3.1 Combinational circuit synthesis tools

Jean-Pierre Deschamps

University Rovira i Virgili, Tarragona, Spain

UAB
Universitat Autònoma de Barcelona

Design of combinational circuits: practical questions

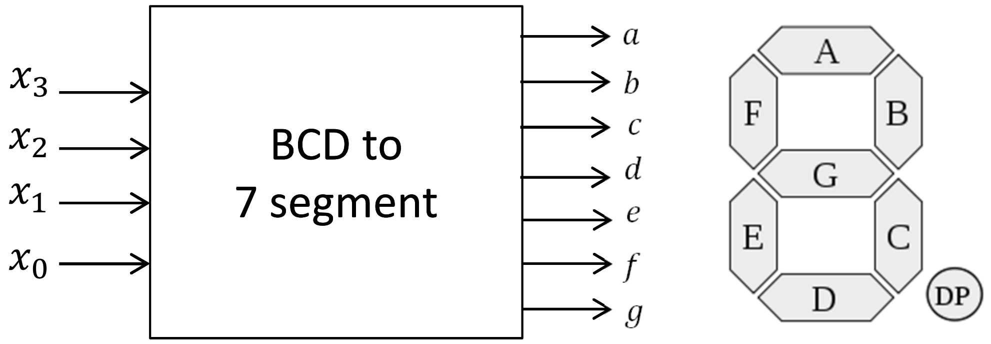
UAB
Universitat Autònoma de Barcelona

- Redundant terms.
- Boolean function optimization tools.

1. Redundant terms (*don’t care*)

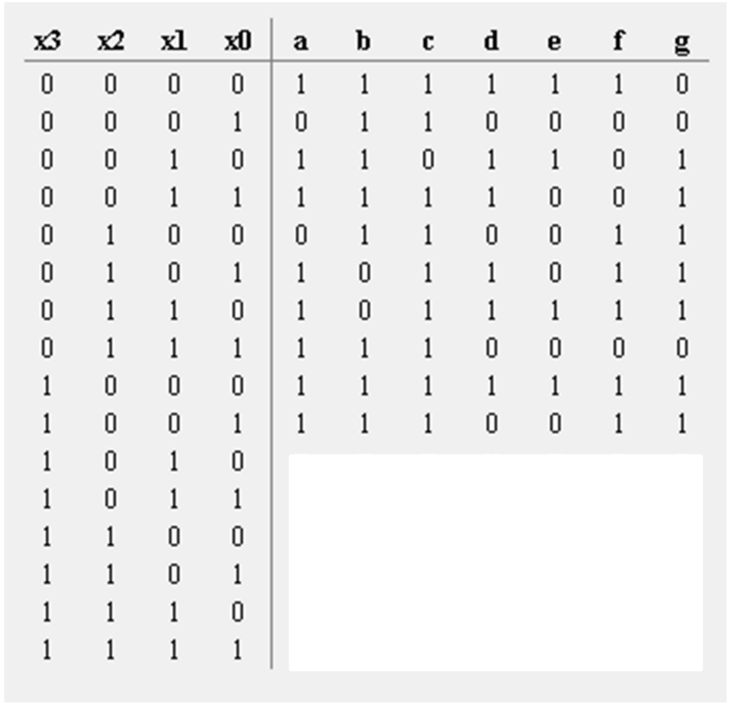
Some combinations of input signal values could **never occur**, or, when they occur, the output signal values do not matter (**don’t care)**. The corresponding minterms can be used, or not, in order to optimize the final circuit.

EXAMPLE: BCD to 7-segment decoder.

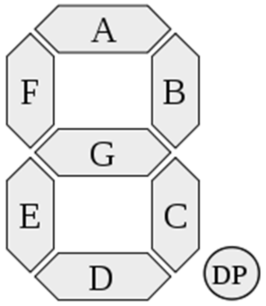


1. Redundant terms (*don't care*)

EXAMPLE: BCD to 7-segment decoder.

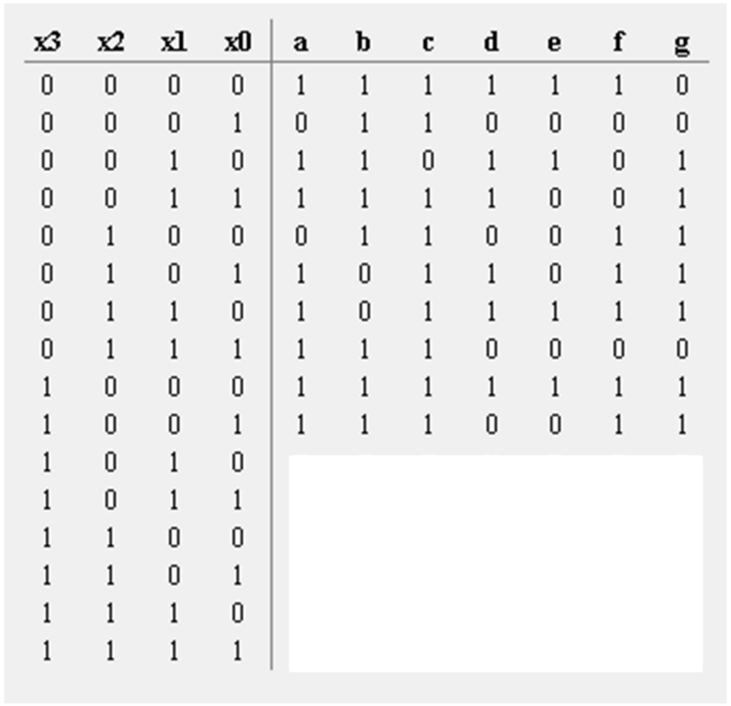


x3	x2	x1	x0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0							
1	0	1	1							
1	1	0	0							
1	1	0	1							
1	1	1	0							
1	1	1	1							



1. Redundant terms (*don’t care*)

All redundant terms = 0:



x3	x2	x1	x0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0							
1	0	1	1							
1	1	0	0							
1	1	0	1							
1	1	1	0							
1	1	1	1							

$$a = \bar{x}_3.x_1 + \bar{x}_3.x_2.x_0 + x_3.\bar{x}_2.\bar{x}_1$$

$$b = \bar{x}_3.\bar{x}_2 + \bar{x}_2.\bar{x}_1 + \bar{x}_3.\bar{x}_1.\bar{x}_0 + \bar{x}_3.x_1.x_0$$

$$c = \bar{x}_2.\bar{x}_1 + \bar{x}_3.x_0 + \bar{x}_3.x_2$$

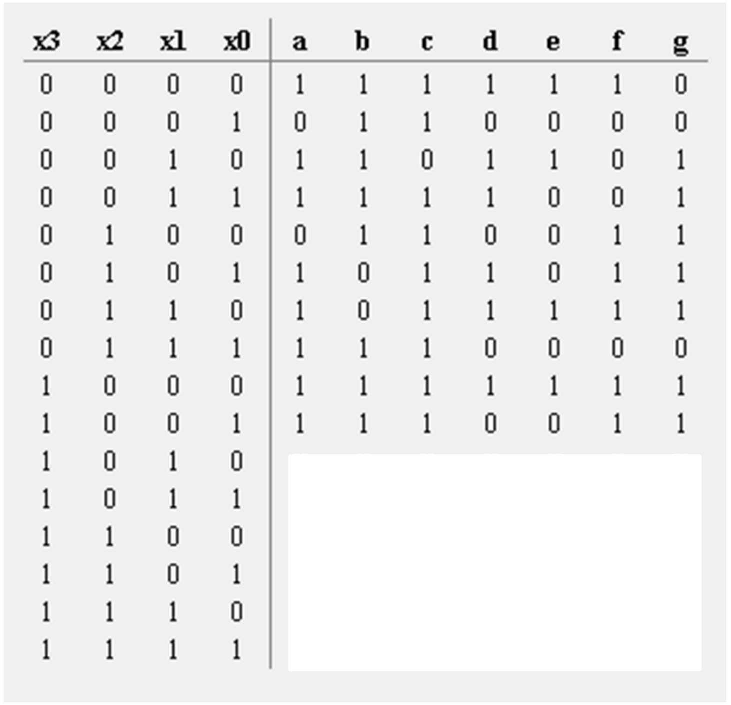
$$d = \bar{x}_2.\bar{x}_1.\bar{x}_0 + \bar{x}_3.\bar{x}_2.x_1 + \bar{x}_3.x_1.\bar{x}_0 + \\ + \bar{x}_3.x_2.\bar{x}_1.x_0$$

$$e = \bar{x}_2.\bar{x}_1.\bar{x}_0 + \bar{x}_3.x_1.\bar{x}_0$$

$$f = \bar{x}_3.\bar{x}_1.\bar{x}_0 + \bar{x}_3.x_2.\bar{x}_1 + \bar{x}_3.x_2.\bar{x}_0 + x_3.\bar{x}_2.\bar{x}_1$$

$$g = \bar{x}_3.\bar{x}_2.x_1 + \bar{x}_3.x_2.\bar{x}_1 + \bar{x}_3.x_2.\bar{x}_0 + x_3.\bar{x}_2.\bar{x}_1$$

1. Redundant terms (*don't care*)



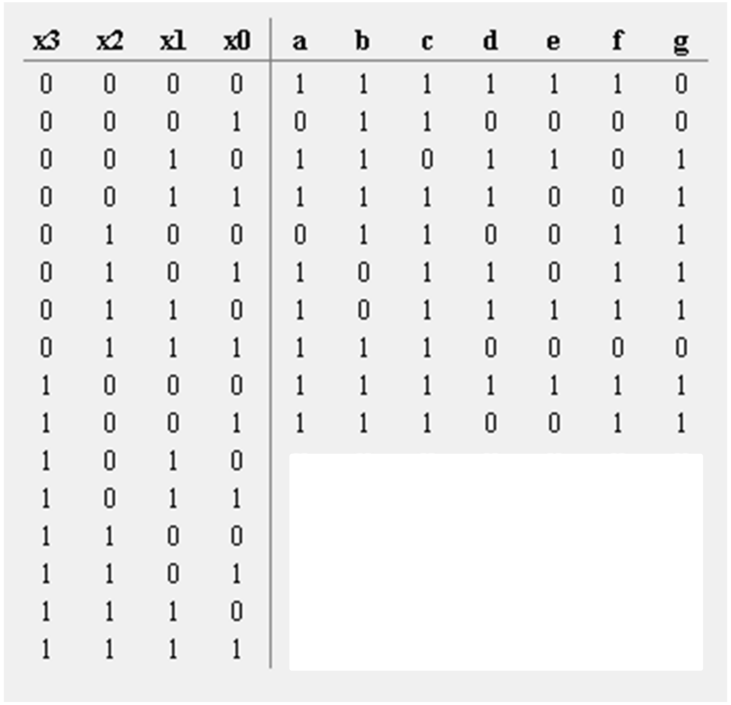
x3	x2	x1	x0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0							
1	0	1	1							
1	1	0	0							
1	1	0	1							
1	1	1	0							
1	1	1	1							

$$b = \bar{x}_3.\bar{x}_2 + \bar{x}_2.\bar{x}_1 + \bar{x}_3.\bar{x}_1.\bar{x}_0 + \bar{x}_3.x_1.x_0$$

x_3 x_2 x_1 x_0	b	c
0 0 0 0	1	1
0 0 0 1	1	1
0 0 1 0	1	0
0 0 1 1	1	1
0 1 0 0	1	1
0 1 0 1	0	1
0 1 1 0	0	1
0 1 1 1	1	1
1 0 0 0	1	1
1 0 0 1	1	1
1 0 1 0	1	0
1 0 1 1	1	1
1 1 0 0	1	1
1 1 0 1	0	1
1 1 1 0	0	1
1 1 1 1	1	1

$$b = \bar{x}_2 + \bar{x}_1.\bar{x}_0 + x_1.x_0$$

1. Redundant terms (*don't care*)



x3	x2	x1	x0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0							
1	0	1	1							
1	1	0	0							
1	1	0	1							
1	1	1	0							
1	1	1	1							

$$c = \bar{x}_2 . \bar{x}_1 + \bar{x}_3 . x_0 + \bar{x}_3 . x_2$$

x_3 x_2 x_1 x_0	b	c
0 0 0 0	1	1
0 0 0 1	1	1
0 0 1 0	1	0
0 0 1 1	1	1
0 1 0 0	1	1
0 1 0 1	0	1
0 1 1 0	0	1
0 1 1 1	1	1
1 0 0 0	1	1
1 0 0 1	1	1
1 0 1 0	1	0
1 0 1 1	1	1
1 1 0 0	1	1
1 1 0 1	0	1
1 1 1 0	0	1
1 1 1 1	1	1

$$c = \bar{x}_1 + x_0 + x_2$$

1. Redundant terms (*don't care*)

$a = \bar{x}_3.x_1 + \bar{x}_3.x_2.x_0 + x_3.\bar{x}_2.\bar{x}_1$

$b = \bar{x}_3.\bar{x}_2 + \bar{x}_2.\bar{x}_1 + \bar{x}_3.\bar{x}_1.\bar{x}_0 + \bar{x}_3.x_1.x_0$

