

# Embedded Systems

## **Hardware Technologies**

ESD\_Cap10 + Other

# Overview

- Hardware Technologies
  - Introduction
  - (AS)IC Technologies
    - Full custom
    - Standard cell
    - Gate array
  - Programmable (Configurable) Technologies
    - From ROM to CPLD
    - FPGA

# Hardware Technologies

## **Introduction**

# Introduction

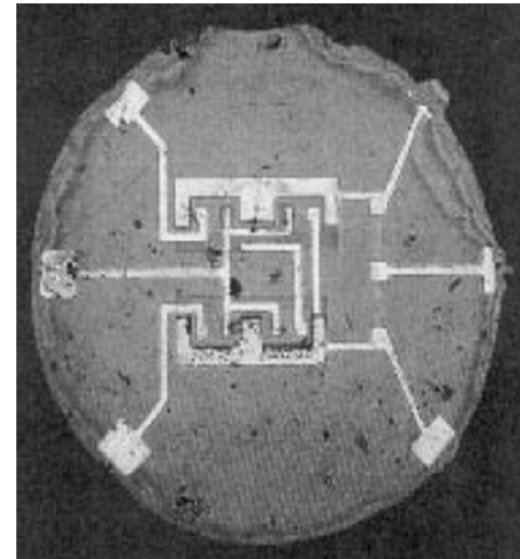
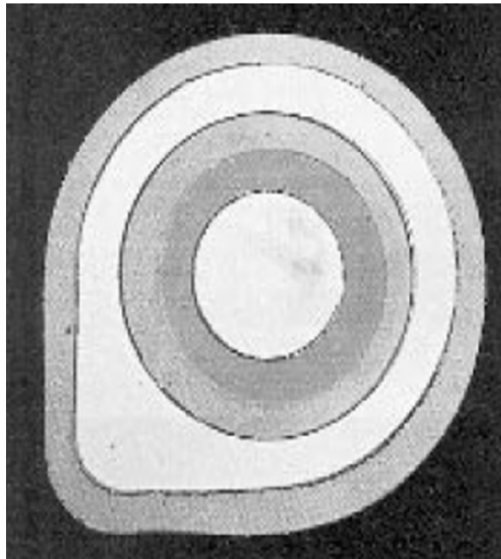
- HW technologies are the basic ones for the realization of all the physical components of an embedded systems
- At a first glance there exists a unique HW technology, i.e. the *integrated technology*
  - In fact, almost all the physical components are *Integrated Circuits*
- At the end of '50 the invention of the *planar process* has radically changed the electronic world
  - It has made possible the concurrent realization of million of elementary components on ever small silicon areas

# Hardware Technologies

**(AS)IC Technologies**

# (AS)IC Technologies

- ASIC technology has born in 1958 with the invention of the *planar process*
  - This technology has opened new frontiers in the electronic field and it is considered as the event that has started the *digital era*
    - First transistor and first integrated circuit



# (AS)IC Technologies

- Basic concepts related to the planar process
  - The planar process is based on the exploitation of a semiconductor (Ge, Si, GaAs) which physical (and so electromagnetic) characteristics are changed in different phases and in controlled areas
  - All the fundamental electronic component (transistors, diodes, resistors, capacitors) can be realized by means of such a semiconductor properly treated
  - The interconnections between the different components are instead realized by means of metals (Al or Cu)

# (AS)IC Technologies

- Basic elements related to the planar process (Si-based)
  - n Silicon
    - Silicon doped in order to augment the number of base band electrons
      - Diffusion or ionic implantation
  - p Silicon
    - Silicon doped in order to decrease the number of base band electrons
  - Insulation
    - Silicon oxide ( $\text{SiO}_2$ ) obtained by means of exposition to oxygen
  - Conductor
    - Deposition of Al or Cu

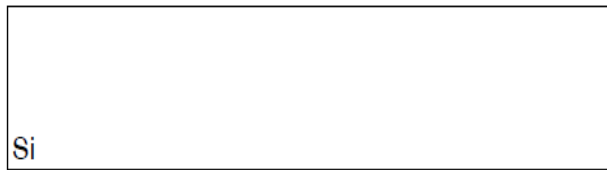


# (AS)IC Technologies

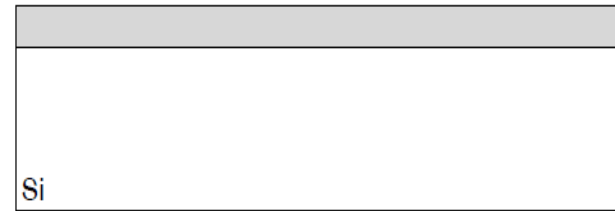
- Each one of these elements is realized during a step of the planar process
  - Big advantage: all the zones of the same kind are realized concurrently on the whole silicon wafer
    - For each step is needed a sequence of similar phases, called **maskings**
  - The goal of such phases is to prepare the silicon so that the specific process (diffusion, implantation, oxidation, etc.) would happen only in selected zones

# (AS)IC Technologies

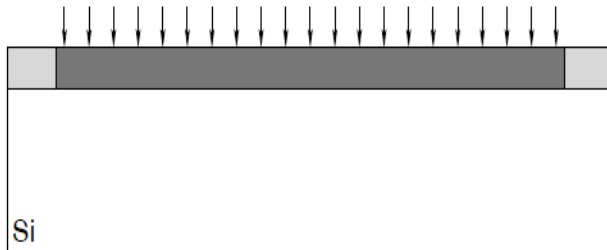
- Planar process: main phases (1)



a. Wafer



b. | Photoresist



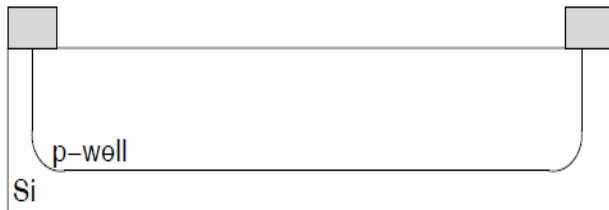
c. Masking



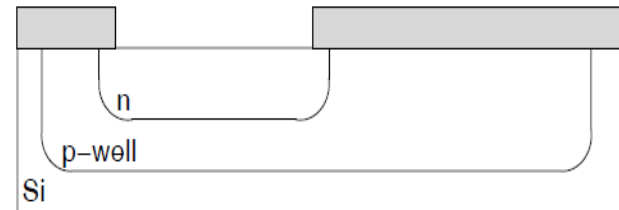
d. Etching/Washing

# (AS)IC Technologies

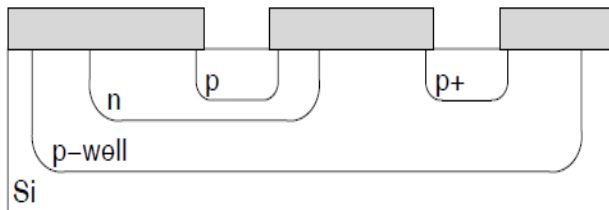
- Planar process: main phases (2)



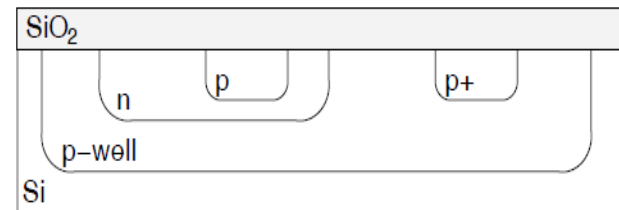
e. P-well



f. Base



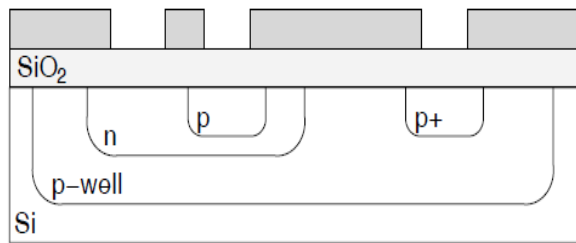
g. Collector and Emitter



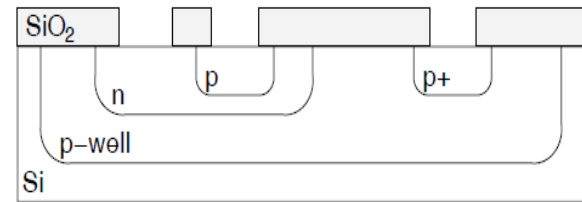
h. Oxidation

# (AS)IC Technologies

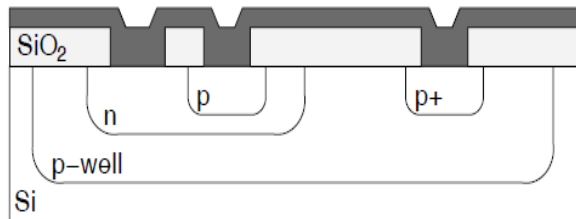
- Planar process: main phases (3)



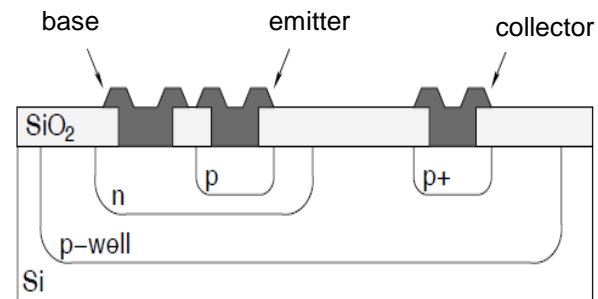
i. Masking



l. Etching/Washing



m. Conductor



n. pnp Transistor (BJT)

# (AS)IC Technologies

- Then the interconnections are realized
  - Since it is not possible to isolate the single lines (net), there is the need for different layers separated by insulating layers
    - *Metal layers*
- The described process is related to BJT transistors replaced by CMOS
  - Different steps but similar techniques

# (AS)IC Technologies

- With the CMOS planar process it is possible to realize generic circuital structures with respect to transistors dimension, position, shape and so on...
  - In the design of small analog/digital circuits all this flexibility and freedom in the organization of basic components over the silicon is a feature very useful
    - This is called **full custom** design

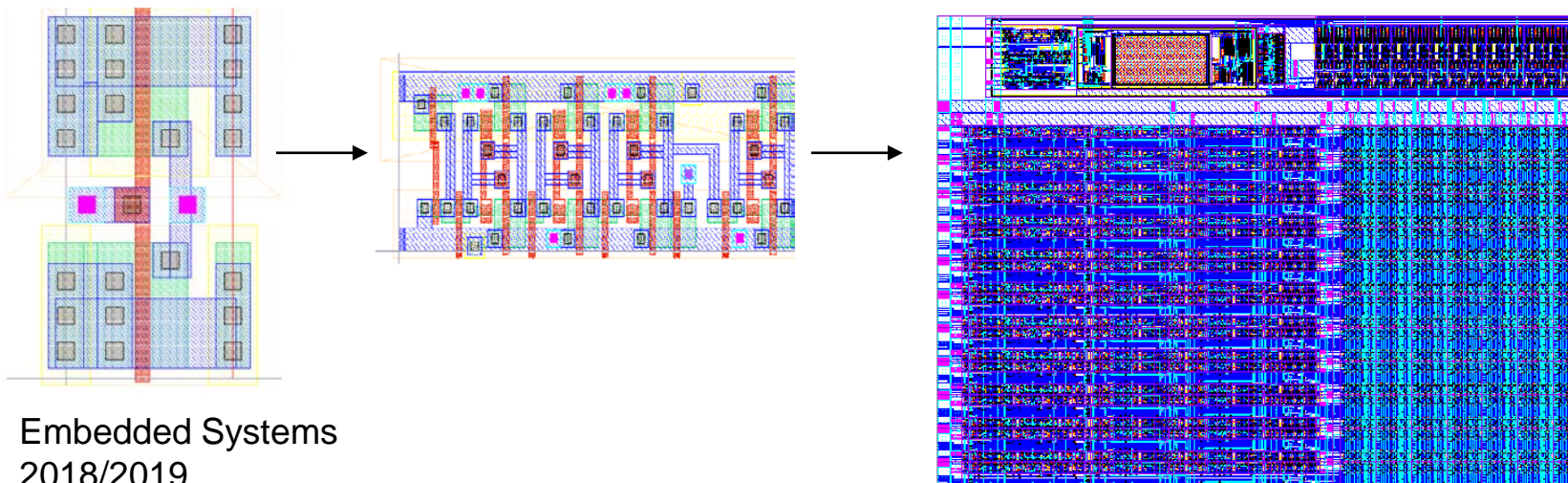
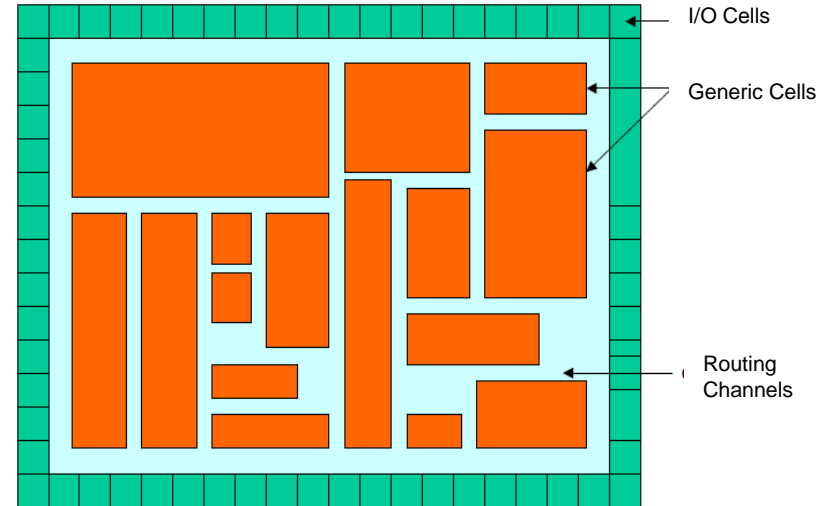
# (AS)IC Technologies

- However, when the design is oriented to medium-big digital circuits, it is too complex to exploit such a feature
  - There is the need to reduce the degrees of freedom by using more regular and less flexible structures
    - This is called ***standard cells design***
- Finally, it is also possible to realize generic digital circuits by using very regular structures
  - This is called ***gate array*** or ***sea of gates design***
    - By following this approach all the transistors are pre-fabricated in fixed places while leaving their terminals disconnected
      - The designer has to “only” to connect them by means of the final phases of the planar process (metallization)

# (AS)IC Technologies

- *Full Custom*

- Hand-made geometry
- Digital and analogic circuits
- Transistor-level simulation
- High density
- High performance
- Very long design time





