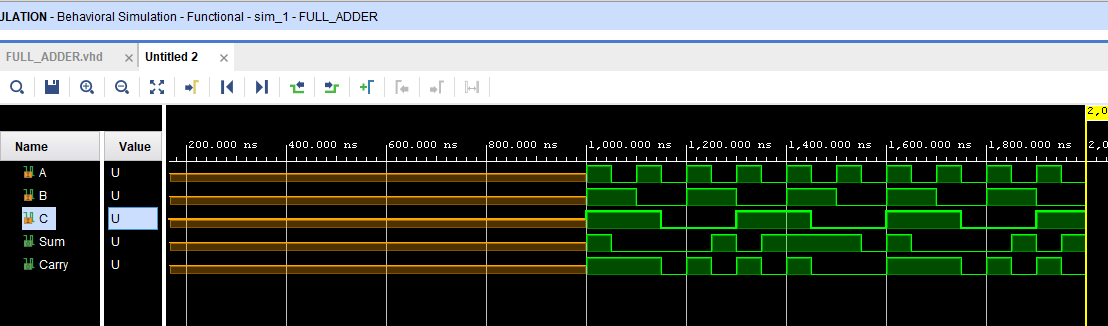


Figure 3 – Waveform Screenshot

**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.

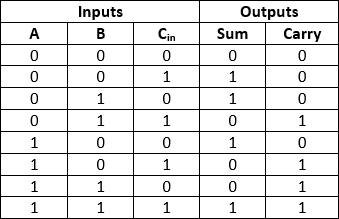
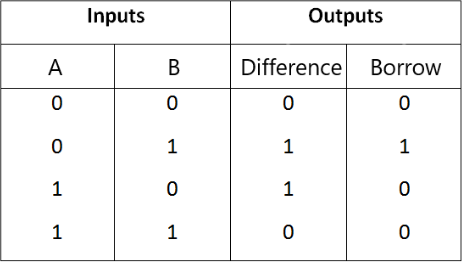
### Experiment – 3

**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: Full-subtractor:



**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:**

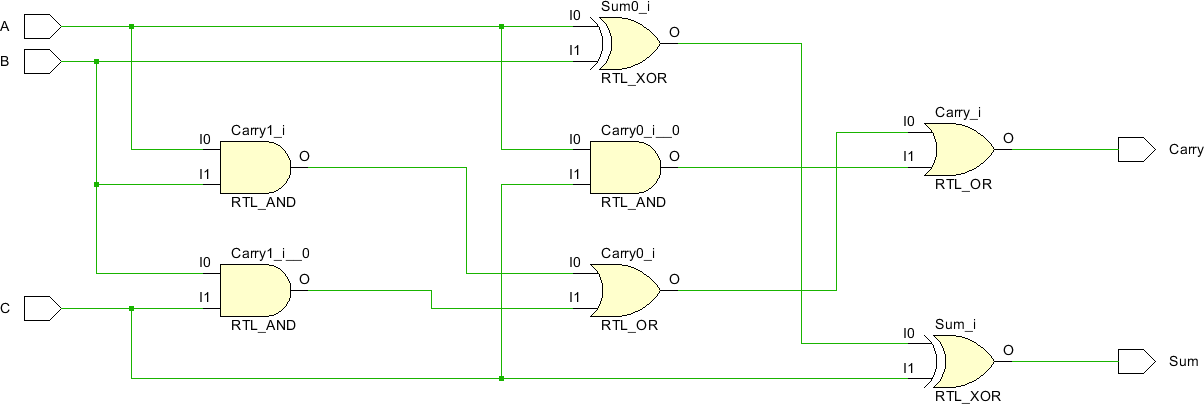


Figure 1 – Schematics Screenshot

**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";

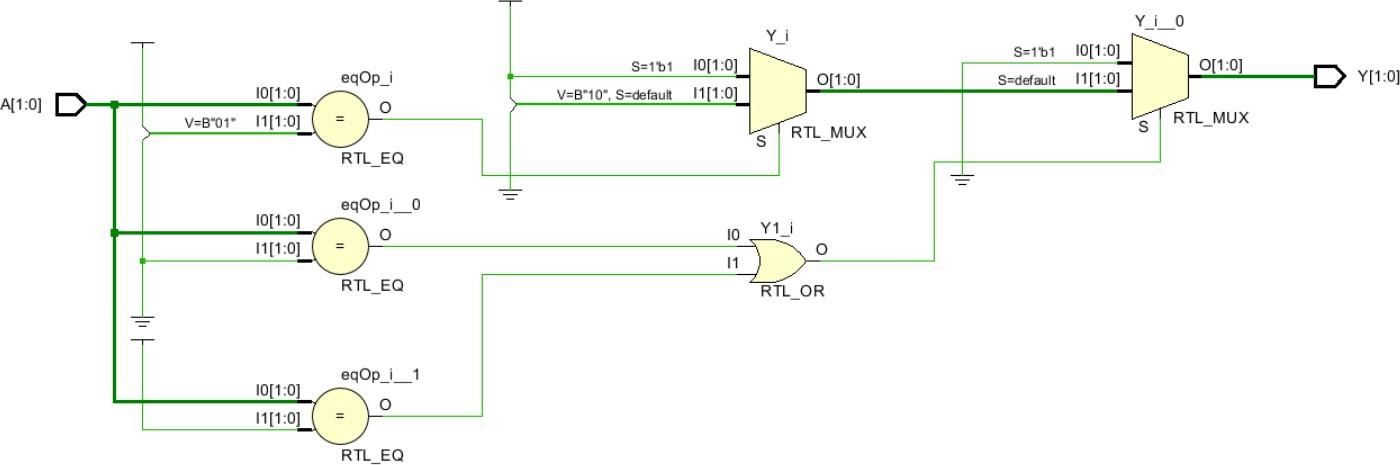
else if (A = "01") then Y<="11";

else Y<="10";

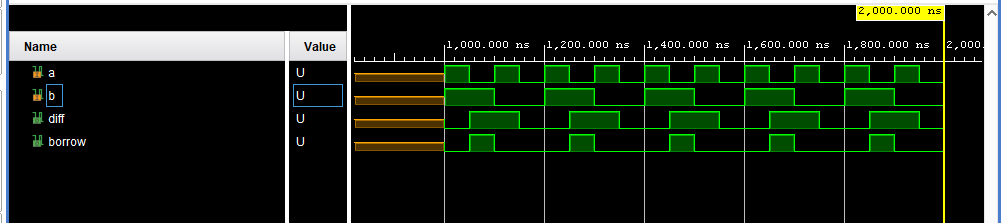
end if; end if;

end process; end Behavioral;

