Which PI's are essential to cover a term?

Essential Prime Implicants: the only PI that covers some min. term: in our example (see table above) each min term has at least 2 PI's → none. Then identify combination(s) with lowest cost that cover all min terms (9)

Lowest cost: use PI's that cover the most terms:

$$\underbrace{\underbrace{\overset{\checkmark}{B}}_{\text{3 terms.}} \quad \underbrace{C,\ \overset{\checkmark}{D}}_{\text{3 terms}}}_{\text{6 terms covered: } m_9, m_{11}, m_{27}\ (B);\ m_{14}, m_{15}, m_{30}\ (D)} \qquad \underbrace{G_{\checkmark}}_{\text{2 terms}} \quad , \quad \underbrace{H,\ I}_{\text{2 terms}}$$

New 3 terms once B is picked.

m_4, m_5

2 terms both are equal. m_{12}

$$\left.\begin{aligned} f_I &= B + D + G + H \\ f_{II} &= B + D + G + I \\ f_{III} &= B + D + F + H \end{aligned}\right\} \text{ same costs.} \leftrightarrow \boxed{\text{Lowest costs}}!$$

$$f_{IV} = B + C + E + G + I \quad \text{higher cost.}$$

...

$$f_I(v, w, x, y, z) = w\bar{x}z + wxy + \bar{v}\bar{w}x\bar{y} + \bar{v}x\bar{y}\bar{z}$$

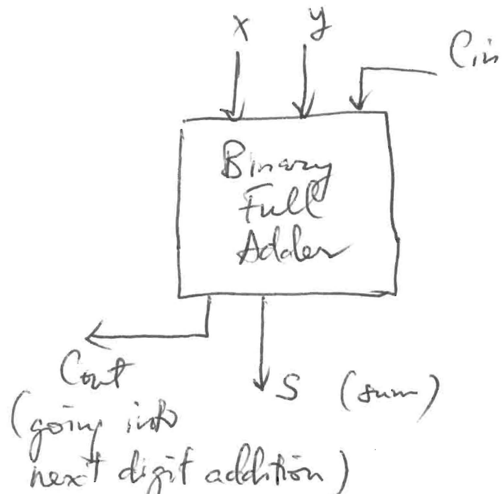
$$f_{II}(v, w, x, y, z) = w\bar{x}z + wxy + \bar{v}\bar{w}x\bar{y} + \bar{v}wx\bar{z}$$

$$f_{III}(v, w, x, y, z) = w\bar{x}z + wxy + \bar{v}\bar{w}\bar{y}z + \bar{v}x\bar{y}\bar{z}$$

Ch5: Logic Circuit Design with MSI's & PLD's

MSI: Medium-scale integrated circuits (10-100 gates)
PLD: Programmable Logic Device

Binary Adders & Subtracters:



Truth Table

x	y	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Truth Table → K-map → Simplified function for logic Design.
(2 functions) (2 K-maps)

S (x \ y Cin)	00	01	11	10
0	0	$\boxed{1}$	0	$\boxed{1}$
1	$\boxed{1}$	0	$\boxed{1}$	0

Four groups of 1 one

$S = \bar{x}\bar{y}C_{in} + \bar{x}y\bar{C}_{in} + x\bar{y}\bar{C}_{in} + xyC_{in}$
(Standard min term canonical or SOP)

Cout (x \ y Cin)	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Three groups of 2 ones

$C_{out} = \underset{(A)}{yC_{in}} + \underset{(B)}{xC_{in}} + \underset{(C)}{xy}$

