Decoders and Encoders
ELCTEC-131
Basic Decoder

- **Decoder**: A digital circuit designed to detect the presence of a particular digital state.
- Can have one output or multiple outputs.
Example:

- 2-Input NAND Gate detects the presence of ‘11’ on the inputs to generate a ‘0’ output.
Single-Gate Decoders

- Uses single gates (AND/NAND) and some Inverters.

- Example: 4-Input AND detects ‘1111’ on the inputs to generate a ‘1’ output.
Inputs

- Inputs are labeled $D_3$, $D_2$, $D_1$, and $D_0$, with $D_3$ the MSB (most significant bit) and $D_0$ the LSB (least significant bit).
Single-Gate Decoders

a. Active-HIGH indication

b. Active-LOW indication
Single-Gate Examples

- If the inputs to a 4-Input NAND are given as $\overline{D_1}, \overline{D_2}, \overline{D_3}, D_4$, then the NAND detects the code 0001. The output is a 0 when the code 0001 is detected.

- This type of decoder is used in Address Decoding for a PC System Board.
Multiple Output Decoders

- Decoder circuit with \( n \) inputs can activate \( m = 2^n \) load circuits.

- Called a \( n \)-line-to-\( m \)-line decoder, such as a 2-to-4 or a 3-to-8 decoder.

- Usually has an active low enable \( \overline{G} \) that enables the decoder outputs.
Truth Table for a 3-to-8 Decoder

<table>
<thead>
<tr>
<th>( \overline{G} )</th>
<th>( D_2 )</th>
<th>( D_1 )</th>
<th>( D_0 )</th>
<th>( Y_0 )</th>
<th>( Y_1 )</th>
<th>( Y_2 )</th>
<th>( Y_3 )</th>
<th>( Y_4 )</th>
<th>( Y_5 )</th>
<th>( Y_6 )</th>
<th>( Y_7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ \text{• • • • • • • • • • • • •} \]
3-to-8 Decoder

![Diagram of 3-to-8 Decoder]
Simulation

- Simulation: The verification of a digital design using a timing diagram before programming the design in a CPLD.

- Used to check the Output Response of a design to an Input Stimulus using a timing diagram.
Simulation
VHDL Binary Decoder

• Use select signal assignment statements constructs or conditional signal assignment statements constructs.
2-to-4 Decoder VHDL Entity

- Using a select signal assignment statement:

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY decode3 IS
PORT(
    d : IN STD_LOGIC_VECTOR (1 downto 0);
    y : OUT STD_LOGIC_VECTOR (3 downto 0));
END decode3;
```
In the previous slide, the Entity used a STD LOGIC Array for Inputs and Outputs.

The $Y : \text{OUT STD\_LOGIC\_VECTOR}(3 \ \text{downto} \ 0)$ is equal to $Y_3, Y_2, Y_1, Y_0$. 
Data Type

- The STD_LOGIC Data Type is similar to BIT but has added state values such as Z, X, H, and L instead of just 0 and 1.
Selected Signal Assignments

- Uses a VHDL Architecture construct called WITH SELECT.

- Format is:
  - WITH (signal input(s)) SELECT.
  - Signal input states are used to define the output state changes.
ARCHITECTURE decoder OF decode3 IS
BEGIN
  WITH d SELECT
    y <= "0001" WHEN "00",
      "0010" WHEN "01",
      "0100" WHEN "10",
      "1000" WHEN "11",
      "0000" WHEN others;
END decoder;
Decoder Architecture

- The decoder Architecture used a SELECT to evaluate \( d \) to determine the Output \( y \).

- Both \( d \) and \( y \) are defined as an Array (or bus or vector) Data Type.
Decoder Architecture

- The last state for WHEN OTHERS is added for the other logic states (Z, X, H, L, etc.).
Seven-Segment Displays

- **Seven-Segment Display:** An array of seven independently controlled LEDs shaped like an 8 that can be used to display decimal digits.
Seven-Segment Displays
Seven-Segment Displays

0 1 2 3 4
5 6 7 8 9
Common Anode Display

- Common Anode Display (CA): A seven-segment display where the anodes of all the LEDs are connected together to $V_{CC}$ and a ‘0’ turns on a segment (a to g).
Common Cathode Display

- Common Cathode Display (CC): A seven-segment display where all the cathodes are connected and tied to ground, and a ‘1’ turns on a segment.
Common Cathode Display

a. Circuit requirements for an illuminated LED

b. Common cathode

b. Common anode
Common Cathode Display
Seven-Segment Decoder/Driver – 1

- Receives a BCD (Binary Coded Decimal) 4-Bit input, outputs a BCD digit 0000 – 1001 (0 through 9).

- Generates Outputs (a–g) for each of the display LEDs.
Seven-Segment Decoder/Driver – 2

- Requires a current limit series resistor for each segment.

- Decoders for a CC-SS have active high outputs while decoders for a CA-SS have active low outputs (a to g).
The outputs generated for the binary input combinations of 1010 to 1111 are “don’t cares”.

The decoder can be designed with VHDL or MSI Logic (7447, 7448).
# SS Decoder/Driver Truth Table

<table>
<thead>
<tr>
<th>Digit</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>h0</th>
<th>h1</th>
<th>h2</th>
<th>h3</th>
<th>h4</th>
<th>h5</th>
<th>h6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
SS Decoder/Driver Truth Table

Segment a

a. K-map

b. Decoder for segment a (common anode)
ENTITY bcd_7seg IS
   PORT(
      d3, d2, d1, d0 : IN  BIT;
      h1, h2, h3, h4, h5, h6 : OUT BIT;
   )
END bcd_7seg;
ARCHITECTURE seven_segment OF bcd_7seg IS

    SIGNAL input       : BIT_VECTOR (3 downto 0);
    SIGNAL output      : BIT_VECTOR (6 downto 0);

BEGIN

    input <= d3 & d2 & d1 & d0;

    -- Uses two intermediate signals called input and output (internal no pins)

    -- Creates an array by using the concatenate operator (&)

    In this case input(3) <= d3, input(2) <= d2, etc.
WITH input SELECT

  output <= "0000001" WHEN "0000",
          "0011111" WHEN "0001",
          "0010011" WHEN "0010",
          "0000111" WHEN "0011",
          "1111111" WHEN others;
h0 <= output(6);
h1 <= output(5);
h2 <= output(4);
h3 <= output(3);
h4 <= output(2);
h5 <= output(1);
h6 <= output(0);

END seven_segment
In the preceding example file, a concurrent select signal assignment was used (WITH (signals) SELECT).

The intermediate output signals were mapped to the segments (0 to 6).
Example:

- when Input \((D_3 - D_0)\) is 0001, the decoder sets for CA display
  
  - \(h0 = h3 = h4 = h5 = h6 = 1\)
  
  - \(h1 = h2 = 0\).
A VHDL Process is a construct that encloses sequential statements that are executed when a signal in a sensitivity list changes.
Sequential Process Basic Format

- Basic Format:
  
  ```
  PROCESS(Sensitivity List)
  BEGIN
  Sequential Statements;
  END PROCESS;
  ```
Encoders

- **Encoder**: A digital circuit that generates a specific code at its outputs in response to one or more active inputs.
- It is complementary in function to a decoder.
Encoders

- Output codes are usually Binary or BCD.
Priority Encoders

- **Priority Encoder**: An encoder that generates a code based on the highest-priority input.

- For example, if input $D_3 = input D_5$, then the output is 101, not 011. $D_5$ has a higher priority than $D_3$ and the output will respond accordingly.
# 8-to-3 Encoder Truth Table

## Active High Inputs

<table>
<thead>
<tr>
<th>D_7</th>
<th>D_6</th>
<th>D_5</th>
<th>D_4</th>
<th>D_3</th>
<th>D_2</th>
<th>D_1</th>
<th>D_0</th>
<th>Q_2</th>
<th>Q_1</th>
<th>Q_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

• •
Priority Encoder Equations

\[ Q_2 = D_7 + D_6 + D_5 + D_4 \]
\[ Q_1 = D_7 + D_6 + \overline{D_5} \overline{D_4} D_3 + \overline{D_5} \overline{D_4} D_2 \]
\[ Q_0 = D_7 + \overline{D_6} D_5 + \overline{D_6} \overline{D_4} D_3 + \overline{D_6} \overline{D_4} \overline{D_2} D_1 \]
-- hi_pri8a.vhd

ENTITY hi_pri8a IS
  PORT(
    d : IN BIT_VECTOR (7 downto 0);
    q : OUT BIT_VECTOR (2 downto 0));
END hi_pri8a;
ARCHITECTURE a OF hi_pri8a IS
BEGIN
-- Concurrent Signal Assignments
  q(2) <= d(7) or d(6) or d(5) or d(4);
  q(1) <= d(7) or d(6)
         or ((not d(5)) and (not d(4)) and d(3))
         or ((not d(5)) and (not d(4)) and d(2));
  q(0) <= -- in a similar fashion
END a;
Another VHDL Encoder – 1

-- hi_pri8b.vhd

ENTITY hi_pri8a IS
PORT(
d : IN BIT_VECTOR (7 downto 0);
q : OUT INTEGER RANGE 0 TO 7);
END hi_pri8b;
ARCHITECTURE a OF hi_pri8b IS
BEGIN
encoder;
  q <=
  7 WHEN d(7) = '1' ELSE
  6 WHEN d(6) = '1' ELSE
  5 WHEN d(5) = '1' ELSE
  4 WHEN d(4) = '1' ELSE
  3 WHEN d(3) = '1' ELSE
  2 WHEN d(2) = '1' ELSE
  1 WHEN d(1) = '1' ELSE
  0;
END a;
VHDL Implementation of a Binary Decoder

- Write a VHDL file for a 3-line-to-8-line decoder with active-LOW outputs. Use a selected signal assignment statement. Save the file as \texttt{drive:qdesigns\labs\lab11\decode3to8_vhdl\decode3to8_vhdl.vhd}. Use the file to create a new project.
• Write a set of simulation criteria for the 3-line-to-8-line decoder you created in step 1 of this procedure.
• Use the simulation criteria to create a set of simulation waveforms to test the correctness of your design.

• Assign pin numbers to the VHDL decoder, as shown in Table 11.3. Compile the design again and download it to the DE1 test board.
Hexadecimal-to-Seven-Segment Decoder

- Create a hexadecimal-to-seven-segment decoder in VHDL, using the segment patterns in Figure 11.3 as a model.

- Save the VHDL file as \texttt{drive:qdesigns\labs\lab11\hex7seg\hex7seg.vhd}. Use the file to create a new project.
• Assign pin numbers.

• Compile the file and download it to the DE1 board.