**RTL Schematics:**



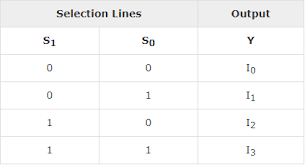
**Waveform:**



**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:**



**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:**

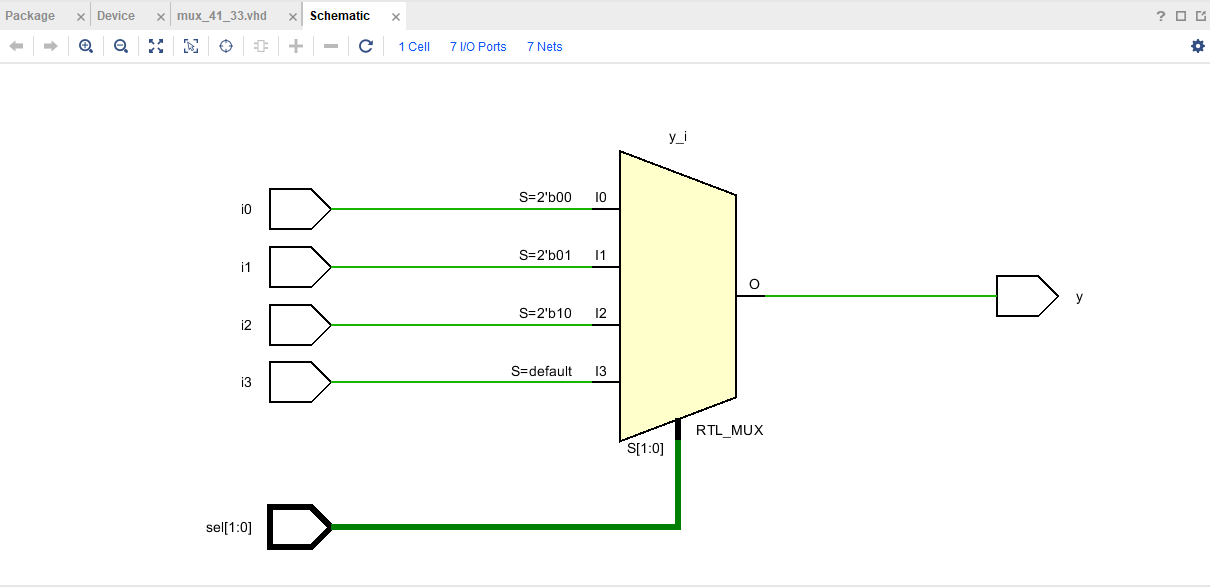


Figure 1 – Schematic Diagram of 4x1 Mux

**Waveform:**

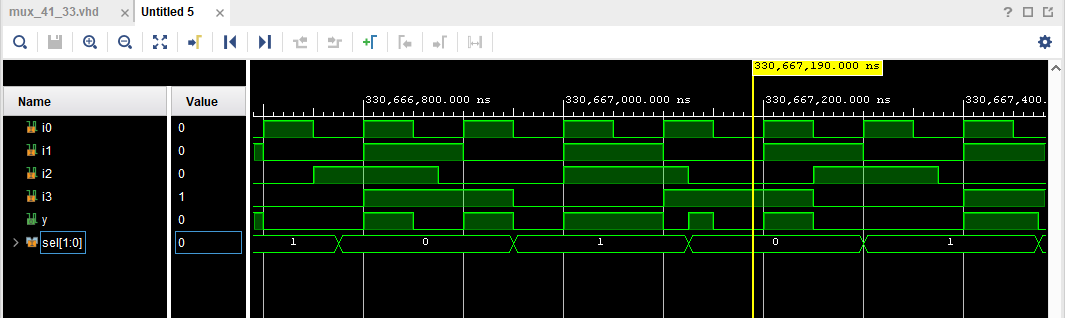


Figure 2 – Waveform of 4x1 Mux

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

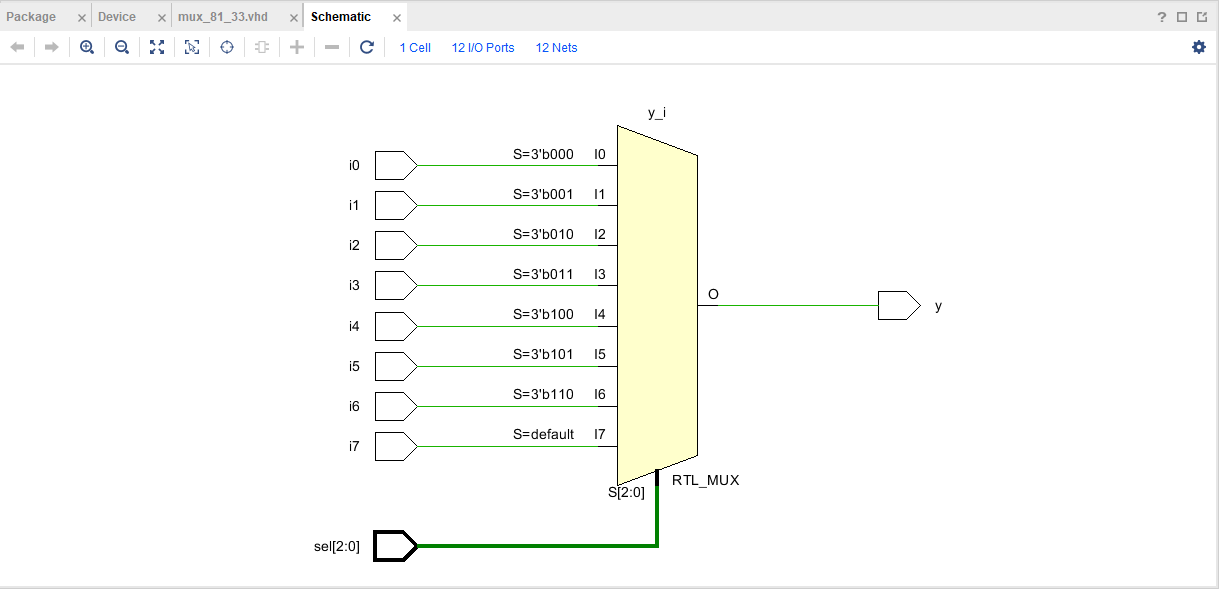
begin process(i0,i1,i2,i3,i4,i5,i6,i7,sel) begin

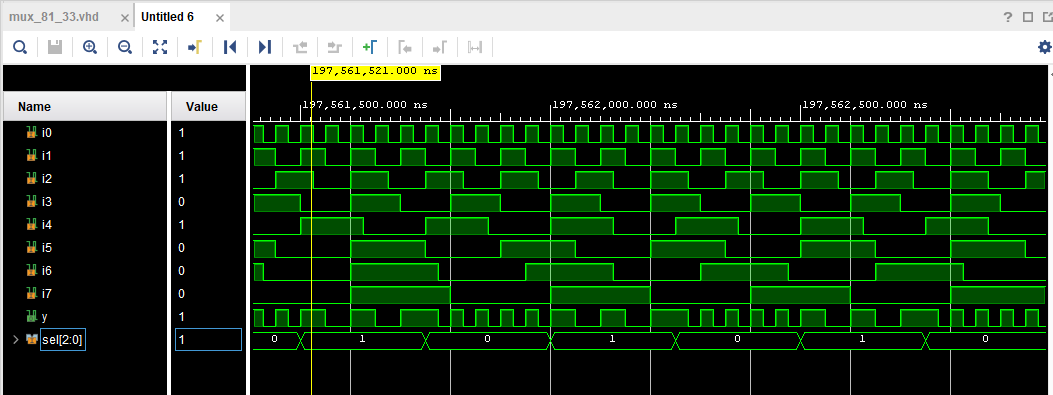
case sel is

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;

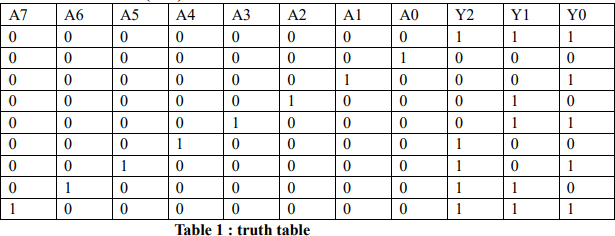




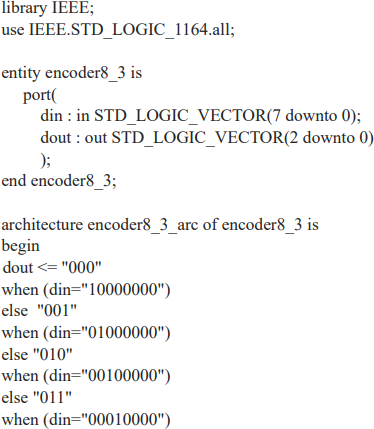
**Learning Outcome:** Implementation of 4x1 and 8:1 mux is successful in Viavdo, Results were also verified.

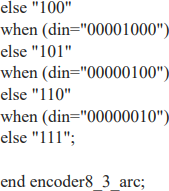
### Experiment – 5

**Aim:** Implementation of encoders **Software Used:** Xilinx Vivado 2020.1 **Truth Table:**



**Program:**





**RTL Schematics:**

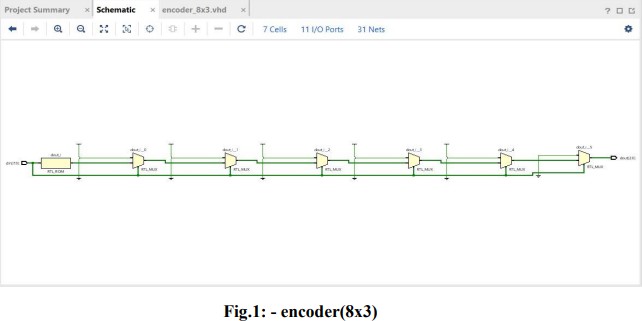


Figure 1 – RTL Schematic

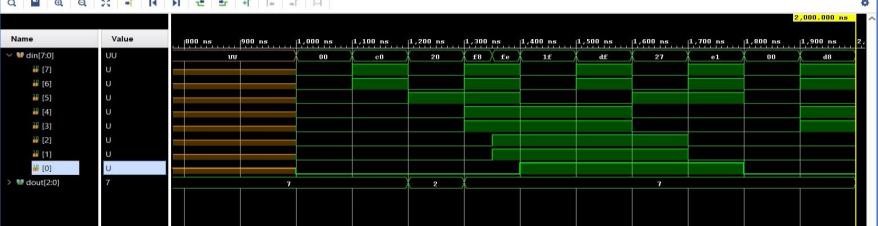
**Waveform:**

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.

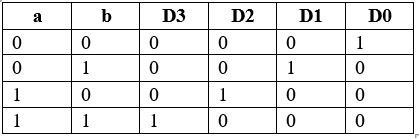
**Software Used:** Xilinx Vivado 2020.1

**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:**



**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”;

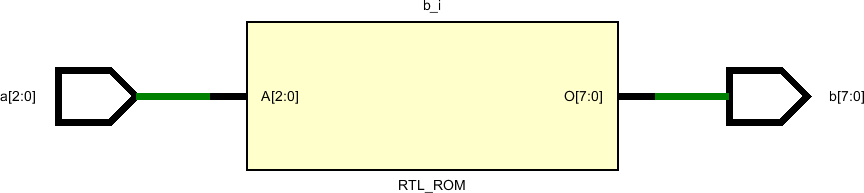
when “110” => d <= “01000000”;

when “111” => d <= “10000000”;

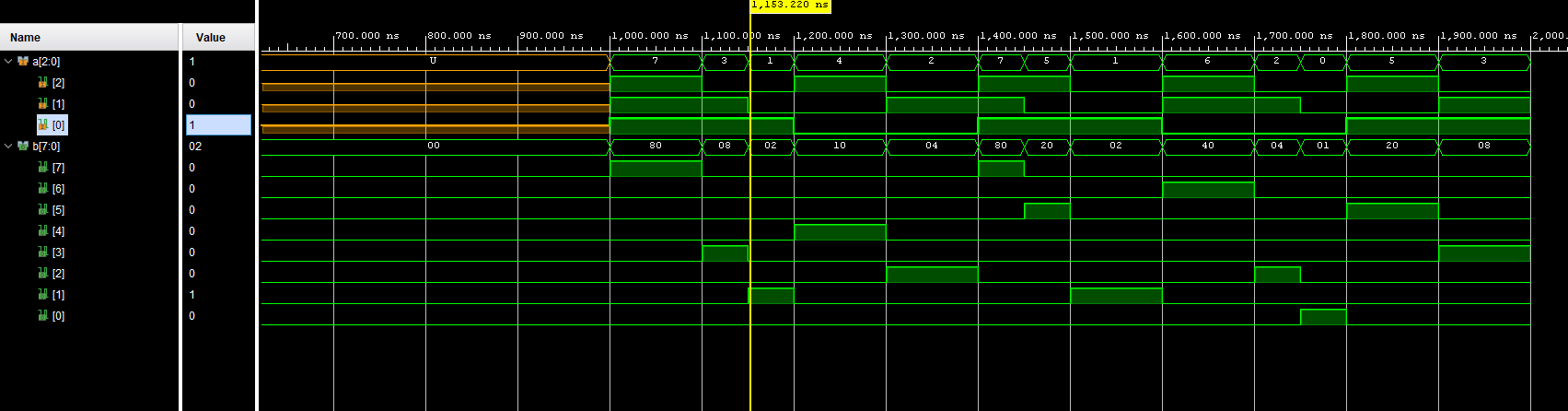
end case; end process;

