Code Converter Design

- A **code converter** transforms one internal representation of data to another.
- **Problem:** Design a BCD to Excess-3 code converter.
- **Specification: BCD code:** 4-bit patterns "0000" to "1001 for digits 0 to 9; **Excess-3 code:** BCD code + "0011" for digits 0 to 9.
- Truth table for code converter:

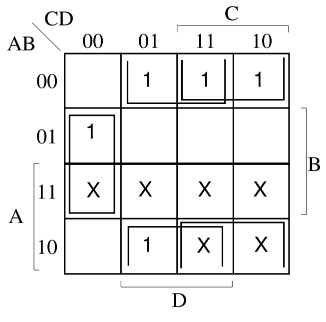
Decimal Digit	Input BCD				Output Excess–3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Code Converter

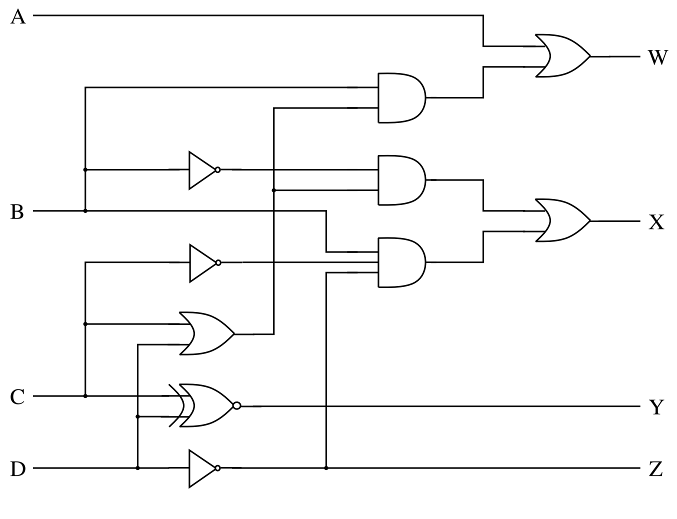
- Equations: (share terms to minimize the implementation cost)

$$\begin{aligned}
W &= A + BC + BD = A + B(C + D) \\
X &= \bar{B}C + \bar{B}D + B\bar{C}\bar{D} = \bar{B}(C + D) + B\bar{C}\bar{D} \\
Y &= CD + \bar{C}\bar{D} = \overline{C \oplus D} \\
Z &= \bar{D}
\end{aligned}$$

- Implement the network:



X=$\bar{B}$C+$\bar{B}$D+B $\bar{C}$ $\bar{D}$

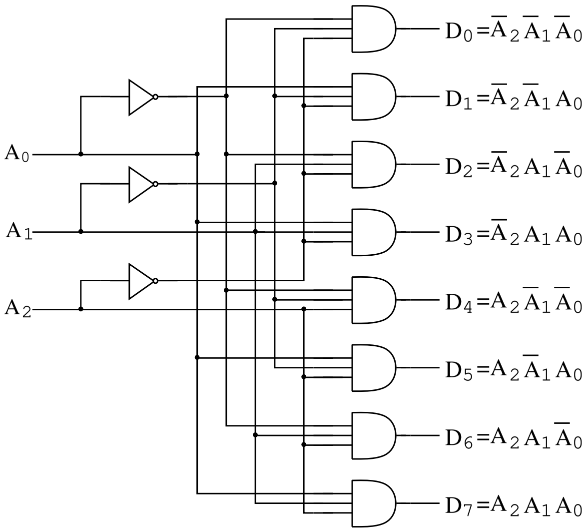


Decoder

- A **decoder** converts n binary bits to a maximum of 2^n unique output lines.

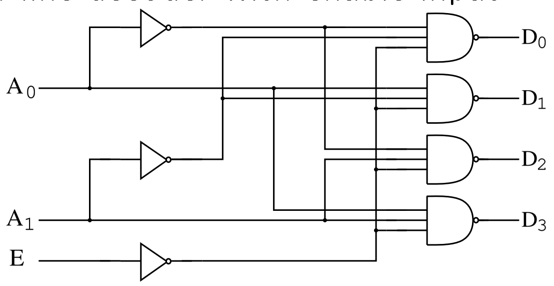
- Decoders implement **minterms** directly.

