Lecture 13
Implementation technologies

Learning outcomes

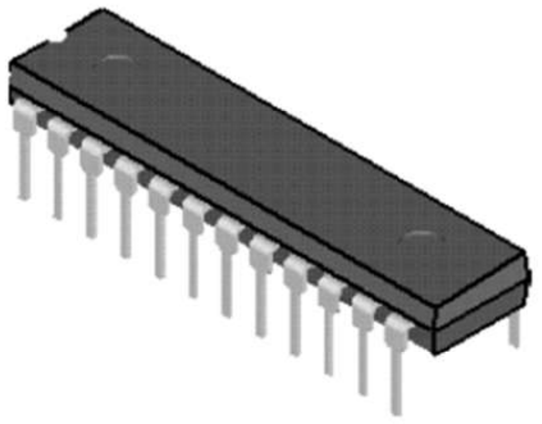
- Identify the appropriate modern realization technologies and synthesis techniques for the implementation of a given digital system
- Discuss the different types of programmable logic
- Describe the basic architecture and operation of programmable logic

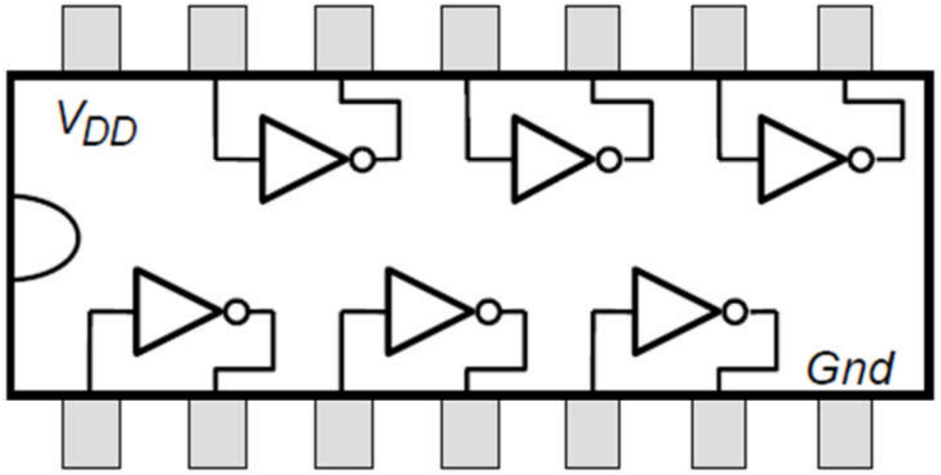
IMPLEMENTATION TECHNOLOGIES

- Discrete parts (e.g. 7400 devices)
- Full Custom design
  - Mostly manual design, long design cycle
  - High performance, high volume
  - Microprocessors, analog, IP …
- Standard cell
  - Pre-designed cells, CAD, short design cycle
  - Medium performance, ASIC
- Programmable logic devices
  - Pre-fabricated, fast automated design, low cost
  - Prototyping, reconfigurable computing

Using Discrete Parts

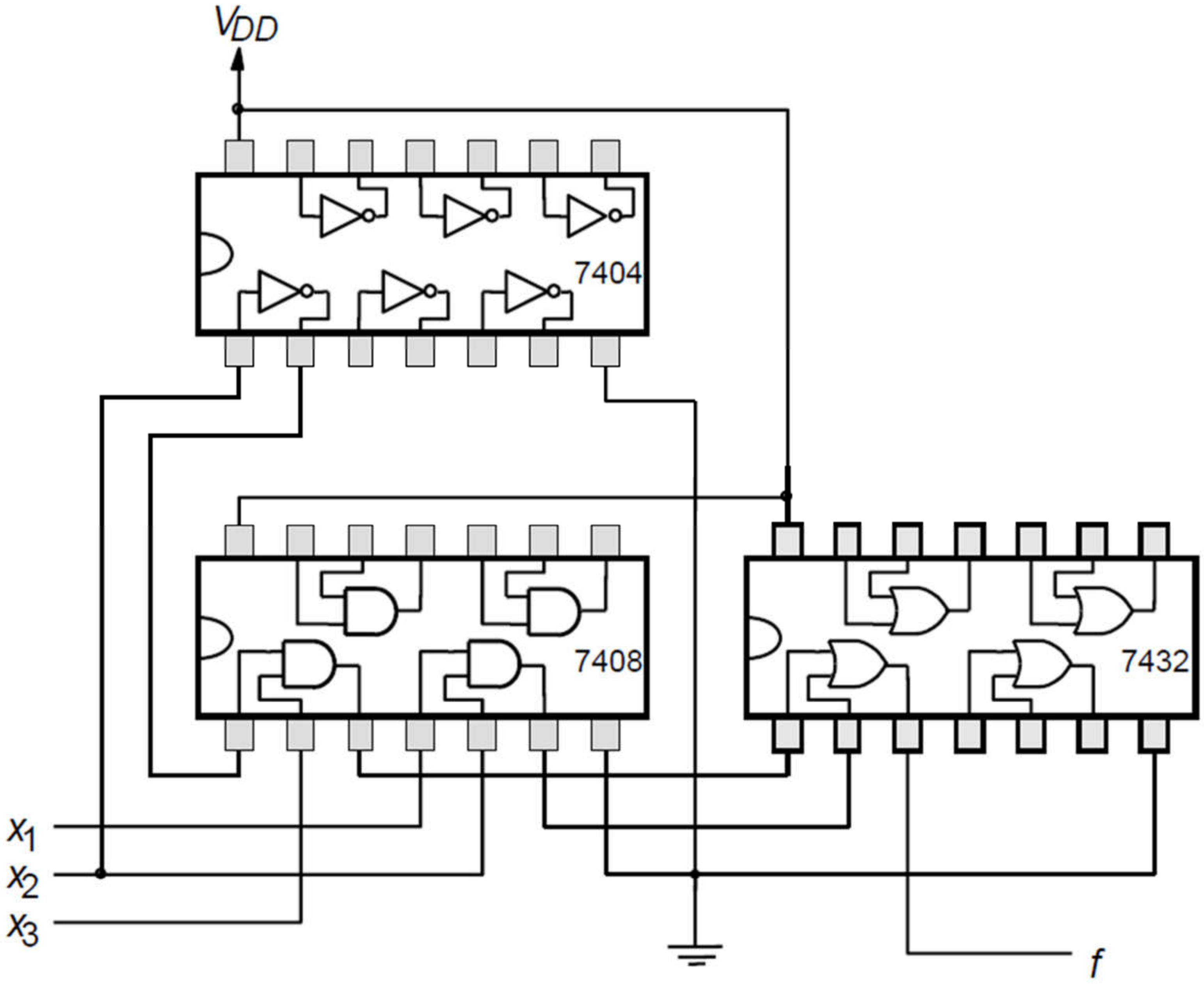
- Discrete parts (e.g. 7400 devices)
- Theoretically, we only need 2-input NAND or NOR gates to build anything
- Bulky, slow, expensive, and tedious.


Dual-inline package



Structure of 7404 chip

Using Discrete Parts

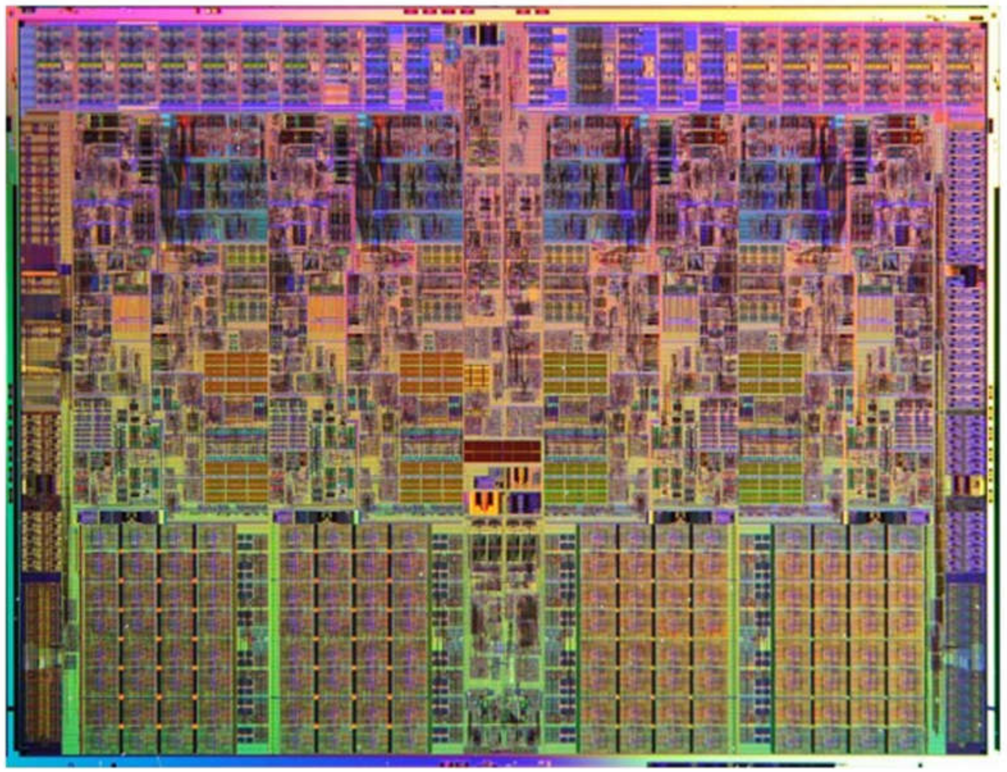


Implementation of $f = \overline{x}_2 x_3 + x_1 x_2$

Full Custom ASIC – Intel Core i7

- A full-custom ASIC design defines all photolithographic layers required for manufacturing. Each transistor/wire is drawn and connected in a full-custom manner by hand!
- Benefits
  - reduced area and power
  - higher performance
- Drawbacks
  - increased design time
  - higher designer skills required
  - increased manufacturing time and cost
  - Not viable unless the market is very large
  - Highly risky: once designed, it cannot be changed easily.

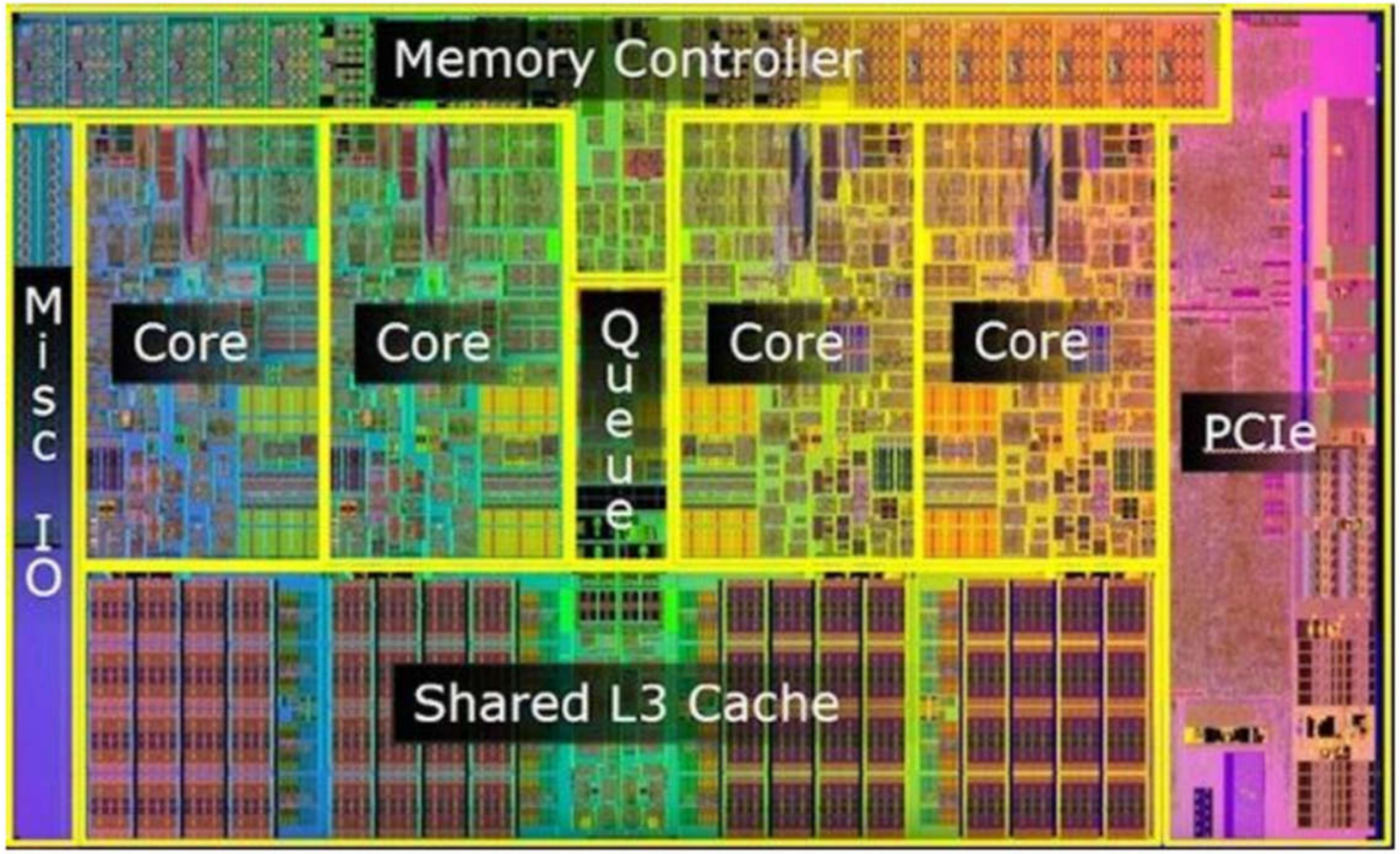
Intel Core i7



over ¾ billion transistors

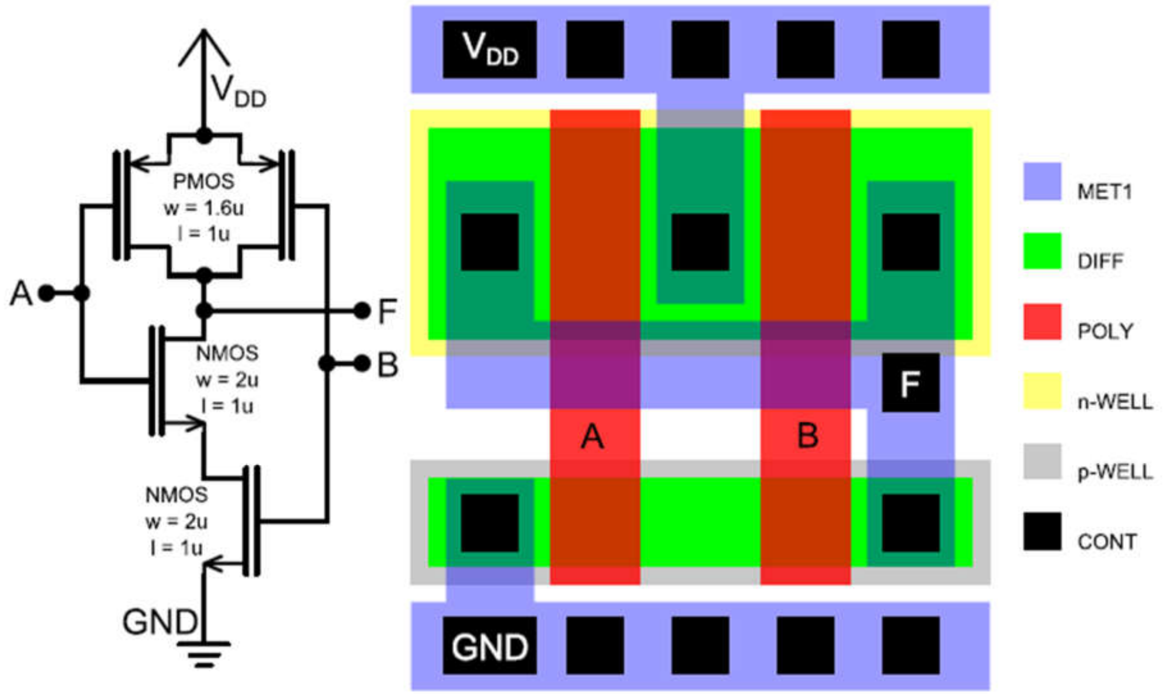
Standard cell ASIC

- A basic cell library (gates, flip-flops, memory cells) is available with different views of the cells provided: schematic symbol, device-level structure, layout.
- CAD tools are used to determine the floorplan that minimizes chip-level global routing.



