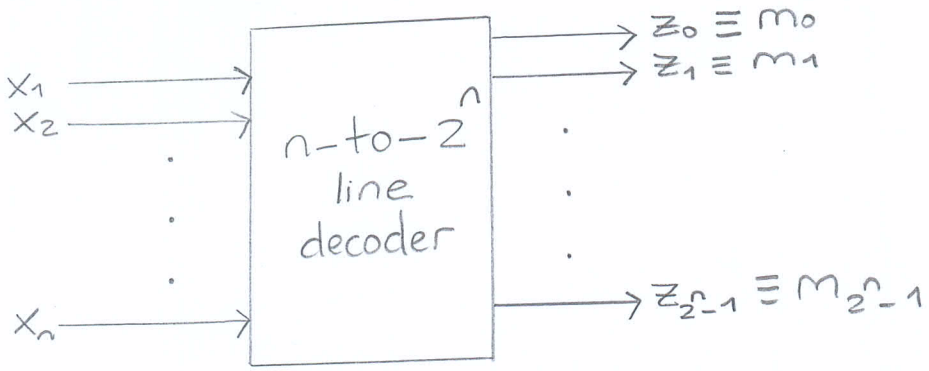
Combinational circuit elements

- We consider some of the important combinational circuit elements along with their descriptive functions
- We then investigate circuits that involve these elements

Decoder

- an n-input and 2^n-output combinational circuit element
- symbolically represented as follows



example. Let $n = 4$, if $(x_1x_2x_3x_4) = (0110)_2 = (6)_{10}$, then $z_6 = 1$ and $z_i = 0$ for $i \neq 6$

Because;

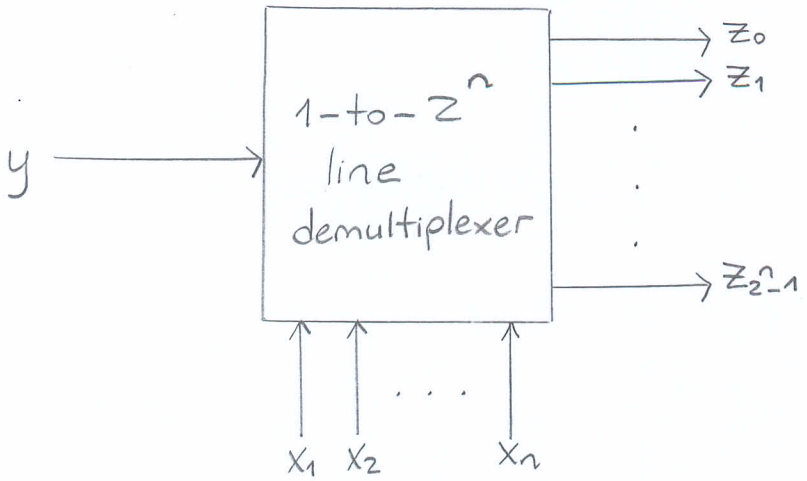
$$z_6 = m_6 = x_1' x_2 x_3 x_4' = 1$$

$$z_i = 0 \quad \text{for} \quad i = 0\text{-}5, 7\text{-}15$$

Demultiplexer (Data distributor)

- an $n+1$-input and 2^n-output combinational circuit element

- represented symbolically as follows



where

y : data input

x_i, $i=1,\dots,n$: control (select) inputs

- we have the following relationship

$$m_i = x_1^* x_2^* \cdots x_n^* \quad , \quad z_i = m_i y$$

example. If $(x_1 x_2 \dots x_n) = (0\,0\dots0\,1\,0\,1)_2$ then $m_5 = 1$ and thus $z_5 = m_5 y = y$ and $z_i = 0$, $i \neq 5$

Hence;

