Introduction to

Digital systems

Syllabus

Prérequis
Logic (IMA3)
Electronics (transistors et AOP) (IMA3)

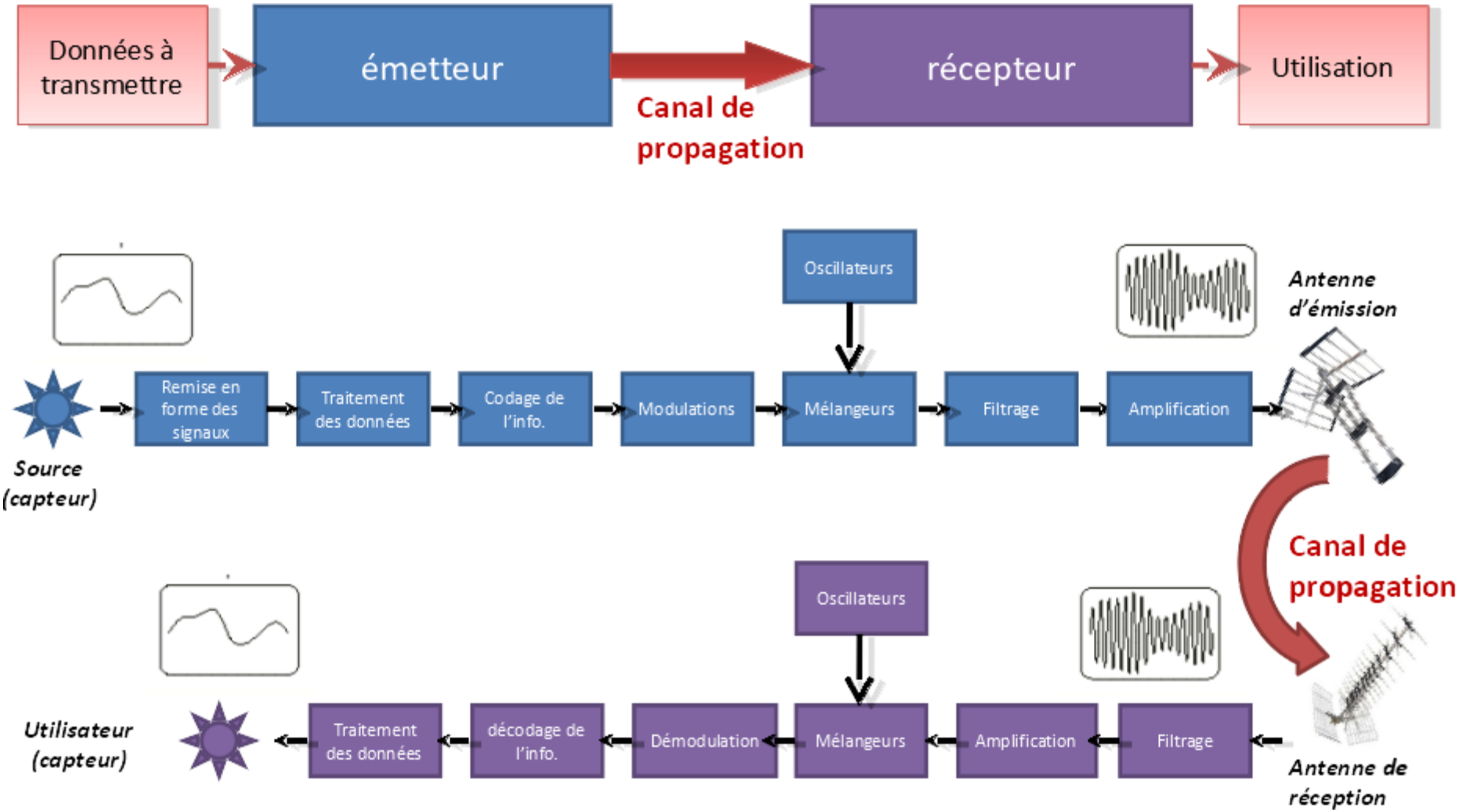
Mots-clés
Circuits numériques, FPGA, Flot de conception, VHDL
Acquisition de données, ADC, DAC

Objectifs
Être capable de choisir entre cibles logicielles (µP, µC, DSP) et cibles matérielles (FPGA, CPLD)
Correctement dimensionner et choisir les convertisseurs de données
Décrire un système électronique complexe en VHDL
Comprendre et utiliser le flot de conception pour FPGA

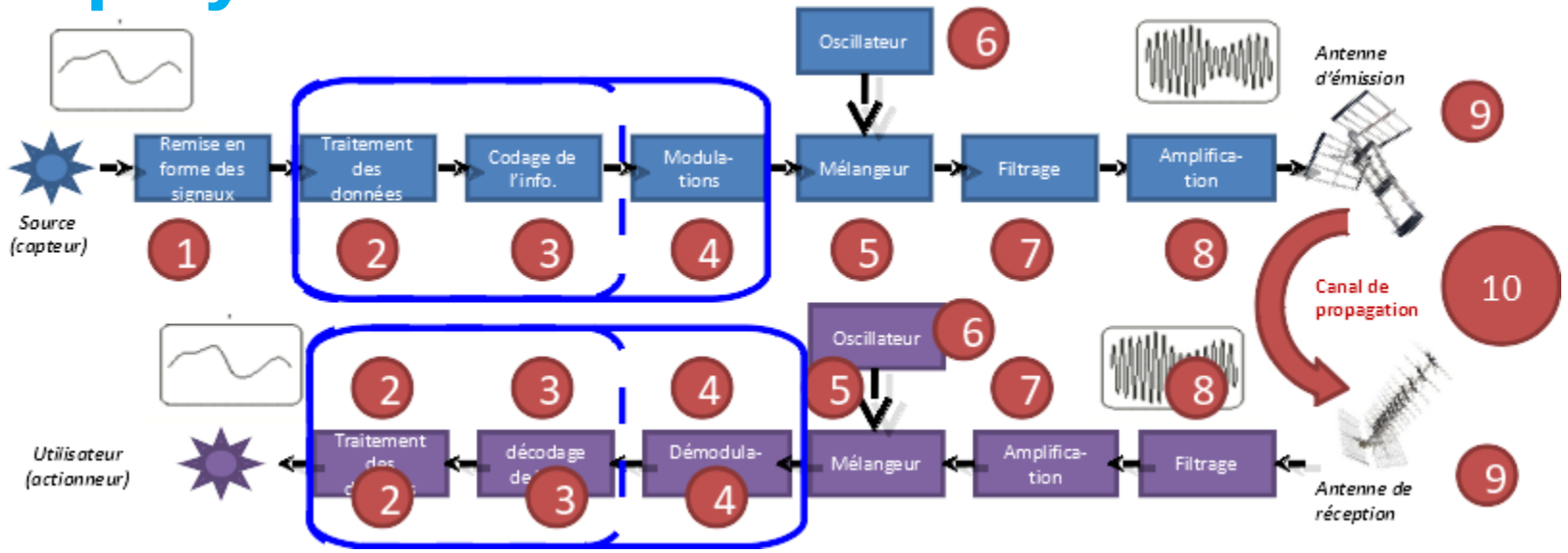
Support pédagogiques
Documents de cours et annexes disponibles sur l'intranet IMA

Intégrité académique
(*Rappel*) Les travaux soumis doivent être les vôtres. Les éventuels emprunts (sources Internet, livre, articles, …) sont autorisés sous réserve d'être correctement cités.

Foreplay


Données à transmettre
émetteur
Canal de propagation
récepteur
Utilisation
Oscillateurs
Antenne d'émission
Remise en forme des signaux
Traitement des données
Codage de l'info.
Modulations
Mélangeurs
Filtrage
Amplification
Source (capteur)
Canal de propagation
Oscillateurs
Utilisateur (capteur)
Traitement des données
décodage de l'info.
Démodulation
Mélangeurs
Amplification
Filtrage
Antenne de réception

Foreplay

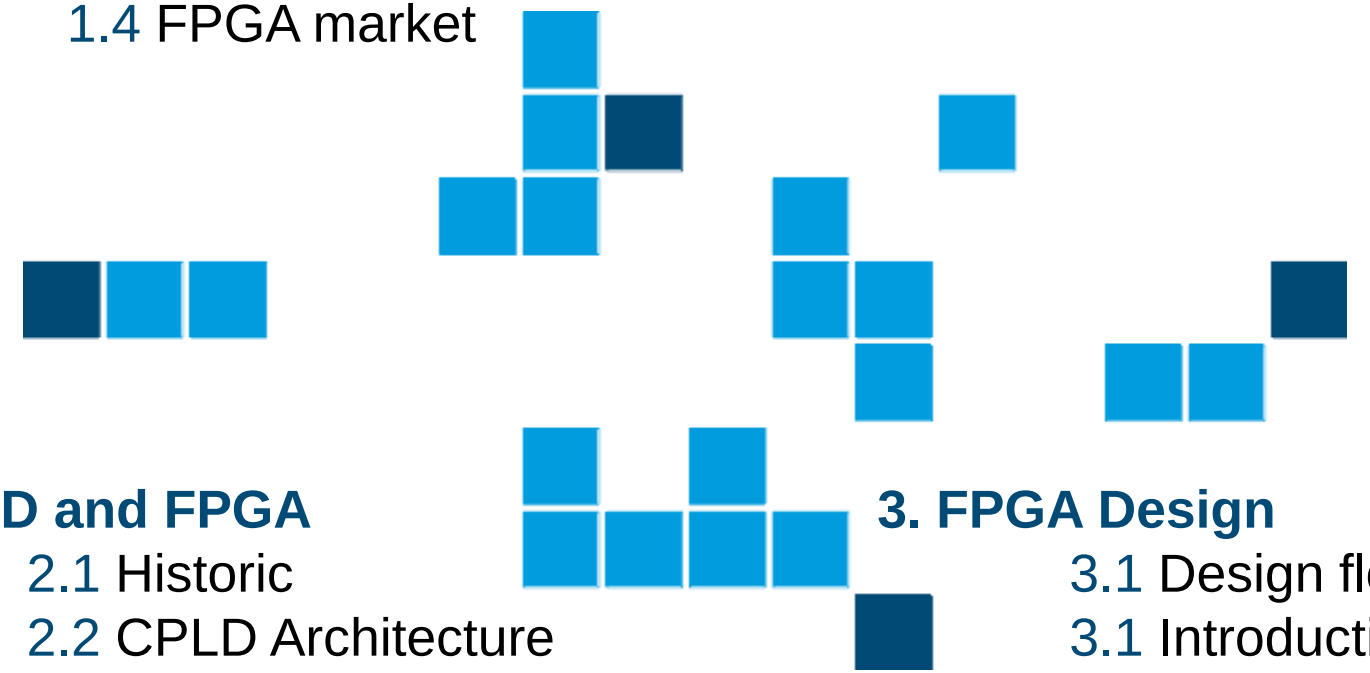


1. Codage de source : électronique analogique (IMA3 et 4) – CAN, CNA (IMA4) – Remise en forme de signaux (IMA3-IMA4) – logique (IMA3).
2. Traitement des données : µcontrolleur/µprocesseur (IMA3) – filtrage numérique (IMA4) – traitement du signal (IMA4) – segmentation matérielle/logicielle, CNP (IMA4)
3. Codage : codage de l'information (IMA4-SC) – CNP (IMA4)
4. Modulateurs : électronique (IMA4) – TIM (IMA4-SC) – ST (IMA4-SC)
5. Mélangeurs : TIM (IMA4-SC) – ST (IMA4-SC) – Conception (IMA5-SC)
6. Oscillateur : TIM (IMA4-SC) – ST (IMA4-SC) – Conception (IMA5-SC)
7. Filtre : électronique (IMA4) – TIM (IMA4-SC)
8. Amplificateur/LNA : ST (IMA4-SC) – Conception (IMA5-SC)
9. Antenne : ST (IMA4-SC) – DSC (IMA5-SC)
10. Canal de propagation : TIM (IMA4-SC) – DSC (IMA5-SC)

Outline

1. Introduction

1.1 Different types of digital components
1.2 Advantages and drawbacks of each type
1.3 Selected applications
1.4 FPGA market



2. CPLD and FPGA

2.1 Historic
2.2 CPLD Architecture
2.3 FPGA Architecture
2.3 Clock and timing management
2.4 Other embedded ressources

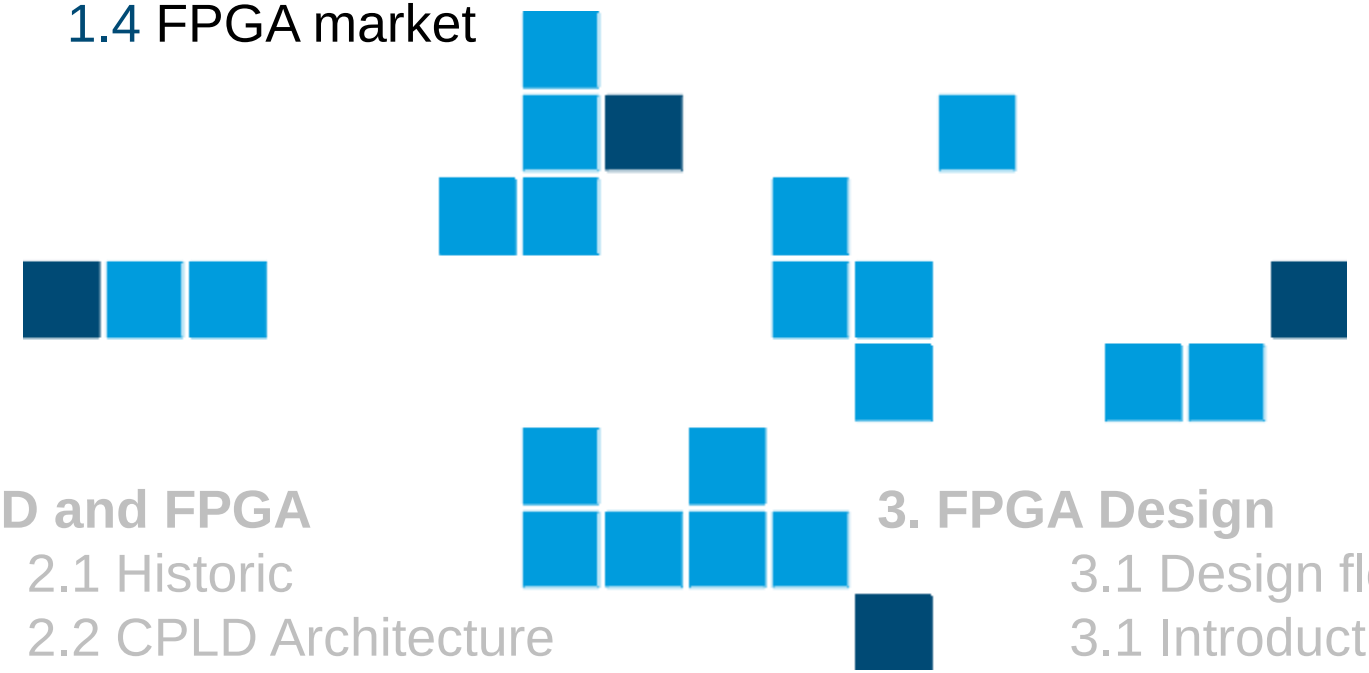
3. FPGA Design

3.1 Design flow
3.1 Introduction to VHDL
3.2 Simulations
3.3 Finite State Machines

Outline

1. Introduction

1.1 Different types of digital components
1.2 Advantages and drawbacks of each type
1.3 Selected applications
1.4 FPGA market



