MANAGEMENT AND ANALYSIS OF PHYSICS DATASET (MOD. A)

Fundamentals of Boolean Algebra
Combinatorial Functions

Postulates of the Boolean algebra

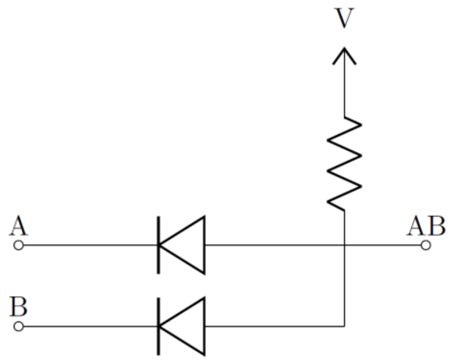
- Digital circuits or logic circuits deal with quantized signals (lack or presence of a signal)
- Fundamental elements of the digital computing
- "Logic" because we can apply analysis and synthesis methods derived from Boolean algebra
- Class of elements M and two operators · and +

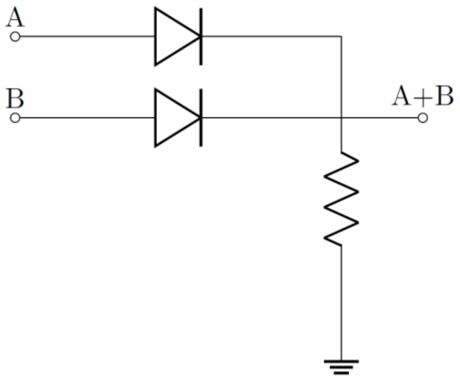
P1:	$A + B = B + A$	$AB = BA$	Commutative
P2:	$A + 0 = A$	$A \cdot 1 = A$	Identity
P3:	$A + \overline{A} = 1$	$A\overline{A} = 0$	Complement
P4:	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$	Distributive

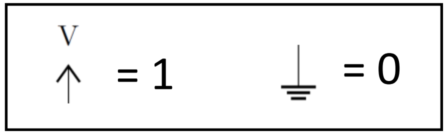
- Duality of the operators

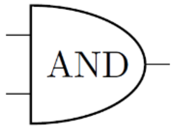
Postulates of the Boolean algebra

- Proof of non contradiction: M are connections, · is the circuital operator AND and + is the OR

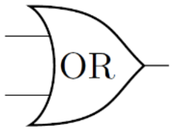








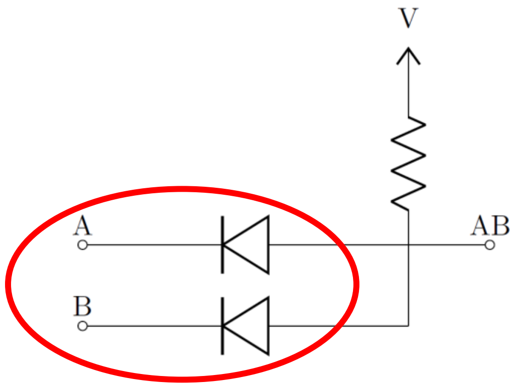
A	B	AB
0	0	0
1	0	0
0	1	0
1	1	1



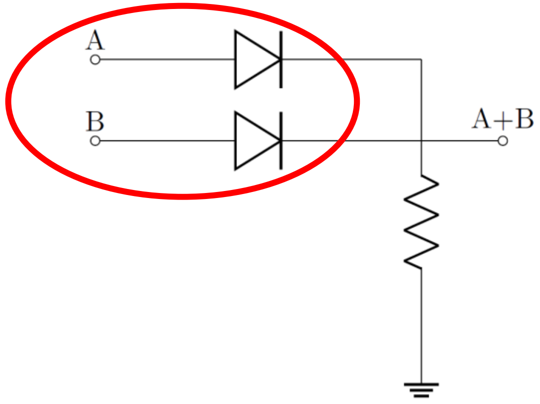
A	B	A+B
0	0	0
1	0	1
0	1	1
1	1	1

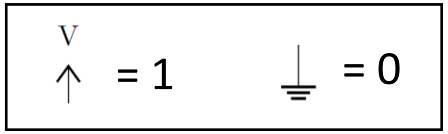
Postulates of the Boolean algebra

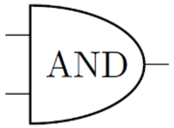
- Proof of non contradiction: M are connections, · is the circuital operator AND and + is the OR



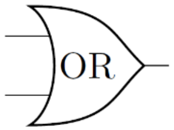
P1







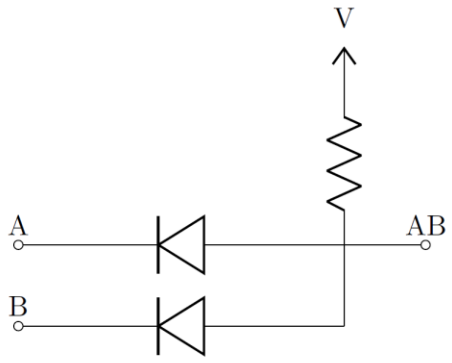
A	B	AB
0	0	0
1	0	0
0	1	0
1	1	1

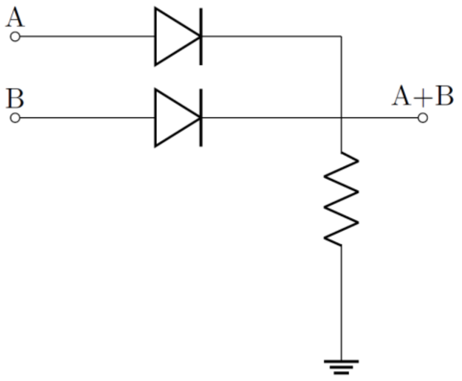


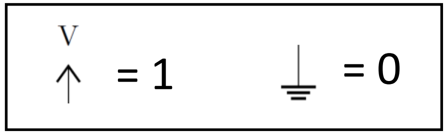
A	B	A+B
0	0	0
1	0	1
0	1	1
1	1	1

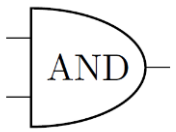
Postulates of the Boolean algebra

- Proof of non contradiction: M are connections, · is the circuital operator AND and + is the OR