Inputs			Outputs							
A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
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



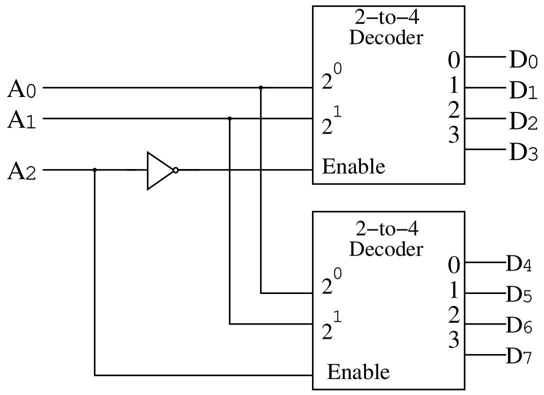
Decoder with Enable Input

- A 2-to-4 line decoder with enable input



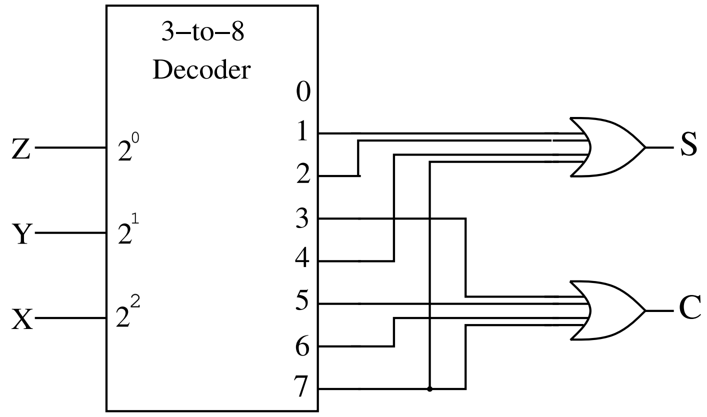
E	A_1	A_0	D_0	D_1	D_2	D_3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	X	X	1	1	1	1

- Construct a 3-to-8 decoder from 2-to-4 decoders



Circuit Implemented by a Decoder

- Decoders implement **minterms** directly.
- Use a decoder and an **OR** gate to form sum-of-minterms directly.
- Exp: $S(X,Y,Z) = \sum_m(1,2,4,7)$; $C(X,Y,Z) = \sum_m(3,5,6,7)$



Encoder

- **Encoders** perform the "inverse" operation of decoders, taking a code in one format and encoding it into another format.
- Exp: Octal-to-binary encoder

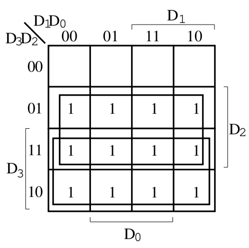
$$\begin{aligned} A_0 &= D_1 + D_3 + D_5 + D_7 \\ A_1 &= D_2 + D_3 + D_6 + D_7 \\ A_2 &= D_4 + D_5 + D_6 + D_7 \end{aligned}$$

Inputs								Outputs		
D7	D6	D5	D4	D3	D2	D1	D0	A2	A1	A0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

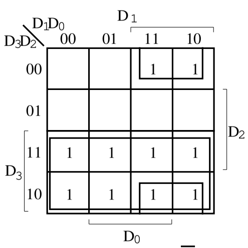
Priority Encoder

- **Priority encoders**: If two or more inputs are equal to 1 at the same time, the input having the highest priority takes precedence.

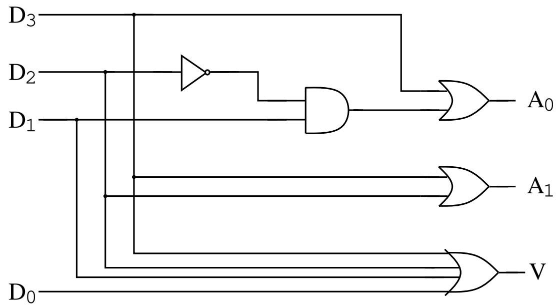
Inputs				Outputs		
D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1



$A_1 = D_2 + D_3$



$A_0 = D_3 + D_1\bar{D}_2$



Multiplexer

- A **multiplexer** uses n binary select bits to choose from a maximum of 2^n unique input lines.
- Like a decoder, it decodes **minterms** internally; unlike a decoder, it has only **one output line**.
- The output of the multiplexer is the data input whose index is specified by the n-bit code.

S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3