# (AS)IC Technologies

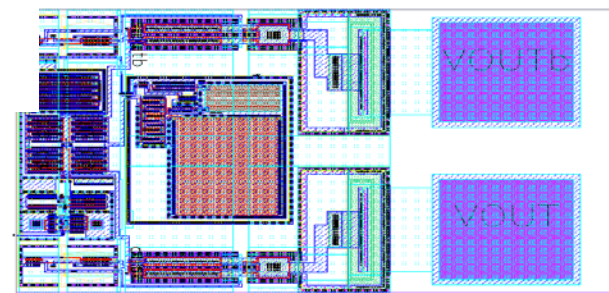
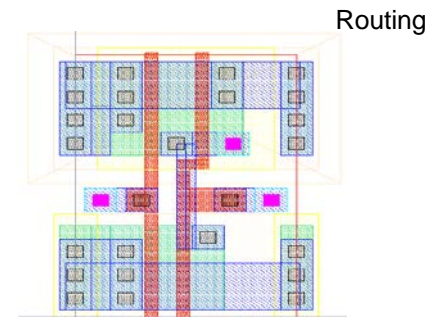
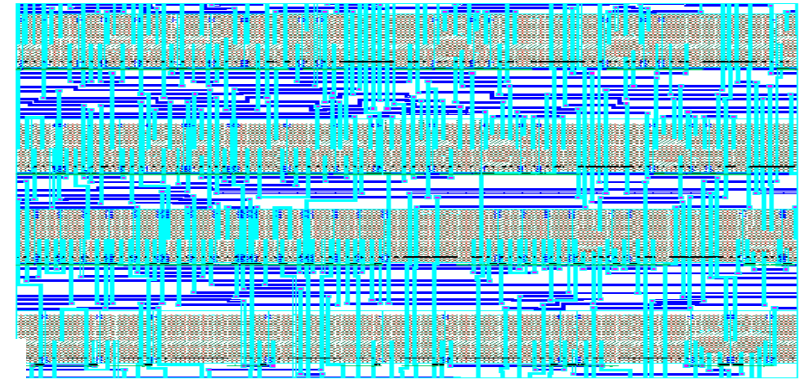
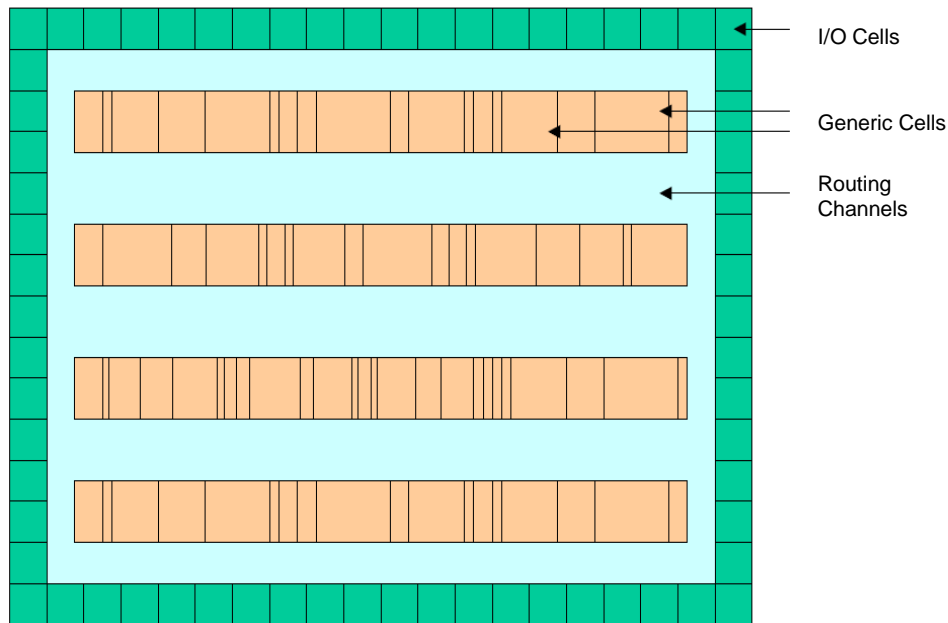
- *Standard Cells*
  - Higher level of abstraction (i.e. less details and DoF)
    - A cell is a small complete and optimized digital circuit that the designer can use ignoring a lot of internal details
      - Custom technology libraries form different *silicon vendors*
        - » 500-2000 kinds of cells that realize a wide set of functions: basic and complex combinatorial circuits, common sequential circuits, memories
  - Since all the cells have the same height, placement and routing are extremely simplified by binding the cell to pre-defined positions
    - Routing is normally based on a grid of nets
      - *Manhattan geometry*

# (AS)IC Technologies

- *Standard Cells*
  - From the point of view of the designer, a cell is characterized by several informations
    - Logic name or reference to identify it
    - Functional model
    - Non functional properties
    - Number, name and pin positions
    - Symmetry axes (*flip*)
    - Rotation angles (*orientation*)
  - Thanks to such information, it is possible to perform all the phases needed for final realization

# (AS)IC Technologies

- *Standard Cells*



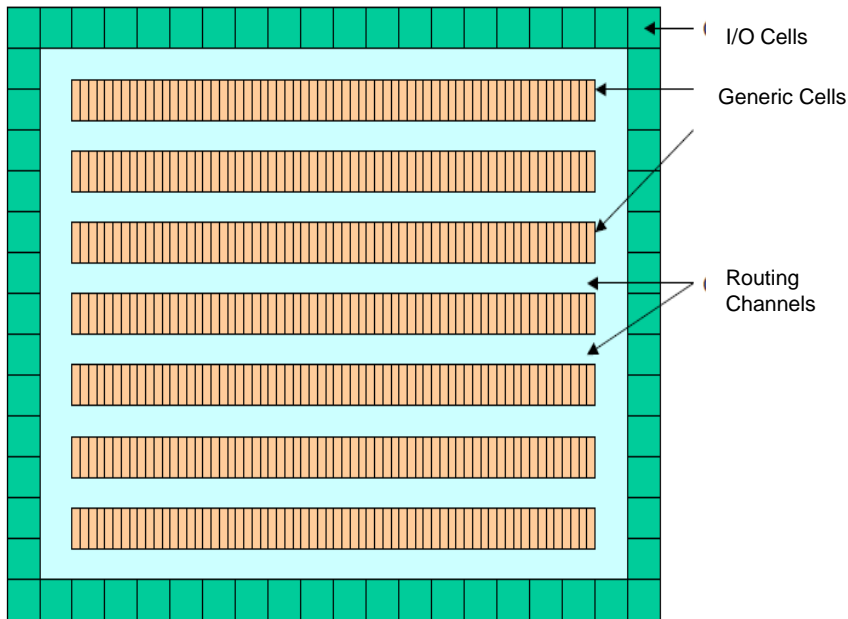
I/O Cell

# (AS)IC Technologies

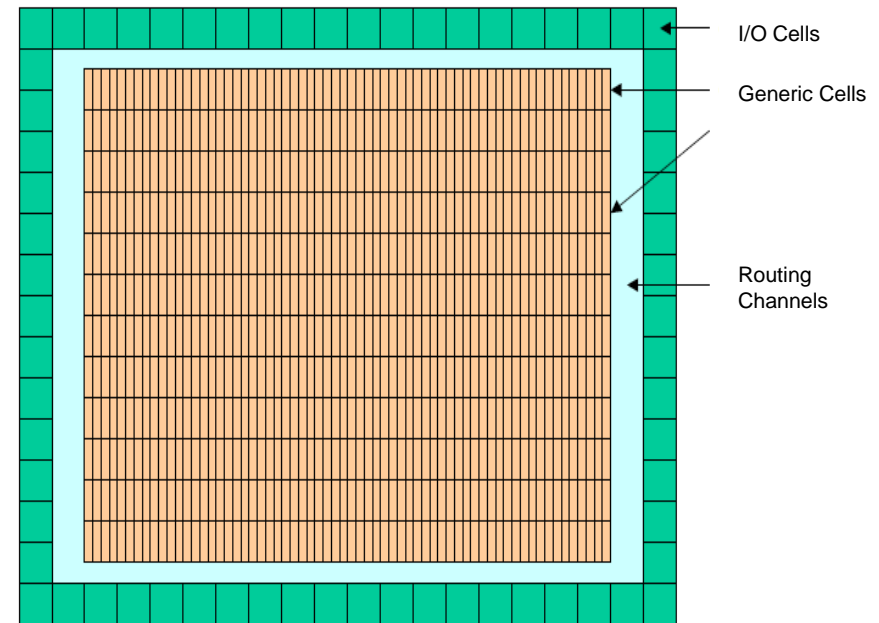
- *Gate Array (o Sea of Gate)*
  - It is another cell-based approach
    - The cells are transistors or NAND ports
      - They are already deployed on the device
      - Their position is completely fixed
  - A new design needs only the masks useful to connect cells in the desired way
    - Also called *Mask Programmable Gate Array* (MPGA)

# (AS)IC Technologies

- *Gate Array (o Sea of Gate)*



Channel based



Sea of gates

# Considerations

- *Have you noted that...*
  - There is always the need of some masking: we need a foundry!
    - Don't worry, we are not going to realize (AS)ICs! ☺
  - The design activity seems to be completely “structural”
    - A designer has to identify the basic components needed to implement a functionality, place (if not fixed) and connect them
      - The goal is to build a structure that implements the system behaviour
  - Unfortunately, when the system behaviour tends to be more and more complex such an approach tends to be no more feasible
    - We need even more complex components or...
    - The designer should be able to express directly the **system behaviour** and to rely on some SW tools that, on the base of a set of available components, are able to generate the proper **system structure**
      - » This process is called *Synthesis*

# Hardware Technologies

**Programmable (Configurable)  
Technologies**

# Programmable Technologies

- They are HW devices that provide more or less complex components that could be connected as needed
  - “Logic” Components
    - Logic gates, Flip-flop, Buffers...
  - Connections (Nets)
- There are different kinds of “programmable” devices
  - ROM, PLA, PAL
  - GAL, CPLD
  - FPGA



# Programmable Technologies

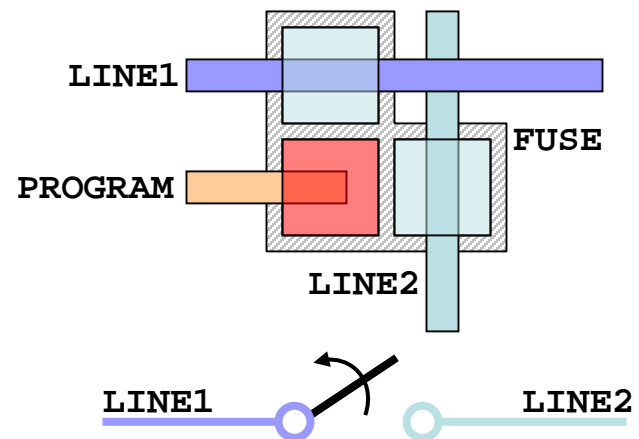
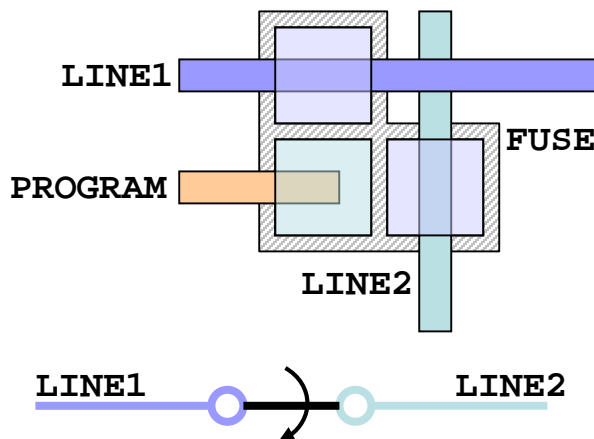
- Such devices can be classified on the base of 2 aspects
  - *“Programming” Modes*
    - *One-Time Programmable (OTP)*
      - Fuse, Antifuse
    - *Reprogrammable*
      - E<sup>2</sup>PROM, SRAM, Flash
    - *Reprogrammable & Reconfigurable*
      - SRAM, Flash
  - *Connections*
    - Globals (i.e. long)
    - Locals (i.e. short) and distributed

# Programmable Technologies

- Programming Modes

- Fuse (*OTP*)

- Nets are realized to be always connected
    - Programming means to burn some connections in order to keep only the needed ones
      - The programming is performed by means of a voltage higher than the operative one

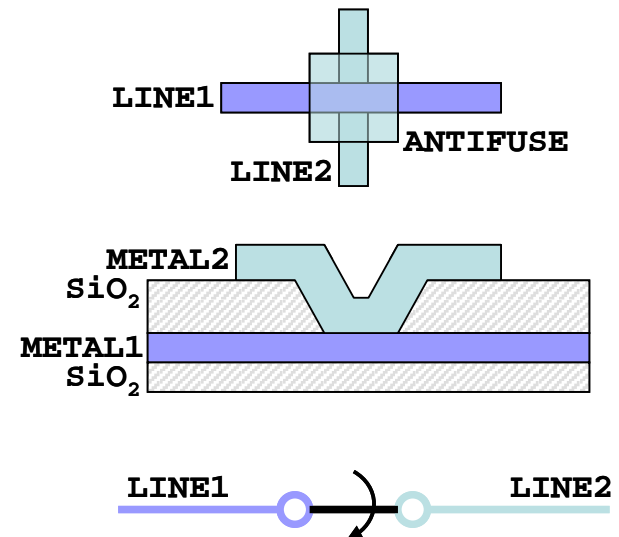
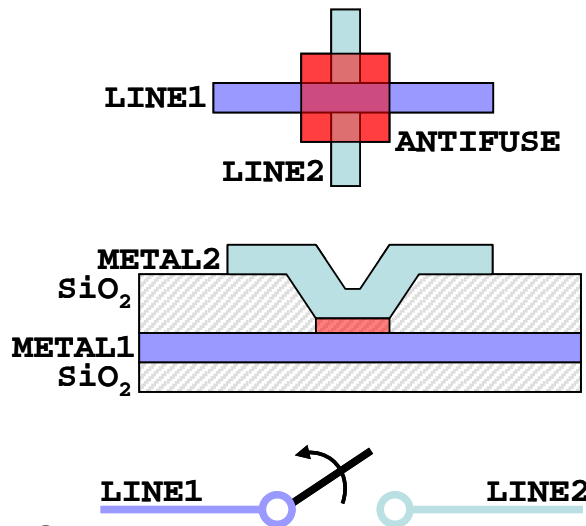


# Programmable Technologies

- Programming Modes

- Antifuse (*OTP*)

- Nets are realized to be always disconnected
    - Programming means to create the needed connections
      - The programming is performed by means of a voltage higher than the operative one

