CSE_2323

Combinational Logic

Digital Logic Design

Course code: CSE-2323
Credit Hour: 3

Md. Mujibur Rahman Maruf

Adjunct Lecturer,

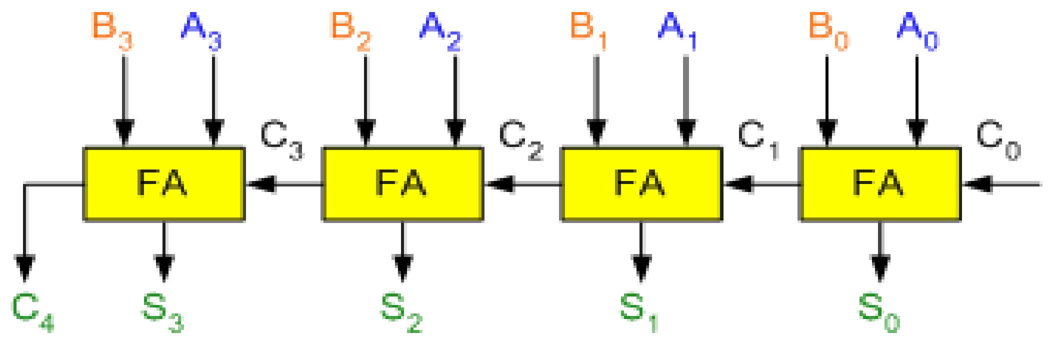
Dept. of CSE, IIUC

Mujiburmaruf.cuet17@gmail.com

International Islamic University Chittagong(IIUC)

Carry look ahead adder

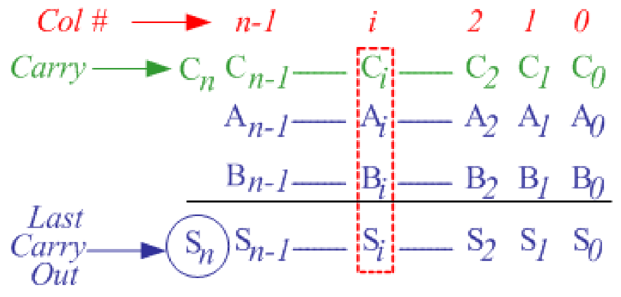
In ***ripple carry adders***, the carry propagation time is the major speed limiting factor

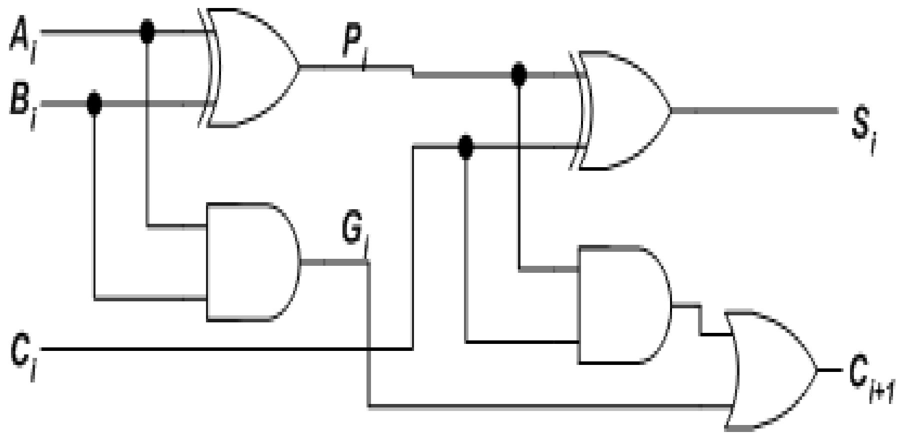

B_3 A_3 B_2 A_2 B_1 A_1 B_0 A_0
C_3 C_2 C_1 C_0
FA FA FA FA
C_4 S_3 S_2 S_1 S_0

carry look-ahead solves carry propagation delay of adders by calculating the carry signals in advance, based on the input signals.

Carry look ahead adder

let's consider the case of adding two *n-bit* numbers ***A*** and ***B***


Col #
n-1
i
2
1
0
Carry
C_n C_{n-1} C_i C_2 C_1 C_0
A_{n-1} A_i A_2 A_1 A_0
B_{n-1} B_i B_2 B_1 B_0
Last Carry Out
S_n S_{n-1} S_i S_2 S_1 S_0



The Figure shows the full adder circuit used to add the operand bits in the *i*th column; *namely A_i & B_i and the carry bit coming from the previous column (C_i).*

Carry look ahead adder

In this circuit, the 2 internal signals P_i and G_i are given by:

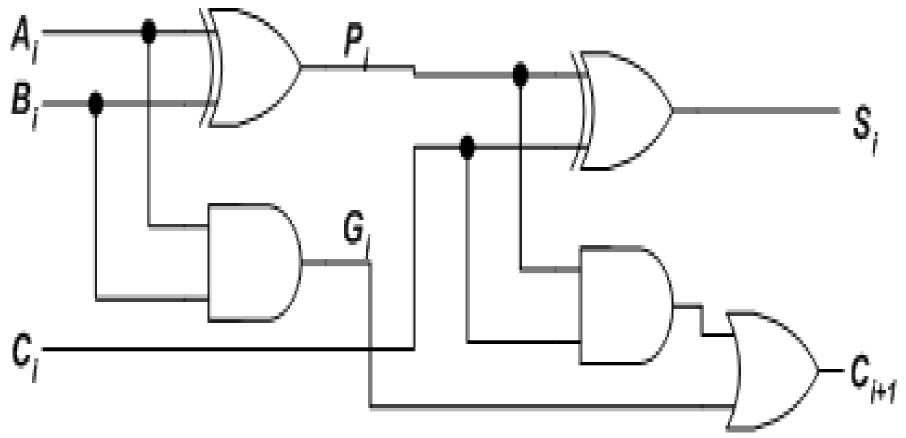
$P_i = A_i \oplus B_i$ **(1)**

$G_i = A_i B_i$..........................**(2)**

The output sum and carry can be defined as :

$S_i = P_i \oplus C_i$ **(3)**

$C_{i+1} = G_i + P_i C_i$ **(4)**



G_i is known as the ***carry Generate*** signal since a carry (C_{i+1}) is generated whenever G_i =1, regardless of the input carry (C_i).