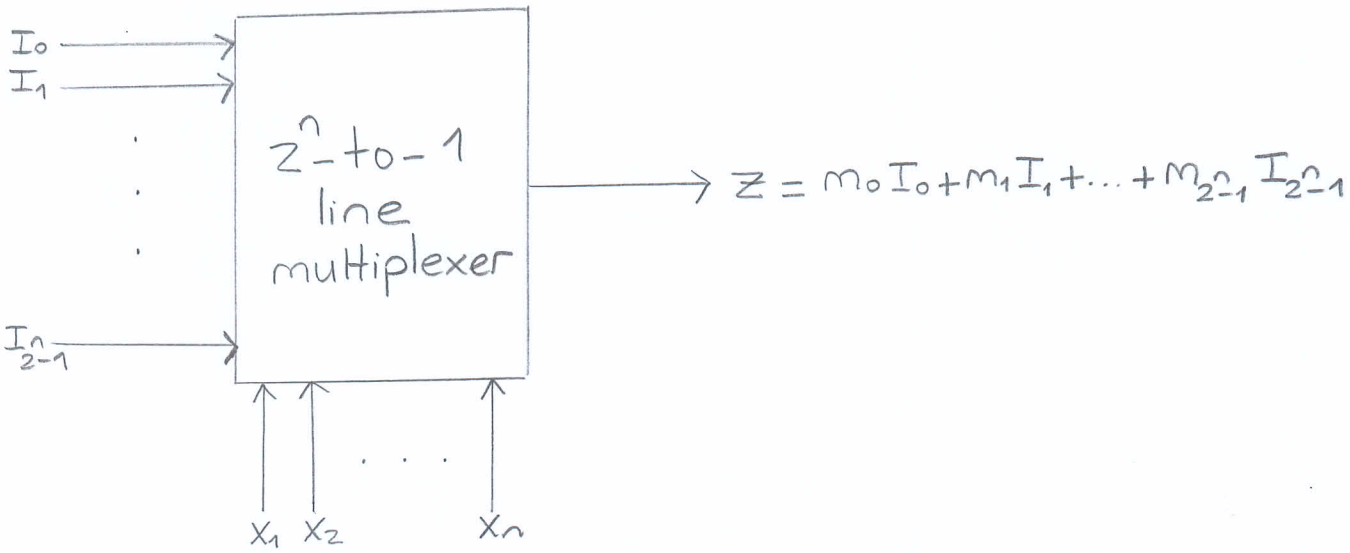
- we can consider a demultiplexer circuit element as controlling from which output line the data input is received

↳ through the control (select) inputs

Remark. Note that if we always keep $y=1$, then the demultiplexer circuit reduces to an n-to-2^n line decoder.

Multiplexer (Data selector)

- a circuit element with $n+2^n$ inputs and one output
- the block diagram representation is given by



where

$I_0, I_1, \dots, I_{2^n-1}$: data inputs

$x_1, x_2, \dots, x_n$: control (select) inputs

z : output

Note that;

- if $(x_1 x_2 \dots x_n) = j$ then the data input I_j is transferred to the output

Implementation of Boolean functions with multiplexers

- It is usually desired that a Boolean function could be implemented by using only a single type of circuit element such as NAND or NOR
- The multiplexer circuit elements can be employed for this purpose
- If not possible, then one can employ some additional elements

(n+k) variable Boolean function

- we investigate the implementation of an (n+k) variable Boolean function

↳ via a 2^n-to-1 line multiplexer plus an AND-OR circuit

- let us consider

$$f(x_1, \dots, x_n, x_{n+1}, \dots, x_{n+k}) = \sum \hat{m}_i$$

$$= m_0 f_0(x_{n+1}, \dots, x_{n+k}) + m_1 f_1(x_{n+1}, \dots, x_{n+k}) + \vdots + m_{2^n-1} f_{2^n-1}(x_{n+1}, \dots, x_{n+k})$$

where $m_i = x_1^* x_2^* \dots x_n^*$

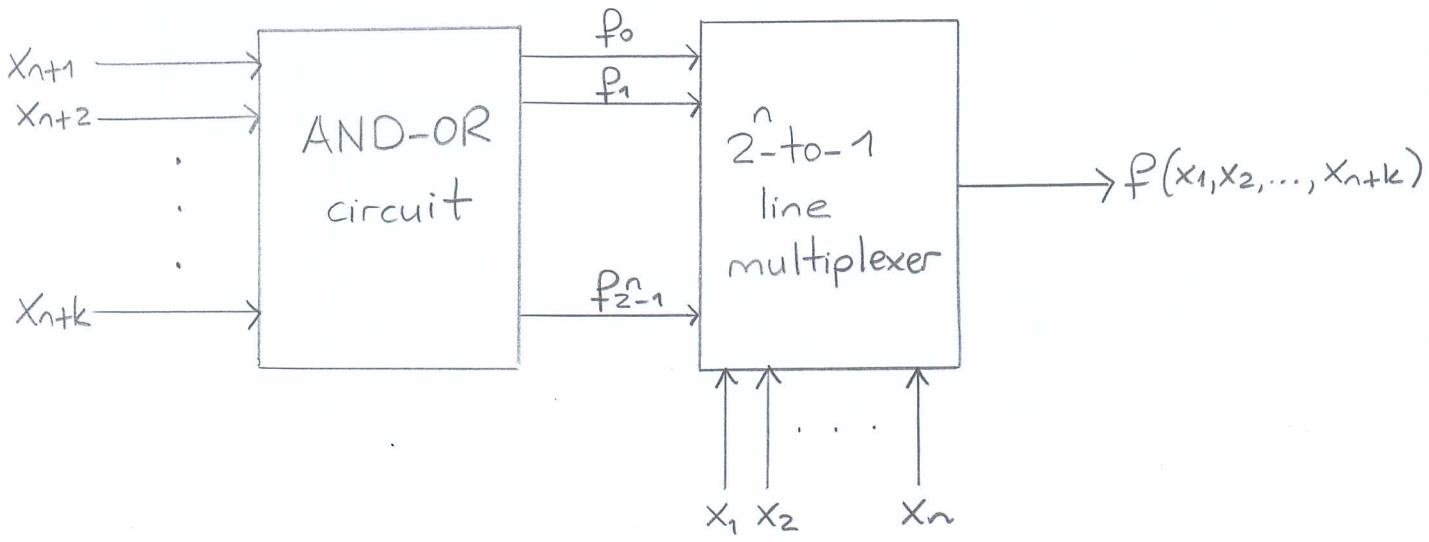
$\hat{m}_i = x_1^* x_2^* \dots x_n^* x_{n+1}^* \dots x_{n+k}^*$