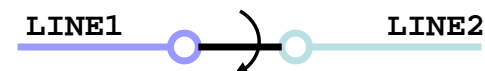
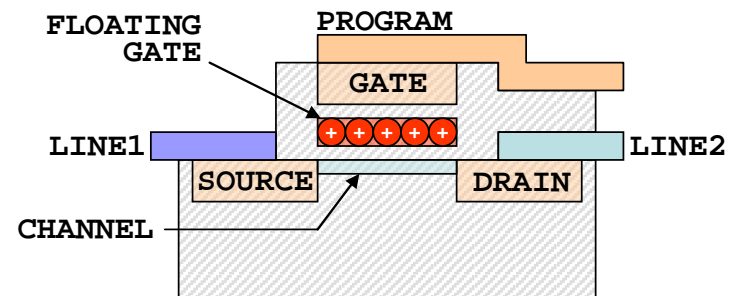
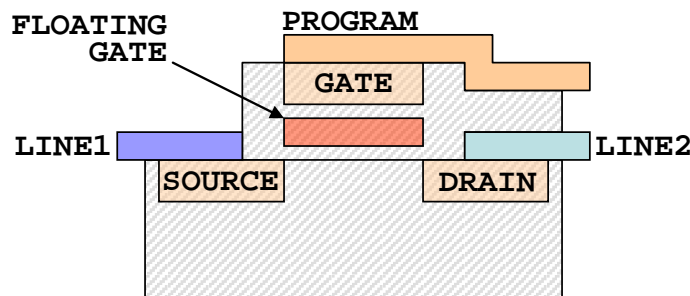


# Programmable Technologies

- Programming Modes

- E<sup>2</sup>PROM (*Reprogrammable*)

- Connections are not active at start-up. They can be electrically activated/deactivated in a non destructive way when the device is **not operative**
- The programming is performed by means of a voltage higher than the operative one in order to deposit a charge on the *floating gate* to keep it in conductive mode

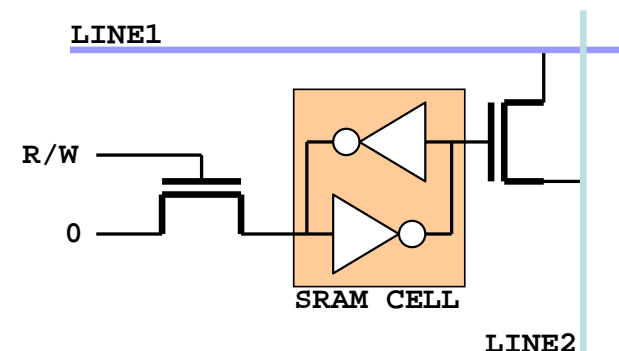
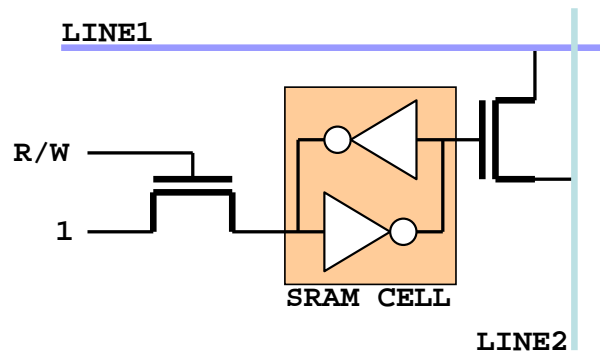


# Programmable Technologies

- Programming Modes

- SRAM (*Reprogrammable & Reconfigurable*)

- Connections are not active at start-up. They can be electrically activated/deactivated in a non destructive way
      - The programming is performed by writing a 0/1 value in a SRAM cell. Depending on the adopted technology this could be done
        - » when the device is not operative (*Reprogrammable*)
        - » when the device is operative (*Reconfigurable*)

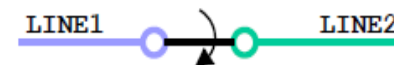
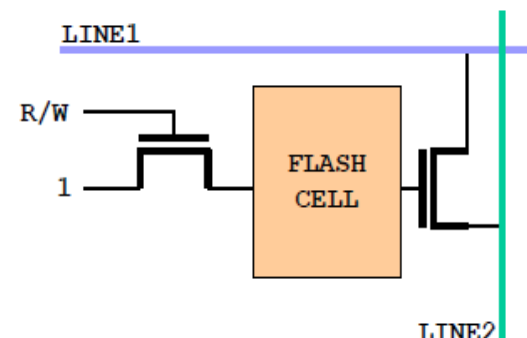
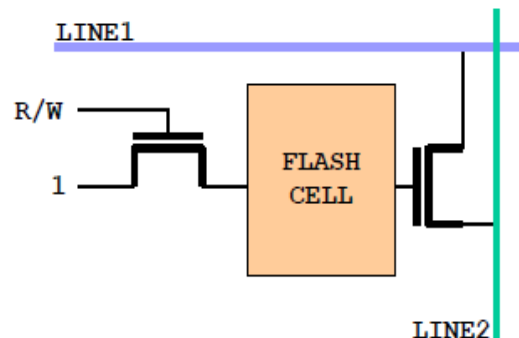


# Programmable Technologies

- Programming Modes

- Flash (*Reprogrammable & Reconfigurable*)

- Connections are not active at start-up. They can be electrically activated/deactivated in a non destructive way
      - The programming is performed by writing a 0/1 value in a Flash memory cell (during operativity or not)
        - » Not volatile

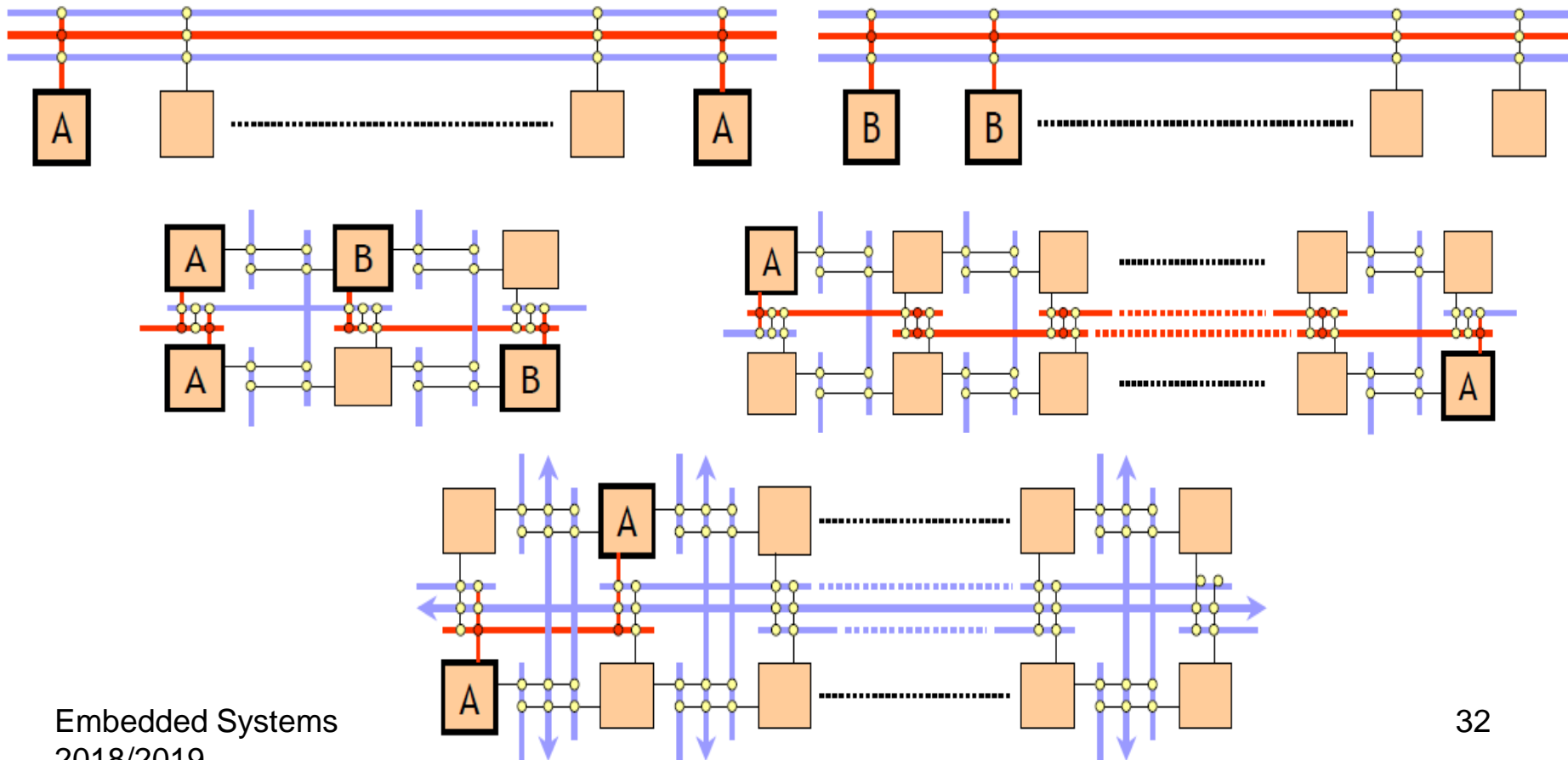


# Programmable Technologies

- Connections
  - Global connections
    - They are nets that cross almost all the device and that are shared by a lot of logic components
      - High delays
      - They can be used as output by **only one** device: limited flexibility
  - Local connections
    - They are nets that cross only a reduced part of the device and that are shared by few logic components
      - Low delays
      - In the same device it is possible to have several local nets with different lengths
        - » High flexibility but high routing complexity
  - Hierarchical connections

# Programmable Technologies

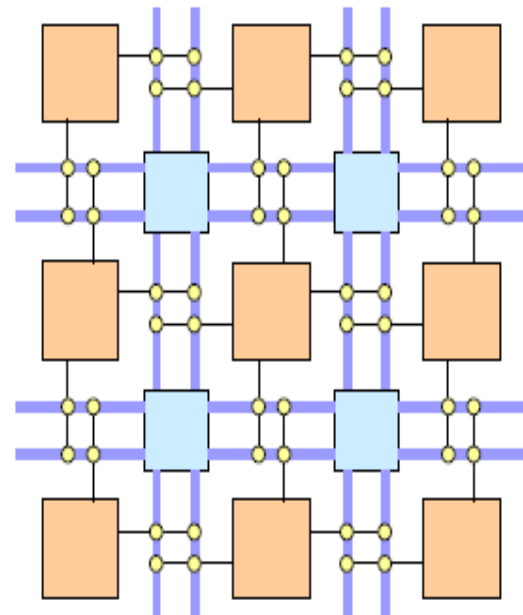
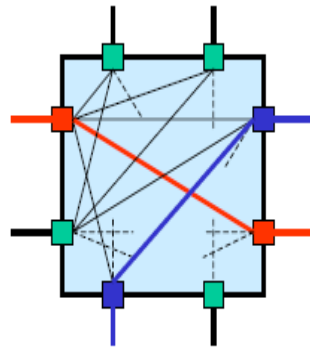
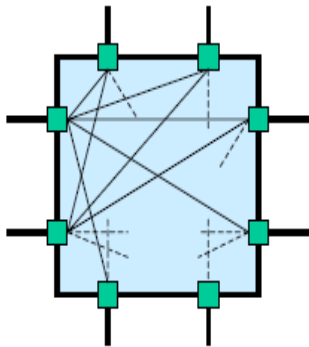
- Connections





# Programmable Technologies

- Connections
  - Other kinds
    - *Programmable Switch Matrix*



# Programmable Technologies

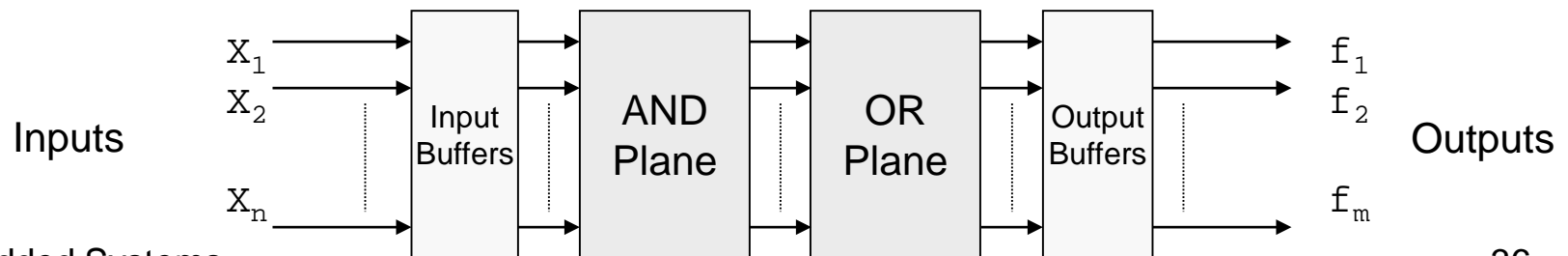
- Connections
  - Global connections
    - 2-Levels Programmable Logic Devices
      - Read-Only Memory (ROM/PROM)
      - Programmable Logic Array (PAL)
      - Programmable Array Logic (PLA)
    - Advanced PAL and PLA
    - Generic Array Logic (GAL)
    - Complex Programmable Logic Devices (CPLD)
  - Local and hierarchical (i.e. short/long) distributed connections
    - Field Programmable Gate Array (FPGA)

# Programmable Technologies

## **2-Levels Programmable Logic Devices**

# 2-Levels PLD

- Used to realize generic combinatorial functions with  $n$  inputs and  $m$  outputs
  - $f_i = f_i(x_1, x_2, \dots, x_n)$  con  $i = \{1, 2, \dots, m\}$ 
    - It is possible to realize multi-level functions or sequential functions by means of feedback loops and storage elements
  - They have
    - A fixed number of inputs
    - A fixed number of outputs
    - An AND plane to build min-terms
    - An OR plane to sum min-terms
    - I/O buffers

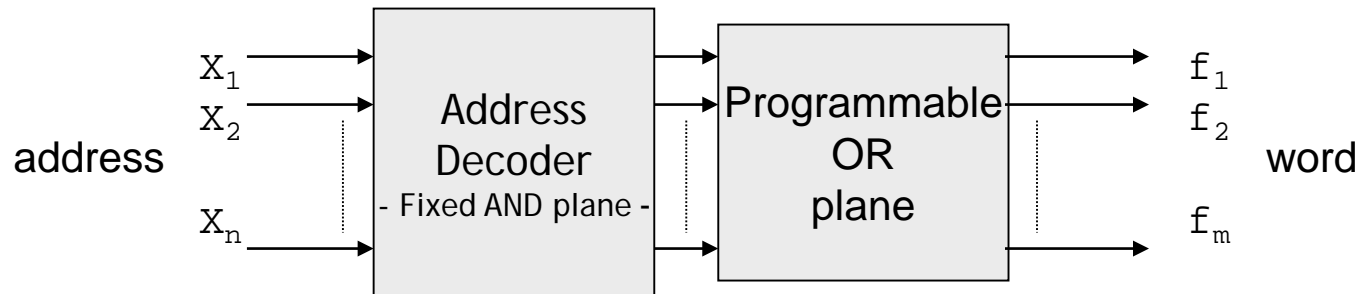


# 2-Levels PLD

- The main kinds of 2L-PLD
  - *Read-Only Memory (ROM/PROM)*
    - Fixed AND plane
      - Realizes all the possible min-terms (decoder)
    - Programmable OR plane
  - *Programmable Logic Array (PLA)*
    - Programmable AND plane
      - Realizes only the needed min-terms
    - Programmable OR plane
  - *Programmable Array Logic (PAL)*
    - Programmable AND plane
      - Realizes only the needed min-terms
    - Fixed OR plane
      - Imposes a constraint to the product terms