For s: can now simplify using XOR's

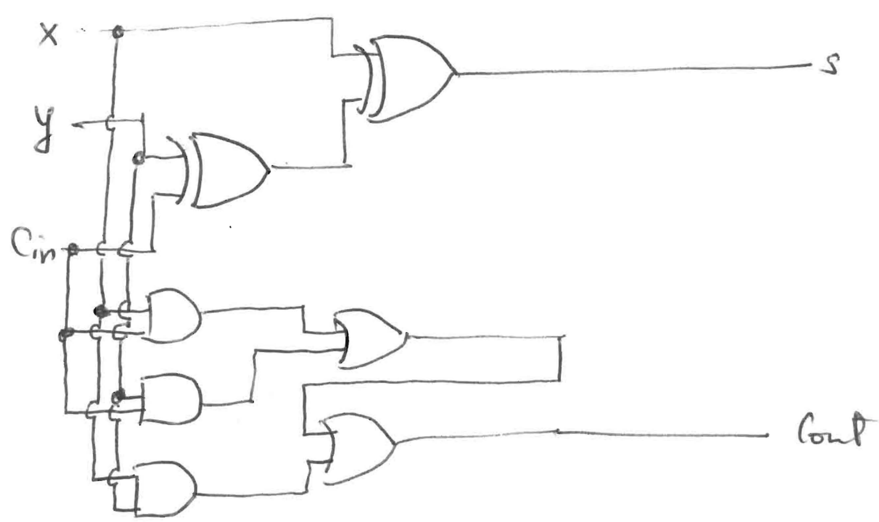
$$a \oplus b = a\bar{b} + \bar{a}b$$

$$\overline{a \oplus b} = \bar{a}\bar{b} + ab$$

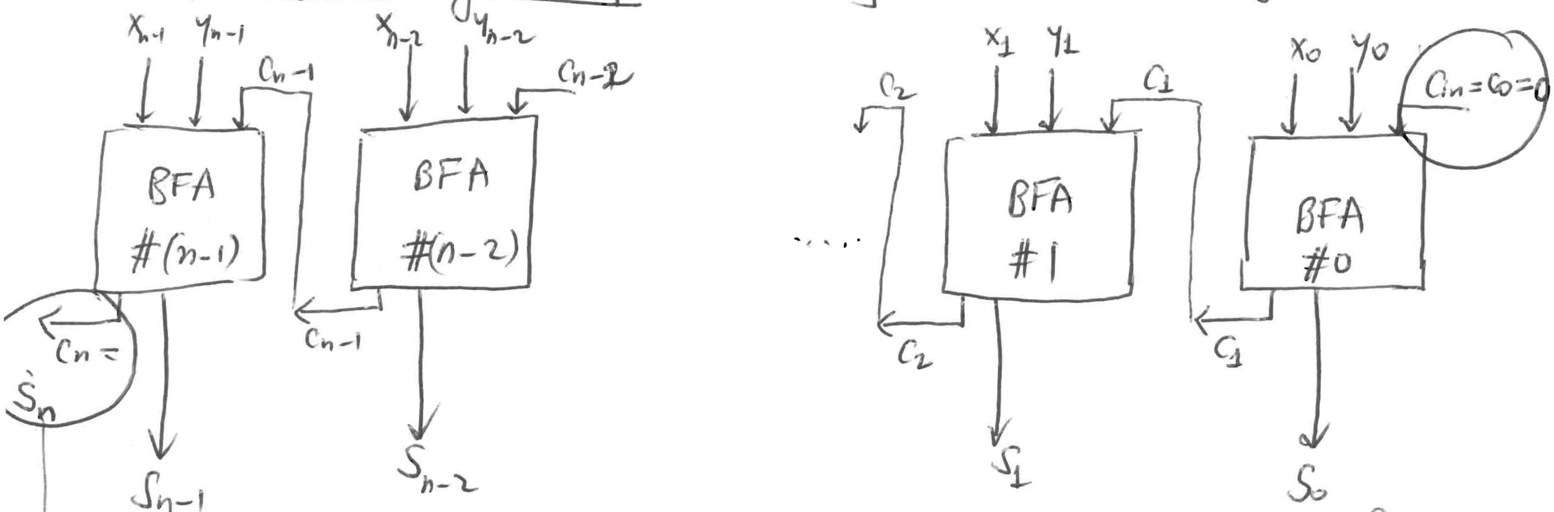
$$S = \bar{x}\,(\bar{y}\,c_{in} + y\,\bar{c}_{in}) + x\,(\bar{y}\,\bar{c}_{in} + y\,c_{in}) = \bar{x}\,(y \oplus c_{in}) + x\,\overline{(y \oplus c_{in})}$$

$$= x \oplus (y \oplus c_{in})$$

<u>Logic design for Full Binary Adder:</u>



$\boxed{\text{n-digit Binary Adder}}$ = combining n 1-digit Binary Adders in series



Note: when we add two n-digit binary numbers, in general we get an n+1 digit binary number: the last Cout is the highest digit of the sum

$$\begin{array}{r} 1\ 0 \\ +\ \ 1\ 1 \\ \hline 1\ 0\ 1 \end{array}$$

The third & highest digit is the last carry-out!

$\boxed{S_n = C_n}$!

$\boxed{\text{n-digit Binary Subtracter:}}$

1) Combining n 1-digit BFS's (Binary Full Subtracters)

2) Add 2's complement of the second number to the first number $N_1 - N_2 = N_1 + \overline{\overline{N_2}}$

$\overline{\overline{N_2}}$: 2's complement of N_2

$\overline{N_2}$: 1's complement of N_2 . Note : $\boxed{\overline{\overline{N_2}} = \overline{N_2} + 1}$

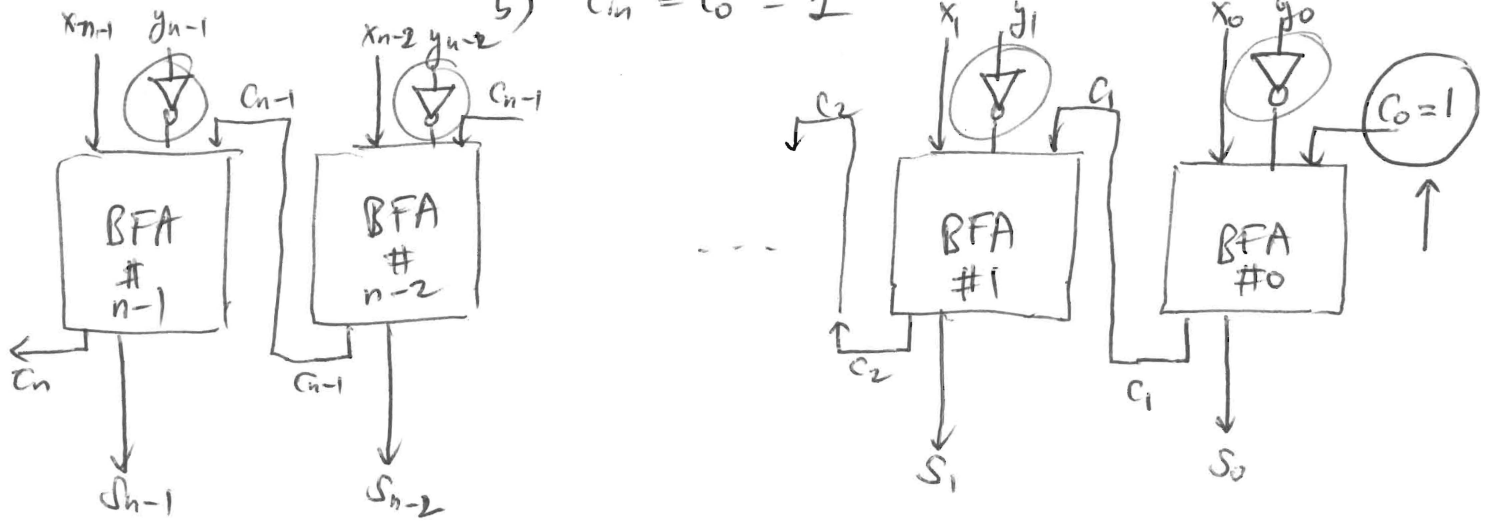
N_2	$\overline{N_2}$	$\overline{\overline{N_2}}$
0	1	10
1	0	01