- if we factor out the expression into a paranthesis of m_i, then the k-variable Boolean function

↳ $f_i(x_{n+1}, \ldots, x_{n+k})$ can be implemented through Karnaugh method

Hence;

- we can implement the (n+k) variable Boolean function using a 2^n-to-1 line multiplexer and an AND-OR circuit



Special conditions:

1. If $k=0$ then $f_i = 0$ or $f_i = 1$. In other words, the AND-OR combinational logic circuit is NO LONGER needed.

- Therefore, any Boolean function with n or less than n variable can be implemented with a 2^n-to-1 line multiplexer

2. If $k=1$ then $f_i \in \{0, 1, x_{n+1}, x'_{n+1}\}$

- Thus, using a multiplexer with n control (select) inputs and complement logic

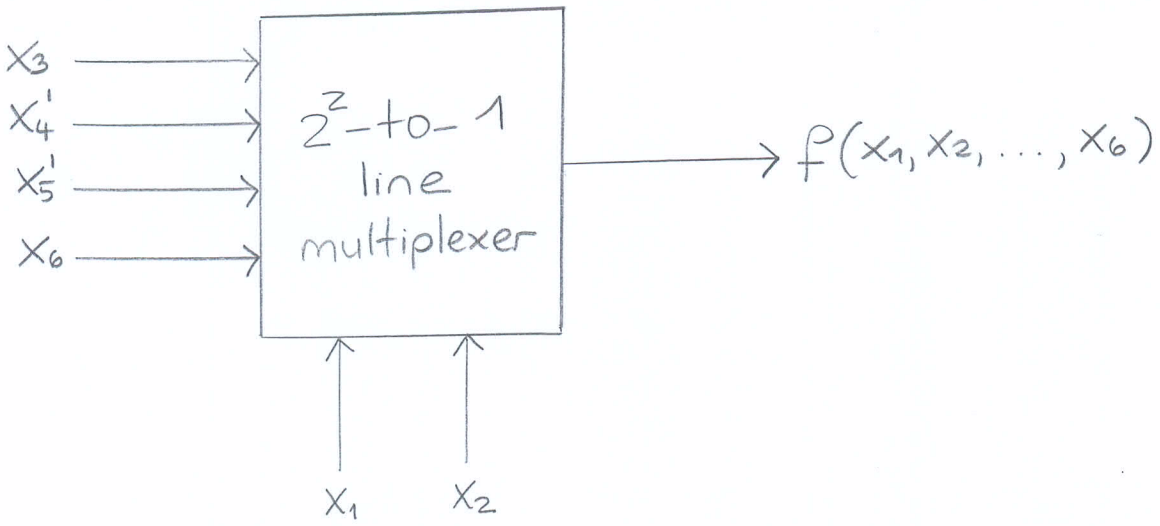
  (1) enables to implement an $(n+1)$ variable Boolean function

example. Consider the following Boolean fn.

$$f(x_1, x_2, \ldots, x_6) = x'_1 x'_2 x_3 + x'_1 x_2 x'_4 + x_1 x'_2 x'_5 + x_1 x_2 x_6$$

- we wish to implement $f(x_1, x_2, \ldots, x_6)$ by employing a multiplexer with 2 control inputs