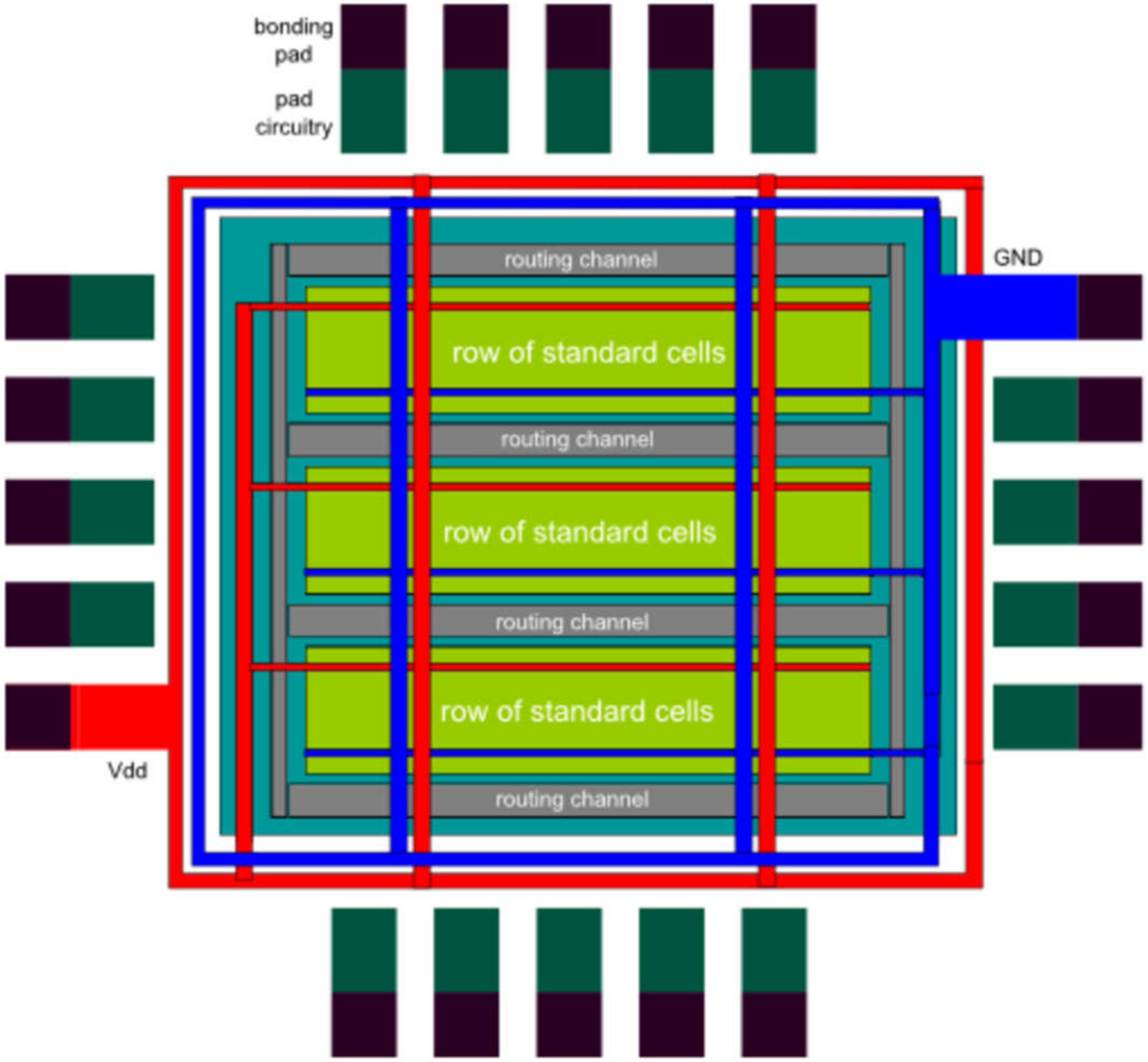
Standard cell ASIC

- Highly optimized design units
- The function and the timing are verified
- Geometric restrictions of the standard cells
  - common height (arbitrary width)
  - position of the supply rails (Vdd, GND)
  - inputs and outputs on grid



Standard cell ASIC

- The basic standard-cell floorplan consists of cell-rows and routing

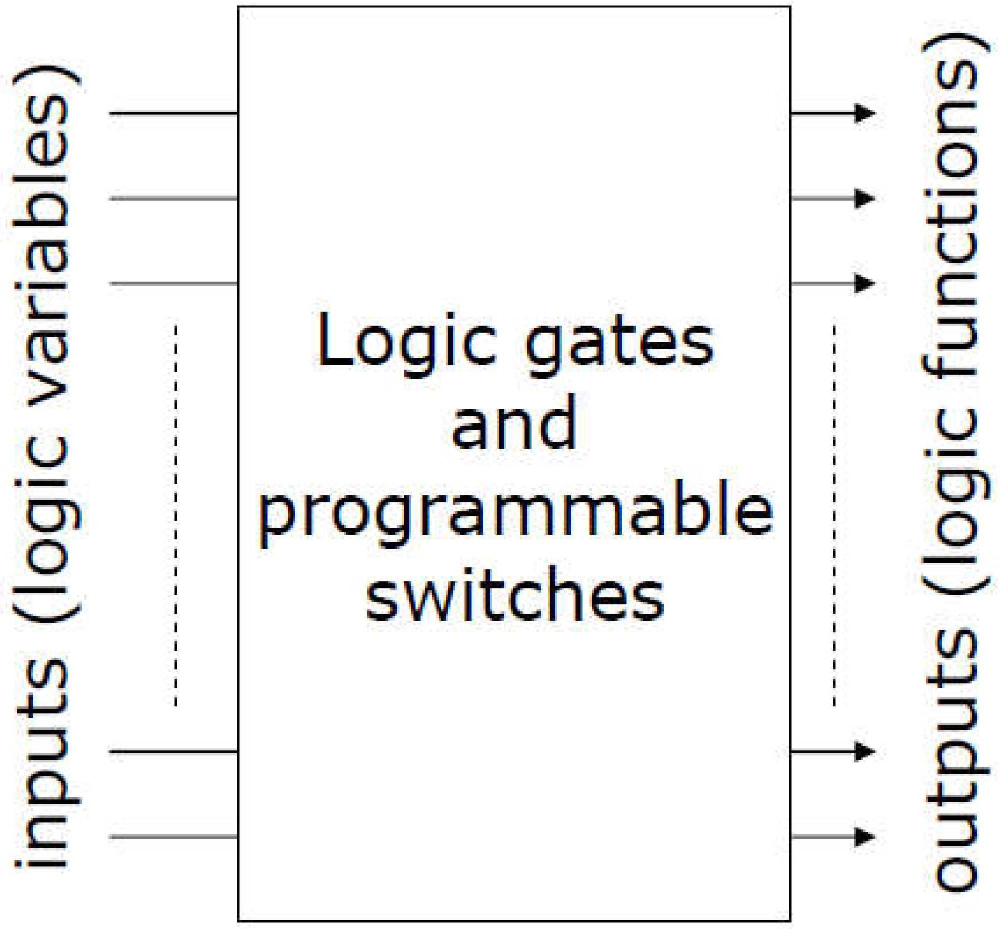


Programmable Logic Device (PLD)

- A PLD is a general purpose chip for implementing logic circuitry

- Can be viewed as a black box containing logic gates and programmable switches that allow for different connections between the logic elements

- Can implement whatever logic circuit is needed – subject to resources of the device

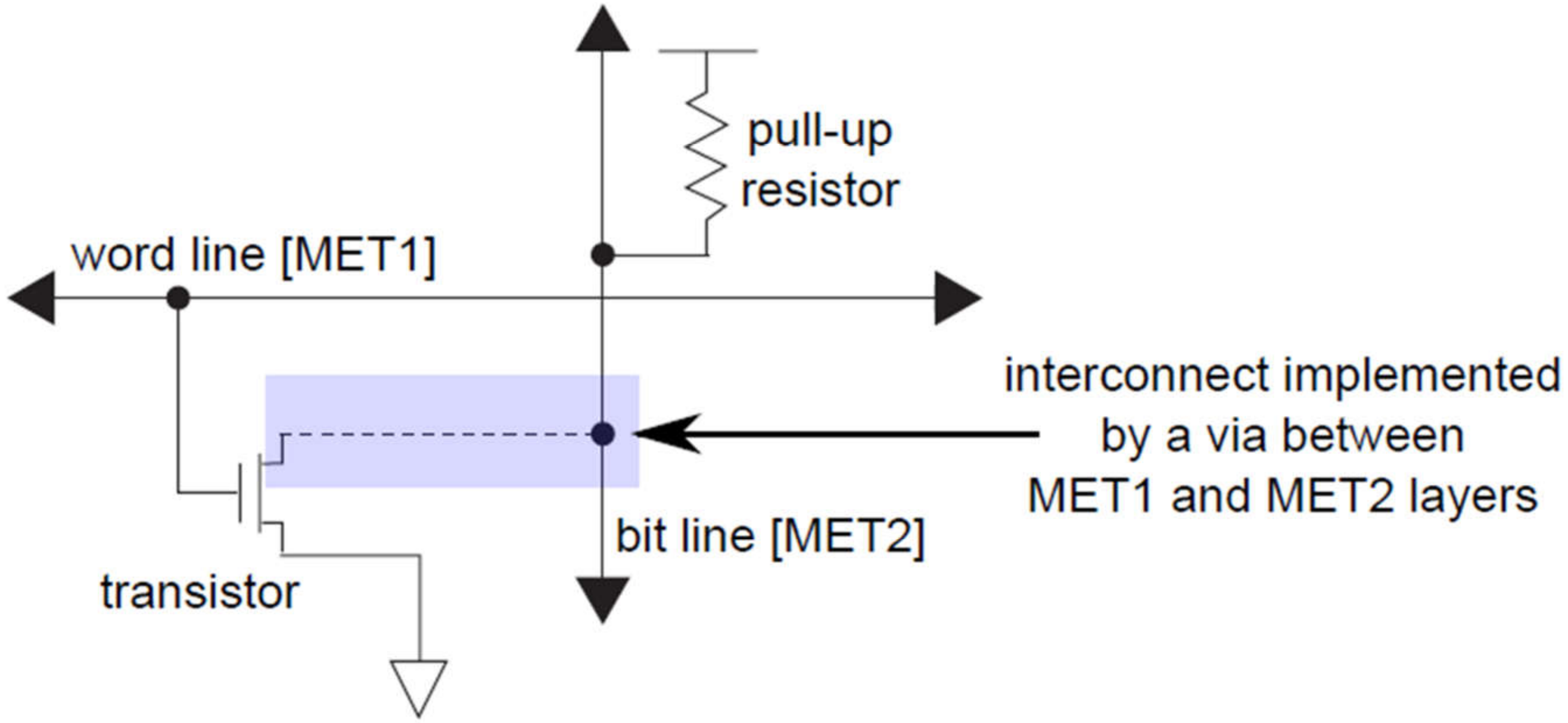

inputs (logic variables)
Logic gates and programmable switches
outputs (logic functions)

Classification — Programmable Logic Devices (PLDs)

- **PLA —** a Programmable Logic Array (PLA) is a relatively small PLD that contains two levels of logic, an ANDplane and an OR-plane, where both levels are programmable
- **PAL —** a Programmable Array Logic (PAL) is a relatively small PLD that has a programmable AND-plane followed by a fixed OR-plane
- **SPLD —** refers to any type of Simple PLD, usually either a PLA or PAL
- **CPLD —** a more Complex PLD that consists of an arrangement of multiple SPLD-like blocks on a single chip.
- **FPGA —** a Field-Programmable Gate Array is a PLD featuring a general structure that allows very high logic capacity.

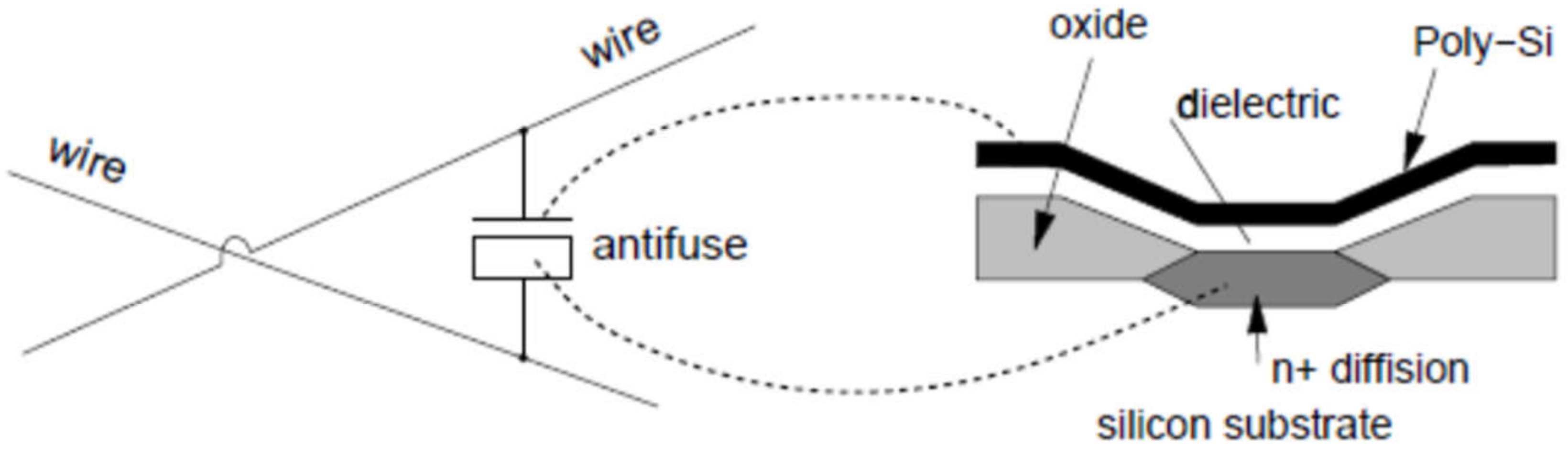
Implementing programmable interconnections

- **Mask-programming:** The interconnection between two wires is implemented by a **metal or contact window layer**. The interconnect configuration or the content of the ROM **cannot be changed after the manufacturing**.