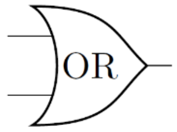








A	B	AB
0	0	0
1	0	0
0	1	0
1	1	1

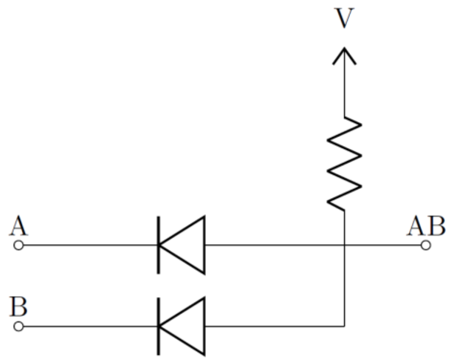


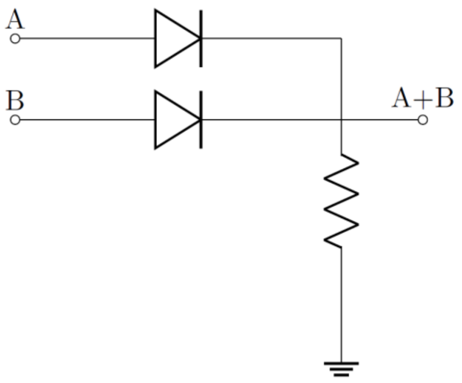
A	B	A+B
0	0	0
1	0	1
0	1	1
1	1	1

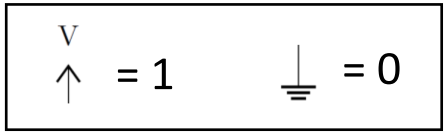
P2

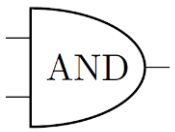
Postulates of the Boolean algebra

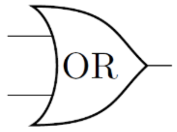
- Proof of non contradiction: M are connections, · is the circuital operator AND and + is the OR











A	B	AB
0	0	0
1	0	0
0	1	0
1	1	1

A	B	A+B
0	0	0
1	0	1
0	1	1
1	1	1

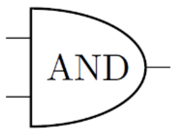
P3

Postulates of the Boolean algebra

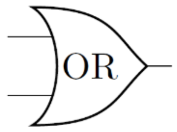
- Proof of non contradiction: M are connections, · is the circuital operator AND and + is the OR

P4

A	B	C	A+BC	(A+B)(A+C)
0	0	0	0	0
1	0	0	1	1
0	1	0	0	0
1	1	0	1	1
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	1	1	1



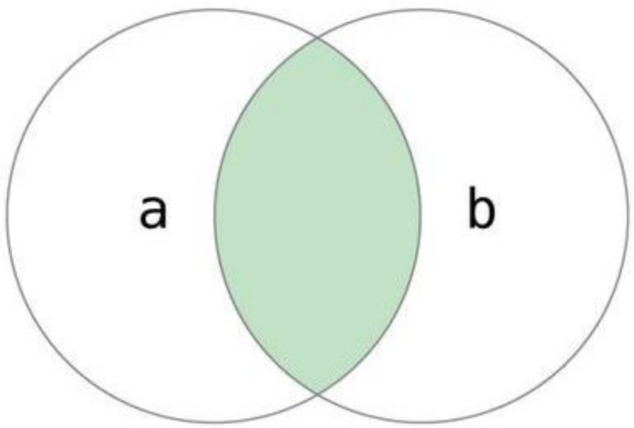
A	B	AB
0	0	0
1	0	0
0	1	0
1	1	1

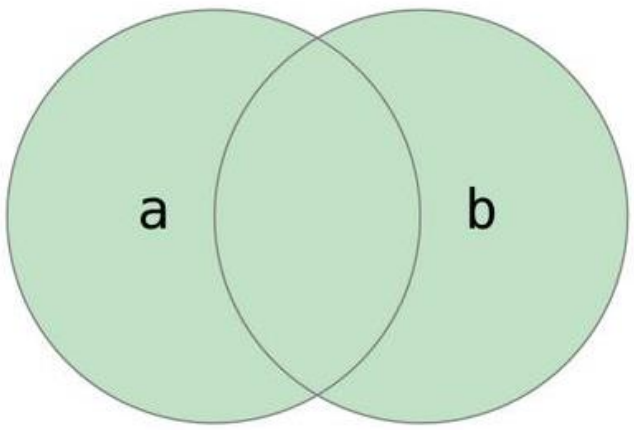


A	B	A+B
0	0	0
1	0	1
0	1	1
1	1	1

Postulates of the Boolean algebra

- The postulates are valid also for other classes
- For instance: M is the class of all subsets of the set M, · is the intersection and + is the union
- Graphical representation can be used to quickly proof theorems





Simple theorems

T1: $A + A = A$ $\quad AA = A$

Proof: $A + A = (A + A) \cdot 1 = (A + A)(A + \bar{A}) = A + A\bar{A} = A + 0 = A$

T2: $A + 1 = 1$ $\quad A \cdot 0 = 0$

Proof: $A + 1 = (A + 1) \cdot 1 = (A + 1)(A + \bar{A}) = A + \bar{A} \cdot 1 = A + \bar{A} = 1$

T3: $A + AB = A$ $\quad A(A + B) = A$

Proof: $A + AB = (A \cdot 1) + AB = A(1 + B) = A \cdot 1 = A$

De Morgan laws

$$\overline{A+B} = \overline{A} \cdot \overline{B}$$

$$\overline{A \cdot B} = \overline{A} + \overline{B}$$

Proof:

$$(A+B) + \overline{A} \cdot \overline{B} = \left(A + B + \overline{A}\right)\left(A + B + \overline{B}\right) = (\mathrm{B}+1)(A+1) = 1 \cdot 1 = 1$$

$$(A+B) \cdot \overline{A} \cdot \overline{B} = A \cdot \overline{A} \cdot \overline{B} + B \cdot \overline{A} \cdot \overline{B} = \overline{B} \cdot 0 + \overline{A} \cdot 0 = 0 + 0 = 0$$

From P3 we can conclude that A+B is the complementary of $\overline{A} \cdot \overline{B}$

Generalization: the complementary of a Boolean function results from the exchange of the dual operators and from the exchange of the operands with their complement

Exercises

1. Demonstrate that $(A + B)\left(\overline{A} + C\right) = \overline{A}B + AC$

2. Demonstrate that $f(A, B) = \overline{A}\,\overline{B} + AB = \overline{f}(\overline{A}, B)$

Combinatorial functions

Combinatorial functions

- A simple way of describing a Boolean function is the truth table

A	B	$f(A,B) = \overline{A} + \overline{B}$
0	0	1
1	0	1
0	1	1
1	1	0

- It describes f as function of its input -> combinatory function
- Many functions can be built from the same truth table

$$
\begin{aligned}
f(A,B) &= \overline{A \cdot B} \\
&= \overline{A}\,\overline{B} + \overline{A}\,B + A\,\overline{B} \\
&= \overline{B} + \overline{A}\,B
\end{aligned}
$$

Which is the best?