$c = \bar{x}_2.\bar{x}_1 + \bar{x}_3.x_0 + \bar{x}_3.x_2$

$d = \bar{x}_2.\bar{x}_1.\bar{x}_0 + \bar{x}_3.\bar{x}_2.x_1 + \bar{x}_3.x_1.\bar{x}_0 + {}$
$\quad + \bar{x}_3.x_2.\bar{x}_1.x_0$

$e = \bar{x}_2.\bar{x}_1.\bar{x}_0 + \bar{x}_3.x_1.\bar{x}_0$

$f = \bar{x}_3.\bar{x}_1.\bar{x}_0 + \bar{x}_3.x_2.\bar{x}_1 + \bar{x}_3.x_2.\bar{x}_0 + x_3.\bar{x}_2.\bar{x}_1$

$g = \bar{x}_3.\bar{x}_2.x_1 + \bar{x}_3.x_2.\bar{x}_1 + \bar{x}_3.x_2.\bar{x}_0 + x_3.\bar{x}_2.\bar{x}_1$

$a = x_1 + x_2.x_0 + x_3$

$b = \bar{x}_2 + \bar{x}_1.\bar{x}_0 + x_1.x_0$

$c = \bar{x}_1 + x_0 + x_2$

$d = \bar{x}_2.\bar{x}_0 + \bar{x}_2.x_1 + x_1.\bar{x}_0 + x_2.\bar{x}_1.x_0$

$e = \bar{x}_2.\bar{x}_0 + x_1.\bar{x}_0$

$f = \bar{x}_1.\bar{x}_0 + x_2.\bar{x}_1 + x_2.\bar{x}_0 + x_3$

$g = \bar{x}_2.\bar{x}_0 + x_2.\bar{x}_1 + x_1.\bar{x}_0 + x_3$

35	***total***	***26***
24	AND	15
7	OR	7
4	INV	26

2. Synthesis tools

Software tools that optimize switching function representations from

- a logic expression,
- a table,

and generate the corresponding circuit.

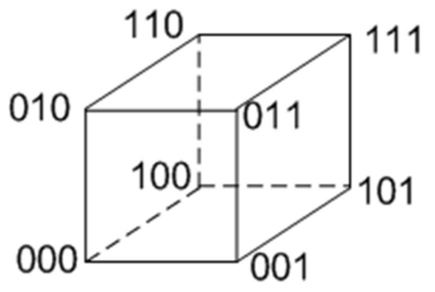
Two basic concepts:

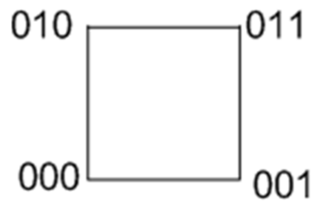
- CUBE REPRESENTATION,
- ADJACENCY.

2. Synthesis tools

1) Cubes and cube representation.

- The set of *n*-component binary vectors $B_2{}^n = \{(x_{n-1}, x_{n-2}, \cdots, x_1, x_0) \in \{0, 1\}^n\}$ can be considered as a **cube of dimension *n*.**
- A subset of $B_2{}^n$ defined by giving a constant value (0 or 1) to *m* coordinates of $(x_{n-1}, x_{n-2}, \cdots, x_1, x_0)$ is a **cube of dimension *n-m*** .





2. Synthesis tools

Assume that n = 4 and consider a function $f(x_3, x_2, x_1, x_0) = x_2 \cdot \overline{x_0}$:

$$f = 1 \text{ iff } (x_3, x_2, x_1, x_0) \in \{(x_3, x_2, x_1, x_0) \in \{0, 1\}^4, x_2 = 1, x_0 = 0\},$$

that is a cube of dimension n-2.

Thus, a **cube** is a set of points of $B_2{}^n$ where a product of literals = 1.

2. Synthesis tools

Cube representation:

Define a variable order, for example *n*-1, *n*-2, ⋯ , 1, 0; then a cube is represented by a sequence of *n* characters belonging to {0, 1, x}.

Examples ($n = 4$):

1x01 represents $\{(x_3, x_2, x_1, x_0) \in \{0, 1\}^4, x_3 = 1, x_1 = 0, x_0 = 1\}$; corresponding product: $x_3 \cdot \overline{x_1} \cdot x_0$.

x1x0 represents $\{(x_3, x_2, x_1, x_0) \in \{0, 1\}^4, x_2 = 1, x_0 = 0\}$; corresponding product: $x_2 \cdot \overline{x_0}$.

1: corresponding variable present under normal form,
0: corresponding variable present under inverted form,
x: corresponding variable missing.

(quiz)

Consider the set of 5-variable switching functions *f*(*e*, *d*, *c*, *b*, *a*). Variable order: *e, d, c, b, a.* What cube is represented by 1x01x ?

1. $a.\bar{c}.d$
2. $\bar{a}.c.\bar{d}$
3. $b.e$
4. $e.\bar{c}.b$
5. $\bar{e}.c.\bar{b}$

2. Synthesis tools

2) Adjacency

Consider two cubes (n = 4) **1x11** and **1x01**. Assume that the corresponding products

$$x_3 \cdot x_1 \cdot x_0 \text{ and } x_3 \cdot \overline{x_1} \cdot x_0$$

belong to the representation of some function f:

$$f = x_3 \cdot x_1 \cdot x_0 + x_3 \cdot \overline{x_1} \cdot x_0 + \cdots$$

As they differ in only one variable (x_1 and $\overline{x_1}$) the representation of f can be simplified:

$$f = x_3 \cdot x_0 + \cdots$$

In terms of cubes: 0x11 $\cup$ 0x01 = 0xx1.

0x11 and 0x01 are **adjacent cubes**.

2. Synthesis tools

EXAMPLE: $f(a,b,c,d)$ defined by a list of cubes

0010, 0011, 0101, 0110, 0111, 1000

(actually a list of minterms of f)

$$0010 \cup 0011 = 001x,$$

$$0110 \cup 0111 = 011x,$$

$$0101 \cup 0111 = 01x1,$$

$$001x \cup 011x = 0x1x.$$

Thus

$$0010 \cup 0011 \cup 0101 \cup 0110 \cup 0111 \cup 1000 = 0x1x \cup 01x1 \cup 1000,$$

$$f = \overline{a} \cdot c + \overline{a} \cdot b \cdot d + a \cdot \overline{b} \cdot \overline{c} \cdot \overline{d}$$

(quiz)

The following set of cubes

{0x111, 0x011, 11100,11101,11110,11111}

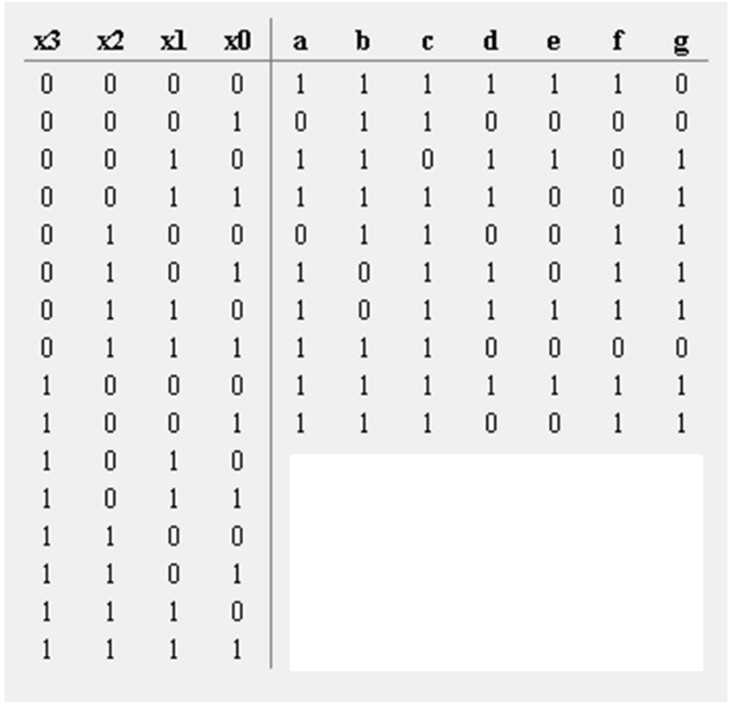
is equivalent to ···

1. 1x111, 111xx, 01011
2. 111xx
3. 00x11, 111xx
4. 0xx11, 111xx

2. Synthesis tools

LogiSim, *Combinational Analysis*:

2. Synthesis tools



x3	x2	x1	x0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0							
1	0	1	1							
1	1	0	0							
1	1	0	1							
1	1	1	0							
1	1	1	1							

$$b = \bar{x}_2 + \bar{x}_1 . \bar{x}_0 + x_1 . x_0$$

$$c = \bar{x}_1 + x_0 + x_2$$

$f = \overline{a} + \overline{b} \cdot \overline{c} + a \cdot b \cdot c + \overline{a} \cdot d \cdot e$

f = !a + !b & !c + a & b & c + !a & d & e

SUMMARY

- Redundant terms and corresponding optimization problem.
- Relation between cubes and products of literals.
- Tools that minimize Boolean expressions and that optimize circuits.

3.2 Propagation time

Jean-Pierre Deschamps

University Rovira i Virgili, Tarragona, Spain

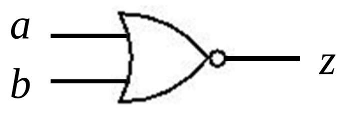
1. Propagation time

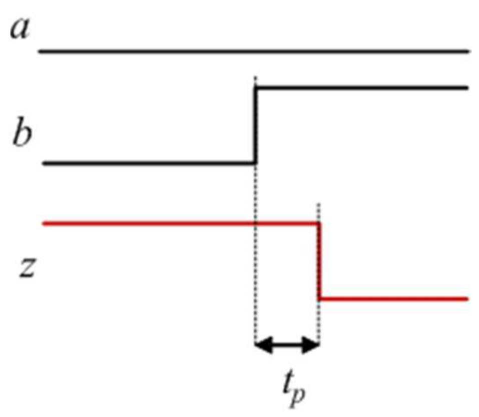
Logic gates are electronic components characterized by their propagation time, from their inputs to their outputs.

1. Propagation time

Logic gates are electronic components characterized by their propagation time, from their inputs to their outputs.