Implementing programmable interconnections

- **Anti-fuse**: An "anti-fuse" physically consists of three sandwiched layers: the top and bottom layers are conductors and the middle layer is an insulator. With a **high voltage pulse** the insulator turns into a low-resistance connection (**One-time programmable** device)

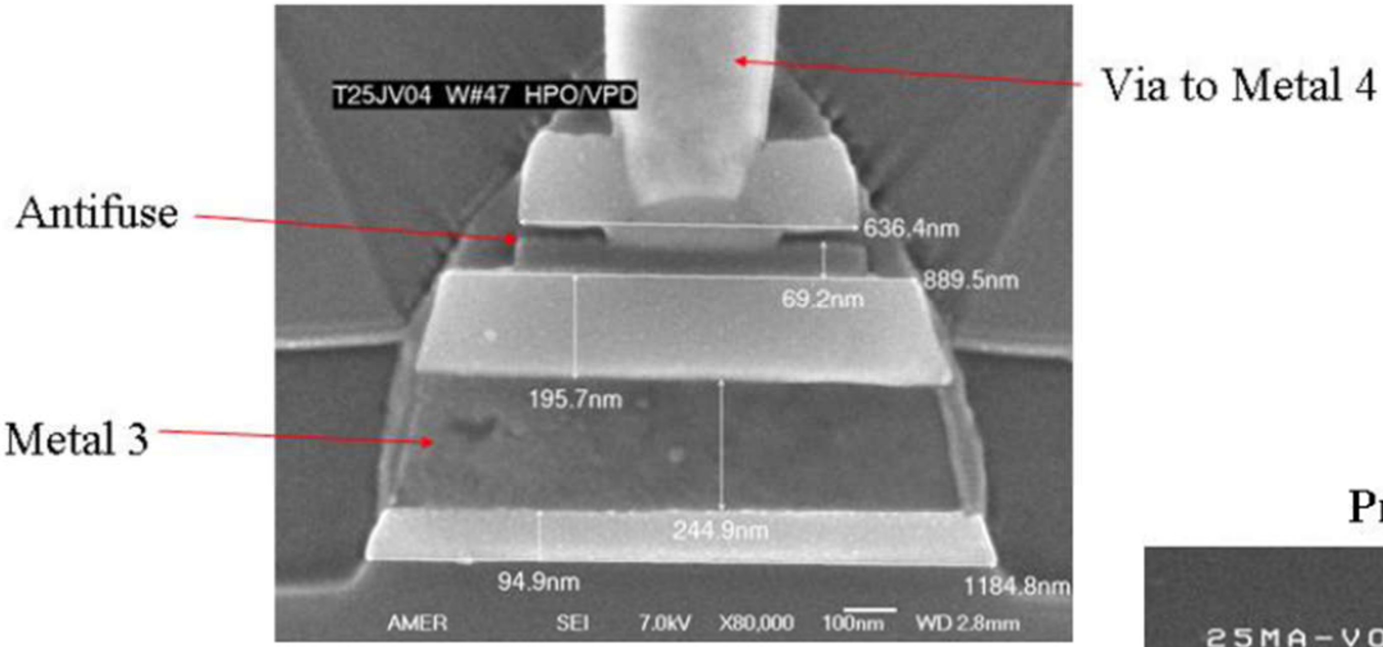


- **Anti-fuse:** normally disconnected, make wanted connections

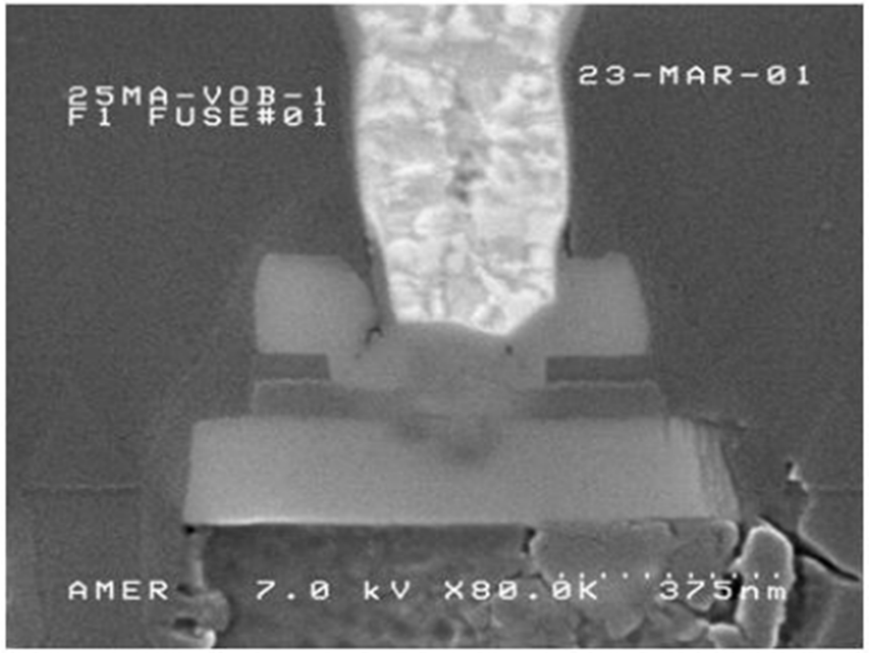
- **Fuse** also exist: normally connected, break unwanted ones

Implementing programmable interconnections

Unprogrammed Antifuse

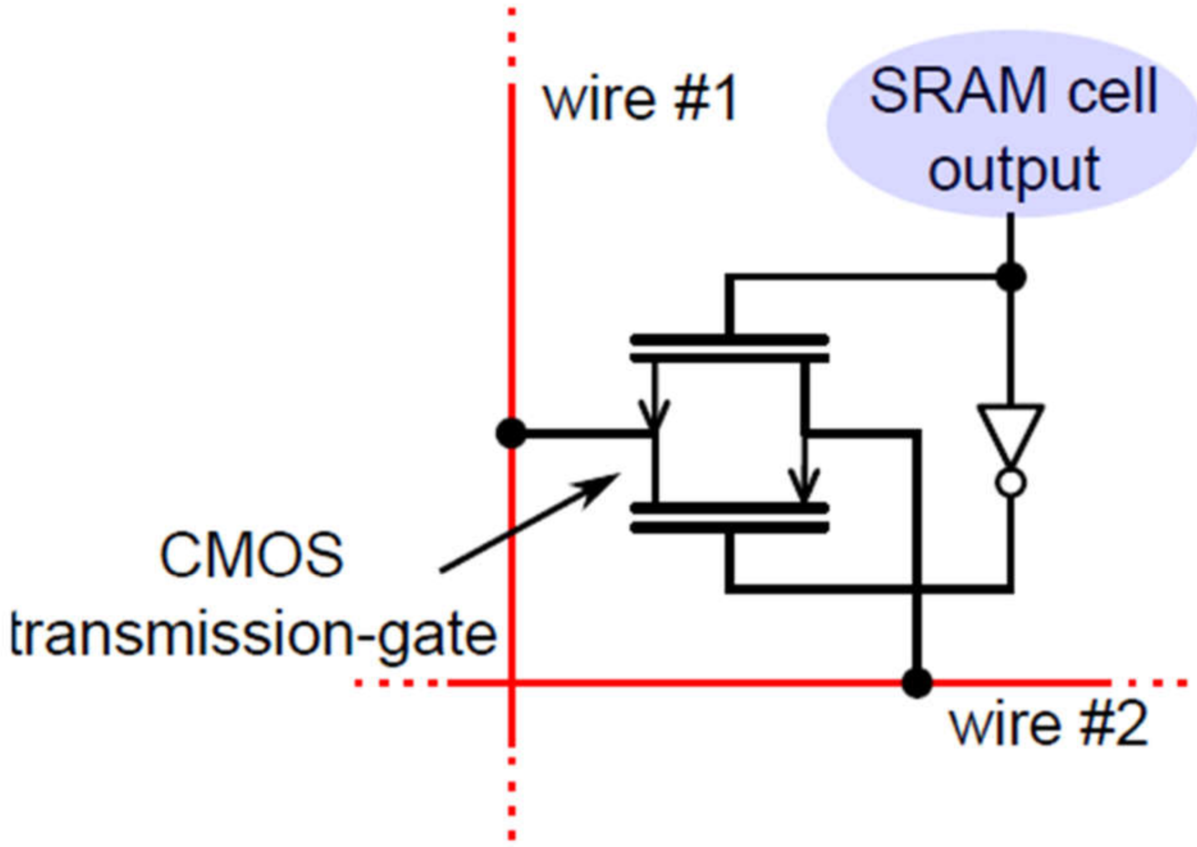


Programmed Antifuse



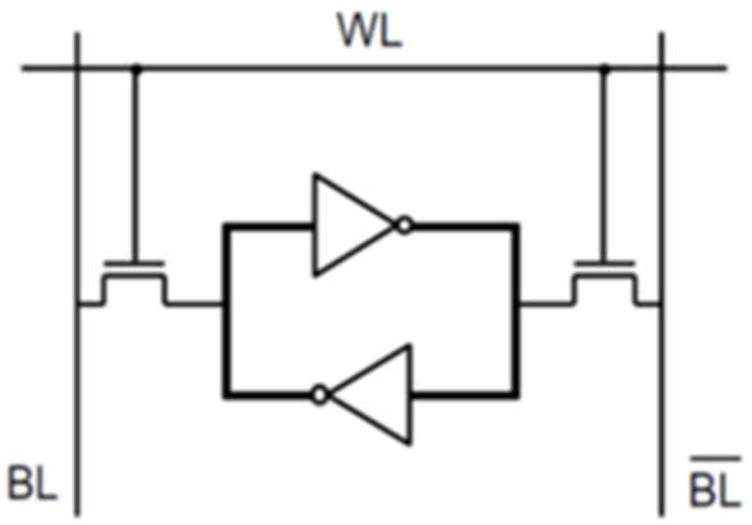
Implementing programmable interconnections

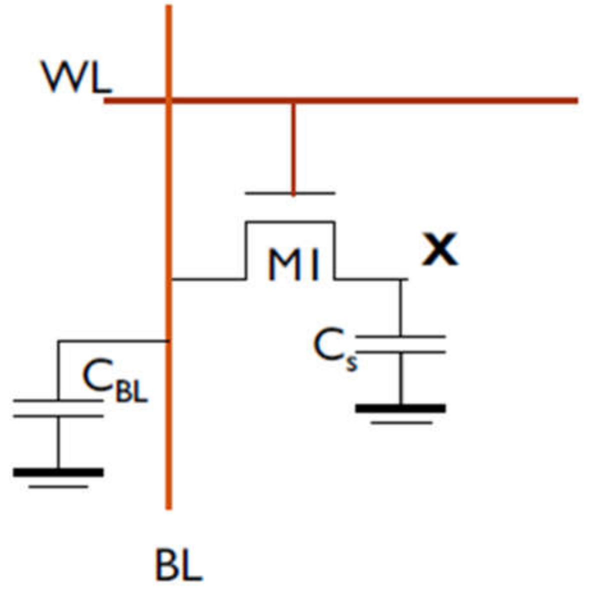
- **<u>SRAM + transmission-gate</u>**: A **transmission-gate** is placed between the wires, whose gate is connected to a specific SRAM configuration bit. The configuration can be changed **by the user**.



SRAM & DRAM

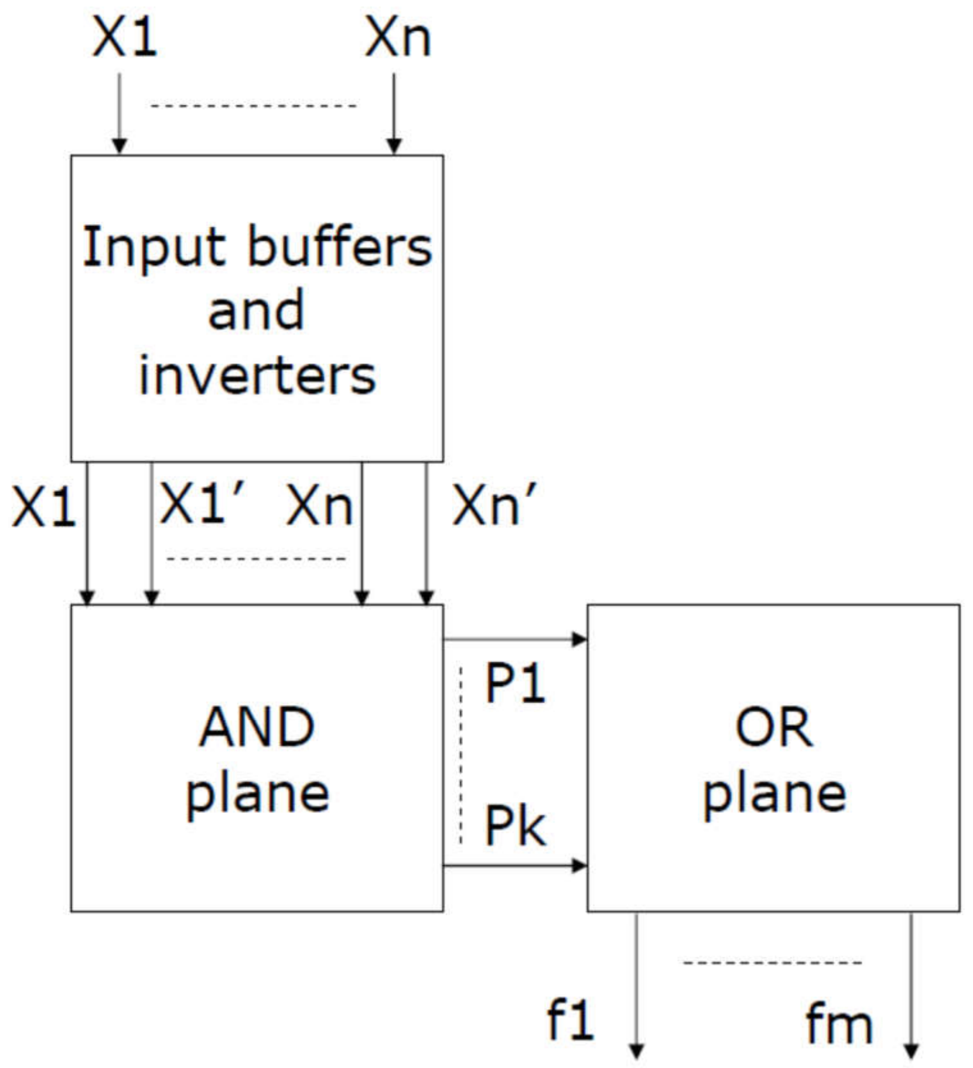
- Static – SRAM
  - data is stored as long as voltage supply is enabled
  - large cells (6 FETs/cell) → fewer bits/chip
  - fast → used where speed is important (e.g., caches)
- Dynamic – DRAM
  - periodic refresh required (every 1 to 4 ms)
  - small cells (1 FET/cell) → more bits/chip
  - slower → used for main memories