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”;

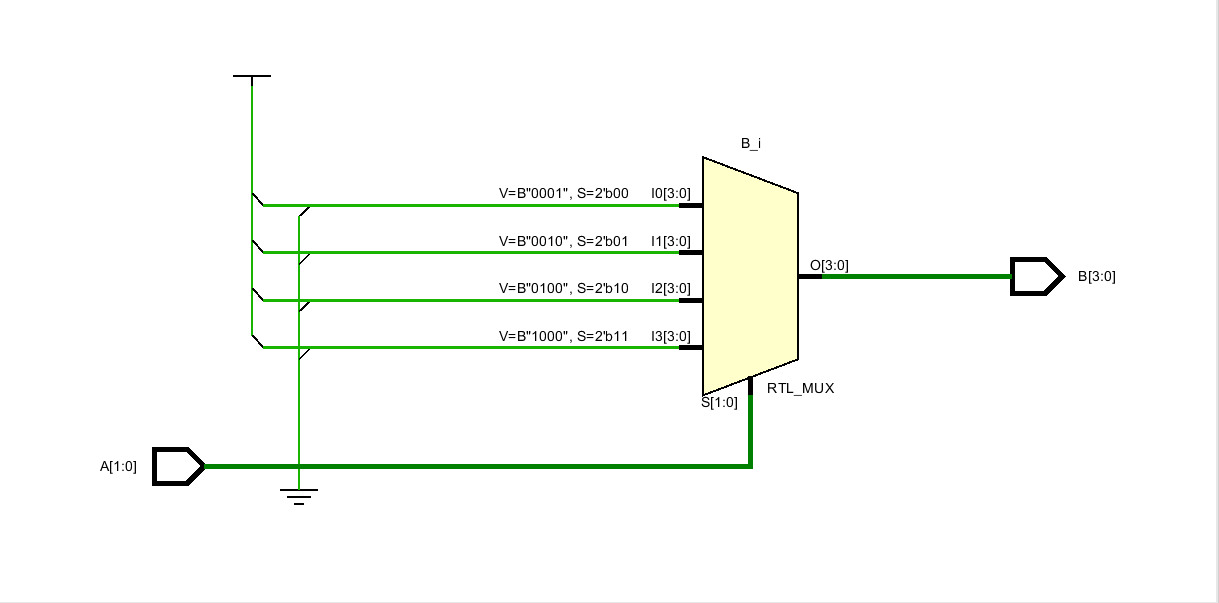
when “10” => d =< “0100”;

when “11” => d =< “1000”;

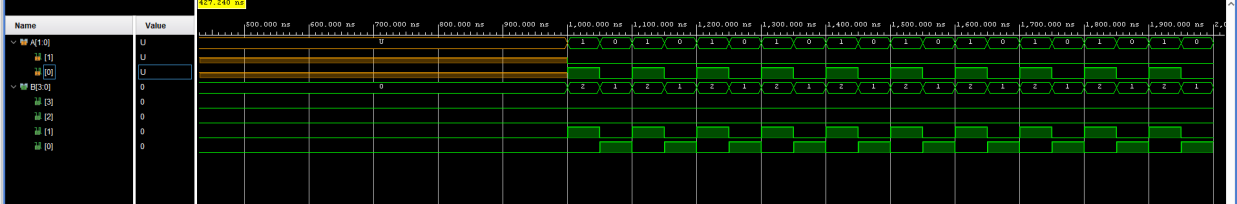
when others => d <= “0000”; end case;

end process; end behavioral;

**RTL Schematics:**



**Waveform:**



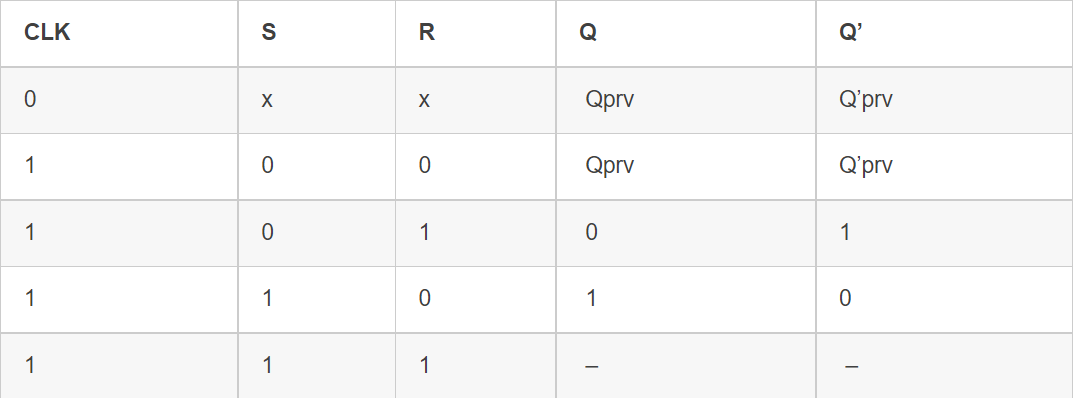
**Learning Outcome:** Mentioned decoder are implemented in VHDL and their properties are verified.

### Experiment – 7

**Aim:** Perform SR flip flop with behavioural modelling.

**Software Used:** Xilinx Vivado 2020.1

**Truth Table:**



**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';

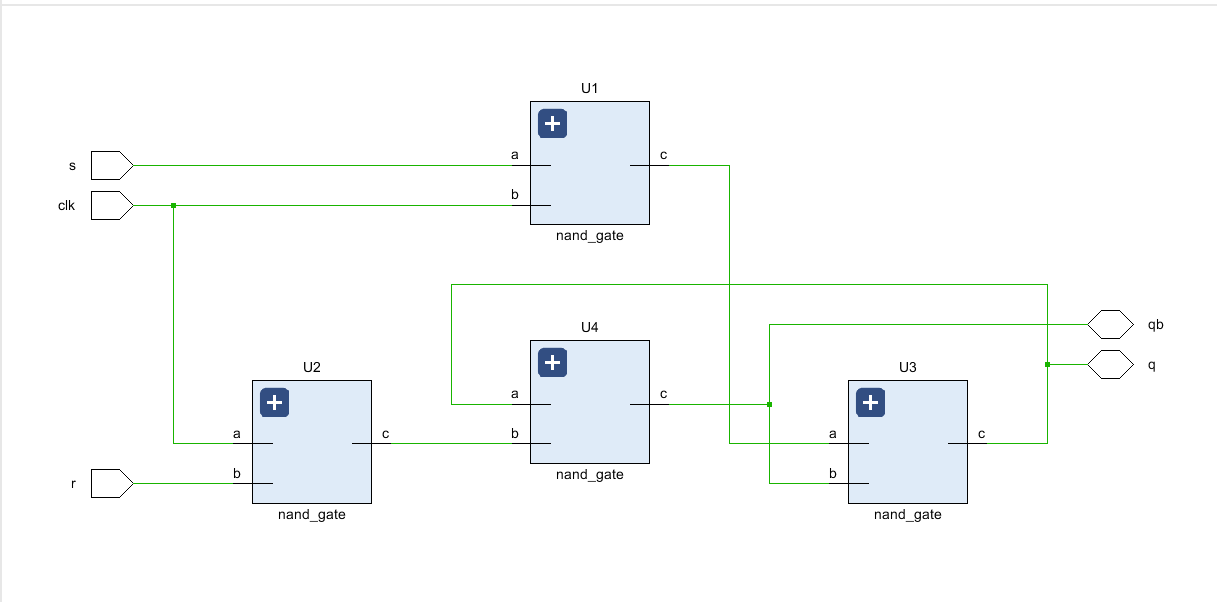
Qb <= 'Z';

end if; end if;

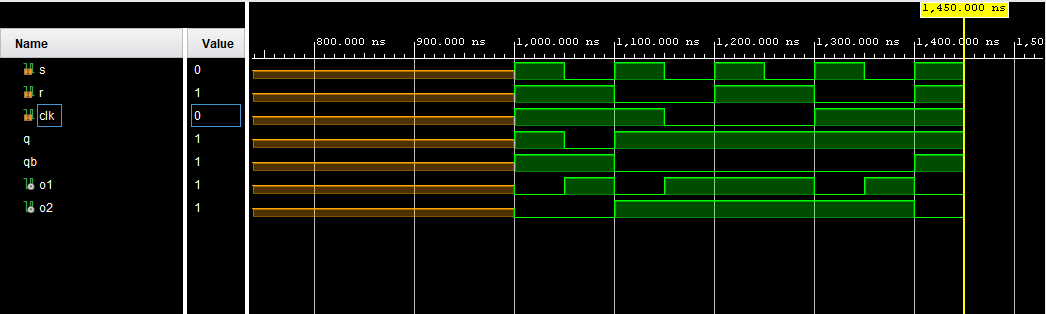
end process;

end Behavioral;

**RTL Schematics:**



**Waveform:**



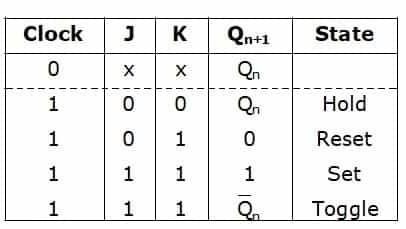
**Learning Outcome:** In this experiment, SR flip flop was successfully implemented in Vivado and results are varified.