P_i is known as the ***carry propagate*** signal since whenever P_i =1, the input carry is propagated to the output carry, i.e., C_{i+1} = C_i (note that whenever P_i =1, G_i =0).

Carry look ahead adder

Computed values of ***all*** the P_i's are valid one XOR-gate delay after the operands A and B are made valid.

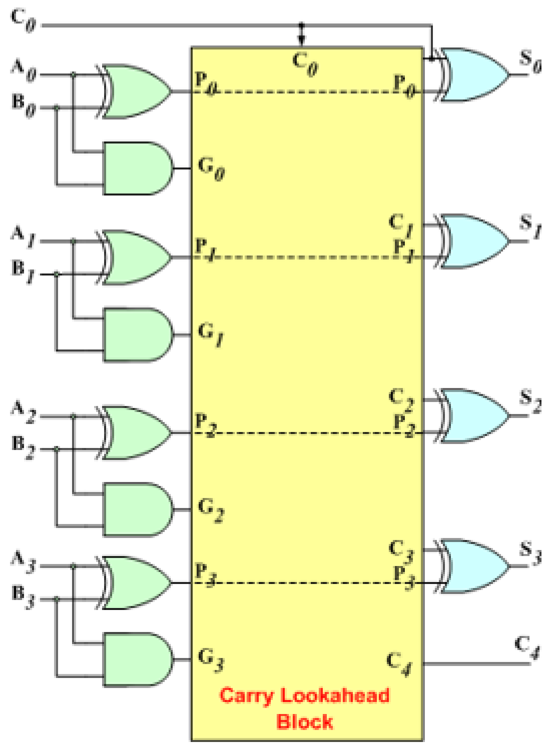
Computed values of ***all*** the G_i's are valid one AND-gate delay after the operands A and B are made valid.

The Boolean expression of the carry outputs of various stages can be written as follows:

$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1C_1 = G_1 + P_1\,(G_0 + P_0C_0)$$
$$= G_1 + P_1G_0 + P_1P_0C_0$$
$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 +$$
$$P_2P_1G_0 + P_2P_1P_0C_0\,C_4 = G_3 + P_3C_3$$
$$= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$$

In general, the $i^{th.}$ carry output is expressed in the form $C_i = F_i$ (P's, G's , C_0).

In other words, each carry signal is expressed as a direct SOP function of C_0 rather than its preceding carry signal



Carry look ahead adder

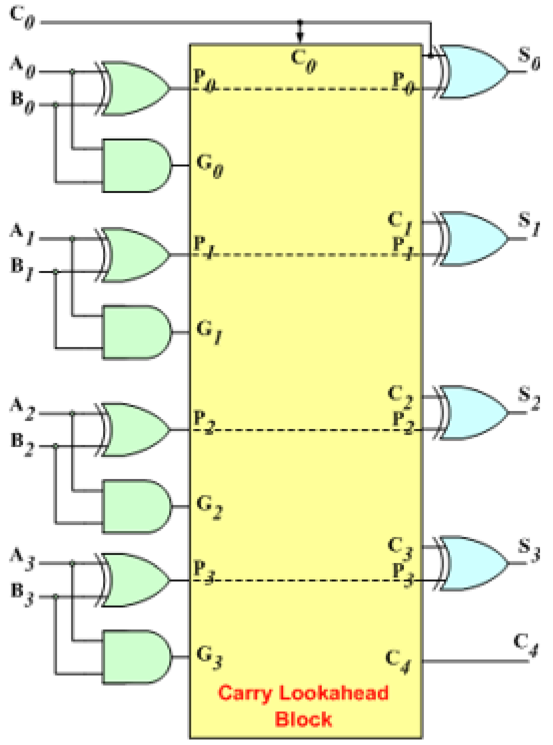
First level: Generates all the P & G signals. Four sets of P & G logic (each consists of an XOR gate and an AND gate). Output signals of this level (P's & G's) will be valid after 1t.

Second level: The Carry Look-Ahead (CLA) logic block which consists of four 2-level implementation logic circuits. It generates the carry signals (C_1, C_2, C_3, and C_4) as defined by the above expressions. Output signals of this level (C_1, C_2, C_3, and C_4) will be valid after 3t.

Third level: Four XOR gates which generate the sum signals (S_i) ($S_i = P_i$ Å C_i). Output signals of this level (S_0, S_1, S_2, and S_3) will be valid after 4t.

Thus, the 4 Sum signals (S_0, S_1, S_2 & S_3) will all be valid after a total delay of 4t compared to a delay of $(2n+1)$t for Ripple Carry adders.

For a 4-bit adder ($n = 4$), the Ripple Carry adder delay is 9t.

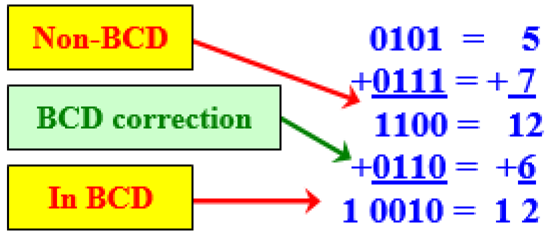


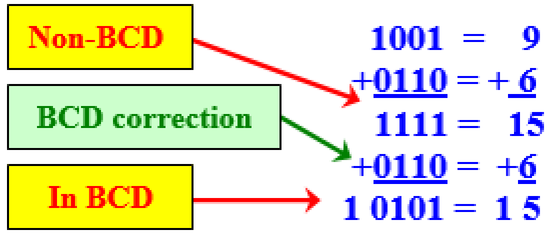
BCD Adder

Four bits are needed to represent all BCD digits (0 – 9). But with four bits we can represent up to 16 values (0000 through 1111). The extra six values (1010 through 1111) are **not valid** BCD digits

A BCD adder is a circuit that adds two BCD digits in parallel and produces a sum BCD digit and a carry out bit.