# 2-Levels PLD

- Read-Only Memory (ROM/PROM)
  - Implements the *sum of products* (SOP)
    - Any input configuration (*address*) is related to an output configuration (*word*)

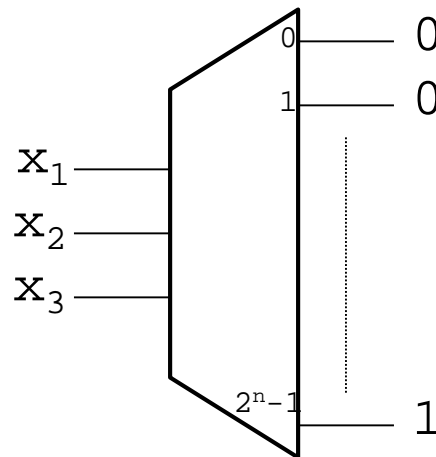


# 2-Levels PLD

- Read-Only Memory (ROM/PROM)
  - *Address decoder*
    - The ROM address decoder realizes all the  $2^n$  min-terms
      - $n$  input variables  $x_i$
  - One output active at a time
    - Outputs correspond to the min-terms built by means of the inputs

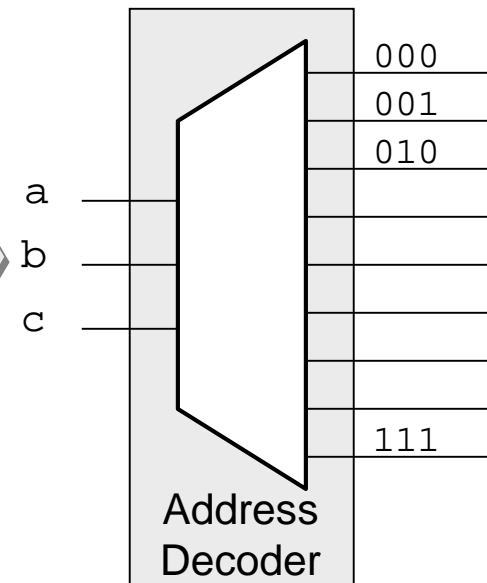
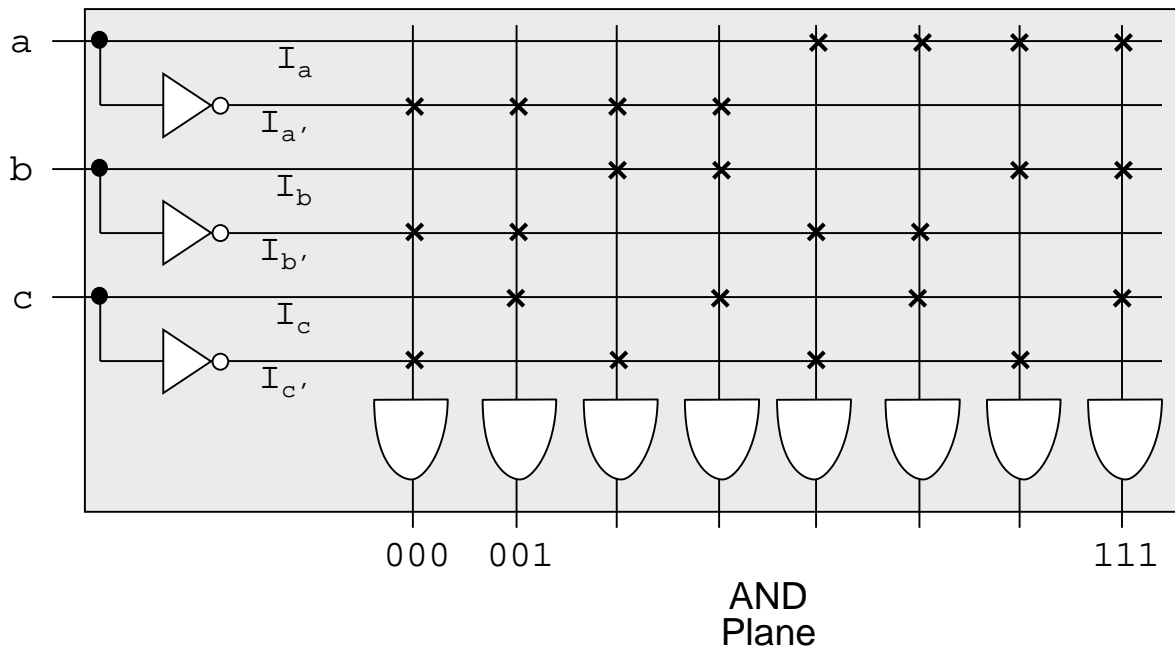
Example:

$$x_1 x_2 x_3 = 111$$



# 2-Levels PLD

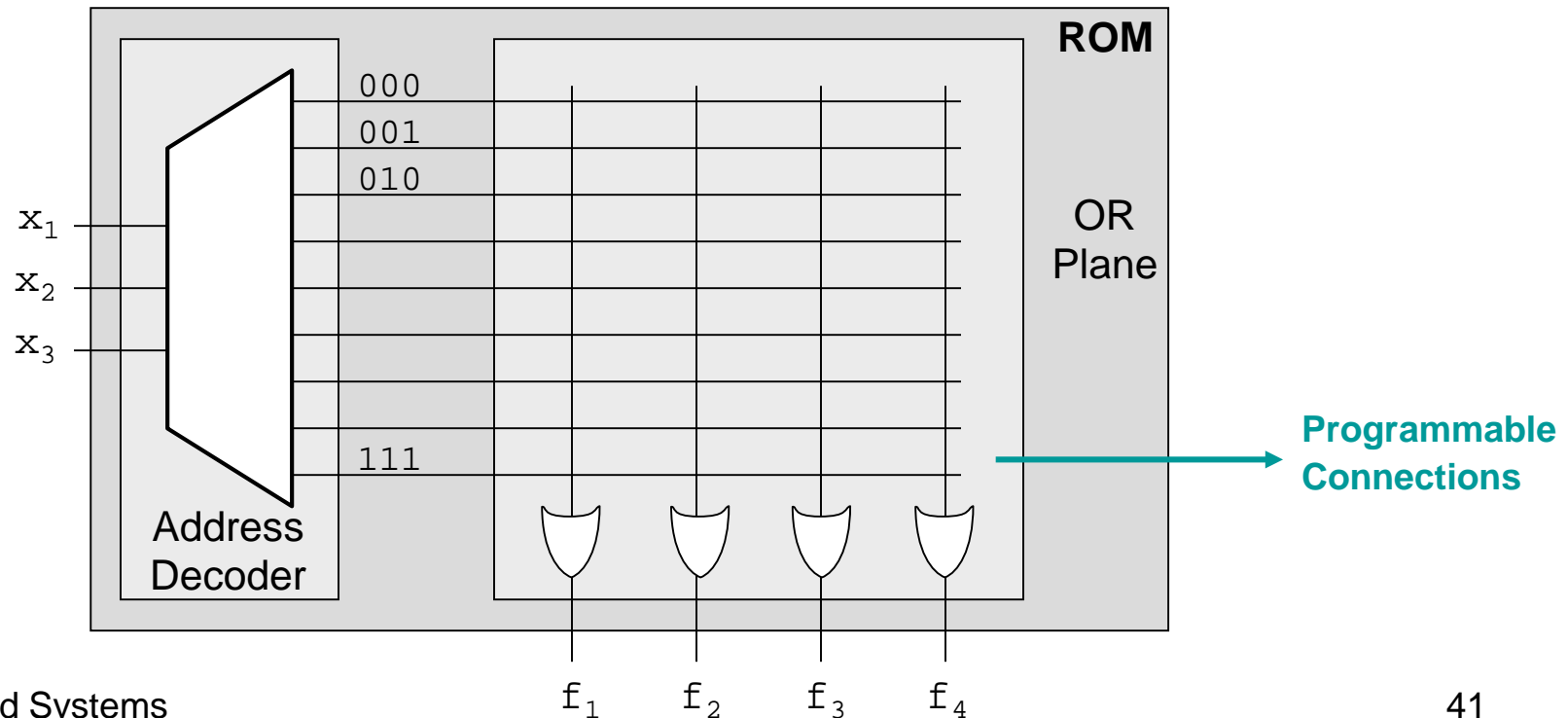
- Read-Only Memory (ROM/PROM)
  - AND plane logic schematic (*Address Decoder*)
    - For simplicity it is used the *decoder* representation
      - 1-hot code outputs





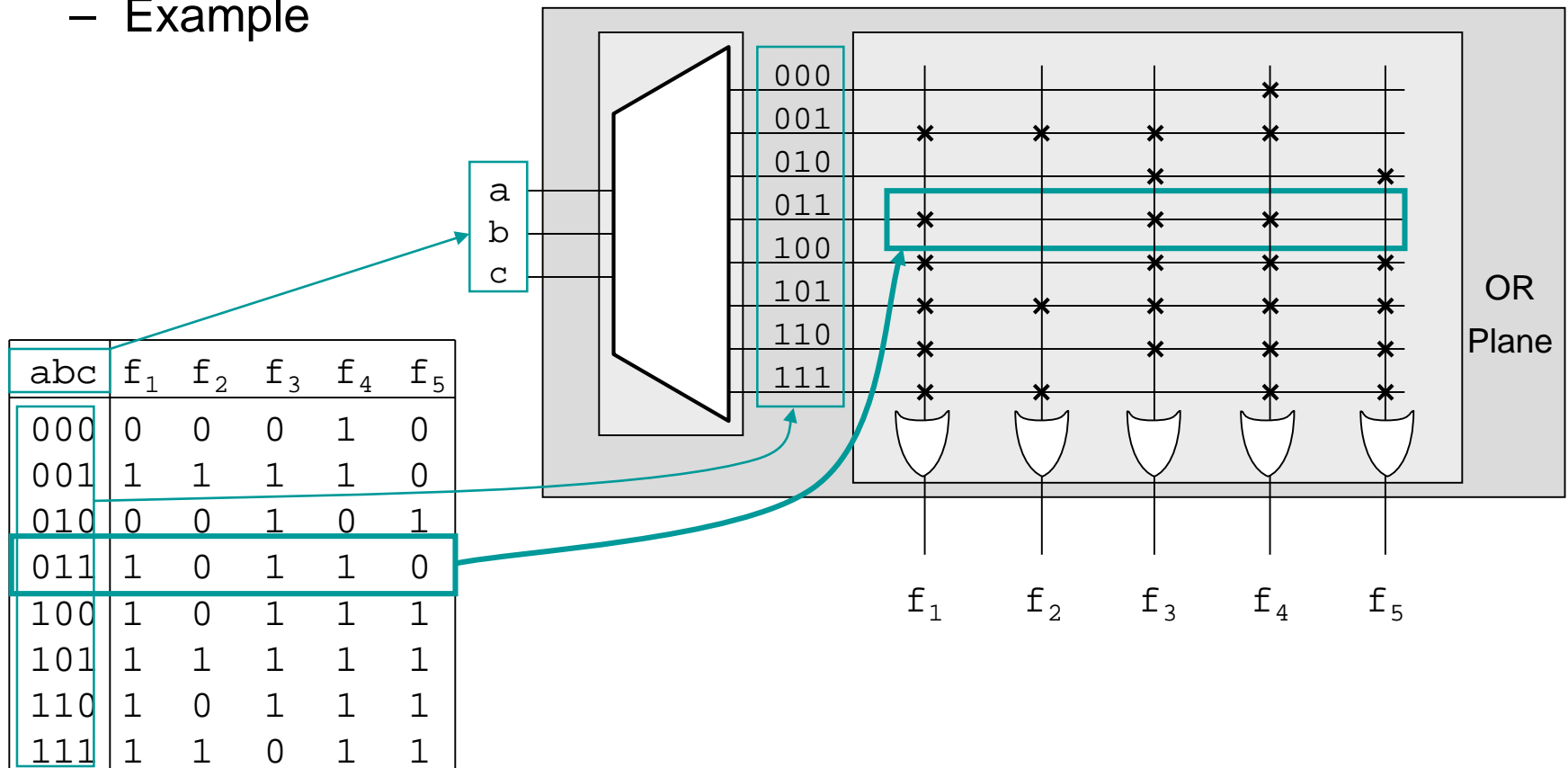
# 2-Levels PLD

- Read-Only Memory (ROM/PROM)
  - ROM logic schematic
    - Example: 3-inputs 4-outputs ROM (not programmed)



