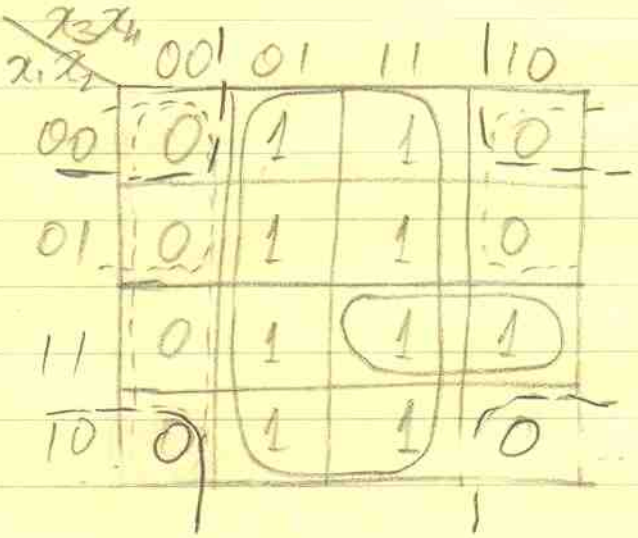
Implementation of Combinational Logic Circuits

$$F = \underbrace{L_1 + \cdots + L_m}_{m \text{ literals}} + \underbrace{P_1 + \cdots + P_k}_{k \text{ products}} \qquad \text{SOP}$$

NAND is complete, hence SOP can be implemented with NAND alone

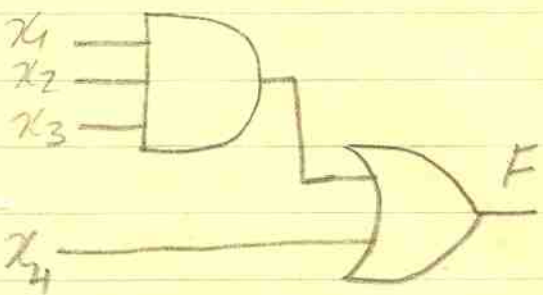
$$(F')' = (L_1' L_2' \cdots L_m' P_1' \cdots P_k')' = F$$

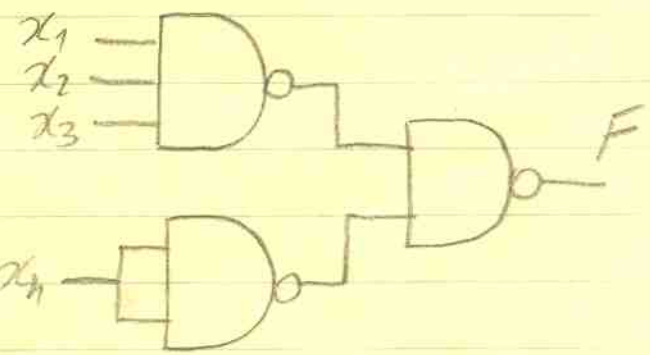
Example:



$$F = x_1 x_2 x_3 + x_4$$

$$F = (x_3 + x_4)(x_1 + x_4)(x_2 + x_4)$$

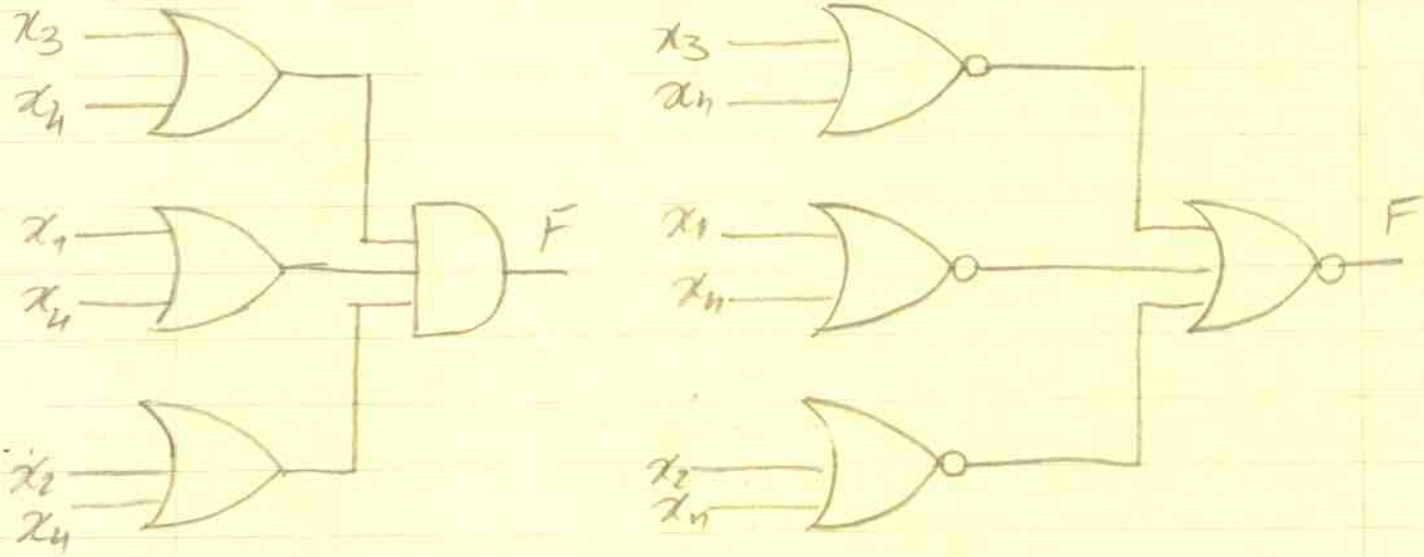




$F = \underbrace{L_1 L_2 \cdots L_m}_{m \text{ literals}} \underbrace{S_1 S_2 \cdots S_k}_{k \text{ sums}}$ POS

NOR is complete

$F = (F')' = (L_1' + L_2' + \cdots + L_m' + S_1' + S_2' + \cdots + S_k')'$



Multilevel Circuit Implementation

Propagation delay. 2-level is the fastest.

Fan-in limit

Example: $F = x_1x_2 + x_1x_3' + x_2x_3x_4$

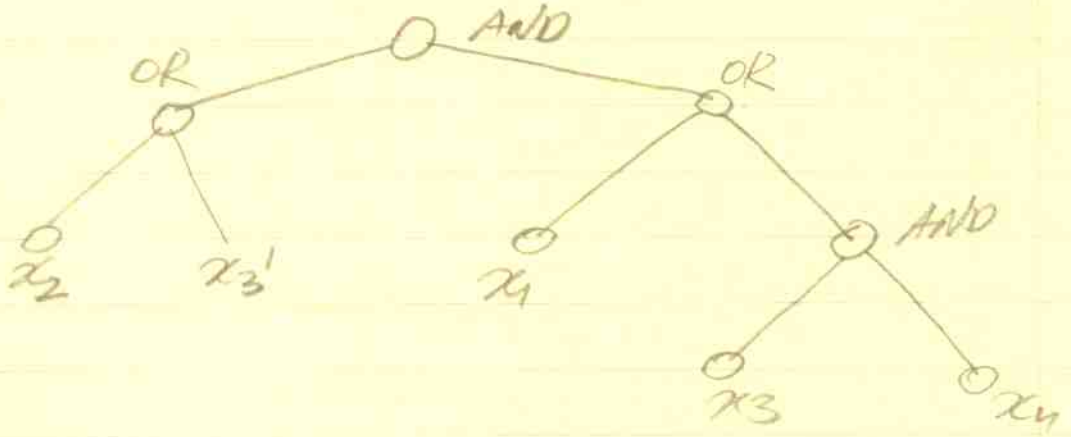
Fan-in is 3. Would like Fan-in 2 implementation

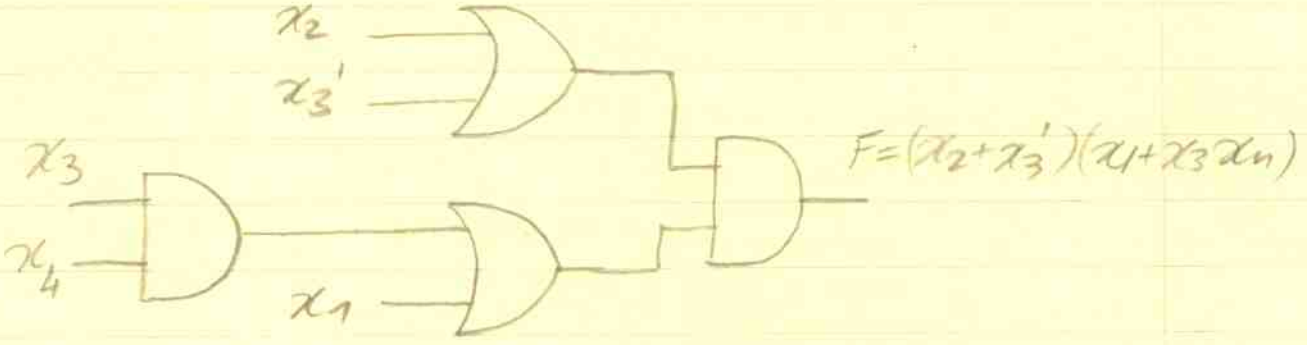
Problematic Problematic

$F = x_1x_2 + x_1x_3' + x_2x_3x_4 = x_1(x_2+x_3') + x_2x_3x_4 +$

$+ x_3x_3'x_4 = x_1(x_2+x_3') + x_3x_4(x_2+x_3') =$

$= (x_2+x_3')(x_1+x_3x_4)$ O.K.!





Multilevel NAND (NOR) Circuits

① Obtain minimal SOP (POS)

② Factor SOP (POS) to satisfy fan-in limit (AND)

③ Make output OR, traverse backwards AND-OR alternating (OR-AND)

④ Replace all gates with NAND (NOR)

⑤ Complements all inputs originally input to OR (AND)

Example: Implement with NOR and fan-in ≤ 3

$F = \Sigma(0,3,4,5,8,9,10,14,15)$

x_1x_2 \ x_3x_4	00	01	11	10
00	1	0	1	0
01	1	1	0	0
11	0	0	1	1
10	1	1	0	1

$F = (x_1'+x_2'+x_3)(x_1+x_2+x_3+x_4')(x_1'+x_2+x_3'+x_4')(x_1+x_2'+x_3)$

factor x_2+x_4' ; factor x_1+x_3'

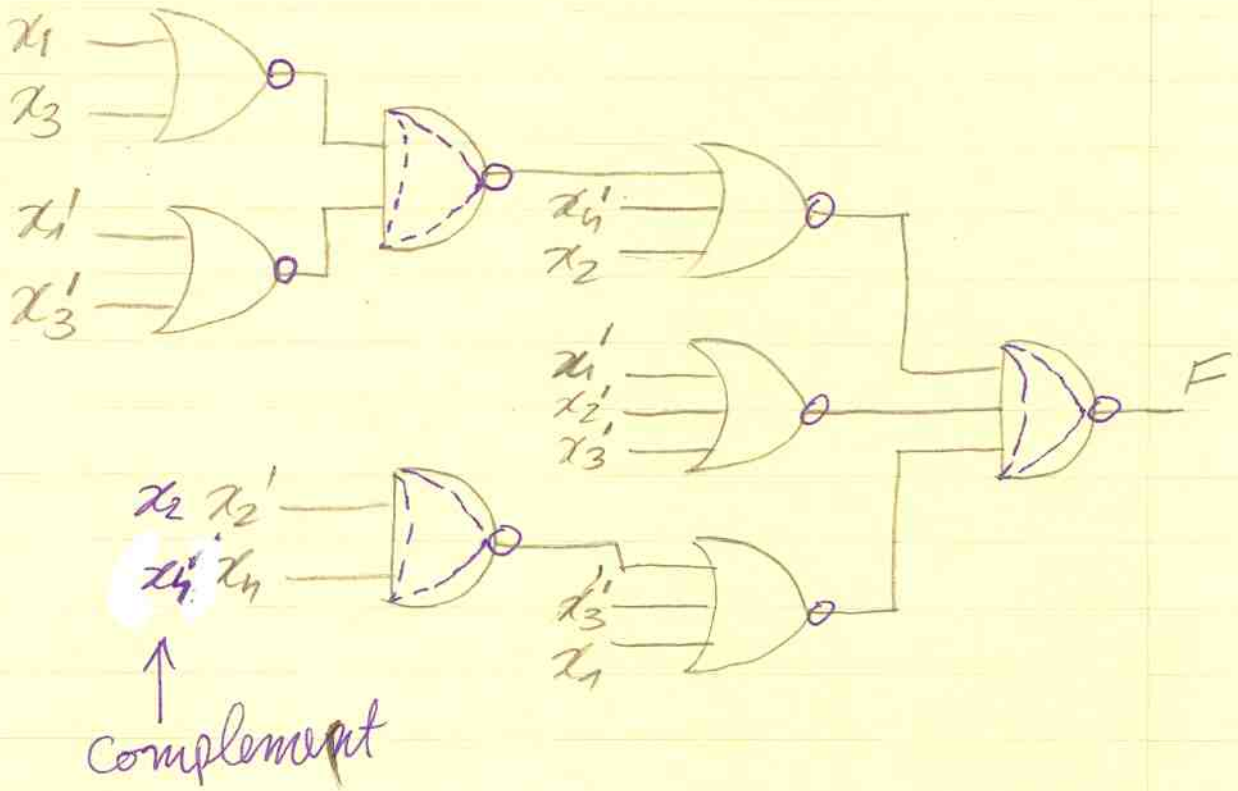
$(x_1+x_3'+x_4)$

AND: 5 fan-in, 2 OR: 4-fan in

$= (x_1'+x_2'+x_3)[(x_2+x_4')+(x_1+x_3)(x_1'+x_3')]$

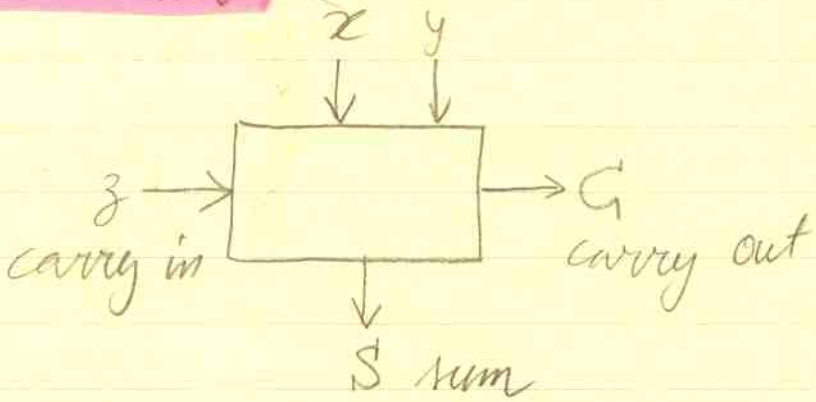
$[(x_1+x_3')+x_2'x_4]$

We obtained products and sums of 3 at most



Arithmetic Circuits

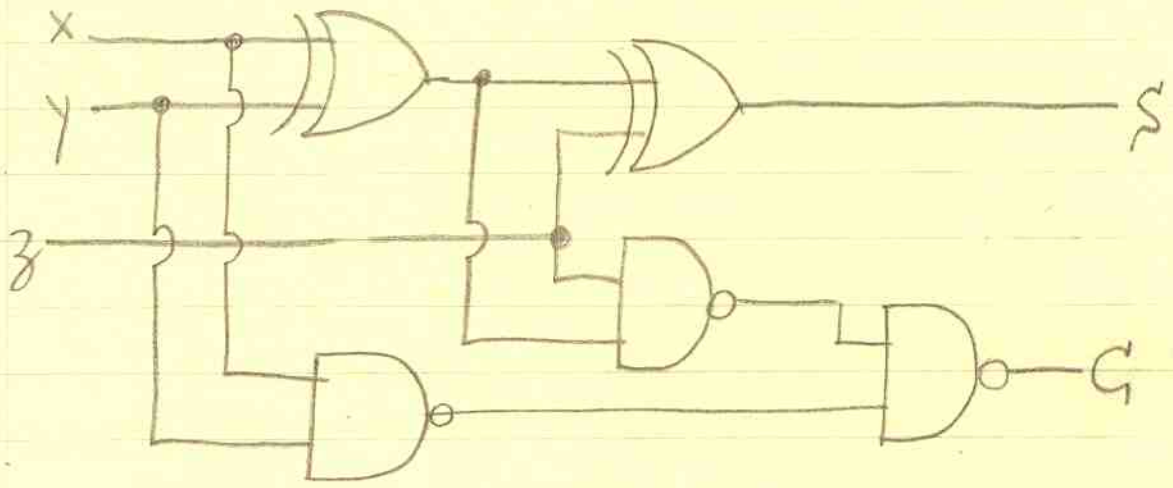
Full adder



$$S = x \oplus y \oplus z$$

$$C = (x \oplus y)z + xy = [(x \oplus y)z]'(xy)']'$$

NAND implementation



Example: Binary to gray code converter

In gray code successive decimal numbers are coded such that they differ on 1-bit only

Decimal	Binary	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
	⋮	
13	1101	1011
14	1110	1001
15	1111	1000

conversion

Gray code of two successive binary is based on the number of successive 01 or 10 in the binary code

The addition of 00...01 to a binary number will change by 1 exactly the 01 or 10.

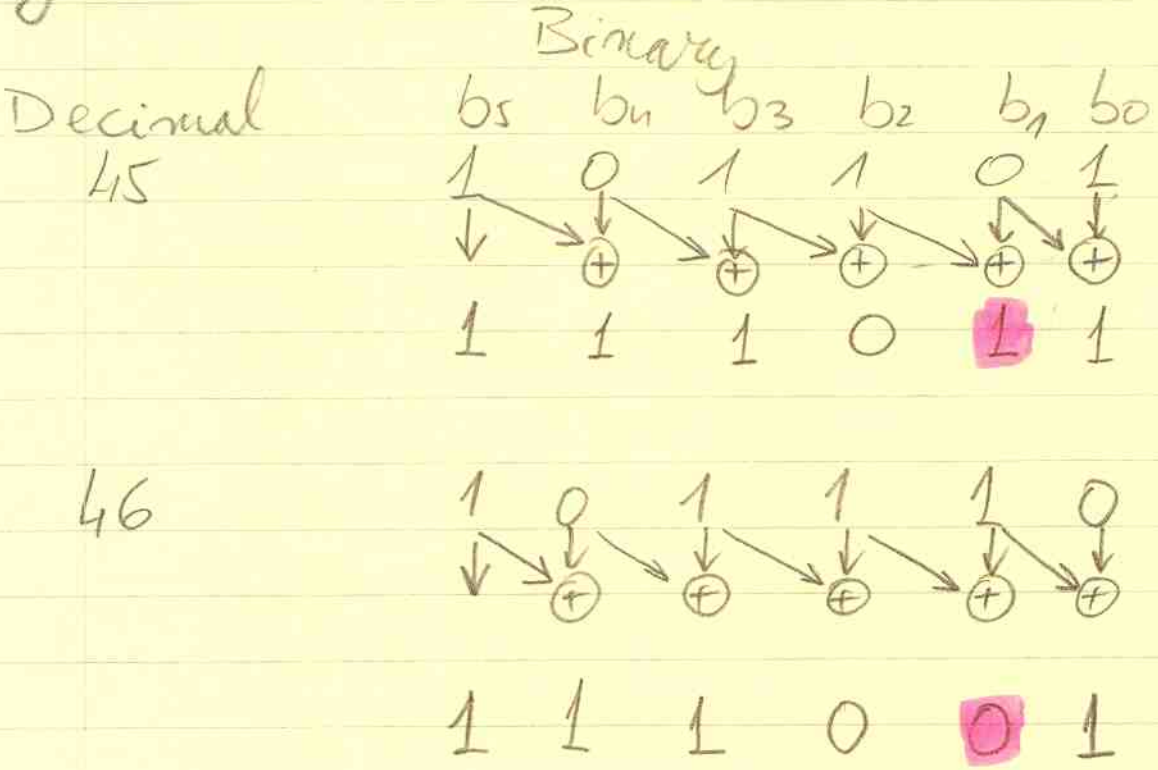
Hence pairity check by ⊕ will do the work.

$b_{n-1} b_{n-2} \cdots b_2 b_0$ - binary

$g_{n-1} g_{n-2} \cdots g_1 g_0$ - Gray

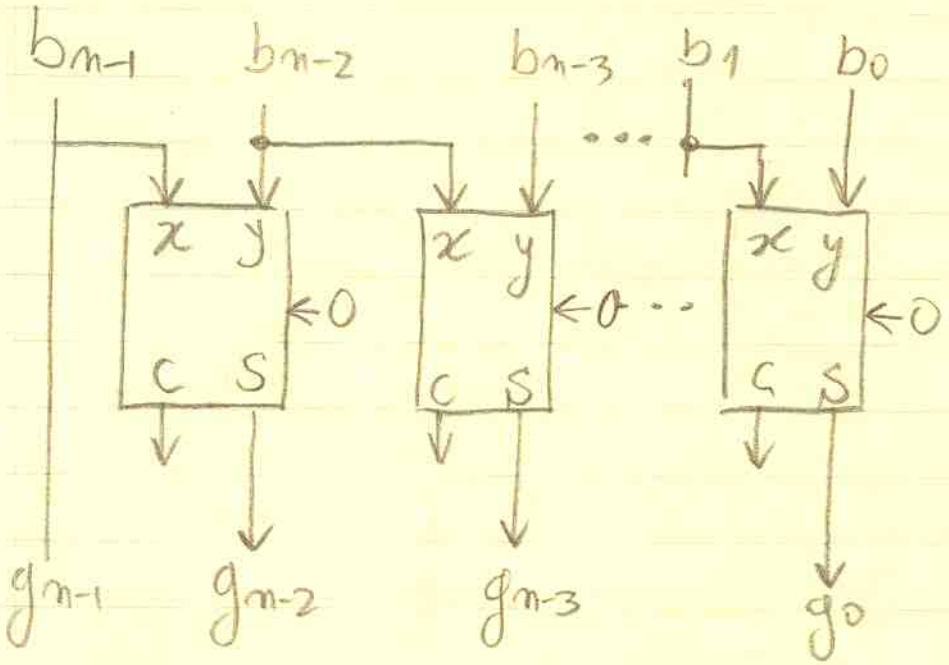
$g_i = b_i \oplus b_{i+1} \quad 0 \le i \le n-2$

$g_{n-1} = b_{n-1}$



We can use full adder to implement the converter

Homework: prove that obtained gray code is unique



Homework: Design subtractor (full)

x - minuend, y - subtrahend, z - borrow

$$x - y - z = x - (y + z)$$

① Write truth table

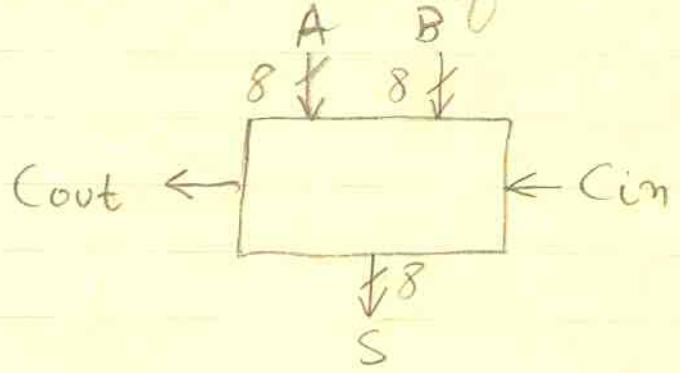
Inputs			outputs	
x	y	z	borrow	difference
1	1	1	1	1
0	1	1	1	0
0	1	0	1	1
...			...	

② k-map

③ Implement

Binary Parallel Adder

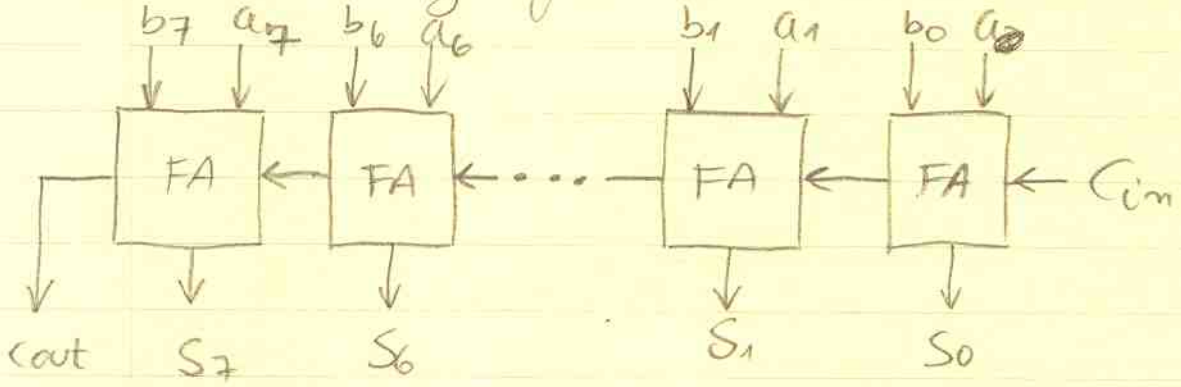
Consider addition of two 8-bit binary numbers



$A = (a_7 a_6 \cdots a_0)$

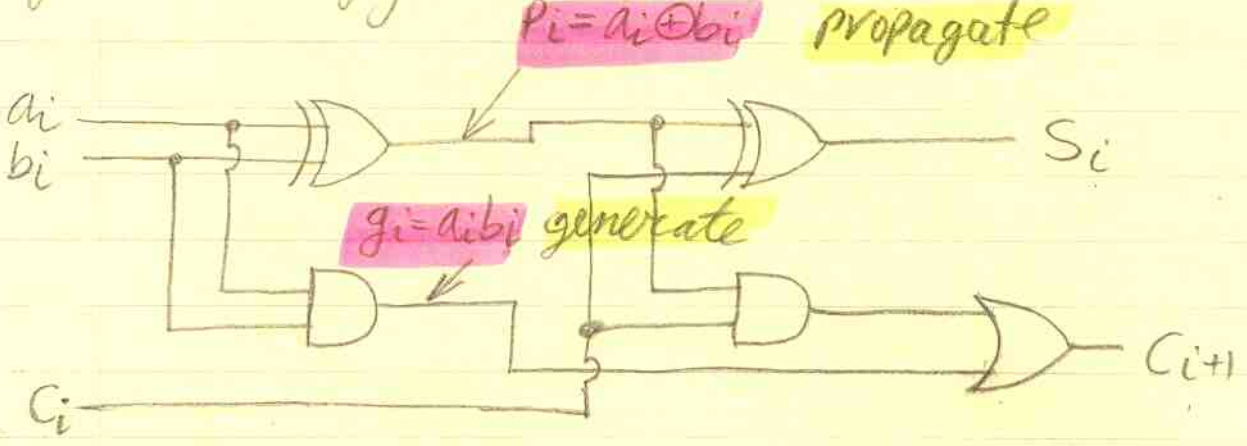
$B = (b_7 b_6 \cdots b_0)$

What is the size of Truth table?



Problem: carry propagation

n-bit parallel adder delay is $O(n)$ regardless of technology



The output carry is a sum (OR) of two variables

1) $g_i = a_i b_i$ - a carry out is generated if $a_i = 1$ and $b_i = 1$

2) $p_i c_i$, namely the carry in is propagated if $p_i = 1$

carry independent

$$p_i = a_i \oplus b_i \quad , \quad g_i = a_i b_i$$

recursive expression

$$S_i = p_i \oplus c_i \quad , \quad c_{i+1} = g_i + p_i c_i$$

We could evaluate the carry out of each adder as follows

$$c_{i+1} = g_i + p_i c_i = g_i + p_i (g_{i-1} + p_{i-1} c_{i-1}) =$$

$$= g_i + p_i g_{i-1} + p_i p_{i-1} (g_{i-2} + p_{i-2} \cdot c_{i-2}) =$$

$$= g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + p_i p_{i-1} p_{i-2} (g_{i-3} + p_{i-3} c_{i-3})$$

$$= g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \cdots + p_i p_{i-1} \cdots p_1 g_0 + p_i p_{i-1} \cdots p_0 c_0$$

We obtained SOP of known variables

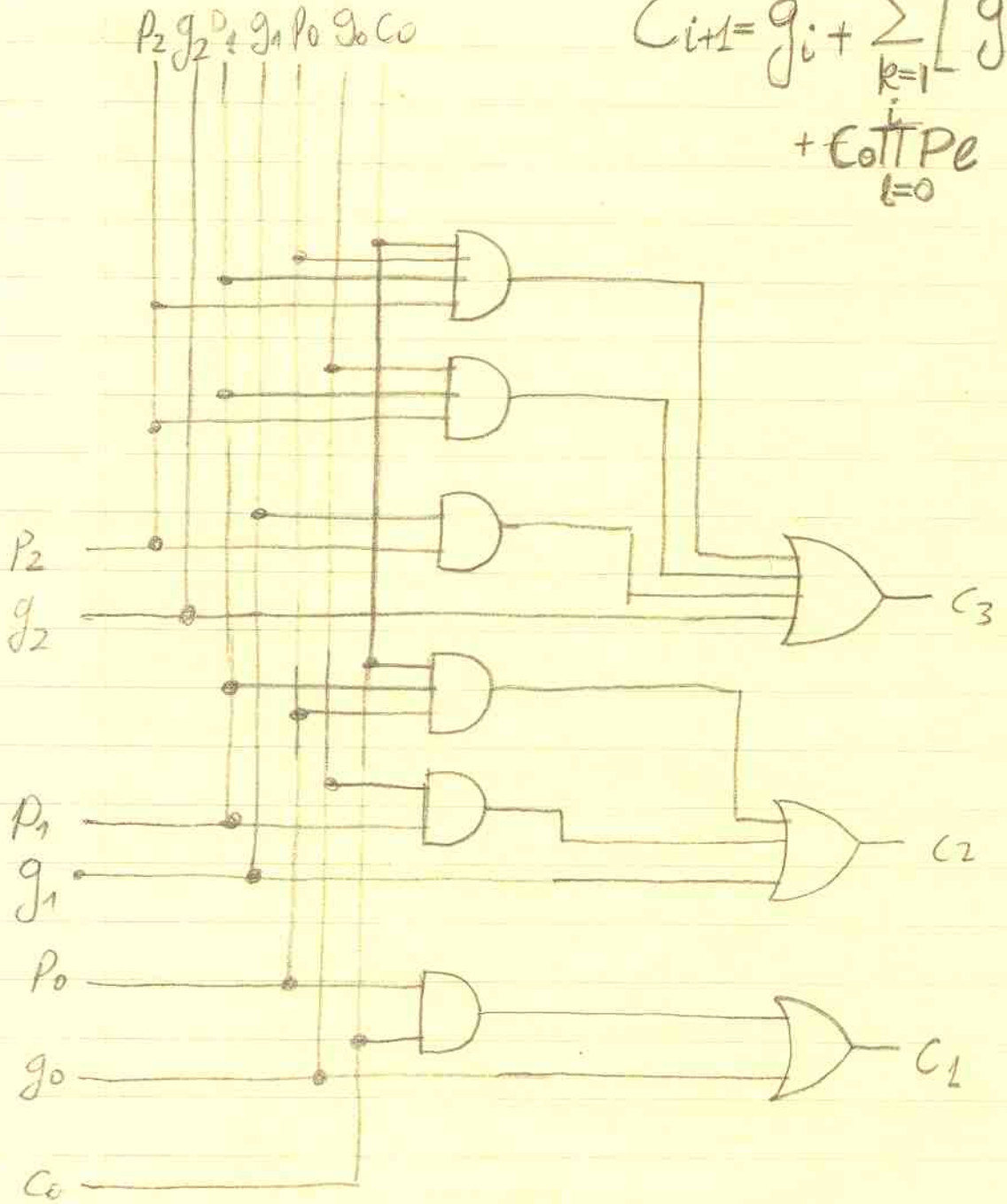
Hardware implementation 3-bit adder

$C_1 = g_0 + P_0 C_0$

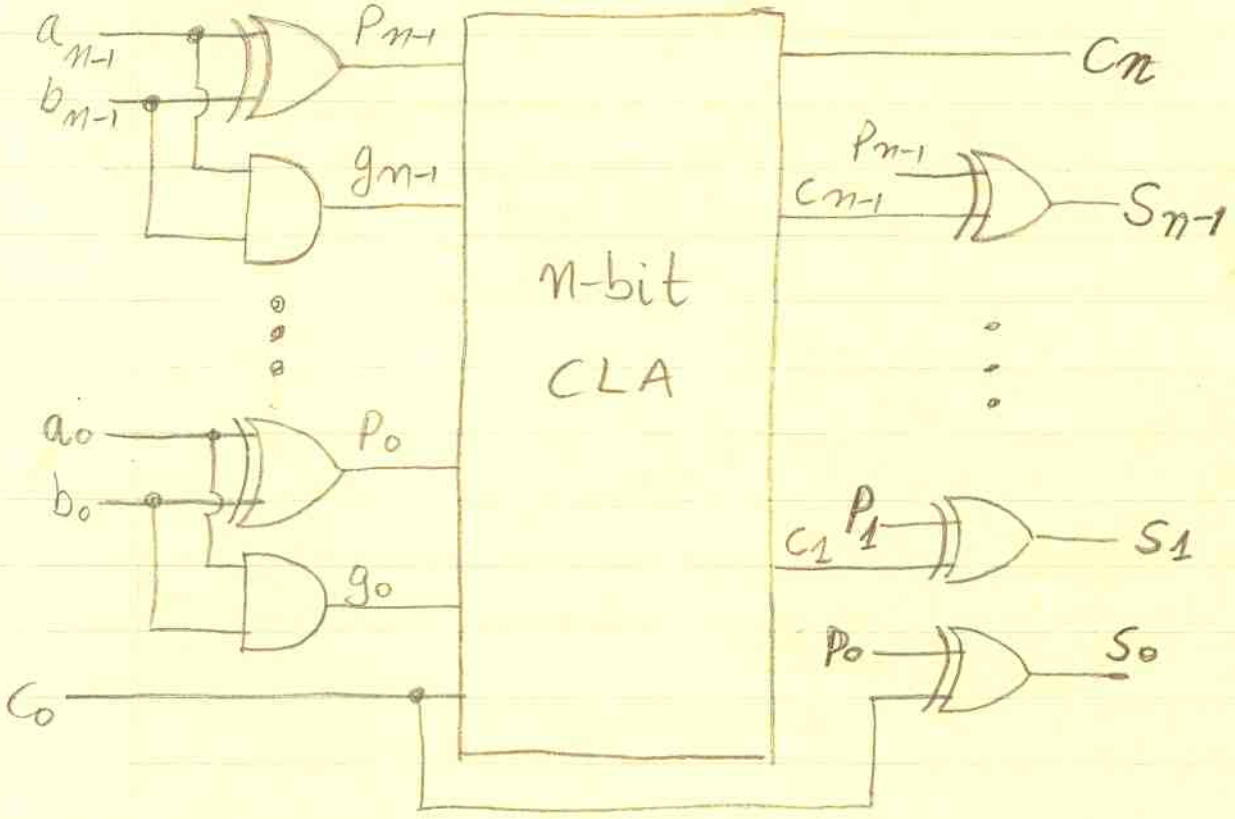
$C_2 = g_1 + P_1 g_0 + P_1 P_0 C_0$

$C_3 = g_2 + P_2 g_1 + P_2 P_1 g_0 + P_2 P_1 P_0 C_0$

$$C_{i+1} = g_i + \sum_{k=1}^{i} \left[g_{i-k} \prod_{l=0}^{i-1} P_{i-l} \right] + C_0 \prod_{l=0}^{i} P_l$$



Combining everything together



and AND trees

Comments:

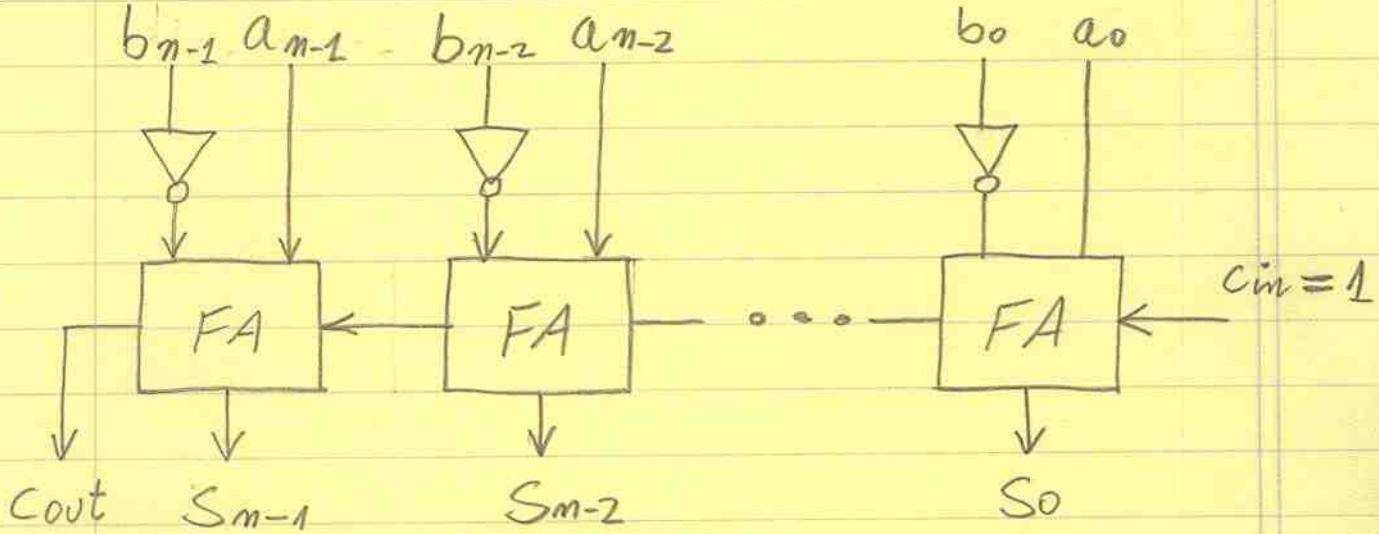
1- CLA by SOP is $O(1)$ but fan-in limit may require OR trees, Hence evaluation time is $O(\log n)$

Show Homework

2- Hardware of CLA is big, in fact $O(n \log n)$

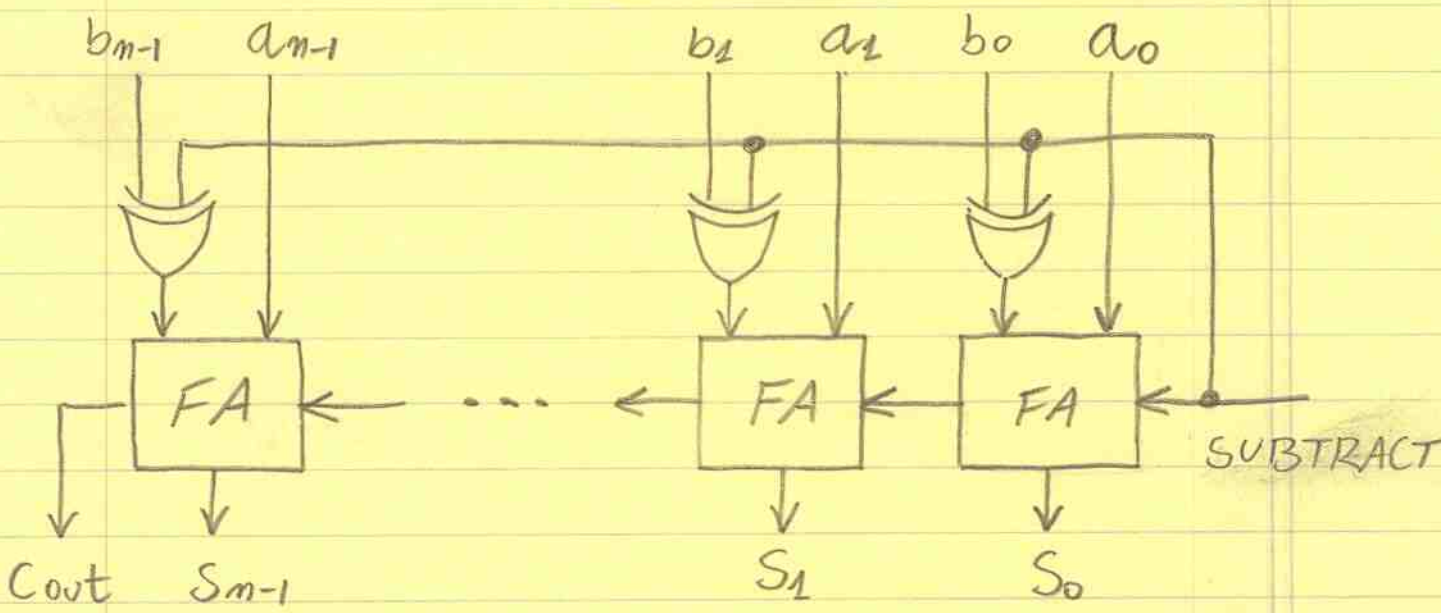
Using Adder for Subtraction

Parallel adder can be used for subtraction by taking the 2's complement of the subtrahend. It is implemented by inverting the bits and adding 1 via the carry in



The same unit can perform both addition and subtraction by entering $\bar{b}$ via XOR controlled to pass it unchanged in case of addition or invert it in subtraction

70.1



if (SUBTRACT == 0)

$\bar{S} = \bar{a} + \bar{b}$ (vectors)

else

$\bar{S} = \bar{a} - \bar{b}$

Carry-Save addition

Carry look ahead adder adds 2 n-bit numbers in $O(\log_2 n)$ time. Adding 3 n-bit numbers doesn't need two CLA additions!

Let $\bar{x} = (x_{n-1}, x_{n-2}, \ldots, x_0)$

$\bar{y} = (y_{n-1}, y_{n-2}, \ldots, y_0)$

$\bar{z} = (z_{n-1}, z_{n-2}, \ldots, z_0)$

$$\begin{array}{llllllll} & \bar{x} & x_{n-1} & x_{n-2} \cdots & x_i & \cdots & x_1 & x_0 \\ + & & + & & & & & \\ & \bar{y} = & y_{n-1} & y_{n-2} \cdots & y_i & \cdots & y_1 & y_0 \\ + & \bar{z} & + z_{n-1} & z_{n-2} \cdots & z_i & \cdots & z_1 & z_0 \end{array}$$

Consider the addition on bit i. There are two implications:

- The result on s_i is the Parity of x_i, y_i, z_i and the carry from $i-1$
- The carry c_i is the majority of the above

Hence, we can represent $x+y+z$ as:

$x + y + z = u + v$, where

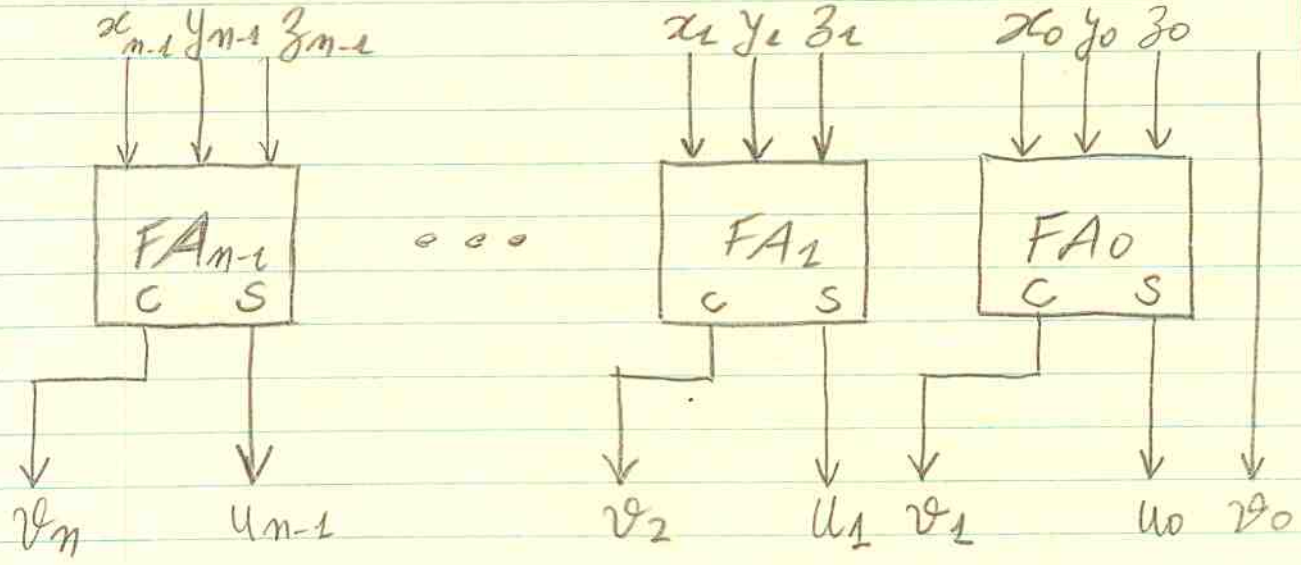
u_i = parity (x_i, y_i, z_i)

v_{i+1} = majority (x_i, y_i, z_i)

$v_0 = 0$

Example:

i	8	7	6	5	4	3	2	1	0
$\bar{x}$		0	0	0	1	1	1	0	1
$\bar{y}$		1	1	0	0	0	1	0	1
$\bar{z}$		1	0	0	1	0	1	1	0
$\bar{u}$		0	1	0	0	1	1	1	0
$\bar{v}$	1	0	0	1	0	1	0	1	0



$\overline{u}$ and $\overline{v}$ are then summed by CLA. Notice that the generation of $\overline{u}$ and $\overline{v}$ takes $O(1)$ time.

Carry-Save adder is useful for fast multiplication by Wallace-Tree multiplier.

(3)

Multiplier (Unsigned)

$$\begin{array}{cccc} x_4 & x_3 & x_2 & x_1 \\ y_4 & y_3 & y_2 & y_1 \end{array}$$

$$\begin{array}{cccc} x_4y_1 & x_3y_1 & x_2y_1 & x_1y_1 \\ x_4y_2 & x_3y_2 & x_2y_2 & x_1y_2 \\ x_4y_3 & x_3y_3 & x_2y_3 & x_1y_3 \\ x_4y_4 & x_3y_4 & x_2y_4 & x_1y_4 \end{array}$$

① There are 16 partial products $x_i y_j$ $1 \le i,j \le 4$

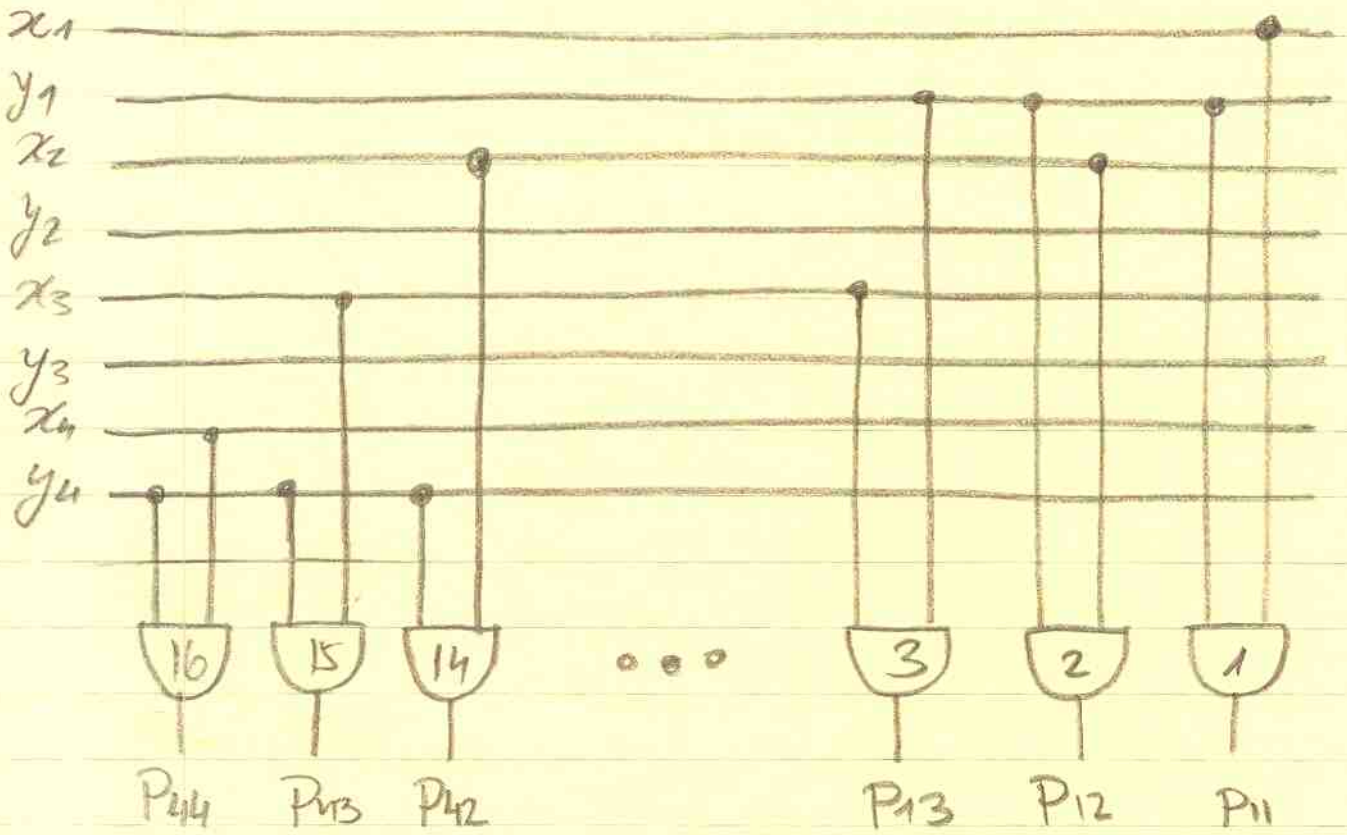
We'll therefore need 16 AND gates to produce

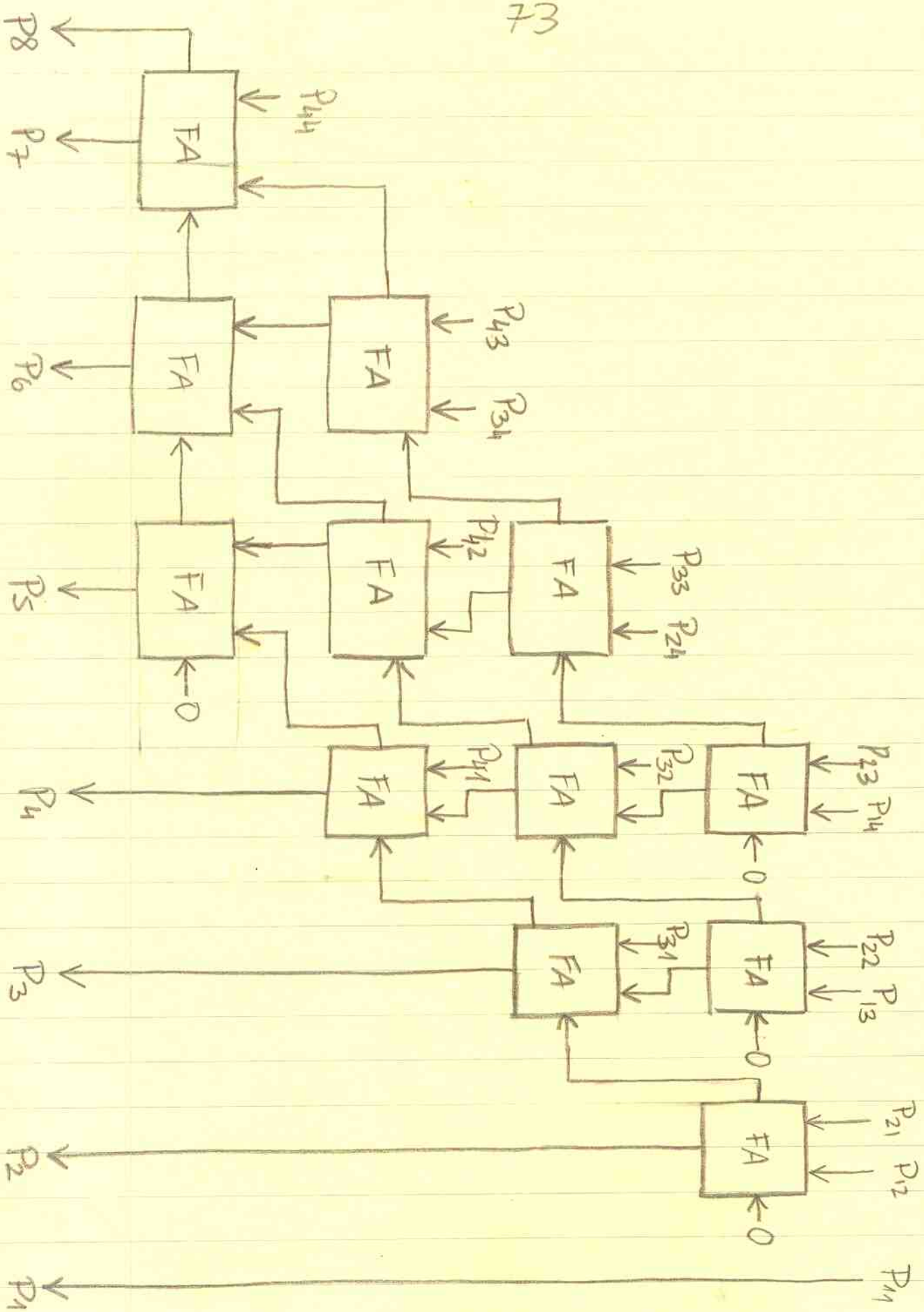
$$P_{ij} = x_j y_i \qquad 1 \le i,j \le 4$$

② Notice that for k's column sum there exist

$$k = i+j-1 \; ; \; k = 1, \ldots, 7$$

We use carry propagate (parallel) addition as follows:


x1
y1
x2
y2
x3
y3
x4
y4
16
15
14
3
2
1
P44
P43
P42
P13
P12
P11


P8
P7
P6
P5
P4
P3
P2
P1
P44
P43
P34
P42
P33
P24
P41
P32
P23
P14
P31
P22
P13
P21
P12
P11
FA
0

2's complement Multiplier

Recall that the most significant bit has negative weight (sign bit), hence some of the (Partial) products may be negative and must be subtracted.

Let

$$X = -x_{n-1}2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i \qquad n\text{-bit}$$

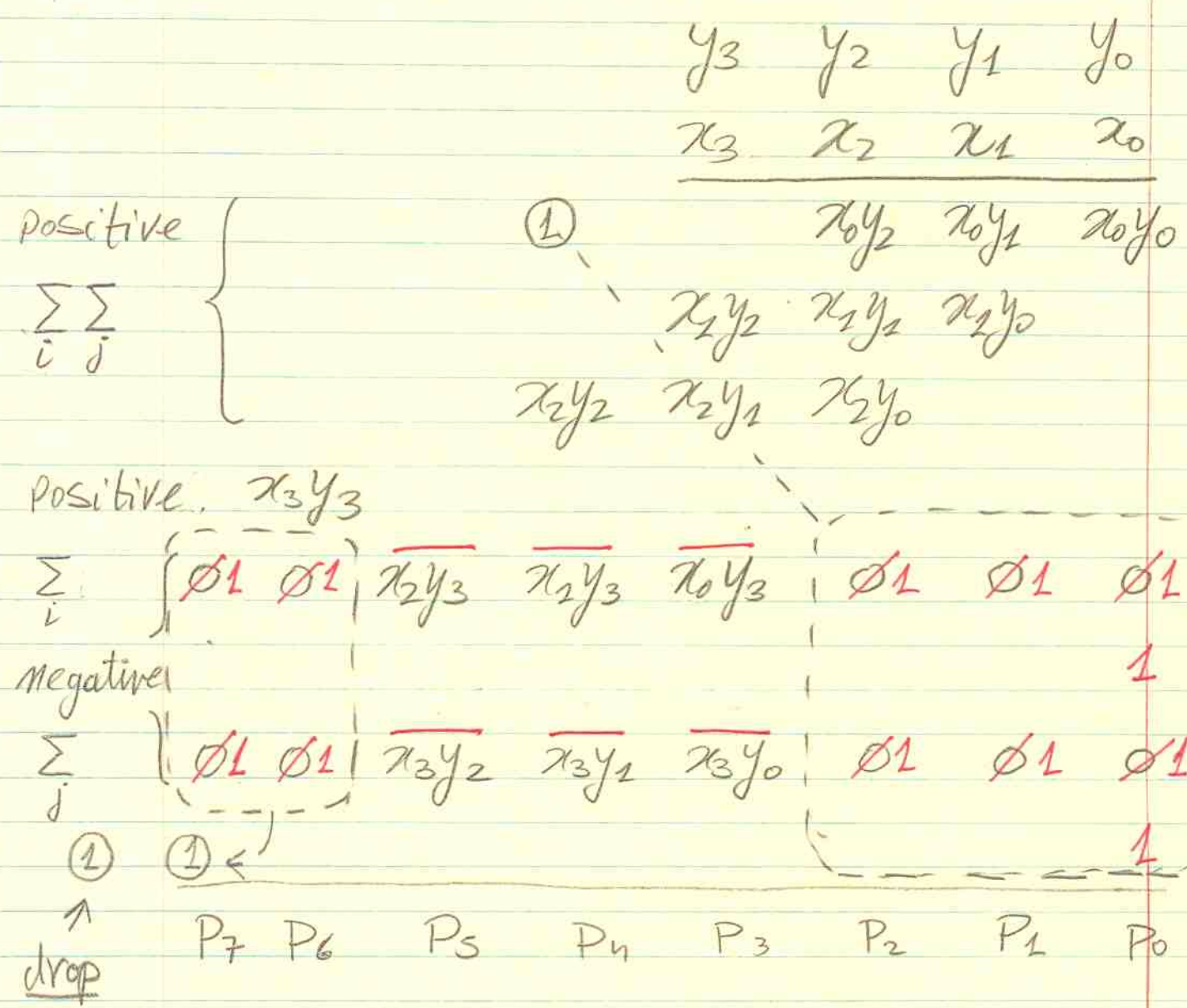
$$Y = -y_{m-1}2^{m-1} + \sum_{j=0}^{m-2} y_j 2^j \qquad m\text{-bit}$$

$$P = XY = \sum_{i=0}^{n-2}\sum_{j=0}^{m-2} x_i y_j 2^{i+j} +$$

$$+ x_{n-1}y_{m-1}2^{m+n-2} -$$

$$-\left(\sum_{i=0}^{n-2} x_i y_{m-1} 2^{i+m-1} + \sum_{j=0}^{m-2} x_{n-1} y_j 2^{j+n-1}\right)$$

Example: x and y are 4-bit each result will occupy 7 bits



The reason for adding implicit $n+m+1$'th bit (in this case 7-th bit) is because the result of multiplication of $(n+1)$-bit and $(m+1)$ bit numbers is obtained in $n+m+2$ bits.

$$
\begin{array}{cccccccc}
 & & & & y_3 & y_2 & y_1 & y_0 \\
 & & & & x_3 & x_2 & x_1 & x_0 \\
\hline
 & & & 1 & \overline{x_0 y_3} & x_0 y_2 & x_0 y_1 & x_0 y_0 \\
 & & & \overline{x_1 y_3} & x_1 y_2 & x_1 y_1 & x_1 y_0 & \\
 & & \overline{x_2 y_3} & x_2 y_2 & x_2 y_1 & x_2 y_0 & & \\
 & 1 & x_3 y_3 & \overline{x_3 y_2} & \overline{x_3 y_1} & \overline{x_3 y_0} & & \\
\hline
P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0
\end{array}
$$

Same hardware as for unsigned multiplica-tion can be used.

Comparators

Compares two n-bit numbers lexicographically: from most significant to least significant

$A = 1\,0\,0\,1\,1\,1\,0\,1\,0$
$B = 1\,0\,0\,1\,1\,0\,0\,1\,1$ $\Rightarrow A > B$

Inputs	outputs G	E	L
$A > B$	1	0	0
$A = B$	0	1	0
$A < B$	0	0	1

Let $A = (a_{n-1} \dots a_0)$
$B = (b_{n-1} \dots b_0)$

Define $x_i = a_i \odot b_i$ ($a_i b_i + a_i' b_i'$ (equivalance))

Then: $A = B$ $(E = 1)$ iff $\prod_{i=0}^{n-1} x_i = 1$

$$E = \prod_{i=0}^{n-1} x_i$$

There exists $a_i > b_i$ iff $a_i b_i' = 1$

Then $A > B$ $(G = 1)$ iff

$$a_{n-1} b'_{n-1} + x_{n-1} a_{n-2} b'_{n-2} + x_{n-1} x_{n-2} a_{n-3} b'_{n-3} + \\ + \cdots + x_{n-1} x_{n-2} \cdots x_1 a_0 b'_0$$

$$G = \sum_{i=0}^{n-1} \left(\prod_{k=0}^{i} x_{n-k} \right) a_{(n-1)-i} \, b'_{(n-1)-i}$$

where $x_n = 1$ by definition

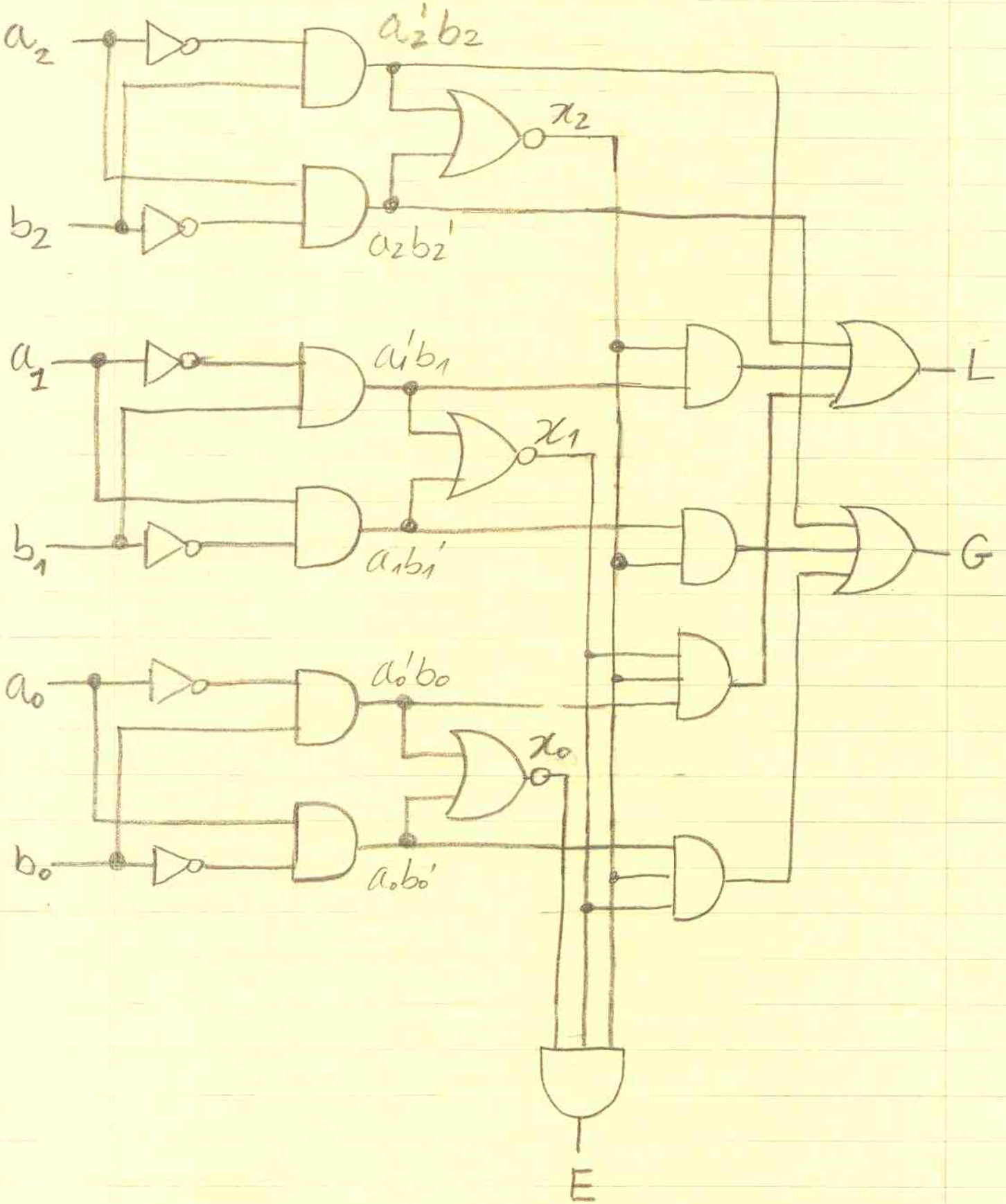
Similarly

$$L = \sum_{i=0}^{n-1} \left(\prod_{k=0}^{i} x_{n-k} \right) a'_{(n-1)-i} \, b_{(n-1)-i}$$

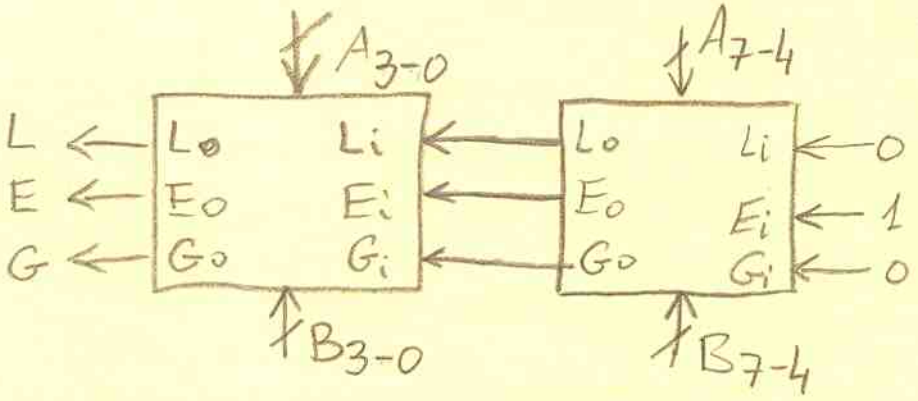
Example: 3-bit comparator

It has 6 inputs $a_0, \ldots, a_2, b_0, \ldots, b_2$ hence ordinary flow with truth table will have $2^6 = 64$ entries, which will be very difficult to minimize (unless Quine-McCluskey is used).

Fan-in may also be a problem

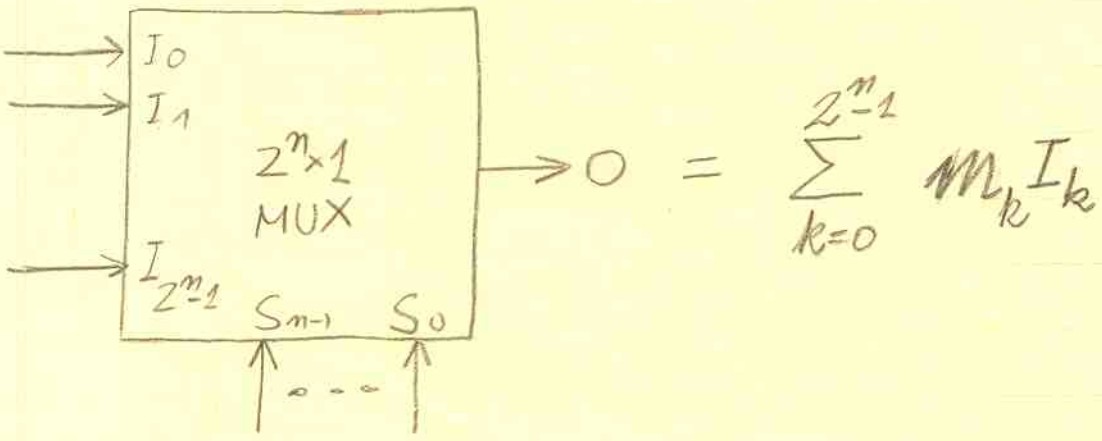

a_2
b_2
$a_2'b_2$
x_2
a_2b_2'
a_1
b_1
$a_1'b_1$
x_1
a_1b_1'
a_0
b_0
$a_0'b_0$
x_0
a_0b_0'
L
G
E

Comparators can be cascaded. It requires 3 inputs E_{in}, L_{in}, G_{in} in addition to A and B, and produces E_{out}, L_{out}, G_{out}. The internal Logic is enhanced to account for new inputs



Homework: Modify comparator to account L, E, G at inputs.

