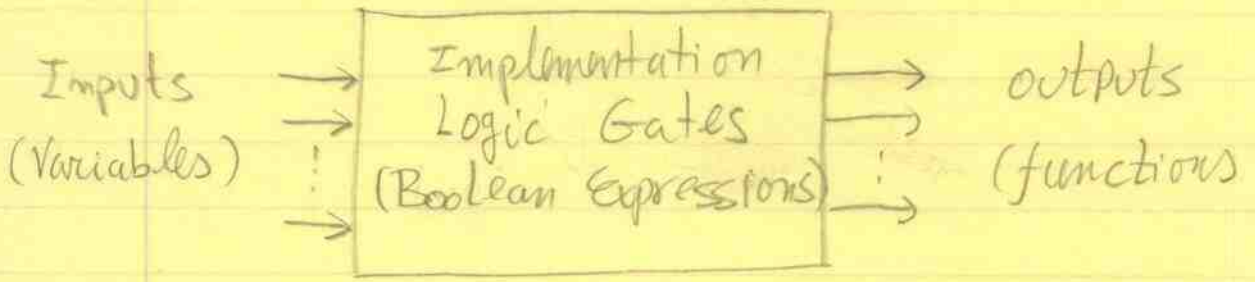
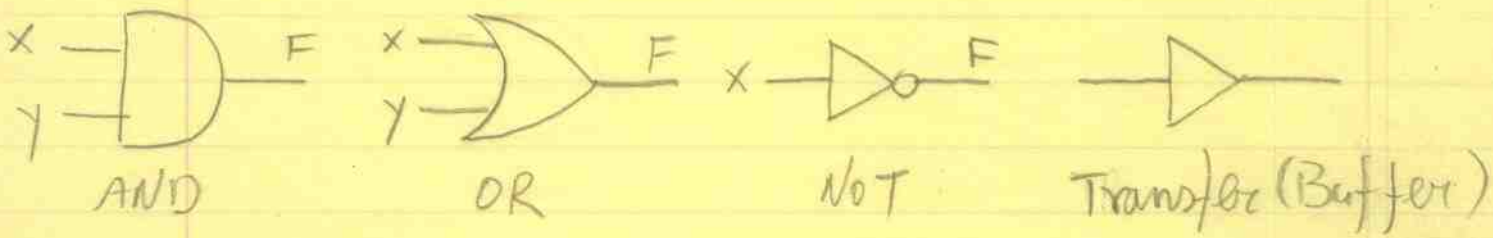
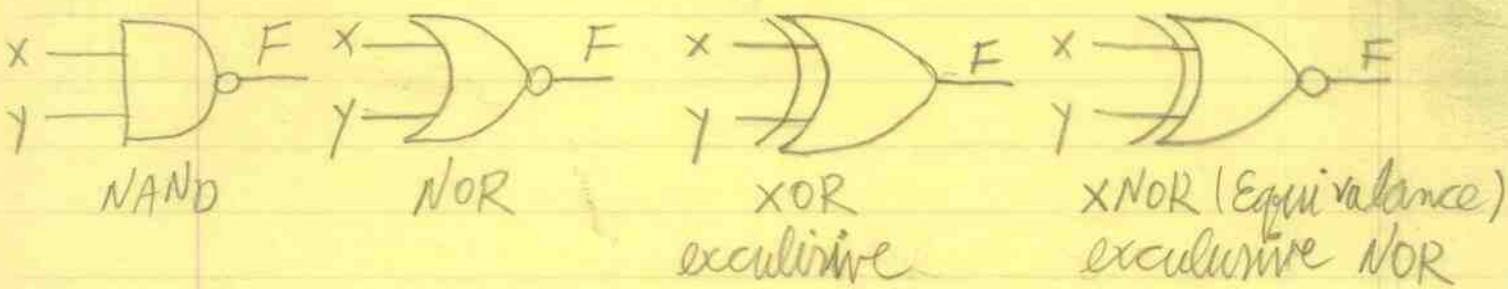
Digital Logic Design