a) Complement the input

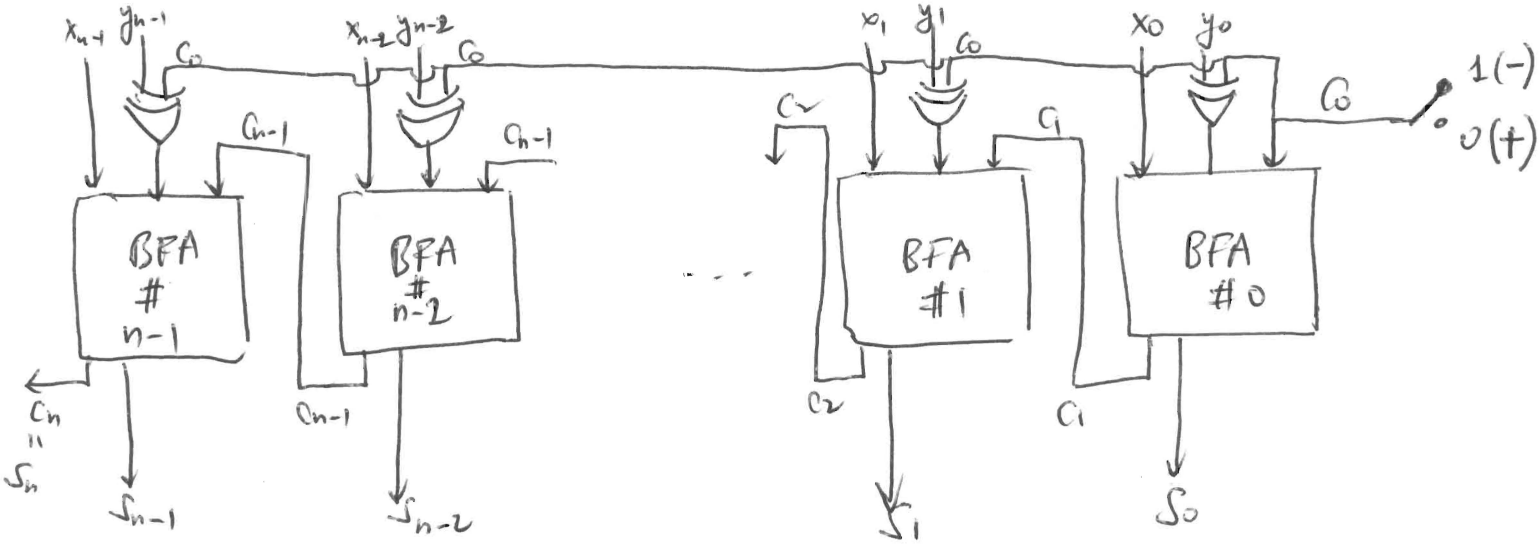
b) use $C_{in} = C_0 = 1$

n-digit binary subtracter: combining n 1-digit BFA's with a) 1-complement the y's (second number)

b) $C_{in} = C_0 = 1$



Can we control the adder or subtracter with a switch? Yes!



Notes: 1)

y_0	C_0	$y_0 \oplus C_0$
0	0	0
0	1	1
1	0	1
1	1	0

it only inverts y_0 if $C_0 = 1$

2) In-series combination disadvantage: the next BFA needs to wait until the previous BFA calculated its carry-out to start its calculation: "ripple effect" ↳ wait for ripple! → slow calculation speed.
→ solution: "Carry lookahead adder"

Carry look-ahead Adder:

$$C_{out} = C_{i+1} = x_i y_i + x_i C_i + y_i C_i$$ (From K-map 3 groups of two consecutive ones)

$$= \underbrace{x_i y_i}_{\equiv g_i} + \underbrace{(x_i + y_i) C_i}_{\equiv p_i \cdot C_i}$$

"carry-generate function" "carry-propagate function"

With these two definitions:

$$C_1 = g_0 + p_0 \cdot C_0$$

$$C_2 = g_1 + p_1 \cdot C_1 = g_1 + p_1 \cdot (g_0 + p_0 \cdot C_0) = g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot C_0$$

(C_2 can be calculated without C_1)

$$C_3 = g_2 + p_2 \cdot C_2 = g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0 + p_2 \cdot p_1 \cdot p_0 \cdot C_0$$

(C_3 can be calculated without C_2 & C_1)

$$C_{i+1} = g_i + p_i \cdot C_i = g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \cdots p_i p_{i-1} \cdots p_2 \cdot p_1 g_0$$

$$+ p_i p_{i-1} \cdots p_2 \cdot p_1 \cdot p_0 C_0$$

We are able to compute C_{i+1} using g_i, p_i & the initial C_0 (no need to wait for $C_1, C_2, \ldots, C_i$ to be computed).

