Implementation of Encoder & Decoder using VHDL

▪ OBJECTIVE:

To implement **Encoder & Decoder** using VHDL and verify its waveform in Xilinx.

▪ INTRODUCTION:

1. Encoder:

In digital electronics, an **encoder** is a combinational circuit that converts information from one format or code to another, typically from a higher number of input lines to a fewer number of output lines. Encoders are used to reduce the number of data lines, simplify circuit design, and enable efficient communication or processing.

➢ Key Features of Encoders

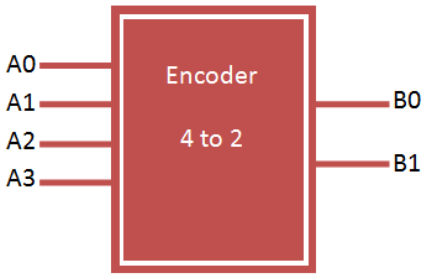
1. **Input and Output Lines**:
   - Encoders have 2^n input lines and n output lines. For example, a 4-to-2 encoder has 4 input lines and 2 output lines.
   - Each input corresponds to a unique binary code output.
2. **Priority**:
   - In a simple encoder, only one input should be active at any given time.
   - A **priority encoder** is used when multiple inputs may be active, assigning priority to the highest-priority input.

➢ Application Area:

- Keyboard encoders (e.g., converting keypresses into ASCII codes).
- Data multiplexing and demultiplexing.
- Memory addressing in microprocessors.

➢ Types of Encoders

1. **Binary Encoder**:
   - Converts a single active input (out of 2^n) into an n-bit binary code.
   - Example: 4-to-2 encoder.



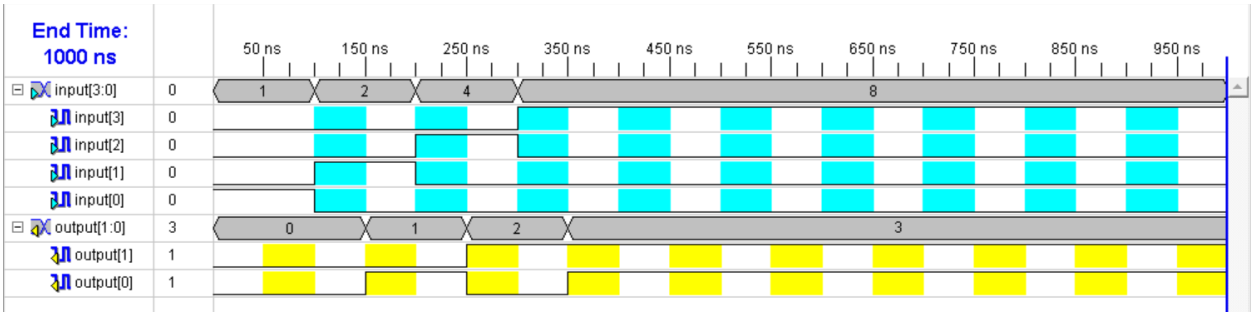
- **Truth Table:**

INPUT				OUTPUT	
A3	A2	A1	A0	B1	B0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

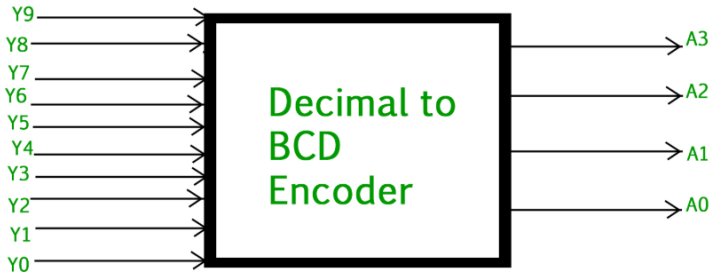
- **Source Code:**

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

entity Encoder4to2 is
    Port ( input : in  STD_LOGIC_VECTOR (3 downto 0);
           output : out  STD_LOGIC_VECTOR (1 downto 0));
end Encoder4to2;