Canonical form

- Minterm (m) of n variables is the Boolean product of all the variables where they appears only once (either in the complemented or uncomplemented form)
- They are 2^n. For instance, for $n = 2$ they are $\overline{A}\,\overline{B}, A\,\overline{B}, \overline{A}\,B$ and $A\,B$
- Similarly, Maxterm (M) can be defined for the sum

A	B	Minterm	Maxterm
0	0	$m_0 = \overline{A}\,\overline{B}$	$M_0 = \overline{A} + \overline{B}$
1	0	$m_1 = A\,\overline{B}$	$M_1 = A + \overline{B}$
0	1	$m_2 = \overline{A}\,B$	$M_2 = \overline{A} + B$
1	1	$m_3 = A\,B$	$M_3 = A + B$

Canonical form

- m_i is always equal to 0 except in one case (variables in line i)
- M_i is always equal to 1 except in one case (variables in line $2^n - 1 - i$)
- The product of two distinct minterms is always 0 and the sum of two distinct maxterm is always 1

$$\overline{m_i} = M_{2^n-1-i} \qquad \overline{M_i} = m_{2^n-1-i}$$

$$\sum_{i=0}^{2^n-1} m_i = 1 \qquad \prod_{i=0}^{2^n-1} M_i = 0$$

Canonical form

- Combinatorial functions expressed as a sum of minterms or product of maxterms are said to be in canonical form

A	B	$f(A,B) = \overline{A} + \overline{B}$
0	0	1
1	0	1
0	1	1
1	1	0

$$f(A,B) = \mathrm{m}_0 + \mathrm{m}_1 + \mathrm{m}_2$$
$$f(A,B) = M_0$$

A	B	F_k
0	0	f_0
1	0	f_1
0	1	f_2
1	1	f_3

$$F_k = \sum_{i=0}^{2^n-1} f_i\, m_i \qquad F_k = \prod_{i=0}^{2^n-1} (f_i + M_{2^n-1-i})$$

Canonical form

- All the possible Boolean function of n variables are 2^{2^n}

f_0	f_1	f_2	f_3	F_k
0	0	0	0	**0**
1	0	0	0	NOR
0	1	0	0	$A\overline{B}$
1	1	0	0	$\overline{B}$
0	0	1	0	$\overline{A}B$
1	0	1	0	$\overline{A}$
0	1	1	0	XOR
1	1	1	0	$NAND$
0	0	0	1	AND
1	0	0	1	$\overline{XOR}$
0	1	0	1	A
1	1	0	1	$A+\overline{B}$
0	0	1	1	B
1	0	1	1	$\overline{A}+B$
0	1	1	1	OR
1	1	1	1	**1**

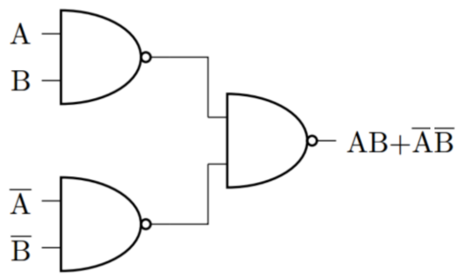
From function to circuit

- Canonical form is the most generic, but it is not the simplest
- For building a circuit it is better to simplify
- Passive circuits cannot be used because they degrades the signal after few logic levels -> active components at least in the output stage
- The simplest amplifier is the common emitter transistor -> signal inversion -> NAND or NOR
- Any combinatorial Boolean function can be written with them
  - The complement of a variable is done wiring together the two inputs or applying a 1 for NAND and 0 for NOR
  - By De Morgan NAND is the OR of complements and NOR is the AND of complements

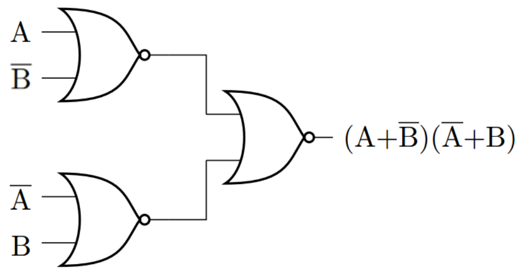
From function to circuit

$f = A\,B + \overline{A}\,\overline{B}$

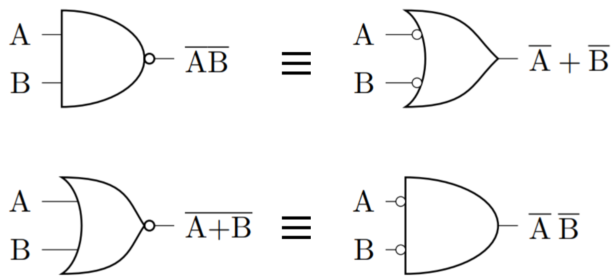
$f = (A + \overline{B})(\overline{A} + B)$



Sum of products



Product of sums

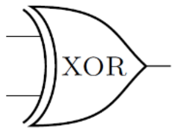


De Morgan duality

XOR

- $A \oplus B = \overline{A} \oplus \overline{B} = A\,\overline{B} + \overline{A}\,B$
- $\overline{A \oplus B} = \overline{A} \oplus B = \overline{A}\,\overline{B} + A\,B$

- $\overline{A} = 1 \oplus A$
- $A + B = \overline{\overline{A}\,\overline{B}} = 1 \oplus [(1 \oplus A)(1 \oplus B)]$
- $A\,B = \overline{\overline{A} + \overline{B}} = 1 \oplus [(1 \oplus A) + (1 \oplus B)]$



A	B	A $\oplus$ B
0	0	0
1	0	1
0	1	1
1	1	0

Graphical simplification

- A function is represented as union of intersections of subsets
- Veitch or Karnaugh maps

	A	
B	m_3	m_2
	m_1	m_0

	-------A-------			
B	m_3	m_7	m_6	m_2
	m_1	m_5	m_4	m_0
		-------C-------		

	-------A-------				
-------B-------	m_3	m_7	m_6	m_2	
	m_{11}	m_{15}	m_{14}	m_{10}	-------D-------
	m_9	m_{13}	m_{12}	m_8	
	m_1	m_5	m_4	m_0	
		--------C-------			

Graphical simplification

- A function is represented as union of intersections of subsets
- Veitch or Karnaugh maps

A

B	m_3	m_2
	m_1	m_0

$$m_0 = \overline{A}\,\overline{B}$$

-------A-------

B	m_3	m_7	m_6	m_2
	m_1	m_5	m_4	m_0

-------C-------

-------A-------

B	m_3	m_7	m_6	m_2	
B	m_{11}	m_{15}	m_{14}	m_{10}	D
	m_9	m_{13}	m_{12}	m_8	D
	m_1	m_5	m_4	m_0	

--------C-------

Graphical simplification

- A function is represented as union of intersections of subsets
- Veitch or Karnaugh maps