→ "Carry-lookahead Adder" or "high-speed adder"

- Adders ✓
- Comparators ✓
- Decoders ✓
- Multiplexers.

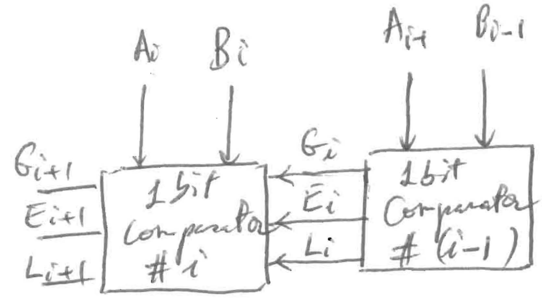
Comparators :

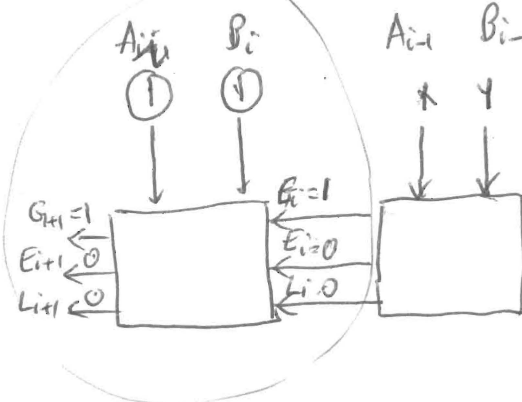
1 bit : >, =, <

G E L

* Compare two n-bit binary numbers using 1-bit comparators

A_i	B_i	G_i	E_i	L_i	G_{i+1}	E_{i+1}	L_{i+1}
0	0	0	0	0	–	–	–
		0	0	1	0	0	1
		0	1	0	0	1	0
		0	1	1	–	–	–
		1	0	0	1	0	0
		1	0	1	–	–	–
		1	1	0	–	–	–
		1	1	1	–	–	–
0	1				–	–	–
					0	0	1
					0	0	1
					–	–	–
					0	0	1
					–	–	–
					–	–	–
					–	–	–
1	0				–	–	–
					1	0	0
					1	0	0
					–	–	–
					1	0	0
					–	–	–
					–	–	–
					–	–	–
1	1				–	–	–
					0	0	1
					0	1	0
					–	–	–
		1	0	0	1	0	0
					–	–	–
					–	–	–
					–	–	–





K-Maps:

$\boxed{G_{i+1}}$

A_iB_i \ $G_iE_iL_i$	000	001	011	010
00				
01				
11				
10				

A_iB_i \ $G_iE_iL_i$	110	111	101	100
00				
01				
11				
10				

$G_{i+1} = A_i\bar{B}_i + A_iG_i + \bar{B}_iG_i$ Ones: 2, 4, 8 ↑

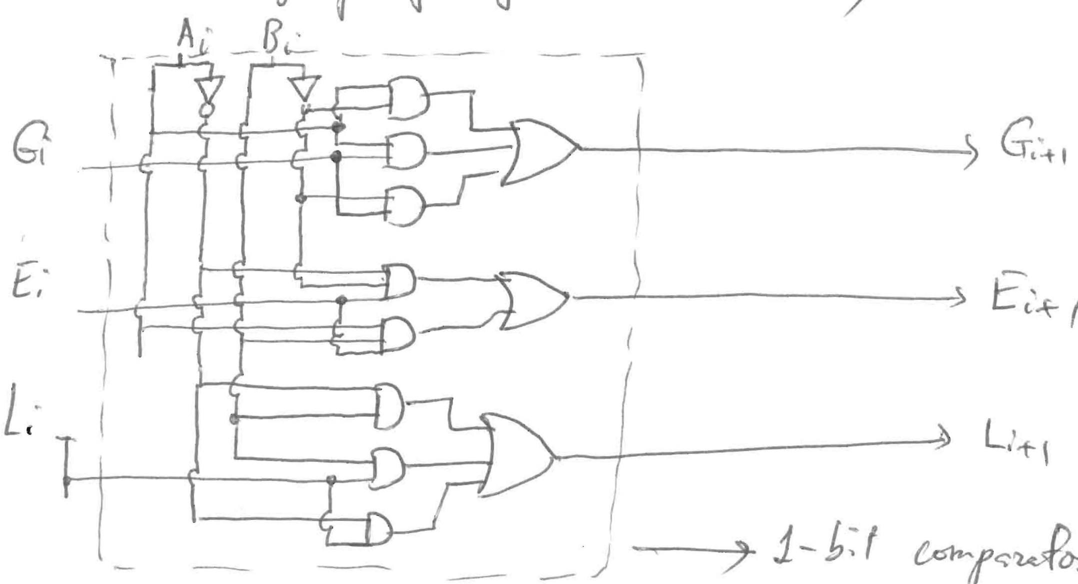
(3 groups of eight ones & bars)