architecture Behavioral of Encoder4to2 is

begin
process(input)
    begin
        case input is
            when "0001" => output <= "00"; -- Input 0 is active
            when "0010" => output <= "01"; -- Input 1 is active
            when "0100" => output <= "10"; -- Input 2 is active
            when "1000" => output <= "11"; -- Input 3 is active
            when others => output <= "00"; -- Default case for invalid inputs
        end case;
    end process;
end Behavioral;
```

- **Test Bench Waveform:**



2. **Octal-to-Binary Encoder**:
   - Encodes an 8-line input into a 3-bit binary output.
3. **Decimal-to-BCD Encoder**:
   - Converts a decimal input into a Binary Coded Decimal (BCD) representation.
   - Example: 16-to-4 encoder (but 6 input are **don't-care conditions**).



Decimal to BCD Encoder

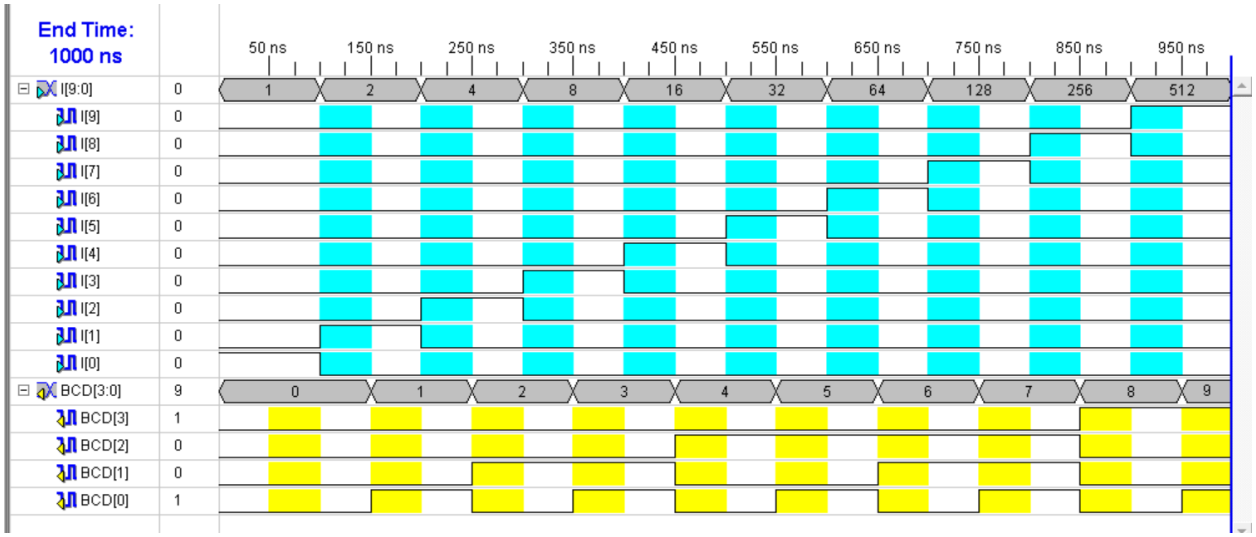
- **Truth Table:**

INPUTS										OUTPUTS			
Y9	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	A3	A2	A1	A0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1

- **Source Code:**

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

entity Decimal_to_BCD is
    Port (
        I : in STD_LOGIC_VECTOR(9 downto 0); -- 10 input lines (decimal 0-9)
        BCD : out STD_LOGIC_VECTOR(3 downto 0) -- 4-bit BCD output
    );
end Decimal_to_BCD;

architecture Behavioral of Decimal_to_BCD is
begin
    process(I)
    begin
        case I is
            when "0000000001" => BCD <= "0000"; -- Decimal 0
            when "0000000010" => BCD <= "0001"; -- Decimal 1
            when "0000000100" => BCD <= "0010"; -- Decimal 2
            when "0000001000" => BCD <= "0011"; -- Decimal 3
            when "0000010000" => BCD <= "0100"; -- Decimal 4
            when "0000100000" => BCD <= "0101"; -- Decimal 5
            when "0001000000" => BCD <= "0110"; -- Decimal 6
            when "0010000000" => BCD <= "0111"; -- Decimal 7
            when "0100000000" => BCD <= "1000"; -- Decimal 8
            when "1000000000" => BCD <= "1001"; -- Decimal 9
            when others => BCD <= "0000"; -- Default case
        end case;
    end process;
end Behavioral;
```

- **Test Bench Waveform:**



4. **Priority Encoder**:
    - Handles multiple active inputs by encoding the highest-priority active input.
    - Often includes a "valid" output signal to indicate if any input is active.

