DIGITAL ELECTRONICS

UNIT III - COMBINATIONAL CIRCUITS

DECODERS, ENCODERS AND HAZARDS CIRCUITS (PART 03)

ENCODERS

Encoder is just opposite of Decoder

An encoder is a combinational logic circuit that converts 2^n input lines into n output lines, where only one input is active at a time.

Encoders are used to reduce the number of bits needed to represent information, commonly in digital systems like keyboards, control panels, and memory.

- There are mainly two types:

1. Binary Encoder ✓

- Converts 2^n inputs to n outputs.
- Only one input line should be HIGH at any time.
- Example: 4-to-2 Binary Encoder

for k input lines the number of O/P lines

$$= \lceil \log_2(K) \rceil$$

Eg → $\log_2(1024) \Rightarrow$ 10 output lines

2. Priority Encoder ✓

- Allows multiple inputs to be HIGH, but gives priority to the highest-order input.
- Also includes a valid bit to indicate if the output is valid.

Applications of Encoders:

- Used to translate the decimal values to the binary in order to perform binary functions such as addition, subtraction, multiplication, etc.
- Encoders may include detecting interrupts in microprocessor applications. (e.g. Priority Encoders)
- Digital circuits – compress address lines.
- Communication systems – encode binary data for transmission.
- Memory addressing – select memory blocks.

Difference Between Encoder and Decoder

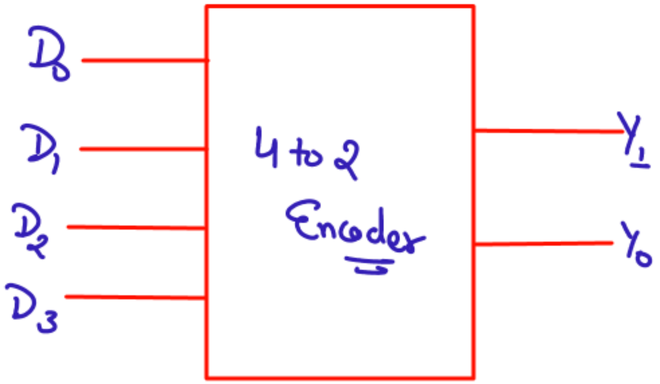
Feature	Encoder	Decoder
Input Lines	2^n input lines	n input lines
Output Lines	n output lines	2^n output lines
Purpose	Compress input to fewer bits	Expand bits to multiple outputs
Example	4×2 binary encoder	2×4 decoder

4 to 2 Encoder

A 4-to-2 binary encoder is a combinational logic circuit that encodes 4 input lines (D_0 to D_3) into 2 output lines (Y_1 and Y_0).

Inputs: D_0, D_1, D_2, D_3 (Only one is HIGH at a time)
Outputs: Y_1, Y_0 (Binary code representing the position of the HIGH input)

	A	B	C	D	E	F
	D_3	D_2	D_1	D_0	Y_1	Y_0
m_0	0	0	0	1	0	0
m_1	0	0	1	0	0	1
m_2	0	1	0	0	1	0
m_3	1	0	0	0	1	1



$$Y_1 = m_2 + m_3 = D_2 + D_3$$

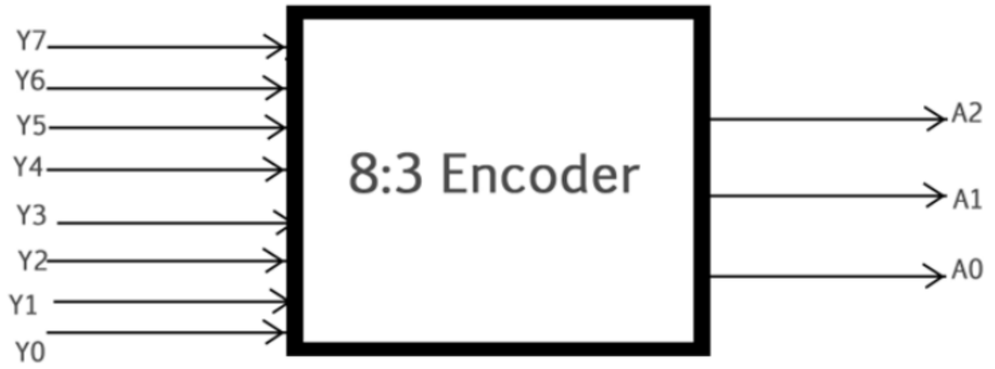
$$Y_0 = m_1 + m_3$$
$$\Rightarrow D_1 + D_3$$