$E_{i+1} = \bar{A}_i\bar{B}_iE_i + A_iB_iE_i$

(2 groups of four ones & bars → dc's)

$L_{i+1} = \bar{A}_iB_i + B_iL_i + \bar{A}_iL_i$

(3 groups of eight ones & bars)



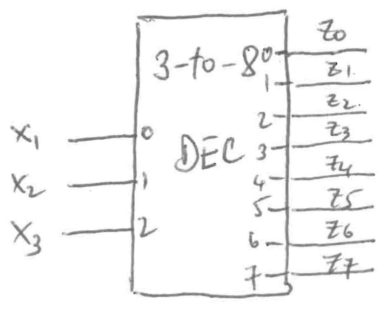
→ 1-bit comparator (5 inputs & 3 outputs)

Decoders: 3-to-8 line Decoder

(3 bit to 8 bit)

Decimal Equiv	x_1	x_2	x_3	z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7
0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
2	0	1	0	0	0	1	0	0	0	0	0
3	0	1	1	0	0	0	1	0	0	0	0
4	1	0	0	0	0	0	0	1	0	0	0
5	1	0	1	0	0	0	0	0	1	0	0
6	1	1	0	0	0	0	0	0	0	1	0
7	1	1	1	0	0	0	0	0	0	0	1

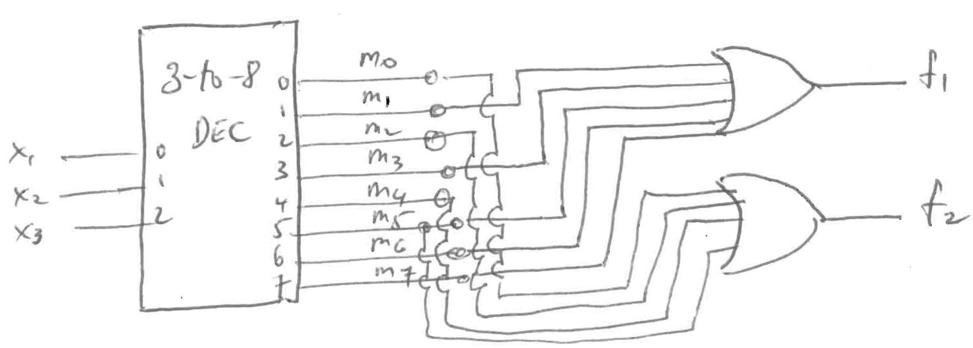
* Decimal equivalent of $x_1 x_2 x_3$ $\rightarrow$ which z_i $(i = 0, 1, 2, \ldots 7)$ will be one.
* This decoder is also a minterm generator



$$z_0 = \bar{x}_1 \bar{x}_2 \bar{x}_3 = m_0$$
$$z_1 = \bar{x}_1 \bar{x}_2 x_3 = m_1$$
$$z_2 = \bar{x}_1 x_2 \bar{x}_3 = m_2$$
$$z_3 = \bar{x}_1 x_2 x_3 = m_3$$
$$z_4 = x_1 \bar{x}_2 \bar{x}_3 = m_4$$
$$z_5 = x_1 \bar{x}_2 x_3 = m_5$$
$$z_6 = x_1 x_2 \bar{x}_3 = m_6$$
$$z_7 = x_1 x_2 x_3 = m_7$$

* As a minterm generator, the 3-to-8 line decoder can be used to implement 3-variable functions:

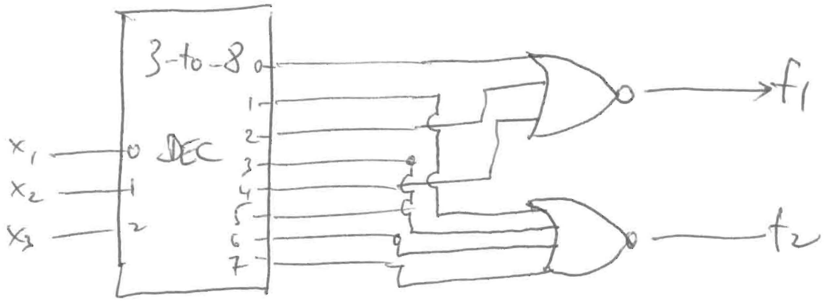
$$f_1(x_1, x_2, x_3) = \sum m(1, 3, 5, 6, 7) \; ; \; f_2(x_1, x_2, x_3) = \sum m(0, 2, 4, 5)$$



Cost-efficiency: less wiring for same functions:

$$f_1 = \overline{\overline{f_1}} = \overline{\underbrace{\sum m(0,2,4)}_{\overline{f_1}}}$$

$$f_2 = \overline{\overline{f_2}} = \overline{\underbrace{\sum m(1,3,6,7)}_{=\overline{f_2}}}$$



Max terms

$$f_3(x_1,x_2,x_3) = \prod M(0,3,5)$$
$$f_4(x_1,x_2,x_3) = \prod M(1,4,7)$$

Can use 3-to-8 line Decoders to implement these function as wells! What property do we need to use?

$$\boxed{\text{How to implement 3-variable functions written as POS (products of sums or max terms) using 3-to-8-Decoders which are min term generators.}}$$