$$f(x_1, x_2, \ldots, x_6) = m_0 x_3 + m_1 x'_4 + m_2 x'_5 + m_3 x_6$$

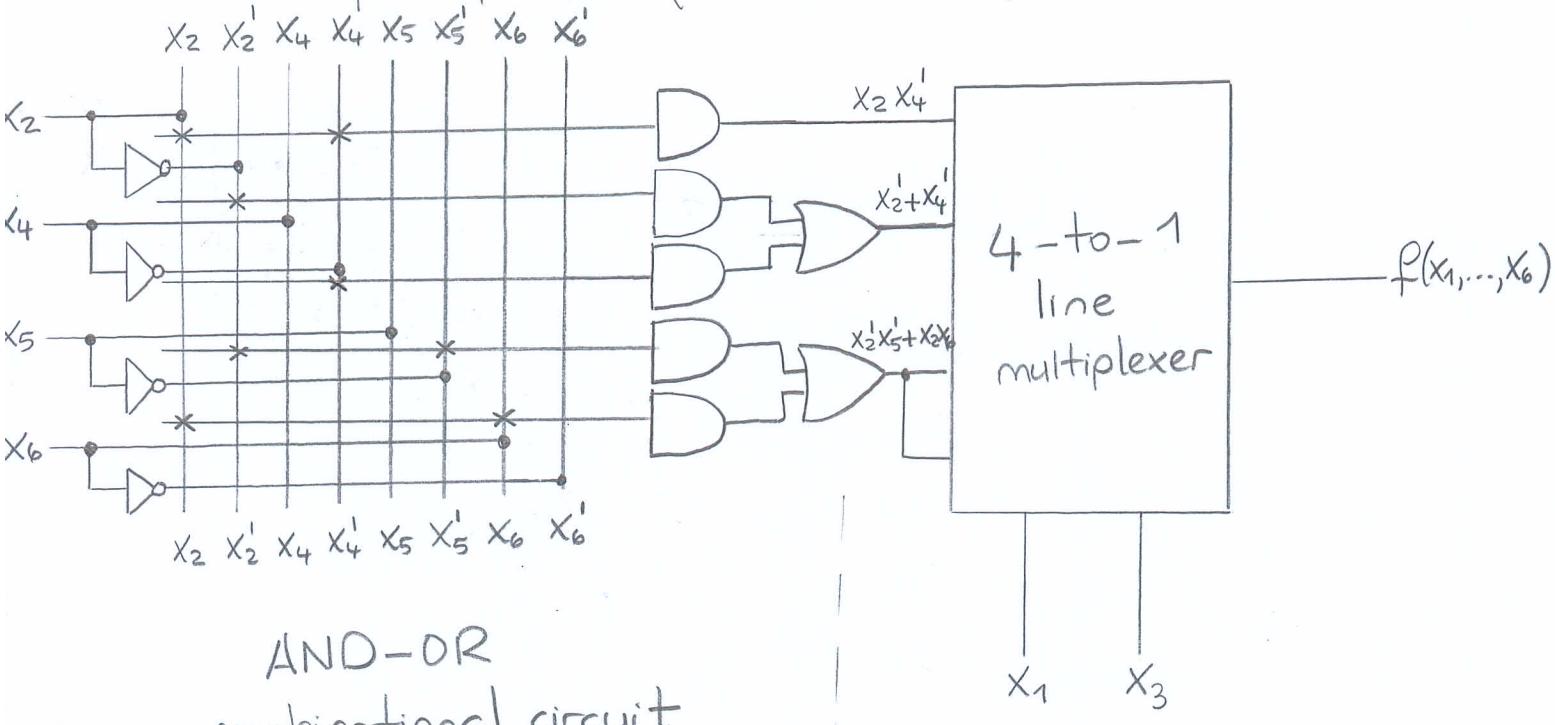


Now;

- what if we choose x_1, x_3 as the control (select) inputs

- then we can rewrite $f(x_1, x_2, \ldots, x_6)$ as follows

$$f(x_1, \ldots, x_6) = x_1' x_2' x_3 + x_1' x_2 (x_3 + x_3') x_4'$$
$$+ x_1 x_2' (x_3 + x_3') x_5' + x_1 x_2 (x_3 + x_3') x_6$$
$$= x_1' x_3' x_2 x_4' + x_1' x_3 (x_2' + x_2 x_4')$$
$$+ x_1 x_3' (x_2' x_5' + x_2 x_6)$$
$$+ x_1 x_3 (x_2' x_5' + x_2 x_6)$$
$$= x_1' x_3' x_2 x_4' + x_1' x_3 (x_2' + x_4')$$
$$+ x_1 x_3' (x_2' x_5' + x_2 x_6)$$
$$+ x_1 x_3 (x_2' x_5' + x_2 x_6)$$