The maximum sum result of a *BCD input* adder can be 19. As maximum number in BCD is 9 and may be there will be a carry from previous stage also, so 9 + 9 + 1 = 19





BCD Adder

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

- When binary sum $\leq$ 1001, identical binary and BCD
  - **No conversion needed**
- When binary sum $>$ 1001, non-valid binary corresponds to BCD number.
  - **Correction required**
  - Add $(0110)_2$ or $(6)_{10}$ to binary sum for correct BCD representation
  - This addition produces output carry.

Boolean function for correction:

$$C = K + Z_8Z_4 + Z_8Z_2$$

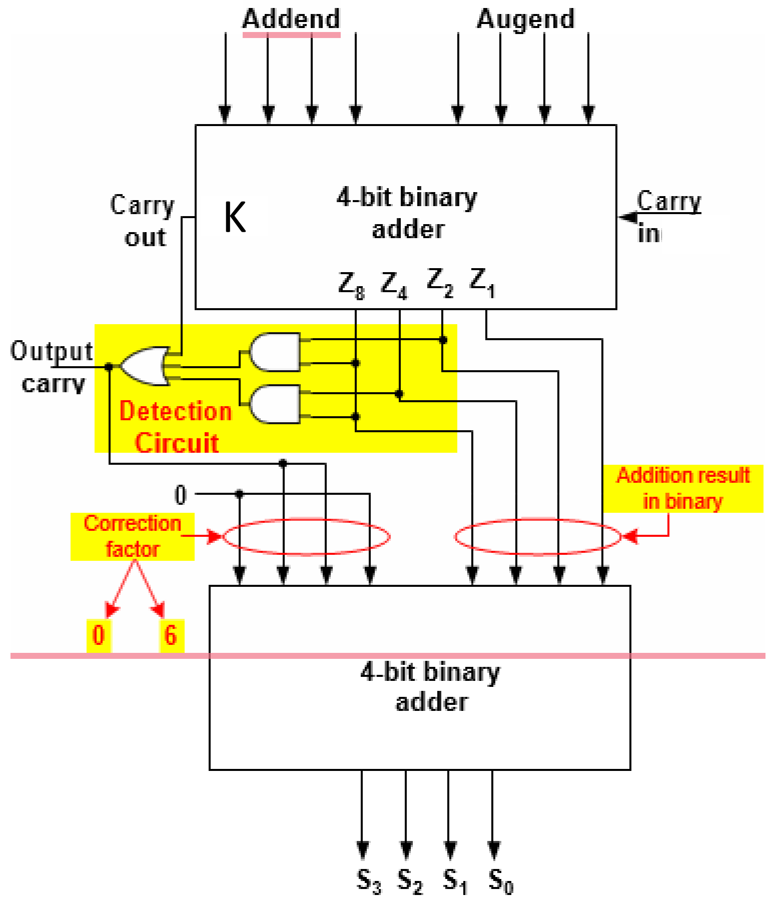
1. C=0, no addition to binary sum.
2. C=1, add $(0110)_2$ via bottom adder.

BCD Adder

The circuit of the BCD adder will be as shown in the figure.

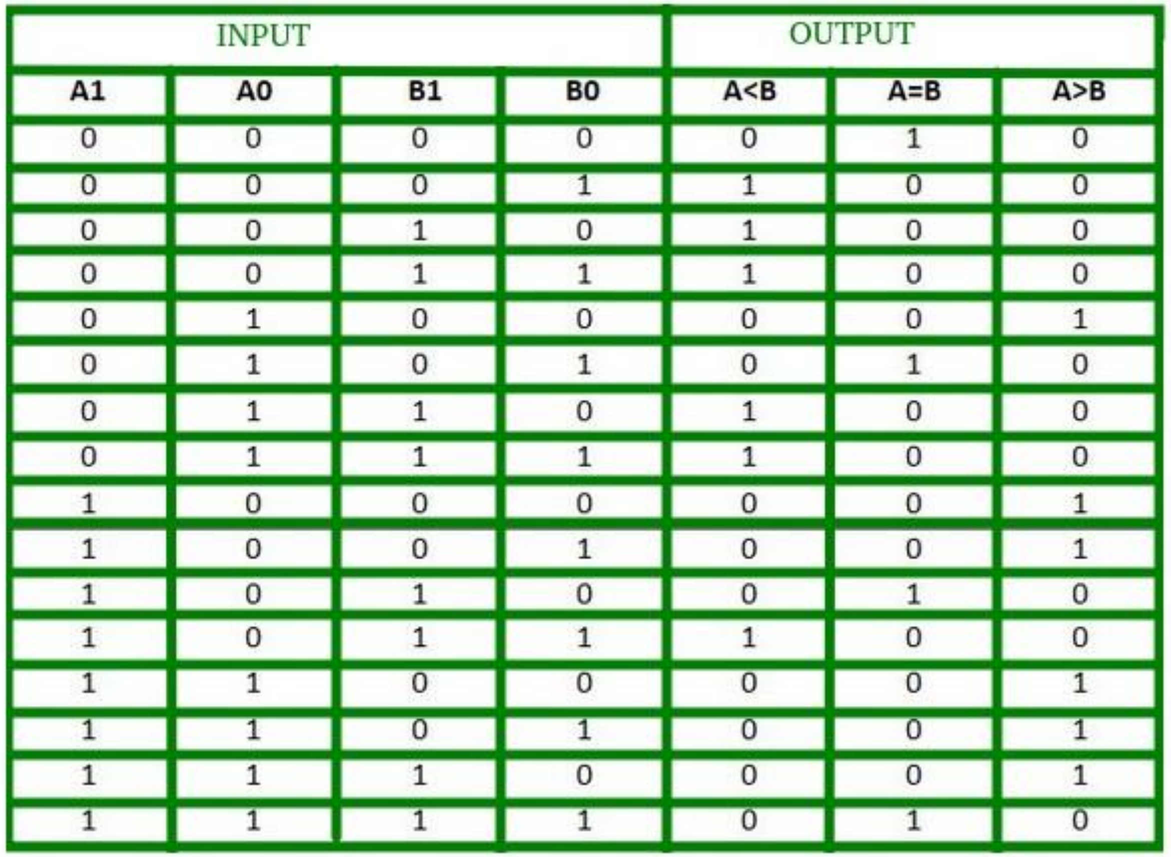
- When the **Output carry** is equal to **zero**, the correction factor equals zero.