A

B

m_3	m_2
m_1	m_0

$$m_1 = A\,\overline{B}$$

-------A-------

B

m_3	m_7	m_6	m_2
m_1	m_5	m_4	m_0

-------C-------

-------A-------

B

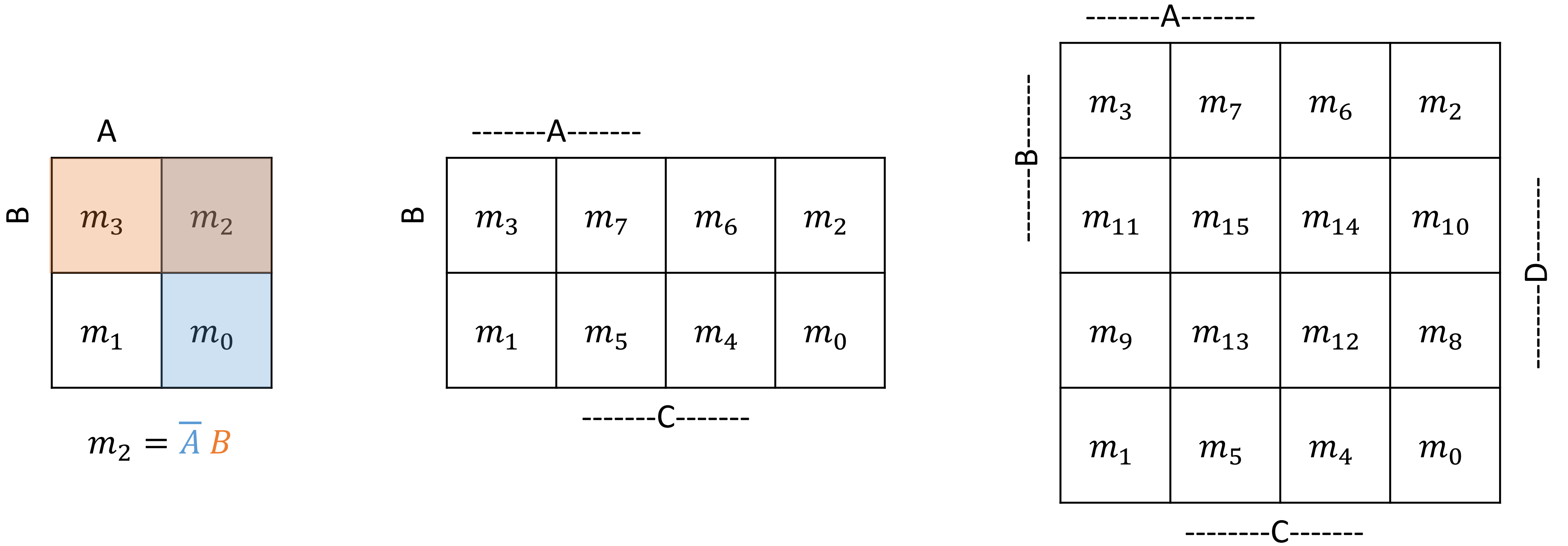
D

m_3	m_7	m_6	m_2
m_{11}	m_{15}	m_{14}	m_{10}
m_9	m_{13}	m_{12}	m_8
m_1	m_5	m_4	m_0

--------C-------

Graphical simplification

- A function is represented as union of intersections of subsets
- Veitch or Karnaugh maps



$$m_2 = \bar{A}\ B$$

Graphical simplification

- A function is represented as union of intersections of subsets
- Veitch or Karnaugh maps

	A	
B	m_3	m_2
	m_1	m_0

$m_3 = A\ B$

	-------A-------			
B	m_3	m_7	m_6	m_2
	m_1	m_5	m_4	m_0
		-------C-------		

	-------A-------				
-------B-------	m_3	m_7	m_6	m_2	
	m_{11}	m_{15}	m_{14}	m_{10}	-------D-------
	m_9	m_{13}	m_{12}	m_8	
	m_1	m_5	m_4	m_0	
		--------C-------			

Graphical simplification

- $f = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + A\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,B\,\overline{C}\,\overline{D} + A\,B\,\overline{C}\,\overline{D} + A\,\overline{B}\,C\,\overline{D} + A\,B\,C\,\overline{D} + A\,\overline{B}\,C\,D + A\,B\,C\,D$

-------A-------

m_3	m_7	m_6	m_2
m_{11}	m_{15}	m_{14}	m_{10}
m_9	m_{13}	m_{12}	m_8
m_1	m_5	m_4	m_0

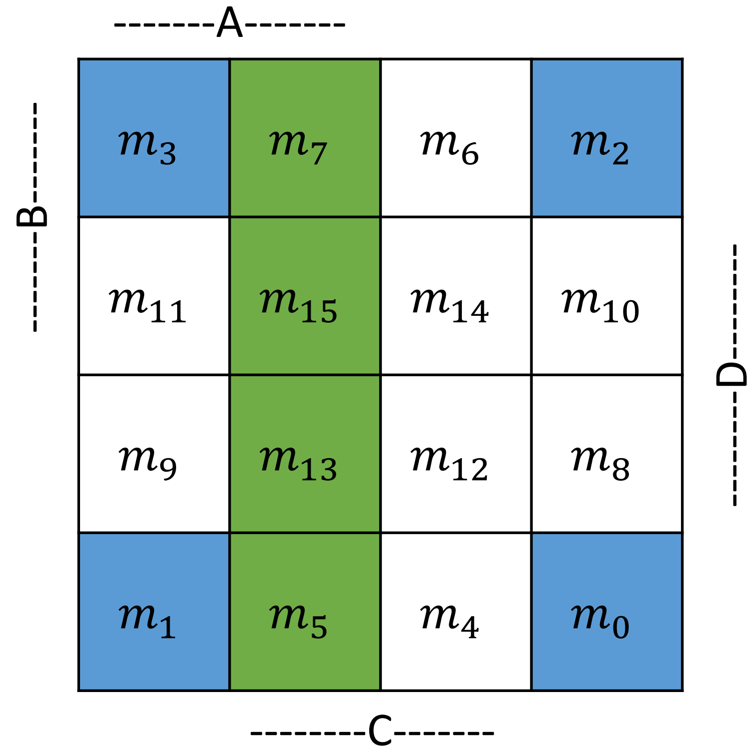
--------C-------

-------B-------

-------D-------

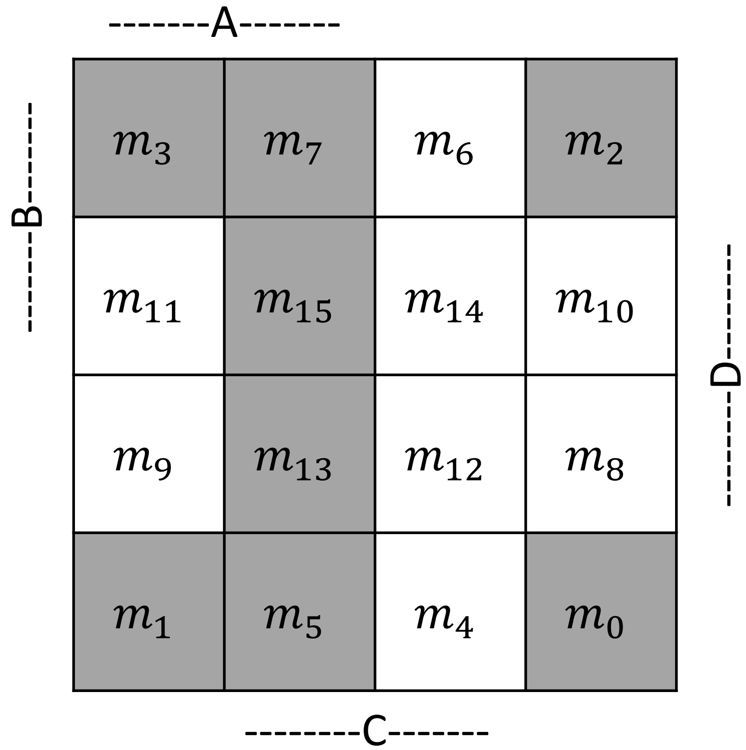
Graphical simplification

- $f = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + A\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,B\,\overline{C}\,\overline{D} + A\,B\,\overline{C}\,\overline{D} + A\,\overline{B}\,C\,\overline{D} + A\,B\,C\,\overline{D} + A\,\overline{B}\,C\,D + A\,B\,C\,D$
- $f = A\,C + \overline{C}\,\overline{D}$ -> NAND