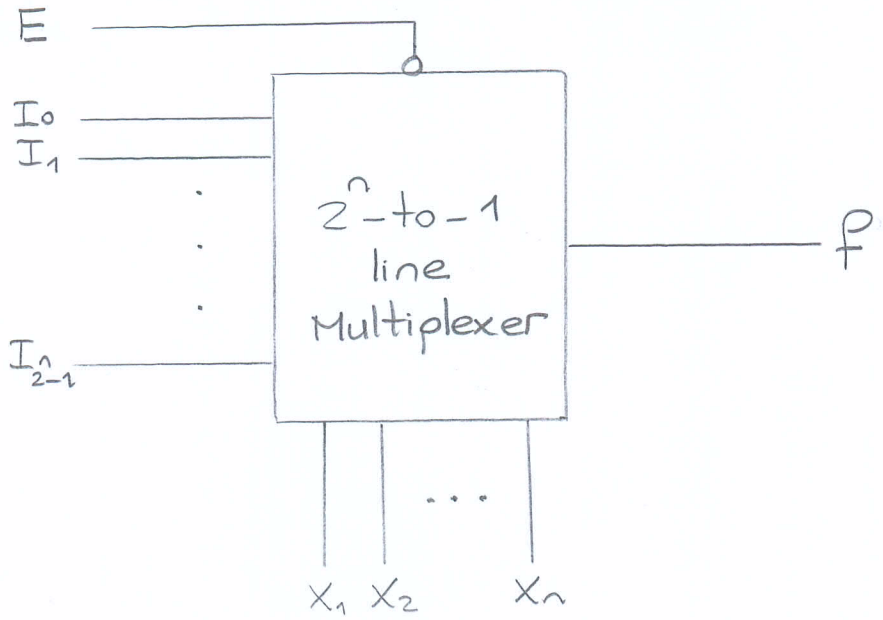


AND-OR combinational circuit

Multiplexer

Multiplexer with enable input

- we consider a multiplexer with enable input



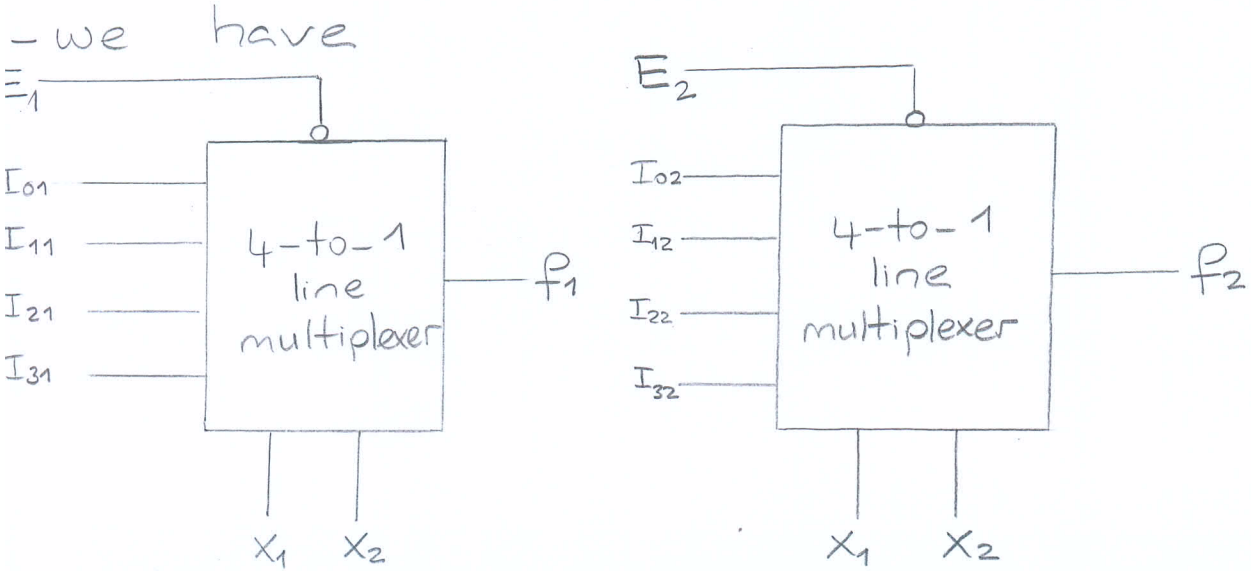
$$f = E' (m_0 I_0 + m_1 I_1 + \dots + m_{2^n-1} I_{2^n-1})$$

Multiplexer expansion

- We consider two multiplexers with 2 control (select) inputs and an enable input

↳ and construct a multiplexer with 3 control (select) inputs

- we have

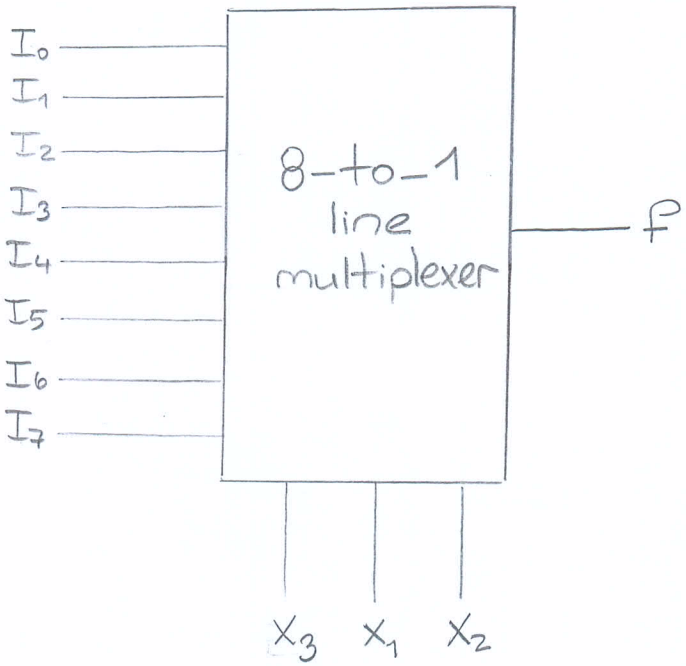


- and it follows that

$$f_1 = E_1' \left(I_{01} x_1' x_2' + I_{11} x_1' x_2 + I_{21} x_1 x_2' + I_{31} x_1 x_2 \right)$$

$$f_2 = E_2' \left(I_{02} x_1' x_2' + I_{12} x_1' x_2 + I_{22} x_1 x_2' + I_{32} x_1 x_2 \right)$$