When the **Output carry** is equal to **one**, the correction factor is 0110.



2 Bit Magnitude Comparator

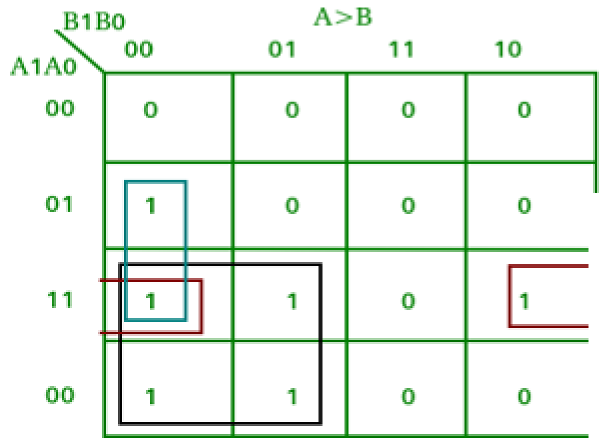
2 Bit Magnitude Comparator – A comparator used to compare two binary numbers each of two bits is called a 2-bit Magnitude comparator. It consists of four inputs and three outputs to generate less than, equal to and greater than between two binary numbers. The truth table for a 2-bit comparator is given below:



INPUT				OUTPUT		
A1	A0	B1	B0	A<B	A=B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

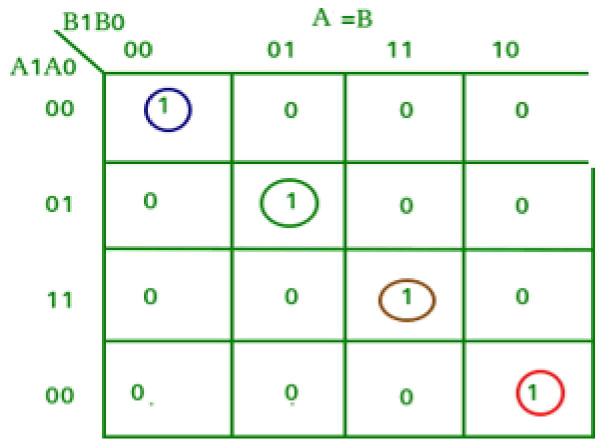
2 Bit Magnitude Comparator

From the above K-maps logical expressions for each output can be expressed as follows



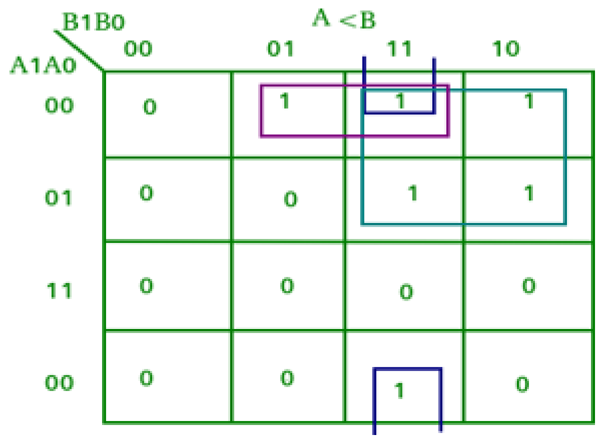
A>B

B1B0 / A1A0	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
00	1	1	0	0



A =B

B1B0 / A1A0	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
00	0	0	0	1



A <B

B1B0 / A1A0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
00	0	0	1	0

A>B: $A_1B_1' + A_0B_1'B_0' + A_1A_0B_0'$

A=B: $A_1'A_0'B_1'B_0' + A_1'A_0B_1'B_0 + A_1A_0B_1B_0 + A_1A_0'B_1B_0'$

: $A_1'B_1'(A_0'B_0' + A_0B_0) + A_1B_1(A_0B_0 + A_0'B_0')$

: $(A_0B_0 + A_0'B_0')(A_1B_1 + A_1'B_1')$

: (A_0 Ex-Nor B_0) (A_1 Ex-Nor B_1)