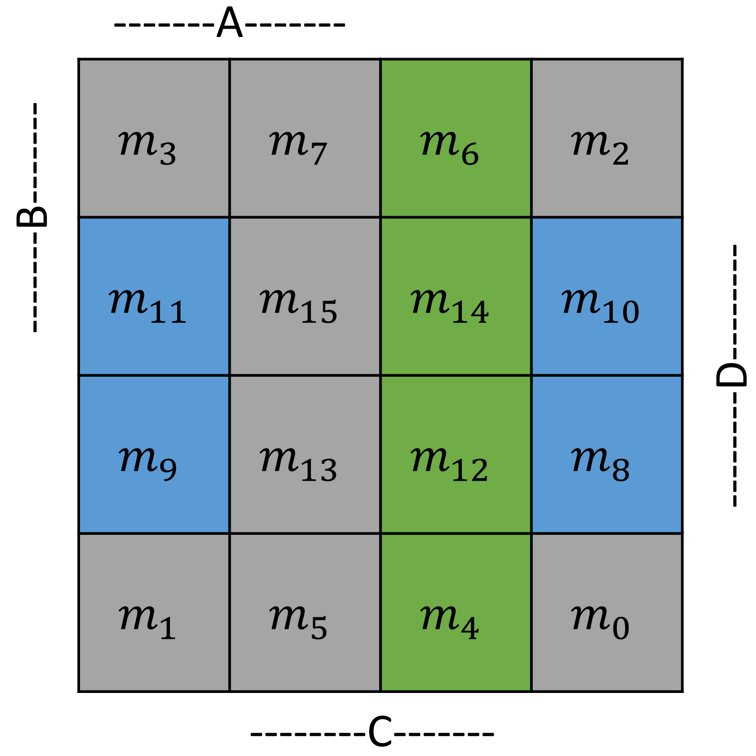
Graphical simplification

- $f = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + A\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,B\,\overline{C}\,\overline{D} + A\,B\,\overline{C}\,\overline{D} + A\,\overline{B}\,C\,\overline{D} + A\,B\,C\,\overline{D} + A\,\overline{B}\,C\,D + A\,B\,C\,D$
- $f = A\,C + \overline{C}\,\overline{D}$ -> NAND



Graphical simplification

- $f = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + A\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,B\,\overline{C}\,\overline{D} + A\,B\,\overline{C}\,\overline{D} + A\,\overline{B}\,C\,\overline{D} + A\,B\,C\,\overline{D} + A\,\overline{B}\,C\,D + A\,B\,C\,D$
- $f = A\,C + \overline{C}\,\overline{D}$ -> NAND
- $\overline{f} = \overline{A}\,C + \overline{C}\,D$
- $f = (A + \overline{C})(C + \overline{D})$ -> NOR



Exercises

1. Demonstrate that $(A \oplus B) \oplus C = A \oplus (B \oplus C) = B \oplus (A \oplus C)$

2. Simplify $AB\overline{C} + ABC + \overline{A}BC + \overline{A}B\overline{C} + A\overline{B}C + \overline{A}\,\overline{B}C$

Examples of combinatorial logic

Adder

- Elementary cell for adding two bits (binary digit)
- Extension to two n-bits numbers

A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

$$S_i = \left(A\,\overline{B}\,\overline{C} + \overline{A}\,B\,\overline{C} + \overline{A}\,\overline{B}\,C + A\,B\,C\right)_i$$

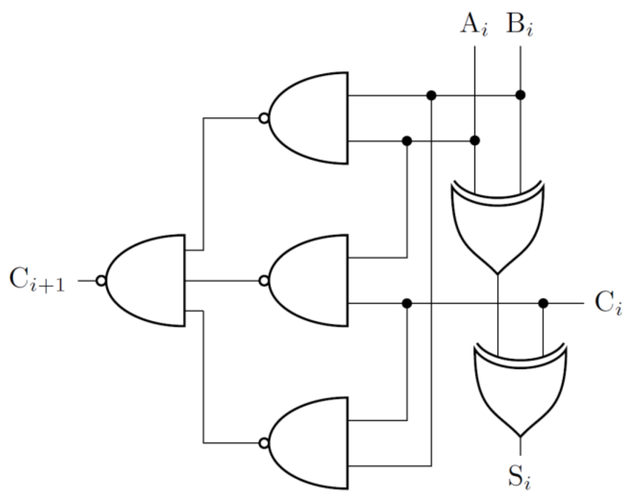
$$C_{i+1} = \left(A\,B\,\overline{C} + A\,\overline{B}\,C + \overline{A}\,B\,C + A\,B\,C\right)_i$$

$$S_i = \left(A\,\overline{B} + \overline{A}\,B\right)_i \overline{C_i} + \left(\overline{A}\,\overline{B} + A\,B\right)_i C_i = (A \oplus B)_i \oplus C_i$$

$$C_{i+1} = (A\,B + A\,C + B\,C)_i$$

Adder

- Elementary cell for adding two bits (binary digit)
- Extension to two n-bits numbers



$$S_i = \left(A\,\overline{B}\,\overline{C} + \overline{A}\,B\,\overline{C} + \overline{A}\,\overline{B}\,C + A\,B\,C\right)_i$$

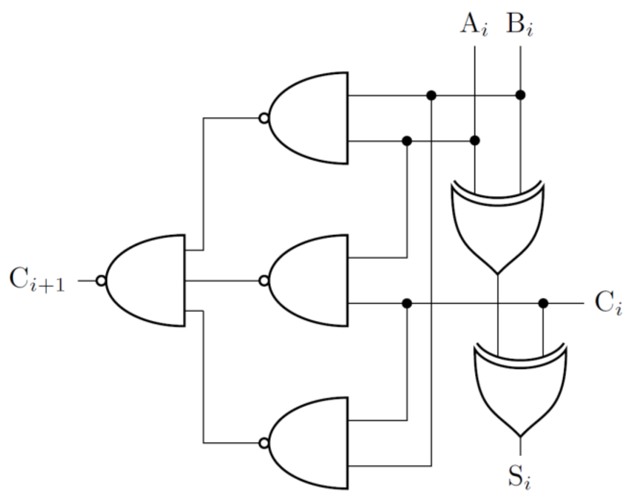
$$C_{i+1} = \left(A\,B\,\overline{C} + A\,\overline{B}\,C + \overline{A}\,B\,C + A\,B\,C\right)_i$$