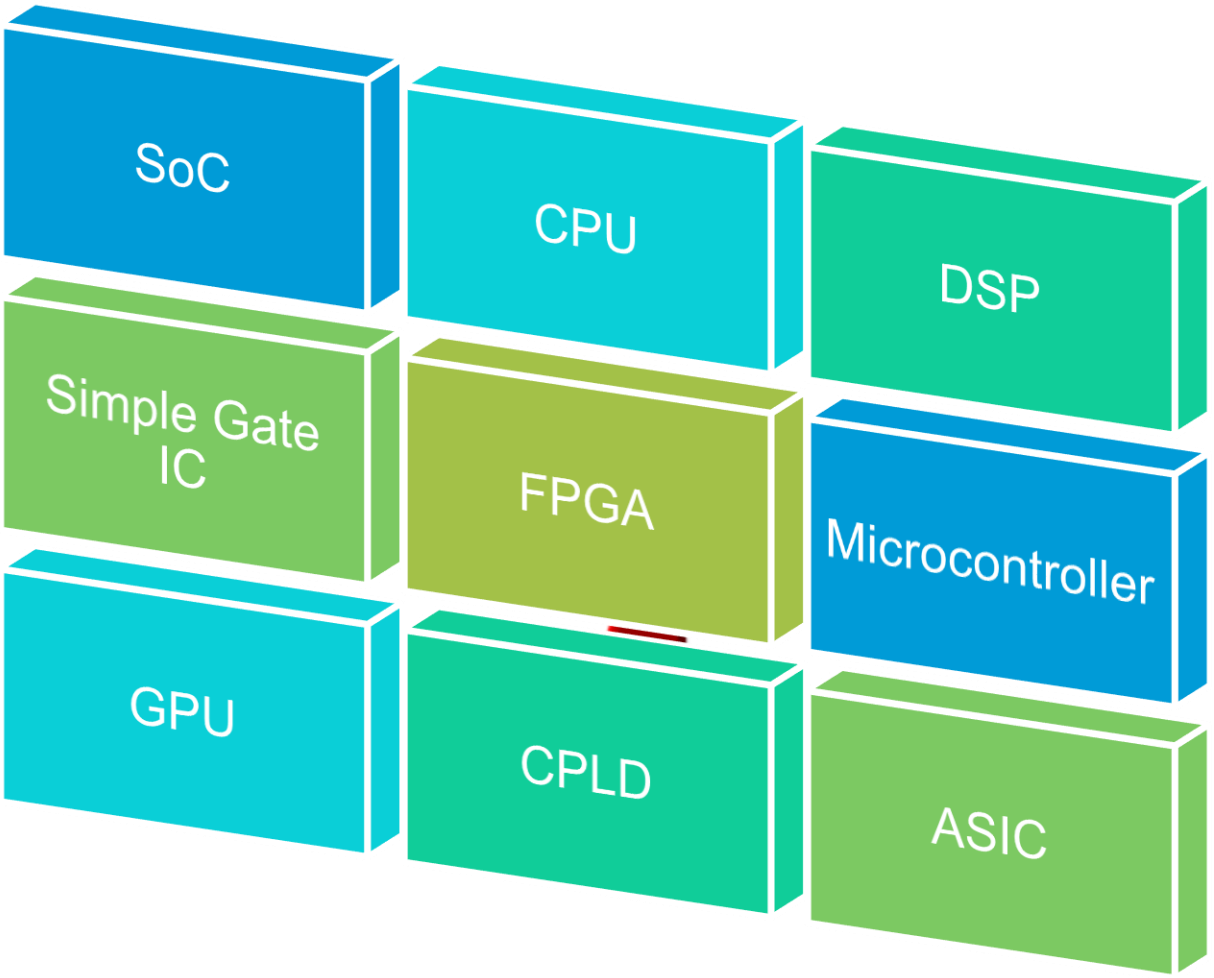
2. CPLD and FPGA

2.1 Historic
2.2 CPLD Architecture
2.3 FPGA Architecture
2.3 Clock and timing management
2.4 Other embedded ressources

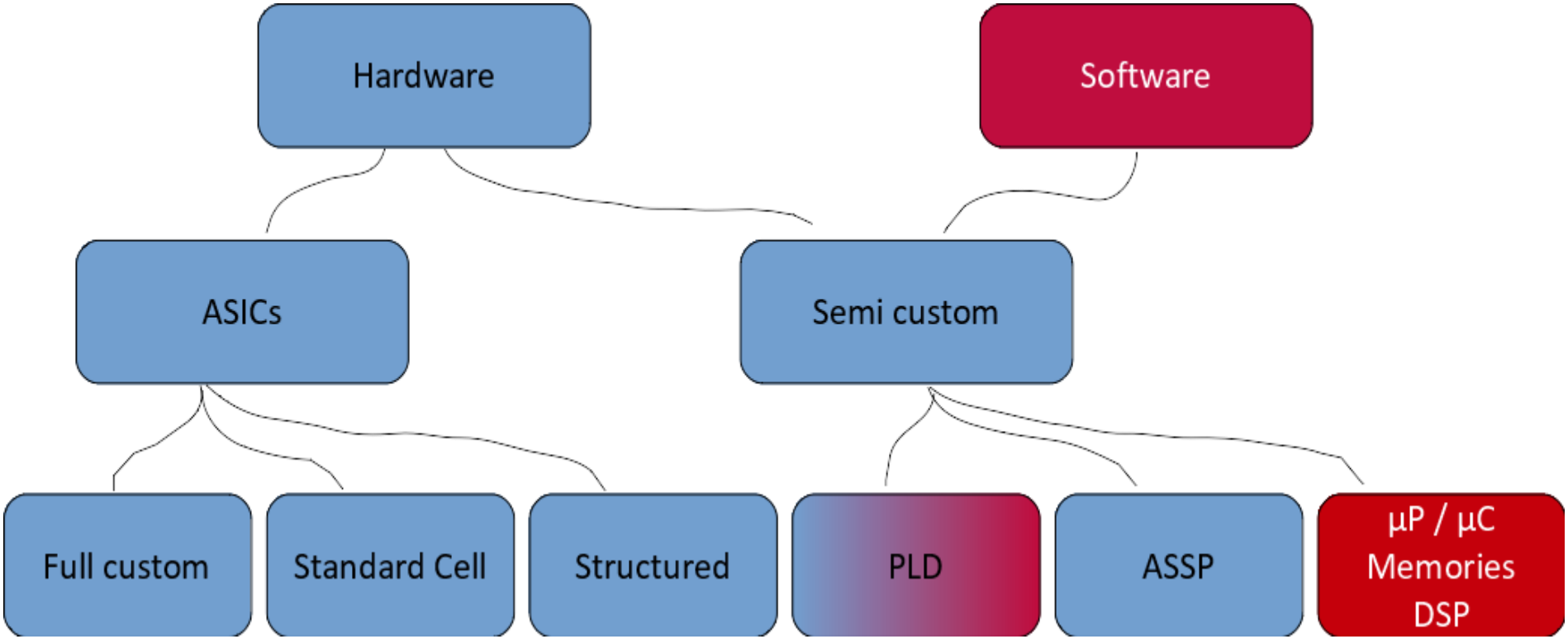
3. FPGA Design

3.1 Design flow
3.1 Introduction to VHDL
3.2 Simulations
3.3 Finite State Machines

Different types of digital components



Introduction



ASIC : *Application Specific Integrated Circuit*
PLD : *Programmable Logic Device*
ASSP : *Application Specific Standard Product*
DSP : Digital Signal Processor
SoC : System on Chip

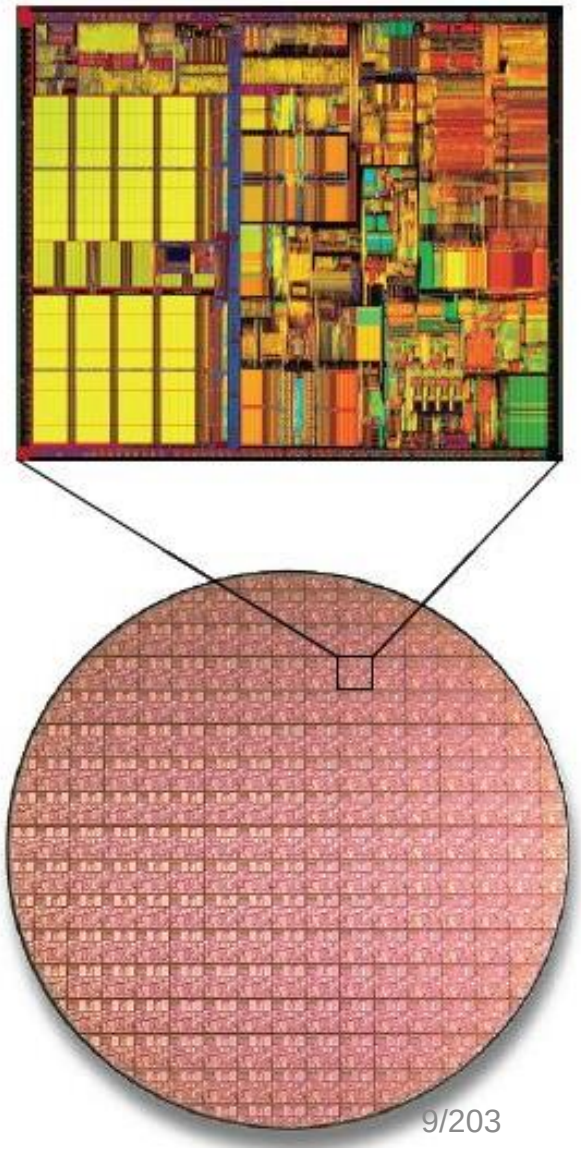
ASIC – Application Specific Integrated Circuit

Full custom

All mask levels are freely modifiable

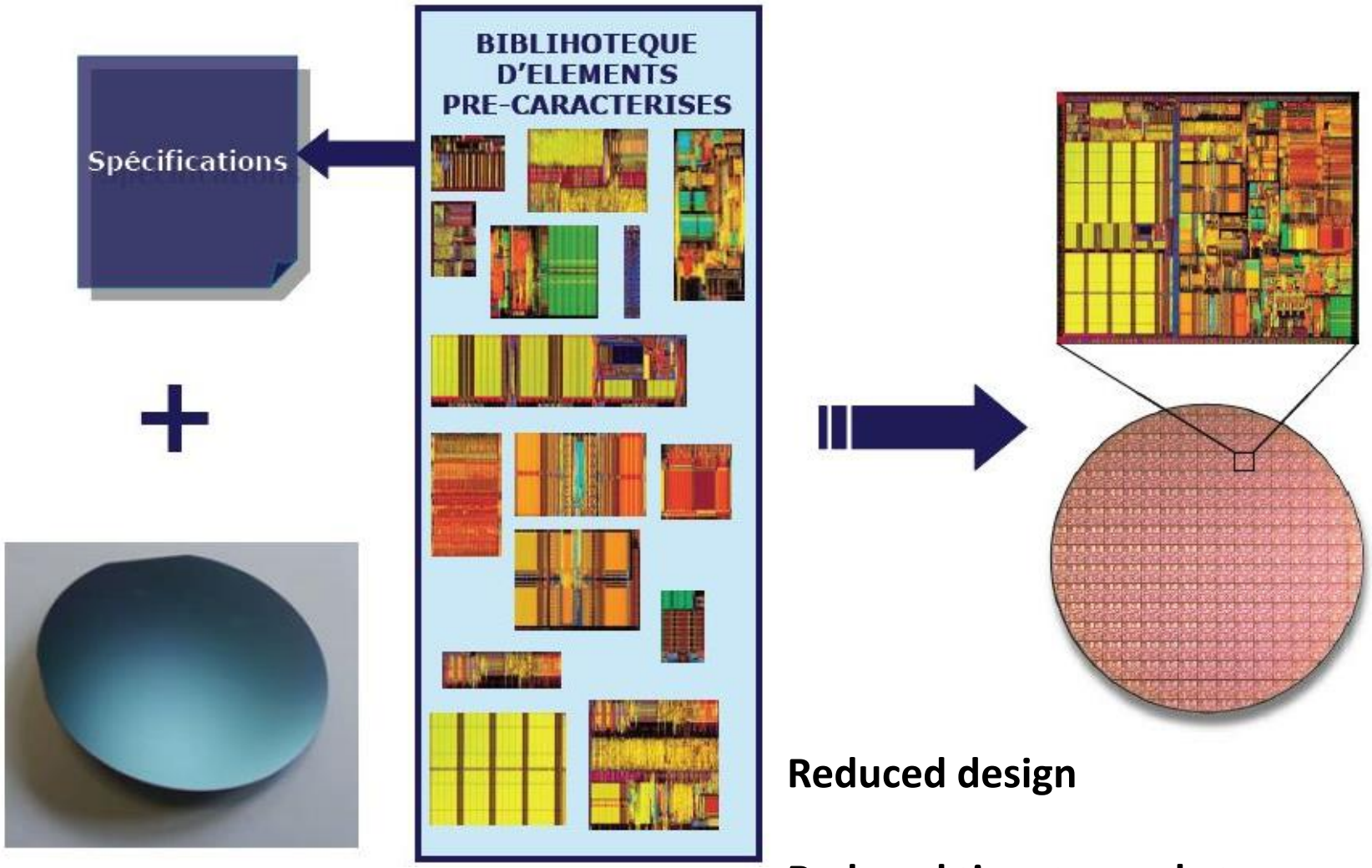
All is configurable, subjected to DRC rules

For large and specific designs



ASIC – Application Specific Integrated Circuit

Standard cell

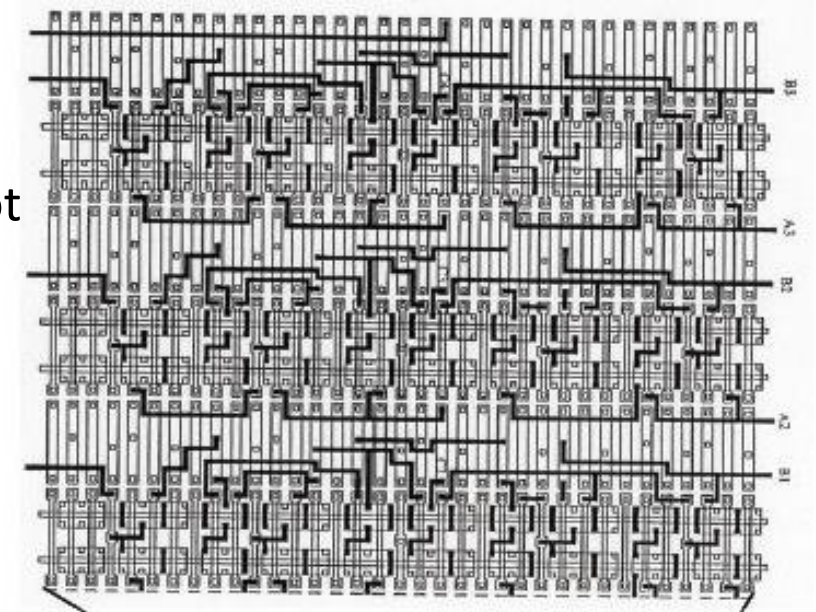


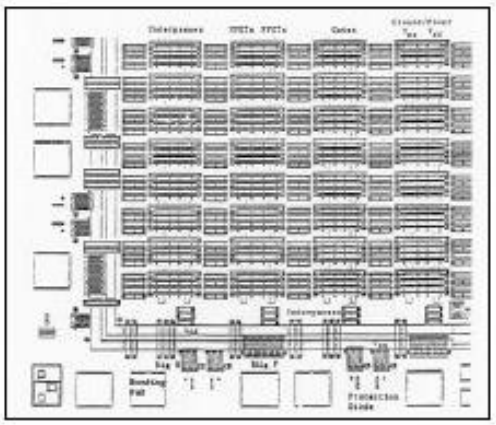
Reduced design

Reduced time to market

ASIC – Structured

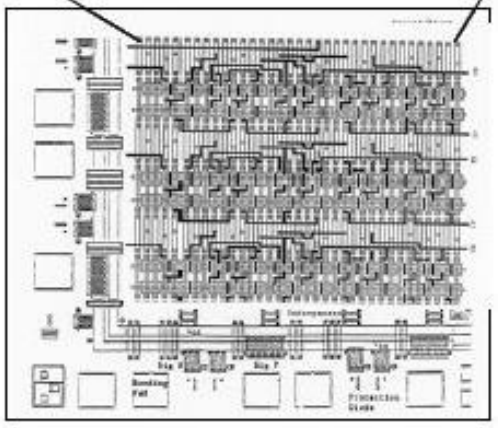
The circuits are partially manufactured: the transistors are made but not interconnected
Metal levels can be configured



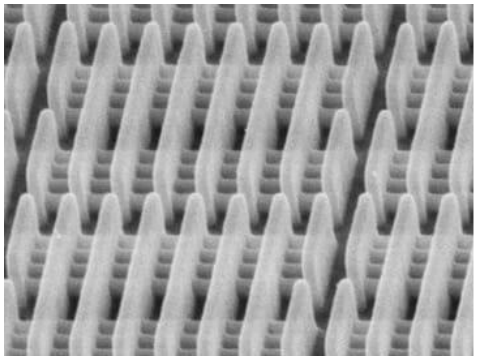


Reduced costs, as all transistors are pre manufactured

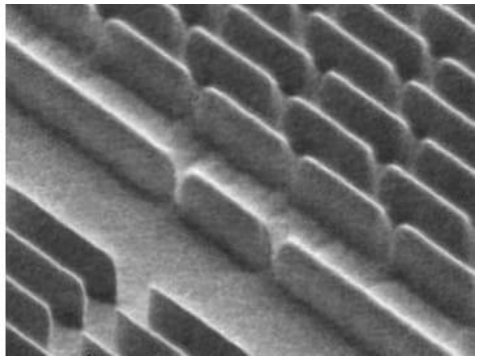
More easy to design



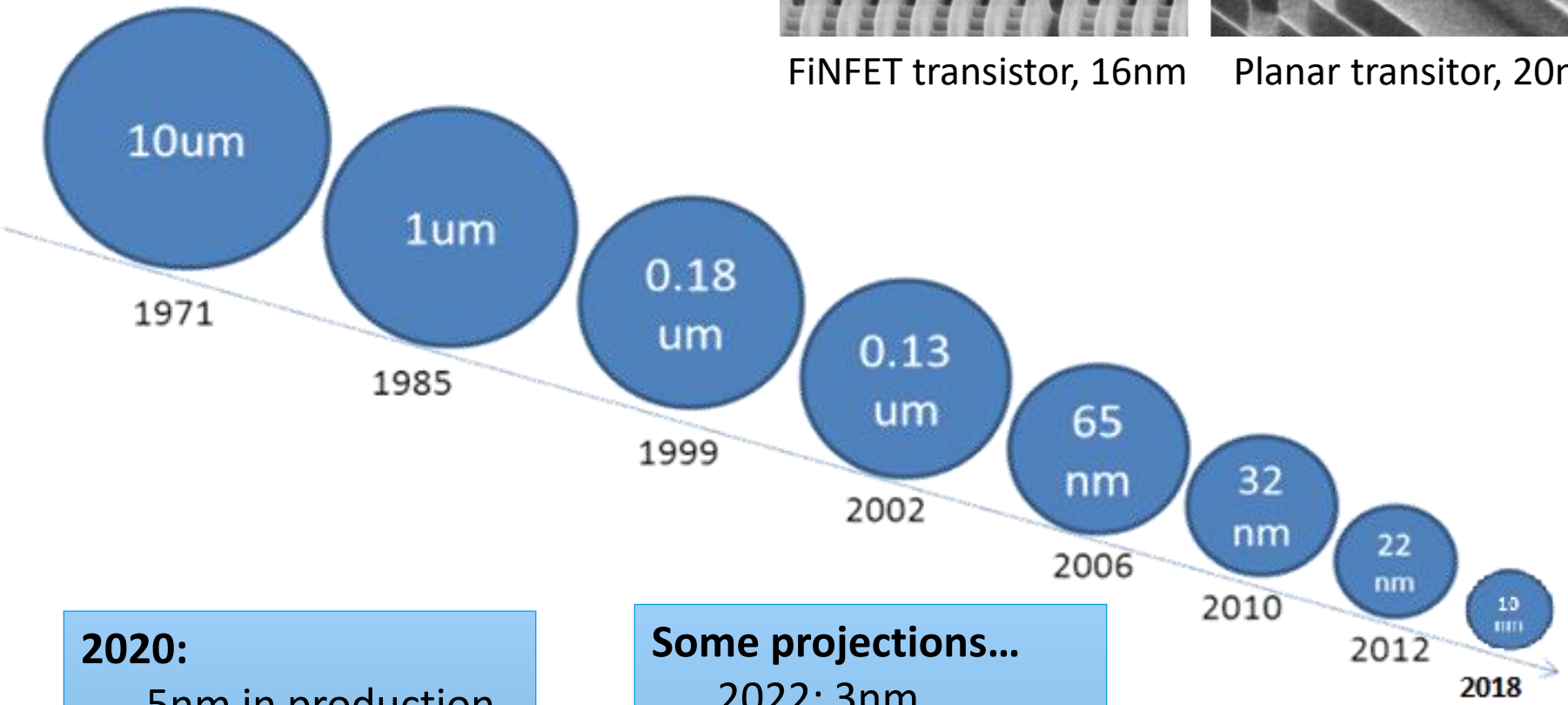
ASIC – costs



FiNFET transistor, 16nm



Planar transitor, 20nm



2020:

5nm in production

up to 100Mtr/mm²

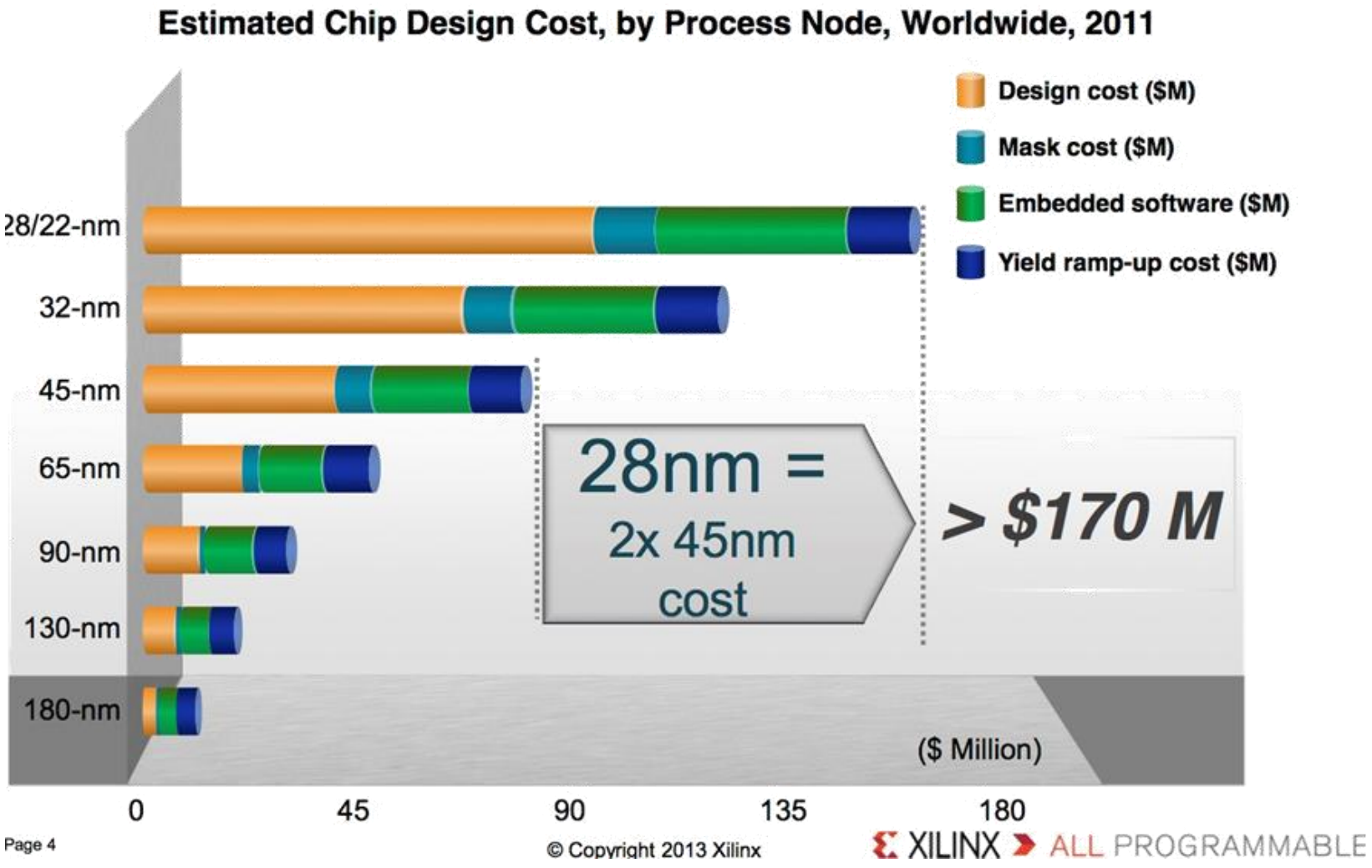
Some projections…

2022: 3nm

2024: 2nm

2029: 1.2nm

ASIC – costs


Estimated Chip Design Cost, by Process Node, Worldwide, 2011
Design cost ($M)
Mask cost ($M)
Embedded software ($M)
Yield ramp-up cost ($M)
28/22-nm
32-nm
45-nm
65-nm
90-nm
130-nm
180-nm
28nm =
2x 45nm
cost
> $170 M
($ Million)
0
45
90
135
180
Page 4
© Copyright 2013 Xilinx
XILINX ALL PROGRAMMABLE

ASIC – Summary

Advantages and drawbacks?

Pro	Con
Performances	Design complexity
Power consumption	Time-to-market
Density	High NRE (Non recurring Engineering)
Low cost, if very large production	

ASSP - Application Specific Standard Product

IC designed for a specific application, but for multiple customer

Initially, two main technologies:

- TTL (*Transistor Transistor Logic*)
- CMOS (*Complementary Metal Oxide Semiconductor*)
- … and subversions

Advantages and drawbacks?

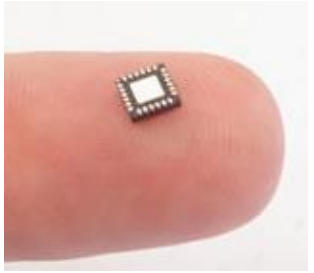
Pro	Con
Robustness / Fiability	Density (surface)
Cost for small designs	Performance
Supply sources	Overall cost
	No flexibility (no reconfiguration)

Example: NE555, HEF4000, MAX232

ASSP - Application Specific Standard Product

IC designed for a specific application, but for multiple customer

Nowadays, ASSP still widespread with more complex functionnalities, more integration.



Even contains SoC.

Advantages and drawbacks?

Pro	Con
Highly integrated functionnality	Performance / density/ consumption NOT as optimized as ASIC
Cost for small/medium production volume	Supplier dependant – EOL to manage carrefully -
No NRE	Often contains unused functionnalities

Examples : Ethernet controller, Audio encoder/decoder chip

Software

- Two families:
  - ✓ Microprocessor / microcontroller
  - ✓ DSP (Digital Signal Processor)

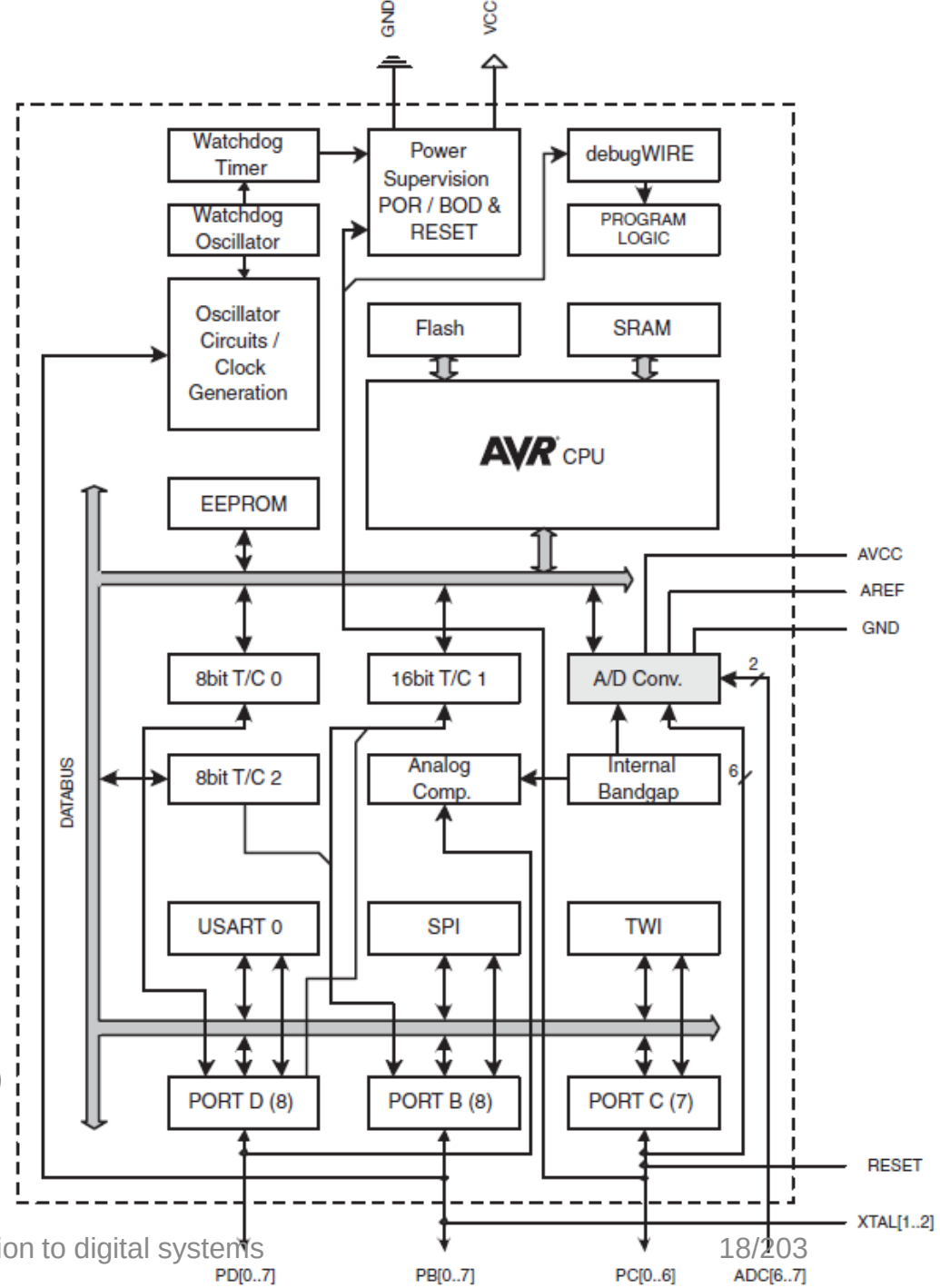
Advantages and drawbacks?

Pro	Con
Flexibility *Application can be changed by modifying the embedded software*	Low rate processing *sequential architecture, memory access*
Quite simple to design *high level languages*	High power consumption *high frequency clock*
Shorter time design	
Low cost (*No NRE)*	

Software

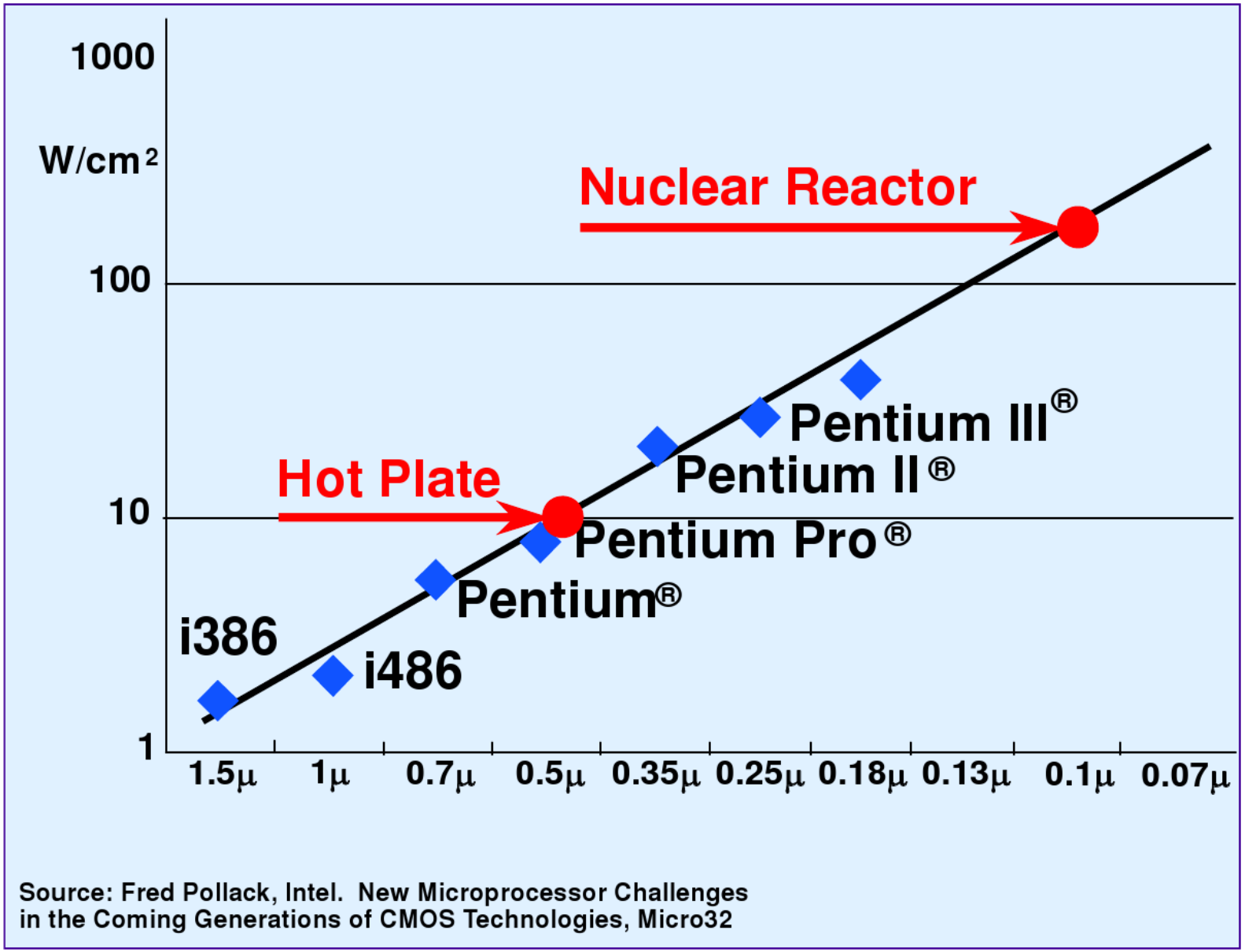
- Example: ATMEGA328P

- ✓ Low cost (few $)
- ✓ Low power (~ 10 mA under 3 V)
- χ Dedicated I/Os
- χ Low speed processing (~1 MIPS/MHz)



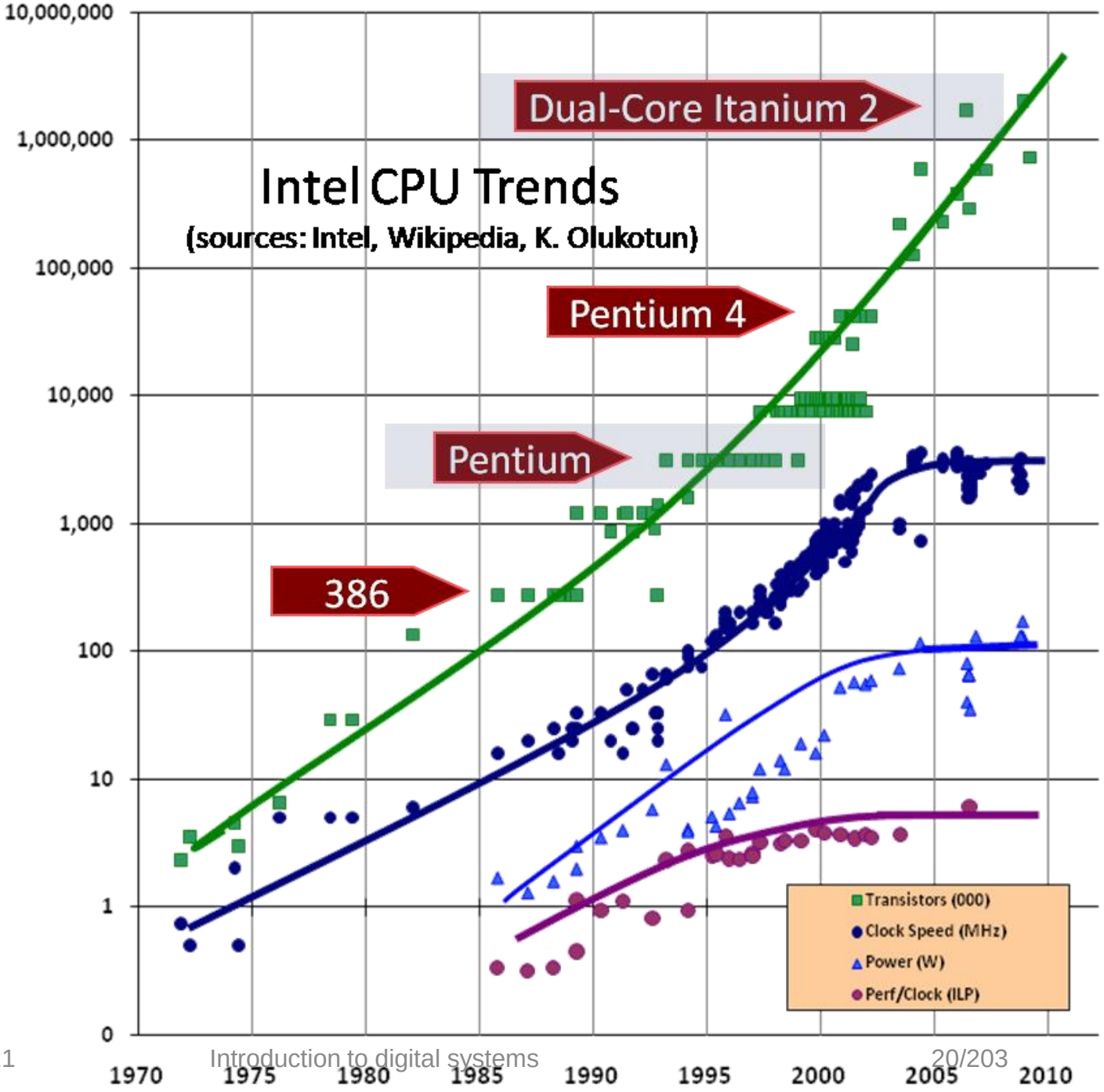
Software

http://www.gotw.ca/publications/concurrency-ddj.htm



Source: Fred Pollack, Intel. New Microprocessor Challenges in the Coming Generations of CMOS Technologies, Micro32