$$\overline{A0 \oplus B0} \cdot \overline{A1 \oplus B1}$$

A<B: $A_1'B_1 + A_0'B_1B_0 + A_1'A_0'B_0$

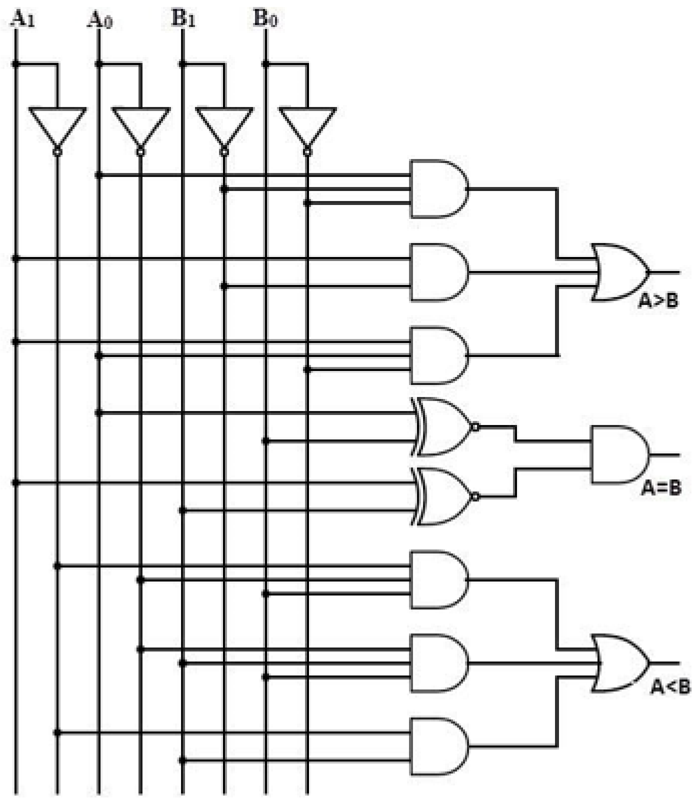
2 Bit Magnitude Comparator

By using these Boolean expressions, we can implement a logic circuit for this comparator.

$$A>B = A_1B_1' + B_0\,A_0B_1' + B_0\,A_0A_1$$

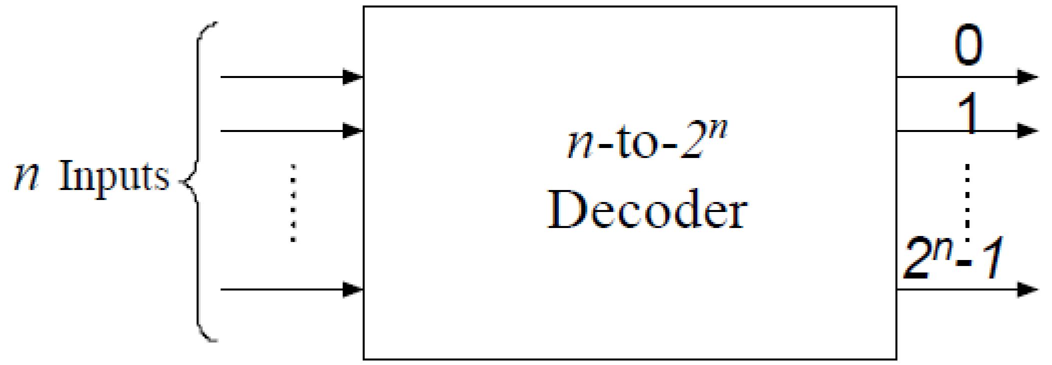
$$A=B = \overline{A0 \oplus B0}\;.\;\overline{A1 \oplus B1}$$

$$A<B = B_1A_1' + B_0B_1A_0' + A_1'A_0'B_0$$

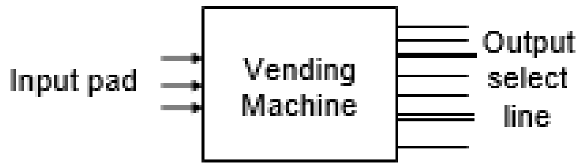


Decoder

- A Decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines



Consider a vending machine that takes 3 bits as input and releases a single product, out of the available 8 product sort



2-to-4 decoders

Decimal #	Input		Output			
	A_1	A_0	D_0	D_1	D_2	D_3
0	0	0	1	0	0	0
1	0	1	0	1	0	0
2	1	0	0	0	1	0
3	1	1	0	0	0	1

Truth table for 2-to-4 decoder

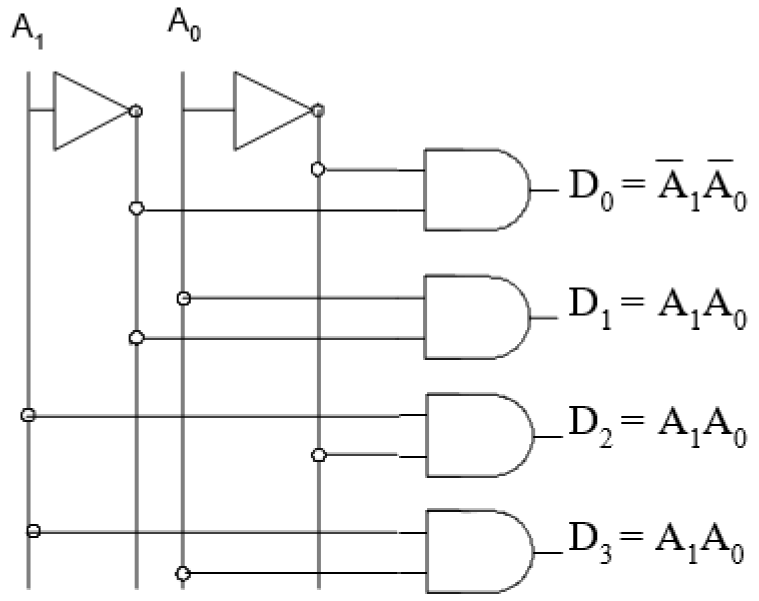
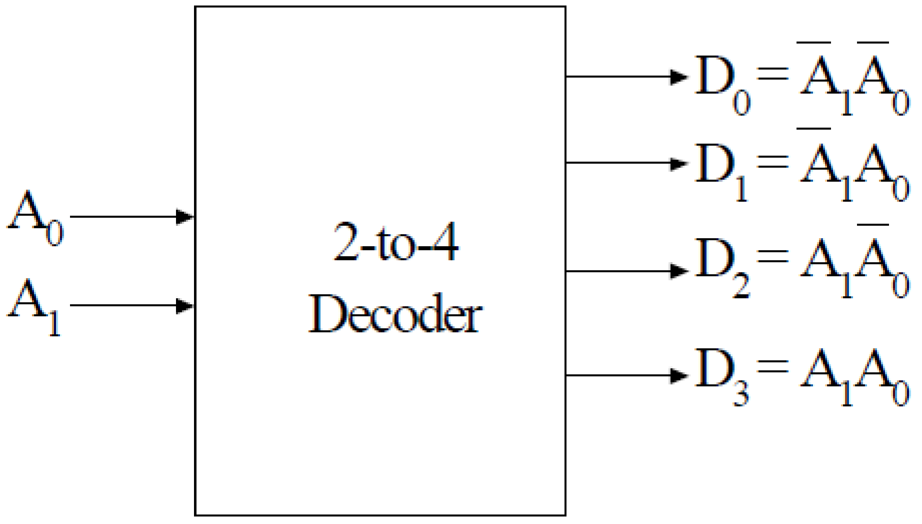