Octal to Binary Encoder

The 8 to 3 Encoder or octal to Binary encoder consists of 8 inputs: Y7 to Y0 and 3 outputs: A2, A1 & A0. Each input line corresponds to each octal digit value and three outputs generate corresponding binary code.

	INPUTS								OUTPUTS		
	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	A2	A1	A0
m_0	0	0	0	0	0	0	0	1	0	0	0
m_1	0	0	0	0	0	0	1	0	0	0	1
m_2	0	0	0	0	0	1	0	0	0	1	0
m_3	0	0	0	0	1	0	0	0	0	1	1
m_4	0	0	0	1	0	0	0	0	1	0	0
m_5	0	0	1	0	0	0	0	0	1	0	1
m_6	0	1	0	0	0	0	0	0	1	1	0
m_7	1	0	0	0	0	0	0	0	1	1	1

← dont Care



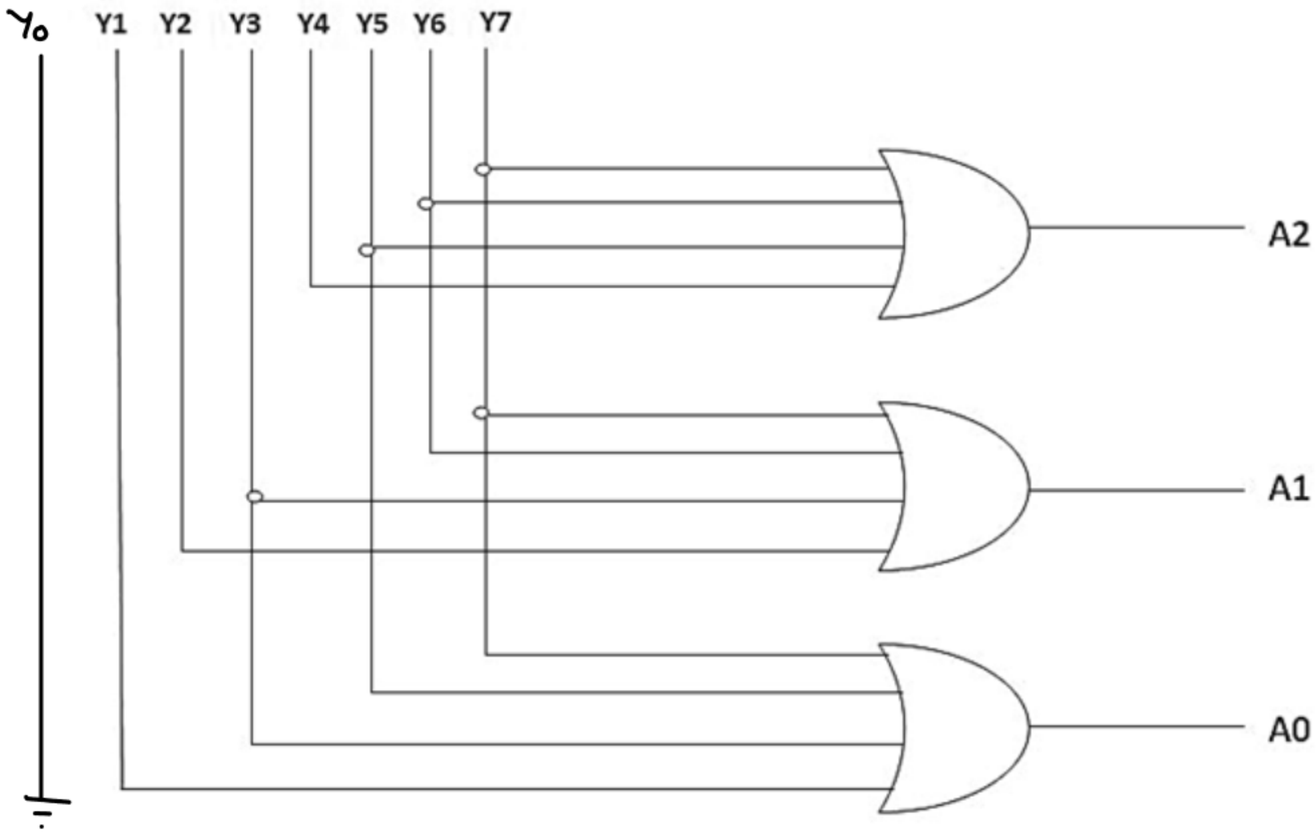
Logical Expressions:

$A2 = Y7 + Y6 + Y5 + Y4 \rightarrow \Sigma m(4,5,6,7)$

$A1 = Y7 + Y6 + Y3 + Y2 \rightarrow \Sigma m(2,3,6,7)$

$A0 = Y7 + Y5 + Y3 + Y1 \rightarrow \Sigma m(1,3,5,7)$

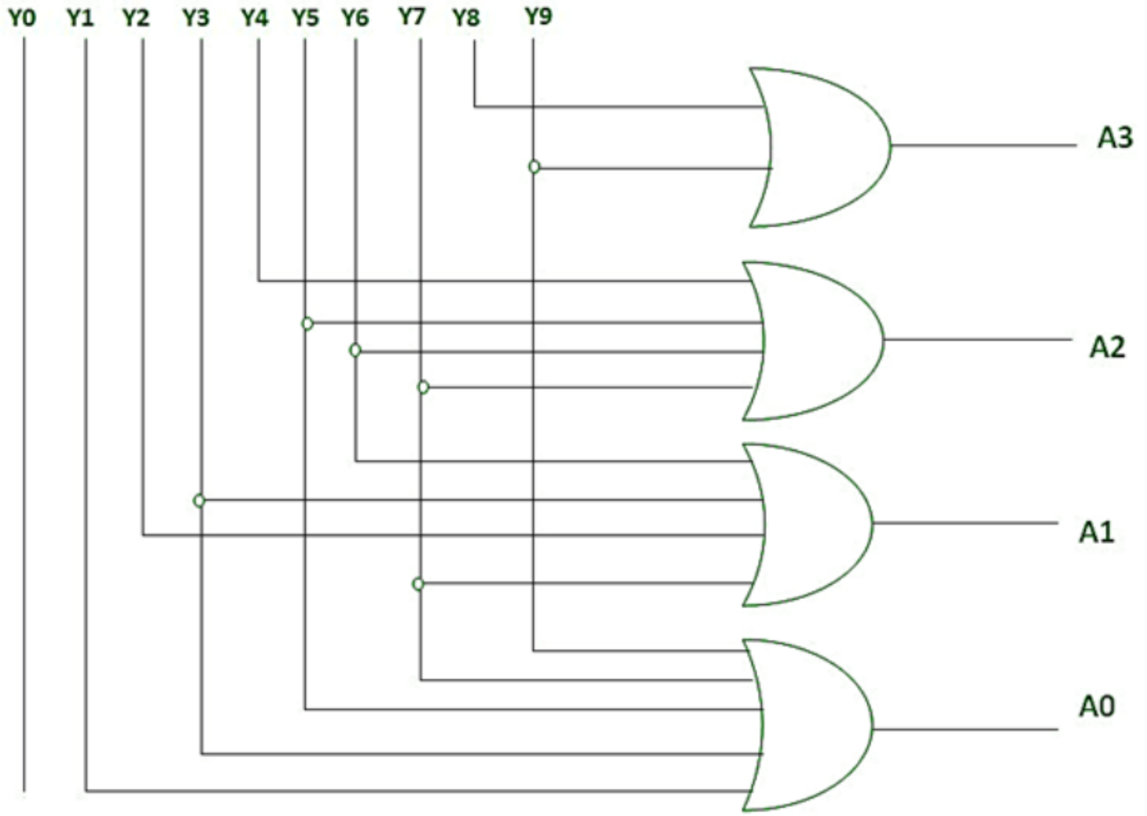
Octal to Binary Encoder



Decimal to BCD Encoders

The decimal-to-binary encoder usually consists of 10 input lines and 4 output lines.