### Experiment – 8

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

**Software Used:** Xilinx Vivado 2020.1

**Truth Table:**



**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';

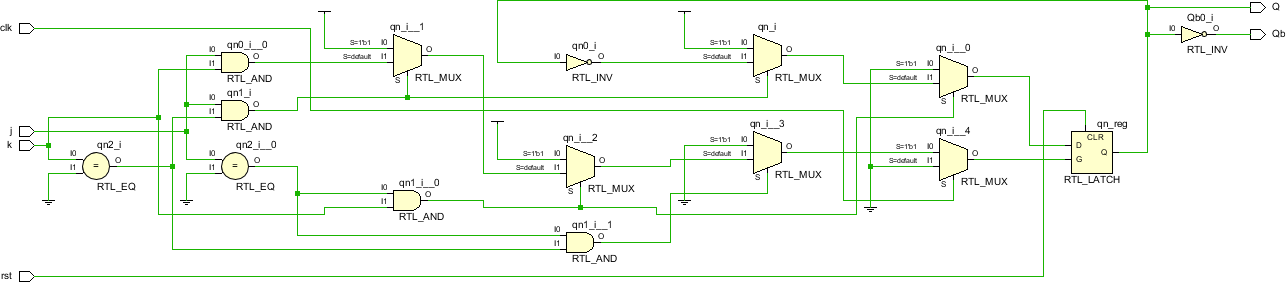
elsif(J='1' and K='1')then qn := not qn;

else null; end if; else null; end if;

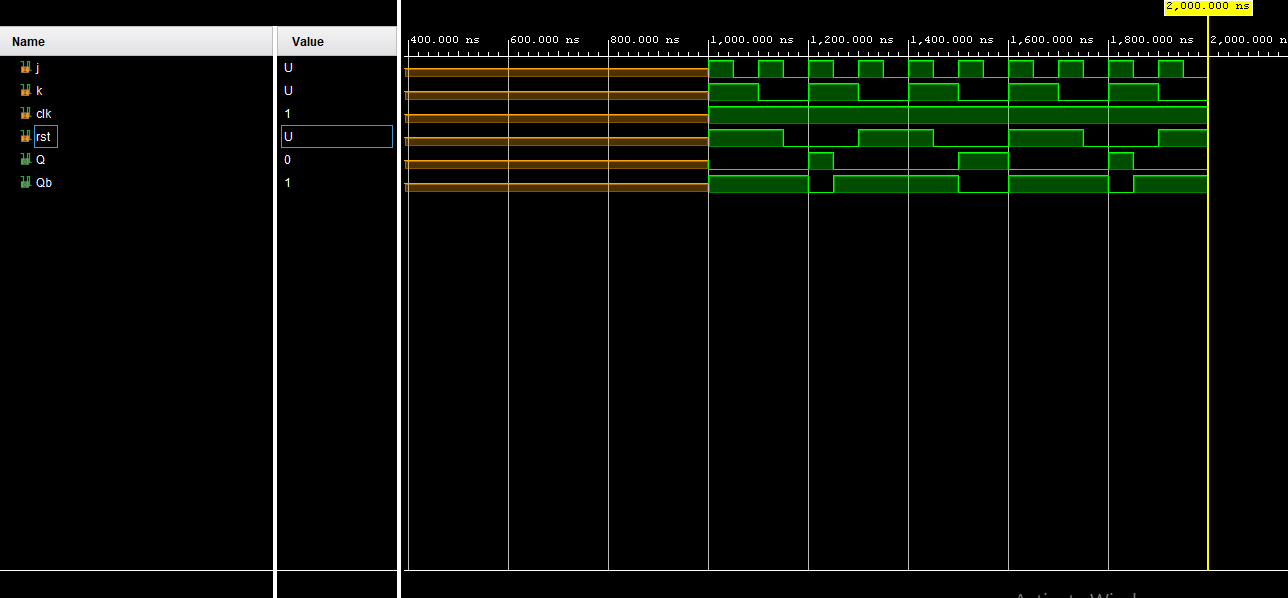
Q <= qn;

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.

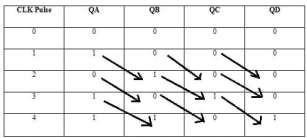
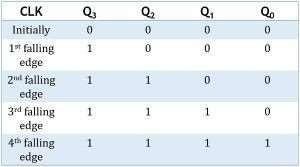
**SIPO:** The shift register, which allows serial input (one bit after the other through a single data line) and produces a parallel output, is known as the Serial-In Parallel-Out shift register.

**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: SIPO:**



**PIPO PISO**

**Coding:**

library IEEE; use IEEE.STD LOGIC 1164. ALL;

entity SISO\_21209 in

## SISO

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

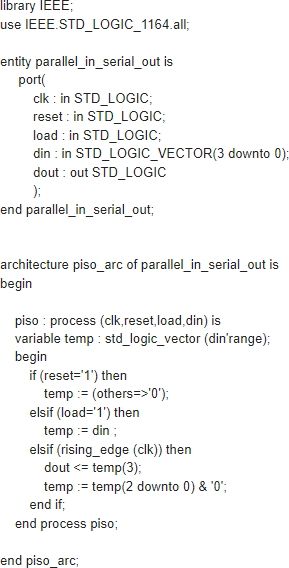
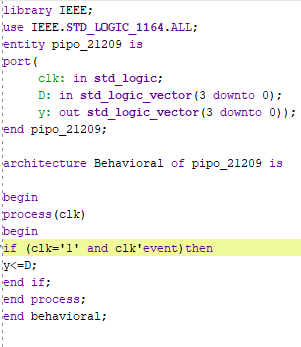
begin

process (clk) begin

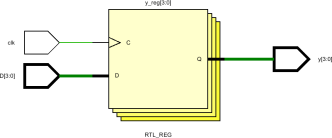
if clear = '1' then Q<= "0000";

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 PISO**

**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.

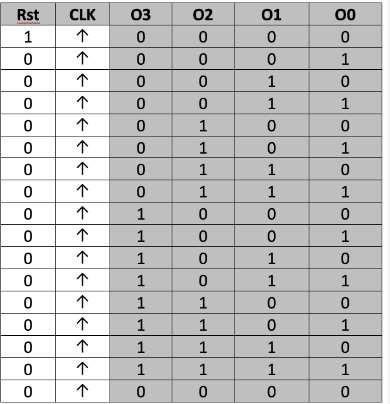
**Software Required:** Xilinx Vivado 2020.1

**Theory:**

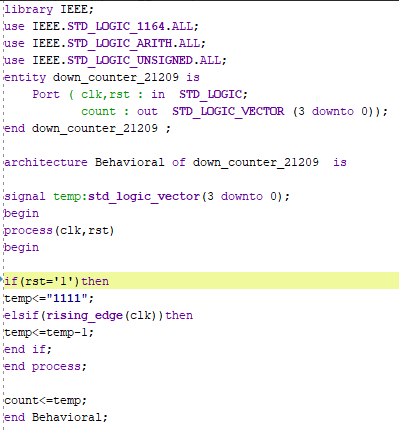
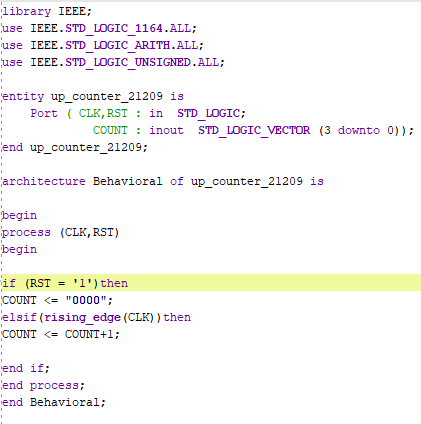
Counter- In digital logic and computing, a Counter is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal. Counters are used in digital electronics for counting purpose, they can count specific event happening 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:**



**Coding:**

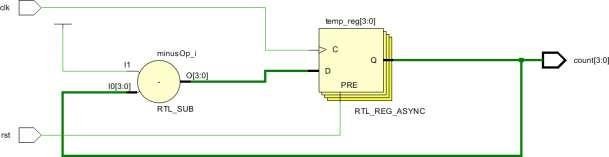


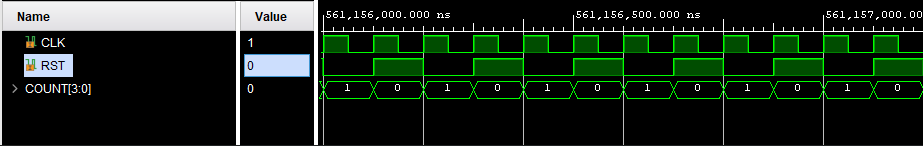
**UP**

**Schematic Result:**

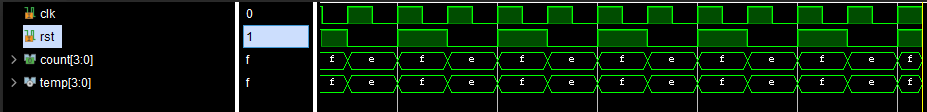
## UP

**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:**

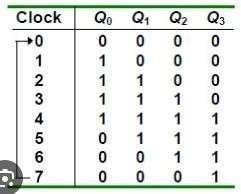
Counter- In digital logic and computing, a Counter is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal.

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.

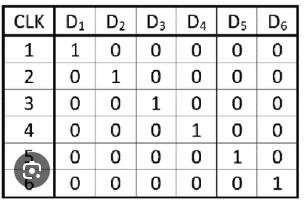
A ring counter is a typical application of the Shift register. The ring counter is almost the same as the shift counter. The only change is that the output of the last flip-flop is connected to the input of the first flip-flop in the case of the ring counter but in the case of the shift register it is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

1. **Truth Table:**

**JOHNSON COUNTER**

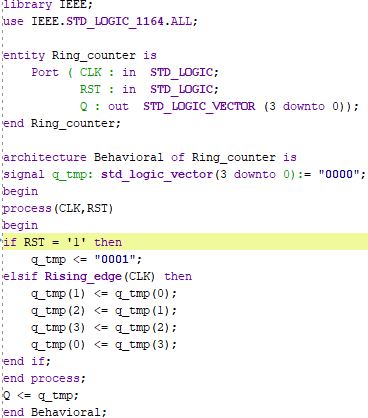
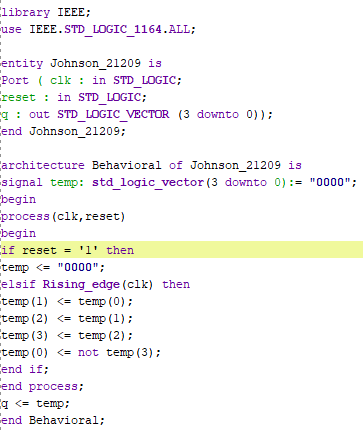