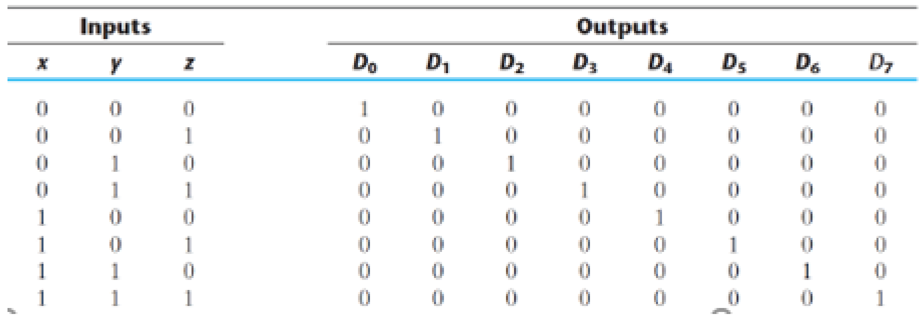
Figure 3: Implementation 2-to-4 decoder



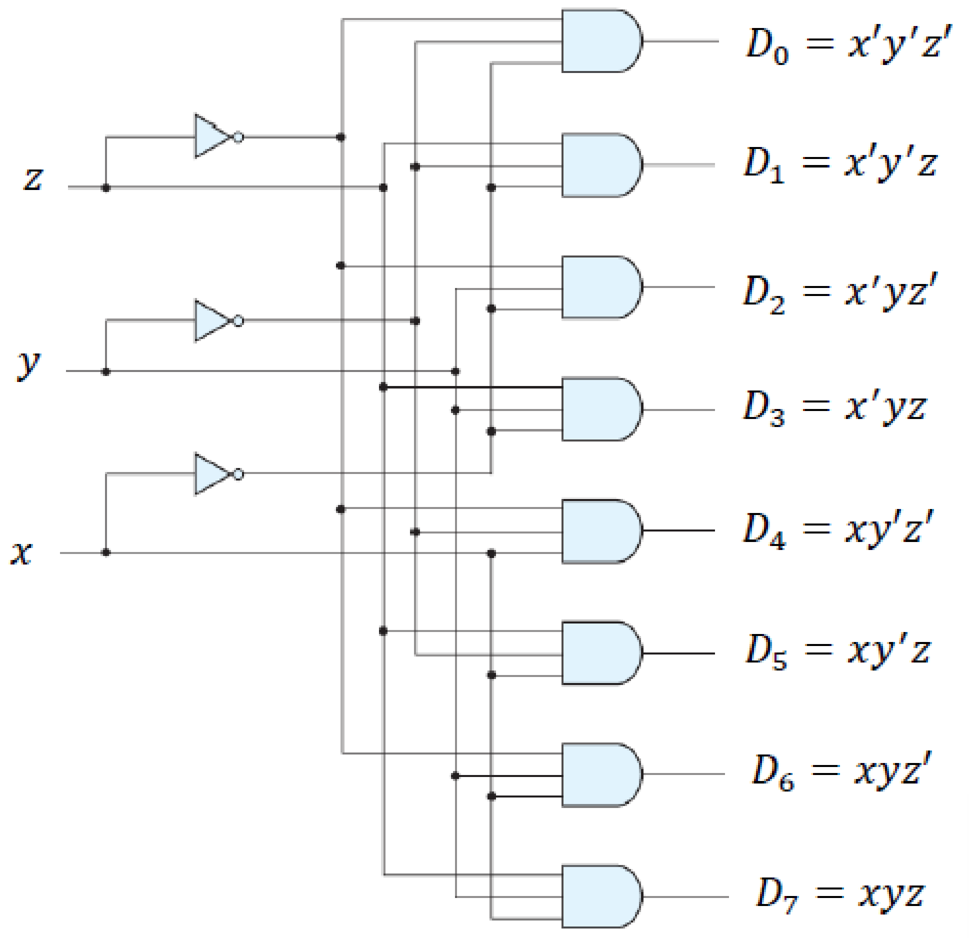
A_0
A_1
2-to-4 Decoder
$D_0 = \overline{A}_1\overline{A}_0$
$D_1 = \overline{A}_1A_0$
$D_2 = A_1\overline{A}_0$
$D_3 = A_1A_0$

3×8 DECODER

- A 3 × 8 line decoder decodes 3 input bits into one of 8 possible outputs x
- Each output representsone of the minterms of the 3 input variables



Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



2-to-4 decoder with enable

Decimal value	Enable	Inputs		Outputs			
	E	A_1	A_0	D_0	D_1	D_2	D_3
	0	X	X	0	0	0	0
0	1	0	0	1	0	0	0
1	1	0	1	0	1	0	0
2	1	1	0	0	0	1	0
3	1	1	1	0	0	0	1