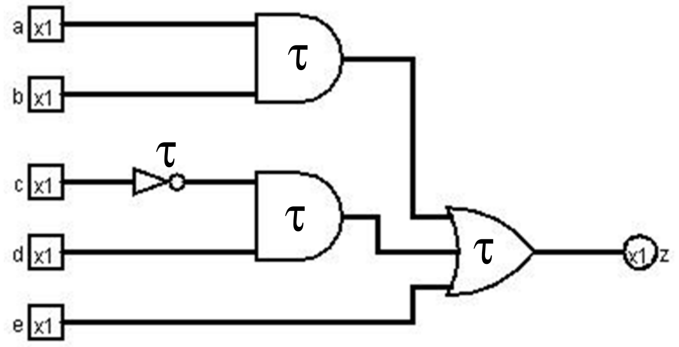
a) Logic gate:

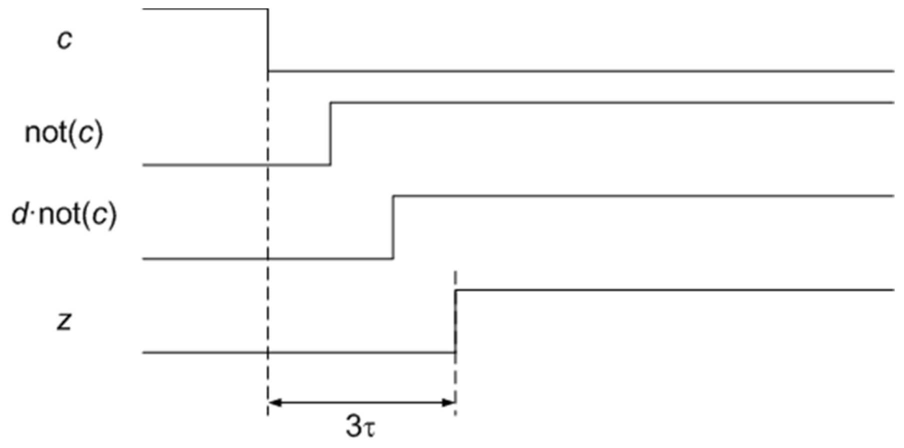




1. Propagation time

b) Combinational circuit ("worst case")





$a = 0,\ b = \text{x},\ c = 1 \rightarrow 0,\ d = 1,\ e = 0$

1. Propagation time

To different implementations of the same function correspond different delays. Example:

$f=[(a+b).(c+d).e]+[(k+g).(h+i).j]$

$f=e.a.c+e.a.d+e.b.c+e.b.d+j.k.h+j.k.i+j.g.h+j.g.i$

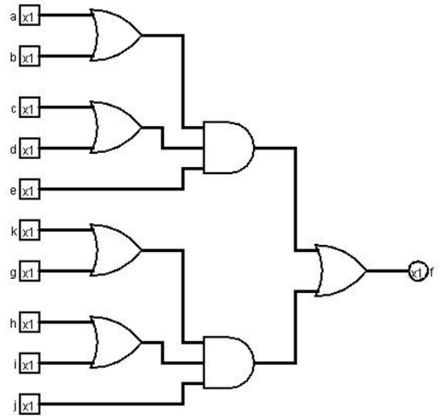
UAB Universitat Autònoma de Barcelona

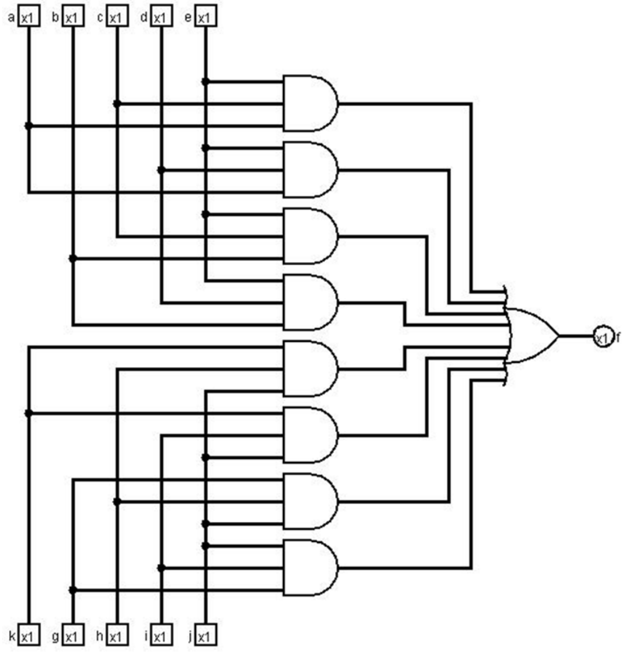
1. Propagation time

To different implementations of the same function correspond different delays. Example:

$f=[(a+b).(c+d).e]+[(k+g).(h+i).j]$

$f=e.a.c+e.a.d+e.b.c+e.b.d+j.k.h+j.k.i+j.g.h+j.g.i$



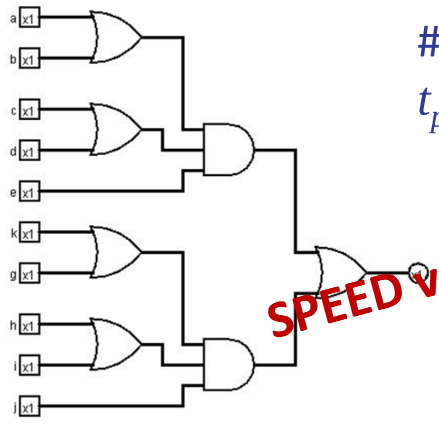


1. Propagation time

To different implementations of the same function correspond different delays. Example:

$f=[(a+b).(c+d).e]+[(k+g).(h+i).j]$

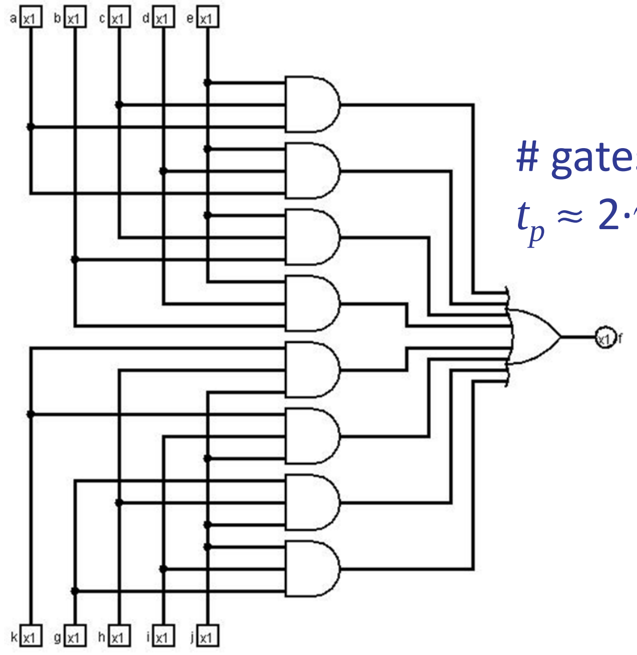
$f=e.a.c+e.a.d+e.b.c+e.b.d+j.k.h+j.k.i+j.g.h+j.g.i$



# gates = 7

$t_p \approx 3 \cdot \tau$

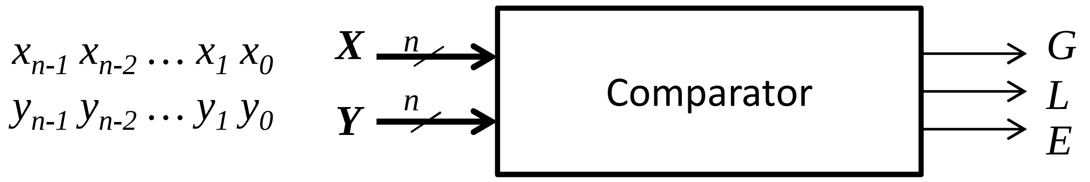
SPEED vs. COST BALANCE



# gates = 9

$t_p \approx 2 \cdot \tau$

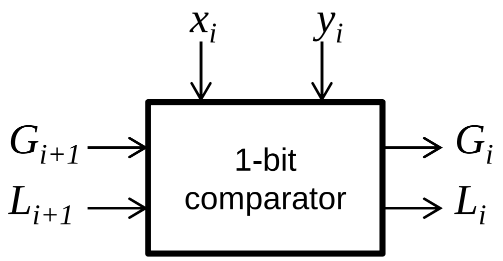
2. Example: *n*-bit comparator



$x_{n-1}\ x_{n-2} \dots x_1\ x_0$
$y_{n-1}\ y_{n-2} \dots y_1\ y_0$
X
Y
n
n
Comparator
G
L
E

```
if X > Y then G <= 1;
          elsif X < Y then L <= 1;
                    else E <= 1;
          end if;
end if;
```

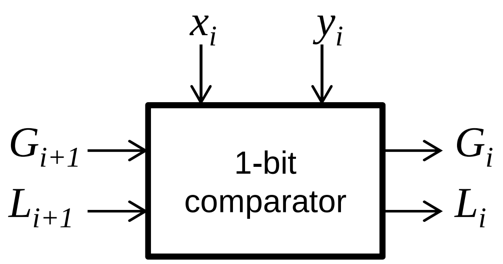
2. Example: *n*-bit comparator



	n-1	*n.2*	*n-3*	*n-4*	…	…	*1*	*0*
X:	1	0	1	1	0	…	0	0
Y:	1	0	1	0	1	…	1	0

$G_n=0$								
$L_n=0$								

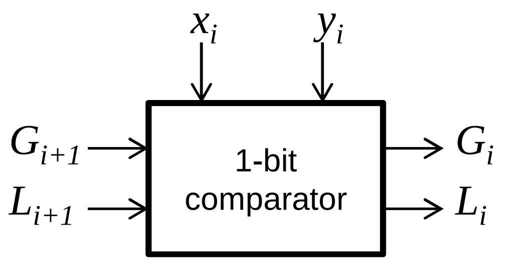
2. Example: *n*-bit comparator

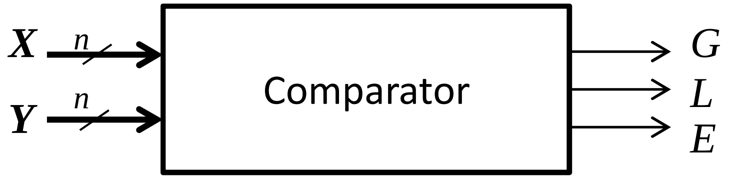


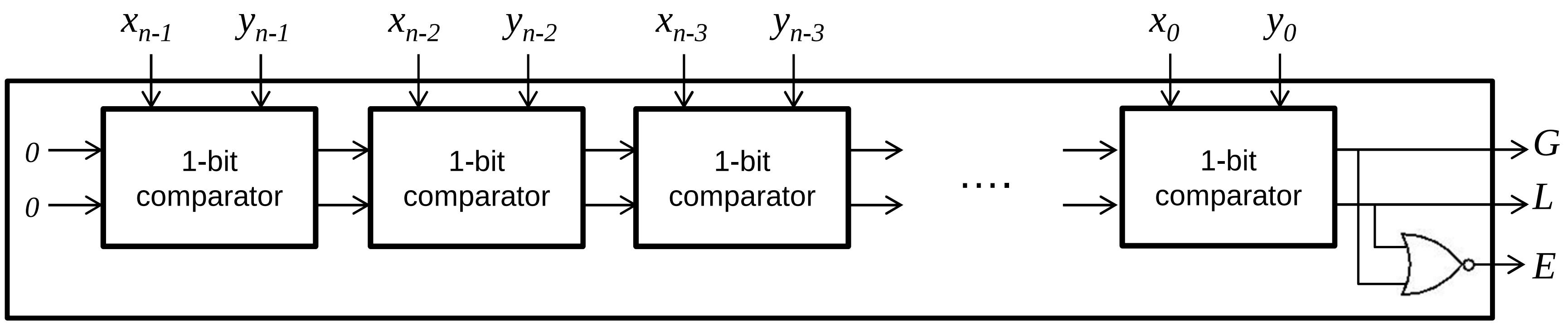
	n-1	*n.2*	*n-3*	*n-4*	…	…	*1*	*0*
X:	1	0	1	1	0	...	0	0
Y:	1	0	1	0	1	...	1	0

$G_n=0$	0	0	0	1	1		1	1
$L_n=0$	0	0	0	0	0		0	0

2. Example: *n*-bit comparator

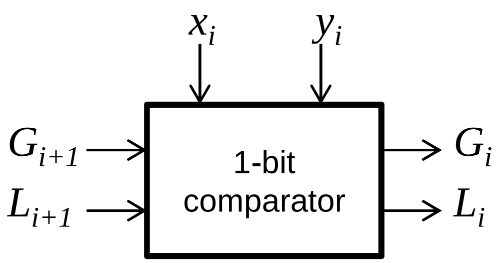






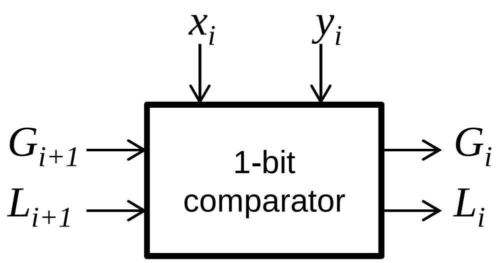
(Exercise)