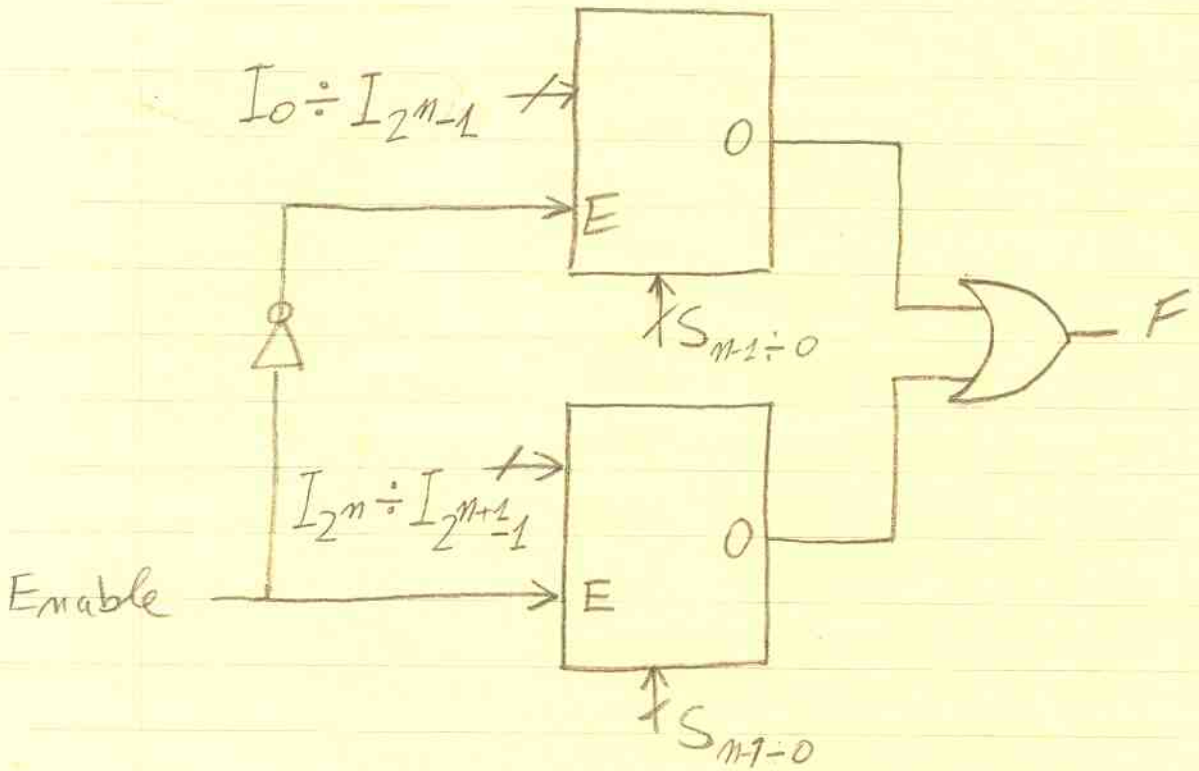
Multiplexers



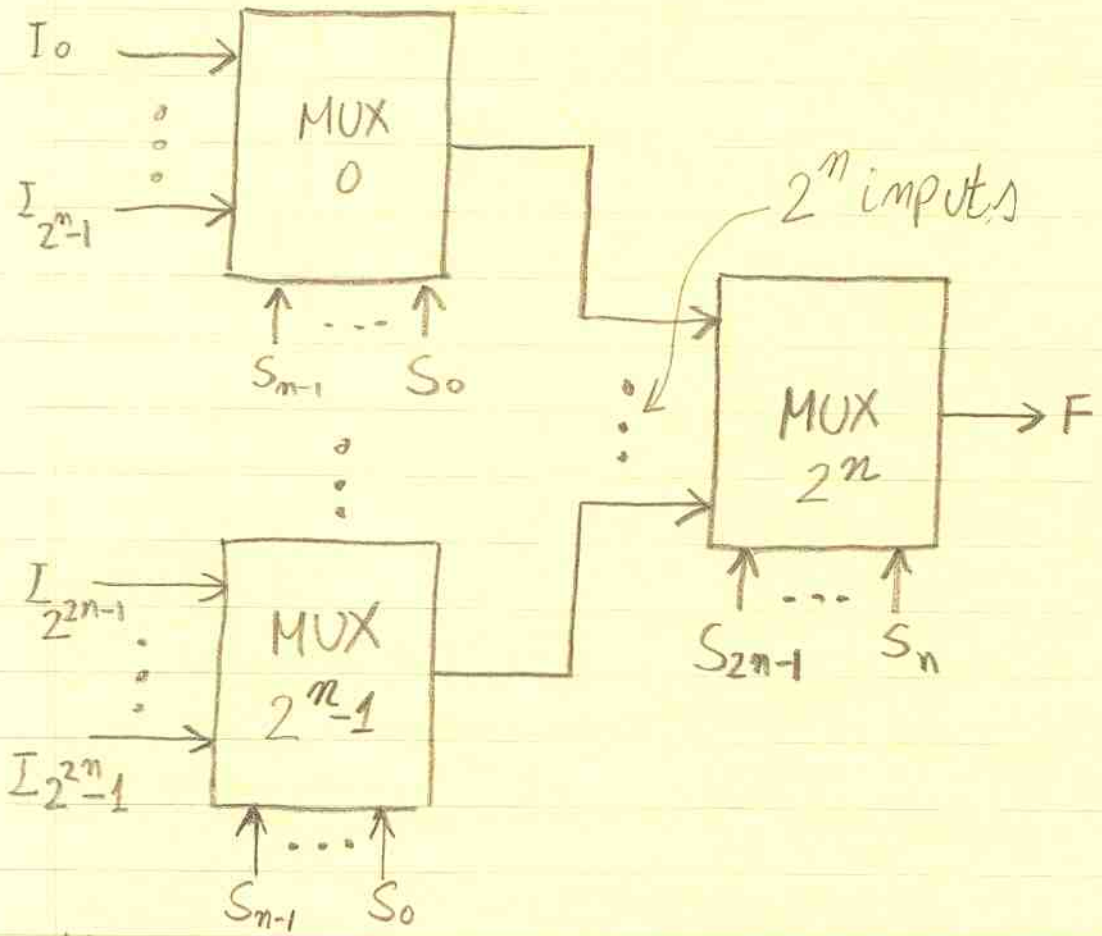
2^n AND gates for $I_0, \ldots, I_{2^n-1}$

(n+1)-input AND gates are AND-ing M_k and I_k.

An E (enable) is also possible for the sake of building MUX trees. E is input of any AND



Selection is made with $S_{n-1} \cdots S_0$ and E, hence selecting one of 2^{n+1} inputs



Multiplexers as Universal Logic Module

Any Boolean function of $n+1$ variables can be implemented by $2^n \times 1$ mux by assigning n variables to selection lines, say $x_0, \ldots, x_{n-1}$, and $\{x_n, x_n', 0, 1\}$ to the 2^n input lines.

Example: $F = \sum(0,1,3,6,7,8,11,12,14)$

$$F = x_1'x_2'x_3'x_4' + x_1'x_2'x_3x_4 + x_1'x_2x_3'x_4 + \underbrace{x_1'x_2x_3x_4'}_{+} + x_1'x_2x_3x_4$$

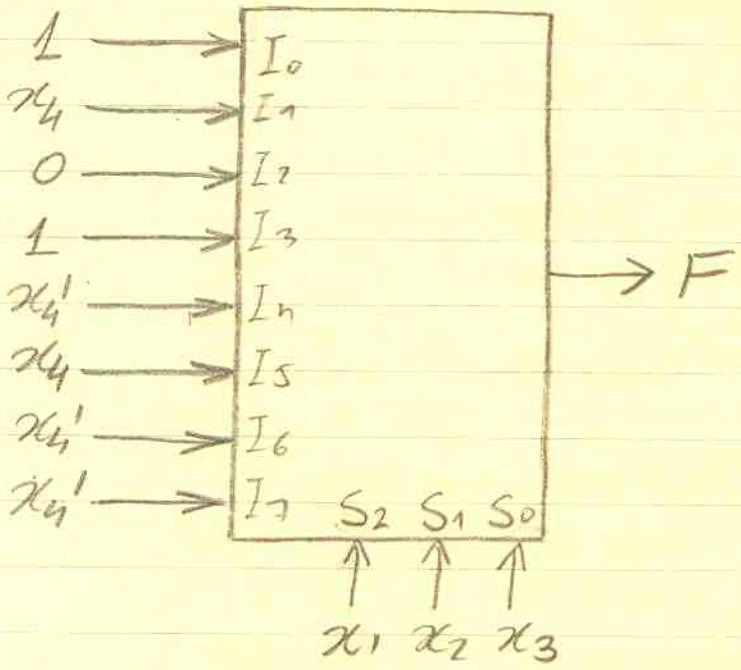
$$+ x_1x_2'x_3'x_4' + x_1x_2'x_3x_4 + x_1x_2x_3'x_4' + x_1x_2x_3x_4 =$$

$$= m_0x_4' + m_0x_4 + m_1x_4 + m_3x_4' + m_3x_4 + m_4x_4'$$

$$m_5x_4 + m_6x_4' + m_7x_4' =$$

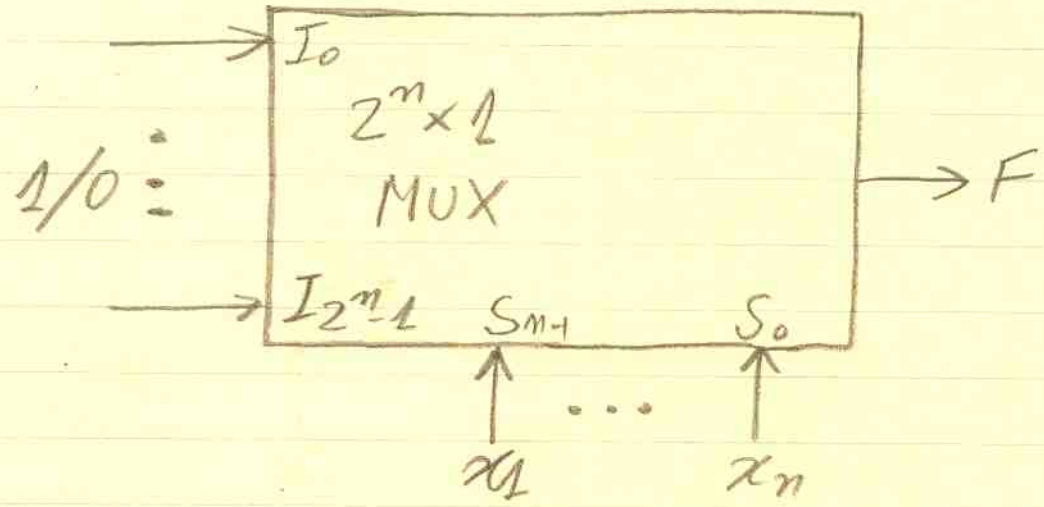
$$= m_0 \cdot 1 + m_1x_4 + m_2 \cdot 0 + m_3 \cdot 1 + m_4x_4' +$$

$$m_5x_4 + m_6x_4' + m_7x_4'$$



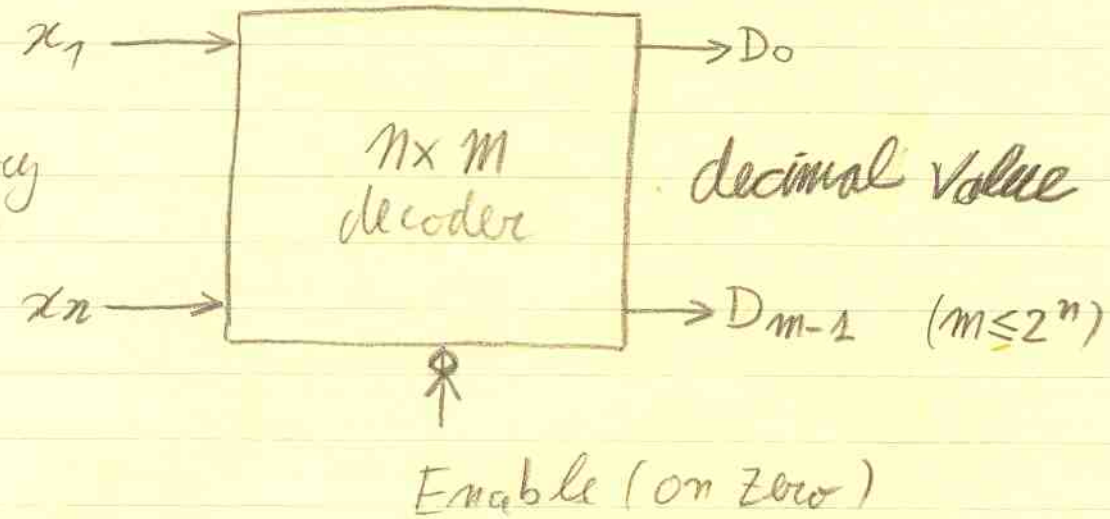
Notice that the assignment of variables to selection lines is arbitrary

More straight forward but less efficient is to use $2^n \times 1$ MUX for n-variable function by connecting all inputs to 0 or 1

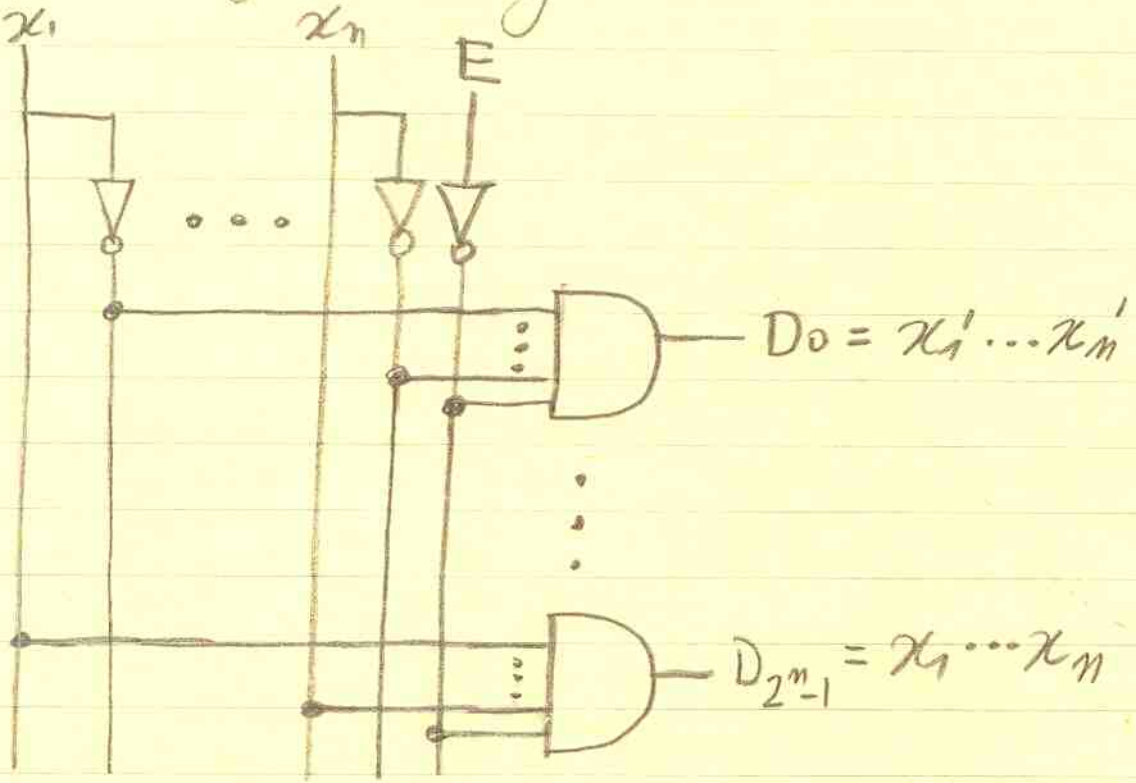


Decoders

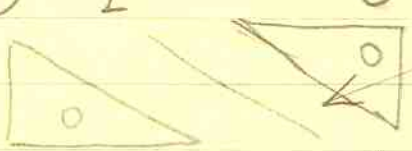
Device with n inputs and $m \leq 2^n$ outputs whose values are mutually exclusive



$n \times m$ decoder comprises m AND gates. If $m = 2^n$ then all AND gates are of n inputs. Otherwise some AND gates may have less than n inputs.