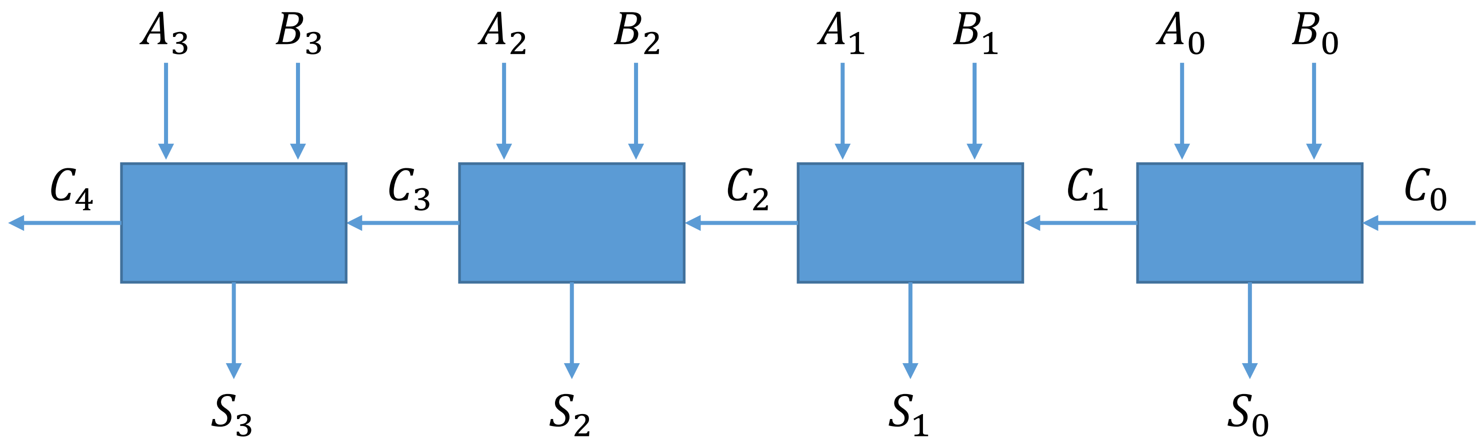
$$S_i = \left(A\,\overline{B} + \overline{A}\,B\right)_i \overline{C_i} + \left(\overline{A}\,\overline{B} + A\,B\right)_i C_i = (A \oplus B)_i \oplus C_i$$

$$C_{i+1} = (A\,B + A\,C + B\,C)_i$$

Adder

- Elementary cell for adding two bits (binary digit)
- Extension to two n-bits numbers





Subtractor

- Exercise
  - Draw the circuit
  - Can be easily obtained from an adder?

Adder

- Modular adder has a propagation delay that depends on the number of bits (n)
- Carry propagation delay: $2T_P \cdot n$
- Carry can be calculated in parallel
- Carry-lookahead
  - Generate a carry if A and B are 1: $G_i = A_i\, B_i$
  - Propagate a carry if A or B are 1: $P_i = A_i + B_i$ but also if $P_i = A_i \oplus B_i$
- $C_{i+1} = G_i + C_i\, P_i$
- $C_4 = A_3B_3 + A_2B_2(A_3 \oplus B_3) + A_1B_1(A_2 \oplus B_2)(A_3 \oplus B_3) + A_0B_0(A_1 \oplus B_1)(A_2 \oplus B_2)(A_3 \oplus B_3) + C_0(A_0 \oplus B_0)(A_1 \oplus B_1)(A_2 \oplus B_2)(A_3 \oplus B_3)$
- Parallel carry propagation delay: $3T_P$

Parity generator

- The parity bit is a bit that added to the data makes the total number of ones (1s) even (even parity) or odd (odd parity)
- Parity is used for error detection during data transmission
- For instance, 1001010 has $P_{odd} = 0$ and $P_{even} = 1$
- Elementary cell for odd parity generation
- Extension to two n-bits numbers

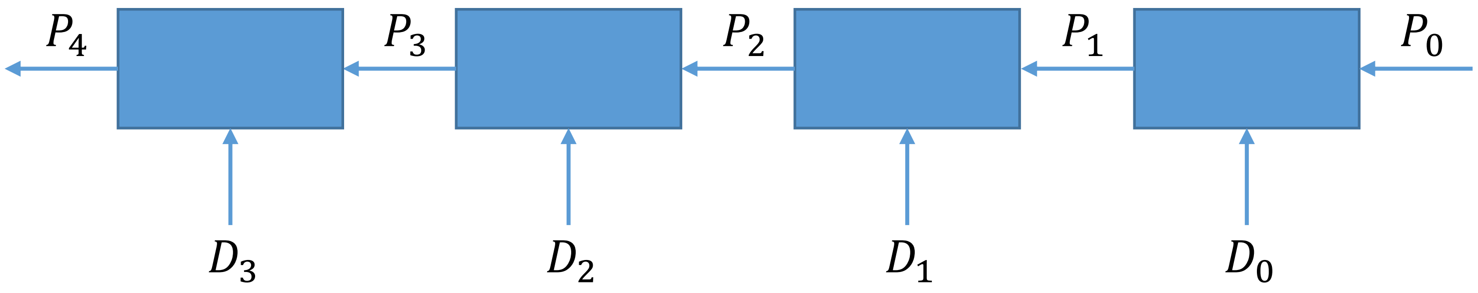
D_i	P_i	P_{i+1}
0	0	0
1	0	1
0	1	1
1	1	0

$$P_{i+1} = \left(D\,\overline{P} + \overline{D}\,P\right)_i = (D \oplus P)_i$$

$$P_0 = 1$$

Parity generator

- The parity bit is a bit that added to the data makes the total number of ones (1s) even (even parity) or odd (odd parity)
- Parity is used for error detection during data transmission
- For instance, 1001010 has $P_{odd} = 0$ and $P_{even} = 1$
- Elementary cell for odd parity generation
- Extension to two n-bits numbers

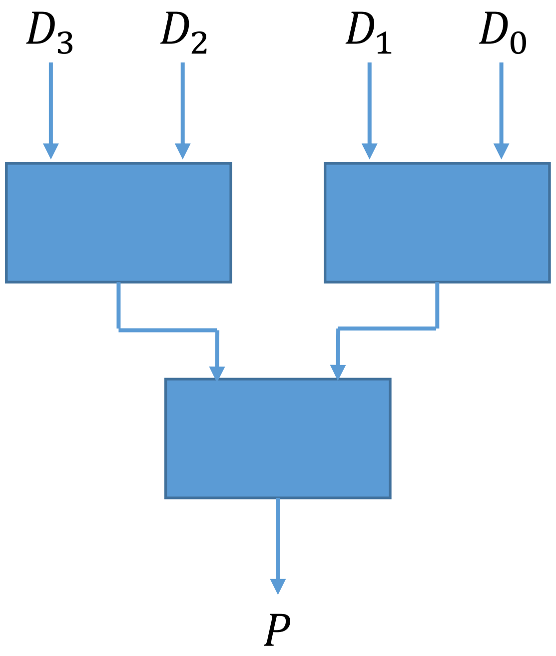


$$P_{i+1} = \left(D\,\overline{P} + \overline{D}\,P\right)_i = (D \oplus P)_i$$

$$P_0 = 1$$

Parity generator

- And for even parity generator?
  - Just change P_0
- As for the adder we can limit the propagation time with a module that works in parallel
- It is possible to connect the single module in a tree-like structure