Implement a 1-bit comparator



(Solution)

Implement a 1-bit comparator

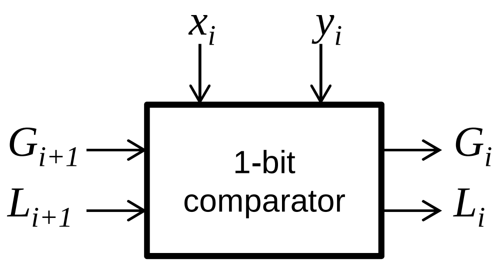


G_{i+1}	L_{i+1}	x_i	y_i	G_i	L_i
0	0	0	0		
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	x	x		
1	0	x	x		
1	1	x	x		

Universitat Autònoma de Barcelona

(Solution)

Implement a 1-bit comparator



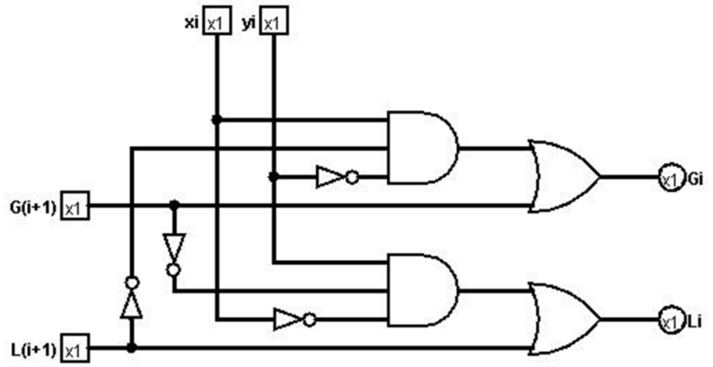
G_{i+1}	L_{i+1}	x_i	y_i	G_i	L_i
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	1	x	x	0	1
1	0	x	x	1	0
1	1	x	x	x	x

$$G_i = \bar{L}_{i+1} . x_i . \bar{y}_i + G_{i+1}$$

$$L_i = \bar{G}_{i+1} . \bar{x}_i . y_i + L_{i+1}$$

UAB Universitat Autònoma de Barcelona

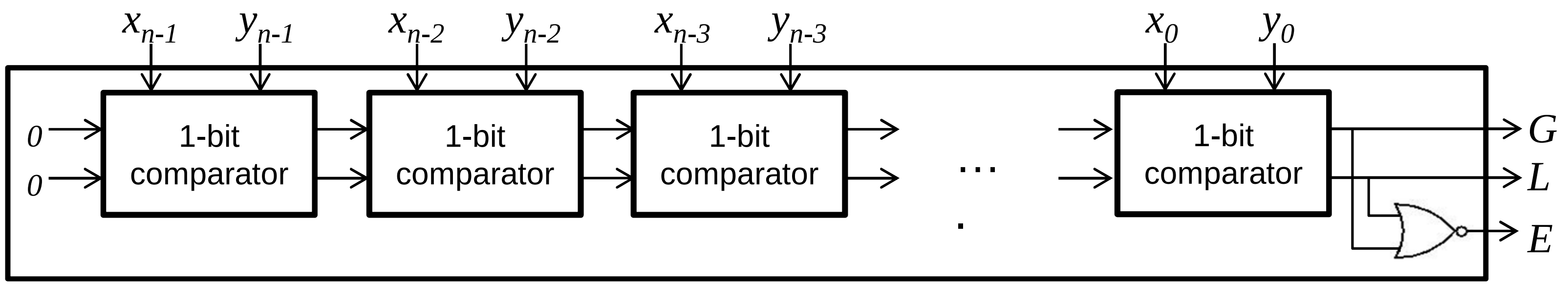
2.1. Component: 1-bit comparator



$$G_i = \bar{L}_{i+1}.x_i.\bar{y}_i + G_{i+1}$$
$$L_i = \bar{G}_{i+1}.\bar{x}_i.y_i + L_{i+1}$$

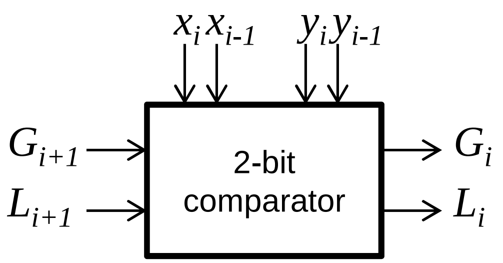
Every 1-bit comparator: 3·τ
8 gates.

n-bit comparator: (3·*n* + 1)·τ
(8·*n* + 1) gates



(Exercise)

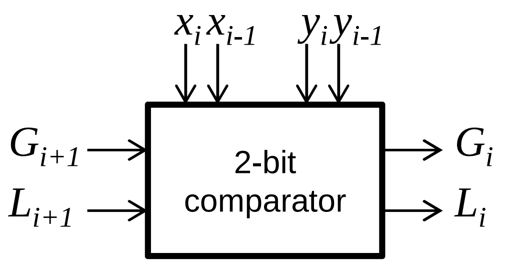
Design a 2-bit comparator



(Solution)

Design a 2-bit comparator

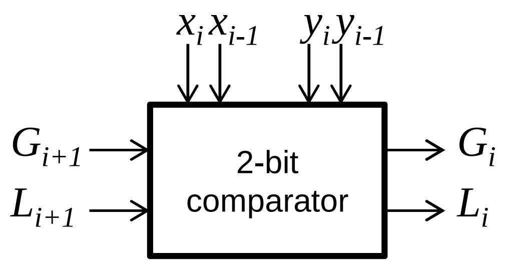
G_{i+1}	L_{i+1}	x_i	x_{i-1}	y_i	y_{i-1}	G_{i-1}	L_{i-1}
0	0	0	0	0	0		
0	0	0	0	1	x		
0	0	0	0	x	1		
0	0	0	1	0	0		
0	0	0	1	0	1		
0	0	0	1	1	x		
0	0	1	0	0	x		
0	0	1	0	1	0		
0	0	1	0	1	1		
0	0	1	1	0	x		
0	0	1	1	x	0		
0	0	1	1	1	1		
0	1	x	x	x	x		
1	0	x	x	x	x		
1	1	x	x	x	x	x	x



(Solution)

Design a 2-bit comparator

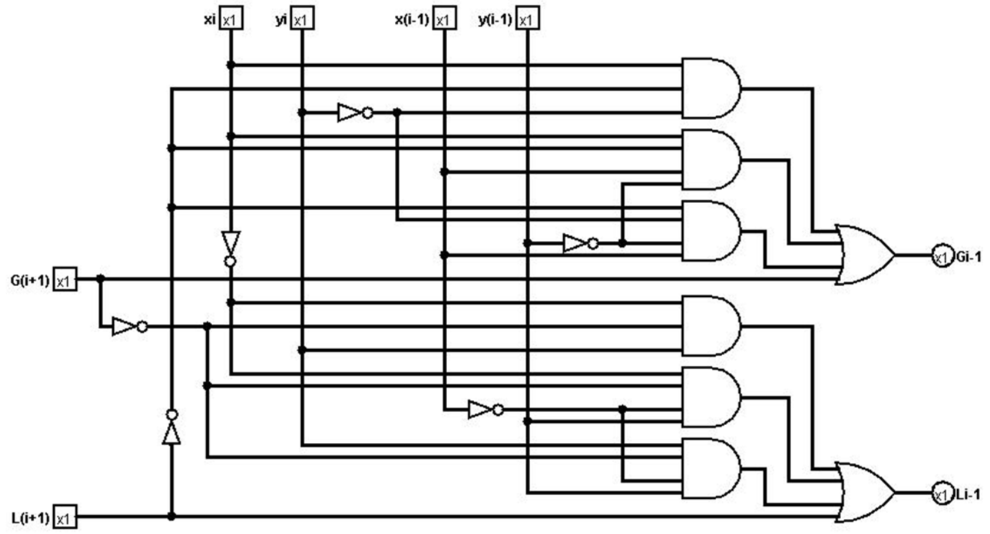
G_{i+1}	L_{i+1}	x_i	x_{i-1}	y_i	y_{i-1}	G_{i-1}	L_{i-1}
0	0	0	0	0	0	0	0
0	0	0	0	1	x	0	1
0	0	0	0	x	1	0	1
0	0	0	1	0	0	1	0
0	0	0	1	0	1	0	0
0	0	0	1	1	x	0	1
0	0	1	0	0	x	1	0
0	0	1	0	1	0	0	0
0	0	1	0	1	1	0	1
0	0	1	1	0	x	1	0
0	0	1	1	x	0	1	0
0	0	1	1	1	1	0	0
0	1	x	x	x	x	0	1
1	0	x	x	x	x	1	0
1	1	x	x	x	x	x	x



$$G_{i-1} = \bar{L}_{i+1}.x_{i-1}.\bar{y}_i.\bar{y}_{i-1} + \bar{L}_{i+1}.x_i.\ \bar{y}_i + \bar{L}_{i+1}.x_i.\ x_{i-1}.\ \bar{y}_{i-1} + G_{i+1}$$

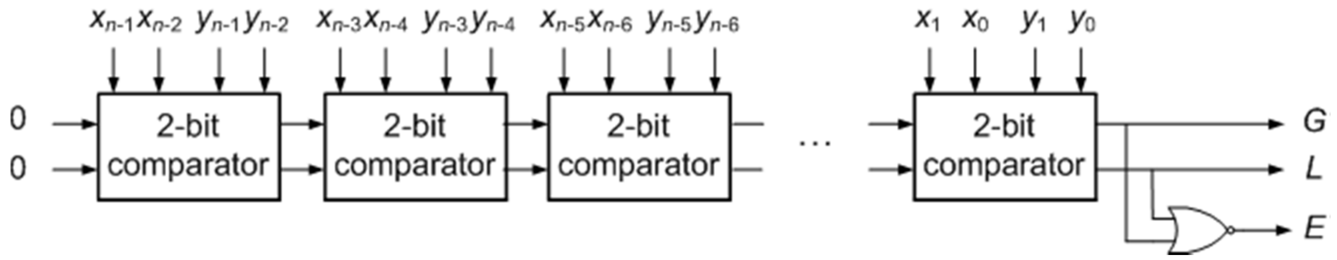
$$L_{i-1} = \bar{G}_{i+1}.\bar{x}_i.\bar{x}_{i-1}.y_{i-1} + \bar{G}_{i+1}.\bar{x}_i.\ y_i + \bar{G}_{i+1}.\bar{x}_{i-1}.\ y_i.\ y_{i-1} + L_{i+1}$$

2.2. Component: 2-bit comparator



Every 2-bit comparator: 3·τ
14 gates.

n-bit comparator: (1.5·*n* + 1)·τ
(7·*n* + 1*)* gates


x_{n-1} x_{n-2} y_{n-1} y_{n-2}
x_{n-3} x_{n-4} y_{n-3} y_{n-4}
x_{n-5} x_{n-6} y_{n-5} y_{n-6}
x_1 x_0 y_1 y_0
0
0
2-bit comparator
2-bit comparator
2-bit comparator
2-bit comparator
…
G
L
E

2.3. *n*-bit comparator

	1bit comparator	2-bit comparator
#gates	8	14
t_p	3	3

	n-bit comparator	
	C 1bit	C 2 bits
#gates	$8 \cdot n+1$	$7 \cdot n+1$
t_p	$(3 \cdot n+1)\tau$	$(1.5 \cdot n+1)\tau$

	32-bit comparator	
	C 1bit	C 2 bits
#gates	257	225
t_p	97τ	49τ

SUMMARY

- Propagation time.
- Speed vs. Cost.
- *n*-bit comparator: two implementations have been compared.