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



Logical expression for A3, A2, A1, and A0.

A3 = Y9 + Y8

A2 = Y7 + Y6 + Y5 +Y4

A1 = Y7 + Y6 + Y3 +Y2

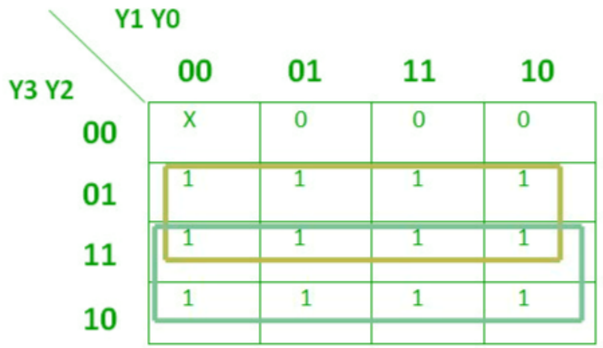
A0 = Y9 + Y7 +Y5 +Y3 + Y1

4-to-2 Priority Encoder

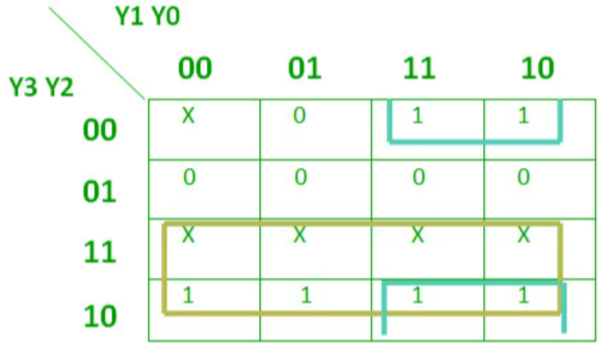
A 4 to 2 priority encoder has 4 inputs: Y3, Y2, Y1 & Y0, and 2 outputs: A1 & A0.
The input, Y3 has the highest priority, whereas the input, Y0 has the lowest priority.
In this case, even if more than one input is '1' at the same time, the output will be the (binary) code corresponding to the input, which is having higher priority.

	INPUTS				OUTPUTS		
	Y3	Y2	Y1	Y0	A1	A0	V
	0	0	0	0	X	X	0
m_0	0	0	0	1	0	0	1
m_1	0	0	1	X	0	1	1
m_2	0	1	X	X	1	0	1
m_3	1	X	X	X	1	1	1