Software

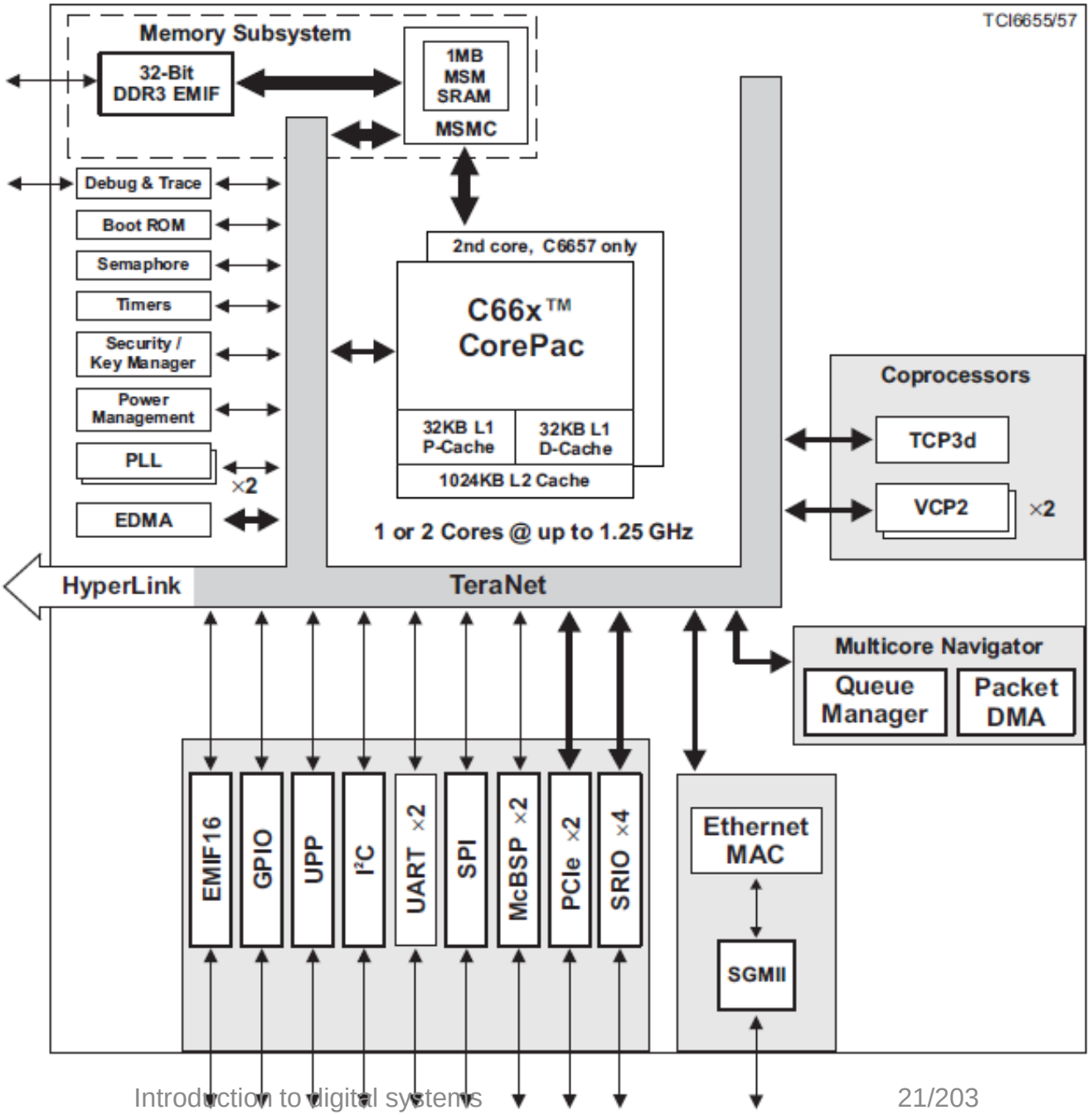

Intel CPU Trends
(sources: Intel, Wikipedia, K. Olukotun)
Dual-Core Itanium 2
Pentium 4
Pentium
386
Transistors (000)
Clock Speed (MHz)
Power (W)
Perf/Clock (ILP)
10,000,000
1,000,000
100,000
10,000
1,000
100
10
1
0
1970
1975
1980
1985
1990
1995
2000
2005
2010

Software

- Solutions ?

  - ✓ Pipeline
  - ✓ Multi cores

Example: TI C6600



Software

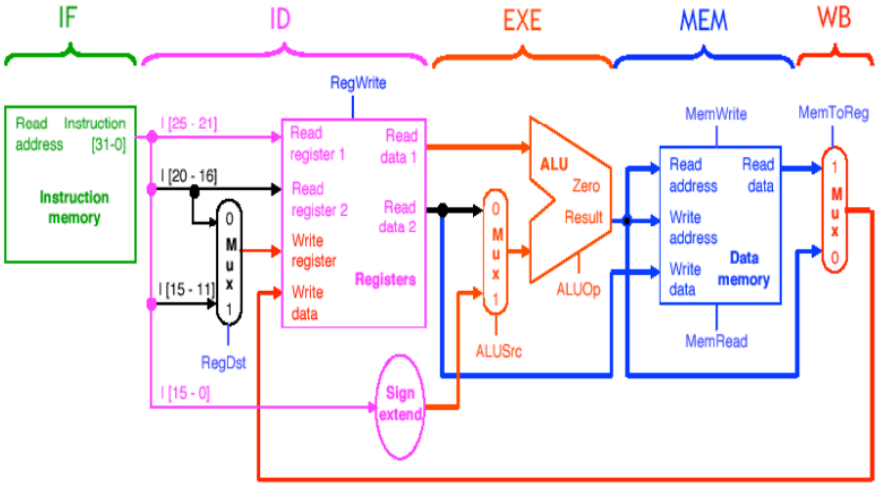
Step	Name	Description
Instruction Fetch	IF	Read an instruction from memory.
Instruction Decode	ID	Read source registers and generate control signals.
Execute	EX	Compute an R-type result or a branch outcome.
Memory	MEM	Read or write the data memory.
Writeback	WB	Store a result in the destination register.

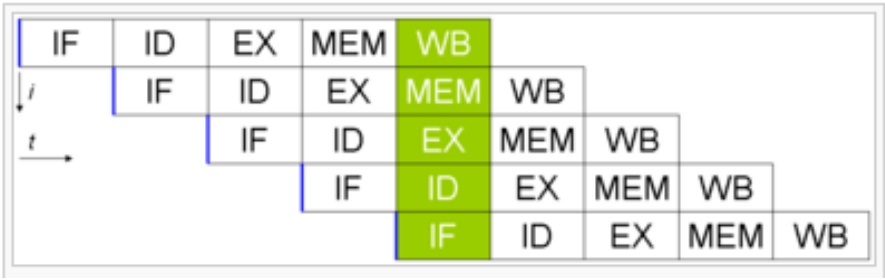
- Pipeline: MIPS example

1 MIPS instruction = up to 5 clock cycles

Without pipeline: One step per clock cycle

With pipeline: Up to Five step in a single clock cycle

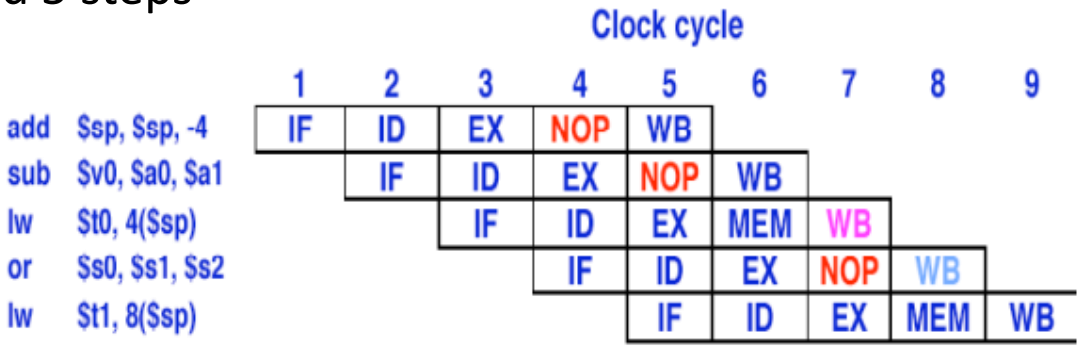




Software

- Pipeline: Limitations and drawbacks
  - Instructions not always need 5 steps

Instruction	Steps required				
beq	IF	ID	EX		
R-type	IF	ID	EX		WB
sw	IF	ID	EX	MEM	
lw	IF	ID	EX	MEM	WB



  - Memory management / data hazard

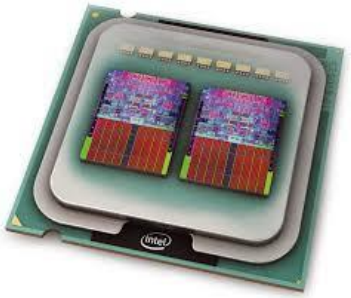
```
lw $s1, 4($sp)      IF  ID  EX  MEM  WB
add $s0, $s1, $s2       IF  ID  EX   MEM  WB
```

« ID » reads register $s1 before « WB » store the result into it

=> Insertion of bubbles

```
lw $s1, 4($sp)      IF  ID  EX  MEM  WB
add $s0, $s1, $s2       IF  nop nop  nop  ID
```

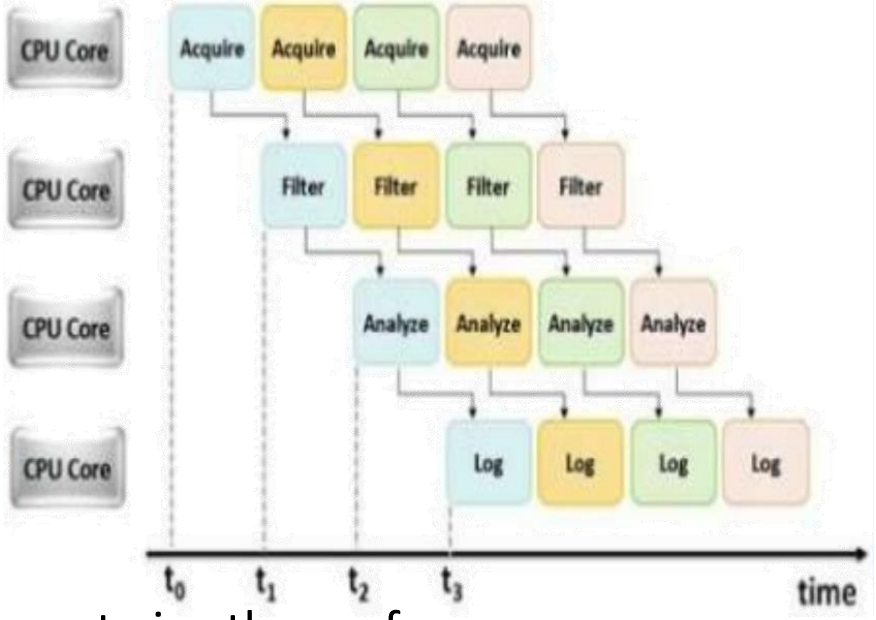
Software



- Multi-cores:

Advantages and drawbacks?

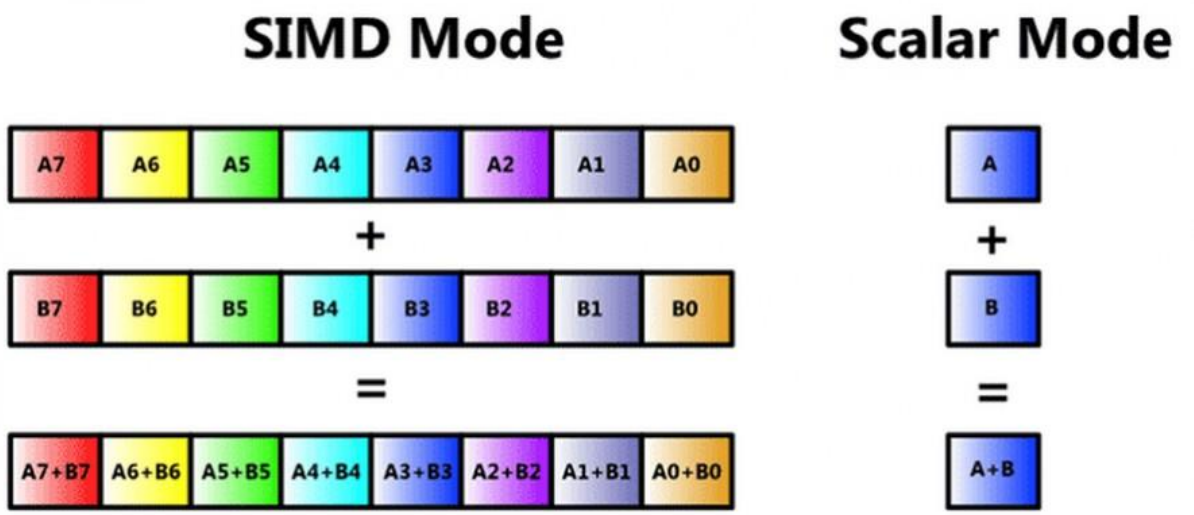
- Pro
  - Simultaneous execution
  - Multithread application
  - Pipelining (application level)

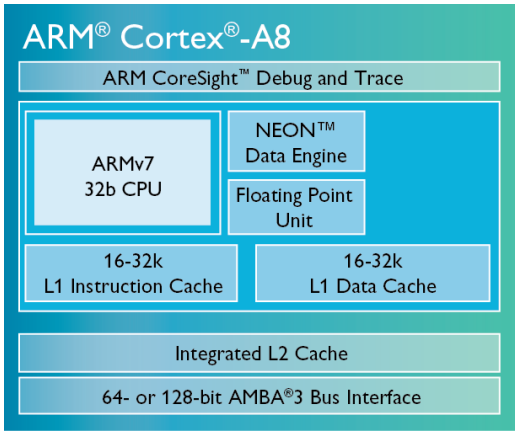


- Cons
  - χ Double HW ressources don't have twice the performance
  - χ Complexity: how to develop the software ?
  - χ Power consumption
  - χ Memory sharing …

Software

- Other solution: SIMD (*Single Instruction on Multiple Data*)





SSE for Intel,
NEON for ARM,
AltiVec for Power PC

Semi-custom

PLD (*Programmable Logic Device*)

- SPLD :
  Simple Programmable Logic Device
- **CPLD :**
  Complex Programmable Logic Device
- **FPGA :**
  Field Programmable Gate Array

What is a FPGA? - First approach -

Field Programmable Gate Array

$\Rightarrow$ OK…. But?