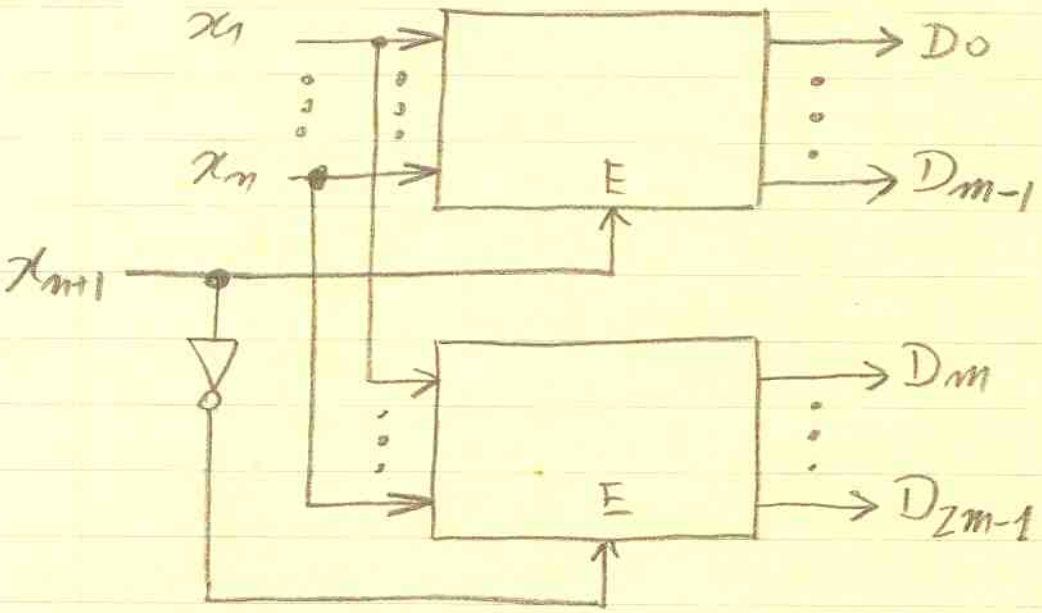


	Inputs				Outputs			
	x_1	x_2	$\cdots$	x_n	D_0	D_1	$\cdots$	D_{2^n-1}
0	0	0	$\cdots$	0	1	0	$\cdots$	0
1	0	1	$\cdots$	0	0	1	$\cdots$	0
			$\vdots$					
2^n-1	1	1	$\cdots$	1	0	0	$\cdots$	1



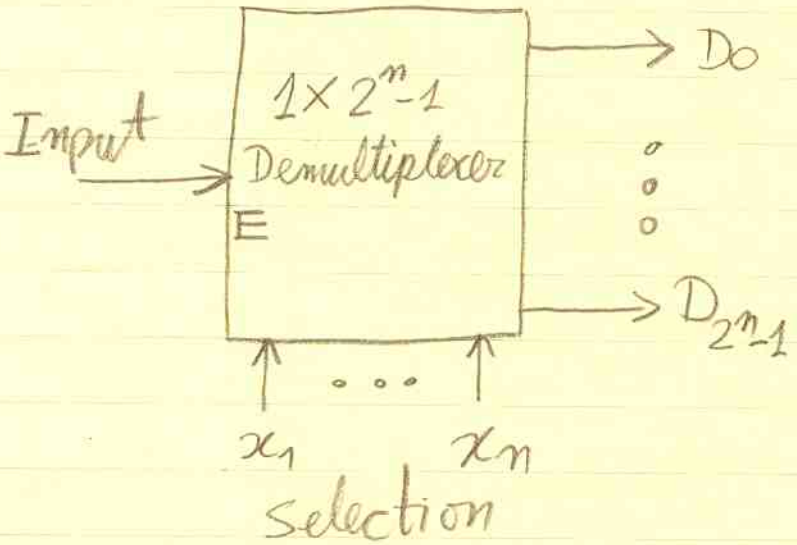
1's on diagonal
0 elsewhere

Two n×m decoders can form $(n+1) \times 2m$ decoding by connecting the additional variable to the enable



Decoders can implement any Boolean function by OR-ing all D_i's corresponding to the minterms of canonical SOP

Decoder can be utilized as demultiplexer, connecting a single input into one of 2^n outputs as follows. ① use the n-inputs for selection ② connect the signal input to E



The inputs of decoder are used as selection lines.

Encoders

$m \leq 2^n$ inputs, n outputs. It generates the binary code of the m input variables.

Inputs must be **mutually exclusive**.

We then **OR** the input lines **which** result (participate) **in the generation of the output code**

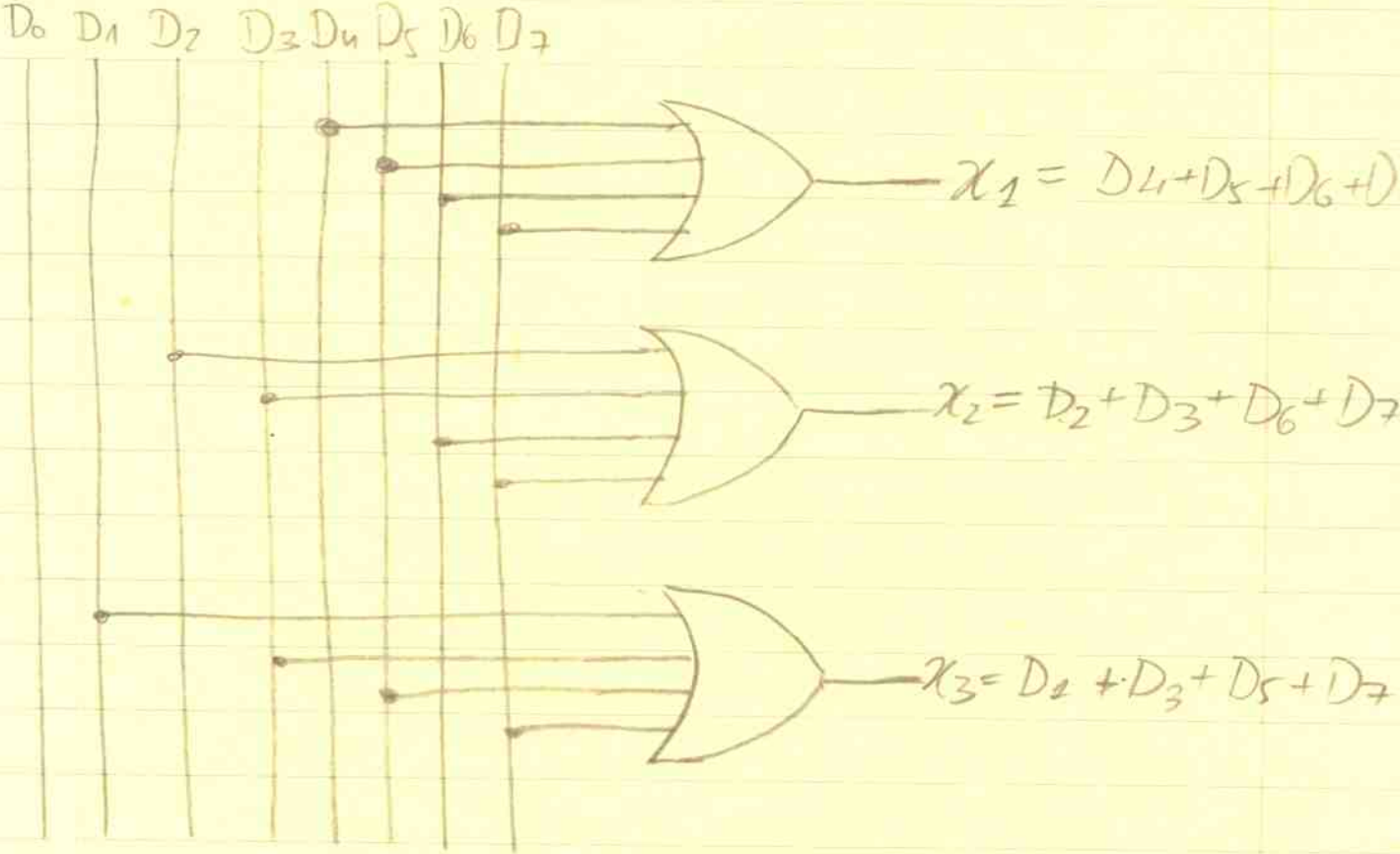
Example: Octal (8-lines) to Binary ($\log_2 8 = 3$ outputs) encoding

Input Data

D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
1							
	1						
		1					
			1				
				1			
					1		
						1	
							1

Output Code

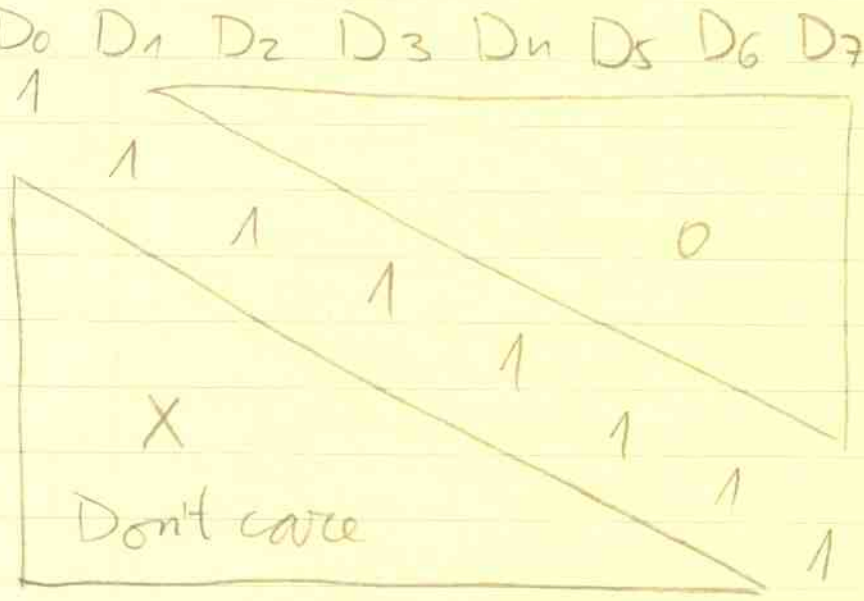
x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1



Priority Encoder

What happens when inputs are not mutually exclusive? This is the reality.

Example: 8 customers are requesting service There maybe simultaneous requests. We use a priority encoder which decides that the highest index call will be served



x	y	z
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

We use the following equality

$$s + s't = \underbrace{s\,s}_{s} + st + \underbrace{s\,s'}_{0} + s't = (s+s')(s+t) = s+t$$

absorption

distribution

$$x = D_4 D_5' D_6' D_7' + D_5 D_6' D_7' + D_6 D_7' + D_7$$

$D_6 + D_7$

$D_5 D_7' + D_6$

$D_5 + D_7$

$D_4 D_6' D_7' + D_5$

$D_4 D_6' + D_7$

$D_4 + D_6$

Is Σ in CNF K-map?

$$x = D_4 + D_5 + D_6 + D_7$$

Homework : prove that

$$y = D_2 D_4' D_5' + D_3 D_4' D_5' + D_6 + D_7$$

$$z = D_1 D_2' D_4' D_6' + D_3 D_4' D_6' + D_5 D_6' + D_7$$