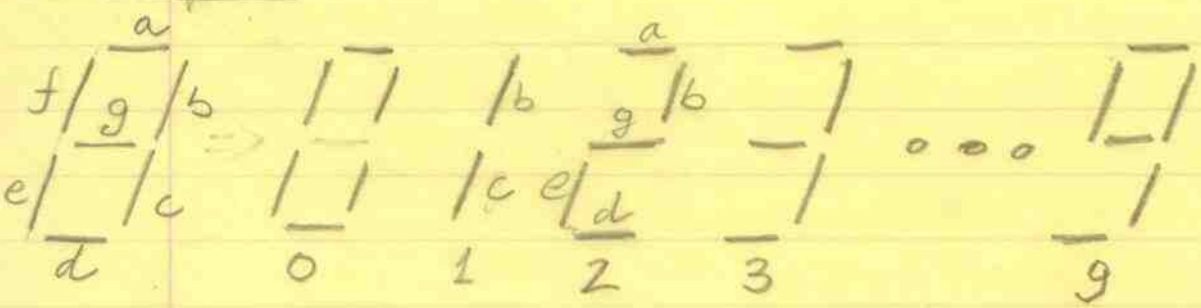


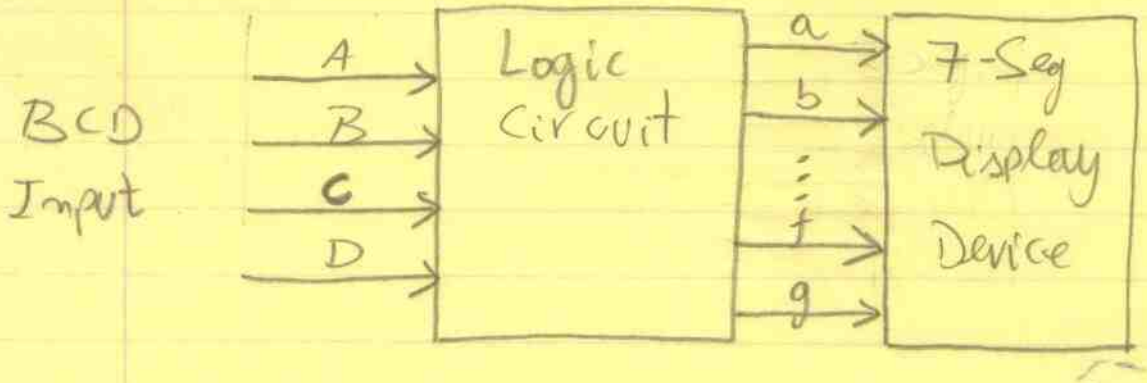
Out of the 16 2-variable functions, 8 are of interest





Example





	Inputs				Outputs						
Decimal	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	0	0	0	0	1
2	0	0	1	0	1	1	0	1	1	0	1
3											
4											
5											
6											
7											
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	1	1	1

We assume that input combinations 10 11 ... 15 never occur. If we complete the table the corresponding outputs a to g are X-don't care

We sum the minterms which turn on every segment

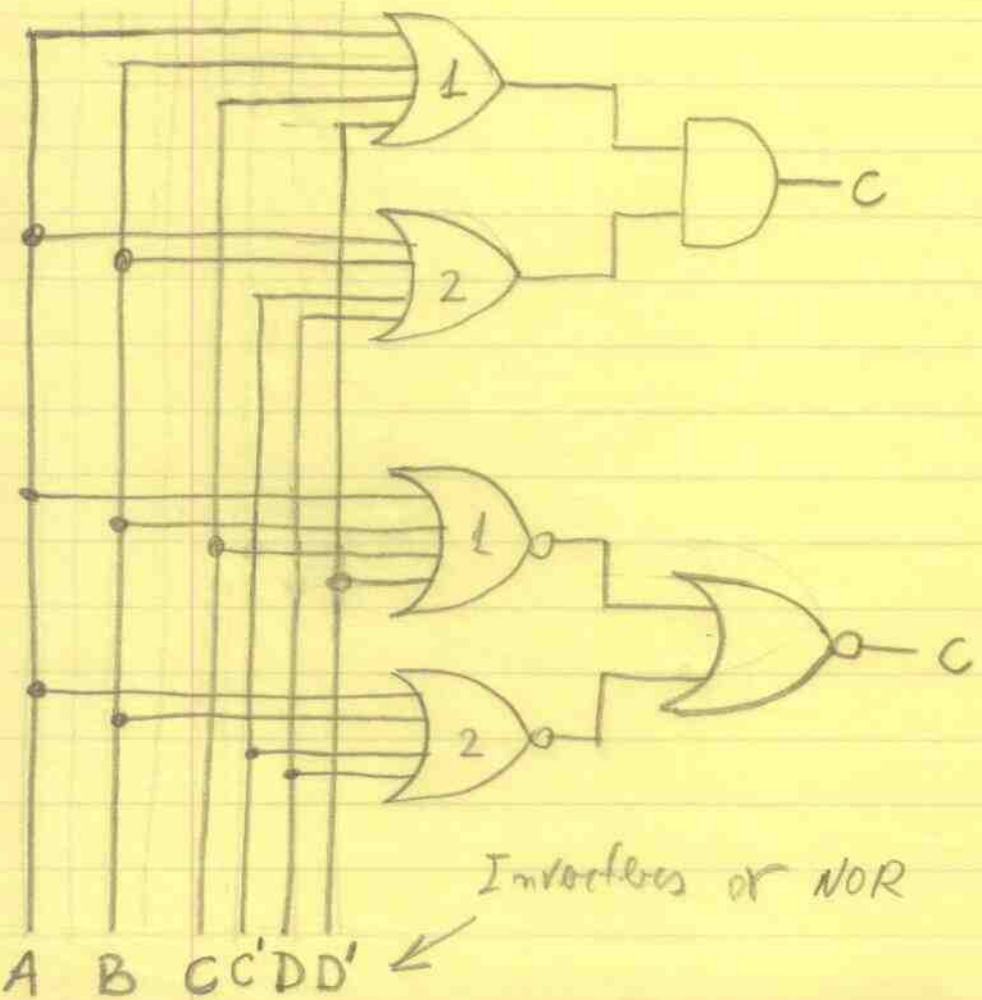
$$a = \Sigma(0,2,3,5,7,8,9) = \Pi(1,4,6)$$