Programmable Logic Array (PLA)

- The first PLD developed was the ***programmable logic array*** (PLA)
- Based on the premise that any function can be written in SOP form, a PLA consists of:
  - Input buffers and inverters that provide the true and complement form for each input variable
  - A collection of AND gates, with inputs that are selectable (programmable)
  - A collection of OR gates, with inputs that are selectable programmable)

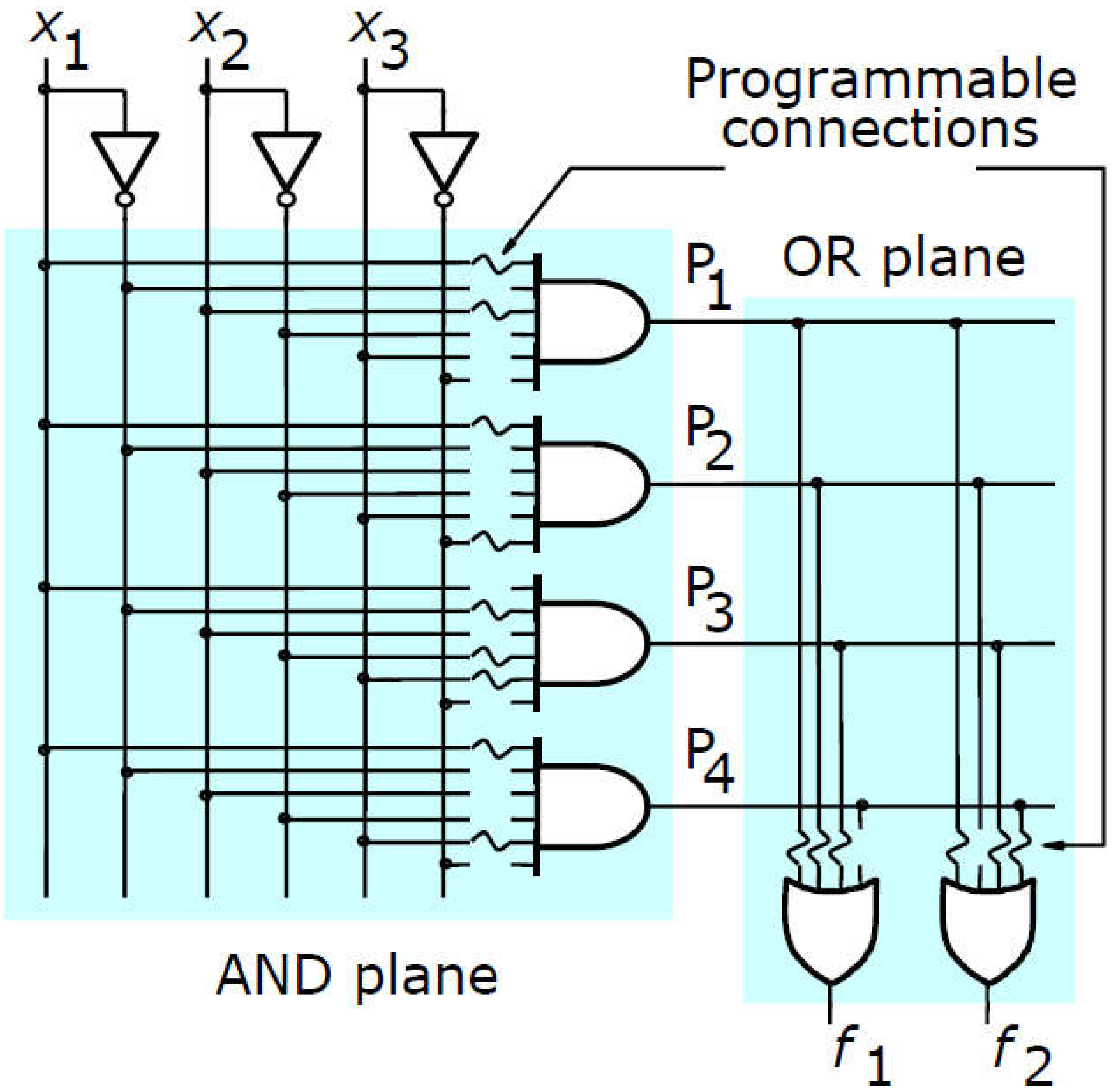


Gate-level diagram of a PLA

Pre-fabricated building block of many AND/OR gates

"Personalized" by making/breaking connections among the gates

Programmable array block diagram for sum of products form



Enabling concept

- Shared product terms among outputs

example:

F0 = A + B' C'
F1 = A C' + A B
F2 = B' C' + A B
F3 = B' C + A

personality matrix

product term	inputs A	B	C	outputs F0	F1	F2	F3
AB	1	1	–	0	1	1	0
B'C	–	0	1	0	0	0	1
AC'	1	–	0	0	1	0	0
B'C'	–	0	0	1	0	1	0
A	1	–	–	1	0	0	1

reuse of terms

input side:

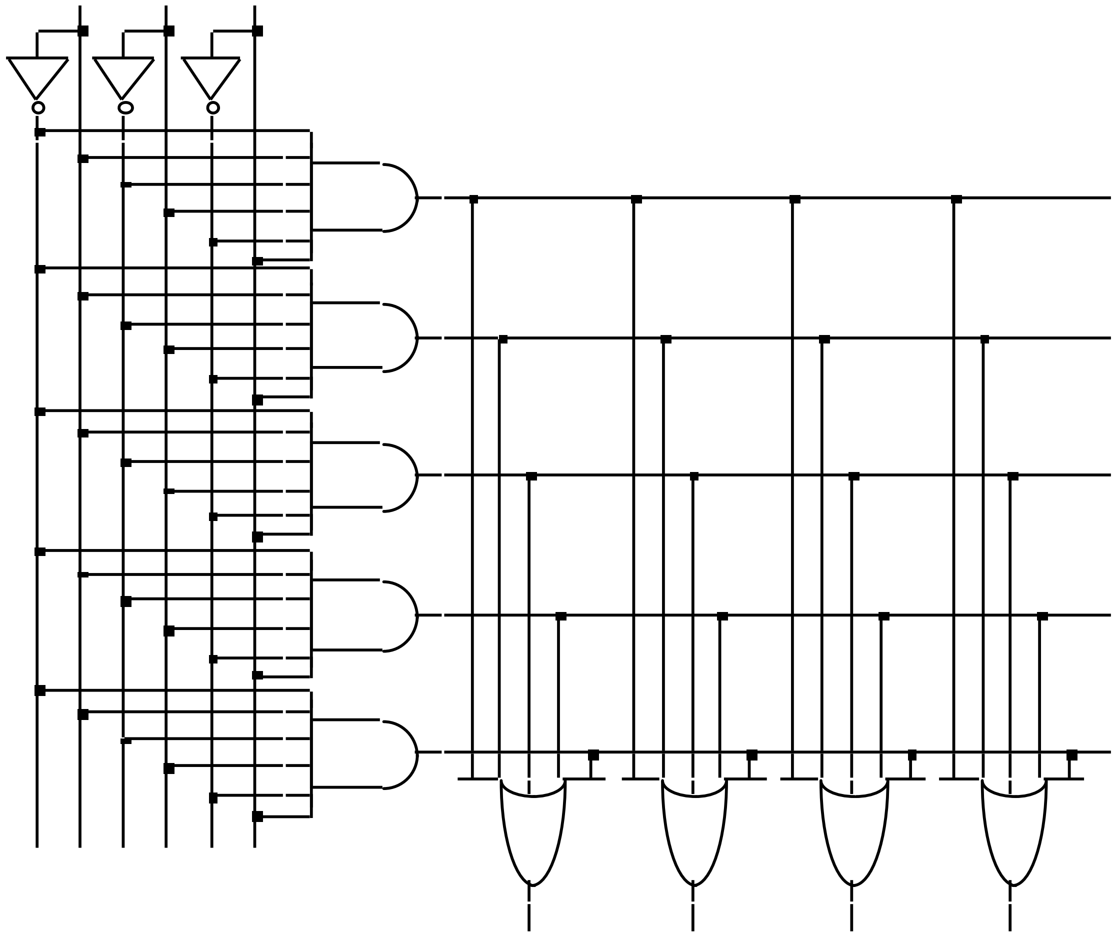
1 = uncomplemented in term
0 = complemented in term
– = does not participate

output side:

1 = term connected to output
0 = no connection to output

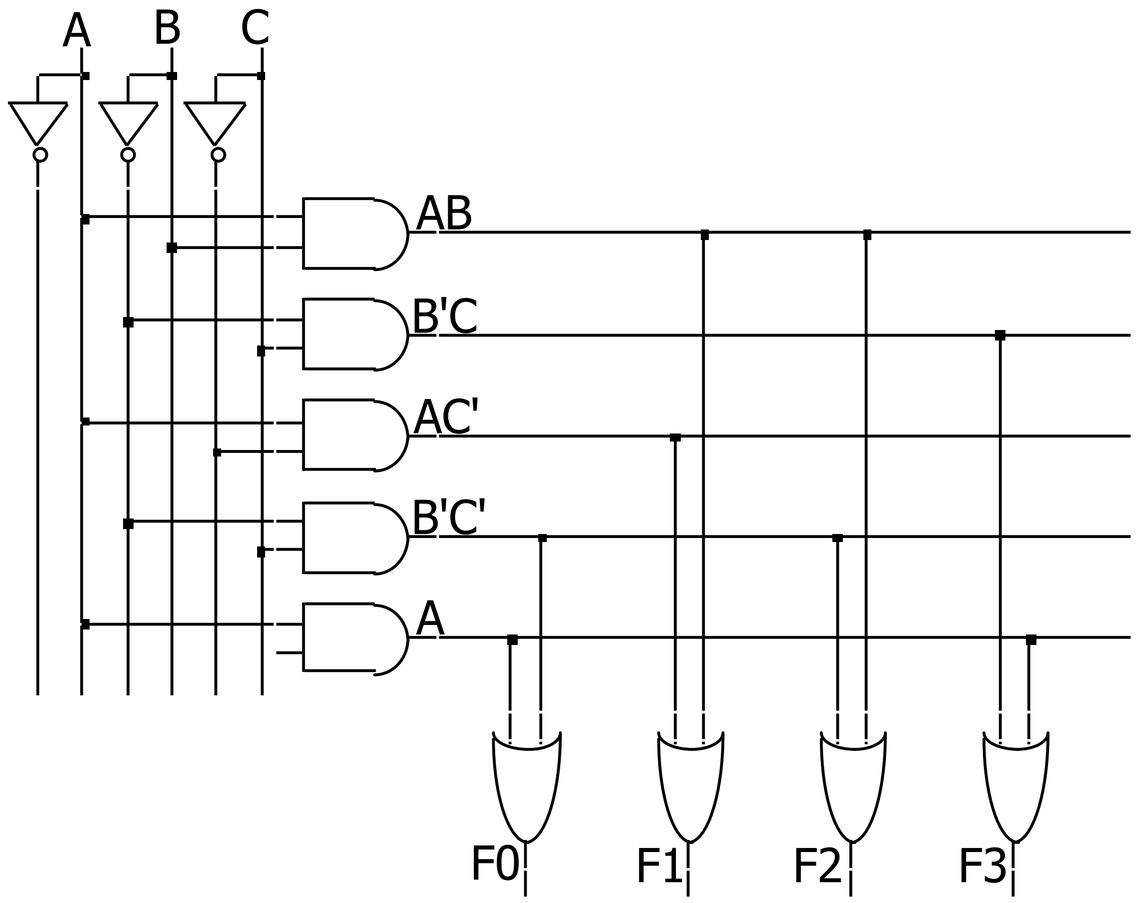
Before programming

- Assume that all possible connections are available before "programming"



After programming

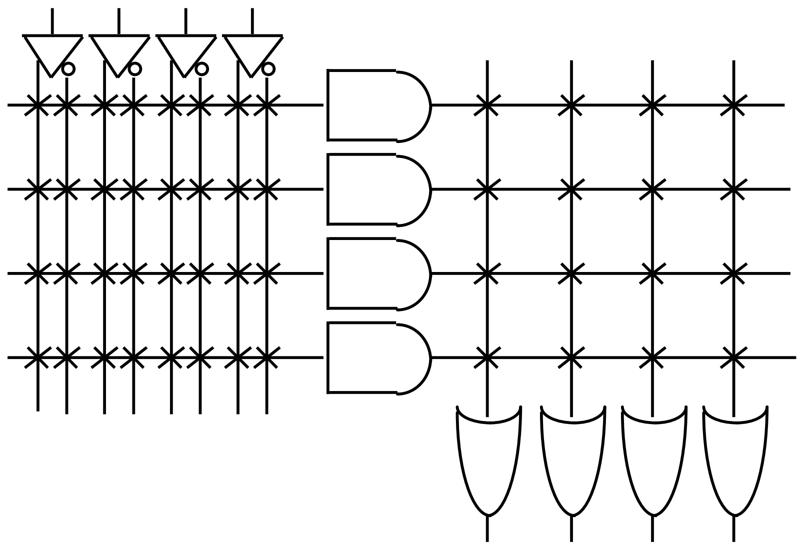
- Unwanted connections are "blown"