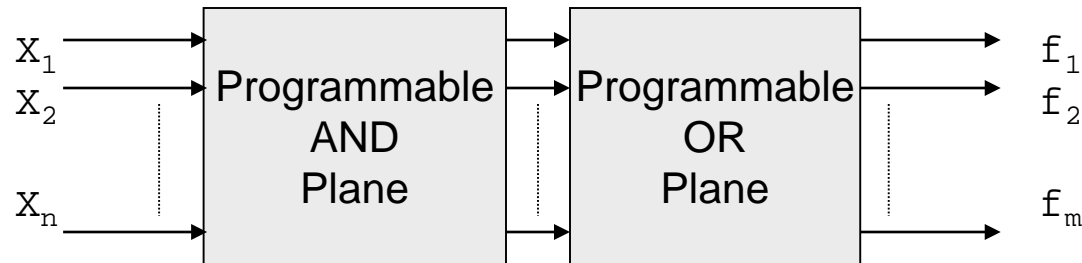
# 2-Levels PLD

- Read-Only Memory (ROM/PROM)
  - Example



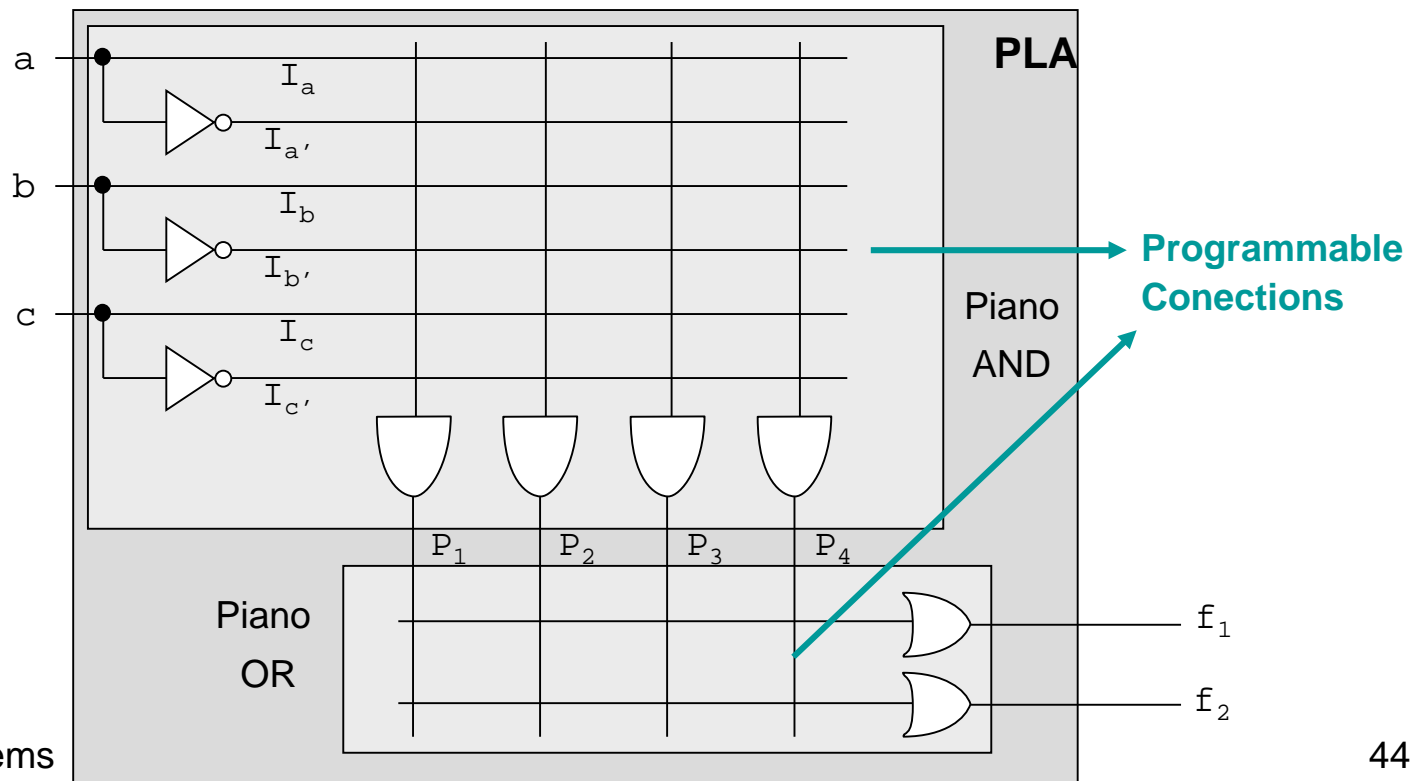
# 2-Levels PLD

- Programmable Logic Array (PLA)
  - A PLA can implement a sum of products



# 2-Levels PLD

- Programmable Logic Array (PLA)
  - PLA schematic
    - 3-inputs 2-outputs PLA (not programmed)



# 2-Levels PLD

- Programmable Logic Array (PLA)

- Example (1/2)

- Functions to be realized

- $f1 = ab + ac' + a'b'c$

- $f2 = ab + ac + a'b'c$

- Products

- $P1 = ab$

- $P2 = ac$

- $P3 = ac'$

- $P4 = a'b'c$

- Sums

- $f1 = P1 + P3 + P4$

- $f2 = P1 + P2 + P4$



## PLA programming

11– 10

1–0 10

001 10

11– 01

1–1 01

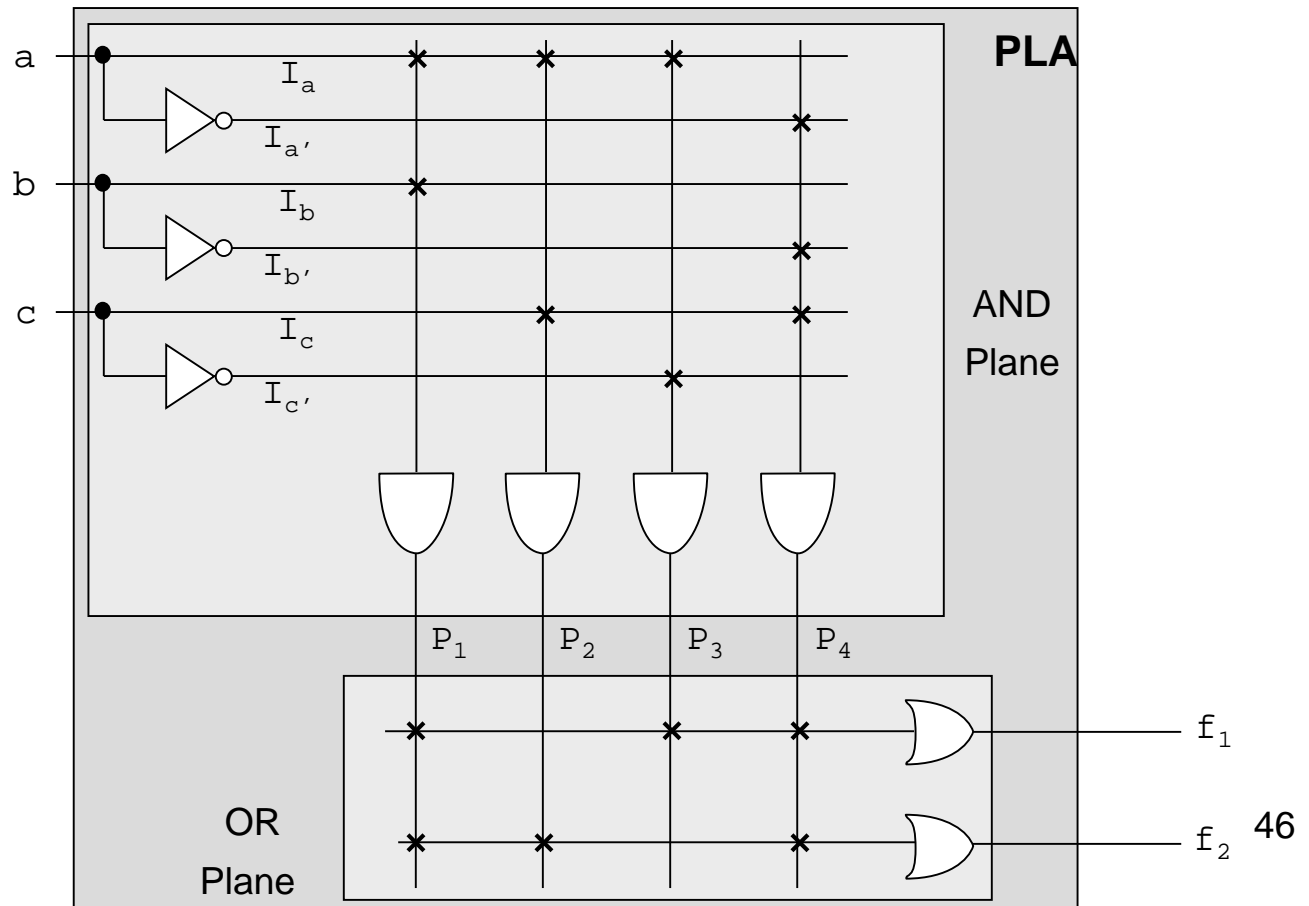
001 01

# 2-Levels PLD

- Programmable Logic Array (PLA)
  - Example (2/2)

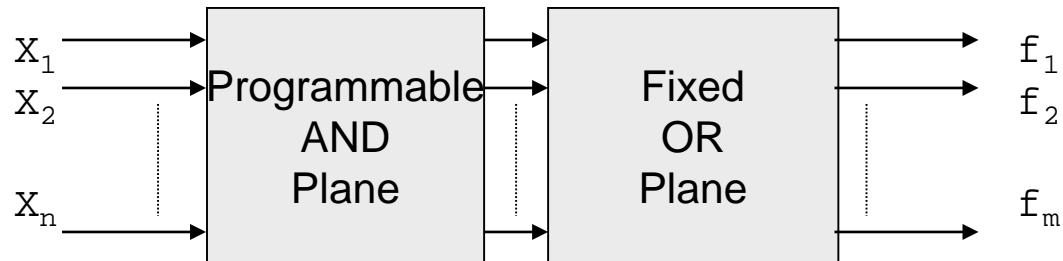
Formato PLA

11-	10
1-0	10
001	10
11-	01
1-1	01
001	01



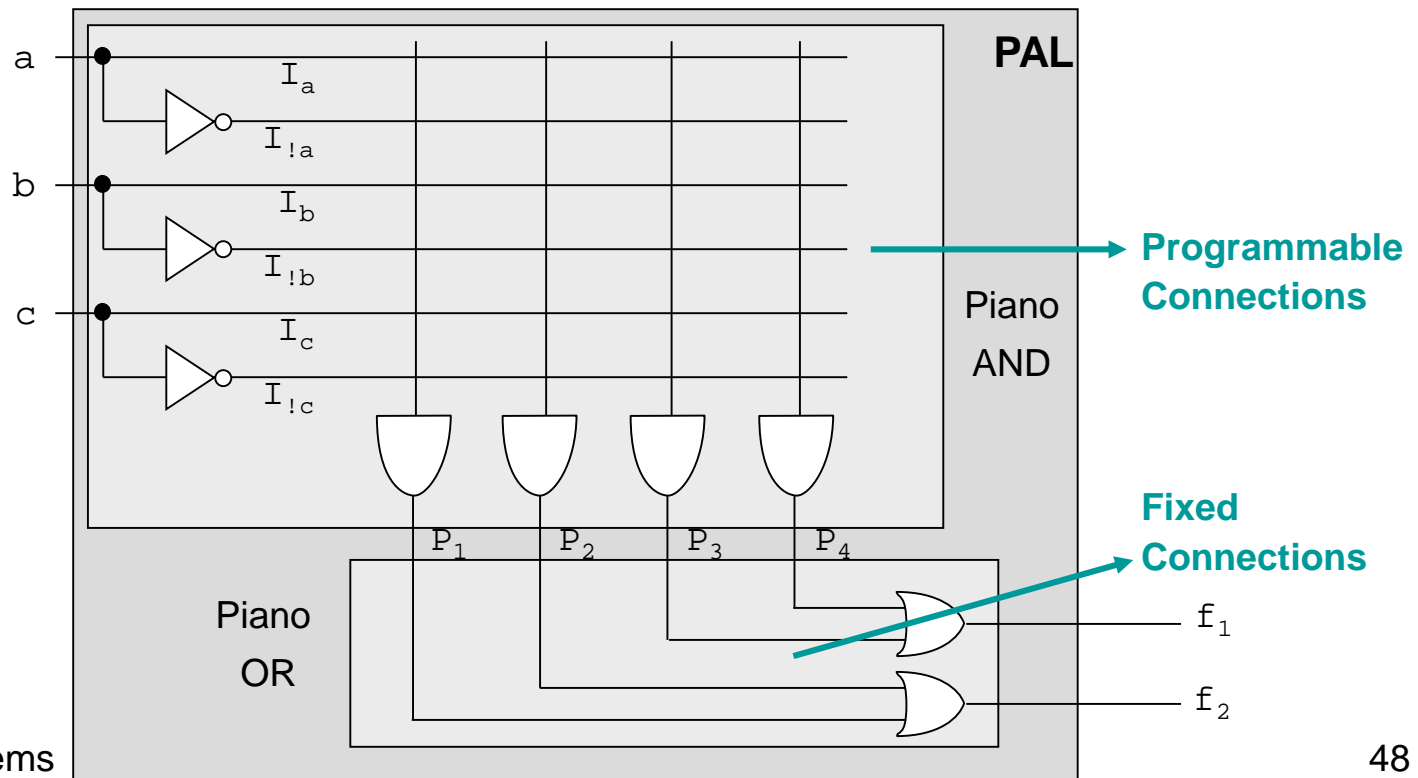
# 2-Levels PLD

- Programmable Array Logic (PAL)
  - A PAL can implement a sum of products
    - The same as PLA but with more limitations



# 2-Levels PLD

- Programmable Array Logic (PAL)
  - PAL schematic
    - Example: 3-inputs 2-outputs PAL (not programmed)





# Programmable Technologies

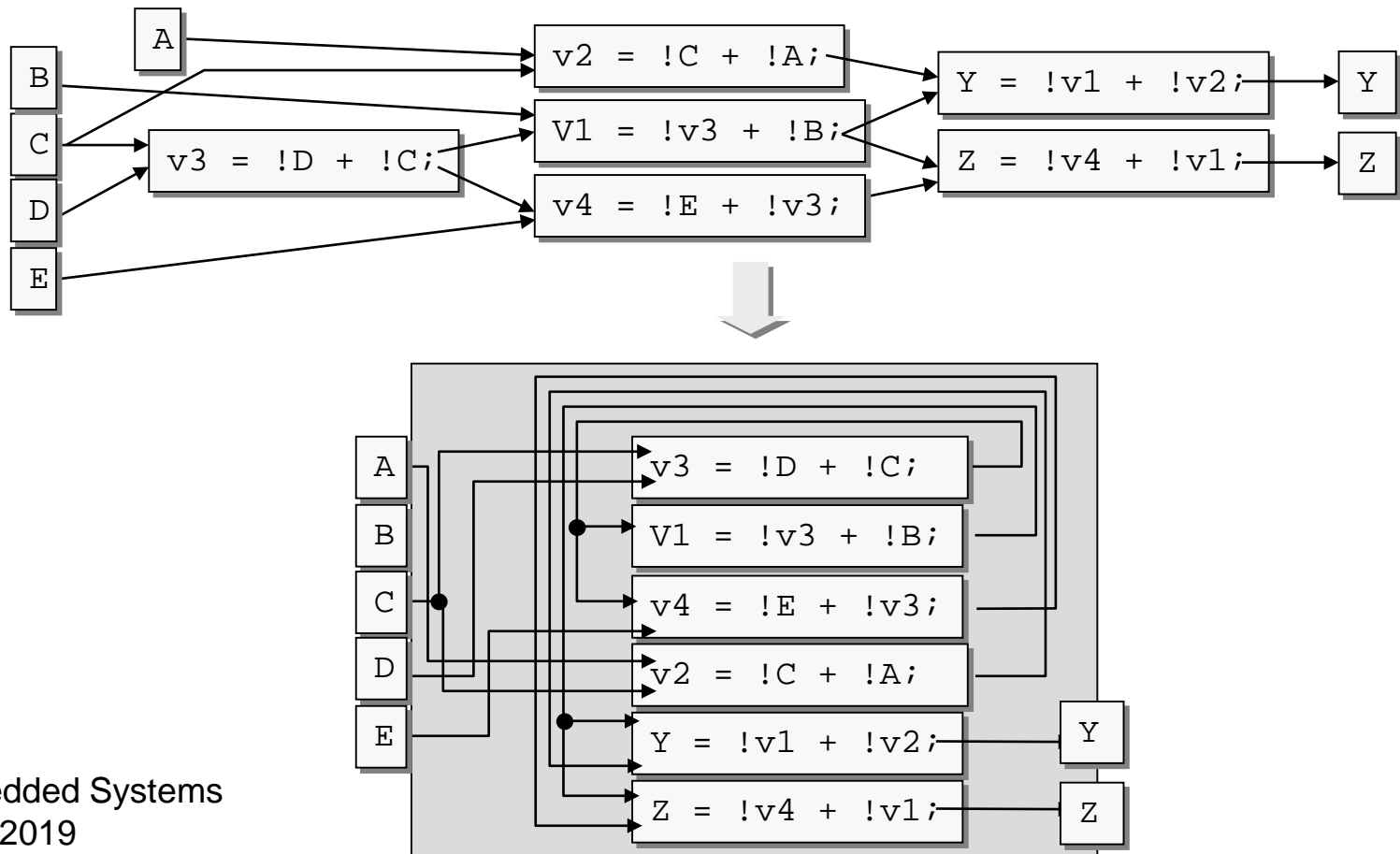
**Multi-level Programmable Logic Devices**