$$c = \Sigma(0,3,4,5,6,7,8,9) = \Pi(1,2)$$

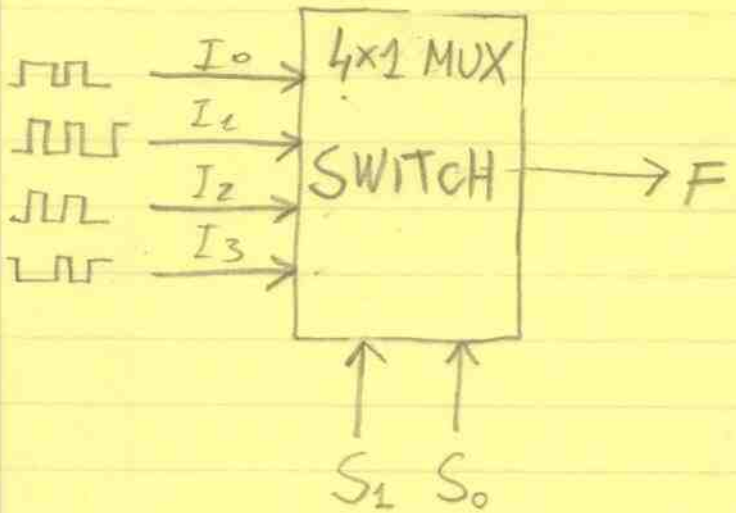
$$\vdots$$

$$g = \Sigma(2,3,4,5,6,8,9) = \Pi(0,1,7)$$

It looks as POS is more efficient than SOP. We can implement any, using OR-AND (NOR-NOR) or AND-OR (NAND-NAND)