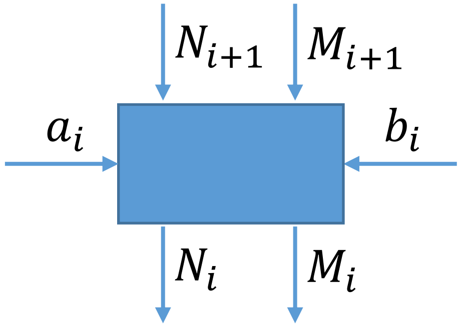


$$P = (D_0 \oplus D_1) \oplus (D_2 \oplus D_3)$$

Comparison

- For comparing two binary numbers (a, b), three output values are needed -> two output bits (N, M)
- Modular approach for n-bits extension

	N	M
$a > b$	0	1
$a < b$	1	0
$a = b$	0	0

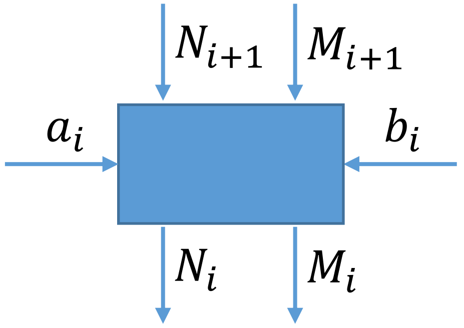


a_i	b_i	N_{i+1}	M_{i+1}	N_i	M_i
0	0	0	0	0	0
1	0	0	0	0	1
0	1	0	0	1	0
1	1	0	0	0	0
0	0	1	0	1	0
1	0	1	0	1	0
0	1	1	0	1	0
1	1	1	0	1	0
0	0	0	1	0	1
1	0	0	1	0	1
0	1	0	1	0	1
1	1	0	1	0	1
0	0	1	1	X	X
1	0	1	1	X	X
0	1	1	1	X	X
1	1	1	1	X	X

Comparison

- For comparing two binary numbers (a, b), three output values are needed -> two output bits (N, M)
- Modular approach for n-bits extension

	N	M
$a > b$	0	1
$a < b$	1	0
$a = b$	0	0



a_i	b_i	N_{i+1}	M_{i+1}	N_i	M_i
0	0	0	0	0	0
1	0	0	0	0	1
0	1	0	0	1	0
1	1	0	0	0	0
0	0	1	0	1	0
1	0	1	0	1	0
0	1	1	0	1	0
1	1	1	0	1	0
0	0	0	1	0	1
1	0	0	1	0	1
0	1	0	1	0	1
1	1	0	1	0	1
0	0	1	1	X	X
1	0	1	1	X	X
0	1	1	1	X	X
1	1	1	1	X	X

Redundant terms

Comparison

- For comparing two binary numbers (a, b), three output values are needed -> two output bits (N, M)
- Modular approach for n-bits extension

$$N_i = N_{i+1} + \overline{a_i}\, b_i\, \overline{M_{i+1}}$$

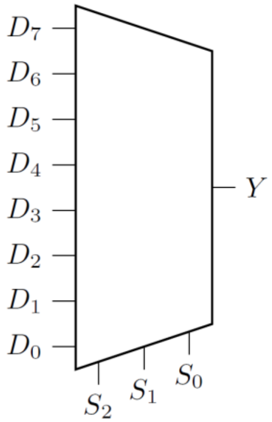
$$M_i = M_{i+1} + a_i\, \overline{b_i}\, \overline{N_{i+1}}$$

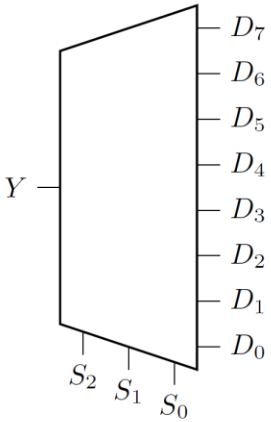
a_i	b_i	N_{i+1}	M_{i+1}	N_i	M_i
0	0	0	0	0	0
1	0	0	0	0	1
0	1	0	0	1	0
1	1	0	0	0	0
0	0	1	0	1	0
1	0	1	0	1	0
0	1	1	0	1	0
1	1	1	0	1	0
0	0	0	1	0	1
1	0	0	1	0	1
0	1	0	1	0	1
1	1	0	1	0	1
0	0	1	1	X	X
1	0	1	1	X	X
0	1	1	1	X	X
1	1	1	1	X	X

Redundant terms

Multiplexer

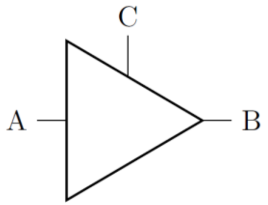
- A Mux selects one signal among 2^n (D_i) thanks to n address lines (S_j)
- For $n = 1$, two levels of NAND ports can easily implement it
- Any combinatorial function can be generated connecting the address lines to the variables and connecting 1 to D_i if the minterm m_i is present, 0 otherwise
- A Demux applies the inverse operation
- It can be used as decoder because it selects the output line D_i corresponding to the minterm m_i



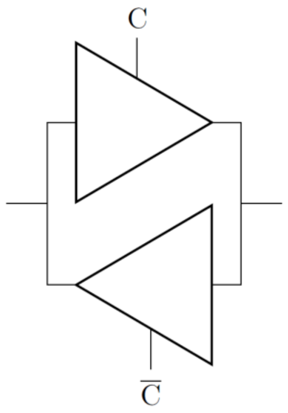


Tristate

- It is used for developing bidirectional connections
- Many data buses are usually tristate because they are used to link devices that can be both source and sink of information
- A control line is used for putting the output in high-impedance