# Multi-level PLD

- Advanced PLA/PAL
  - As seen, basic PLA/PAL allow to realize only 2-levels combinatorial circuits
  - How is it possible to overcome such a limit?
    1. By introducing feedback loops
      - Multi-level multi-outputs combinatorial circuits
    2. By introducing storage elements (*flip-flop*)
      - Synchronous sequential circuits where the combinatorial part is a multi-level multi-outputs combinatorial circuit

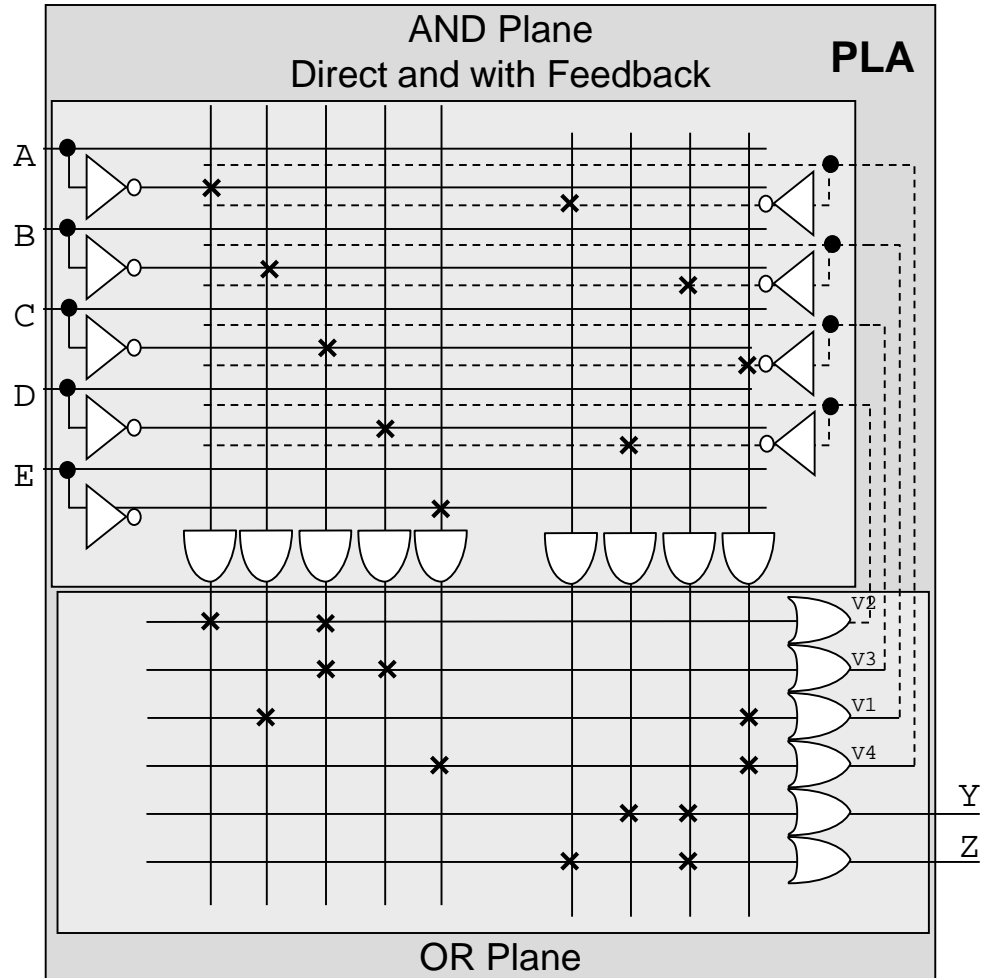
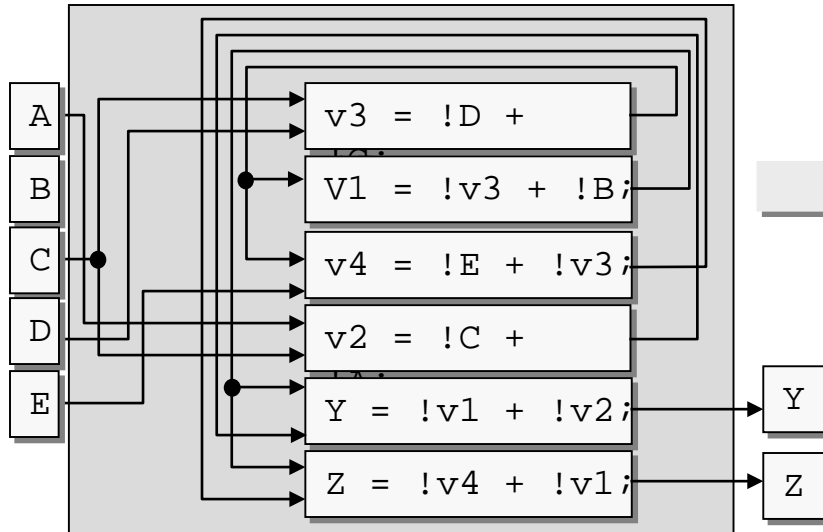
# Multi-level PLD

- Advanced PLA/PAL
  - Example: multi-level multi-outputs combinatorial circuit (1/2)



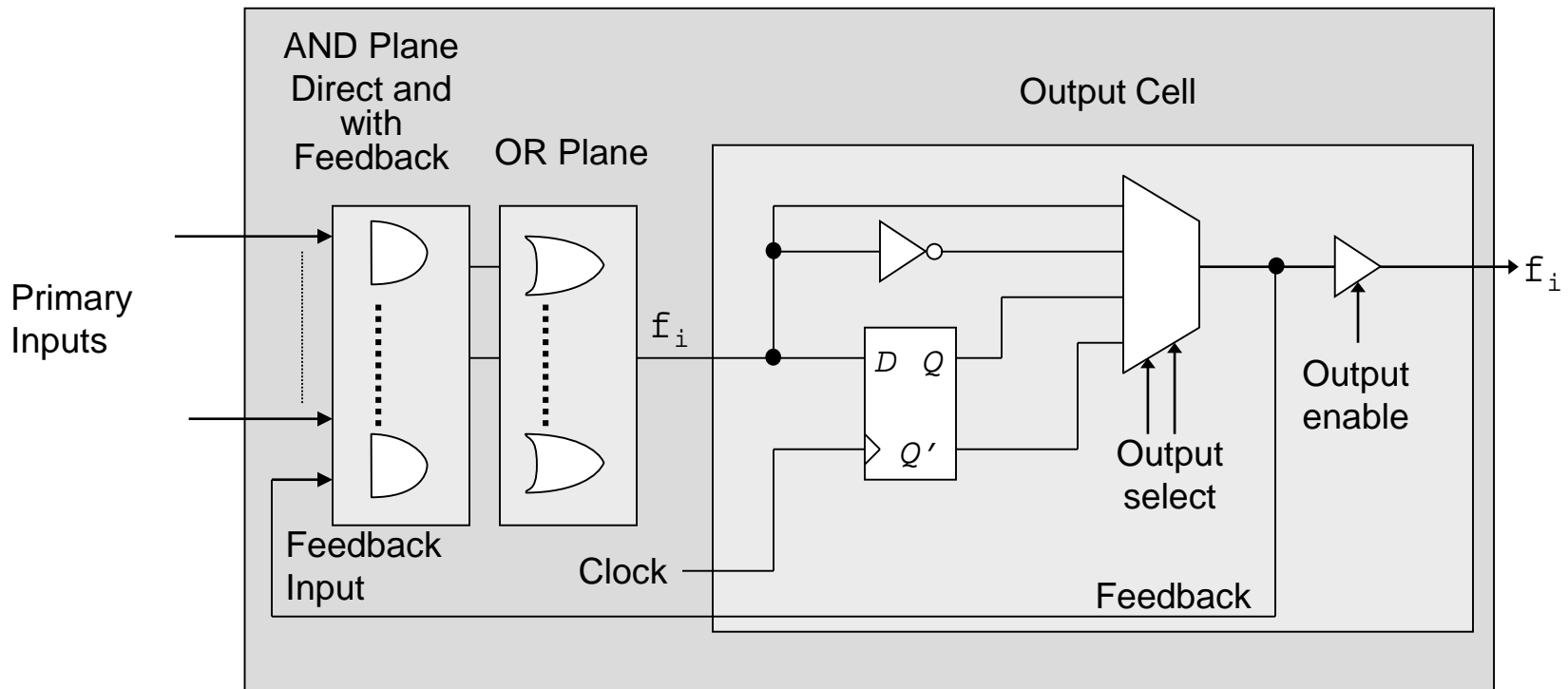
# Multi-level PLA

- Advanced PLA/PAL
  - Example (2/2)



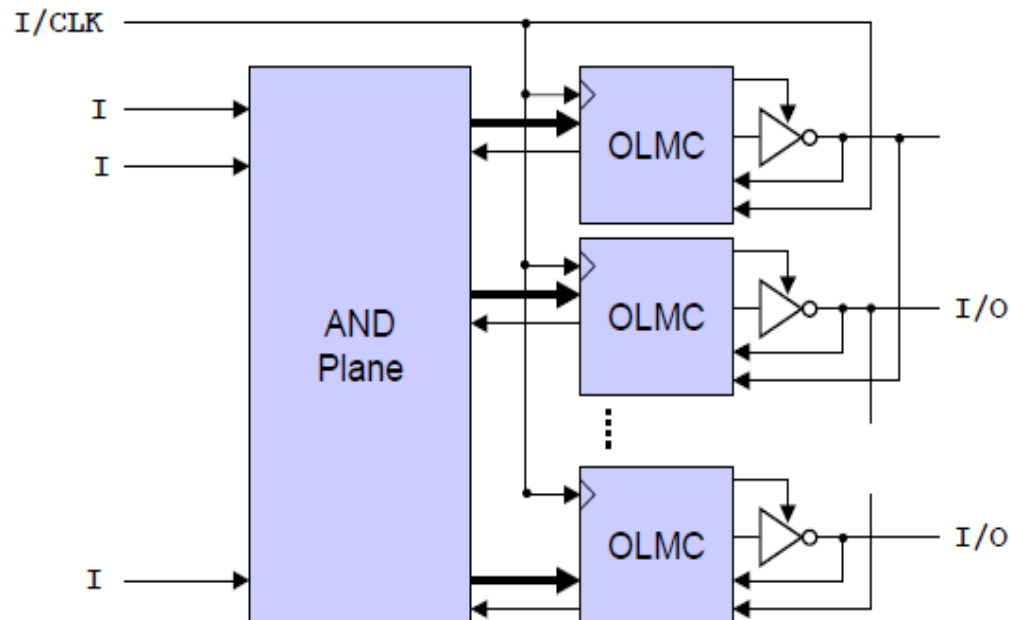
# Multi-level PLD

- Advanced PLA/PAL
  - Generic schematic



# Multi-level PLD

- Generic Array Logic (GAL)
  - GAL are a further enhancement of PAL/PLA
    - The output cells OLMC (*Output Logic Macro Cell*) make this kind of device very flexible

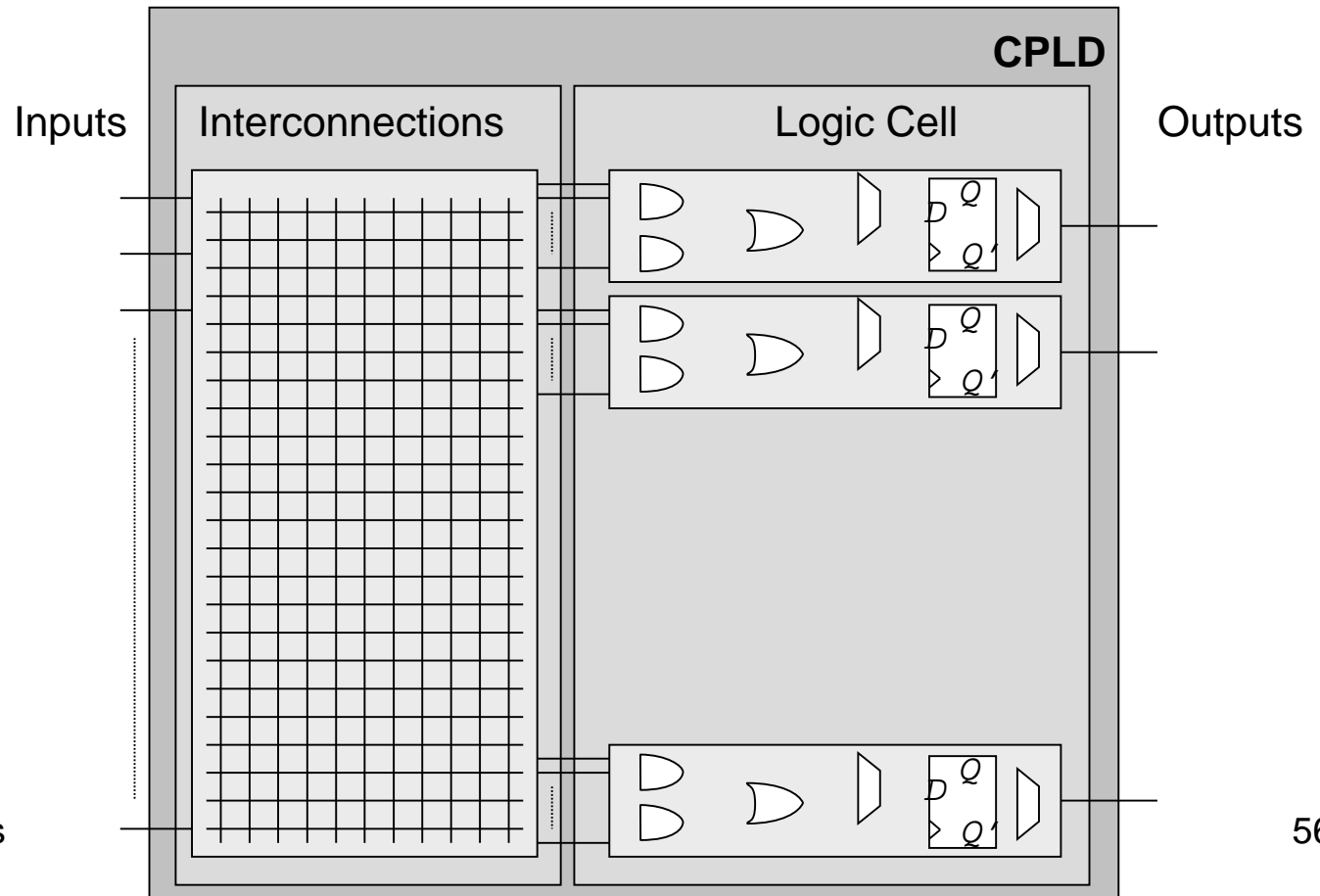


# Multi-level PLD

- Complex Programmable Logic Devices
  - The last evolution of PLA/PAL/GAL
    - They are characterized by
      - Global connections
      - Concentrated logic
    - With respect to PAL/PLA/GAL
      - Greater integration
      - More complex cells
      - Higher performances
      - More regular structure
        - » Easier to program!