3.3 OTHER LOGIC BLOCKS

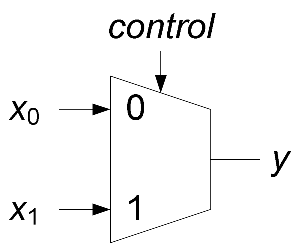
Jean-Pierre Deschamps

University Rovira i Virgili, Tarragona, Spain

1. Multiplexers.

Mainly used to implement controllable connections.

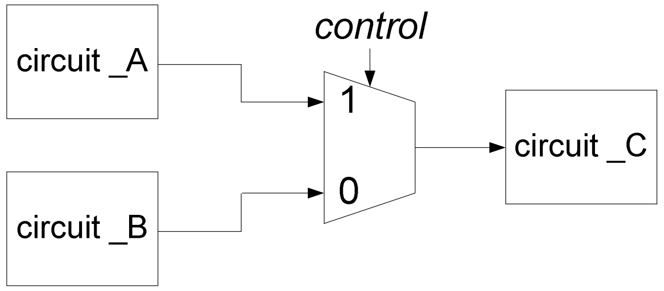
MUX2-1
(2-to-1 multiplexer)



control	*y*
0	x_0
1	x_1

1. Multiplexers.

typical application:



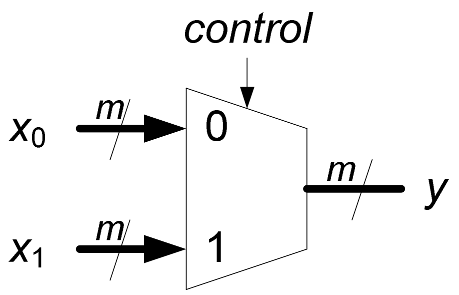
control = 1: *circuit_C* input connected to *circuit_A* output;

control = 0: *circuit_C* input connected to *circuit_B* output.

1. Multiplexers.

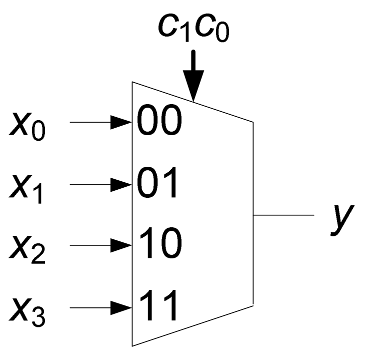
Other multiplexers:

- *m*-bit 2-to-1 multiplexer:
  m 2-to-1 multiplexers controlled by the same signal *control*
  (x_0, x_1: *m*-bit vectors)



1. Multiplexers.

MUX4-1
(4-to-1 multiplexer,
2 control signals)



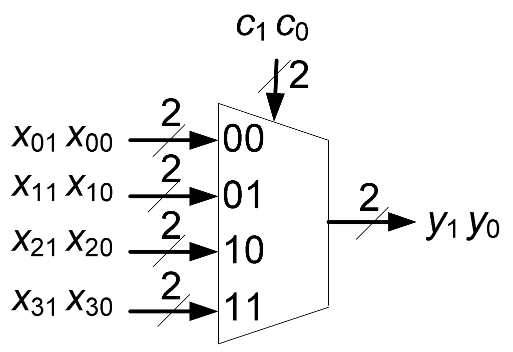
c_1 c_0	y
00	x_0
01	x_1
10	x_2
11	x_3

8-to-1 multiplexers with 3 control signals,
16-to-1 multiplexers with 4 control signals,
...
can be defined.

Universitat Autònoma de Barcelona

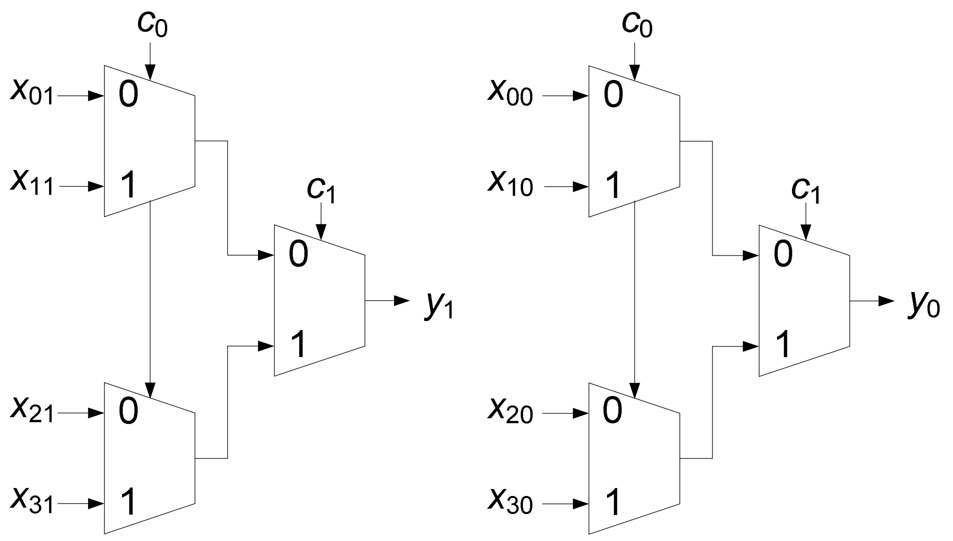
(Exercise)

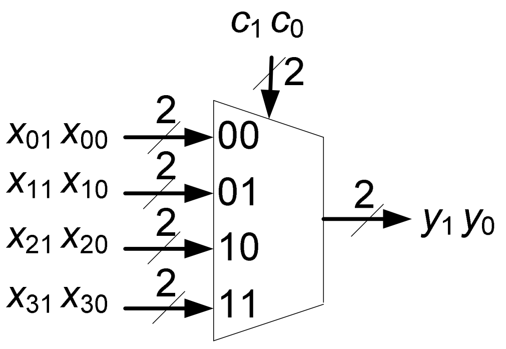
Design a 2-bit 4-to-1 multiplexer using 1-bit 2-to-1 multiplexers.



(Solution)

Design a 2-bit 4-to-1 multiplexer using 1-bit 2-to-1 multiplexers.





(Quiz)

Mark the correct statements (MUX=multiplexer)

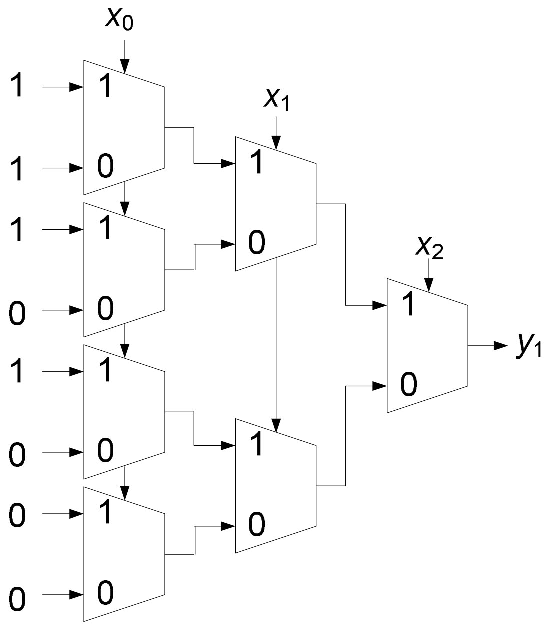
- ❑ 4 1-bit 2-1 MUXs are required to synthesize a 1-bit 4-1 MUX
- ❑ 12 1-bit 2-1 MUXs are required to synthesize a 4-bit 4-1 MUX
- ❑ 7 1-bit 2-1 MUXs are required to synthesize a 1-bit 8-1 MUX
- ❑ 21 1-bit 2-1 MUXs are required to synthesize a 3-bit 8-1 MUX

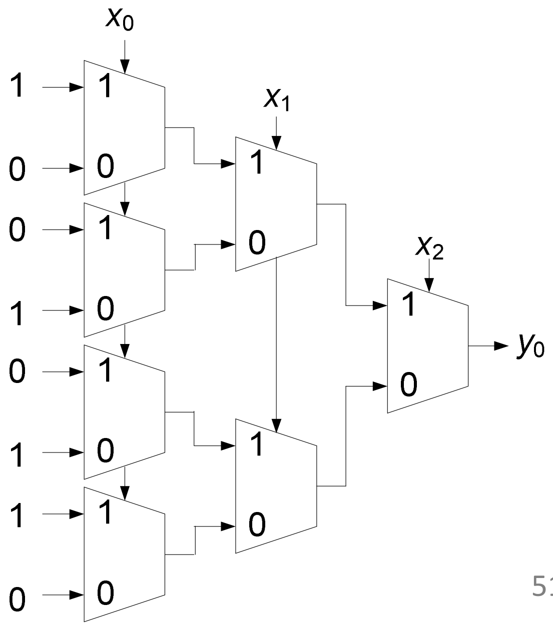
1. Multiplexers.

Multiplexers can also be used to implement functions defined by tables.

Example:

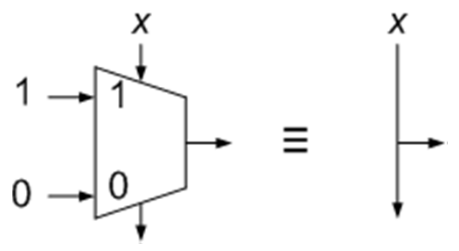
x_2 x_1 x_0	y_1 y_0
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	0 1
1 0 1	1 0
1 1 0	1 0
1 1 1	1 1

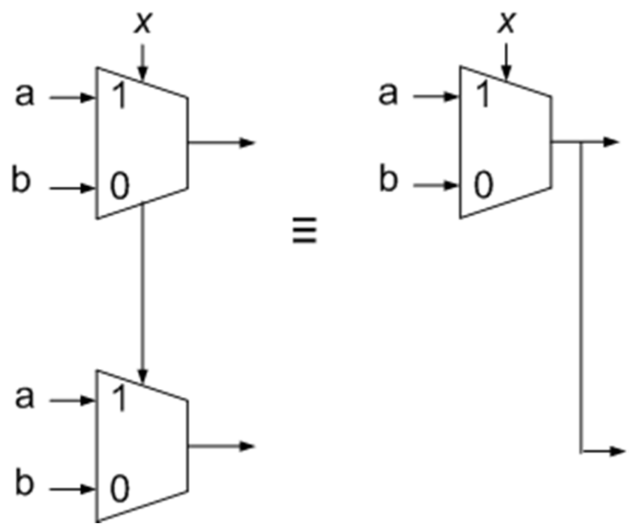




1. Multiplexers.

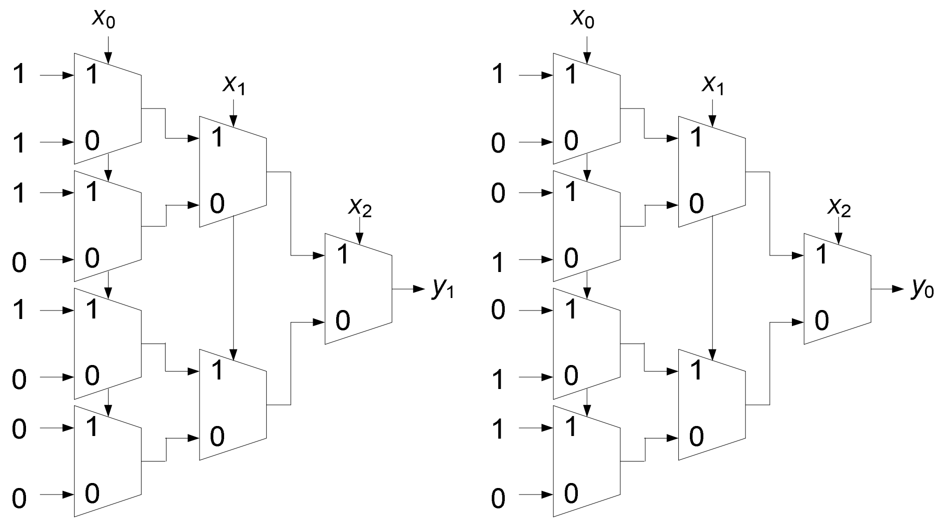
Example of (trivial) optimization rules:





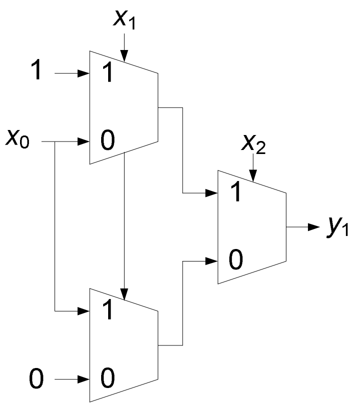
(Exercise)

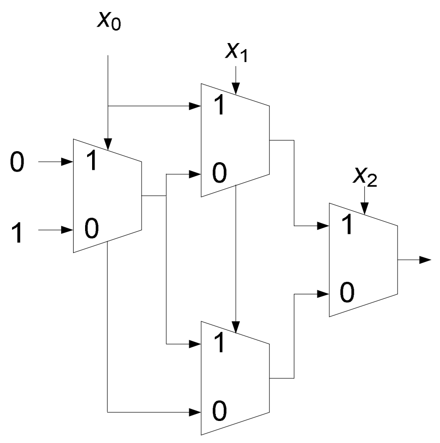
Optimize the preceding circuit.