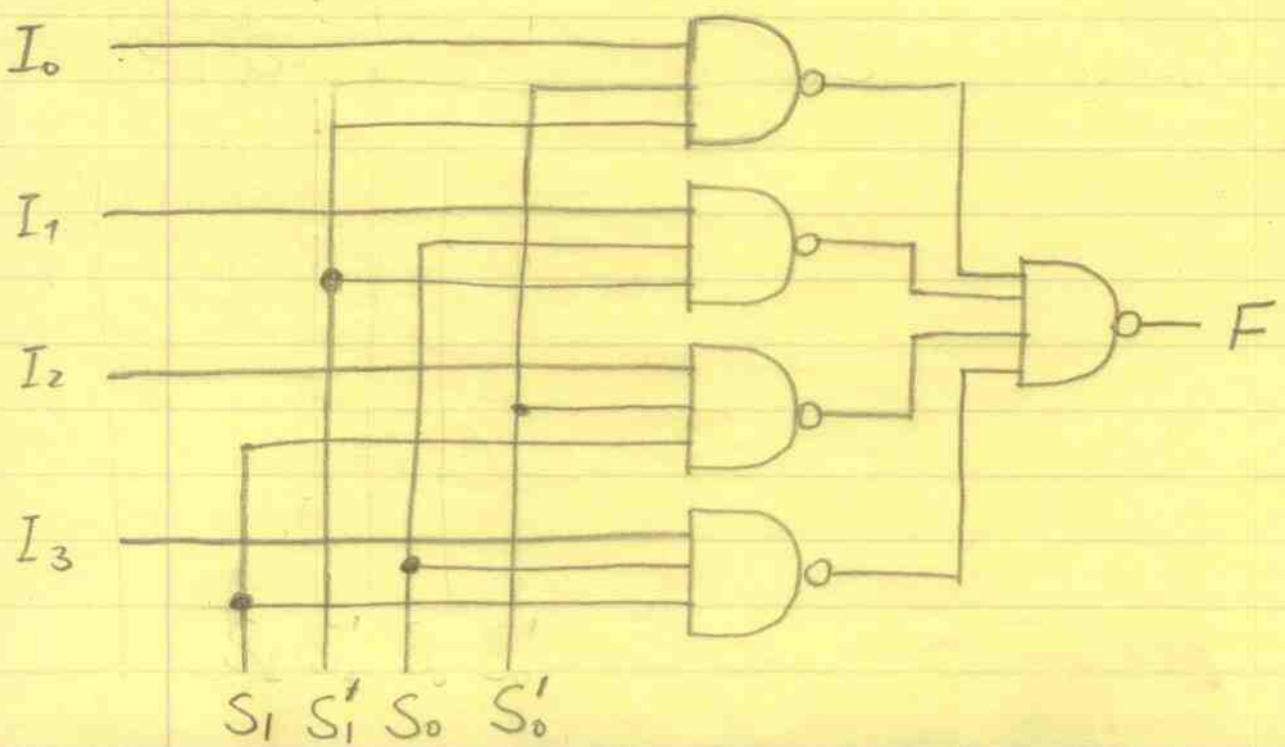
Example 4×1 multiplexer



S_1	S_0	F
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

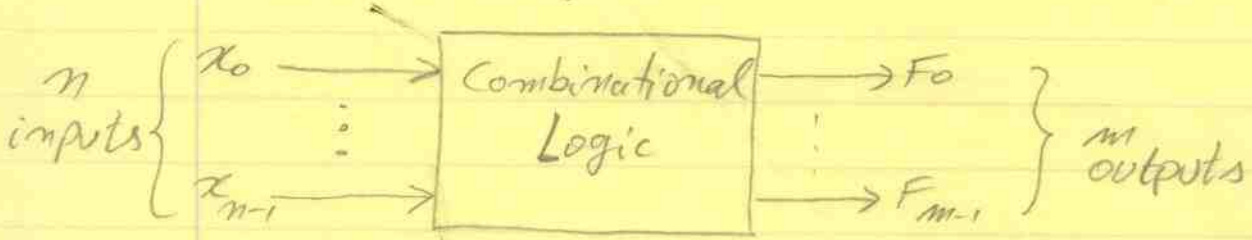
I_0 is selected by 00, I_1 by 01, I_2 by 10 and I_3 by 11.

$$F = (S_1+S_0+I_0)(S_1+S_0'+I_1)(S_1'+S_0+I_2)(S_1'+S_0'+I_3) =$$
$$S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_3$$



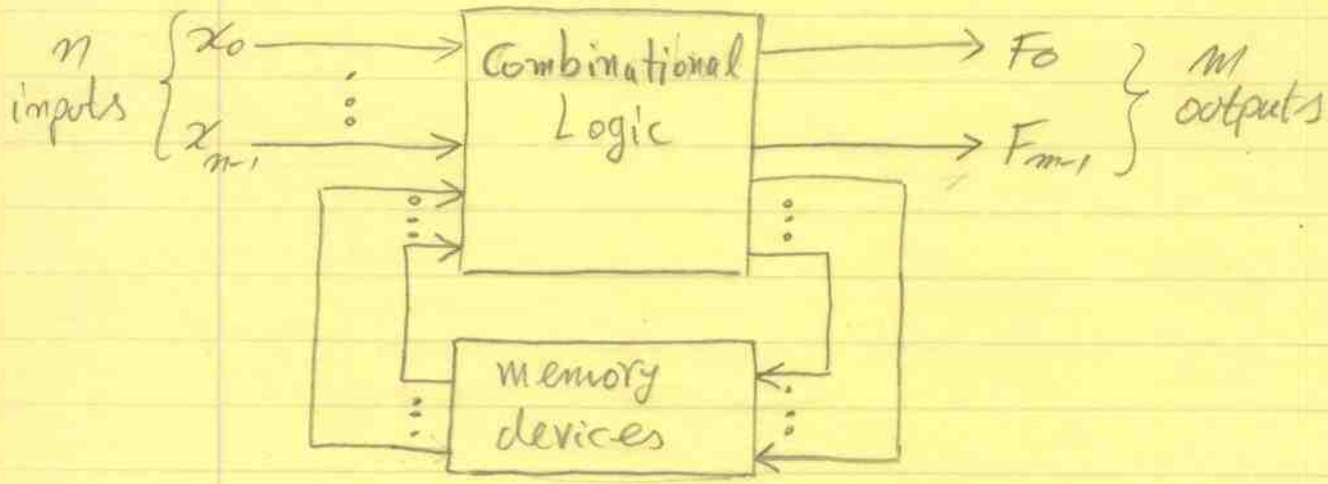
Combinational Logic Circuits

Combinational System



Outputs are fully characterized by the 2^n input combinations

Sequential System



Output depends on both current inputs and previous inputs, hence memory devices are required.

Combinational logic should be minimized - cost of implementation, Silicon area, power, yield

There are however delay implication.

Analysis and design of combinational circuits

- Verbal description
- circuit logic diagram
- Boolean expressions
- truth table

- Functional description - what does it do?
- Structural description - How it does it