- Now, we wish to implement



with

$$f = \left(I_0 x_3' x_1' x_2' + I_1 x_3' x_1' x_2 + I_2 x_3' x_1 x_2' + I_3 x_3' x_1 x_2 \right)$$
$$+ \left(I_4 x_3 x_1' x_2' + I_5 x_3 x_1' x_2 + I_6 x_3 x_1 x_2' + I_7 x_3 x_1 x_2 \right)$$
$$= \left[x_3' \left(I_0 x_1' x_2' + I_1 x_1' x_2 + I_2 x_1 x_2' + I_3 x_1 x_2 \right) \right]$$
$$+ \left[(x_3')' \left(I_4 x_1' x_2' + I_5 x_1' x_2 + I_6 x_1 x_2' + I_7 x_1 x_2 \right) \right]$$

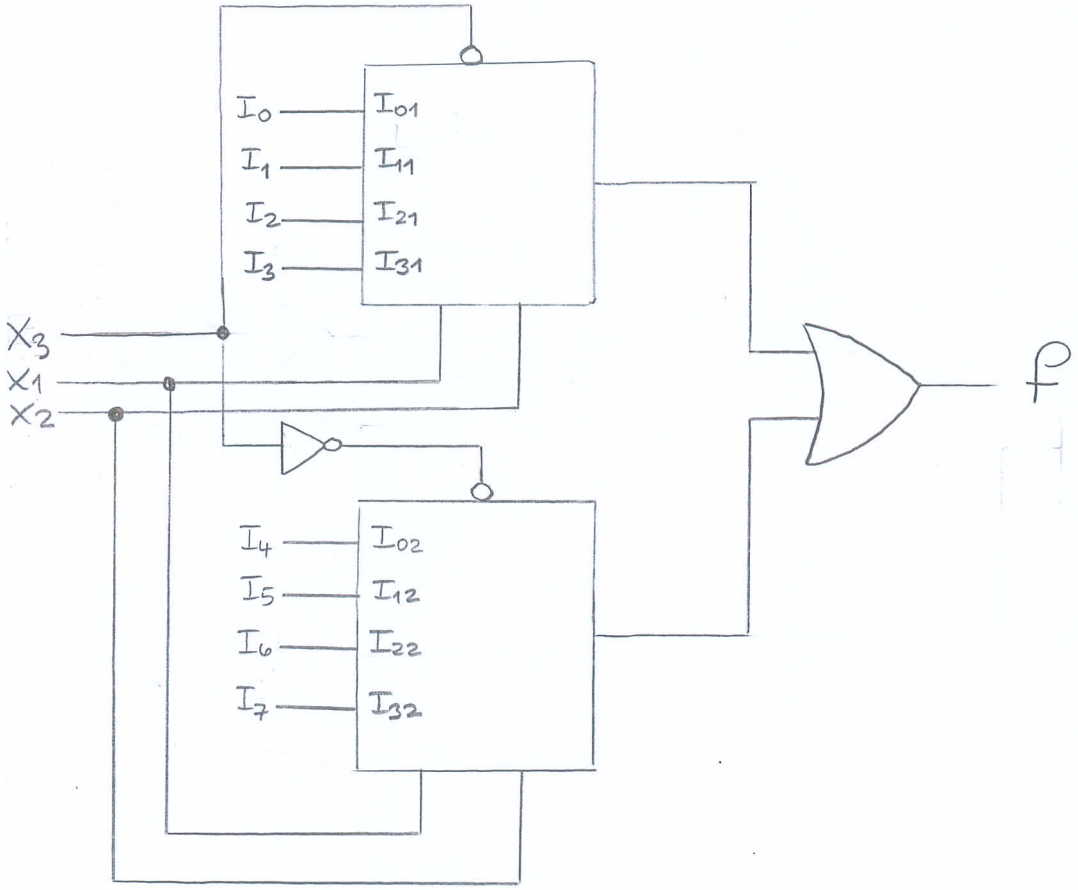
- let us set $E_1 \equiv x_3$, $I_{i1} \equiv I_i$, $i = 0,1,2,3$