F0 = A + B' C'
F1 = A C' + A B
F2 = B' C' + A B
F3 = B' C + A

Alternate representation for high fan-in structures

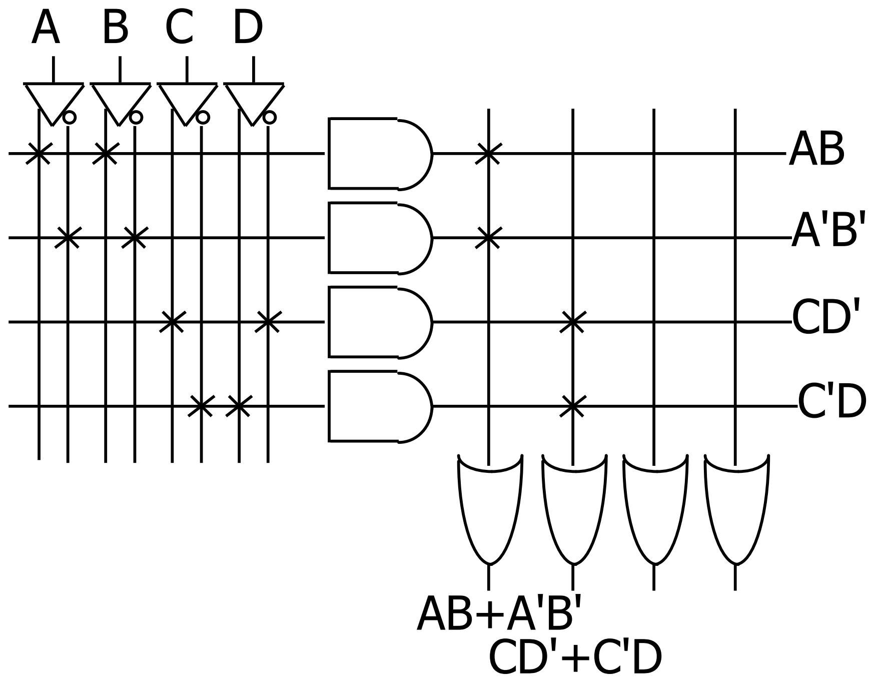
- Short-hand notation so we don't have to draw all the wires
  - × signifies a connection is present and perpendicular signal is an input to gate



notation for implementing

F0 = A B + A' B'

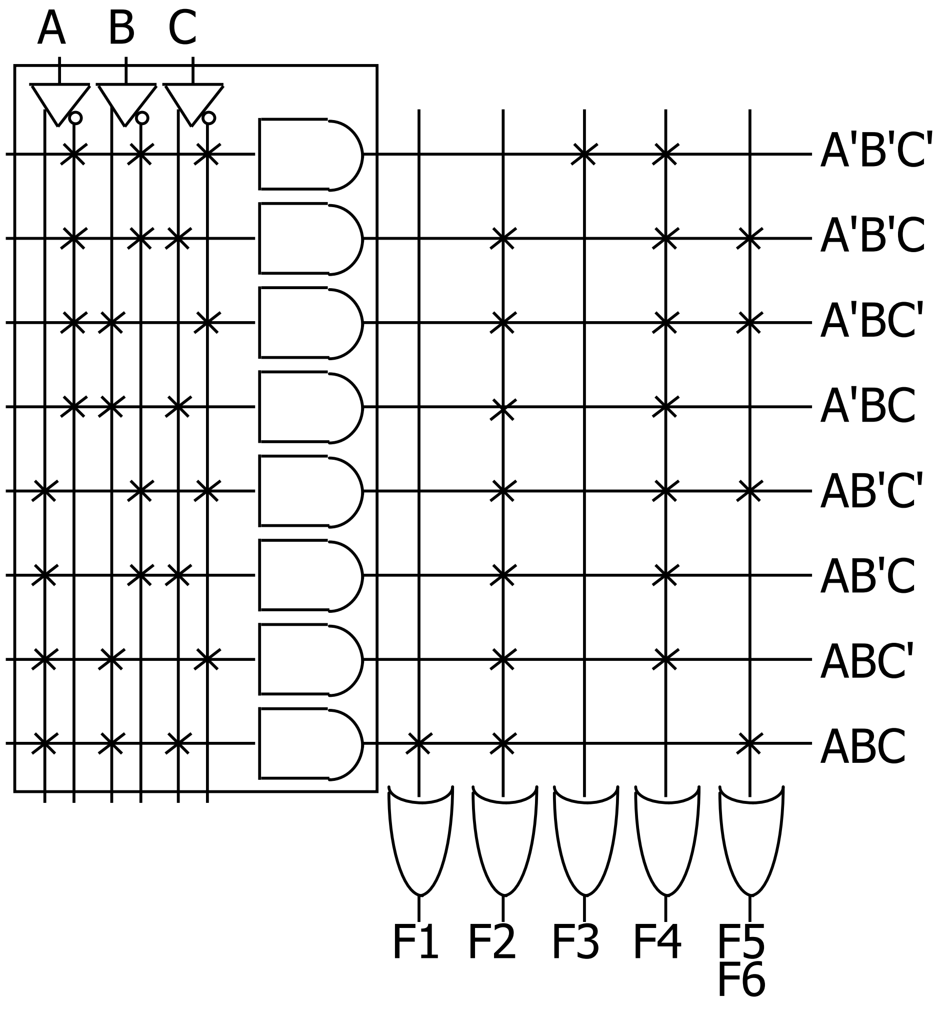
F1 = C D' + C' D



Programmable logic array example

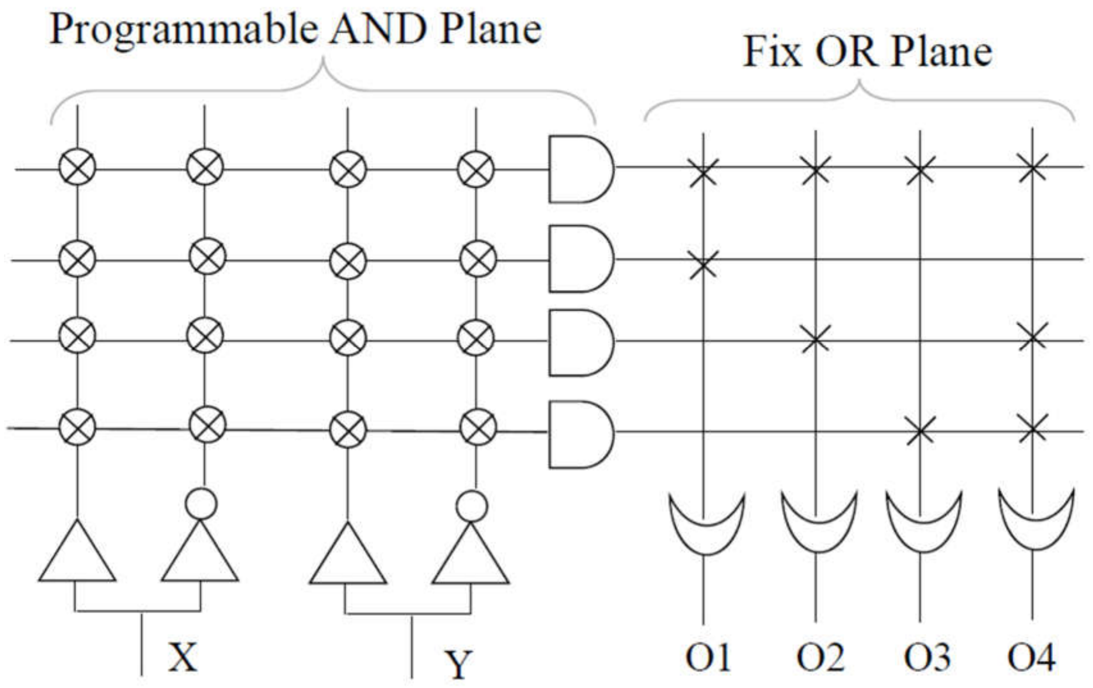
- Multiple functions of A, B, C
  - F1 = A B C
  - F2 = A + B + C
  - F3 = A' B' C'
  - F4 = A' + B' + C'
  - F5 = A xor B xor C
  - F6 = A xnor B xnor C

A	B	C	F1	F2	F3	F4	F5	F6
0	0	0	0	0	1	1	0	0
0	0	1	0	1	0	1	1	1
0	1	0	0	1	0	1	1	1
0	1	1	0	1	0	1	0	0
1	0	0	0	1	0	1	1	1
1	0	1	0	1	0	1	0	0
1	1	0	0	1	0	1	0	0
1	1	1	1	1	0	0	1	1



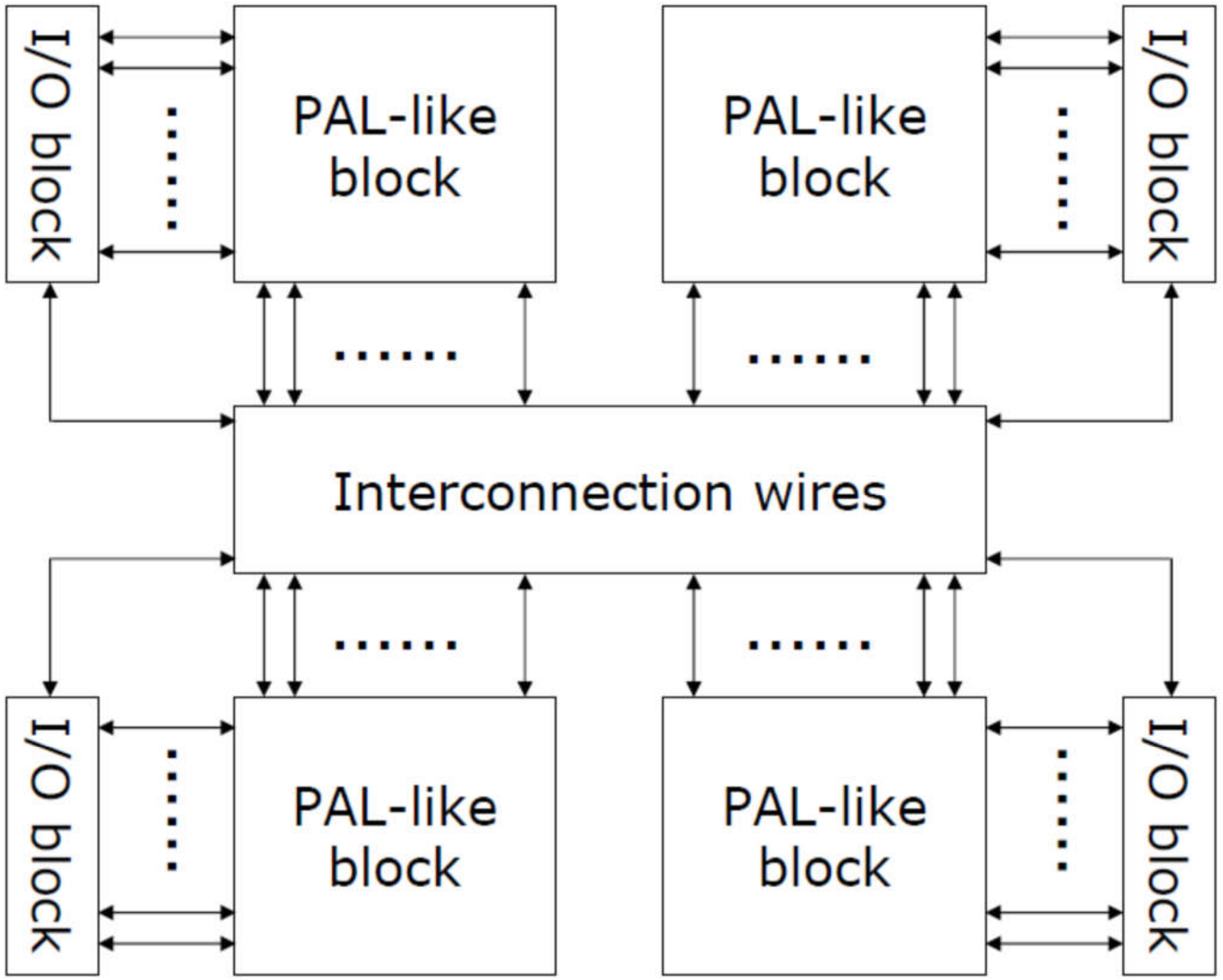
Programmable Array Logic (PAL)

- The PAL (programmable array logic) is a special case of the programmable logic array in which the AND array is programmable and the OR array is fixed.
- The basic structure of the PAL is the same as the PLA.
- Because only the AND array is programmable, the PAL is less expensive than the more general PLA, and the PAL is easier to program.



Complex Programmable Logic Device (CPLD)

- For larger designs that single PLAs or PALs cannot accommodate, a ***complex programmable logic device*** (CPLD) can be utilized

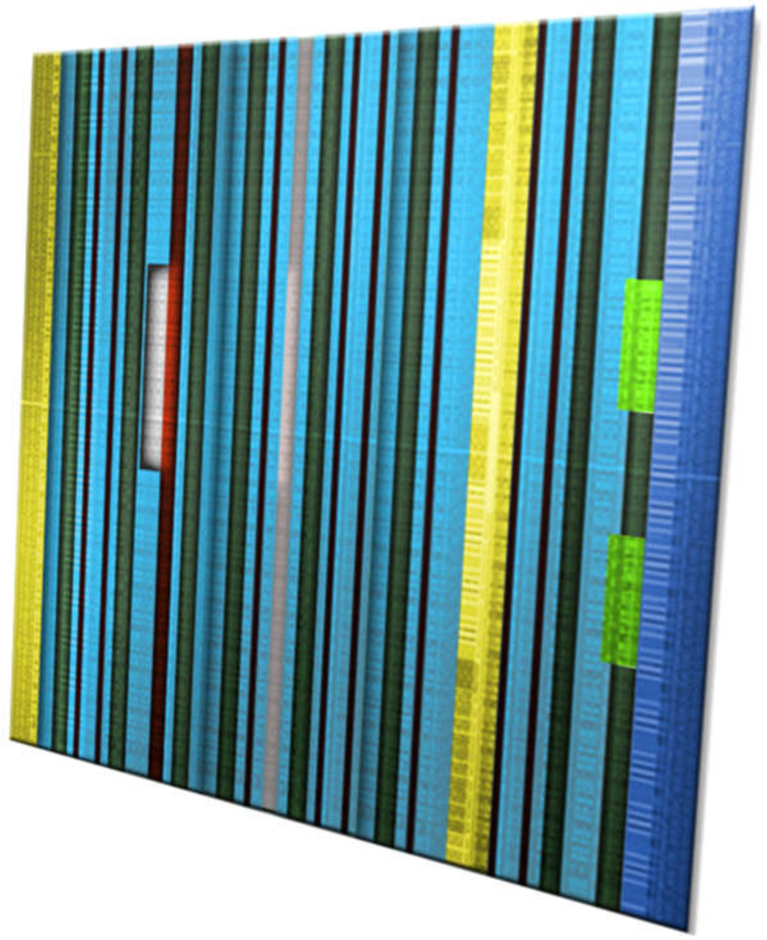


Complex Programmable Logic Devices (CPLD)

- A CPLD consists of multiple circuit blocks with internal wiring to connect the blocks together and to the pins on the chip
- Each circuit block is similar to a PAL
  - PAL-like blocks
- Commercial CPLDs have from 2 to more than 100 PAL-like blocks, with 16 macrocells in each block
  - Each macrocell is the equivalent of approximately 20 gates
  - About 20,000 equivalent gates in a CPLD of 1000 macrocells
- Can construct moderately large logic circuits in a single chip