Design and Analysis are opposites

Analysis Procedure

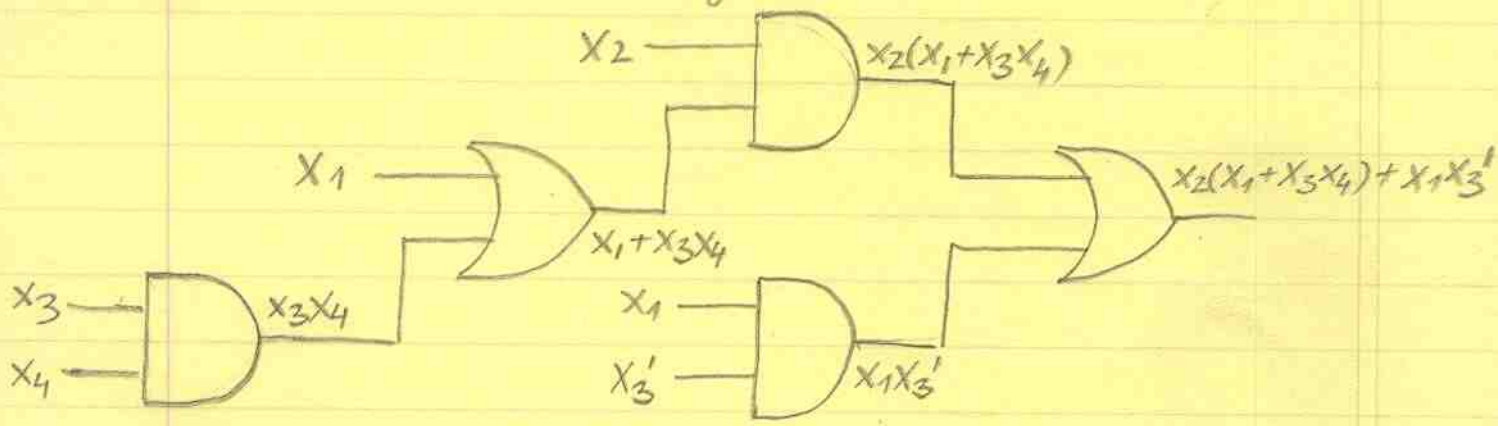
① There must be gates whose inputs are all defined

② Write output of gates in ①

③ Stop if all outputs of diagram are defined

④ go to 1

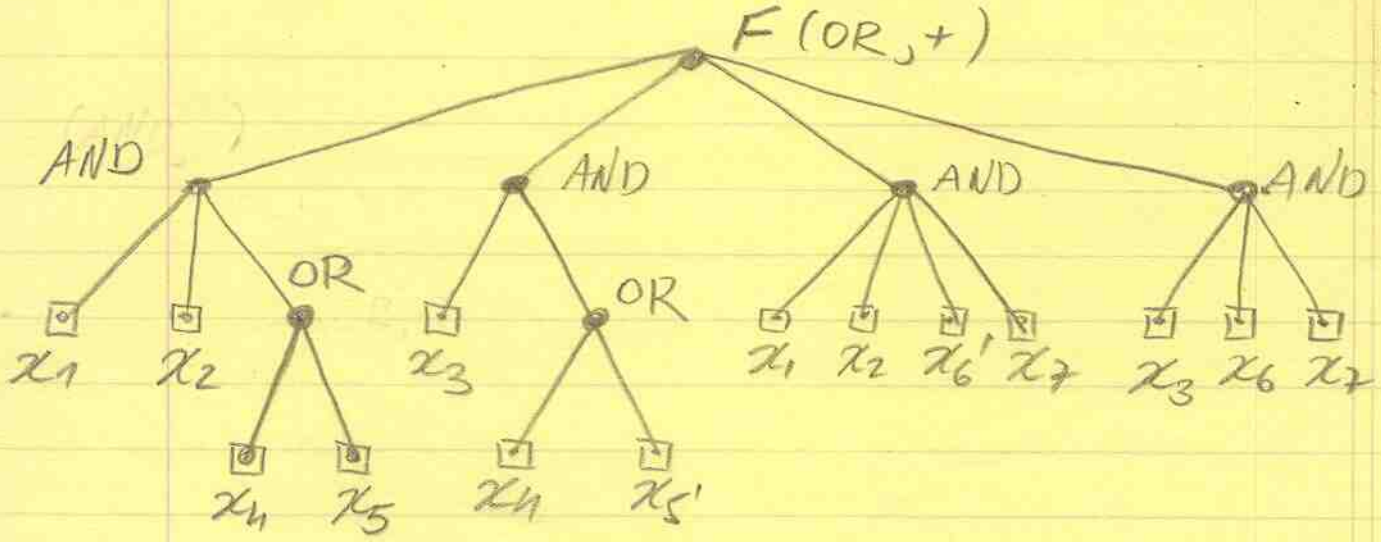
Example: circuit diagram → expression



Example: expression → circuit diagram

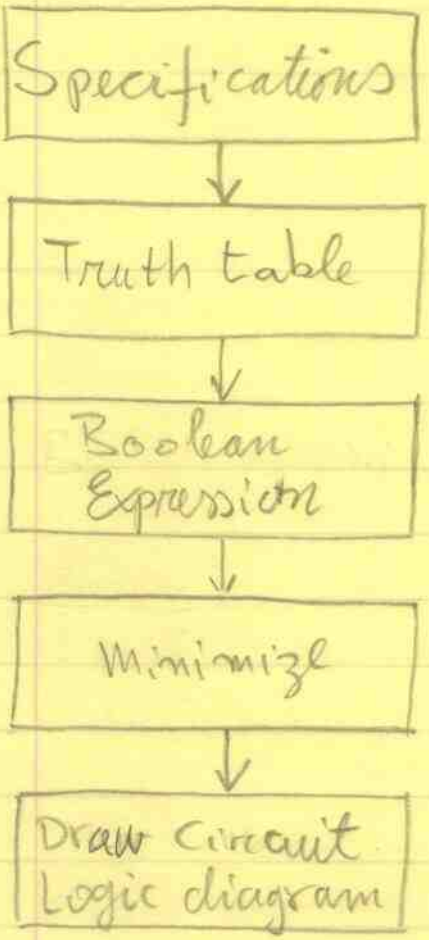
$$F = x_1x_2(x_4+x_5)+x_3(x_4+x_5')+x_1x_2x_6'x_7+x_3x_6x_7$$