and $E_2 = x_3'$, $I_{i2} \equiv I_j$, $i = 0,1,2,3$; $j = 4,5,6,7$

– therefore, we obtain

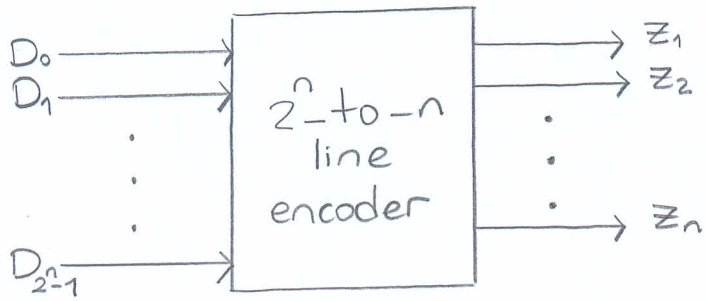
$$f = f_1 + f_2$$

for which the logic circuit diagram can be given as follows



More on circuit elements: Encoder

– a combinational circuit element with 2^n input and n output shown as



- only one of the 2^n inputs, $D_i = 1$, while remaining inputs are 0, i.e.

$$D_i = 1 \Rightarrow D_j = 0 \text{ for } j \neq i$$

- and if $D_i = 1$ then

$$(z_1 z_2 \ldots z_n)_2 \equiv (i)_{10}$$

example. let $n = 4$ and $D_{10} = 1$, then

$$z_1 z_2 z_3 z_4 = 1010 = (10)_{10}$$

Binary adder

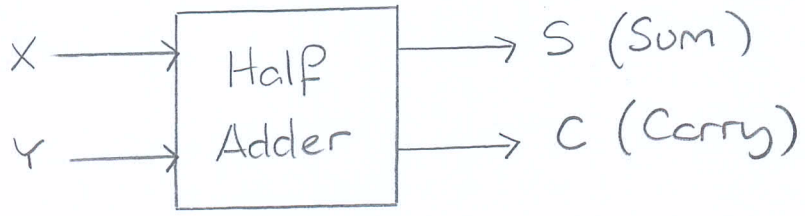
- we consider two types of binary adder:

i) Half adder

ii) full adder

Half adder

- used to add two single bits



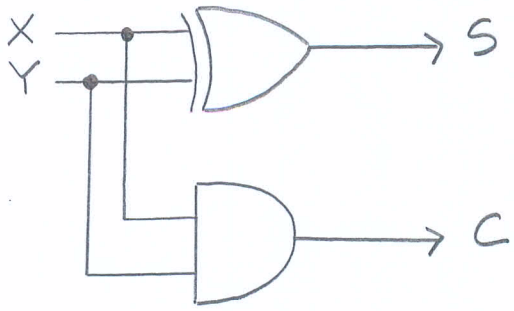
- the truth table is given by

X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = X'Y + XY' = X \oplus Y$$

$$C = XY$$

- the logic circuit implementation is given as

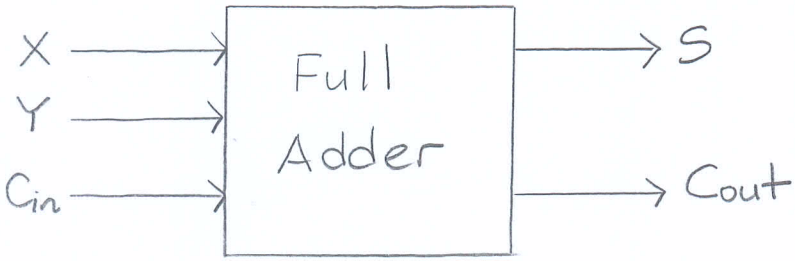


Full adder

- in order to add binary numbers with more than two single bits

  ↳ a carry input needs to be considered with a half adder

- this type of an adder is referred to as full adder



- some examples of IC's are

  7483 : 4-bit Binary Full Adder

  74283 : 4-bit Binary Full Adder with Fast Carry

  74181 : ALU

- the truth table can be given as follows

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

$$S = X'Y'C_{in} + X'Y C_{in}' + XY'C_{in}' + XY C_{in}$$

$$= (X'Y' + XY) C_{in} + (X'Y + XY') C_{in}'$$

$$= (X \oplus Y)' C_{in} + (X \oplus Y) C_{in}'$$

$$= X \oplus Y \oplus C_{in}$$

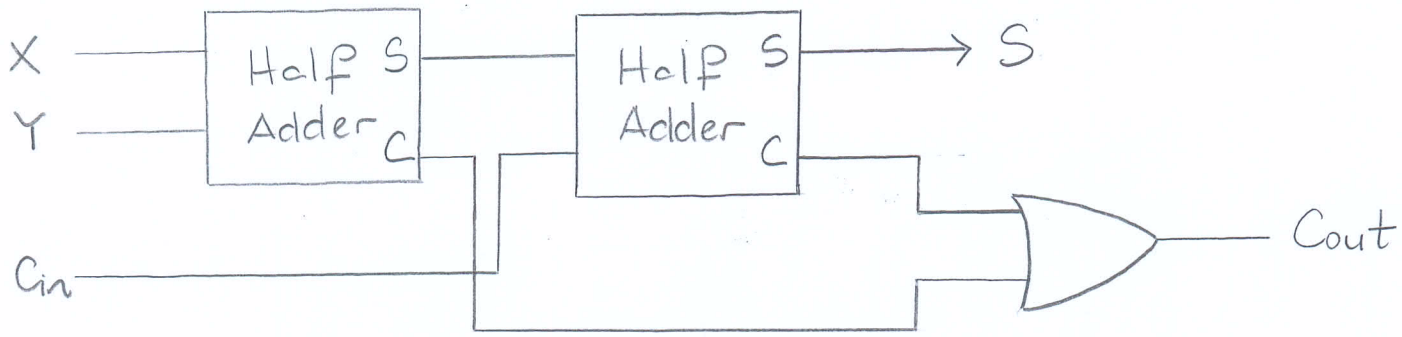
- and

$$C_{out} = X'Y C_{in} + XY'C_{in} + XY C_{in}' + XY C_{in}$$

$$= (X'Y + XY') C_{in} + XY (C_{in}' + C_{in})$$

$$= (X \oplus Y) C_{in} + XY$$

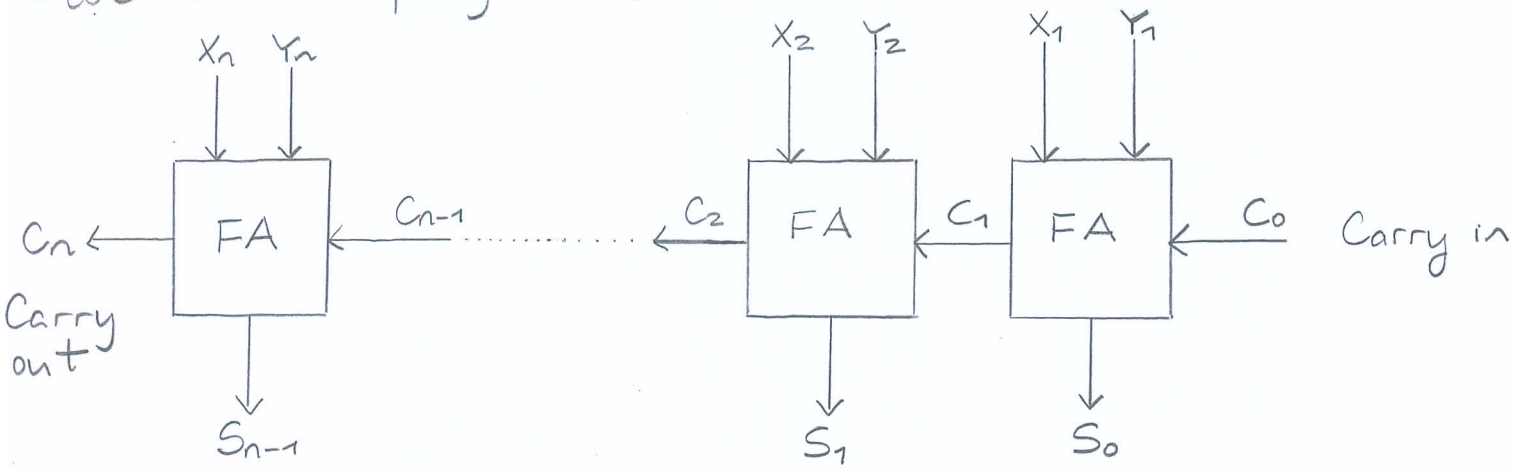
- then the full adder circuit is given in terms of half-adders



n-bit parallel adder

- now we consider the addition of two n-bit binary numbers

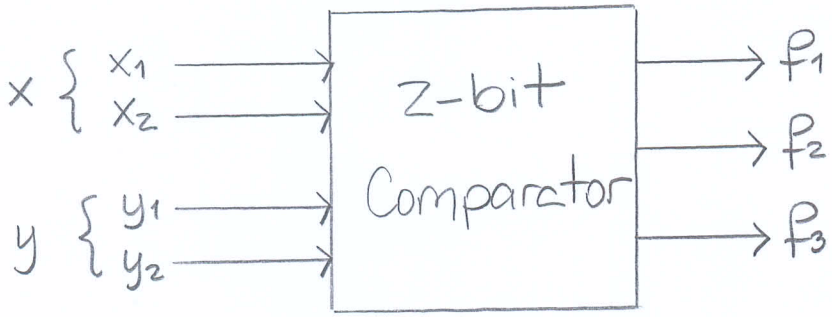
- we can employ n full-adders



Comparator

- a combinational logic circuit that compares two binary numbers

- a 2-bit comparator is represented as



- operates as follows

$x > y \Rightarrow f_1 = 1 \quad (f_2 = f_3 = 0)$

$x = y \Rightarrow f_2 = 1 \quad (f_1 = f_3 = 0)$

$x < y \Rightarrow f_3 = 1 \quad (f_1 = f_2 = 0)$