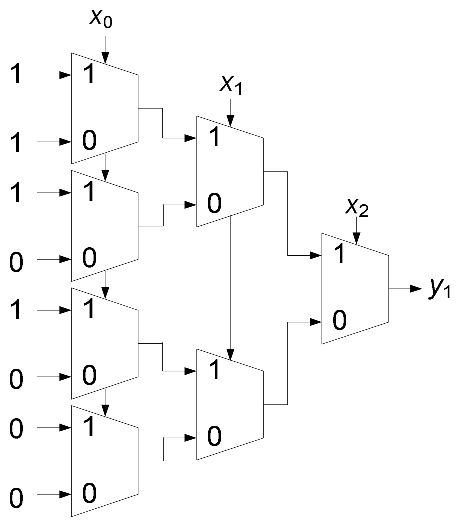


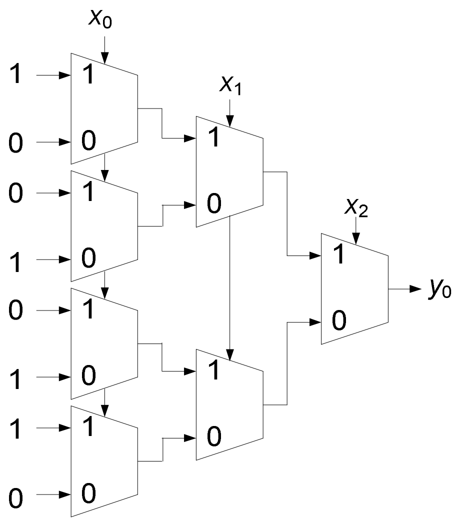
(Solution)

Optimize the preceding circuit.







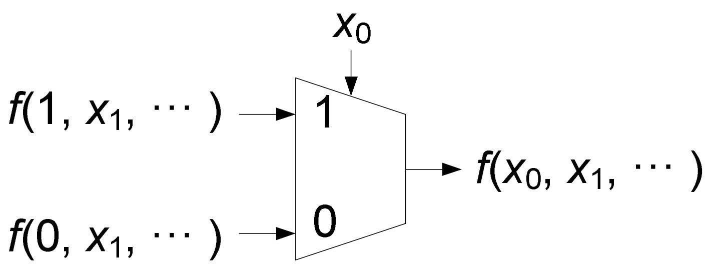


1. Multiplexers.

Comment

A multiplexer-based synthesis method is based on an iterative application of the following simple rule:

$$f(x_0, x_1, \cdots) = \bar{x}_0 \cdot f(0, x_1, \cdots) + x_0 \cdot f(1, x_1, \cdots)$$



Subfunctions $f(0, x_1, \cdots)$ and $f(1, x_1, \cdots)$ are functions of n-1 variables.

Iteratively application => constant values, simple variables, or already available subfunctions.

2. Multiplexers and memory blocks.

Read Only Memory (ROM) blocks have been used to implement functions defined by a table.

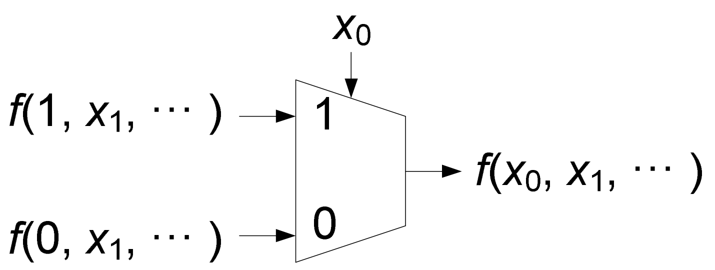
More generally consider the use of

small ROM (LUT = Look Up Tables) + multiplexers.

Example: assume that 6-input LUT's are available. Iteratively apply

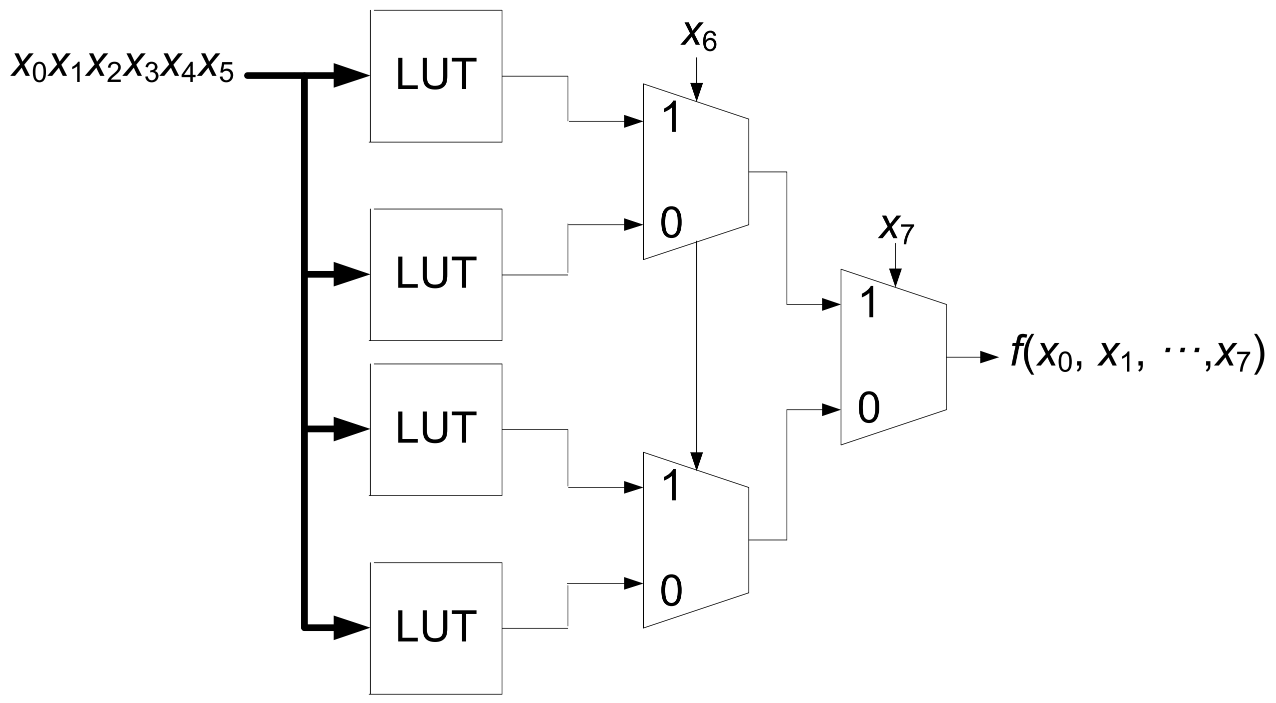
$$f(x_0, x_1, \cdots) = \bar{x}_0 \cdot f(0, x_1, \cdots) + x_0 \cdot f(1, x_1, \cdots)$$

until all obtained subfunctions are functions of at most 6 variables.



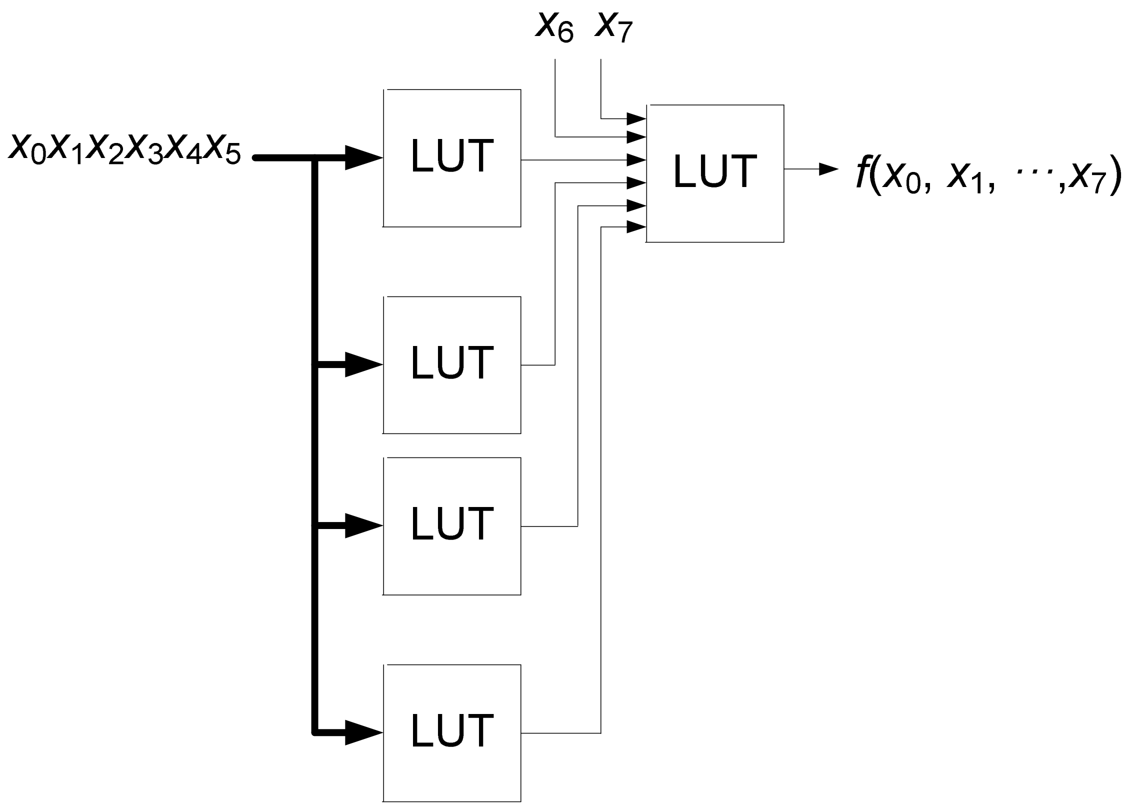
2. Multiplexers and memory blocks.

Example: 8-variable function



2. Multiplexers and memory blocks.

Example: 8-variable function (with LUTs)



2. Multiplexers and memory blocks.

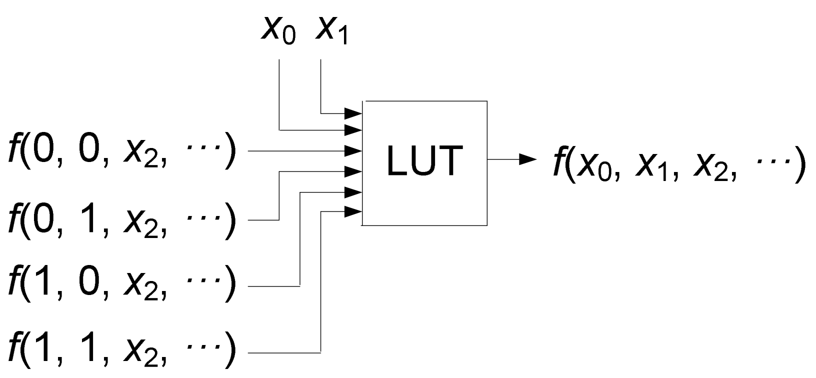
Comment

The preceding example is based on an iterative application of the following simple rule:

$f(x_0, x_1, \cdots) = \bar{x}_0 \cdot \bar{x}_1 \cdot f(0, 0, x_2, \cdots) + x_0 \cdot \bar{x}_1 \cdot f(1, 0, x_2, \cdots) +$
$\bar{x}_0 \cdot x_1 \cdot f(0, 1, x_2, \cdots) + x_0 \cdot x_1 \cdot f(1, 1, x_2, \cdots)$.