Combination of 3 very simple elements:

Digital gate

Boolean algebra

Primary operations: AND, OR, NOT

Value of variables: True (1) or False (0)

Register

« One bit short-term memory »

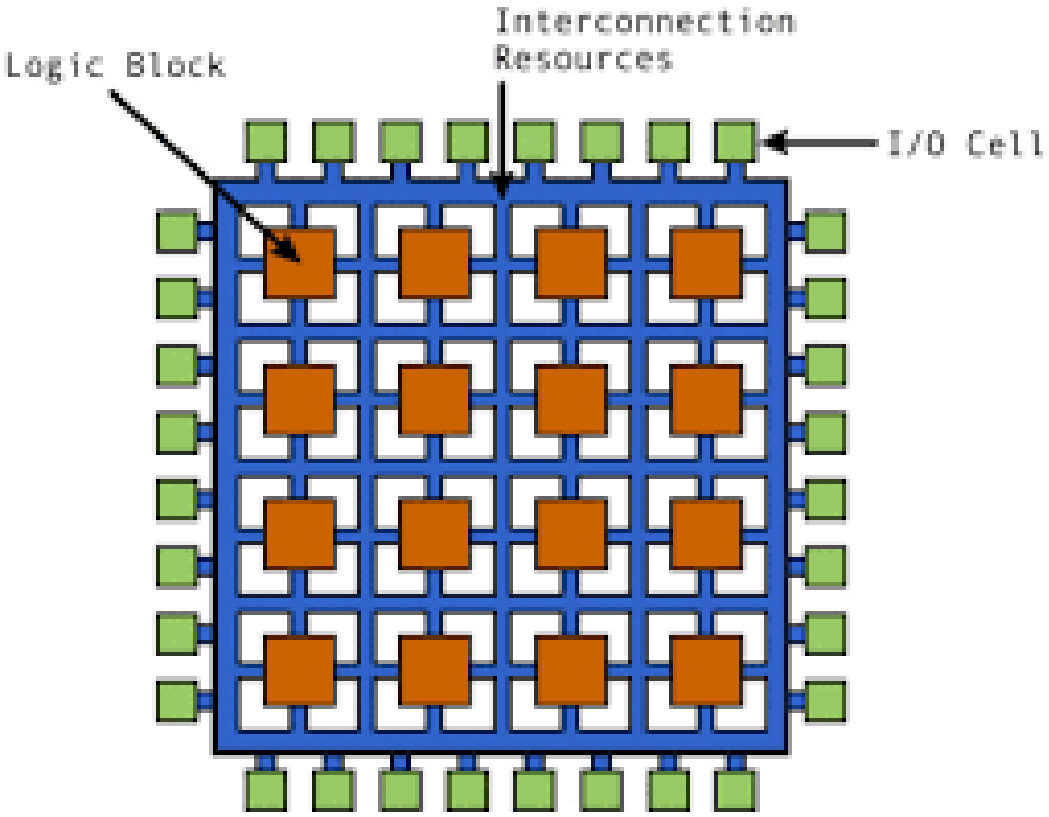
Wire

interconnections

What is a FPGA? – First approach -

FPGA = Millions of Gates and Registers, with customized Interconnection

$\Rightarrow$ Build exactly the Hardware you need



Applications of FPGA

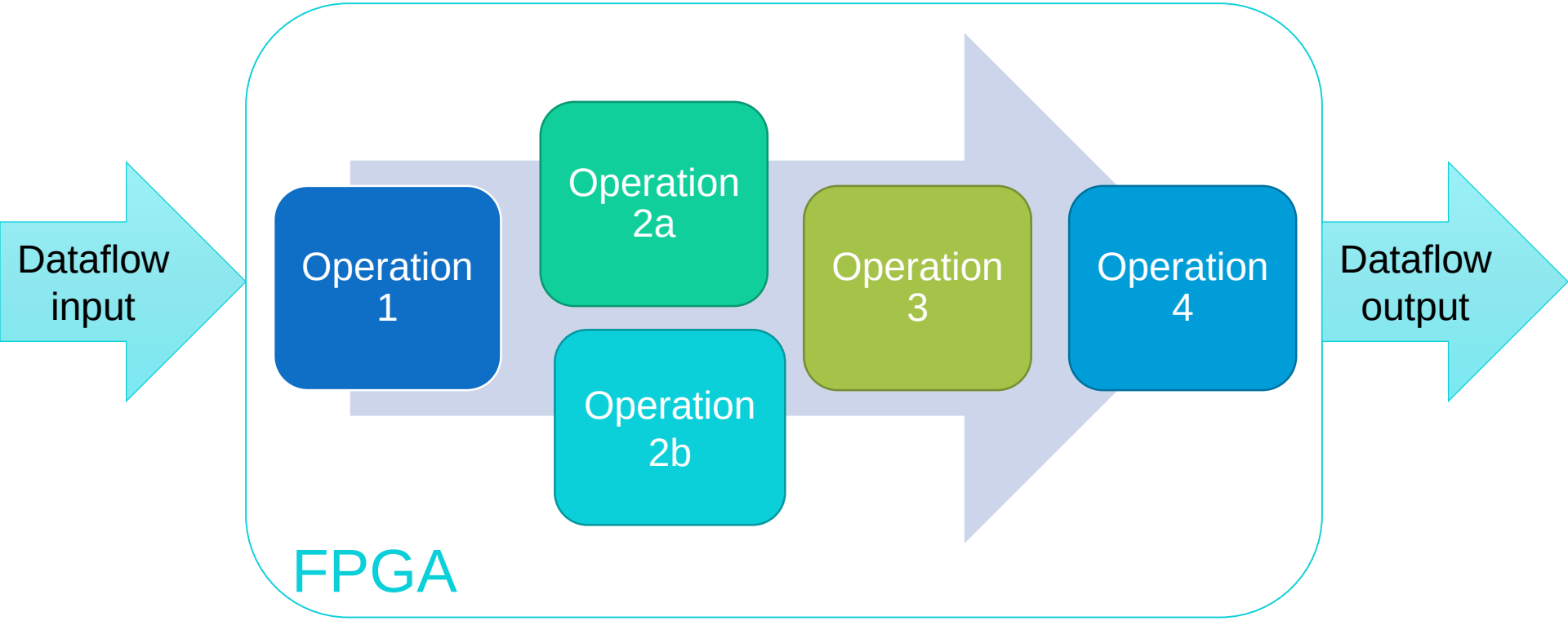
Application fields: everywhere **logic functions** should be implemented with **high performances** (high bandwith, high speed processing, …)

- Militay & aerospace
- Medical
- Communications (Wired, optical, radio)
- Automotive
- Industrial management (motor, actuators)
- Research
- Video processing
- ASIC prototyping
- Computers (servers, data mining, bitcoin mining, ...)

Applications of FPGA

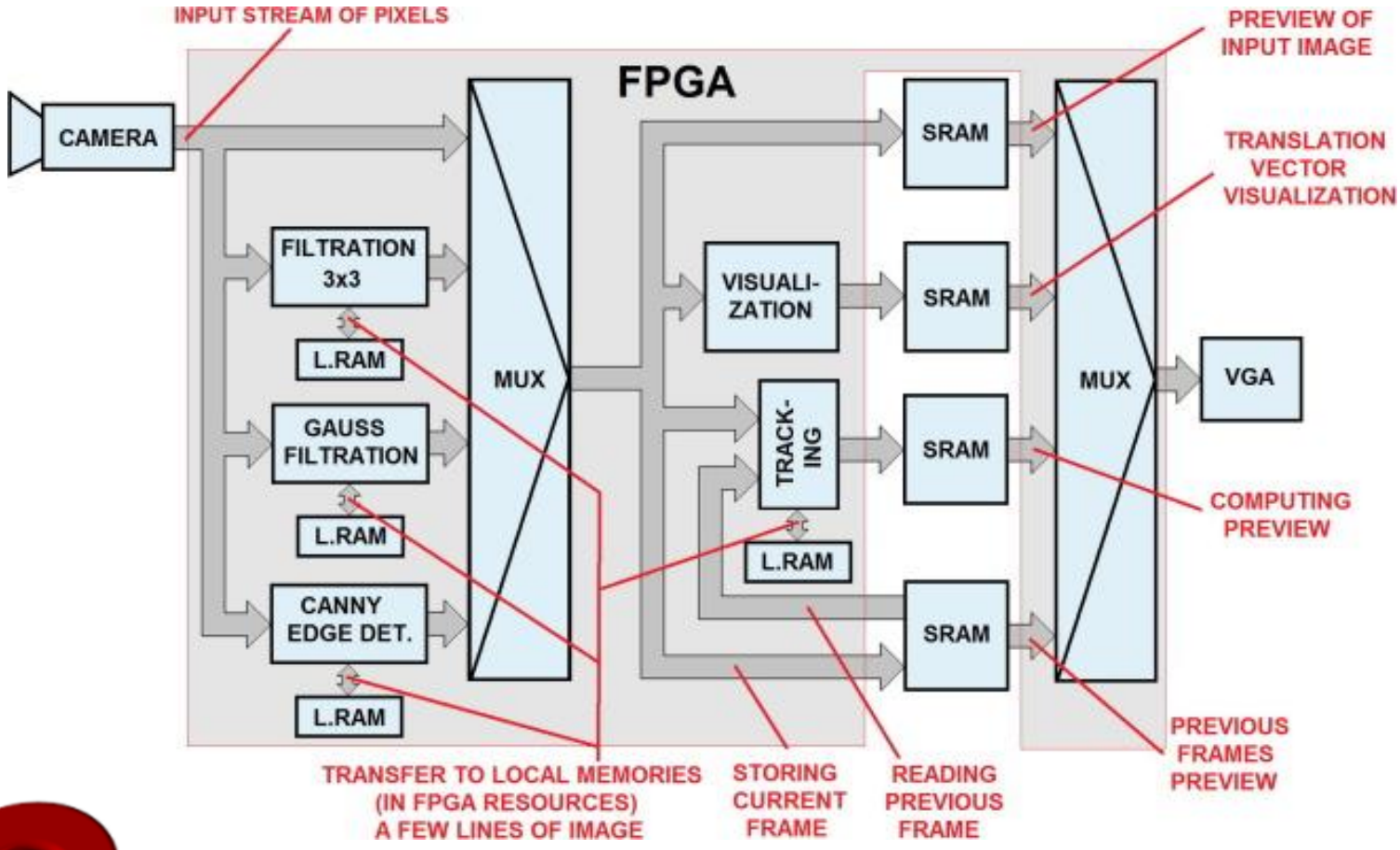
FPGA especially efficient on dataflow computing

- Examples: encoding, image processing, cryptography,…



No real computing time, only latency

Applications of FPGA



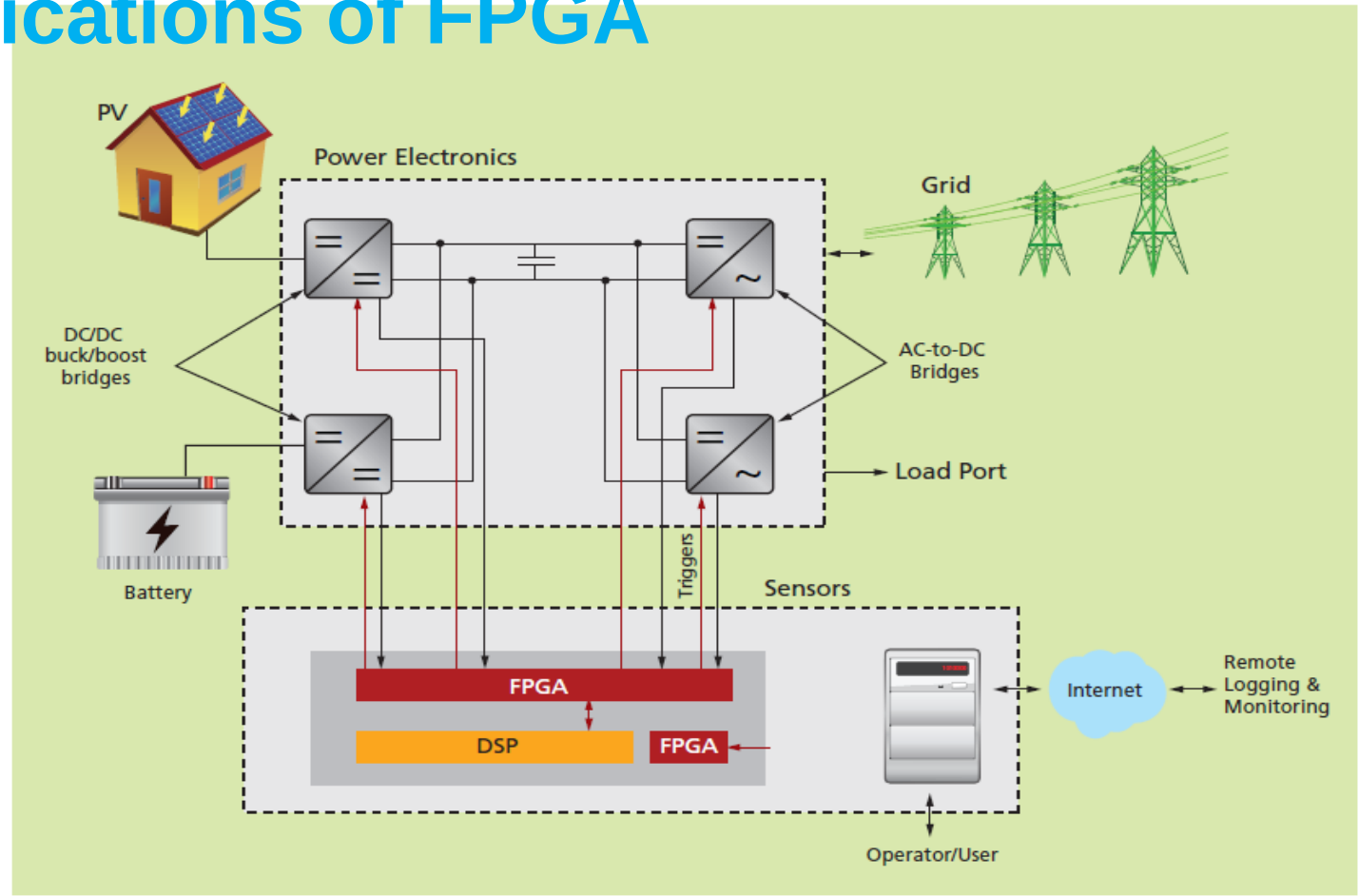
Why a FPGA ?

High speed dataflow

Data computing « on the flow » : high performance

Parallel processing

Applications of FPGA

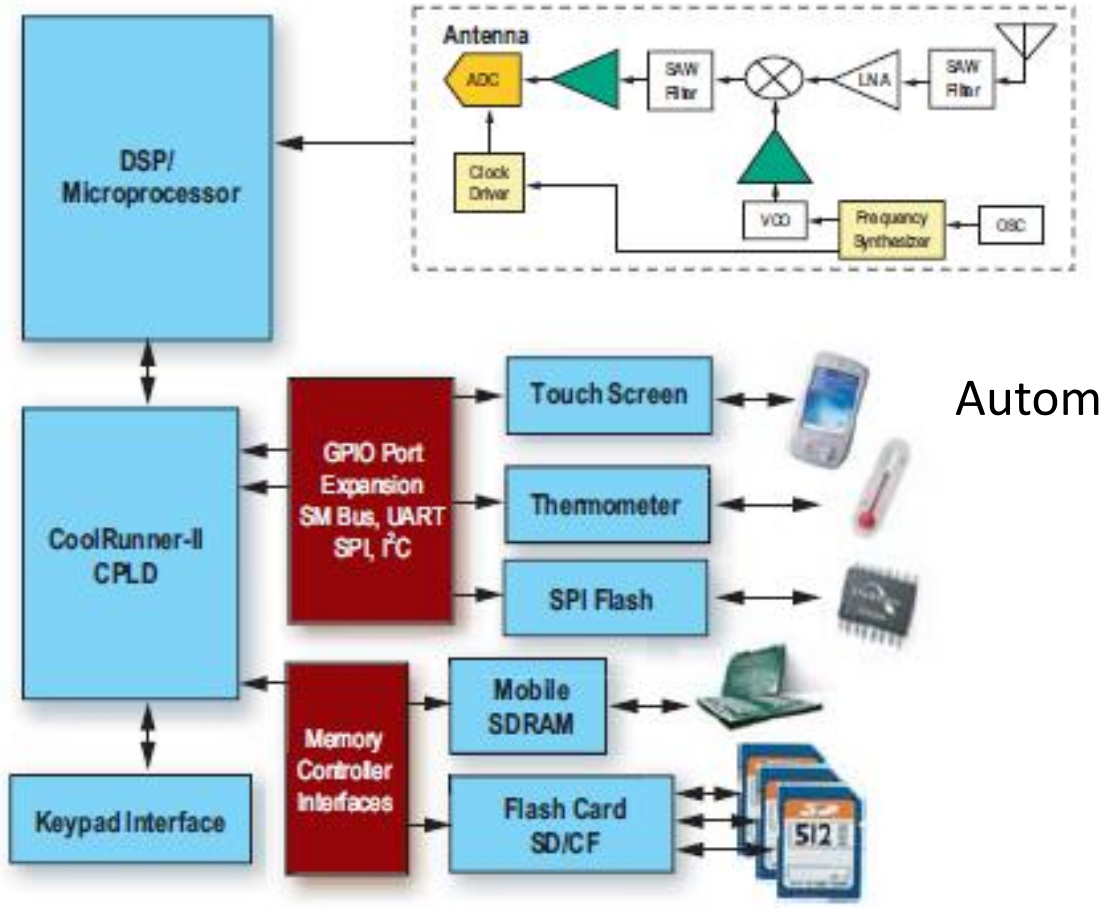


Smart grid: management of a small photovoltaic electrical plant

Why a FPGA ?