Answer: use De Morgan's Law:
$$\begin{cases} \overline{y_1 y_2 y_3} = \bar{y}_1 + \bar{y}_2 + \bar{y}_3 \\ \overline{y_1 + y_2 + y_3} = \bar{y}_1 \bar{y}_2 \bar{y}_3 \end{cases}$$

$$\overline{M} = m \quad \text{or} \quad M = \overline{m}$$

$$f_3 = \prod M(0,3,5) \; ; \quad f_4 = \prod M(1,4,7)$$

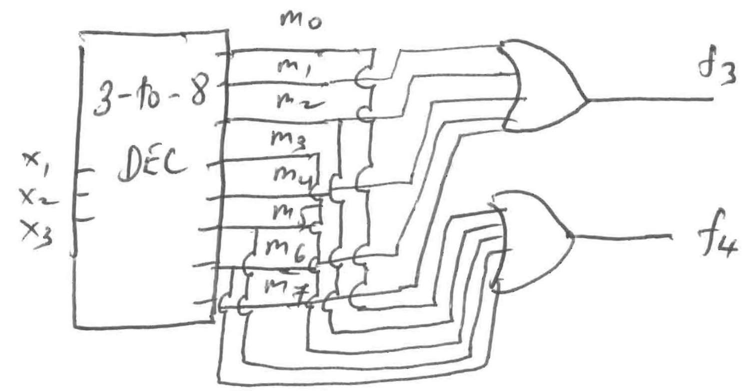
$$\rightarrow \boxed{f_3 = \bar{\bar{f_3}} = \overline{\sum m(1,2,4,6,7)}}$$

$$\bar{f_3} = \prod M(1,2,4,6,7) = \sum \bar{m}(1,2,4,6,7)$$

Similarly,

$$\boxed{f_4 = \bar{\bar{f_4}} = \overline{\sum m(0,2,3,5,6)}}$$

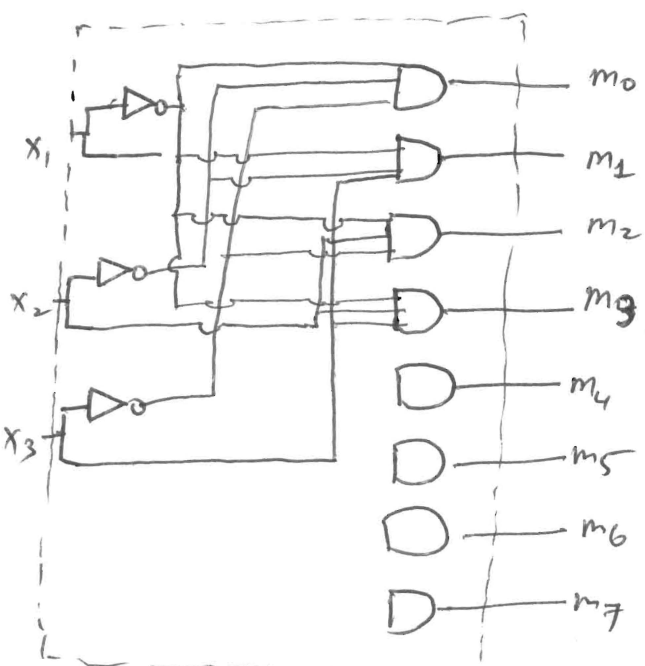
Logic diagram:



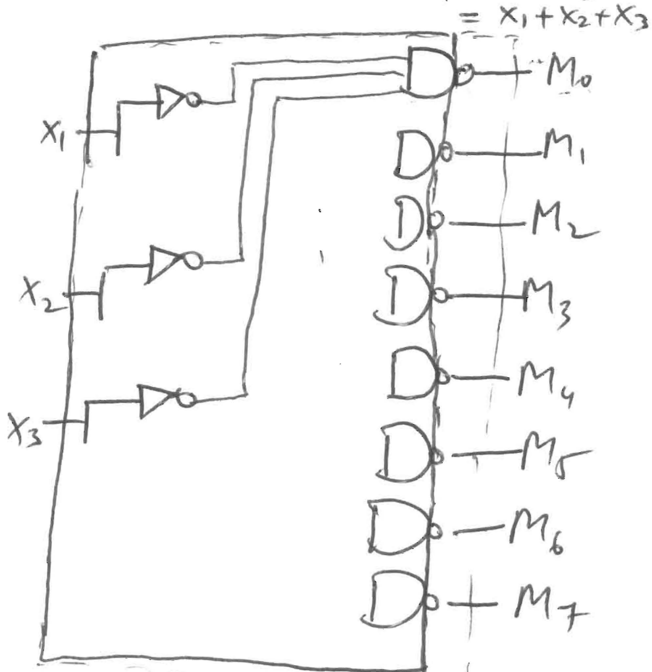
Logic diagrams for the decoders themselves: 3-to-8 Dec is Min term Generator

AND

NAND $\quad \overline{\bar{x}_1 \bar{x}_2 \bar{x}_3} = \bar{\bar{x}}_1 + \bar{\bar{x}}_2 + \bar{\bar{x}}_3 = x_1 + x_2 + x_3$

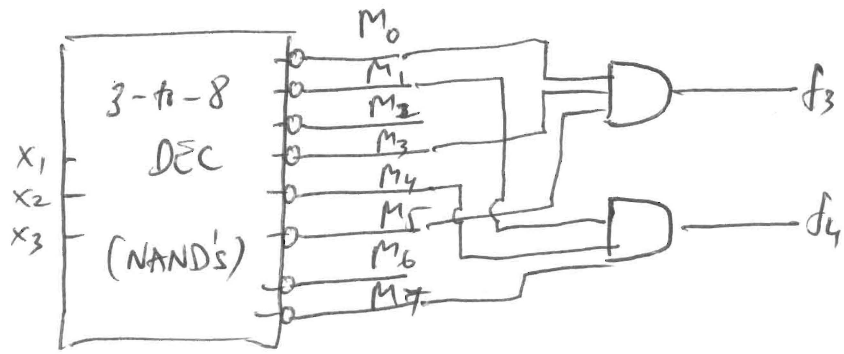


3-to-8 DEC - Min Term Gen.