Subfunctions $f(0, 0, x_2, \cdots)$, $f(0, 1, x_2, \cdots)$, $f(1, 0, x_2, \cdots)$ and $f(1, 1, x_2, \cdots)$ are functions of n-2 variables.

Iteratively application => constant values, simple variables, or already available subfunctions.



UAB
Universitat Autònoma de Barcelona

(Quiz)

How many 6-input LUTs are necessary to synthesize any 10-variable switching function?

1. 21

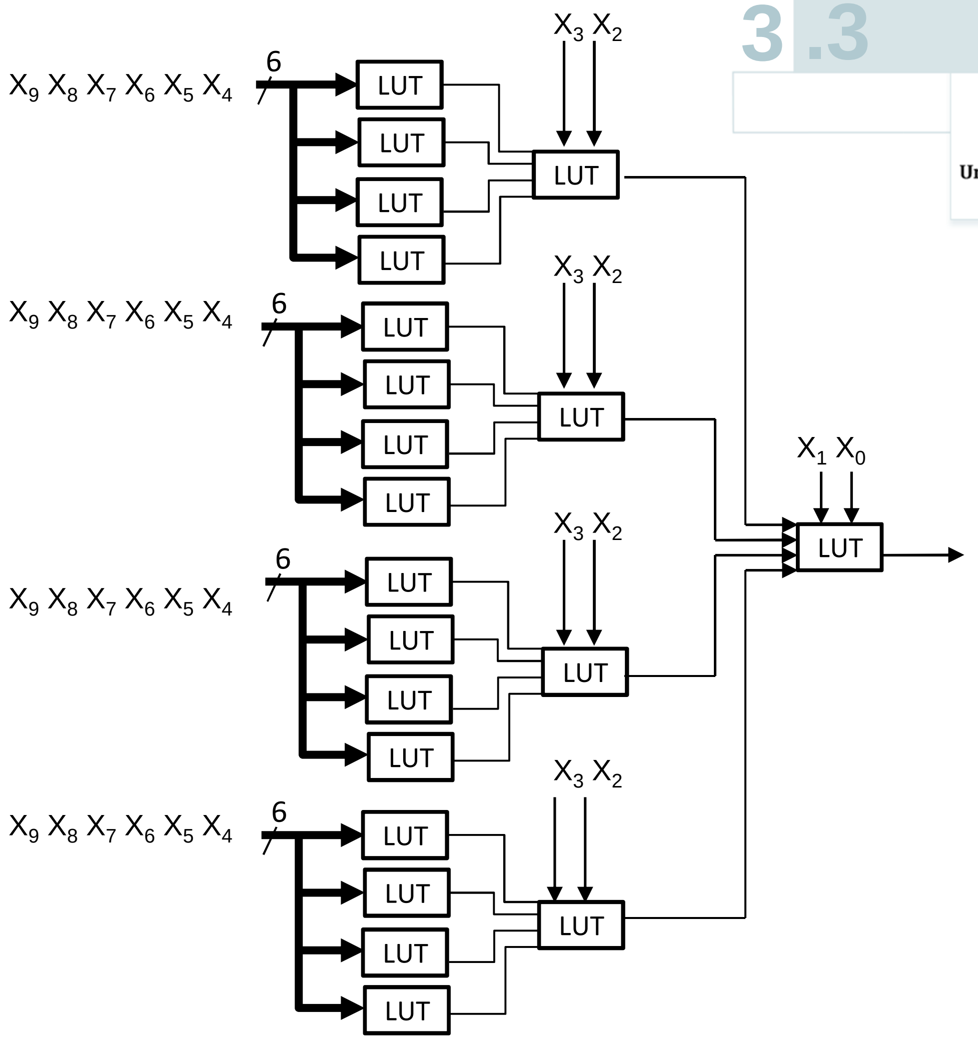
2. 32

3. 64

4. 128

Circuit implementing the 10 variables switching function

(this slide is not included in the video)



3. Planes.

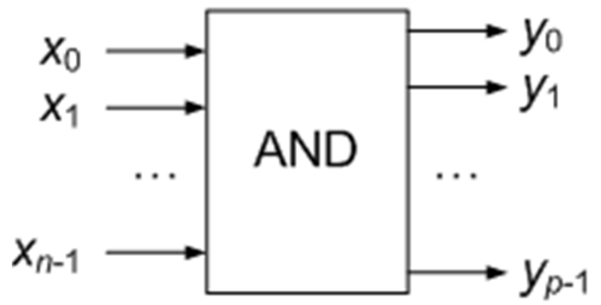
(n, p) AND-plane: implements p n-variable functions y_j;

each y_j is a product of literals:

$y_j = w_{j,0}w_{j,1} \cdots w_{j,n-1}$ where $w_{j,i} \in \{1, x_i, \bar{x}_i\}$.

According to the technology, functions y_j can be

- predefined
- defined by the user (field programmable).



3. Planes.

(p, s) OR-plane: implements
s p-variable functions z_j;

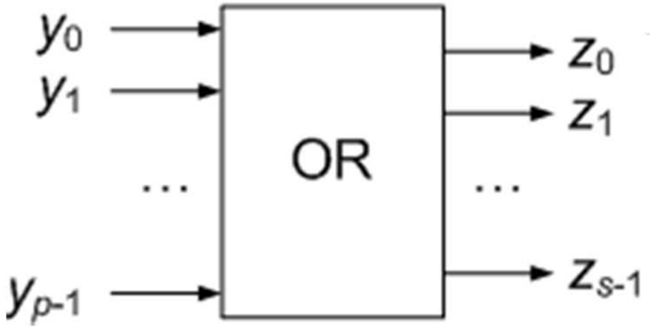
each z_j is a Boolean sum of variables :

$$z_j = w_{j,0} \vee w_{j,1} \vee \cdots \vee w_{j,p-1}$$

where $w_{j,i} \in \{0, y_i\}$.

Functions z_j can be

- predefined
- defined by the user (field programmable).

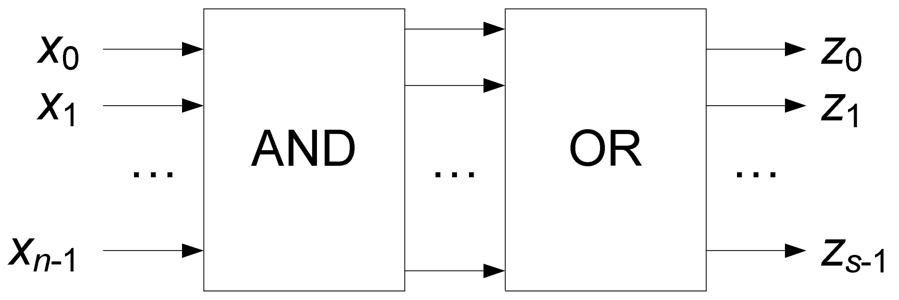


3. Planes.

Any set of *s* Boolean functions that can be expressed as Boolean sums of at most *p* products of at most *n* literals can be implemented by a circuit made up of

- an (*n*, *p*) AND plane,
- a (*p*, *s*) OR plane.

Different names according to the technology and/or the manufacturer (PAL, PLA, PLD, ···).

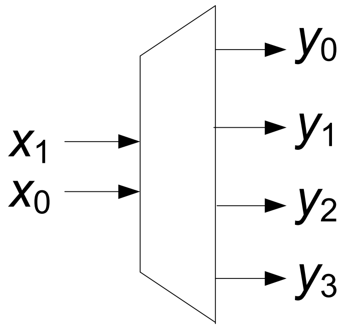


4. Address decoders and 3-state buffers.

n-to-2^n address decoder:

- n inputs $x_{n-1}\, x_{n-2} \cdots x_0$,
- $m = 2^n$ outputs $y_0\, y_1 \cdots y_{m-1}$,
- $y_i = 1$ iff $x_{n-1} \cdot 2^{n-1} + x_{n-2} \cdot 2^{n-2} + \cdots + x_1 \cdot 2 + x_0 = i$.

Example: 2-to-4 address decoder



$x_1\, x_0$	$y_0\, y_1\, y_2\, y_3$
0 0	1 0 0 0
0 1	0 1 0 0
1 0	0 0 1 0
1 1	0 0 0 1

4. Address decoders and 3-state buffers.

Comments

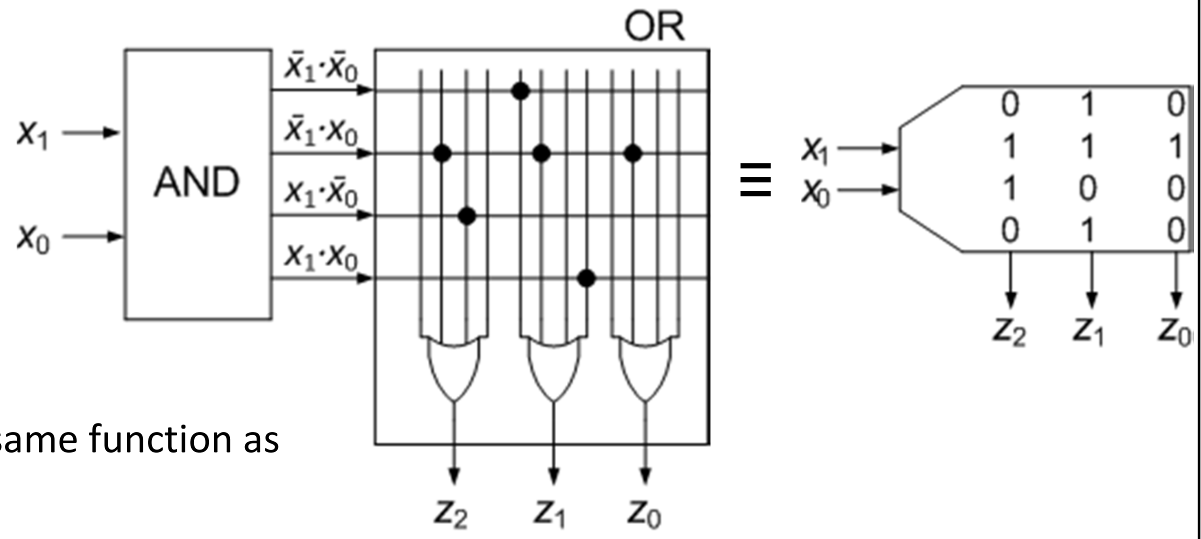
- An address decoder implements the same function as an (n, 2^n) AND plane that generates the 2^n n-variable minterms

$$m_j = w_{j,0} w_{j,1} \cdots w_{j,n-1}$$

where $w_{j,i} \in \{x_i, \bar{x}_i\}$.

- Address decoder + (2^n, s) OR plane: same function as a ROM storing 2^n s-bit word.