Build expression tree, rooted at F



Each node is translated into AND or OR gate, depending on level in tree. Leaves are inputs

Design Procedure



State the problem

Assign letter symbols to inputs and output functions

Derive for each function

Simplify as much as possible

Example: Binary full-adder

```
  1 1 1 1 0
+ 1 0 1 1 0
  1 1 0 1 1
-----------
1 1 0 0 0 1
```

Basic operation: adding 3 bits

addend + augend + carry in

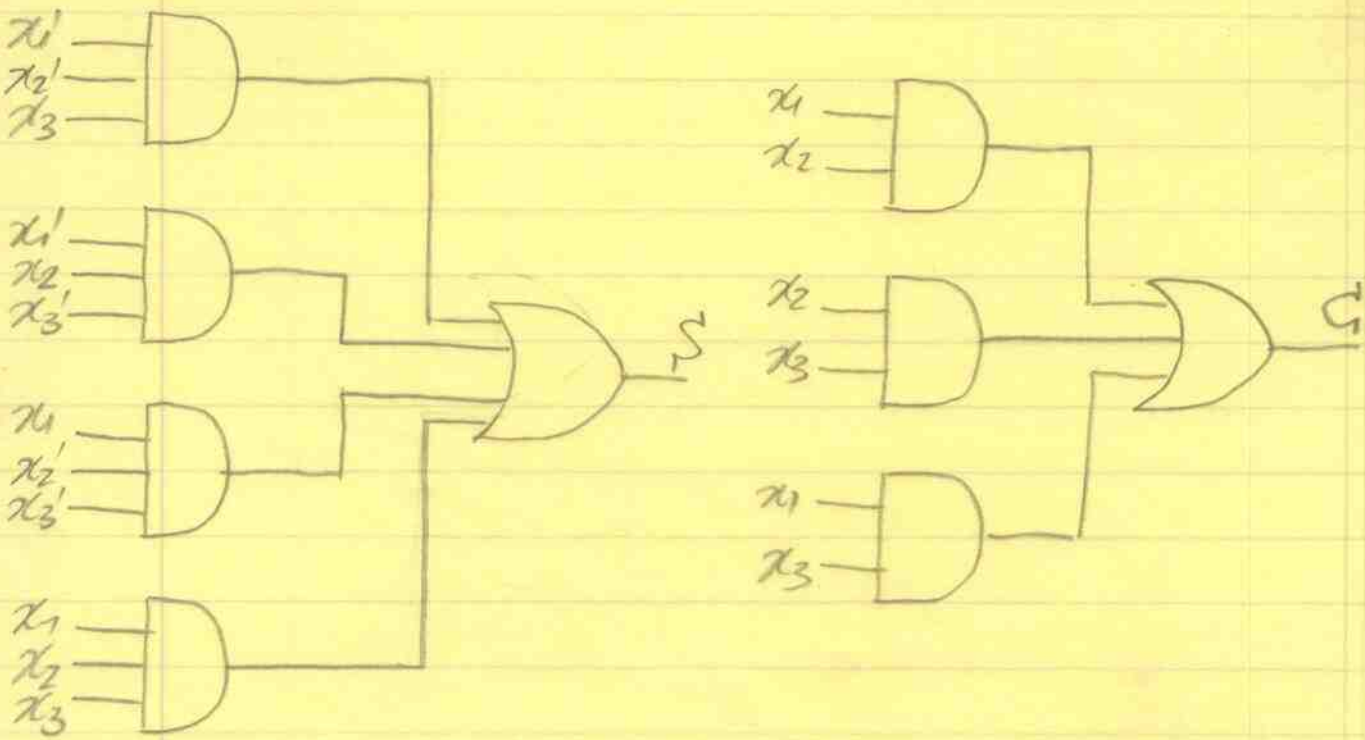
result: Sum bit, carry out bit

addend x_1	augmend x_2	carry in x_3	Sum S	carry out C
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

$$S = \sum(1,2,4,7) = x_1'x_2'x_3 + x_1'x_2x_3' + x_1x_2'x_3' + x_1x_2x_3$$

$$C = \sum(3,5,6,7) = x_1'x_2x_3 + x_1x_2'x_3 + x_1x_2x_3' + x_1x_2x_3 =$$
$$= x_1x_2 + x_2x_3 + x_1x_3$$

(x_1x_3, x_2x_3, x_1x_2)