Example: 4-to-2 Priority Encoder

Inputs and Outputs:

- **Inputs**: I_3, I_2, I_1, I_0.
- **Outputs**:
    - O_1, O_0: 2-bit binary output representing the highest-priority active input.
    - V: Valid signal indicating if any input is active.

- **Truth Table:**

I_3	I_2	I_1	I_0	O_1	O_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

- X: Don't care condition.

➢ Applications of Priority Encoders

1. **Interrupt Controllers**:
    - Prioritize multiple interrupt requests in a microcontroller or processor.
2. **Data Compression**:
    - Encode sparse input signals into a compact representation.
3. **Resource Allocation**:
    - Assign resources based on priority in systems with limited availability.

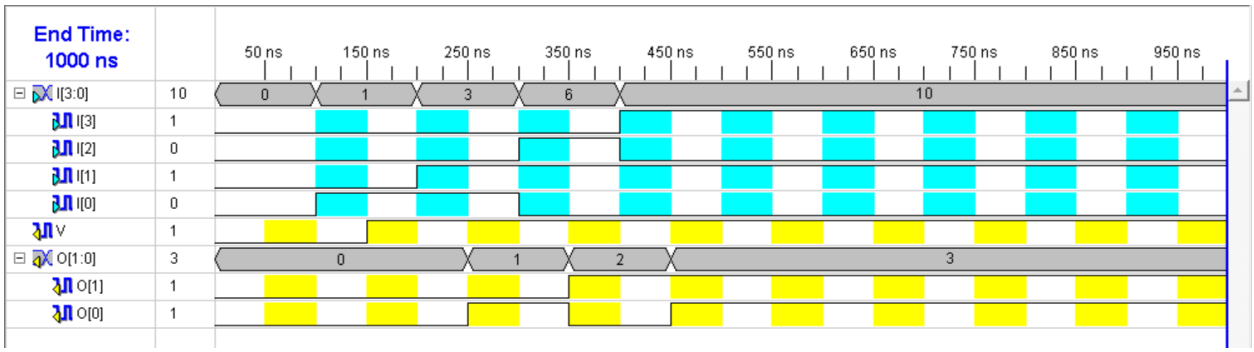
➢ Example:

- If I2 and I0 are high (I2 = 1, I0 = 1, I1 = 0, I3 = 0), the output will be based upon **I2**, since I2 has a higher priority than I0.
- If I3 and I1 are high (I3 = 1, I1 = 1, I0 = 0, I2 = 0), the output will be based upon **I3**, because I3 has the highest priority.