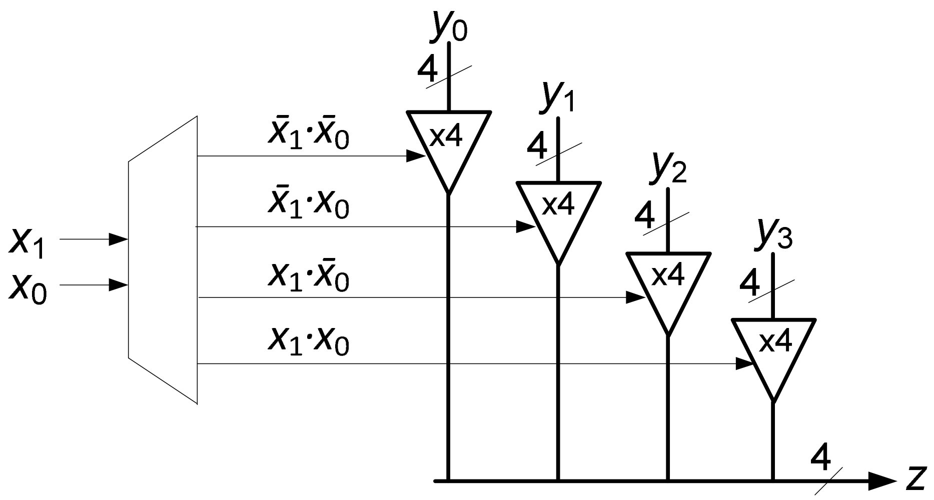
Example: 2-variable functions.



4. Address decoders and 3-state buffers.

Address decoders can be used to control 3-state buffers.

Example (y_0, y_1, y_2, y_3 and z are 4-bit vectors):
equivalent to a 4-bit MUX4-1

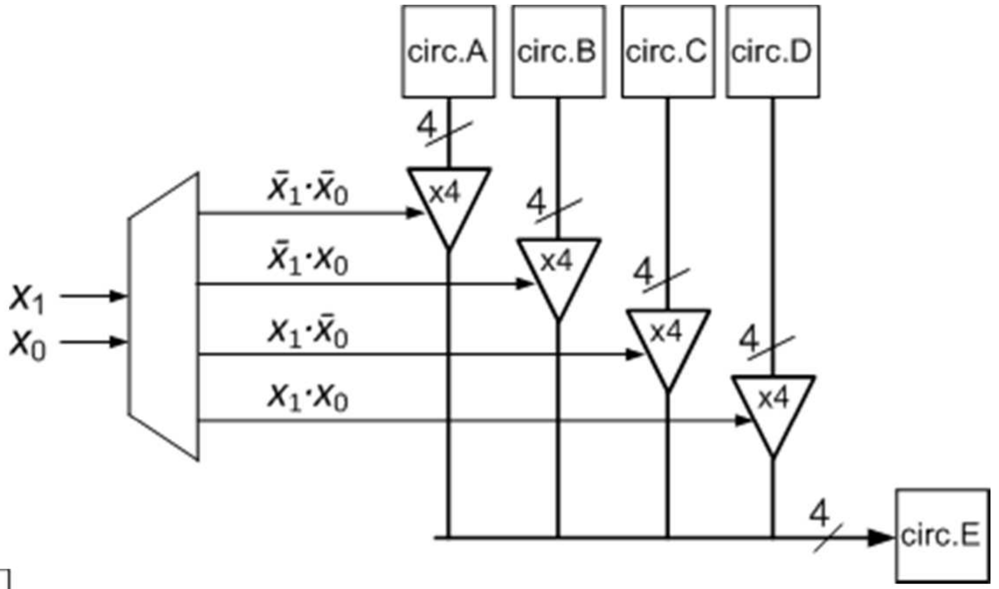


4. Address decoders and 3-state buffers.

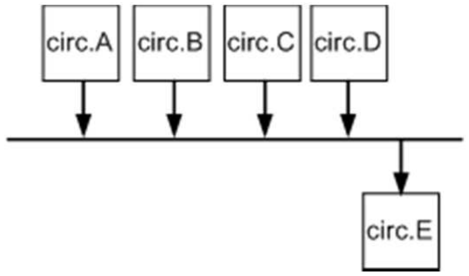
When $x_1\, x_0$ =

00: *circuit_E <= circuit_A;*
01: *circuit_E <= circuit_B;*
00: *circuit_E <= circuit_C;*
00: *circuit_E <= circuit_D;*

= example of BUS



Symbol:



SUMMARY

- Multiplexers implement controllable connections.
- Multiplexers can also be used to implement Boolean functions.
- Small memory blocks (Look Up Tables) are used to implement Boolean functions.
- Predefined or field programmable planes are used to implement Boolean functions.
- Address decoders and 3-state buffers implement buses.

3.4 PROGRAMMING LANGUAGE STRUCTURES

Jean-Pierre Deschamps

University Rovira i Virgili, Tarragona, Spain

Some classical programming language instructions have a straightforward translation in terms of digital components. Examples:

1. If ··· then ··· else ···

2. Case ··· is ···

3. For ··· loop ···

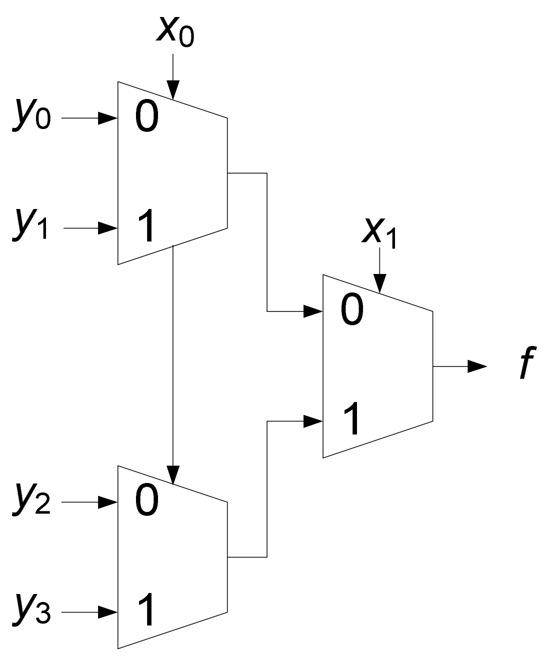
4. Procedure call

UAB Universitat Autònoma de Barcelona

1. If … then … else …

Example: a binary decision algorithm.

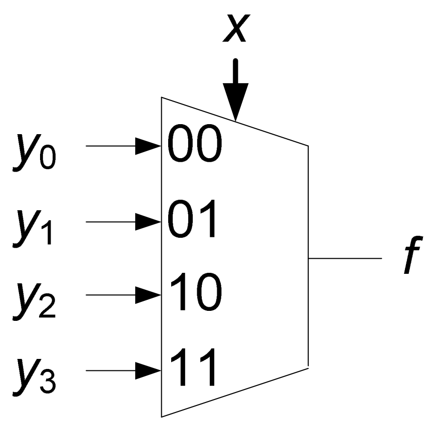
```
if $x_1$ = 0 then
   if $x_0$ = 0 then f <= $y_0$;
   else f <= $y_1$;
   end if;
else
   if $x_0$ = 0 then f <= $y_2$;
   else f <= $y_3$;
   end if;
end if;
```



2. Case … is …

Example: program equivalent to the preceding binary decision algorithm.

```
case x is
  when "00" => f <= y0;
  when "01" => f <= y1;
  when "10" => f <= y2;
  when "11" => f <= y3;
end case;
```



3. For ··· loop ···

UAB Universitat Autònoma de Barcelona

Loops are associated with iterative structures.

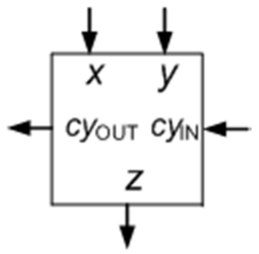
Example: addition algorithm that computes $z = x + y$
(x and y: 4-digit decimal numbers,
z: 5-digit decimal number)

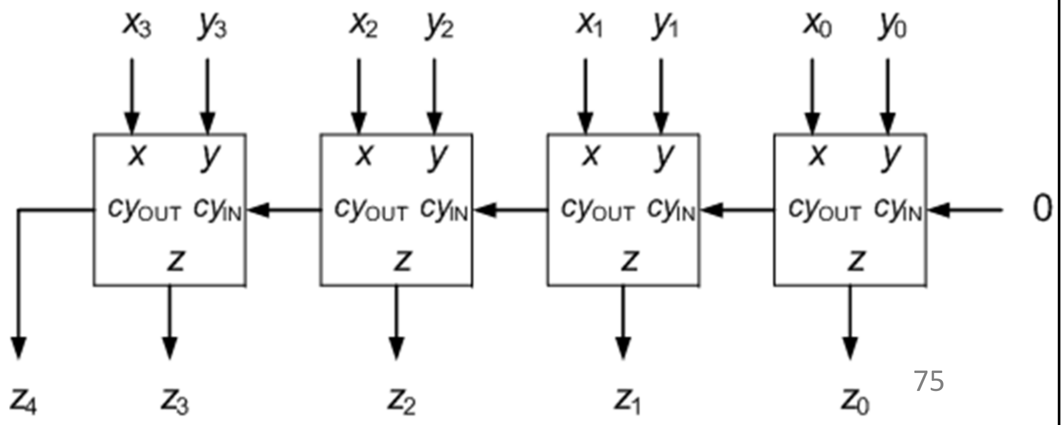
```
cy(0) <= 0;
for i in 0 to 3 loop
  s(i) <= x(i) + y(i) + cy(i);
  if s(i) > 9 then z(i) <= s(i) - 10; cy(i+1) <= 1;
  else z(i) <= s(i); cy(i+1) <= 0;
  end if;
end loop;
z(4) <= cy(4);
```

3. For ··· loop ···

```
cy(0) <= 0;
for i in 0 to 3 loop
-------- loop body --------------------------------
   s(i) <= x(i) + y(i) + cy(i);
   if s(i) > 9 then z(i) <= s(i) - 10; cy(i+1) <= 1;
   else z(i) <= s(i); cy(i+1) <= 0;
   end if;
---------------------------------------------------------

end loop;
z(4) <= cy(4);
```





3. For ··· loop ···

Comments.

- Other (sequential) implementations will be seen later on.

- Not any loop can be implemented in this way.

  Example: "while *condition* loop *operation*“.

  If the maximum number of times that *condition* holds true
  - ✓ is unknown,
  - ✓ is a too large number,

  a sequential implementation must be considered.

4. Procedure call

Example: assume that a procedure MAC(w, x, y, z) (multiply and accumulate) has been defined; it executes

$$z = w + x \cdot y.$$

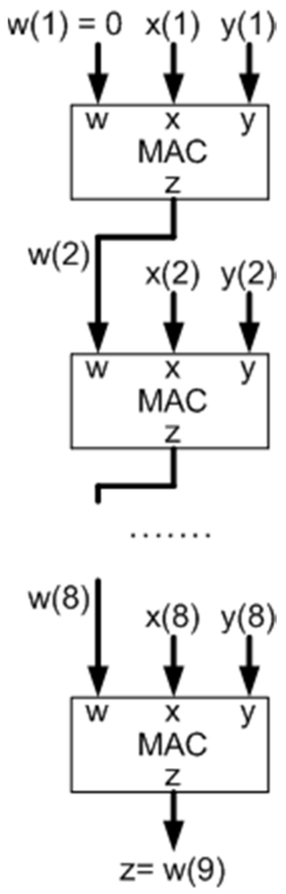
The following algorithm computes

$$z = x_1 \cdot y_1 + x_2 \cdot y_2 + \cdots + x_8 \cdot y_8:$$

```
w(1) <= 0;
for i in 1 to 8 loop
  MAC(w(i), x(i), y(i), w(i+1));
end loop;
z <= w(9);
```

4. Procedure call

```
w(1) <= 0;
for i in 1 to 8 loop
  MAC(w(i), x(i), y(i), w(i+1));
end loop;
z <= w(9);
```



Conclusion:

- for ··· loop => iterative structure;
- MAC calls => blocks to be defined.

Procedure calls are associated with hierarchical descriptions.

5. COMMENTS

Conclusion: some classical programming language instructions have a straightforward translation in terms of digital components =>

- define programming languages to specify digital systems (VHDL, Verilog, C/C++); (the so-called Hardware Description Languages)

- develop synthesis tools (software) to translate programs to circuits.

SUMMARY

- Binary decision algorithm can be implemented with multiplexers.
- Case constructs can also been implemented by multiplexers.
- Loops correspond to iterative structures.
- Procedure calls correspond to hierarchical descriptions
- A conclusion about Hardware Description Languages and Synthesis tools.