Sum can be implemented as $x_1 \oplus x_2 \oplus x_3$

$$S = x_1' x_2' x_3 + x_1' x_2 x_3' + x_1 x_2' x_3' + x_1 x_2 x_3 =$$

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

but $(x_1' x_2' + x_1 x_2)' = (x_1 + x_2)(x_1' + x_2') = x_1 x_2' + x_1' x_2 =$
$= x_1 \oplus x_2$

hence

$$S = (x_1 \oplus x_2)x_3' + (x_1 \oplus x_2)' x_3 = x_1 \oplus x_2 \oplus x_3$$

Example: Monitoring device

Electric motor supplied by 3 generators

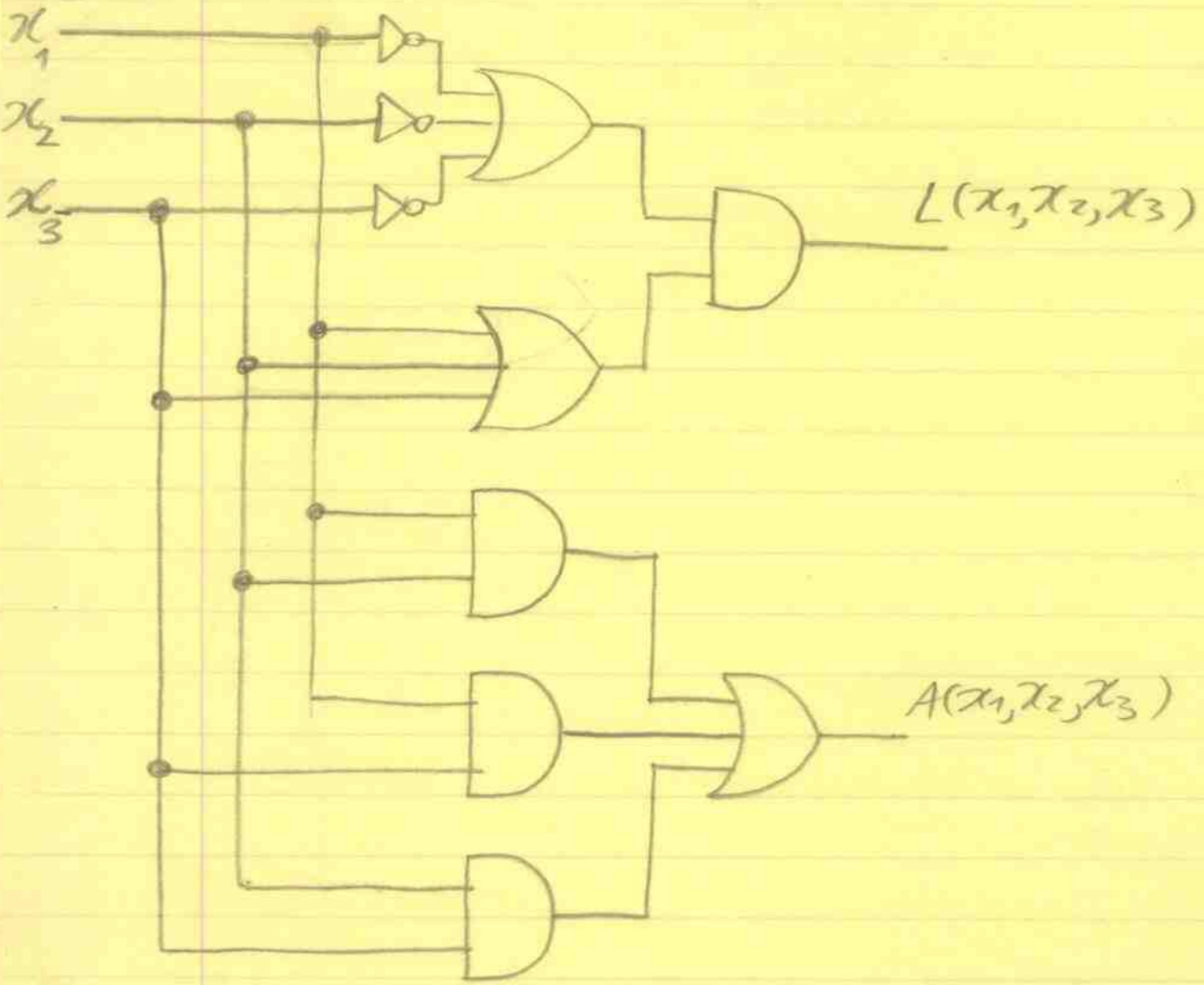
Warning Lamp: iff 1 or 2 generators fail $L=1$

acoustic alarm: iff 2 or 3 generators fail $A=1$

$x_i = 1$: motor i fails, $x_i = 0$ otherwise

$$L'(x_1, x_2, x_3) = x_1' x_2' x_3' + x_1 x_2 x_3$$
$$L(x_1, x_2, x_3) = (x_1 + x_2 + x_3)(x_1' + x_2' + x_3')$$

$$A(x_1, x_2, x_3) = x_1 x_2 x_3' + x_1 x_2' x_3 + x_1' x_2 x_3 + x_1 x_2 x_3 =$$
$$= x_1 x_2 + x_2 x_3 + x_1 x_3$$