High number of I/O
Numerous interface types

Applications of FPGA

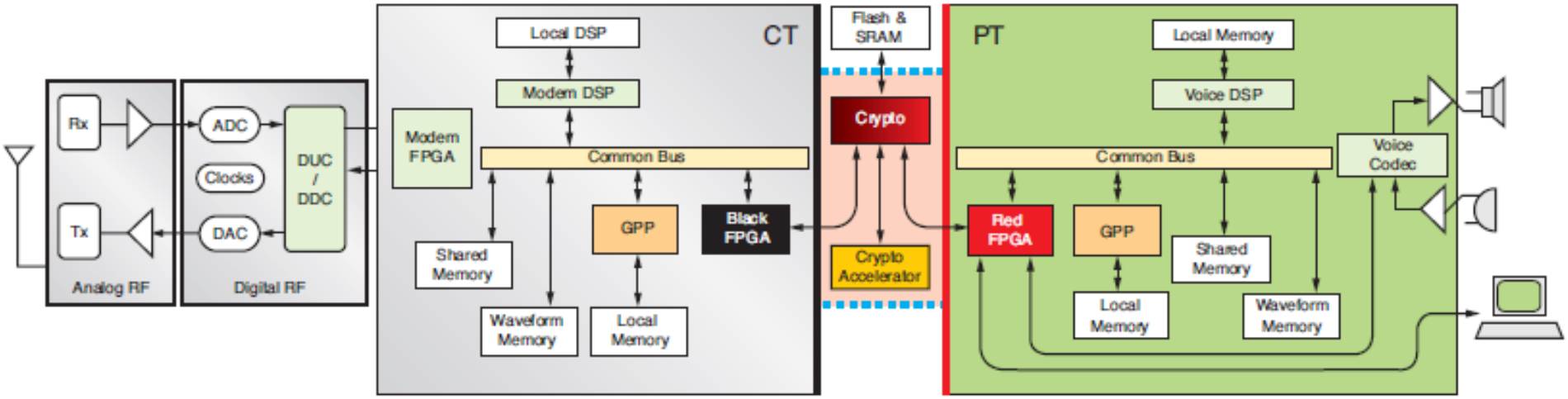


Automotive GPS receiver

Why a PLD ?

Data computing
Logic glue
Multiple interfaces

Applications of FPGA



Tactical SDR (Software Defined Radio)

Several FPGA:

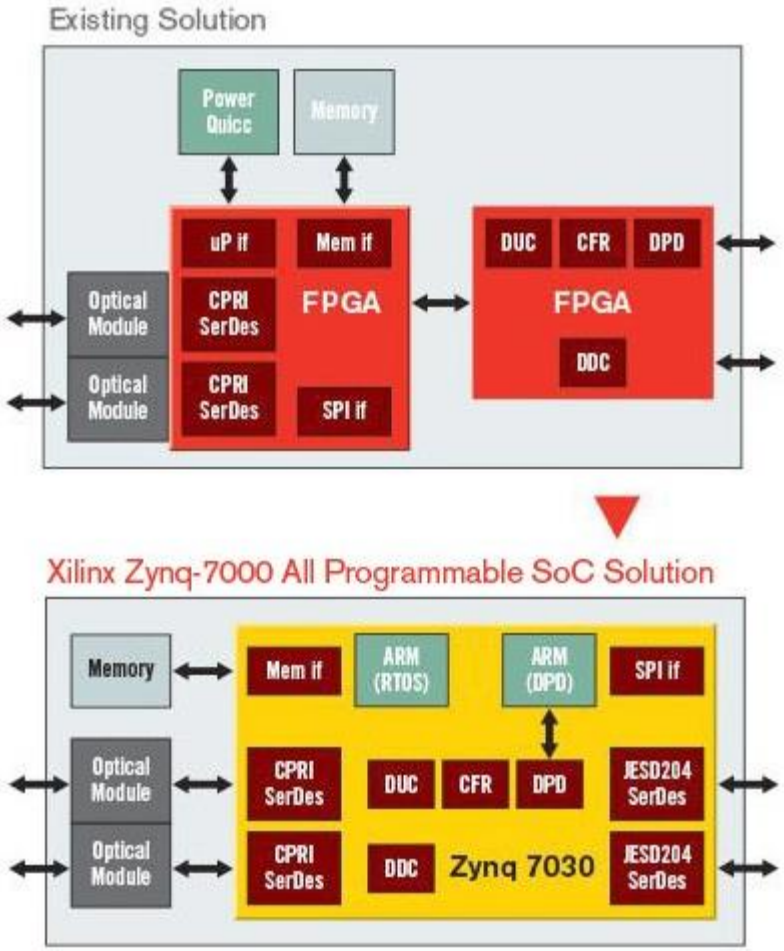
- Cryptography and coded communications (Red and black FPGA)
- Modulation / protocol generation (green FPGA)

Why a FPGA ?

« On the flow » computing
Cryptography
Adaptability to multi protocol

Applications of FPGA

Telecom: prototyping next genetation (3G, 4G, 5G,…)



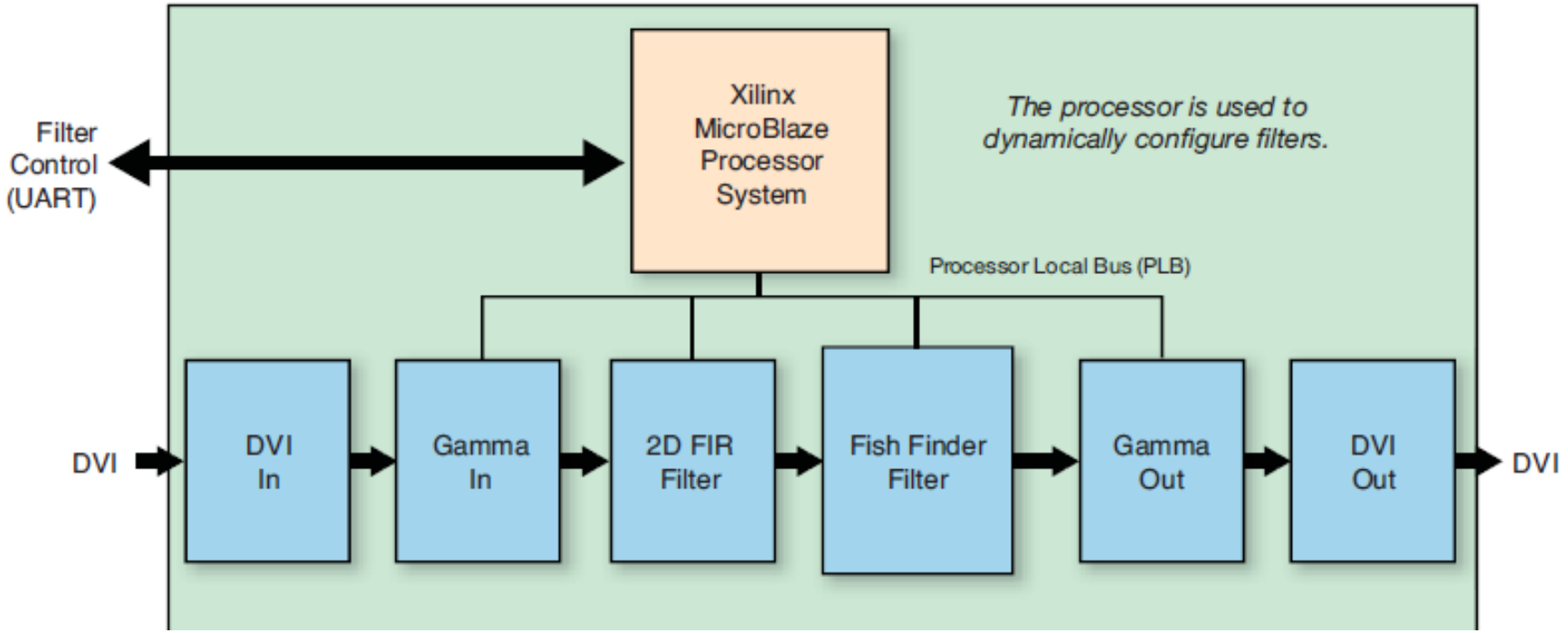
Why a FPGA ?

All protocols in one device
Reconfiguration
High bandwith

2 Antenna 2x20MHz (40MHz) LTE (Long Term Evolution)

Applications of FPGA

Real-time video processing

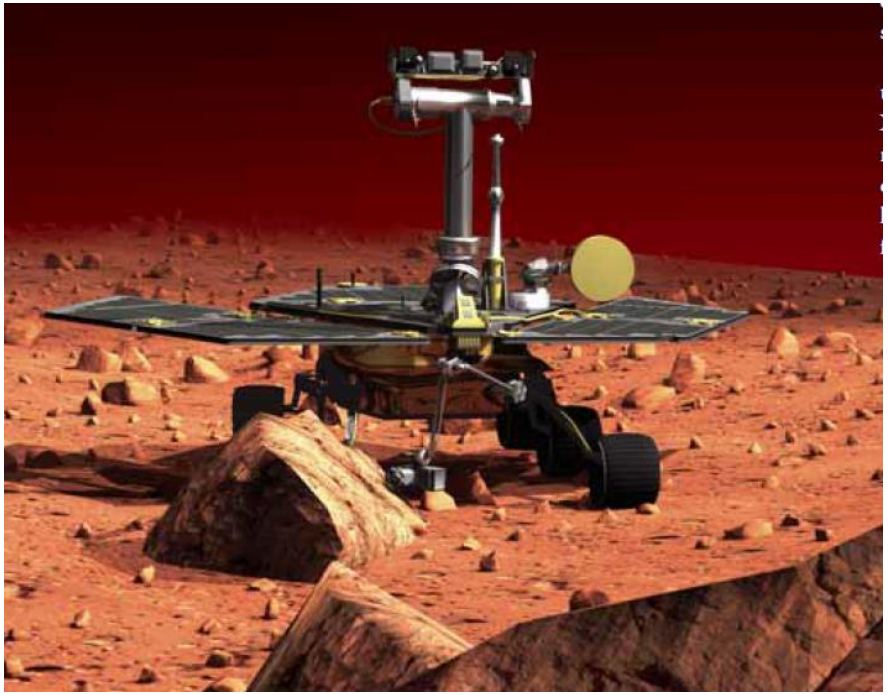


HD video data rate: 720 Mo/s

Why a FPGA ?

High bandwith
Parallel processing
Integrated µP

Applications of FPGA

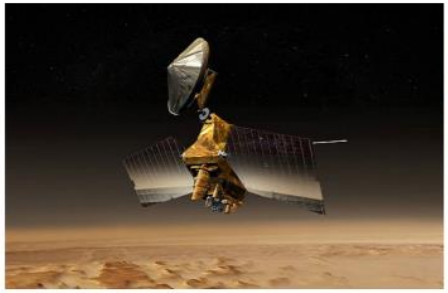


FPGA on Mars ...

FPGA test to be tolerant to radiations.

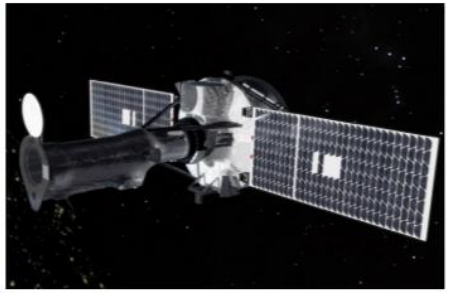
Two FPGA to manage the motors.

Four FPGA are used to control the ignition system.



Mars Reconnaissance Orbiter

Curiosity (Mars Science Lab)



NASA IRIS

Why a FPGA ?

- ✓ High computing capacity
- ✓ High number of IOs
- ✓ Radiation tolerant

Applications of FPGA

Give projects where FPGA can be used ?

where µP can be preferred

where both are usefull

Semi-custom

PLD (*Programmable Logic Device*)

SPLD : *Simple Programmable Device*

CPLD : *Complex Programmable Logic Device*

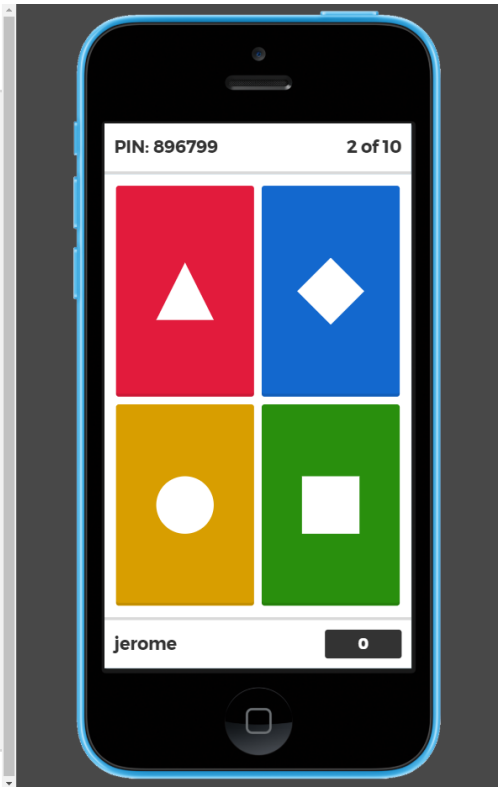
FPGA : *Field Programmable Gate Array*

Advantages and drawbacks?

Pro	Con
Configurable	Power consumption
High performances (parallel and « on the flow » processing)	Specific design
High bandwith	
High number of I/O	