Table 2: Truth table of 2-to-4 decoder with enable

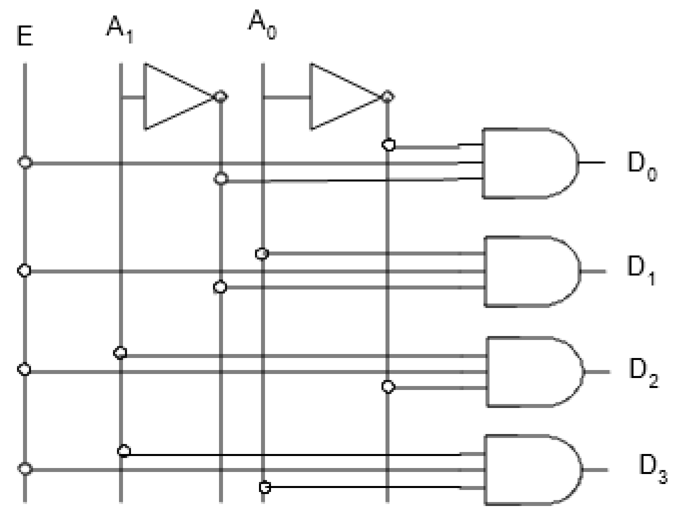
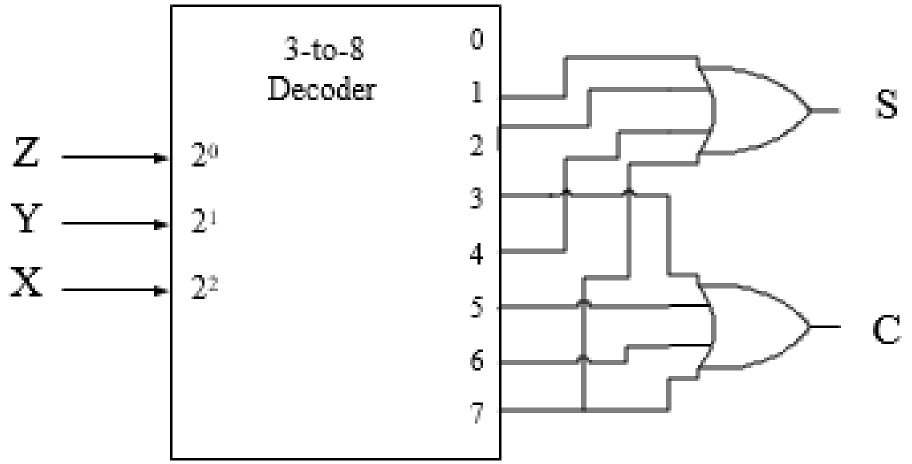


Figure 4: Implementation 2-to-4 decoder with enable

Decoder implementation of a Full Adder

Decimal value	Input			Output	
	X	Y	Z	S	C
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

Truth table of the Full Adder

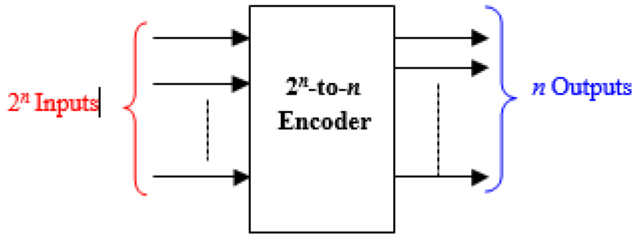


The output functions S & C can be expressed in sum-of-minterms forms as follows:

$$S(X,Y,Z)=\sum(1,2,4,7)$$
$$C(X,Y,Z)=\sum(3,5,6,7)$$

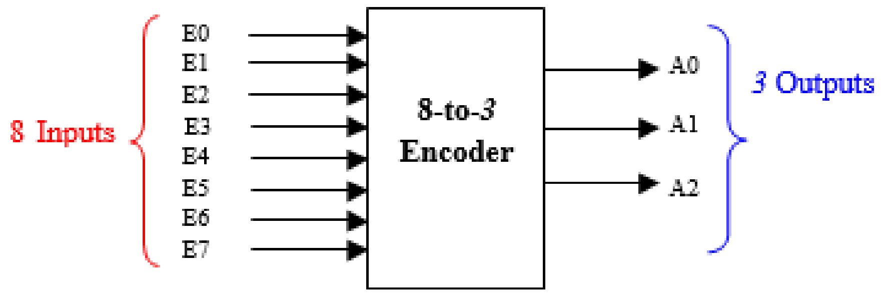
ENCODERS

- An encoder is a digital circuit that performs the inverseoperation of a decoder
- An encoder has 2^n input lines and n output lines
- The output lines generate the binary equivalent of the input line whose value is 1