# Multi-level PLD

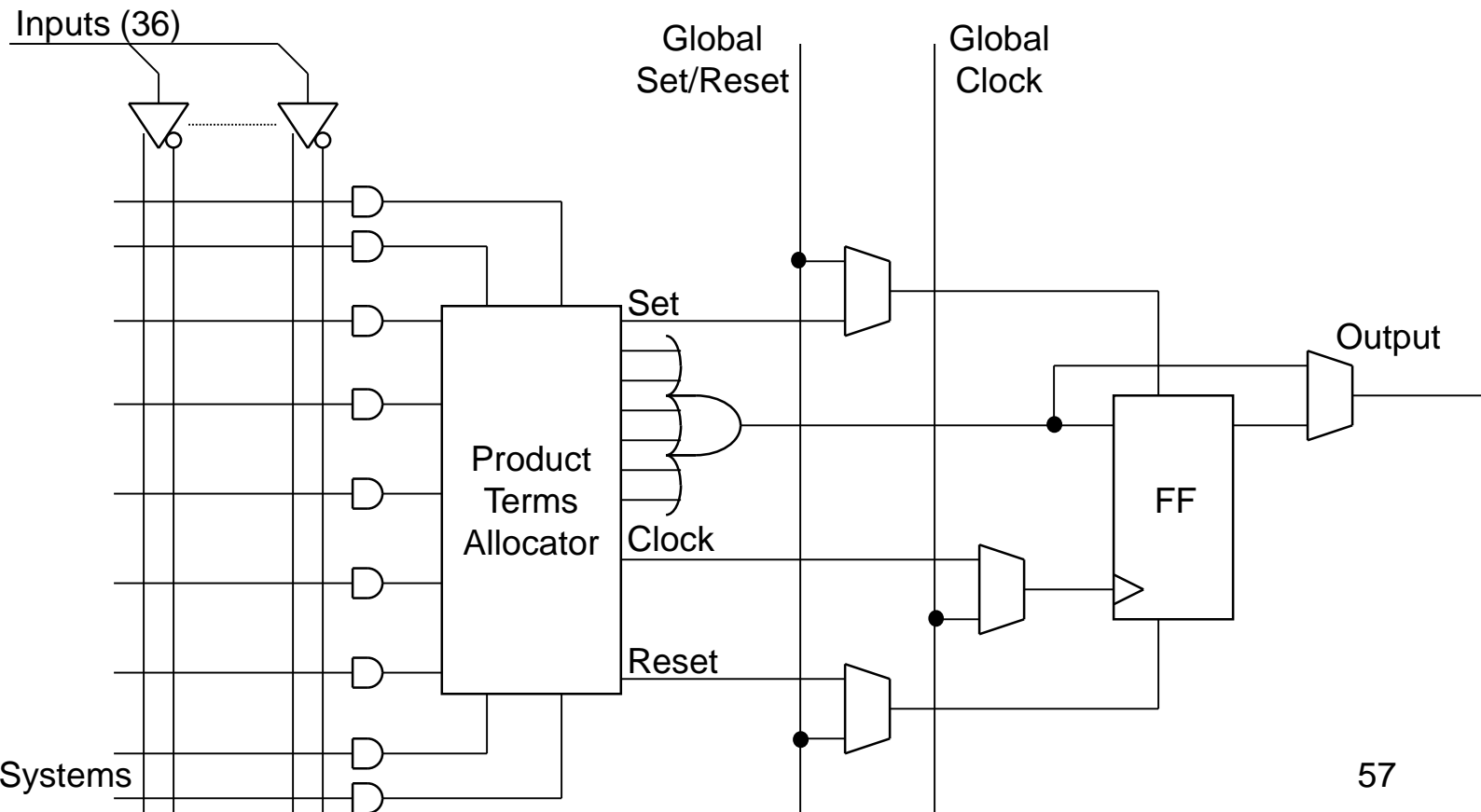
- Complex Programmable Logic Devices
  - Generic architecture





# Multi-level PLD

- Complex Programmable Logic Devices
  - Example: *Xilinx XC9500 logic cell*



# Considerations

- *Would you be able to manually (i.e. by placing **x** on a grid) program a CPLD?*

# Programmable Technologies

**FPGA**

***(Field Programmable Gate Arrays)***

# FPGA

- FPGA are *programmable* devices composed of an array of logic components that could be connected as needed
  - They provides
    - *Logic components* that offer different kinds of functionality (that could be used also partially)
      - Very complex
        - » They allow to localize a complex functionality to improve performance and reduce connectivity needs (maybe reducing degree of exploitation)
      - Little complex
        - » They allow a greater degree of exploitation (i.e. flexibility) but requires a lot of connectivity

# FPGA

- FPGA are *programmable* devices composed of an array of logic components that could be connected as needed
  - They provide
    - *Local and distributed connections*
      - Nets that cross a limited part of the device and that are shared by few logic components
        - » Limited delays and power consumption
      - In the same device can coexist local nets of different lengths
        - » Greater flexibility but complex programmability

# FPGA

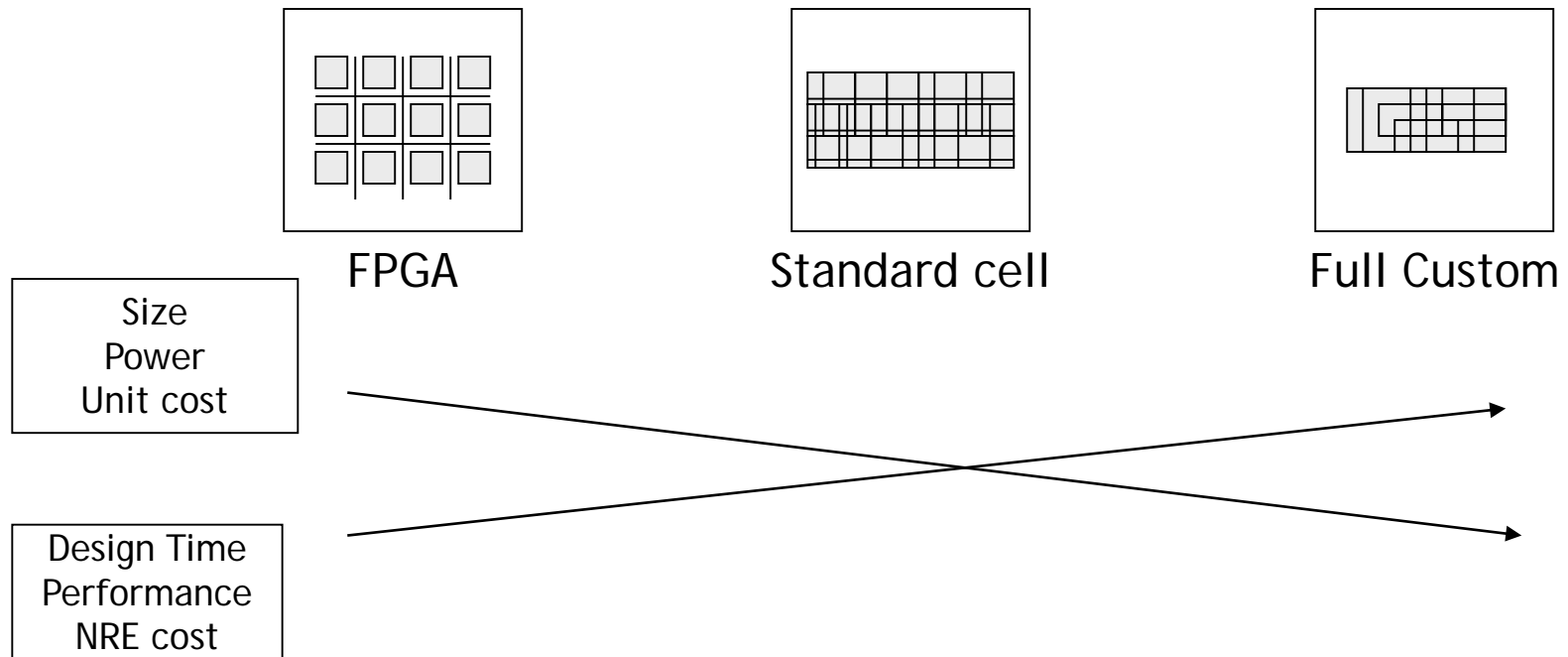
- Relevant issues
  - A good compromise between **performance** and **flexibility**
    - $\text{PLD} \Rightarrow \text{FPGA} \Leftarrow \text{MPGA}$
  - A good compromise between **performance** and **cost**
    - FPGA vs. ASIC
  - They are very useful for a fast verification of HW designs
    - *Fast prototyping*
  - A good compromise between **specialization** and **generality**
    - $\text{SW} \Rightarrow \text{FPGA} \Leftarrow (\text{AS})\text{IC}$

# FPGA

- Relevant issues
  - They allow to be programmed, in several and different times, with configurations that describe different systems or parts of a system
    - Multi-modal platforms (*off-line*)
      - The configurations are stored in a non volatile memory and copied in RAM at start-up time
    - Reconfigurables platforms (*on-line*)
      - The configurations (related to the whole system or a single part of it) are written in RAM (overwriting other ones) at run-time

# FPGA

- Comparison of technologies





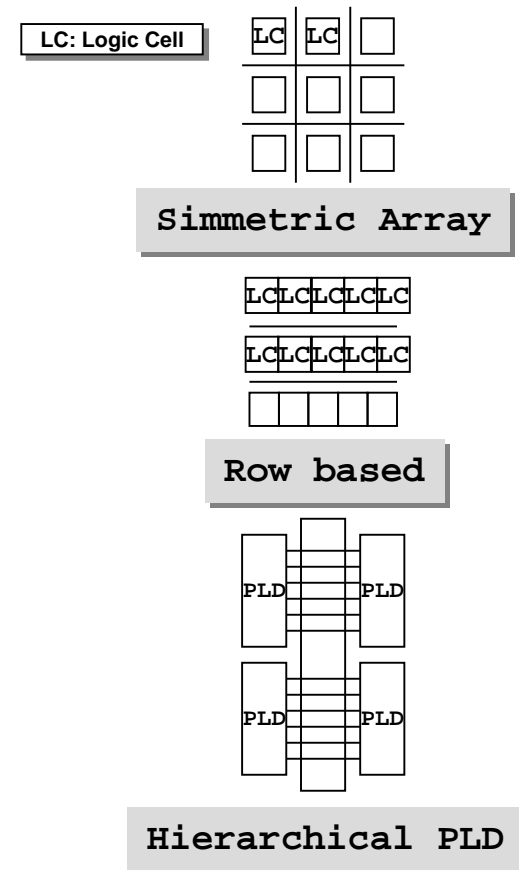
# FPGA

- Comparison of technologies

	<b>FPGA</b>	<b>Gate array</b>	<b>Standard Cell</b>	<b>Full Custom</b>
<b>Density</b>	L	M	M	H
<b>Flexibility</b>	L (H)	L	M	H
<b>Analog</b>	No	No	No	Yes
<b>Performance</b>	L	M	H	VH
<b>Design Time</b>	L	M	M	H
<b>Design Cost</b>	L	M	M	H
<b>Tools</b>	Simple	Complex	Complex	Very complex
<b>Volume</b>	L	M	H	H

# FPGA

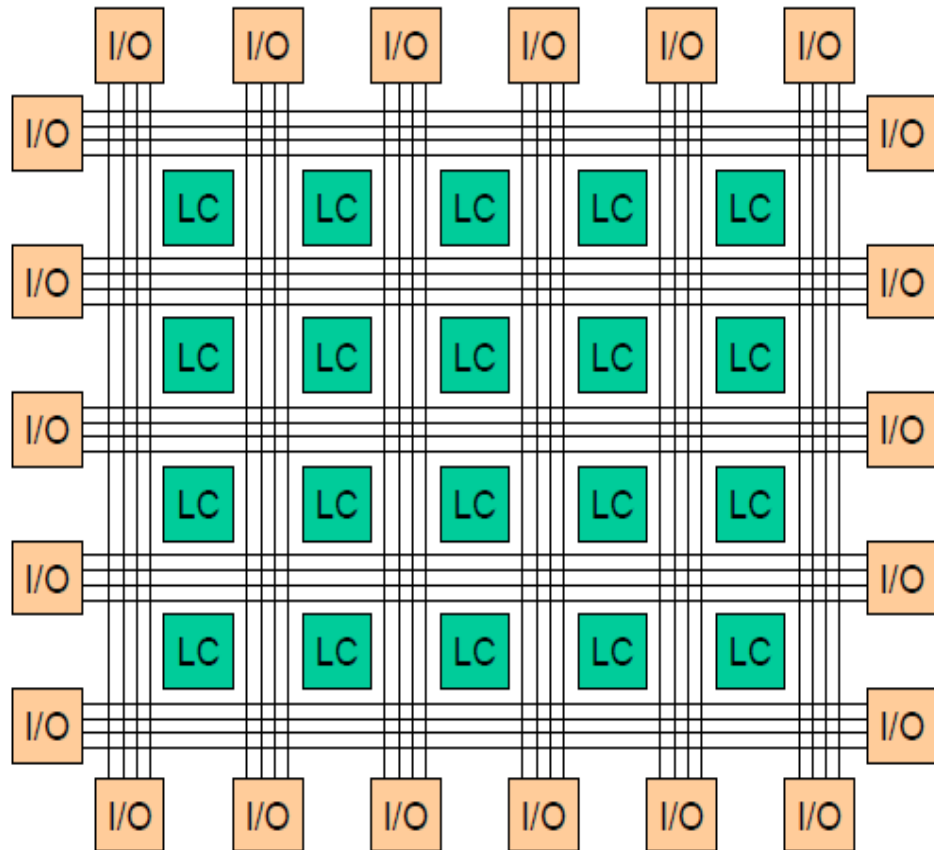
- FPGA families are characterized by 3 main features
  - General architectures
    - Simmetric array (e.g. *Xilinx*)
    - Row based (e.g. *Actel/Microsemi*)
    - Hierarchical PLD (e.g. *Altera/Intel*)
  - Logic blocks
    - Look-up Table based (e.g. *Xilinx*)
    - Multiplexer based (e.g. *Actel/Microsemi*)
    - PLD blocks (e.g. *Altera/Intel*)
  - Programming technologies
    - Static RAM (e.g. *Xilinx*)
    - EPROM/E<sup>2</sup>PROM (e.g. *Altera/Intel*)
    - Anti-Fuse (e.g. *Actel/Microsemi*)