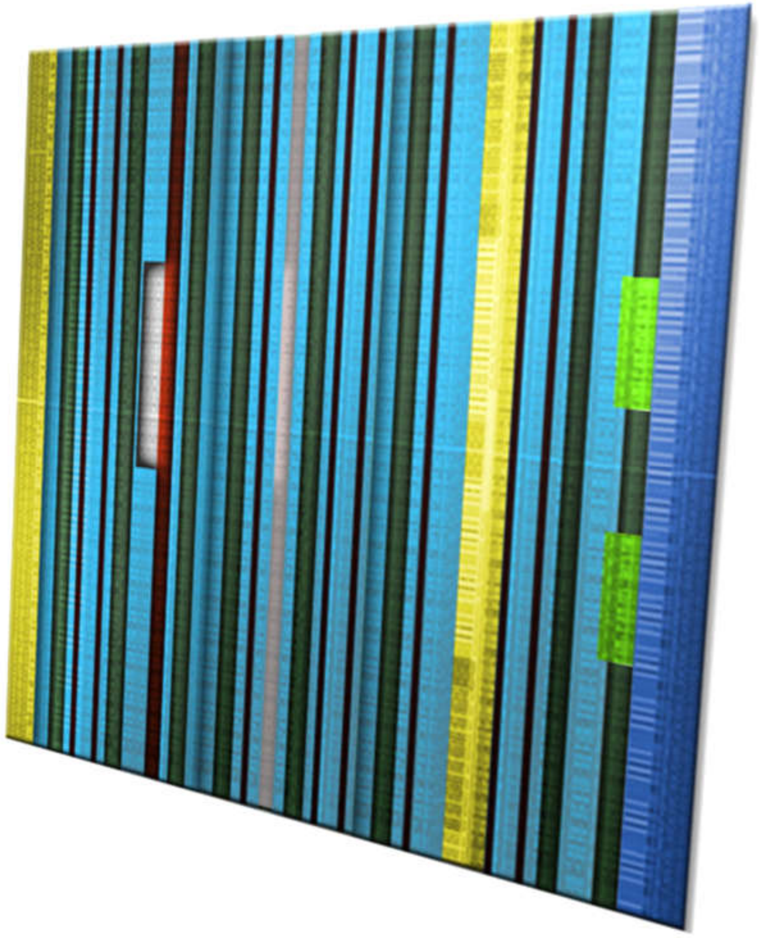
Field Programmable Gate Arrays

- To implement even larger circuits, need to use a different chip that has an even larger logic capacity:
  - **An Field Programmable Gate Array (FPGA)** introduced by *Xilinx Inc.* in 1985.
  - Misleading name - there is no array of gates.
  - Does not contain AND and OR planes.
  - Consists of general purpose and application specific logic elements, connection resources, and I/O resources.
  - Capable of implementing logic functions of millions of equivalent gates.
  - FPGAs can be programmed by the user "in the field“.



What FPGA can do?

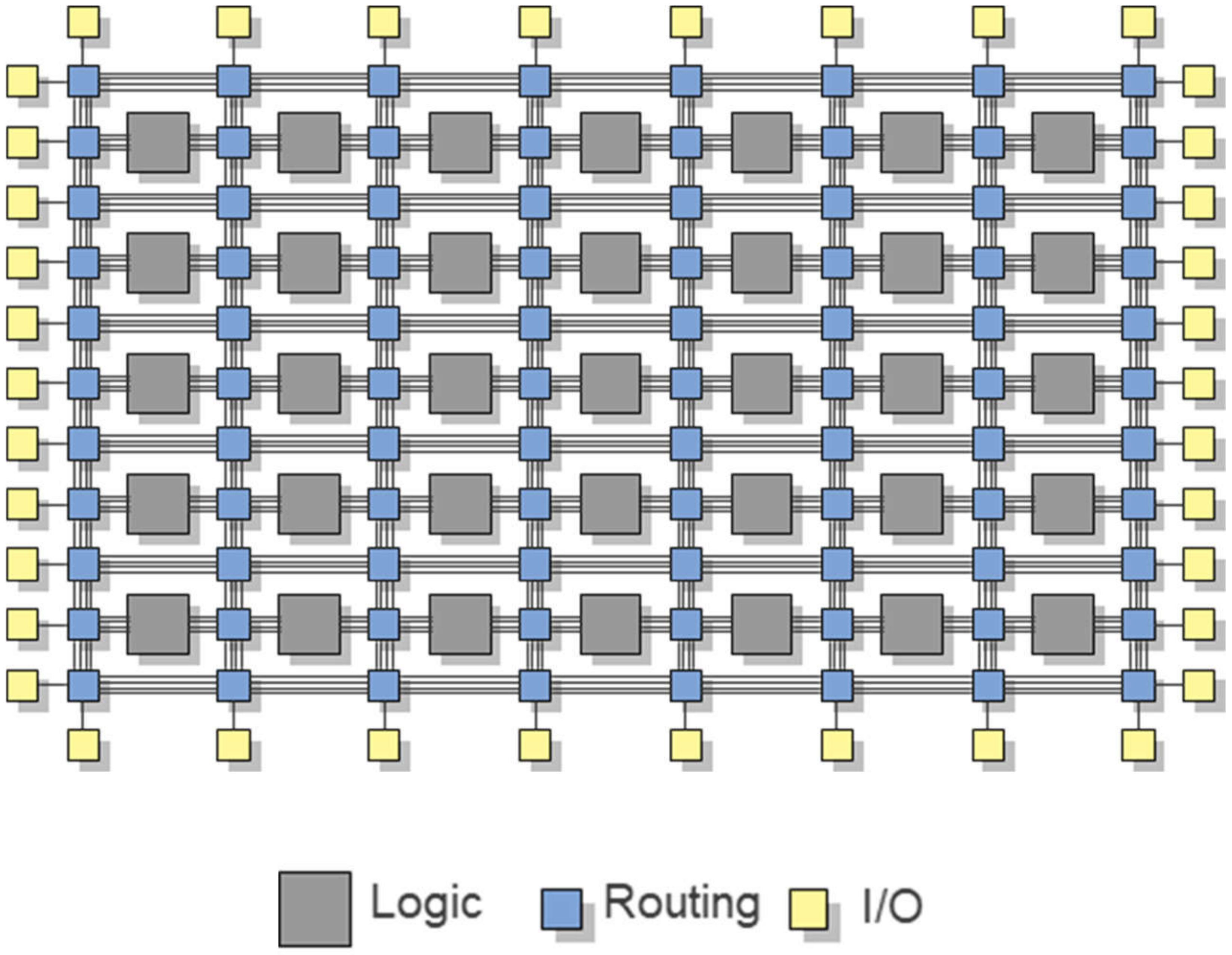
- Can you use your Intel processor as an AMD processor tomorrow and again as an Intel processor the day after tomorrow?
- You receive software updates frequently, did you ever received a hardware update? Can you upgrade your core i5 to core i7?
- Can you fix a bug in your chip after you have already received it?



How Do FPGAs Work?

- FPGAs offer **device-wide reprogrammability,** which is achieved by:
- Programmable **logic blocks**
  - Performs computation
- Programmable **routing**
  - Connects computation (logic blocks) inside the chip fabric
- Programmable **interface (I/O)**
  - Connects to the outside world
  - Various I/O standards and speeds supported

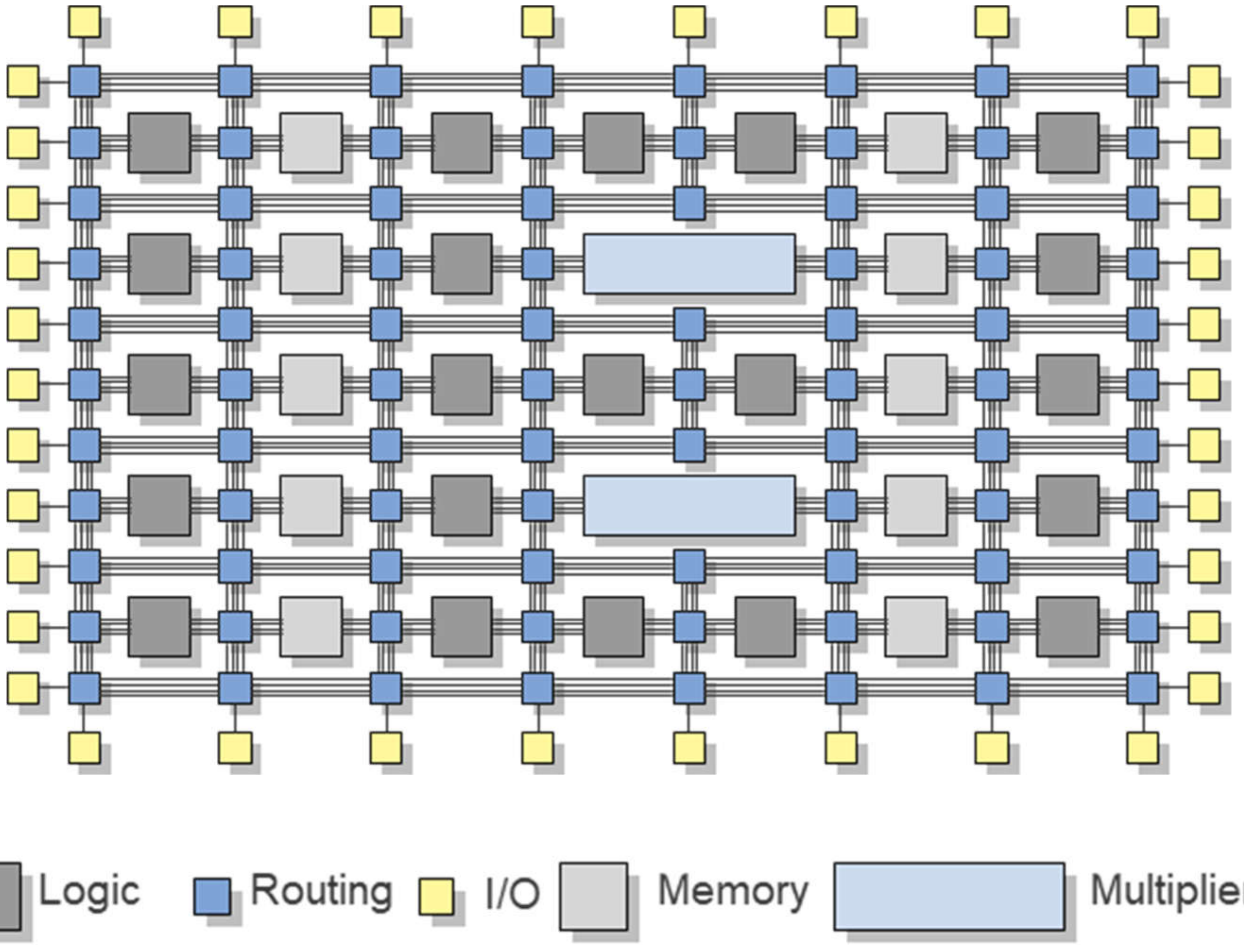
Basic FPGA Architecture



Other Components in Modern FPGAs

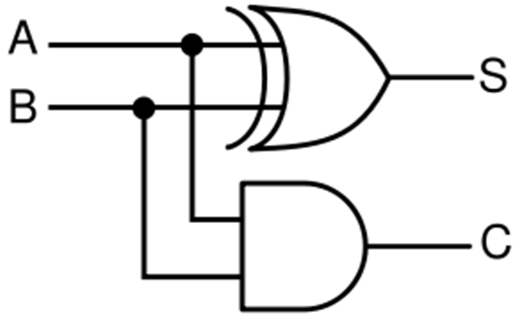
- Modern FPGAs are not limited to only LUTs, routing, and I/Os.
- **Memory blocks** (e.g., 16384 bits per block) are available in basically all modern FPGAs.
- **Hardwired multipliers** (often called DSP blocks, e.g., 18 $\times$ 18-bit multiplier) can be used to speed up multiplication
- Many FPGAs have **hardwired processor cores** (ARM Cortex-A9, Cortex-A53) embedded into the chip
- PLLs, Ethernet MACs, PCI Express, ADCs, . . .
- More on this later.

Modern FPGA Architecture



How can we implement any circuit in an FPGA?

- First, focus on combinational logic
- Combinational logic represented by truth table
- What kind of hardware can implement a truth table?
- Example: Half adder