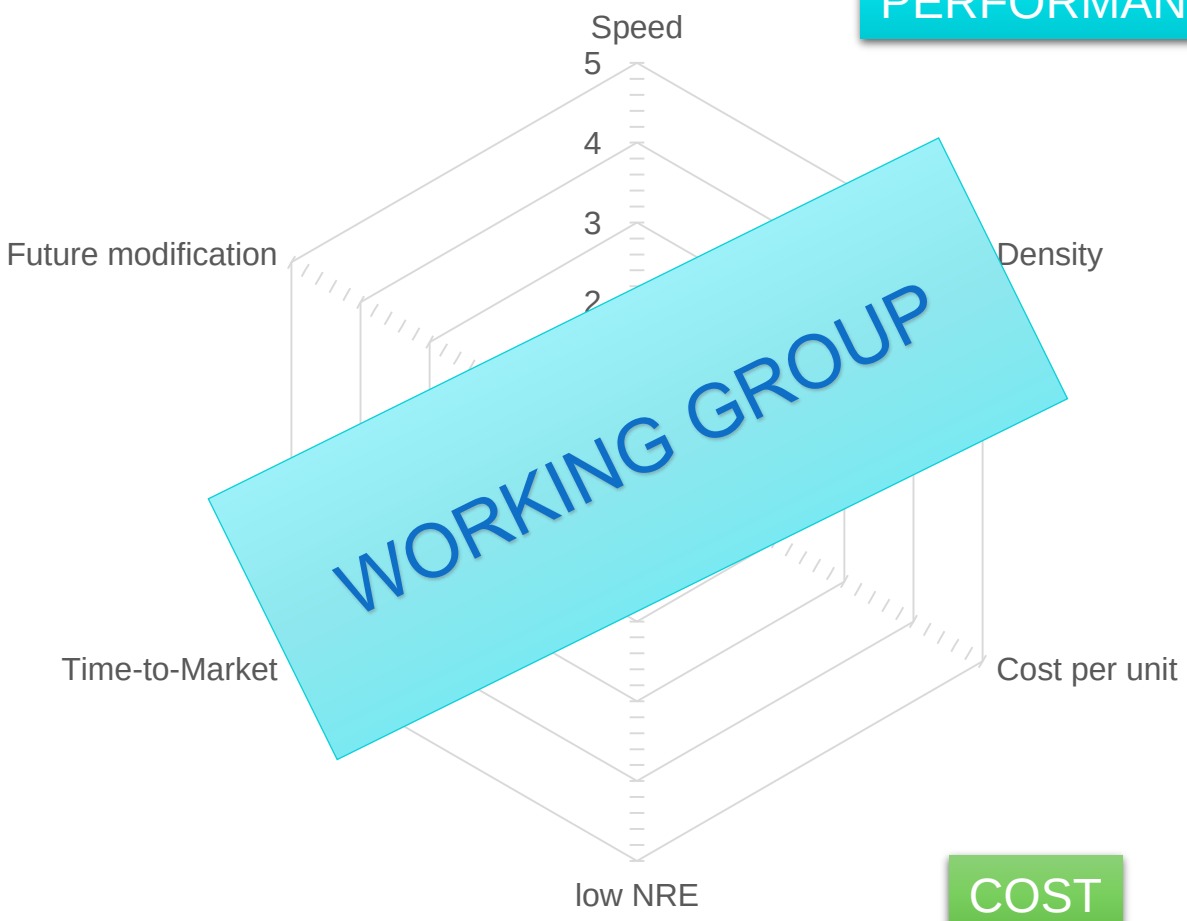
QUIZ

Let's go to https://kahoot.it/



Comparison: FPGA vs CPU vs ASIC

PERFORMANCE



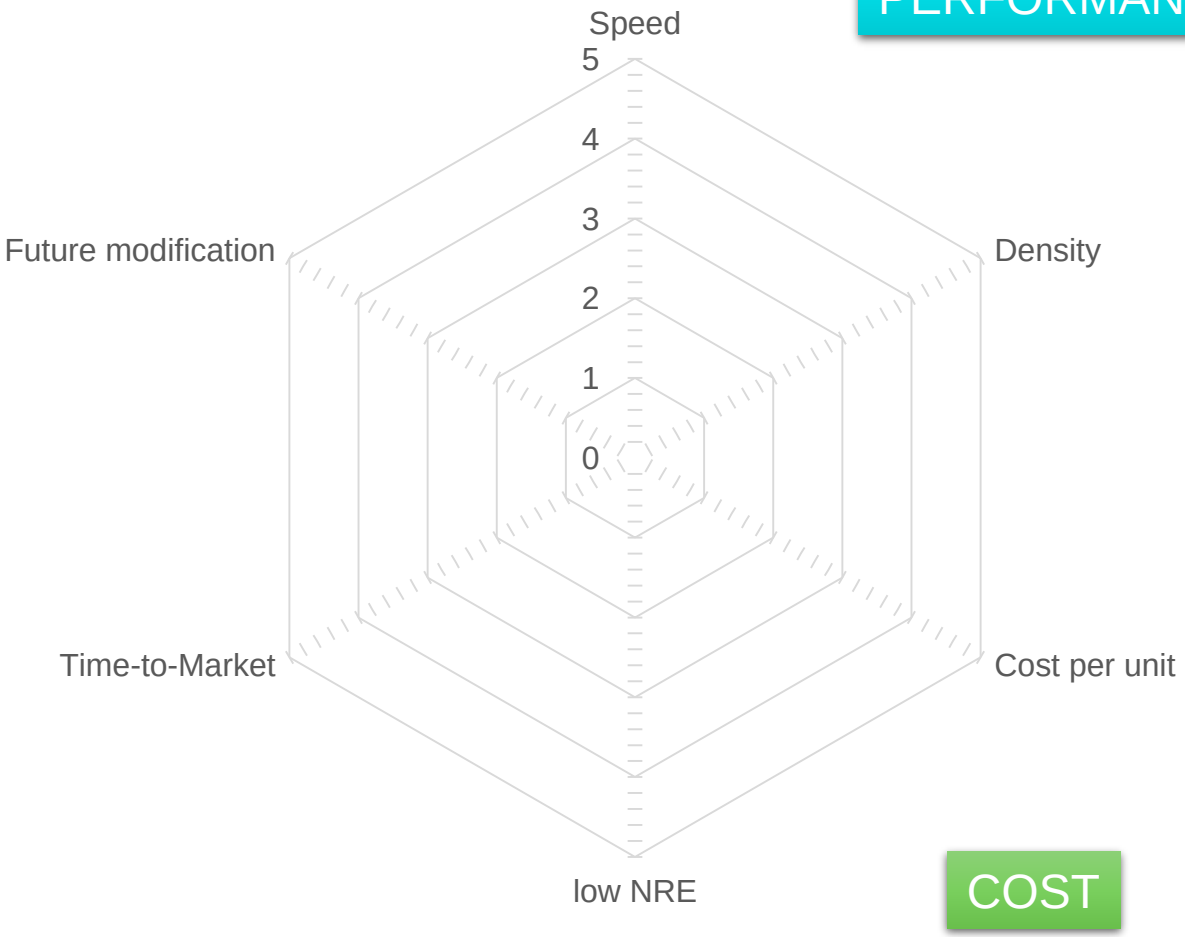
DESIGN

COST

Comparison: FPGA vs ASSP

DESIGN

PERFORMANCE



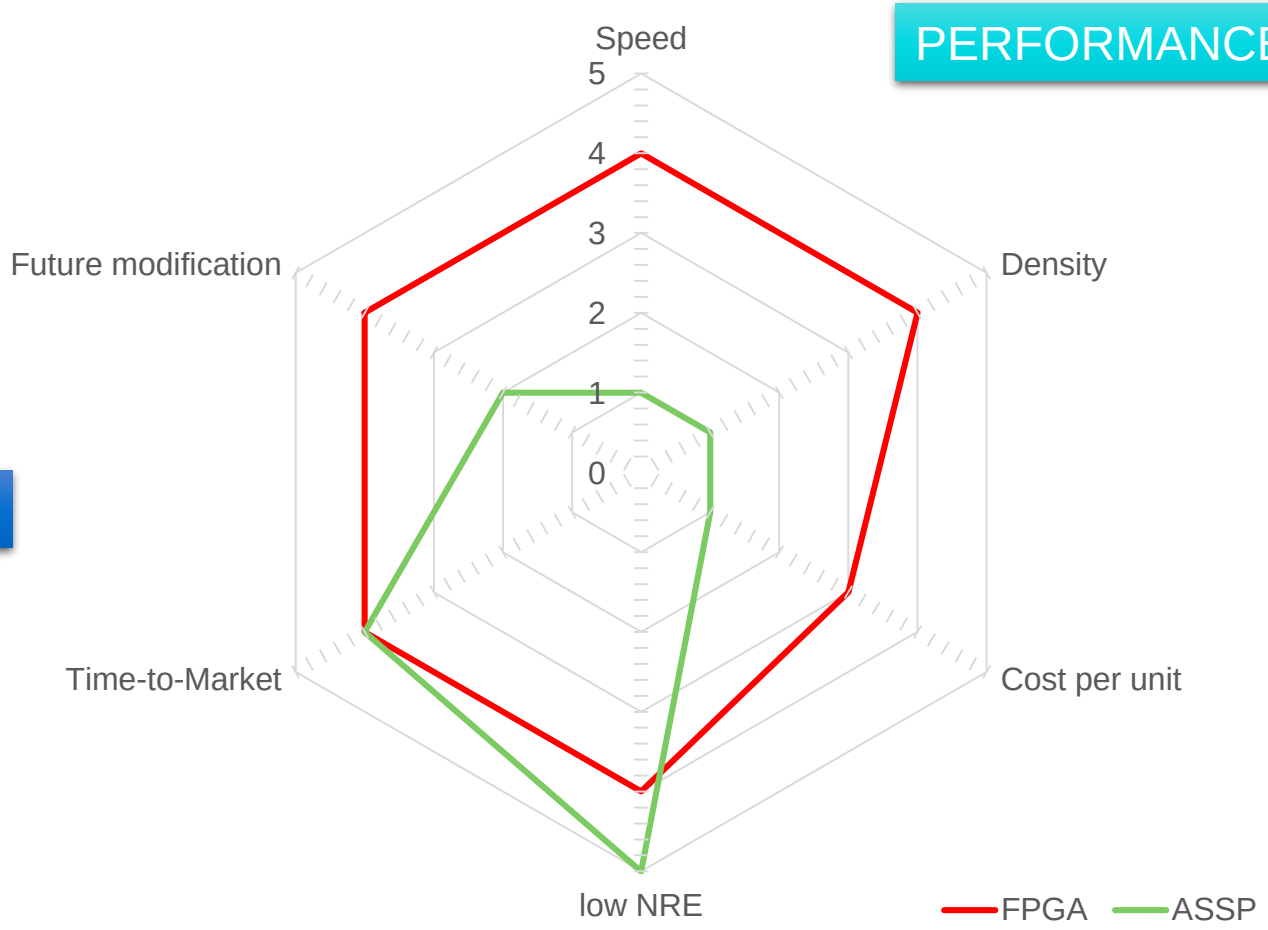
COST

Comparison: FPGA vs ASSP (without SOC)

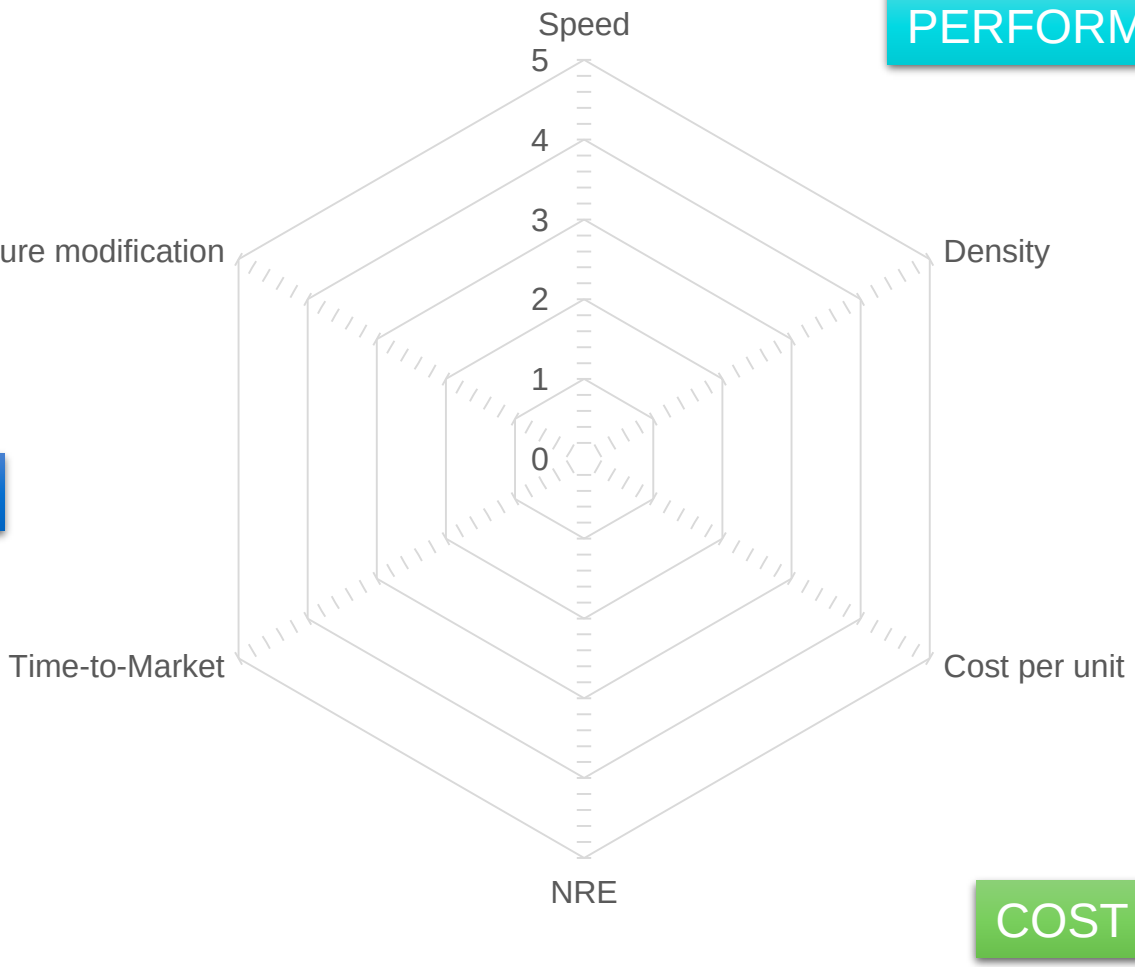
PERFORMANCE

DESIGN

COST



Comparison: FPGA vs ASIC

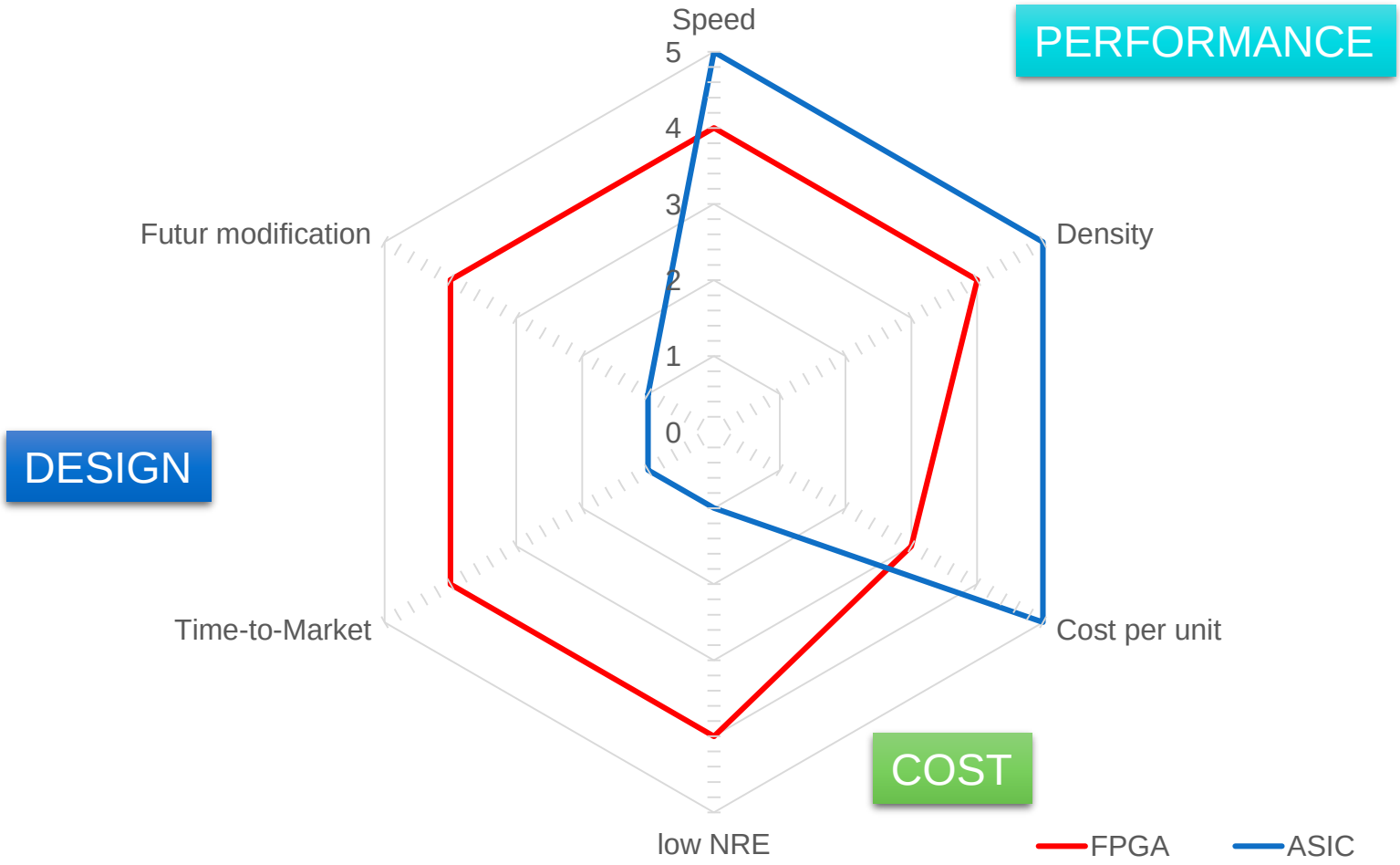


PERFORMANCE

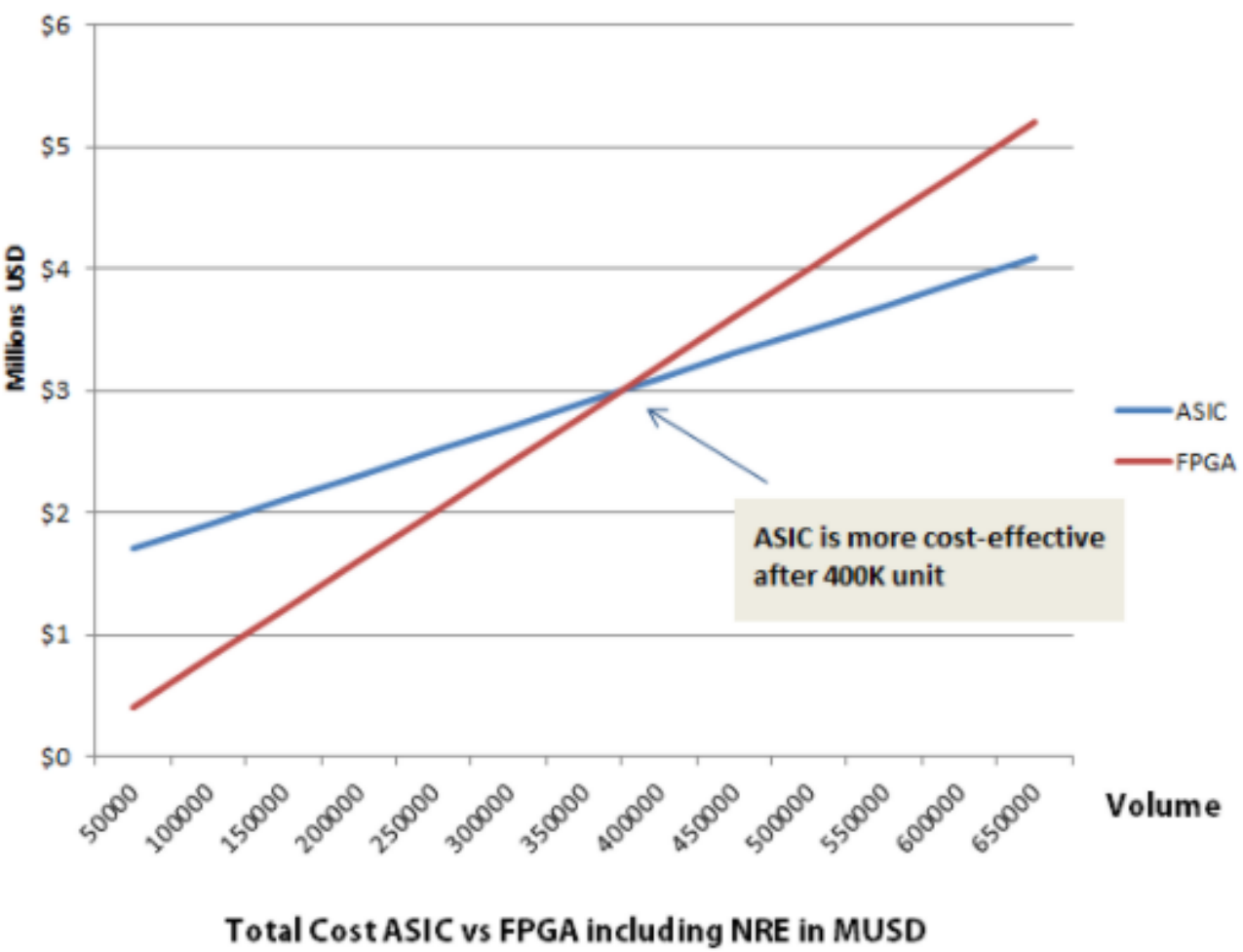
DESIGN

COST

Comparison: FPGA vs ASIC

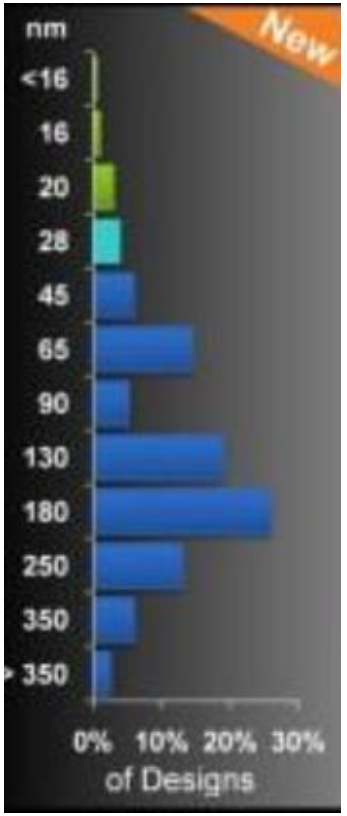


Comparison: FPGA vs ASIC



Total Cost ASIC vs FPGA including NRE in MUSD

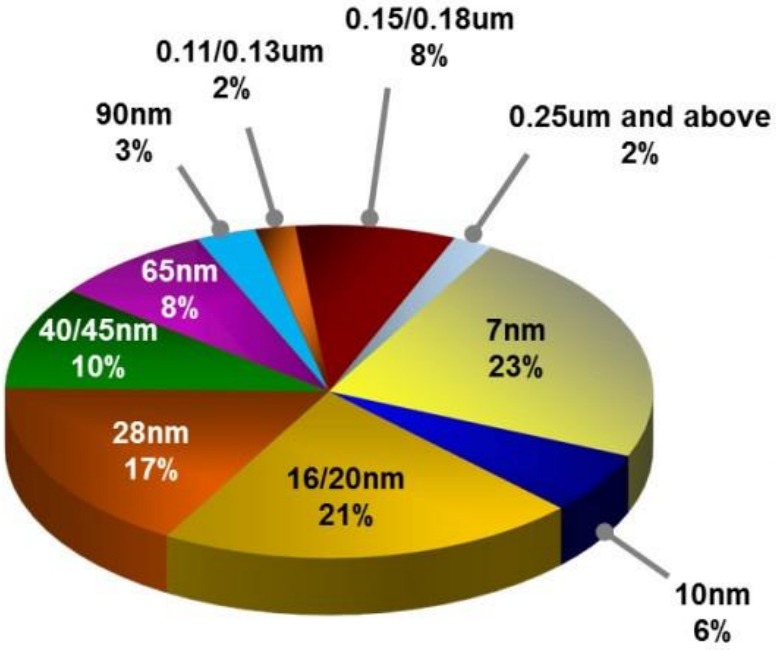
(order of magnitude only: the crossing point could vary significantly)