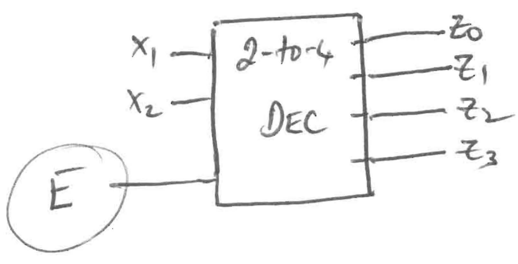


3-to-8 DEC → Max Term Gen.

Using NAND's version of 3-to-8 DEC:

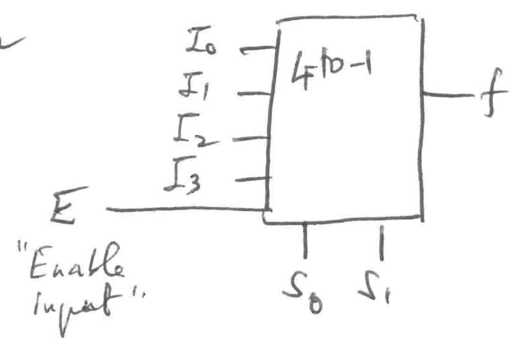


2-to-4 Decoder with an <u>Enable Input</u>



E	x_1	x_2	z_0	z_1	z_2	z_3
0	–	–	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

<u>Multiplexers</u> = input selector

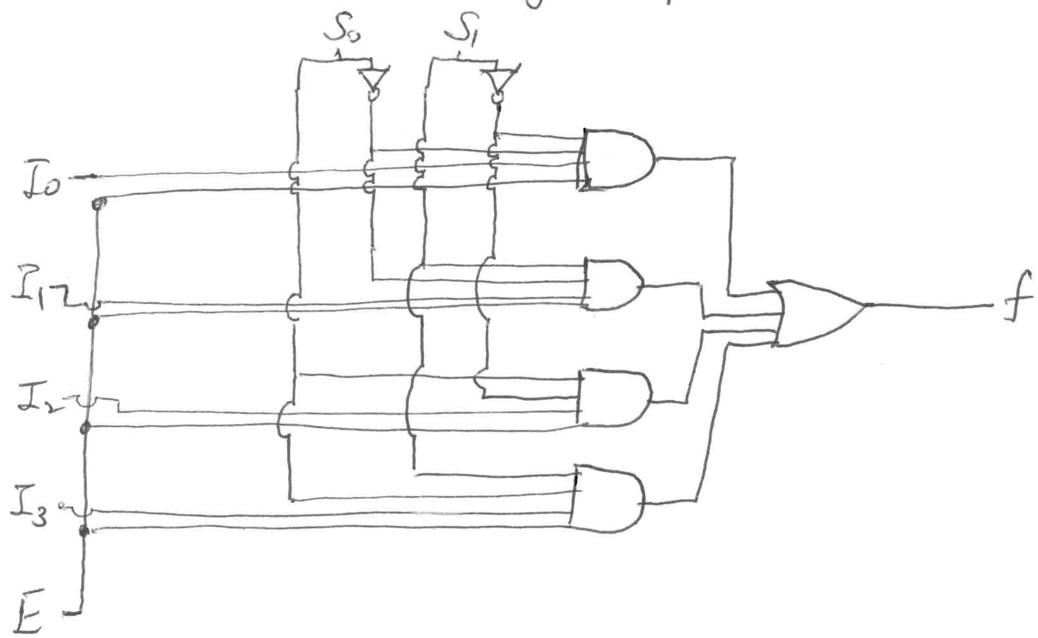


- 4 inputs $\xrightarrow{\text{need}}$ 2 switches
- 8 inputs $\rightarrow$ 3
- ⋮
- 2^n inputs $\rightarrow$ n switches or n bits

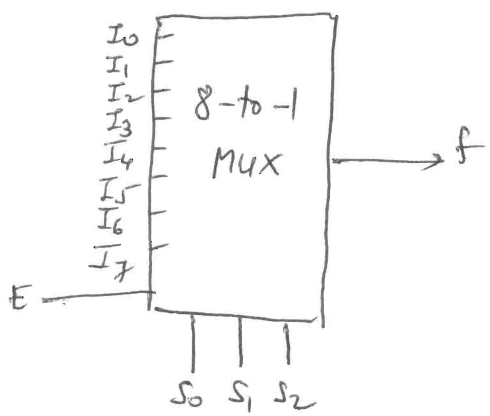
E	S_0	S_1	f
1	0	0	I_0
1	0	1	I_1
1	1	0	I_2
1	1	1	I_3
0	–	–	0

Using these logic switches S_0 & S_1 we can select which input line will show at output.