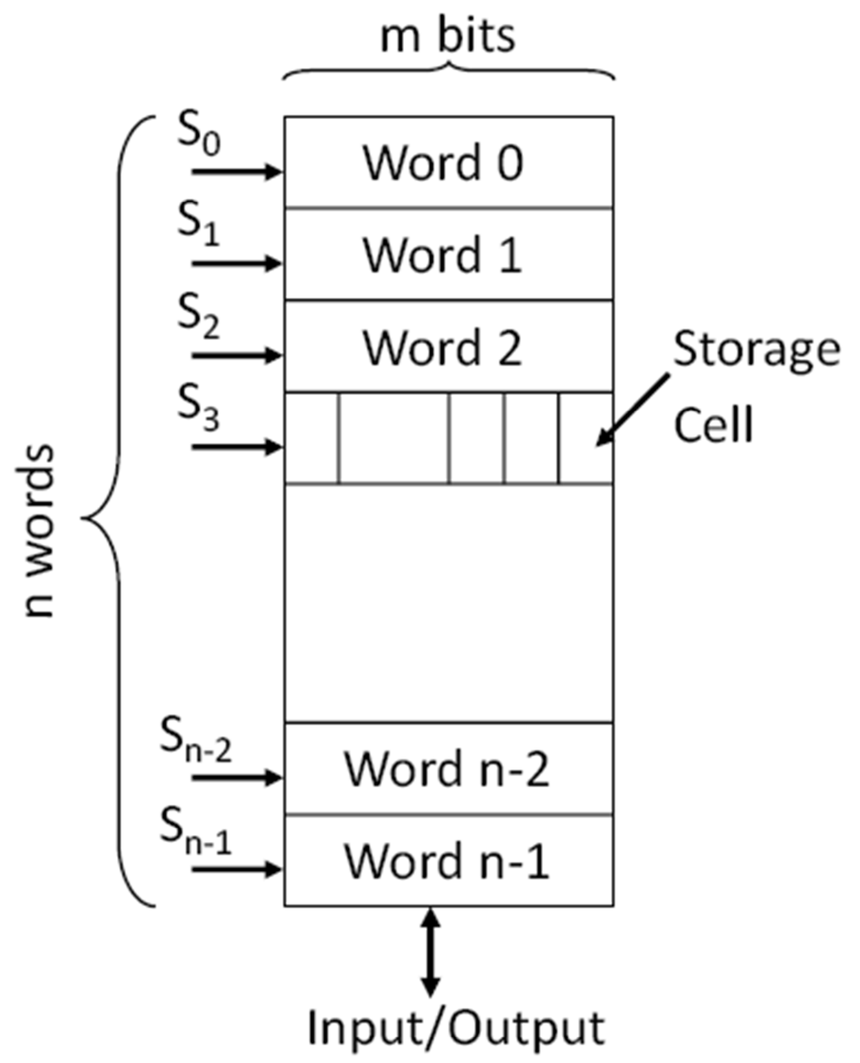


Input		Out
A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

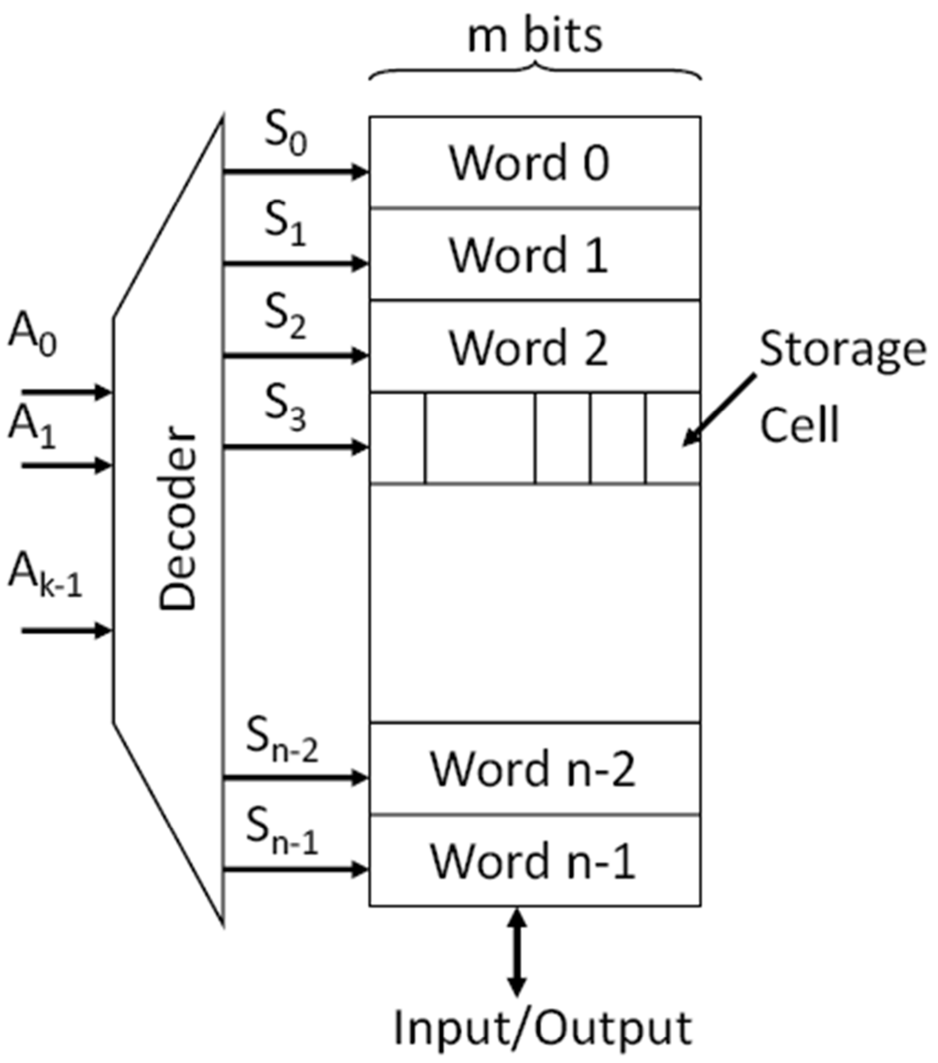
Input		Out
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Look-up-tables (LUTs)

- Implement truth table in small memories



n words → n select signals



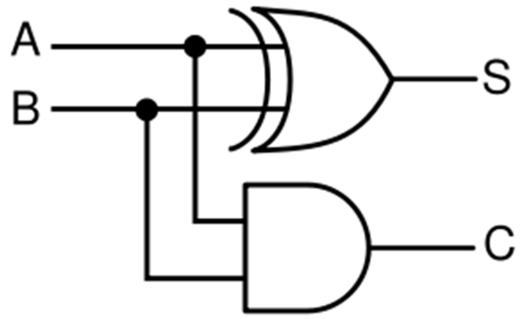
Decoder reduces # of inputs

$k = \log_2 n$

Look-up-tables (LUTs)

- Implement truth table in small memories

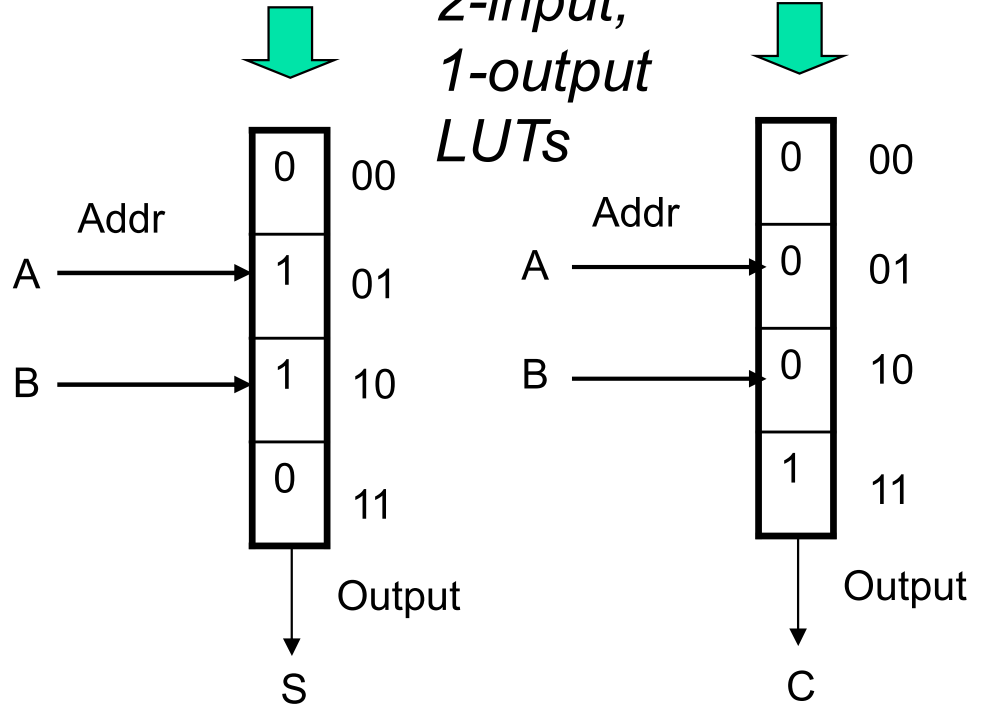
  Usually SRAM



A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

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

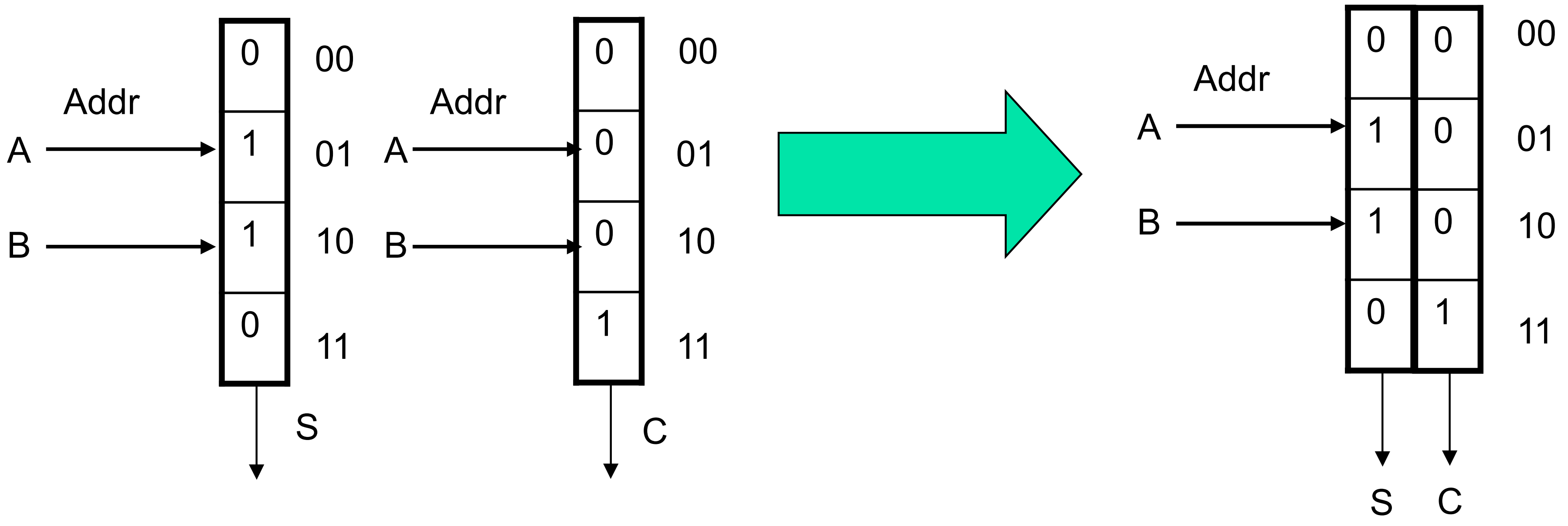
2-input, 1-output LUTs



Logic inputs connect to address inputs, logic output is memory output

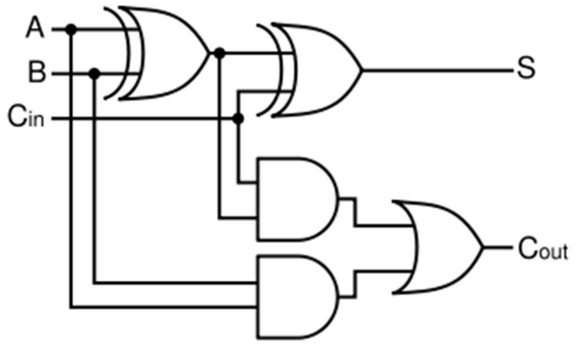
Look-up-tables (LUTs)

- Alternatively, could have used a 2-input, 2-output LUT
    - Outputs commonly use same inputs



Look-up-tables (LUTs)

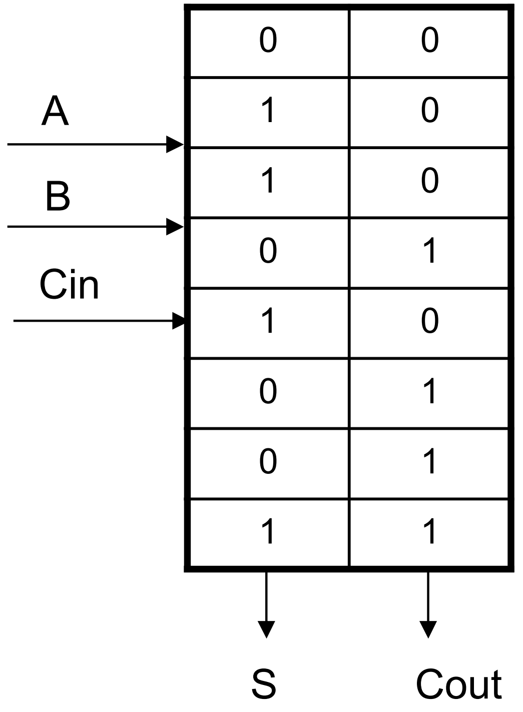
- The great thing about **an n-input LUT** is that it **can implement any possible n-input combinatorial logic function.**
- Full adder example → 3-input, 2-ouput LUT



Truth Table

Inputs			Outputs	
A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

3-input, 2-output LUT

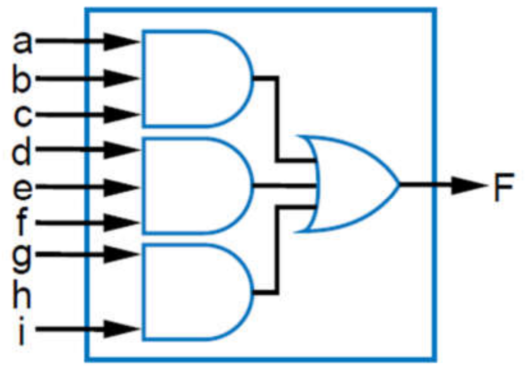


Look-up-tables (LUTs)

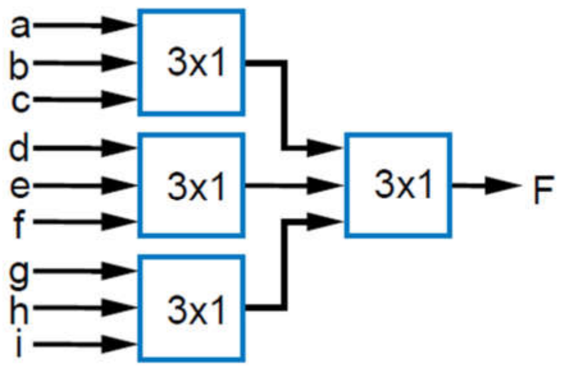
- Why aren't FPGAs just a big LUT?
  - Size of truth table grows exponentially based on # of inputs
    - 3 inputs = 8 rows, 4 inputs = 16 rows, 5 inputs = 32 rows, etc.
  - Same number of rows in truth table and LUT
  - LUTs grow exponentially based on # of inputs
- Number of SRAM bits in a LUT = $2^i * o$
  - i = # of inputs, o = # of outputs
  - Example: 64 input combinational logic with 1 output would require 2^{64} SRAM bits
    - 1.84×10^{19}
- Clearly, not feasible to use large LUTs
  - So, how do FPGAs implement logic with many inputs?