**RING COUNTER**



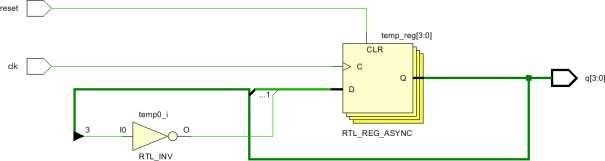
**Coding:**

Johnson Counter Ring Counter

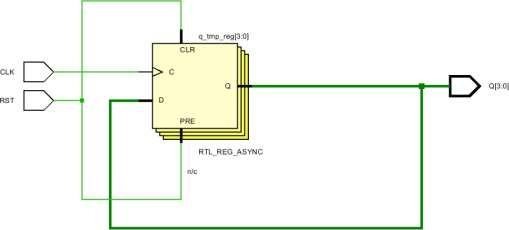


**Schematic Result:**

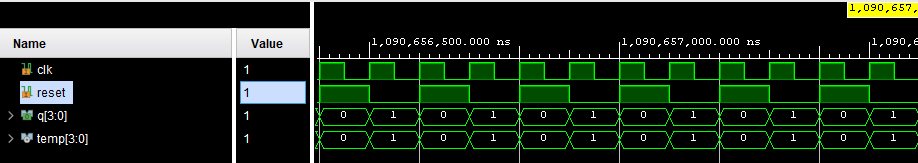
#### JOHNSON



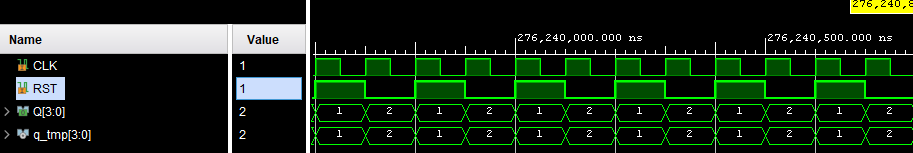
**RING**



1. **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.

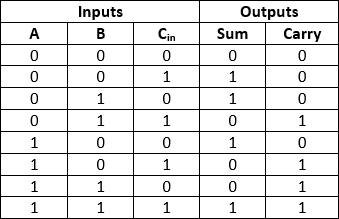
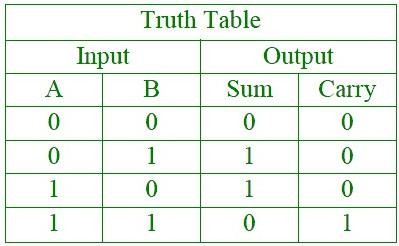
### Experiment – 12

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

**Software Used:** Xilinx Vivado 2020.1

**Truth Table:**

**Half-Adder: Full-Adder:**



**Program:**

entity H\_adder is port(

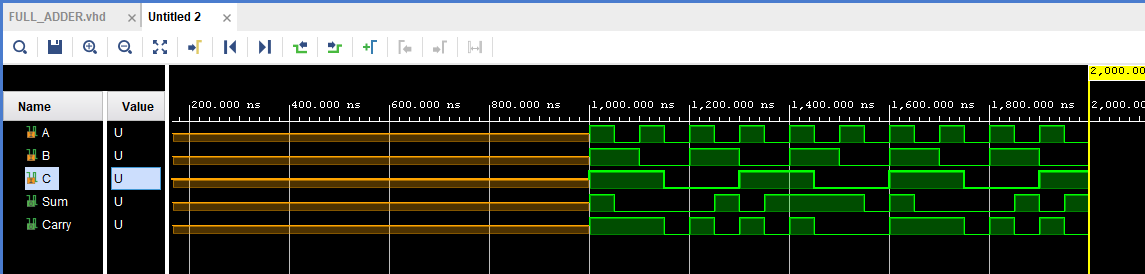
a,b : IN std\_logic; sum,carry : OUT std\_logic); end H\_adder;

architecture dataflow of H\_adder is begin

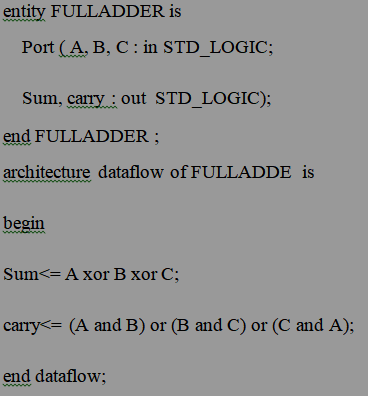
sum <= a xor b; carry <= a and b; end dataflow;

**RTL Schematics:**

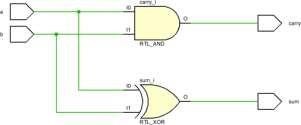
**Waveform:**

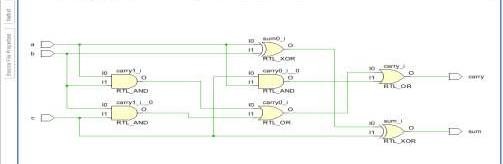


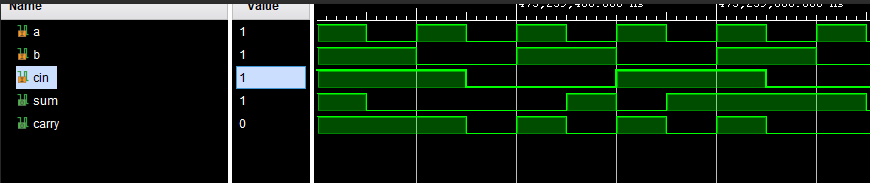
**Full Adder**



**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.

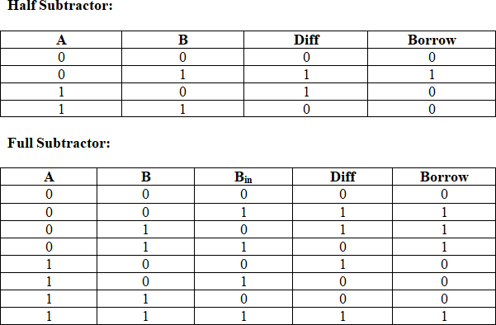
Create and simulate simple RTL designs using VHDL and was able to simulate them.

### Experiment – 13

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

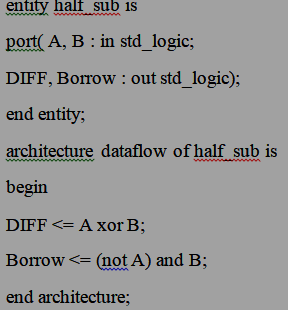
**Software Used:** Xilinx Vivado 2020.1

**Truth Table:**

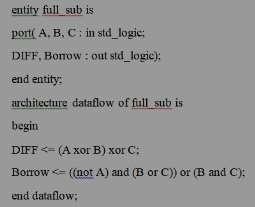


**Program:**

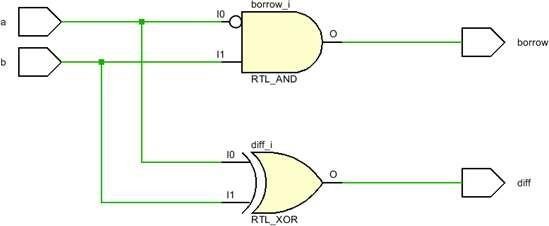
**Half-Subtractor**



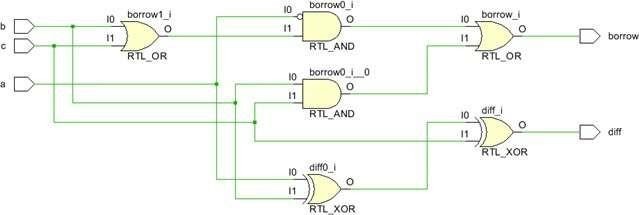
**Full Subtractor**



**RTL Schematics Half Subtractor**

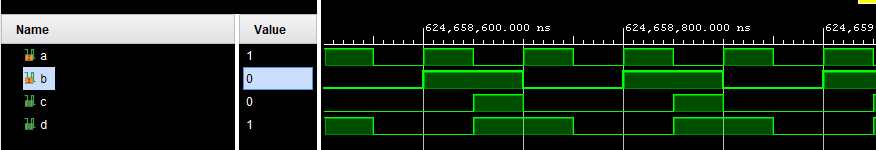


**Full Subtractor**

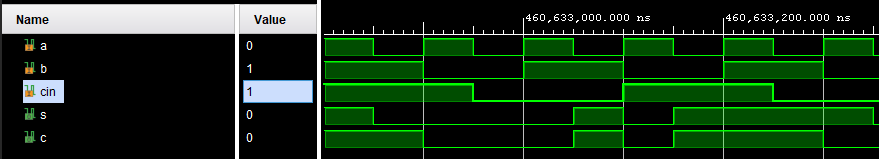


**RTL Result**

**Half Subtractor**



**Full Subtractor**



**Learning Outcome:**

Understood the basic concepts of VHDL design.

Familiarity with the Vivado design environment and user interface.

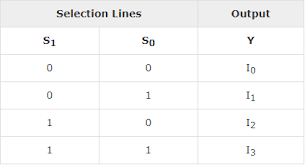
Create and simulate simple RTL designs using VHDL and was able to simulate them.

# Experiment – 14

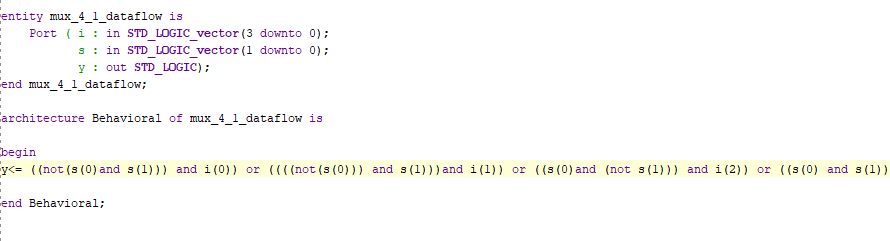
**Aim:** Write a program in VHDL for the implementation of multiplexer using dataflow model.

**Software Used:** Xilinx Vivado.

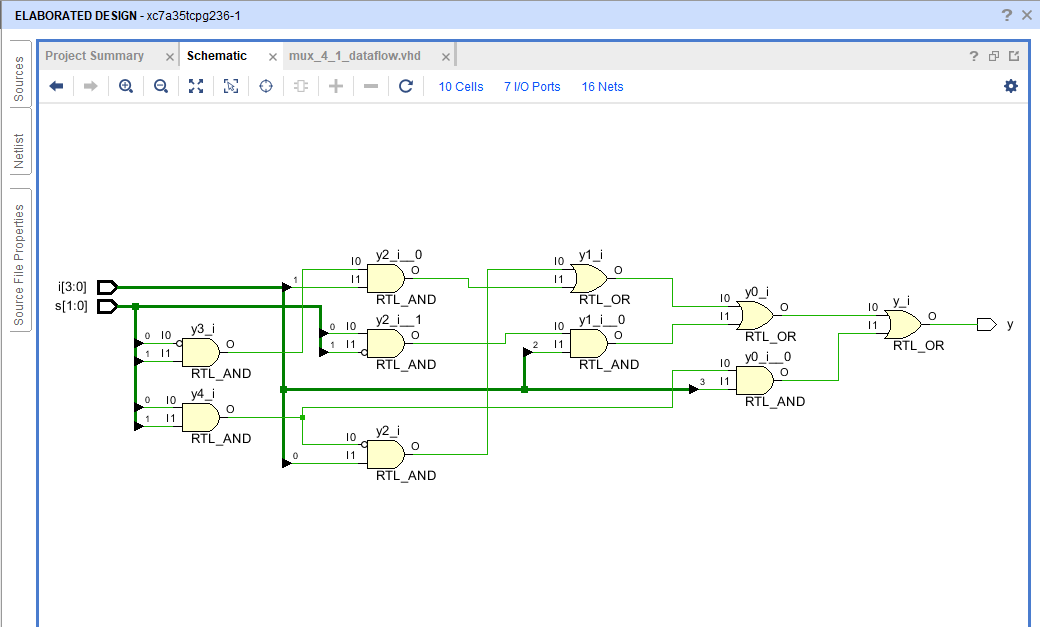
**Truth Table:**



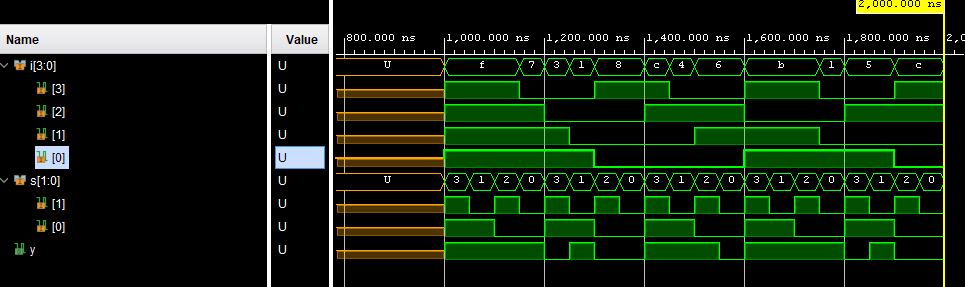
**Program:**



**RTL Schematics:**



**Waveform:**



**Learning Outcome:** Understood the basic concepts of VHDL design. Familiarity with the Vivado design environment and user interface.

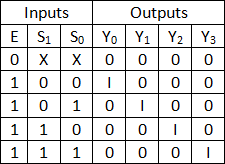
Create and simulate simple RTL designs using VHDL and was able to simulate them.

# Experiment – 15

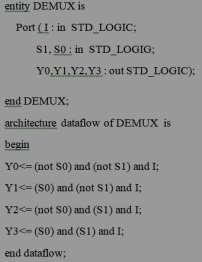
**Aim:** Write a program in VHDL for the implementation of 1x4 demultiplexer using dataflow model.

**Software Used:** Xilinx Vivado.

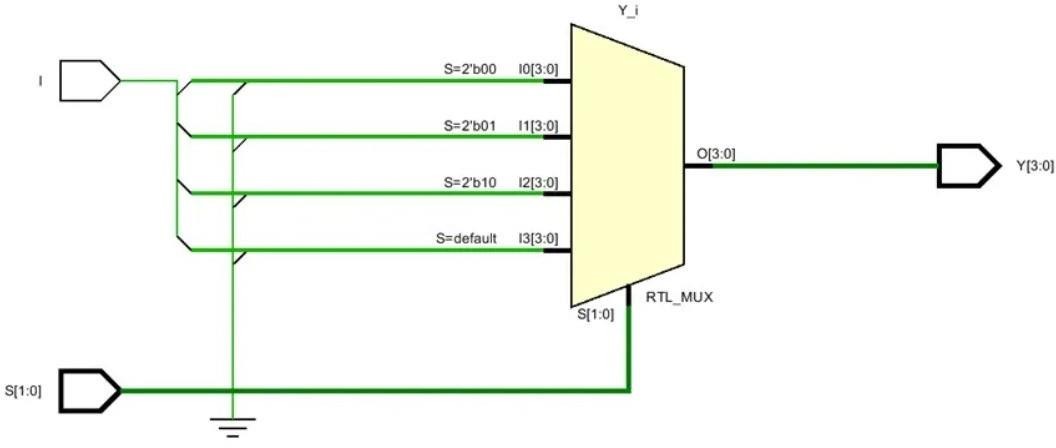
**Truth Table:**



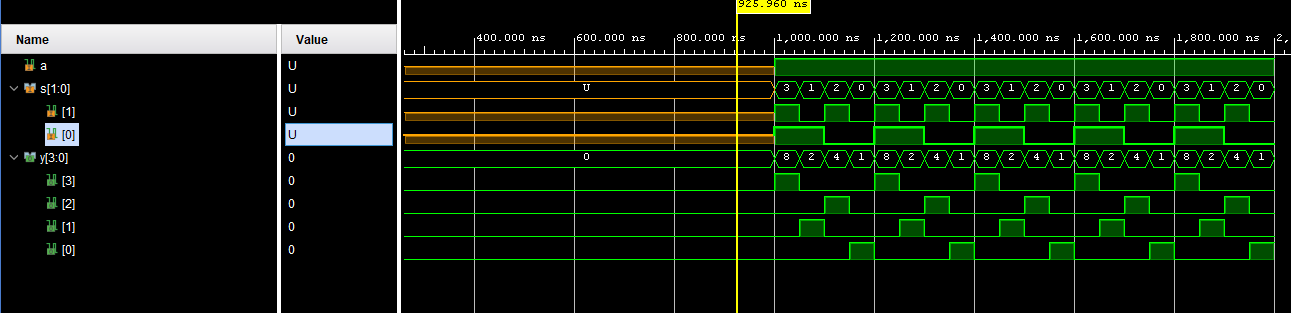
**Program:**



**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);

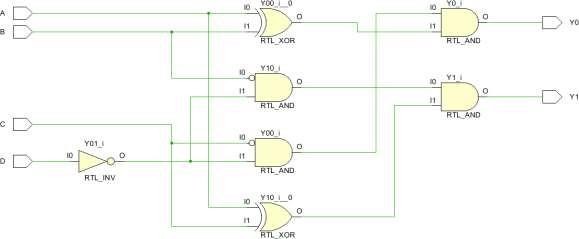
end data\_flow\_21231\_encoder;

architecture dataflow of data\_flow\_21231\_encoder is begin

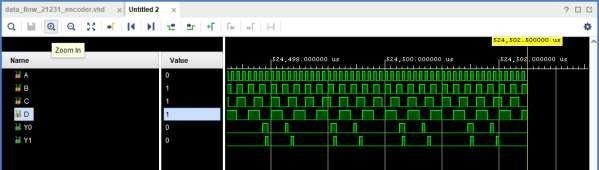
Y0 <= ((not C)and(not D))and(A xor B); Y1 <= ((not B)and(not D))and(a xor C);

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.

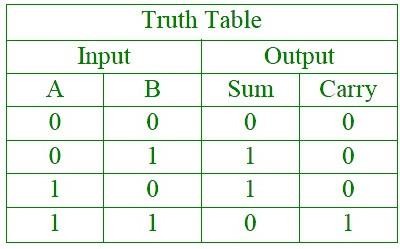
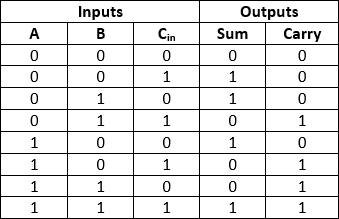
# Experiment – 17

**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: Half-Adder:**



**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;

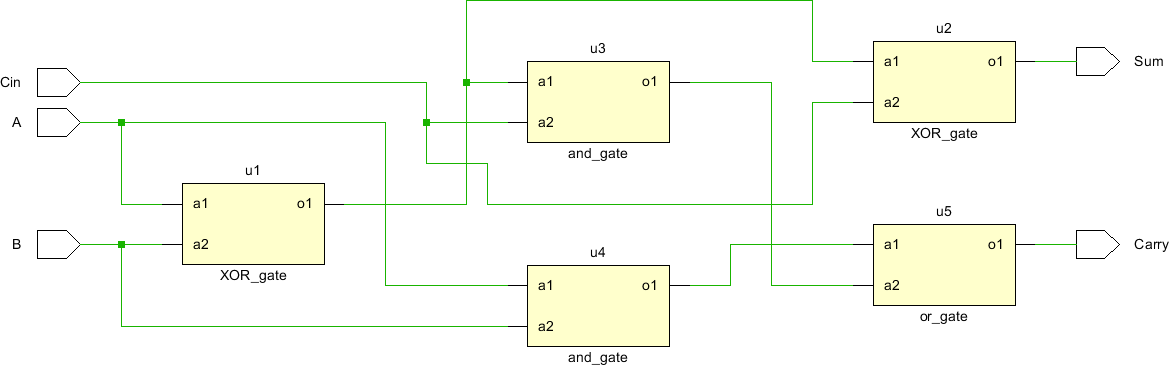
begin

U1:HA PORT MAP(A=>FA,B=>FB,S=>S0,C=>S1); U2:HA PORT MAP(A=>S0,B=>FC,S=>FS,C=>S2);

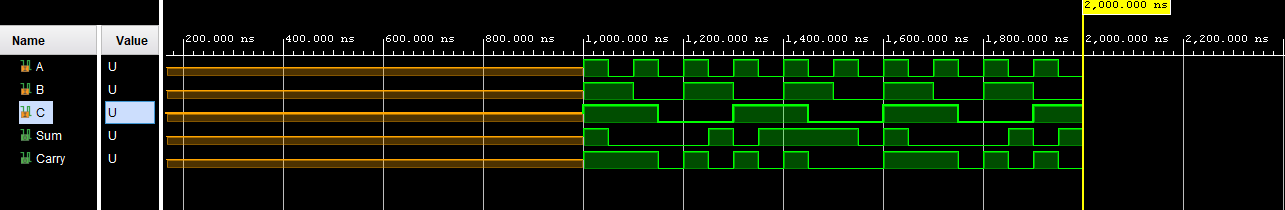
U3:ORGATE PORT MAP(X=>S2,Y=>S1,Z=>FCA);

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);

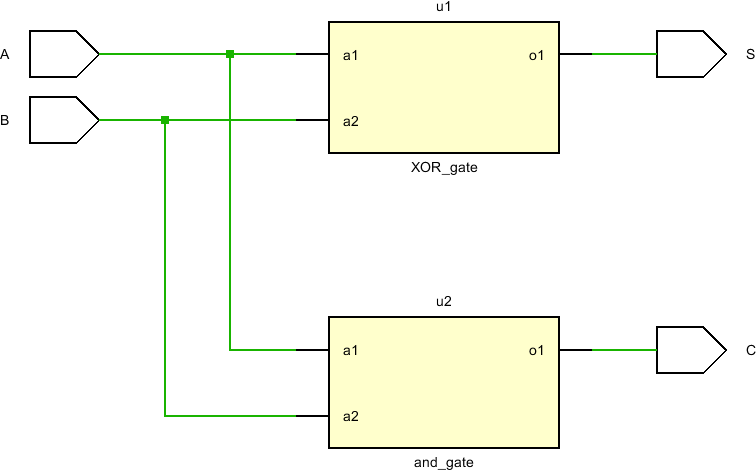
end component;

begin

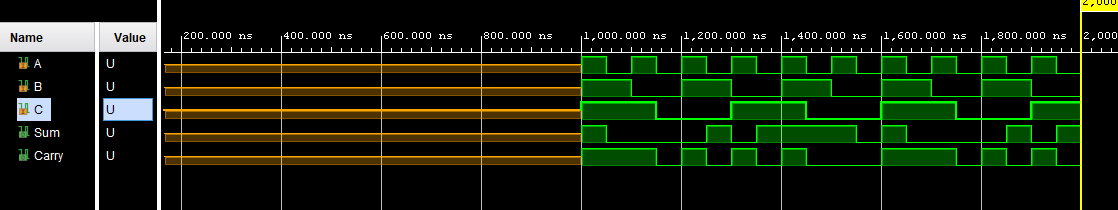
u1: xor\_gate port map (i1 => a, i2 => b, o1 => sum);

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.

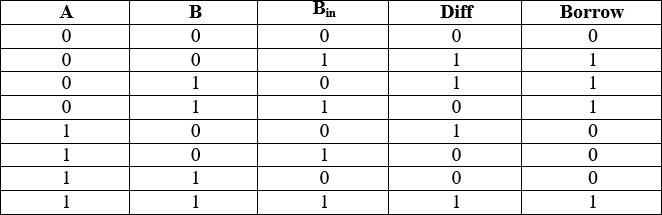
Create and simulate simple RTL designs using VHDL and was able to simulate them.

# 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:**



**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

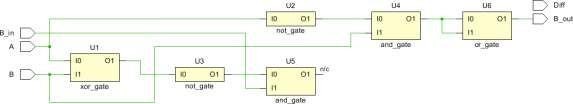
U1 : xor\_gate port map(A, B, 61

N1); U2 : not\_gate port

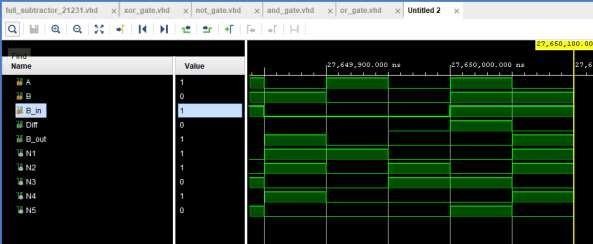
map(A, N2); U3 : not\_gate port map(N1, N3);

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:**

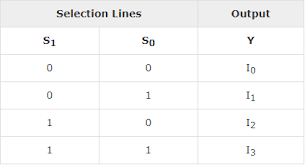
* Understood the basic concepts of VHDL design.
* Familiarity with the Vivado design environment and user interface.
* Create and simulate simple RTL designs using VHDL and was able to simulate them.

# 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:**



**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:

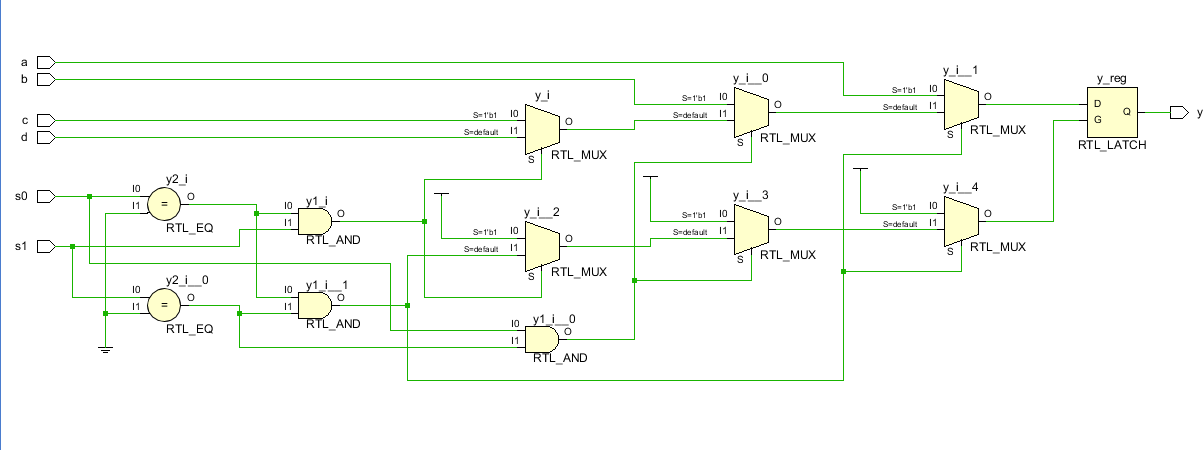
inv port map (Sel1, selbar1);

A1: and3 port map (A, selbar0, selbar1, t1); A2:

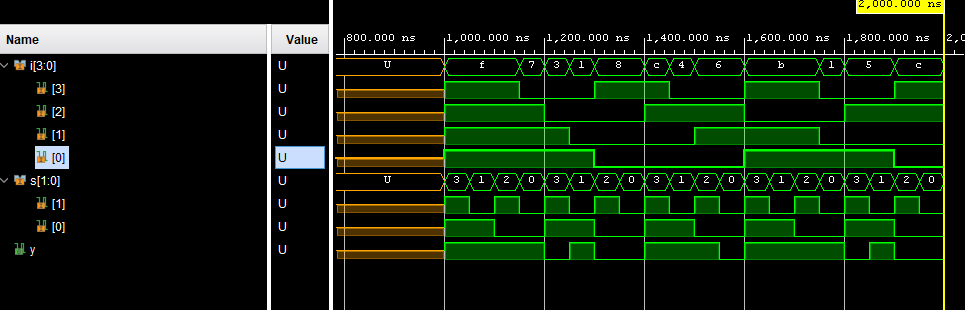
and3 port map (B, Sel0, selbar1, t2); A3: and3 port map (C, selbar0, Sel1, t2); A4: and3 port map (D, Sel0, Sel1, t4);

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.

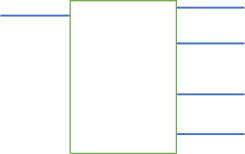
# Experiment - 20

**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:**



**a**

**y[0**

**b**

**y[1**

2:4 decoder

**y[2**

**y[3**

**en**

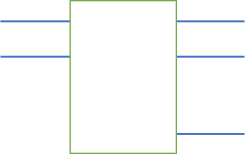


**a**

**b**

2:4 decoder

**en**



**y[4**

2:4 decoder

**y[5**

**y[6**

**en**

**y[7**

**y[8**

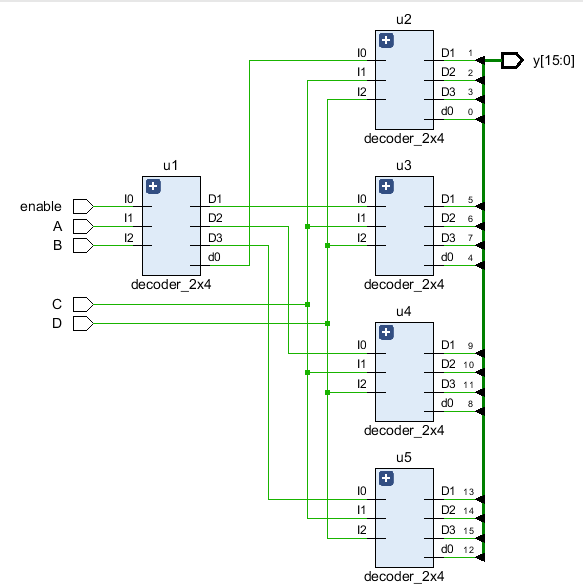
**y[9**

2:4 decoder

**y[10**

**y[11**

|  |  |  |
| --- | --- | --- |
|  | 2:4 decoder |  |
|  | **y[12** |
| **y[13** |
|  |
| **y[14** |
| **y[15** |

**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));

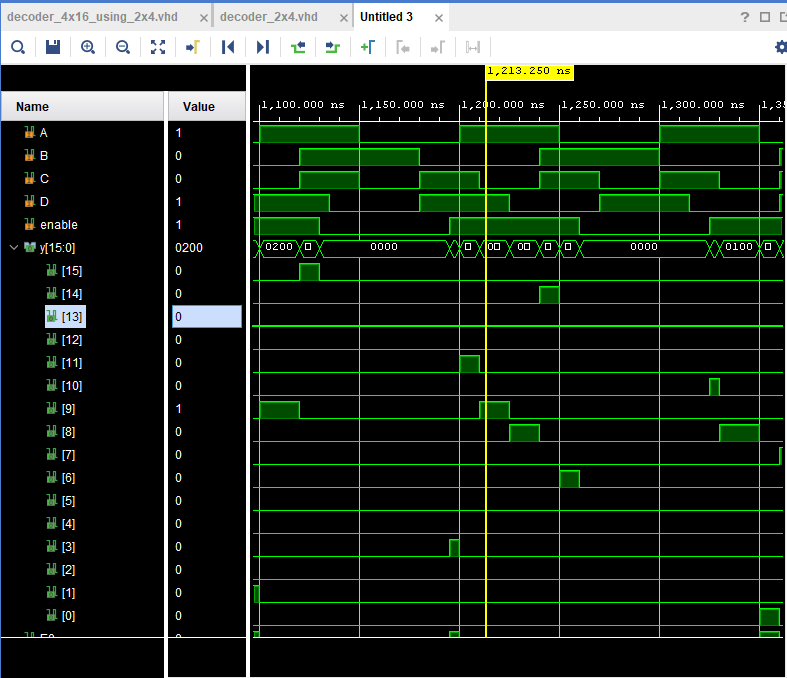
U3 : decoder\_2x4 port map(N2, c, d, Y(4), Y(5), Y(6), Y(7));

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;

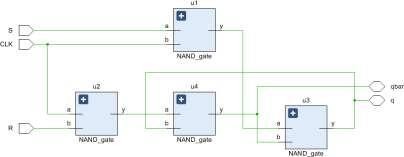
architecture structural of SR\_FF\_STR\_21231 is component NAND\_gate

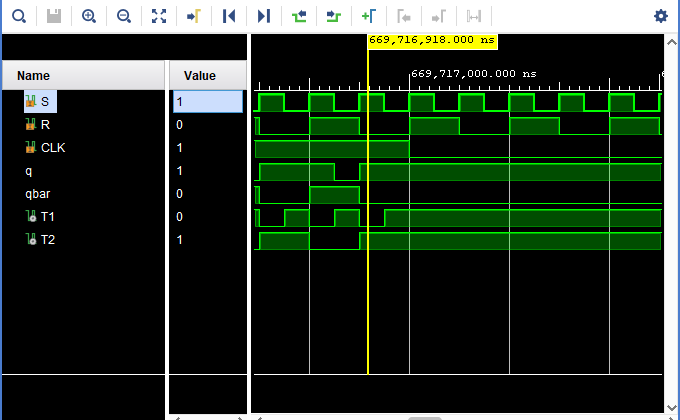
port(a, b:in std\_logic; y:OUT std\_logic); end component; signal T1,T2:

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';

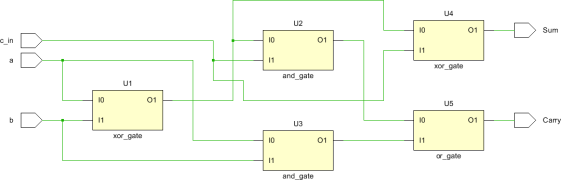
b <= 'l';

wait for 10ns; end process;

end 71

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;

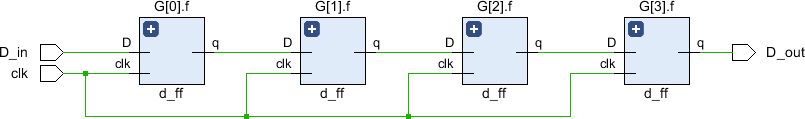
'G:for i in 0 to 3 generate

f: d\_ff port map(temp(i), clk, temp (i+1) ); end generate G;

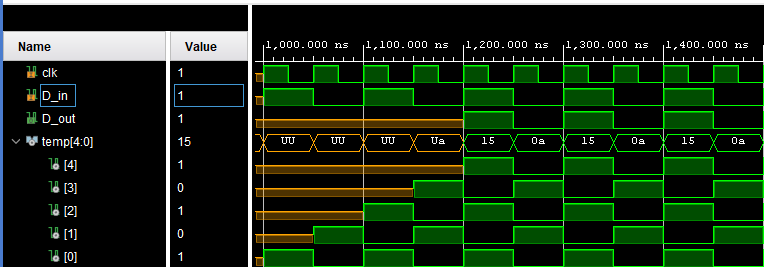
D\_out <= temp

(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));

Architecture Behavioral of counter is begin

process(clk

) begin

if (clk' event and clk='1') then

if (temp <"1001") then

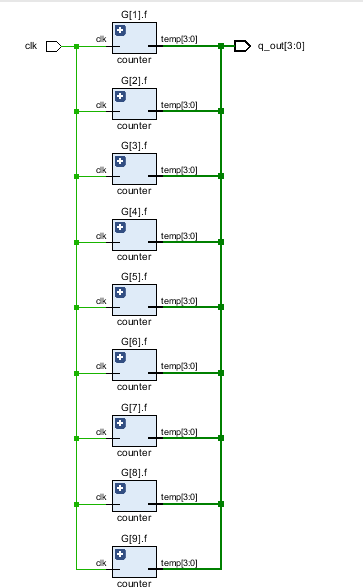
temp <= conv\_std\_logic\_vector(conv\_integer (temp) +1,

4); else temp <=

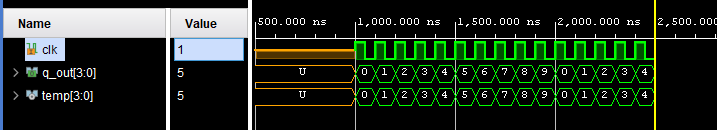
"0000"; end if; end if;

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;

* 1. **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)]

* 1. **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:**

A

O

C

B

RTL\_AND

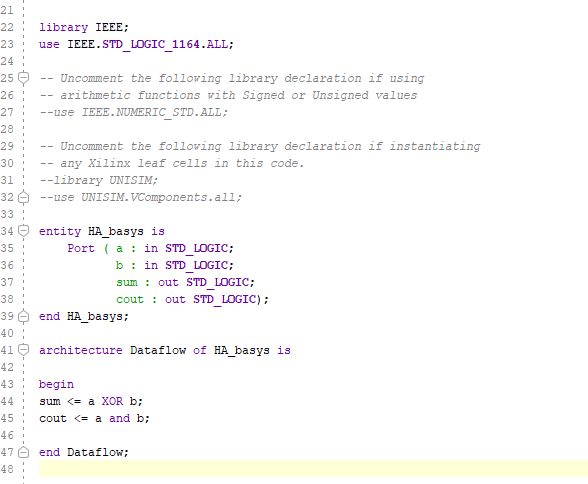
**Learning Outcome:** Here we can burn and gate on FPGA with Basys3 on Vivado with help of the same basic commands of VHDL.

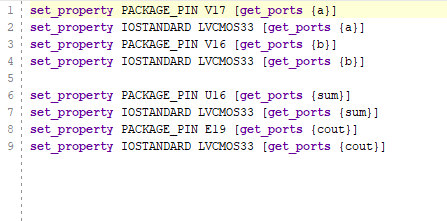
# Experiment – 26

**Aim:** Perfrom FPGA burning of half adder with Basys3.

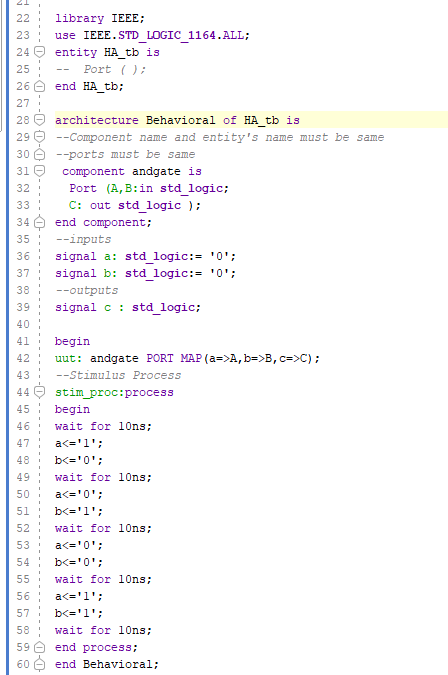
**Software Used:** Xilinx Vivado.2020.1

**Program:**

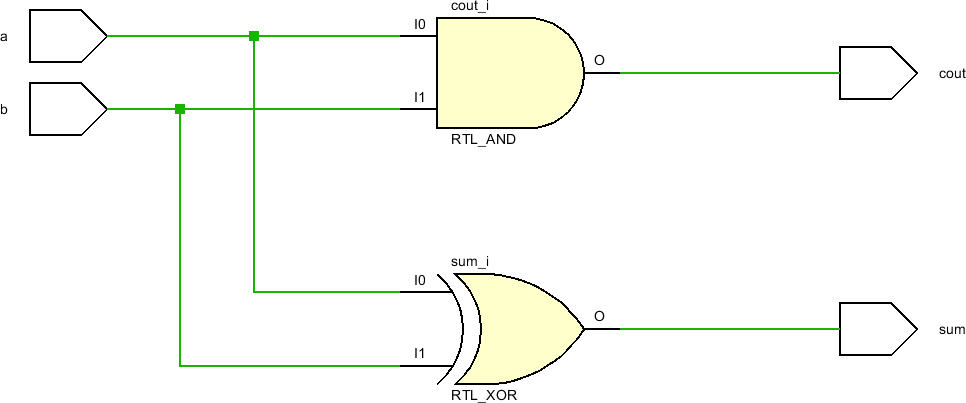
1. **Design Source File**
2. **Constraint File**



1. **Simulation File**



**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