What is the logic diagram for a 4-to-1 MUX?



→ Similarly the <u>8-to-1 MUX</u> is built:
- 3 switches.
- 8 inputs + E
- 1 output.
- 8 AND's + 1 OR



E	S_0	S_1	S_2	f
0	–	–	–	0
1	0	0	0	I_0
	0	0	1	I_1
	0	1	0	I_2
	0	1	1	I_3.
	1	0	0	I_4
	1	0	1	I_5
	1	1	0	I_6
	1	1	1	I_7

$$f(S_0, S_1, S_2) = E\Big(I_0 \bar{S}_0 \bar{S}_1 \bar{S}_2 + I_1 \bar{S}_0 \bar{S}_1 S_2 + I_2 \bar{S}_0 S_1 \bar{S}_2 + I_3 \bar{S}_0 S_1 S_2$$
$$I_4 S_0 \bar{S}_1 \bar{S}_2 + I_5 S_0 \bar{S}_1 S_2 + I_6 S_0 S_1 \bar{S}_2 + I_7 S_0 S_1 S_2 \Big)$$

→ <u>16-to-1 MUX</u>:
- 4 switches
- 16 inputs + E
- 1 output

can be built with four 4-to-1 MUX's plus another to produce the outputs.

MUX's can be used to implement logic functions.

$f = f(x, y, z) = \Sigma m(0, 2, 3, 5) \rightarrow$ 3 variable $\rightarrow 2^3 = 8$ inputs.

$\rightarrow$ use 8-to-1 MUX :

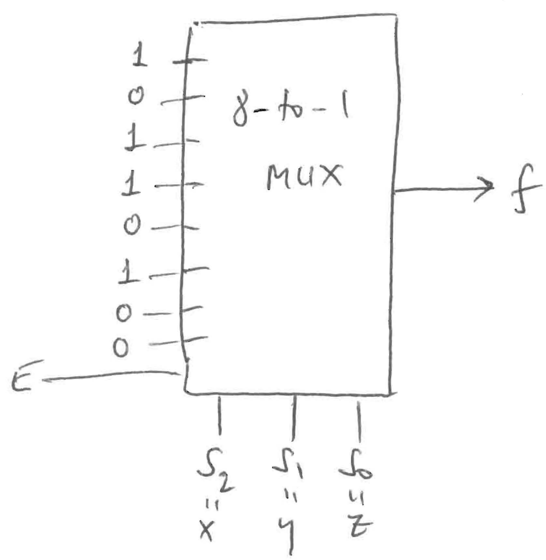
$S_2 \rightarrow x$
$S_1 \rightarrow y$
$S_0 \rightarrow z$

$m_0 \rightarrow I_0$
$m_1 \rightarrow I_1$
$\vdots$
$m_7 \rightarrow I_7$

$\underline{f}$
$m_0 = 1$
$m_2 = 1$
$m_3 = 1$
$m_5 = 1$

E		$x = S_2$	$y = S_1$	$z = S_0$	f
0		–	–	–	0
1	m_0	0	0	0	1
	m_1	0	0	1	0
	m_2	0	1	0	1
	m_3	0	1	1	1
	m_4	1	0	0	0
	m_5	1	0	1	1
	m_6	1	1	0	0
	m_7	1	1	1	0

To implement using 8-to-1 MUX $\rightarrow$ preset its inputs accordingly.



Should check if same function f can be implemented using a smaller MUX: 4-to-1 MUX?

What is simplest form for f?

$$f(x,y,z) = \underbrace{\bar{x}\bar{y}\bar{z}}_{m_0} + \underbrace{\bar{x}y\bar{z}}_{m_2} + \underbrace{\bar{x}yz}_{m_3} + \underbrace{x\bar{y}z}_{m_5}$$

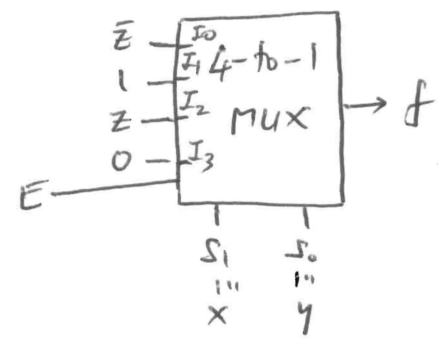
$$= \bar{x}\bar{z} + (\bar{x}y + x\bar{y})z$$

$$= \bar{x}\bar{y}\bar{z} + \bar{x}y + x\bar{y}z$$

<u>4-to-1 MUX</u>

E=1

$x=S_1$	$y=S_0$	f
0	0	$I_0 \equiv \bar{z}$
0	1	$I_1 \equiv \bar{x}y = 1$
1	0	$I_2 \equiv x\bar{y}z = z$
1	1	$I_3 = 0$



<u>Programmable Logic Devices (PLD's):</u>

PLD's = combinations of AND's & OR's arrays

- PROM (Programmable Read only Memory): AND's fixed, OR's programmable
- PLA (Prog. Logic Array): AND's & OR's programmable
- PAL (Prog. Array Logic): AND's programmable, OR's fixed

Programmable
- field programmable (done by customer)
- mask programmable (done by manufacturer)

One way to program connections is to use fuses { blow fuses of unwanted connections.

Open connections {
- AND gate → open is 1
- OR gate → open is 0

Simplified Notations for PLD's: