**Experiment 3: Introduction to VHDL**

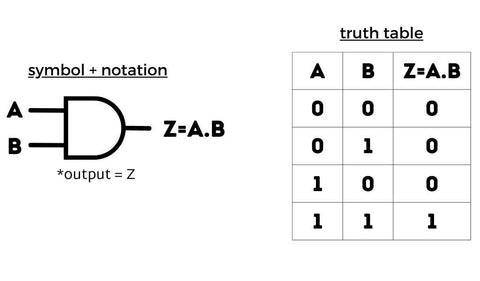
**Objective:**

**To implement the Basic AND gate using VHDL**

**Theory:**

VHDL (VHSIC Hardware Description Language) is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits. VHDL can also be used as a general-purpose parallel programming language.

An AND gate is a logic gate with two or more inputs and a single output. It operates based on logical multiplication rules. The output is only high (1) if all inputs are high (1). If any input is low (0), the output will also be low. AND gates are fundamental components in digital circuits and can be found in various electronic devices like smartphones, tablets, and memory devices.



**VHDL code:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity And\_gate is

port(A: in STD\_LOGIC;

B: in STD\_LOGIC;

y: out STD\_LOGIC);

end And\_gate;

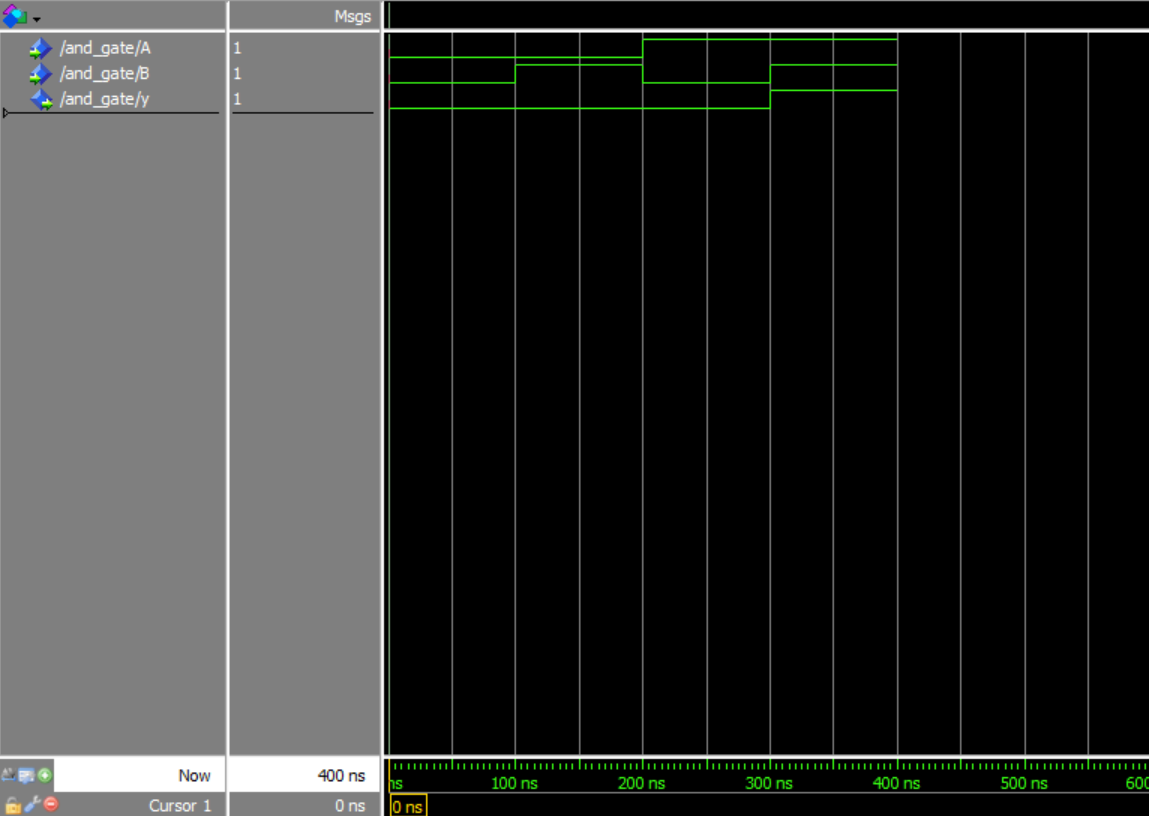
architecture Behavioral of And\_gate is

begin

y<=A and B;

end Behavioral;

**OUTPUT:**



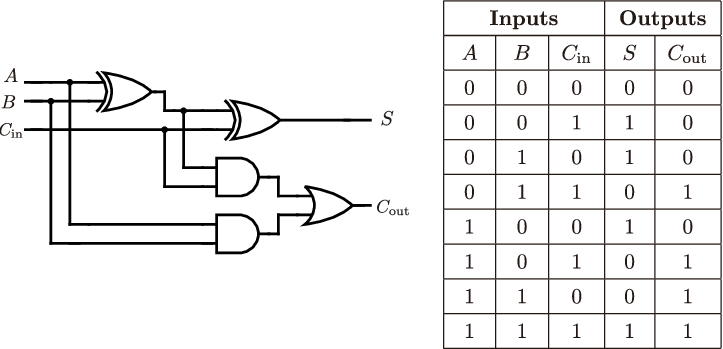
**Experiment 4: Full adder**

**Objective:**

To implement the Full adder using VHDL

**Theory:**

A full adder is a digital circuit that adds three binary inputs and produces two output bits. It has three inputs: A, B, and Cin (carry-in), and two outputs: sum and carry. The circuit can be implemented using logic gates like XOR, AND, and OR gates. By using multiple full adders, you can create circuits to add N-bit numbers, known as ripple-carry adders. These adders have carry bits that ripple through each full adder. The advantages of full adders include their ability to handle multiple bits for addition and their applications in arithmetic operations in digital systems.

****

**VHDL code:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Full\_Adder is

port(A: in STD\_LOGIC;

B: in STD\_LOGIC;

C: in STD\_LOGIC;

Sum: out STD\_LOGIC;

carry: out STD\_LOGIC);

end Full\_Adder;

architecture Behavioral of Full\_Adder is

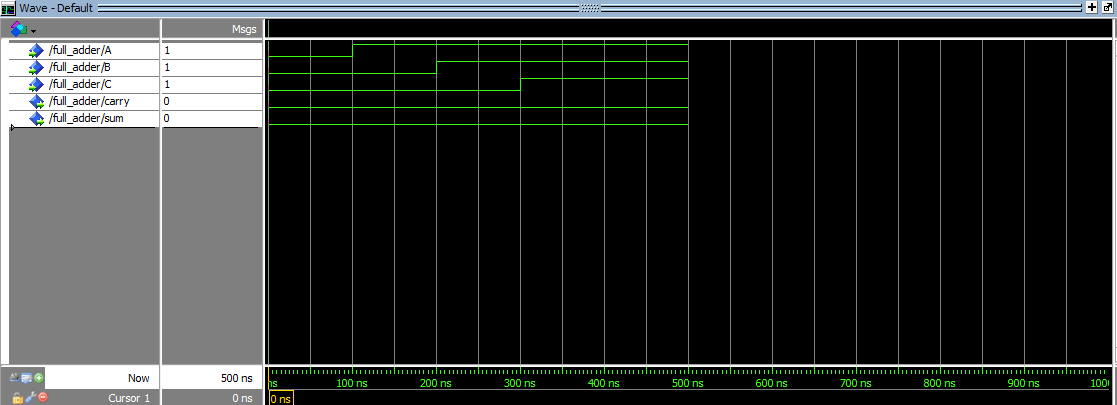
begin

Sum<= A XOR B XOR C;

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

end Behavioral;

**OUTPUT:**



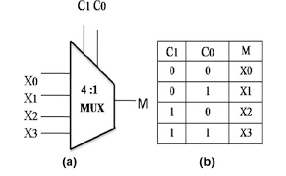
**Experiment 5: 4:1 Multiplexer**

**Objective:**

To implement the 4:1 Mux using VHDL

**Theory:**

A 4:1 multiplexer is a combinational logic circuit that selects one input from four different inputs based on a binary selection line. It consists of four data input lines, two control lines (select lines), and one output line. To implement a 4:1 multiplexer, you can use 2:1 multiplexers along with other logic gates. An 8-to-1 multiplexer can be created by cascading two 4-to-1 and one 2-to-1 multiplexer, providing a total of 3 select inputs equivalent to an 8-to-1 multiplexer. This device is used in digital and analog electronics for various applications due to its design and advantages.

****

**VHDL code:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

entity mux is

port(

A,B,C,D : in STD\_LOGIC;

S0,S1: in STD\_LOGIC;

Z: out STD\_LOGIC );

end mux;

architecture Behavioral of mux is

begin

process (A,B,C,D,S0,S1) is

begin

if (S0 ='0' and S1 = '0') then

Z <= A;

elsif (S0 ='1' and S1 = '0') then

Z <= B;

elsif (S0 ='0' and S1 = '1') then

Z <= C;

else

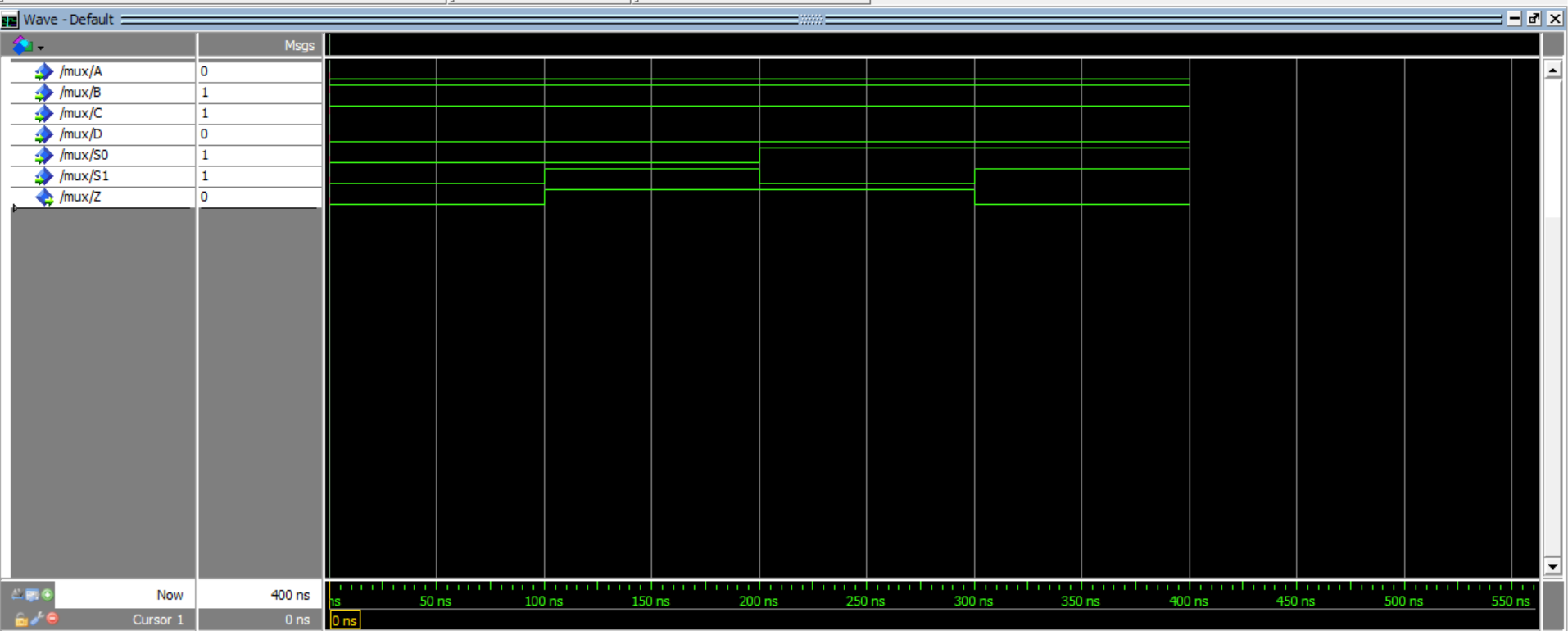
Z <= D;

end if;

end process;

end Behavioral;

**OUTPUT:**



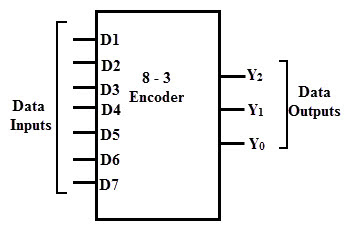
**Experiment 6: 8:3 Encoder**

**Objective:**

To implement the 8:3 Encoder using VHDL

**Theory:**

An 8:3 Encoder is a digital circuit that encodes 8 inputs into 3 outputs. It consists of 8 input lines, each corresponding to an octal digit, and 3 output lines that generate the corresponding binary code. The encoder can be built using logic gates and combinational logic design. The truth table for an 8-to-3 binary encoder shows the relationship between the inputs and outputs. The encoder is used in various applications where multiple data inputs need to be converted into a binary code efficiently.



**VHDL code:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity encoder is

Port ( input\_vector : in STD\_LOGIC\_VECTOR(7 downto 0);

output\_vector : out STD\_LOGIC\_VECTOR(2 downto 0));

end encoder;

architecture Behavioral of encoder is

begin

process(input\_vector)

begin

case input\_vector is

when "00000001" => output\_vector <= "000";

when "00000010" => output\_vector <= "001";

when "00000100" => output\_vector <= "010";

when "00001000" => output\_vector <= "011";

when "00010000" => output\_vector <= "100";

when "00100000" => output\_vector <= "101";

when "01000000" => output\_vector <= "110";

when "10000000" => output\_vector <= "111";

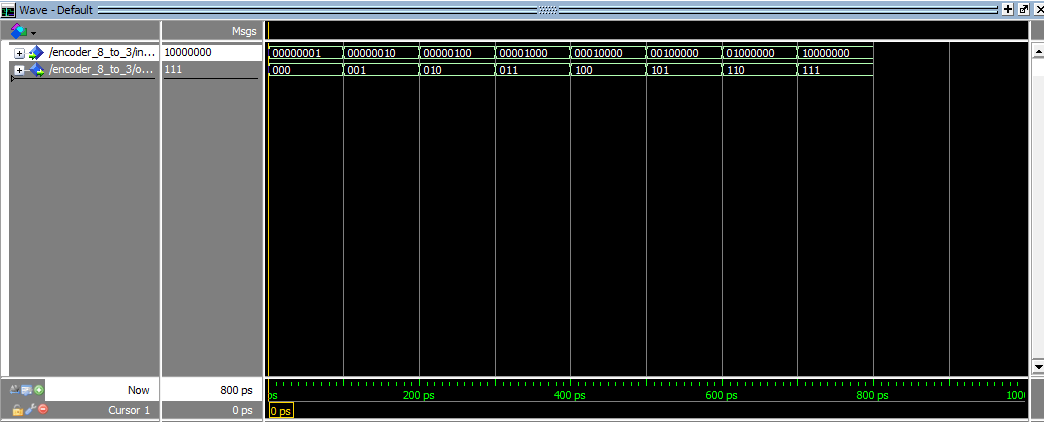
when others => output\_vector <= "000"; -- Default case, in case no input is matched

end case;

end process;

end Behavioral;

**OUTPUT:**



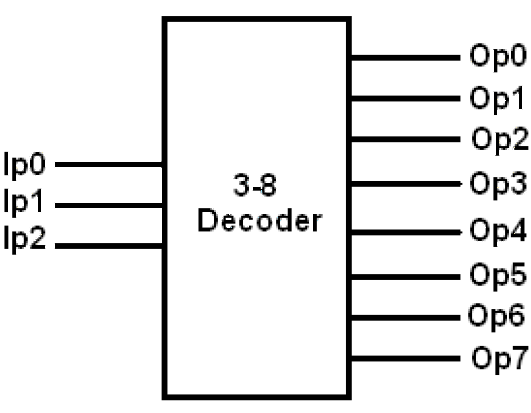
**Experiment 7: 3:8 Decoder**

**Objective:**

To implement the 3:8 Decoder using VHDL

**Theory:**

A 3 to 8 decoder is a digital circuit that converts a 3-bit binary input into one of eight possible outputs. It can be designed using 2 to 4-line decoders. The 3 selector inputs, A0, A1, and A2, can create 8 combinations (2^3=8) for the outputs. This decoder can be used for tasks like demultiplexing or multiplexing. The logic diagram, truth table, and block diagram of a 3 to 8 decoder illustrate its functionality.

****

**VHDL code:**

library IEEE;

use IEEE.std\_logic\_1164.All;

entity Decoder is

port(

A: in std\_logic\_vector(2 downto 0);

Y: out std\_logic\_vector (7 downto 0) );

end Decoder;

Architecture behaviour of Decoder is

begin

process(A) is

begin

if(A="000") then

Y<="00000001";

elsif(A="001") then

Y<="00000010";

elsif(A="010") then

Y<="00000100";

elsif(A="011") then

Y<="00001000";

elsif(A="100") then

Y<="00010000";

elsif(A="101") then

Y<="00100000";

elsif(A="110") then

Y<="01000000";

elsif(A="111") then

Y<="10000000";

end if;

end process;

end behaviour;

**OUTPUT:**



**Experiment 8: 3-Segment Pipeline**

**Objective:**

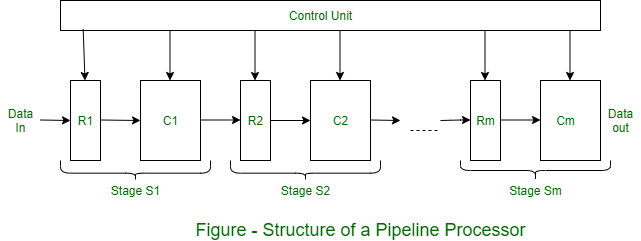
To implement the 3-Segment Pipeline using VHDL

**Theory:**

A 3-segment pipeline refers to a computer processor architecture where the instruction execution process is divided into three distinct stages or segments. Each stage corresponds to a specific operation in the instruction execution cycle, and multiple instructions can be processed simultaneously, each at a different stage of the pipeline.

In a three-segment pipeline, the stages are as follows:

* Instruction fetched from memory is decoded.
* Effective address is calculated.
* Operand from memory is fetched, and the instruction is executed

****

**VHDL code:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

Entity pipeline is

Port (a:integer;

b: IN INTEGER;

c: in integer;

clk:in STD\_LOGIC;

y:out integer);

end pipeline;

architecture Behavioral of pipeline is

signal r1,r2,r3,r4,r5,r6: integer :=0;

begin

y<=r5;

process(clk)

begin

if(rising\_edge(clk))then

for i in 0 to 2 loop

case(i) is

when 0=>

r1<=a;

r2<=b;

r3<=c;

when 1=>

r4<=r1+r2;

when 2=>

r5<=r4\*r3;

when others=>

null;

end case;

end loop;

end if;

end process;

end Behavioral;

**OUTPUT:**