- **Source Code:**

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

entity priority_encoder_4to2 is
    Port ( I : in STD_LOGIC_VECTOR (3 downto 0); -- 4 input lines (I0 to I3)
           O : out STD_LOGIC_VECTOR (1 downto 0); -- 2-bit binary output (O1, O0)
           V : out STD_LOGIC); -- Valid output
end priority_encoder_4to2;
architecture Behavioral of priority_encoder_4to2 is
begin
    process(I)
    begin
        -- Default values
        O <= "00";
        V <= '0';
        -- Priority encoding logic (I3 > I2 > I1 > I0)
        if I(3) = '1' then
            O <= "11";  -- If I3 is high, output 11
            V <= '1';   -- Valid signal is high
        elsif I(2) = '1' then
            O <= "10";  -- If I2 is high, output 10
            V <= '1';   -- Valid signal is high
        elsif I(1) = '1' then
            O <= "01";  -- If I1 is high, output 01
            V <= '1';   -- Valid signal is high
        elsif I(0) = '1' then
            O <= "00";  -- If I0 is high, output 00
            V <= '1';   -- Valid signal is high
        else
            V <= '0';   -- If no input is high, valid signal is low
        end if;
    end process;
end Behavioral;
```

- **Test Bench Waveform:**



✓ **Summary of Encoders and Their Characteristics:**

Encoder Type	Inputs	Outputs	Key Features	Applications
Binary Encoder	2^n inputs	n-bit code	Encodes one active input at a time	Data compression, digital systems
Priority Encoder	2^n inputs	n-bit code	Resolves multiple active inputs	Interrupt controllers
Decimal-to-BCD Encoder	10 inputs	4-bit BCD	Encodes decimal digits (0–9) into BCD	Numeric input encoding
Octal-to-Binary Encoder	8 inputs	3-bit binary	Encodes octal digits (0–7) into binary	Octal systems
Rotary Encoder	Rotational input	Analog/Digital	Measures angular position or speed	Robotics, motor control
Quadrature Encoder	Rotational input	Digital (A & B)	Detects direction and speed of rotation	Servo motors, CNC systems
Optical Encoder	Position/Movement	Digital/Analog	Uses optical sensing for precise measurements	Industrial automation
Magnetic Encoder	Position/Movement	Digital/Analog	Uses magnetic fields, robust in harsh environments	Automotive, industrial systems

2. Decoder:

A **decoder** is a combinational logic circuit that converts binary input signals into a unique pattern of outputs. It has **n** input lines and **2^n** output lines, where each output corresponds to one specific combination of the inputs.

➢ Key Features of a Decoder:

1. **Input and Output**:
   - For **n** inputs, the decoder generates **2^n** outputs.
   - Each output corresponds to one binary combination of the inputs.
   - Only one output is active (high) at a time for a given input combination.
2. **Operation**:
   - A decoder performs the reverse operation of an encoder, translating binary input codes into distinct output lines.
   - Commonly used in memory address decoding, data demultiplexing, and enabling devices.

✓ Summary of Decoders and Their Characteristics:

Type	Inputs	Outputs	Key Features	Applications
Binary Decoder	n	2^n	Activates one output for each input combo	Address decoding, data routing
BCD-to-Decimal	4	10	Converts BCD to decimal output lines	Numeric input decoding
BCD-to-7-Segment	4	7	Drives 7-segment displays	Digital displays
Priority Decoder	n	2^n	Resolves multiple active inputs	Interrupt handling
Demultiplexer	$n+1$	2^n	Routes single input to one of many outputs	Signal routing, communication
Line Decoder	Varies	Varies	Decodes specific line signals	Control systems
2D Decoder	Row + Col	Grid	Selects specific row-column combination	Memory addressing

➢ Example: 2-to-4 Decoder

Inputs and Outputs:

- **Inputs**: A_1,A_0 (2 bits)
- **Outputs**: O_3, O_2, O_1, O_0 (4 outputs)

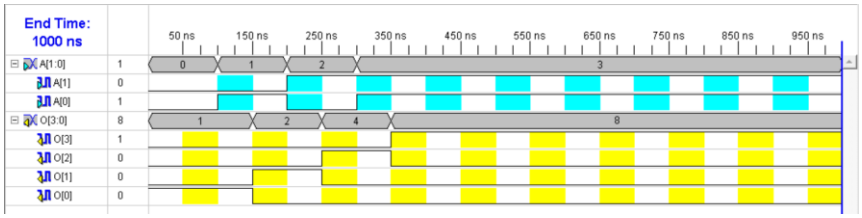
- **Truth Table:**

A_1	A_0	O_3	O_2	O_1	O_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

- ## Source Code:

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

entity Decoder2to4 is
    Port (
        A : in STD_LOGIC_VECTOR(1 downto 0); -- 2 input lines
        O : out STD_LOGIC_VECTOR(3 downto 0) -- 4 output lines
    );
end Decoder2to4;

architecture Behavioral of Decoder2to4 is
begin
    process(A)
    begin
        case A is
            when "00" => O <= "0001"; -- A = 00
            when "01" => O <= "0010"; -- A = 01
            when "10" => O <= "0100"; -- A = 10
            when "11" => O <= "1000"; -- A = 11
            when others => O <= "0000"; -- Default case
        end case;
    end process;
end Behavioral;
```

- **Test Bench Waveform:**



- ## Discussion:

In this lab, I implemented various encoders and decoders using Xilinx VHDL, including a 4:2 encoder, 10:4 encoder, 4:2 priority encoder, and a 2:4 decoder. These components are crucial for optimizing data transmission, simplifying complex signal systems, and ensuring efficient communication between embedded systems. The hands-on experience helped me understand the logic behind encoding and decoding signals, a key aspect of digital systems.

The lab also introduced me to **magnetic encoders**, which have significant applications in the automotive and electric vehicle (EV) industries. Magnetic encoders are highly reliable and durable, making them ideal for automotive systems where environmental factors like vibrations and moisture can affect performance.

▪ Conclusion:

As an aspiring electrical sub-engineer and computer engineering student passionate about the automotive embedded industry, this lab reinforced my understanding of digital encoding and decoding. The introduction to magnetic encoders has sparked my interest in how they can enhance automotive technologies, particularly in electric vehicles. This experience motivates me to further explore their applications in the automotive sector, which I plan to pursue in my future studies and career.