LSB, MSB, Valid, Invalid, decoder



A1=Y3 + Y2



A0=Y3 + Y2' Y1

$m_2 + m_3$

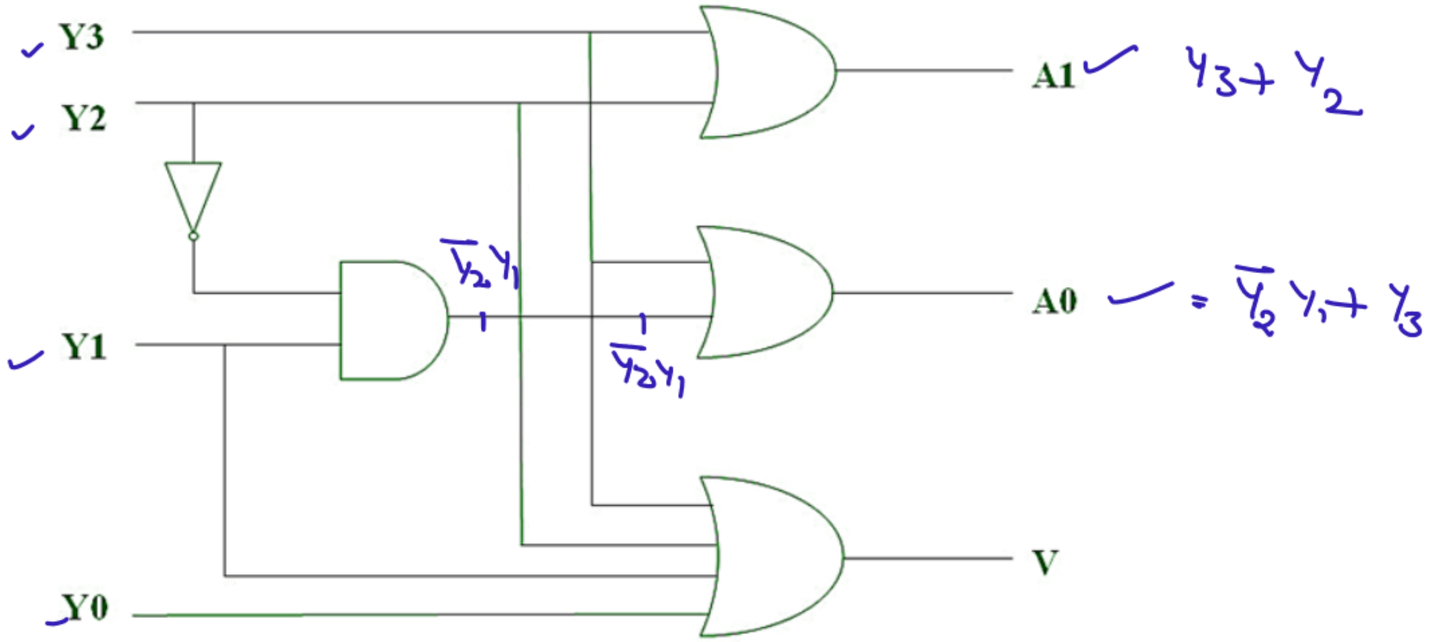
01XX

1XXX

⇒ [0100 0101 0110 0111] [1000 1001 1010 1011 1100 1101 1110 1111]

Com SOP ⇒ 4 5 6 7 8 9 10 11 12 13 14 15

4-to-2 Priority Encoder



ELECTRICAL HAZARDS IN DIGITAL CIRCUITS

ELECTRICAL HAZARDS IN DIGITAL CIRCUITS

A hazard is an unwanted or momentary fluctuation (glitch or disturbance) in the output of a combinational logic circuit, even when the inputs change in a way that should not change the output.

These glitches occur due to different signal propagation delays through the logic gates.

🔍 Why Do Hazards Occur?

- Logic gates take **time to switch** from 0 to 1 or 1 to 0.
- If multiple paths exist from input to output, and they have **different delays**, temporary glitches may appear during transitions.
- These **do not affect the logic design** but can affect **real-time behavior** (e.g., in clocked circuits, memory writes, control units).

🧩 Types of Hazards

1. Static Hazard

- **Static-1 Hazard**: Output **should stay at 1**, but temporarily goes to **0**.
- **Static-0 Hazard**: Output **should stay at 0**, but temporarily goes to **1**.

} $f = A\bar{B} + BC$ ⟵ time graph

2. Dynamic Hazard

- Occurs when output **changes multiple times** instead of once when inputs change.
- Usually happens in circuits with **more than two paths** and **multiple delays**.

Example:
Expected transition: Output goes from **0 → 1**

But due to delays, it goes like:
0 → 1 → 0 → 1 (or more times)

🌀 **Multiple transitions → Dynamic Hazard**

3. Functional Hazard

- Caused when **multiple inputs change simultaneously**.
- Rare, difficult to predict.
- Happens when transitions happen in such a way that two stable states are bypassed due to unequal delays.

⚠️ Hazard Detection

1. 🗺️ Use K-map to find ungrouped adjacent 1s (or 0s).
2. 🔁 Check transitions between minterms not in the same group.
3. 📉 Observe glitches using waveform simulation tools.
4. ⏱️ Analyze delay paths and multiple input changes.

🛠️ Hazard Elimination

1. ➕ Add redundant groups in the K-map.
2. 🧮 Use SOP form to avoid static-1 hazards.
3. 📘 Use POS form to avoid static-0 hazards.
4. ⏳ Balance delay paths using buffers.
5. ⏰ Use clocked (synchronous) circuits to ignore glitches.
6. 🖌️ Simplify logic — reduce gate levels and critical paths.