Look-up-tables – Partitioning

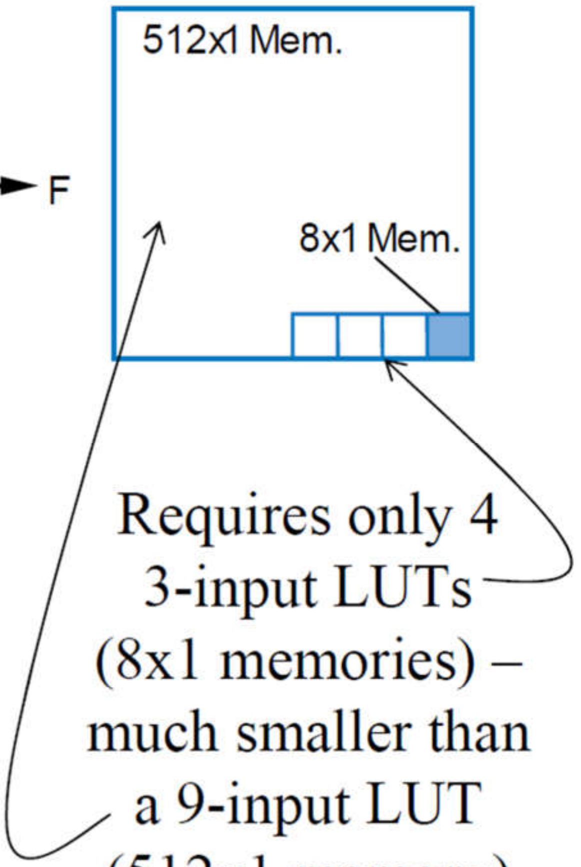
- Fortunately, we can map circuits onto multiple LUTs
  - Divide circuit into smaller circuits that fit in LUTs (same # of inputs and outputs)
  - Example: 9-input circuit
  - Partitioning among smaller LUTs is more size efficient



Original 9-input circuit



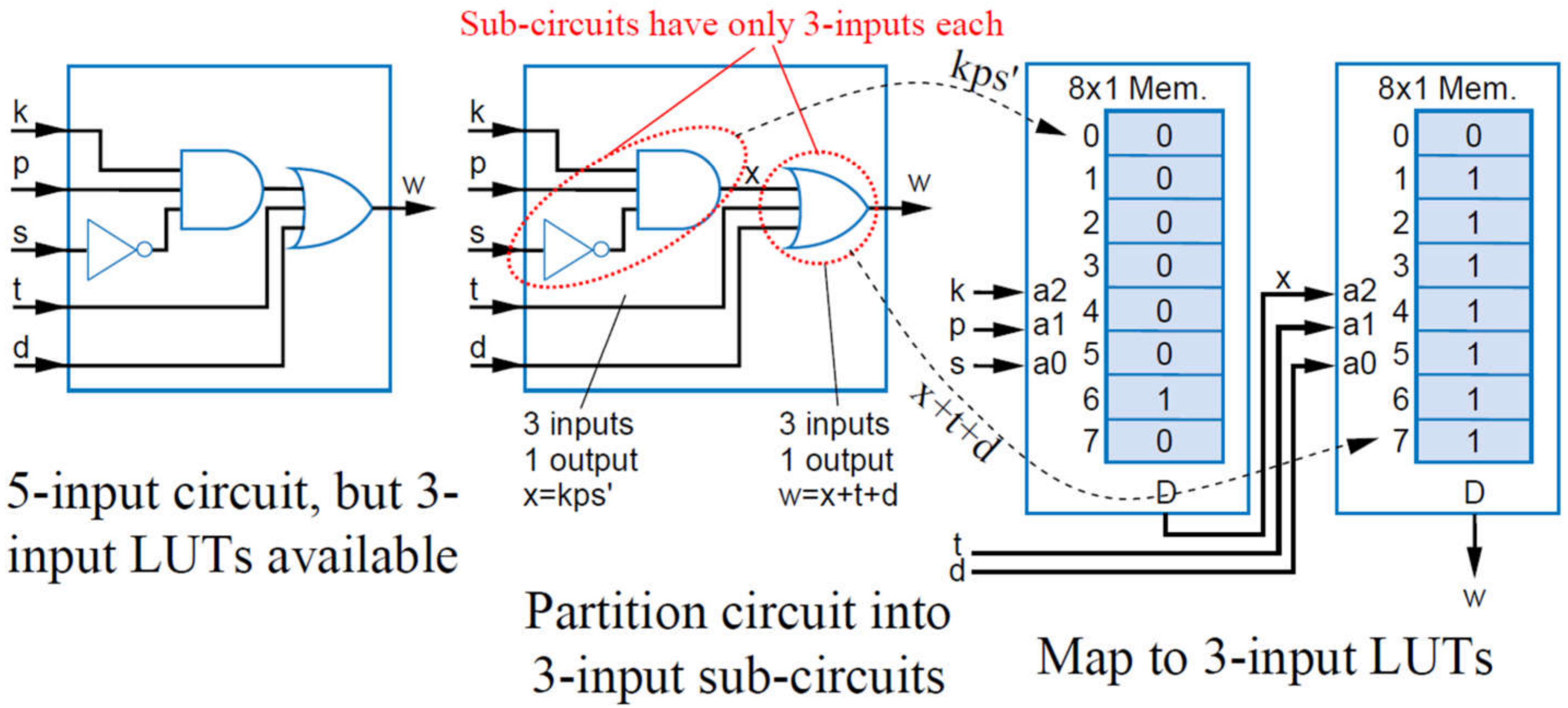
Partitioned among 3x1 LUTs



Requires only 4 3-input LUTs (8x1 memories) – much smaller than a 9-input LUT (512x1 memory)

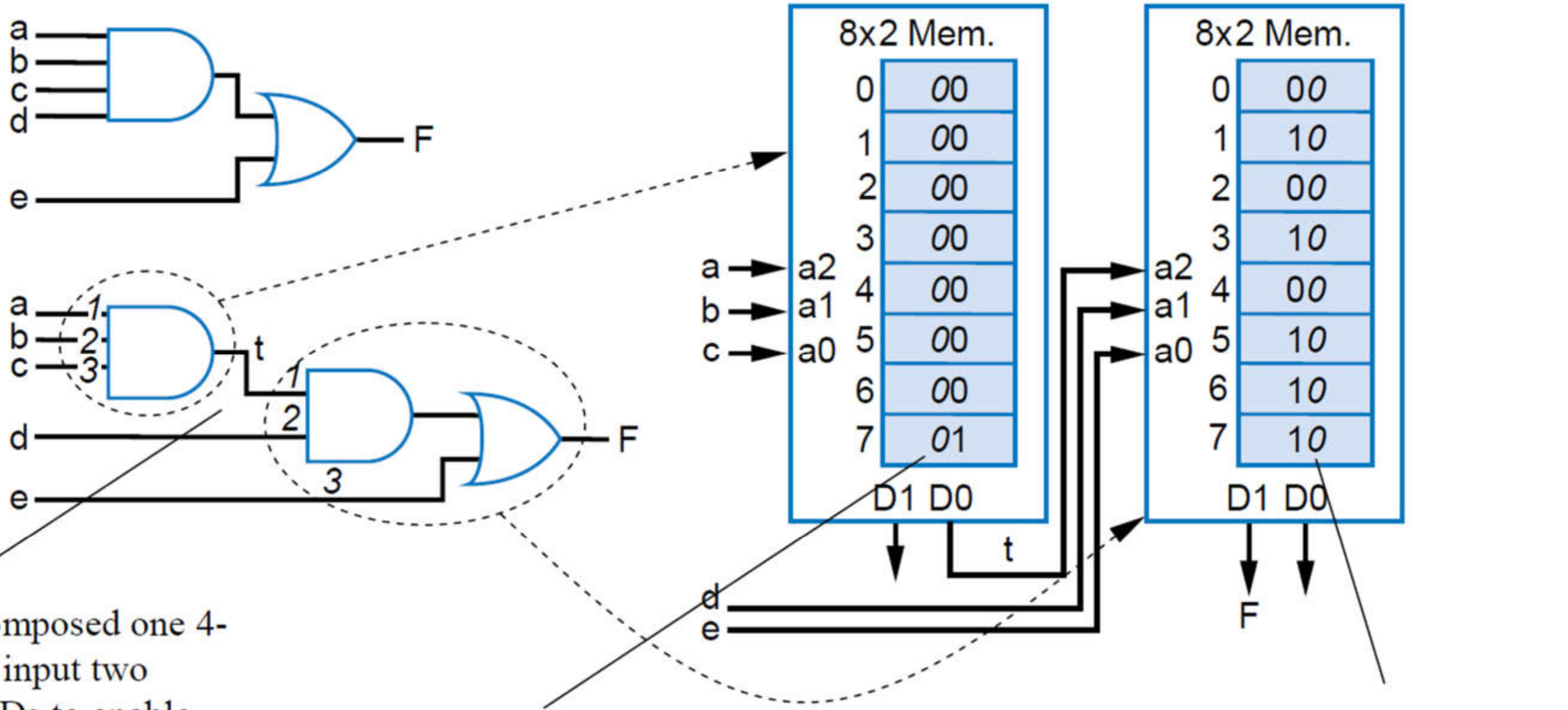
Partitioning LUTs – Example 2

- FPGAs thus have numerous small (3, 4, 5, or even 6-input) LUTs
- If circuit has more inputs, must partition circuit among LUTs



Partitioning LUTs – Example 3

- LUT typically has 2 (or more) outputs, not just one
- Partitioning a circuit among 3-input 2-output lookup tables



(Note: decomposed one 4-input AND input two smaller ANDs to enable partitioning into 3-input sub-circuits)

First column unused; second column implements AND

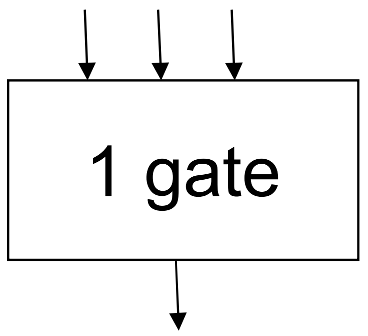
Second column unused; first column implements AND/OR sub-circuit

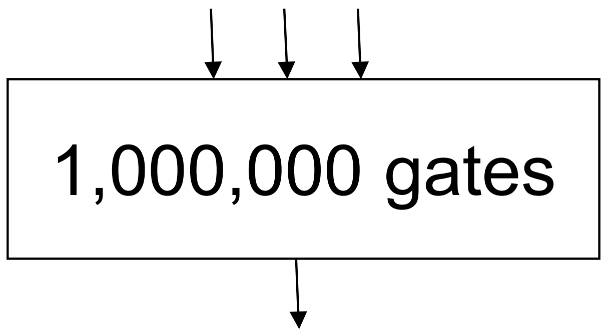
Look-up-tables (LUTs)

- Important Point

  The number of gates in a circuit has no effect on the mapping into a LUT

  - All that matters is the number of inputs and outputs
  - Unfortunately, it isn't common to see large circuits with a few inputs





Both of these circuits can be implemented in a single 3-input, 1-output LUT

Look-up-tables

- What if circuit doesn’t map perfectly?
  - More inputs in LUT than in circuit
    - Truth table handles this problem
    - Unused inputs are ignored
  - More outputs in LUT than in circuit
    - Extra outputs simply not used
      - Space is wasted, so should use multiple outputs whenever possible
- Many successful FPGA architectures are based on 4-input LUT.
- Mixture of different LUT sizes is possible but uniformity and regularity are preferred by synthesis tools.
- Very fast moving sector so things may change in the near future.

Typical Applications for FPGAs

- Communications, both wired and wireless
- Industrial control systems
- Robotics, motor control
- Defence and Military
- Healthcare, especially imaging
- “Big Data” applications (genetics, financial)
- Artificial Intelligence and Machine Learning
- ASIC prototyping

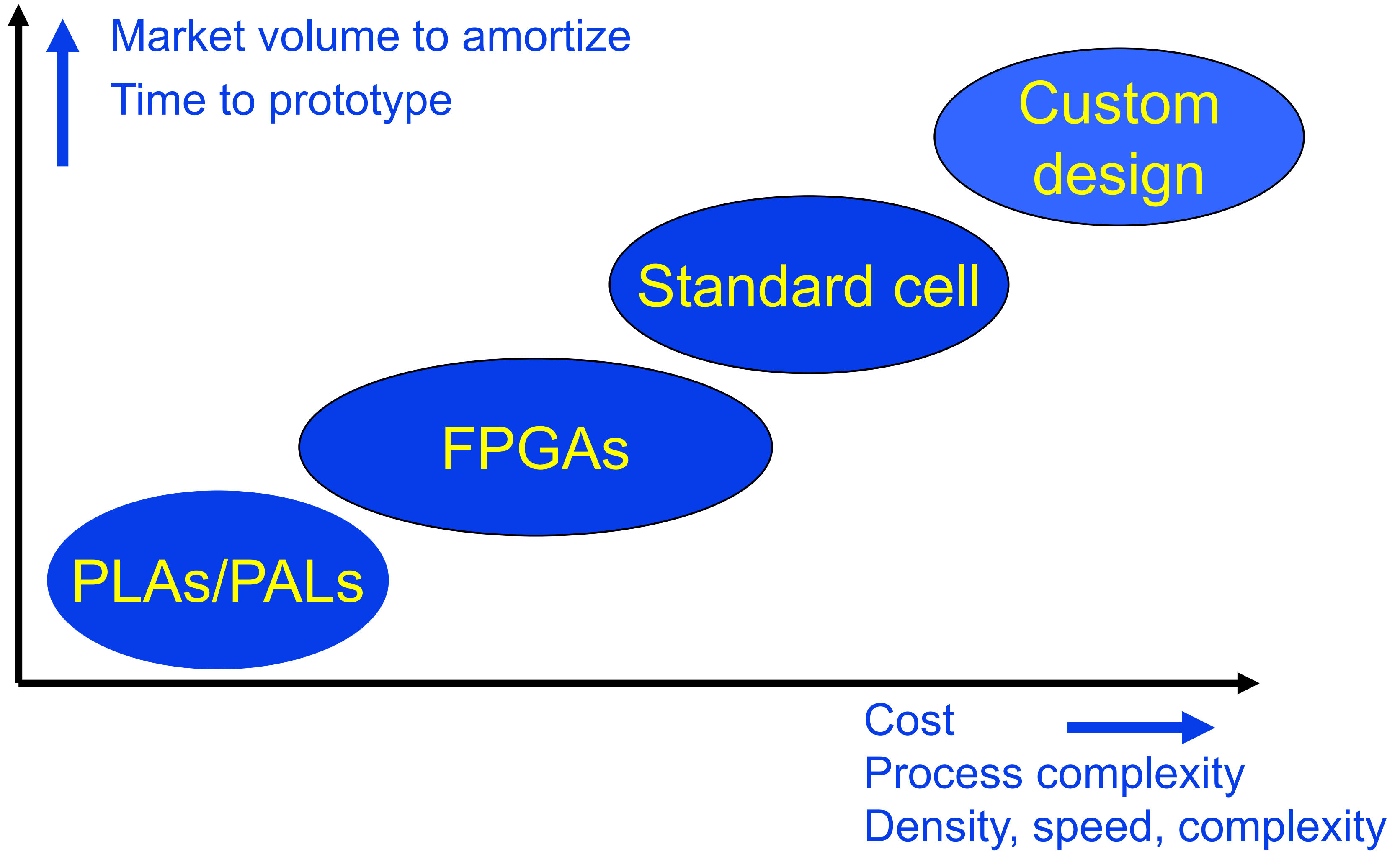
Advantages of FPGAs

- Off-the-shelf
- Flexibility
- Upgradability
- Low cost for small projects
- Short Time to the market
- Prototyping
- High performance DSP algorithms

Disadvantages of FPGAs

- ❑ Wastage of on-chip resources in programmable routing
- ❑ High power consumption compared to the same implementation on full custom design
- ❑ Low operating speed compared to the same implementation on full custom design

COMPARISON OF TECHNOLOGIES



Acknowledgments

- Credit is acknowledged where credit is due. Please refer to the full list of references.