# FPGA

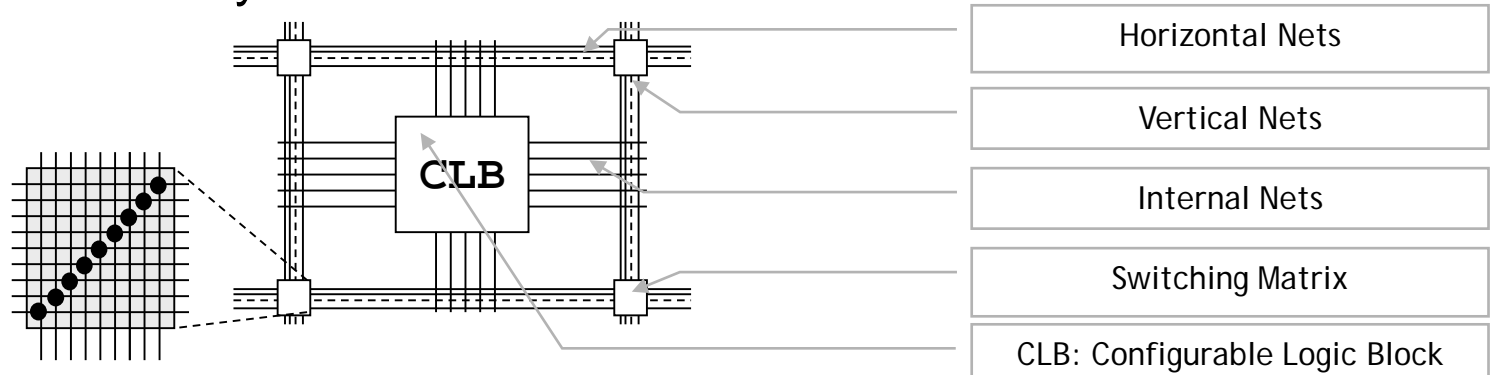
- General Architecture
  - Symmetric Array

LC: Logic Cell

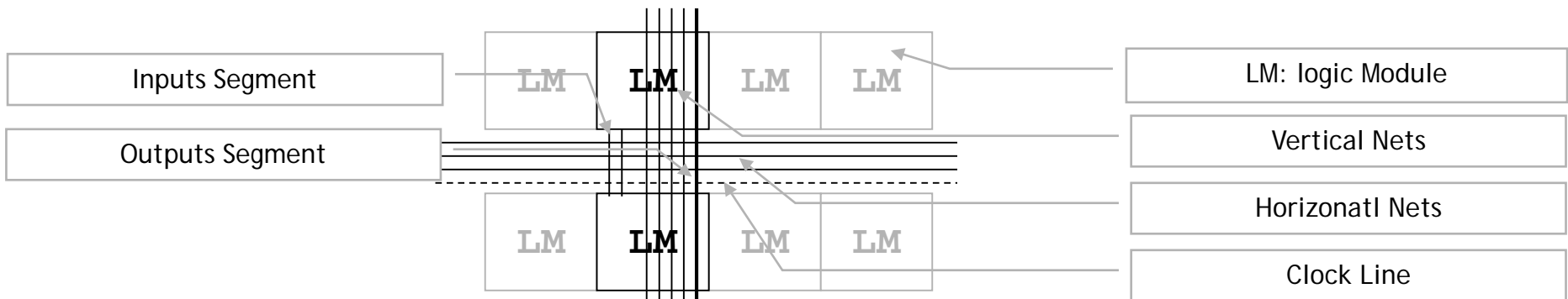


# FPGA

- General Architecture
  - Simmetric Array



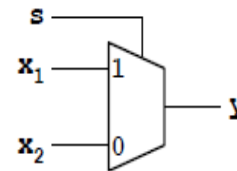
- Row based



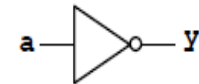
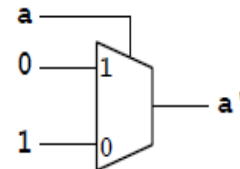
# FPGA

- Logic Blocks
  - Basic elements
    - Multiplexer

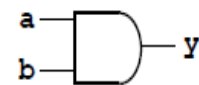
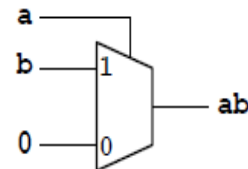
$$y = x_1 s + x_2 s'$$



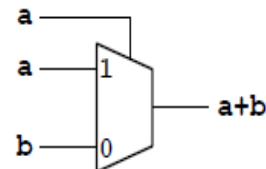
$$\begin{aligned} s &= a \\ x_1 &= 0 \\ x_2 &= 1 \\ y &= (0)a + (1)a' = a' \end{aligned}$$



$$\begin{aligned} s &= a \\ x_1 &= b \\ x_2 &= 0 \\ y &= ba + (0)a' = ab \end{aligned}$$

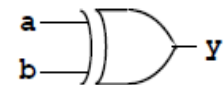
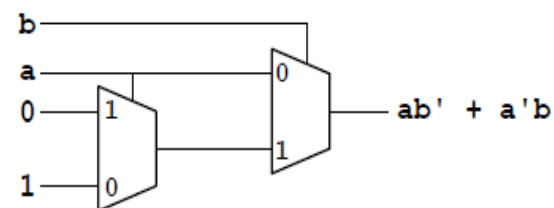
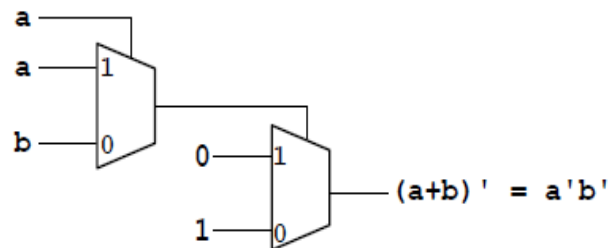
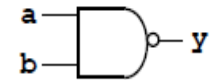
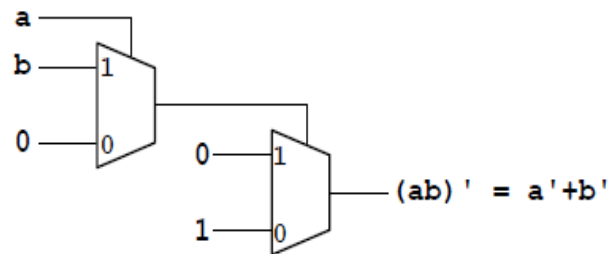


$$\begin{aligned} s &= a \\ x_1 &= a \\ x_2 &= b \\ y &= aa + ba' = a + a'b \\ &= a + b \end{aligned}$$



# FPGA

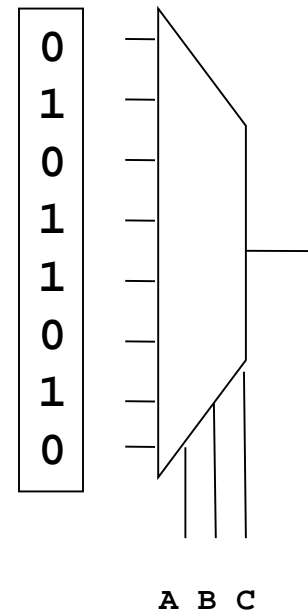
- Logic Blocks
  - Basic elements
    - Multiplexer



# FPGA

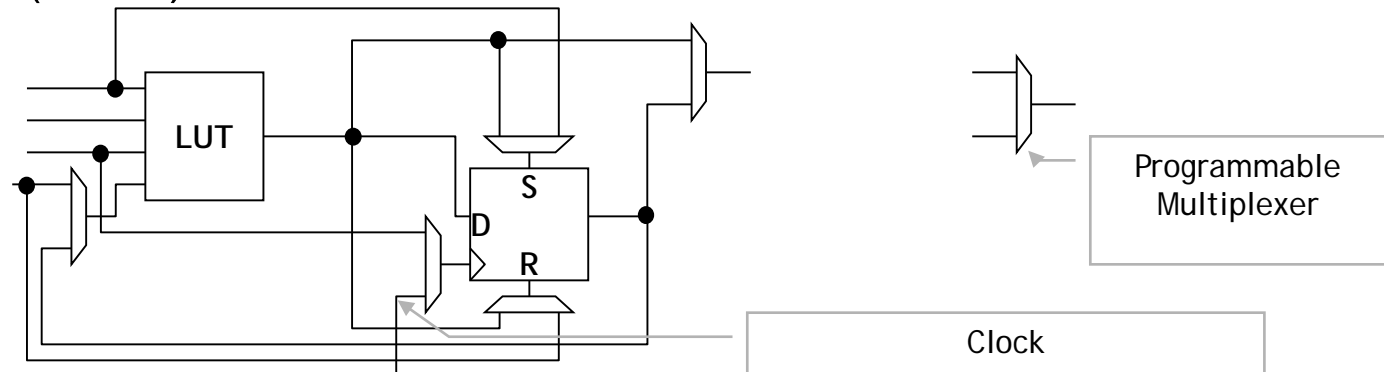
- Logic Blocks
  - Basic elements
    - $2^n$  inputs multiplexer:  $n$  variables combinatorial function
      - *Look-up table*

A	B	C	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

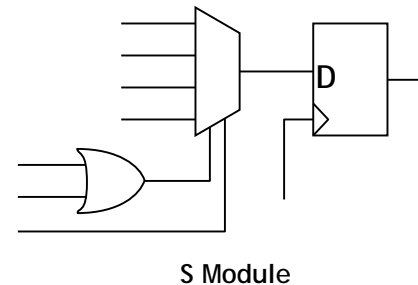
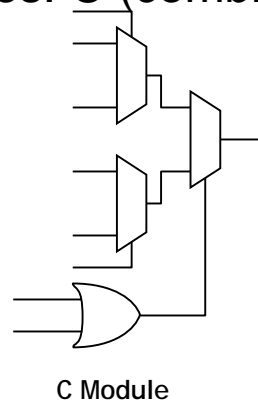


# FPGA

- Logic Blocks
  - Simplified (Xilinx) CLB schematic



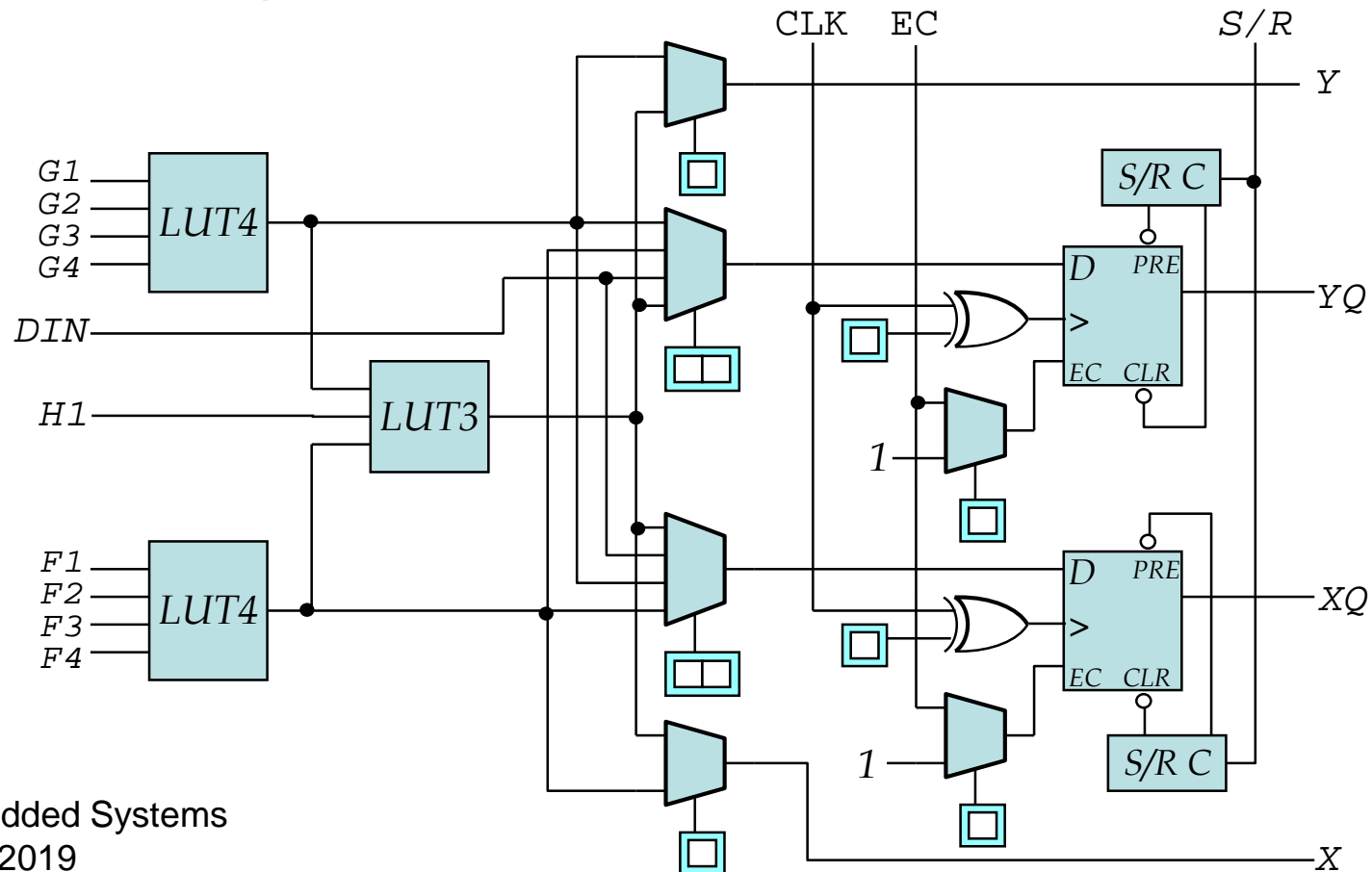
- Simplified (Altera/Intel) LM schematic
  - 2 modules: C (combinatorial) and S (sequential)





# FPGA

- Logic Blocks
  - Less simplified CLB schematic



# Considerations

- *Would you be able to manually (i.e. by placing **x** on a grid) program a FPGA?*
  - A design activity able to consider the FPGA details is not manageable without a proper *Design Methodology* providing:
    - The possibility to model at an higher abstraction level the desired system behaviour and/or system structure
      - *Hardware Description Languages*
        - » But also more abstract languages (UML/MARTE, Simulink, SysML, etc.)
    - SW tools to verify/validate the model and to support/automate FPGA configuration in order to obtain the modeled system
      - *Simulation/Synthesis/Mapping Tools*
        - » But also formal verification
    - A *Design Flow* that the designer can follow/adopt in order to be driven from the model to the implementation

# FPGA

- *Typical FPGA Design Flow*