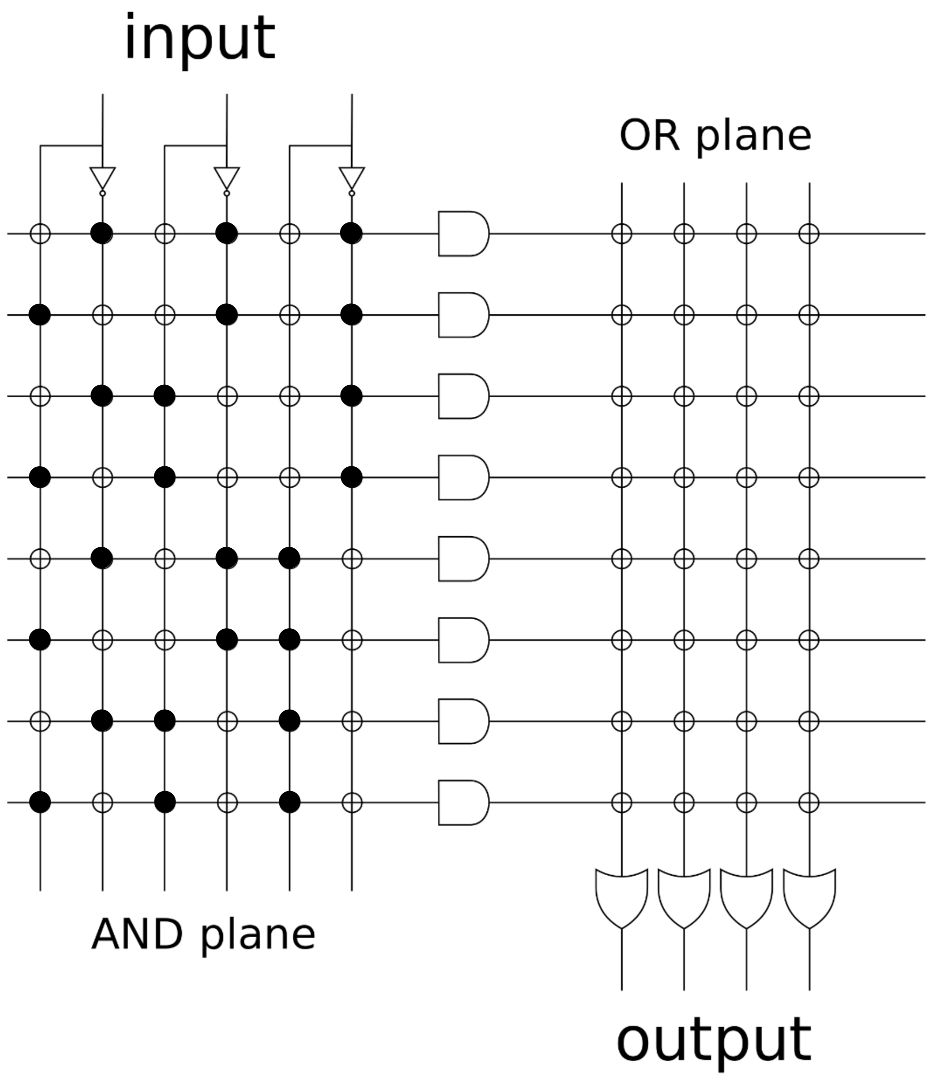


A	*C*	*B*
0	0	Z
1	0	Z
0	1	0
1	1	1



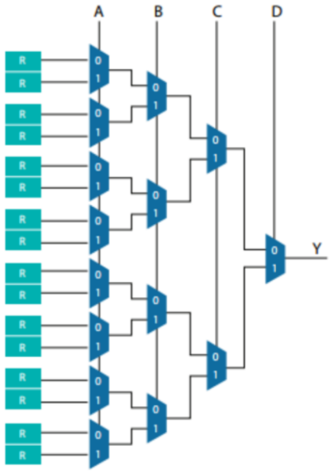
Programmable Logic Array

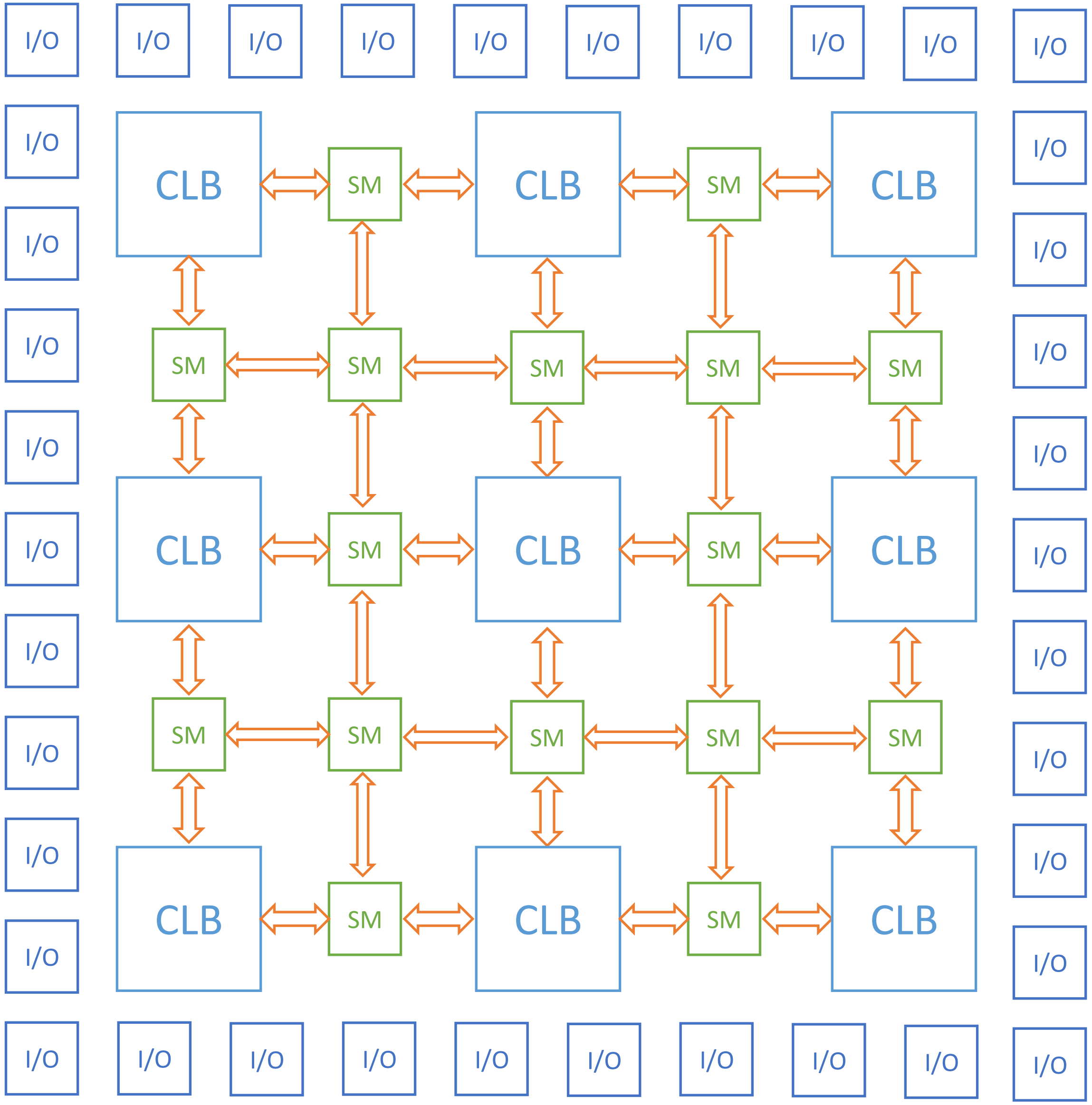
- PLA consists of an AND array connected to an OR array
- Constant propagation delay
- It can build any sum of products
- The AND array can be fixed while the OR array is programmable
- Programmable logic-planes grow too quickly in size as the number of inputs is increased



Field-Programmable Gate Array

- Programmable logic blocks linked by programmable interconnections
- Programmable Input and Output
- Configurable Logic Block can be either a combinatorial or a sequential circuit
- Look up tables are used to implement the truth table of a combinatorial function





Exercise

- Design a circuit able to compare two 2-bit numbers at a time