A typical Encoder

8×3 OCTAL-TO-BINARY ENCODER

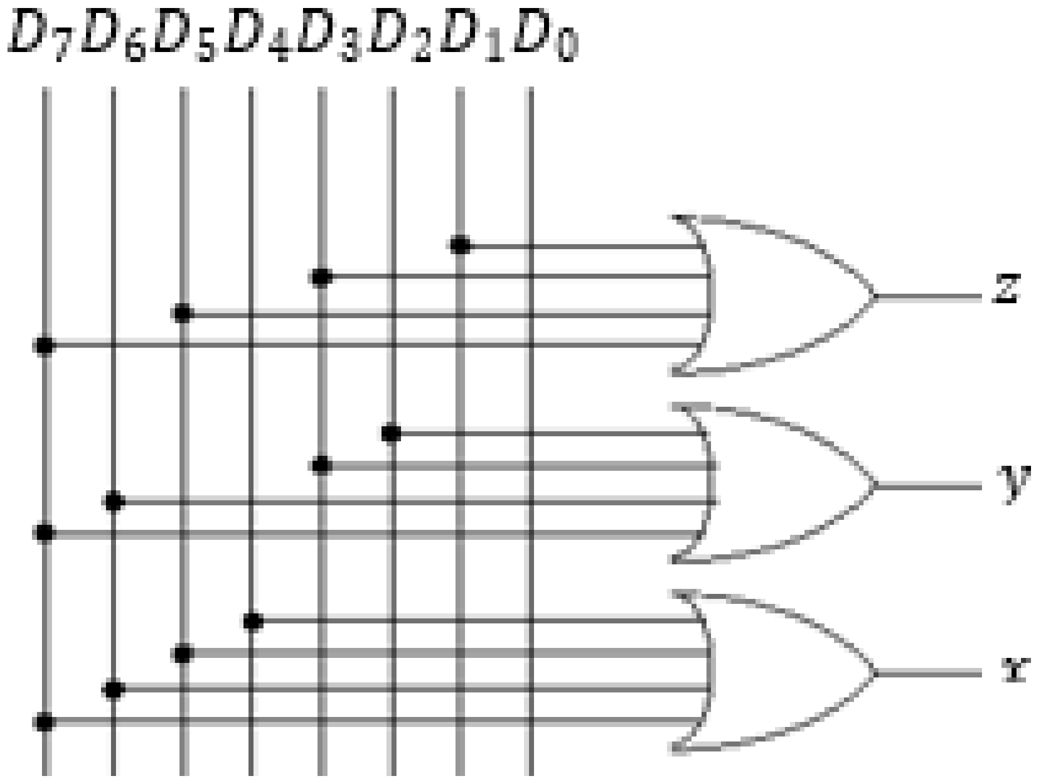


: Octal-to-binary encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Truth table of Octal-to-binary encoder

$$x = D_4 + D_5 + D_6 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$z = D_1 + D_3 + D_5 + D_7$$



Major Limitation of Encoders

- Exactly one input must be active at any given time.
- If the number of active inputs is less than one or more than one, the output will be incorrect.

For example, if $D_1 = D_2 = 1$, the output of the encoder $A_1A_0 = 11$, which implies incorrect output.

In the previous example,

if $D_1 = D_2 = 1$, the output corresponding to D_2 will be produced ($A_1A_0 = 10$) since D_2 has higher priority than D_1.

Major Limitation of Encoders

Two Problems to Resolve.

1. If two or more inputs are active at the same time, *what should the output be?*
2. An output of all 0's is generated in 2 cases:
   - when all inputs are 0
   - when E_0 is equal to 1.

How can this ambiguity be resolved?

Solution To Problem 1:

- Use a *Priority Encoder* which produces the output corresponding to the input with higher priority.

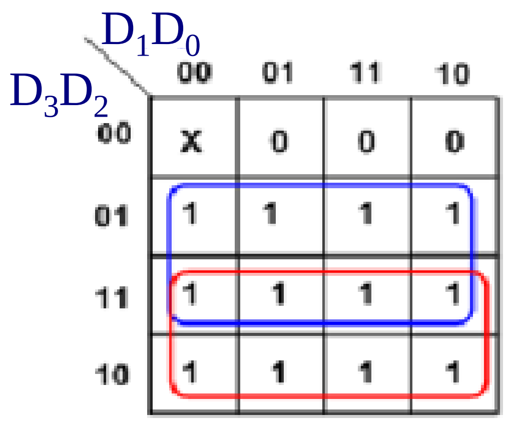
Solution To Problem 2:

- Provide one more output signal *V* to indicate *validity* of input data.
- *V* = 0 if none of the inputs equals 1, otherwise it is 1

4-to-2 Priority Encoder

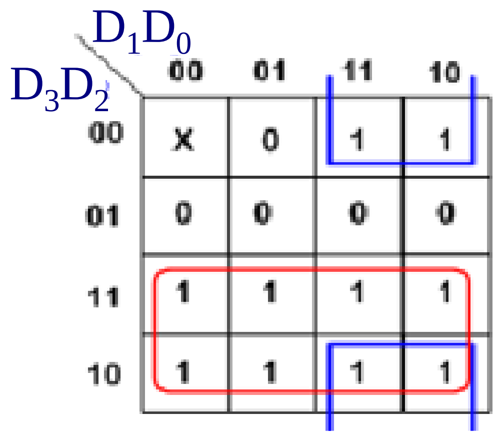
	Inputs				Outputs		
	D3	D2	D1	D0	A1	A0	V
1	0	0	0	0	X	X	0
2	0	0	0	1	0	0	1
3	0	0	1	X	0	1	1
4	0	1	X	X	1	0	1
5	1	X	X	X	1	1	1

Truth table of 4-to-2 priority encoder



$D_3D_2 \backslash D_1D_0$	00	01	11	10
00	x	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$A_1 = D_3+D_2$



$D_3D_2 \backslash D_1D_0$	00	01	11	10
00	x	0	1	1
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

$A_2 = D_3+D_1D'_2$

$V = D_3+D_2+D_1+D_0$

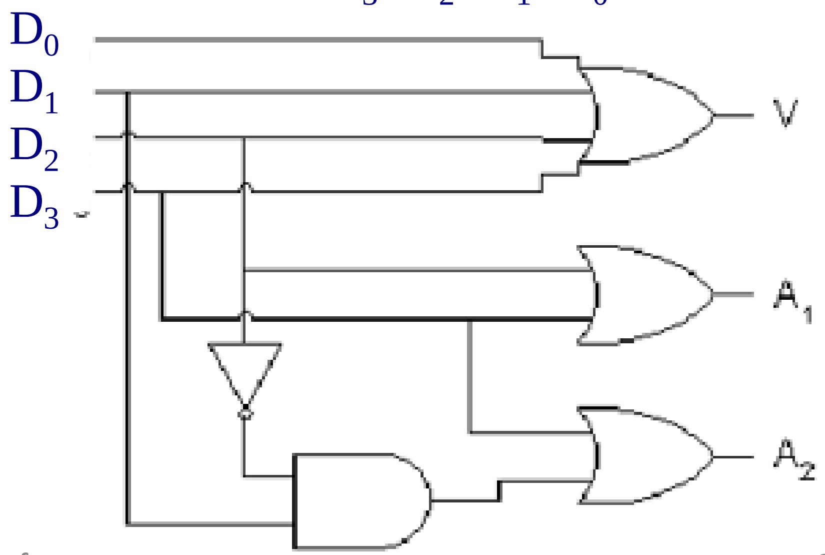


Figure : Equations and circuit for 4-to-2 priority encoder