Logic Minimization

Combinational design problem → Boolean function

Complexity: # literals → # transistors
terms → # gates + # transistors

Minimize Boolean functions, redundancy removal

redundancy and reliability

Example: $\Sigma(0,2,4,5,7) = x_1'x_3' + x_1x_2' + x_1x_3$
① ②

① 5-input OR gate + 5 3-input AND gate

② 3-input OR gate + 3 2-input AND gate

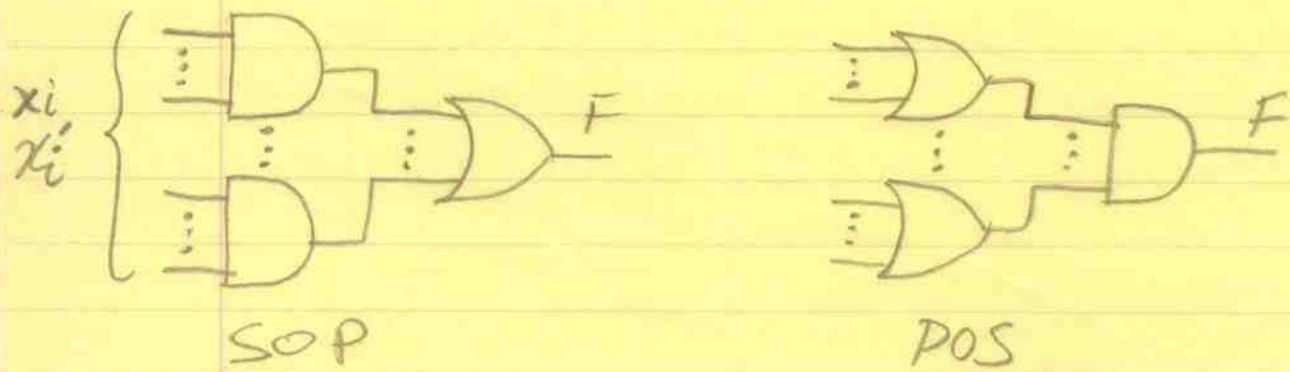
minimality 1st # terms 2nd # literals in SOP (or POS)

Two-level circuit (SOP or POS)	multi-level (mixed)
- Less propagation delay	more delay
- High fan in	less fan in
- Many literals	less literals

$x_1'x_3'x_4' + x_1'x_2x_4 + x_2x_3x_4 = x_2x_4(x_1' + x_3) + x_1'x_3'x_4'$

2-level	3-level
9 literals	7 literals
4 gates	4 gates

levels: maximum # gates from any input to any output



Minimization by Boolean Algebra Theorems

Example: $F(x_1, x_2, x_3) = \sum(0, 2, 3, 4, 5, 7)$

$= \underline{x_1' x_2' x_3'} + x_1' x_2 x_3' + x_1' x_2 x_3 + x_1 x_2' x_3' + x_1 x_2' x_3 +$
$+ \underline{x_1 x_2 x_3} =$

duplicate m_0 and m_7 and use logical adjacency

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

which cannot further be simplified. But different pairing m_0-m_2, m_4-m_5, m_3-m_7

$F = x_1' x_3' + x_1 x_2' + x_2 x_3$

Which is simpler

Example: repeated application of logical adjacency

$F(x_1, x_2, x_3, x_4) = \sum(12, 13, 14, 15) =$

$= \underline{x_1 x_2} x_3' x_4' + \underline{x_1 x_2} x_3' x_4 + \underline{x_1 x_2} x_3 x_4' + \underline{x_1 x_2} x_3 x_4 =$

$= \underline{x_1 x_2} x_3' (x_4' + x_4) + \underline{x_1 x_2} x_3 (x_4' + x_4) = \underline{x_1 x_2} (x_3' + x_3)$

$= x_1 x_2$

Theorem: Let $F(x_0, \ldots, x_{n-1})$ be expressed by a sum of 2^m minterms, $m \leq n$. Let $n-m$ variables be identical in all minterms. Then F can be simplified into single product of the $n-m$ variables

Proof: By induction on m - the number of non identical variables. Assume that these are $x_0, \ldots, x_{m-1}$.

There are 2^m distinct products of $x_0, \ldots, x_{m-1}$, hence they span all minterms

2^{m-1} of which have x_0 literal and 2^{m-1} have x_0'. Therefore

$$F = x_0 \Sigma_1(\quad) + x_0' \Sigma_2(\quad).$$

The induction hypothesis holds for Σ_1 and Σ_2, which can be simplified into product term of $n-m$ variables. But these products are identical.
$(n-1)-(m-1) = n-m$

Now we apply the rule of logical adjacency and the theorem follows.

We need more systematic way for simplifying Boolean Functions