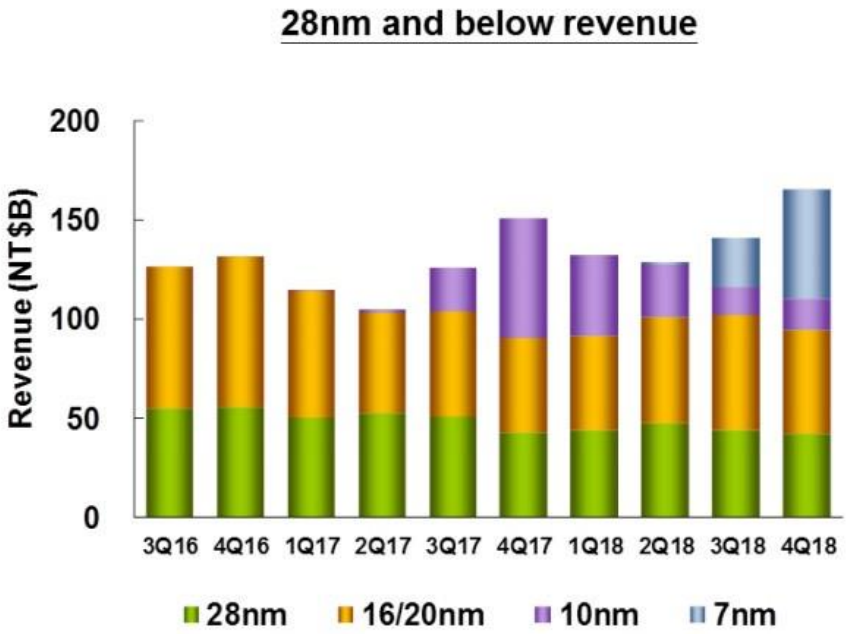


Comparison: FPGA vs ASIC

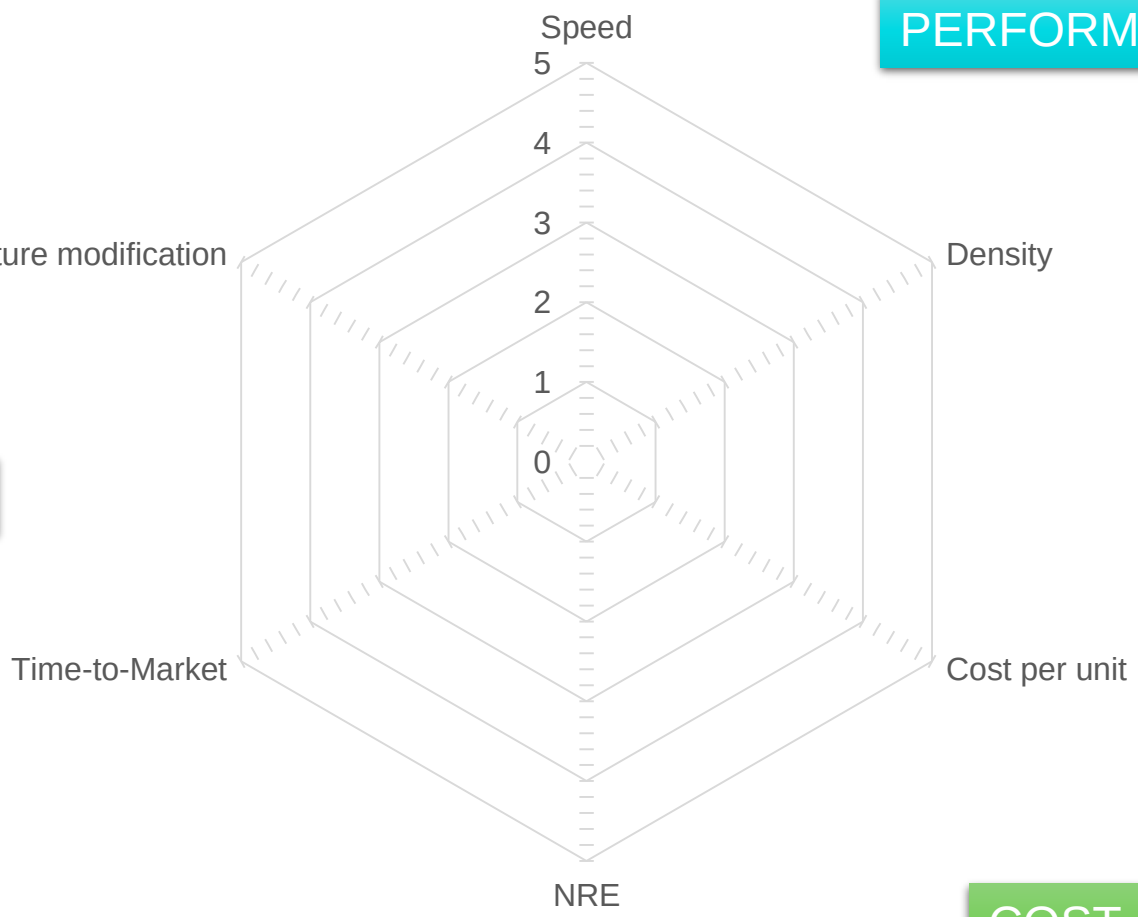
4Q18 Revenue by Technology

TSMC Property





Comparison: FPGA vs CPU

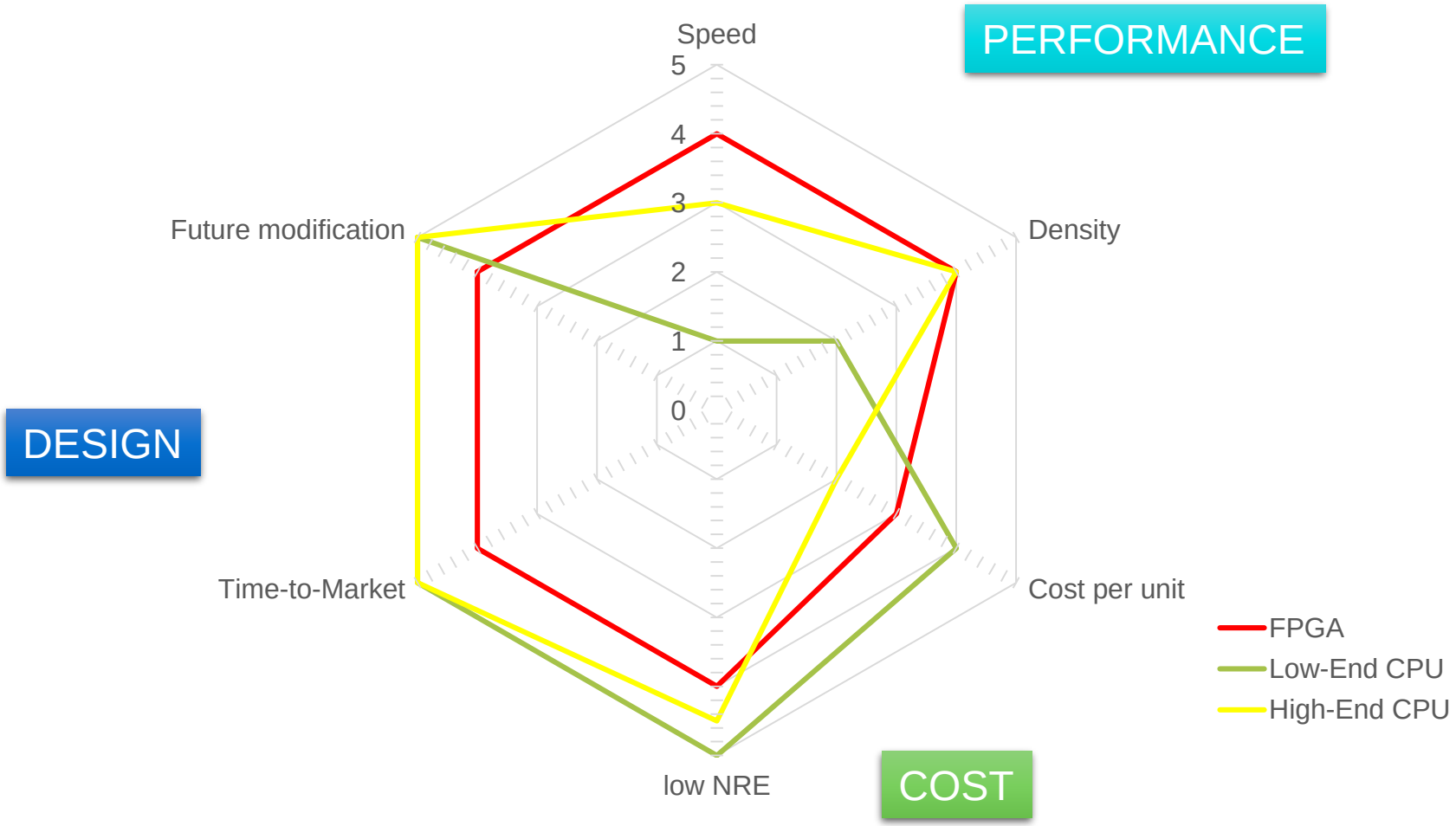


PERFORMANCE

DESIGN

COST

Comparison: FPGA vs CPU



Comparison: FPGA vs CPU

CPU: able to do everything…
but optimized for nothing

FPGA: multiple dedicated operators

Comparison: FPGA vs CPU

1st order Edge detection on 5Mpixels RGB picture:

- Compare FPGA and µC for high performance computing between a 1 GIPS processor and a 100 MHz FPGA using embedded multipliers

$$L_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} L \quad \text{and} \quad L_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} L$$

CPU:

- 5M x (6 additions+ 2 multiplications) x 2 directions x 3 colors @ 1GIPS
  => 240ms
- Picture shall be stored in memory before processing => ~15ms @ 60fps

FPGA:

- Could be done directly on the data flow outgoing from the picture sensor
- No processing time, only latency of 2 lines before starting to calculate

Summary

- Programmable architectures:
  - ✓ **Software – µP, DSP**
  - ✓ Hardware – ASIC
  - ✓ Hardware – CPLD
  - ✓ **Hardware – FPGA**

  - **How to choose ?**

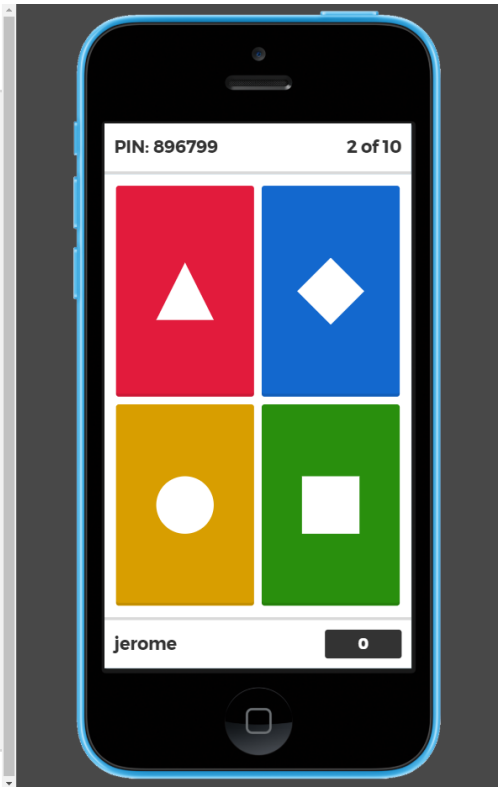
Summary

- Position the application:
  - Power consumption, speed, calculus …
  - Time-to-Market
  - Cost (how many units to be sold)
  - Long-term evolution
  - IOs

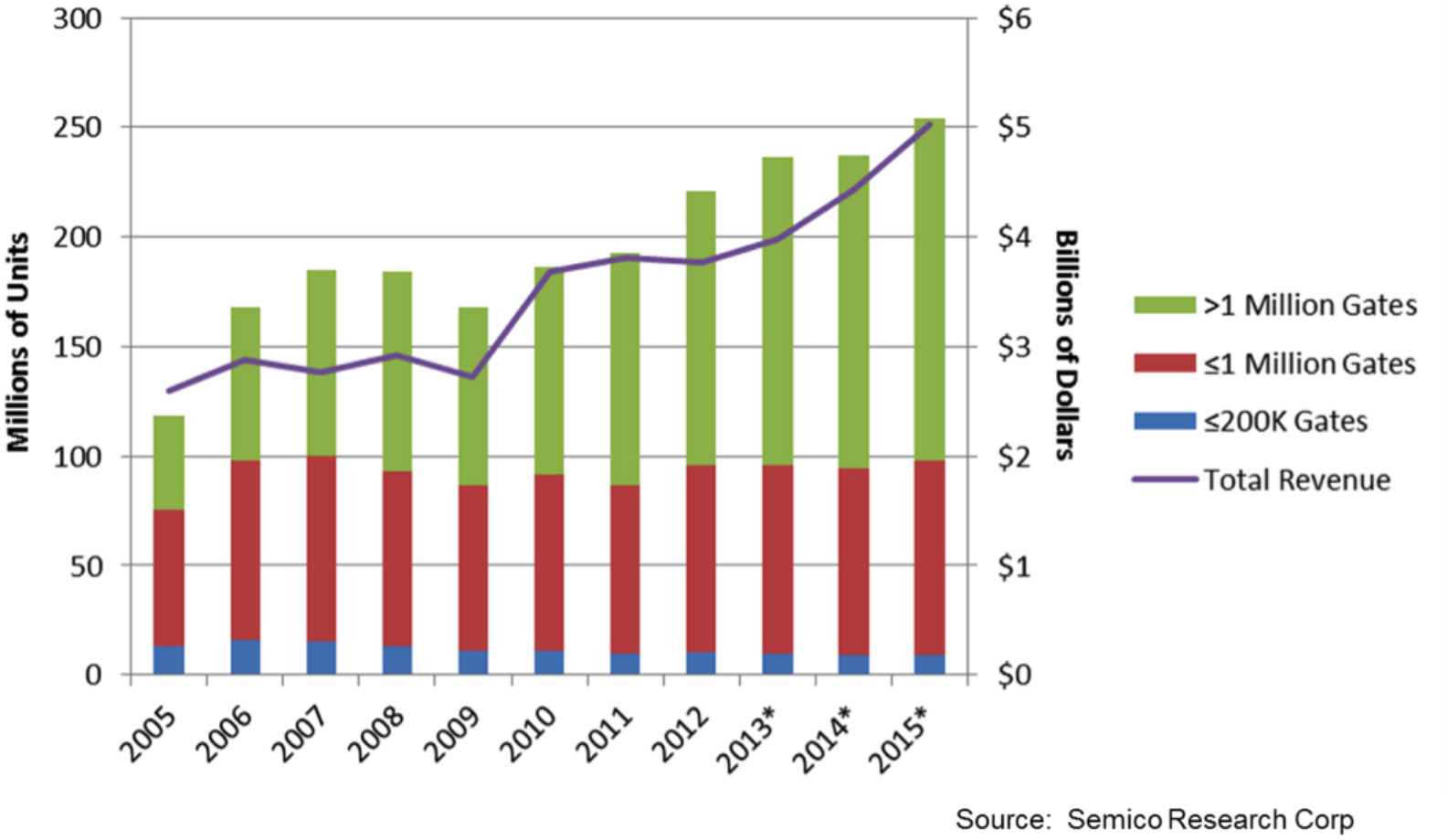
- Choose the right FPGA
  - Technology
  - Basic ressources
  - GPIOs and Gigabit IOs
  - Embedded ressources (multipliers, RAM …)
  - IP (Intellectual Property) availability

QUIZ

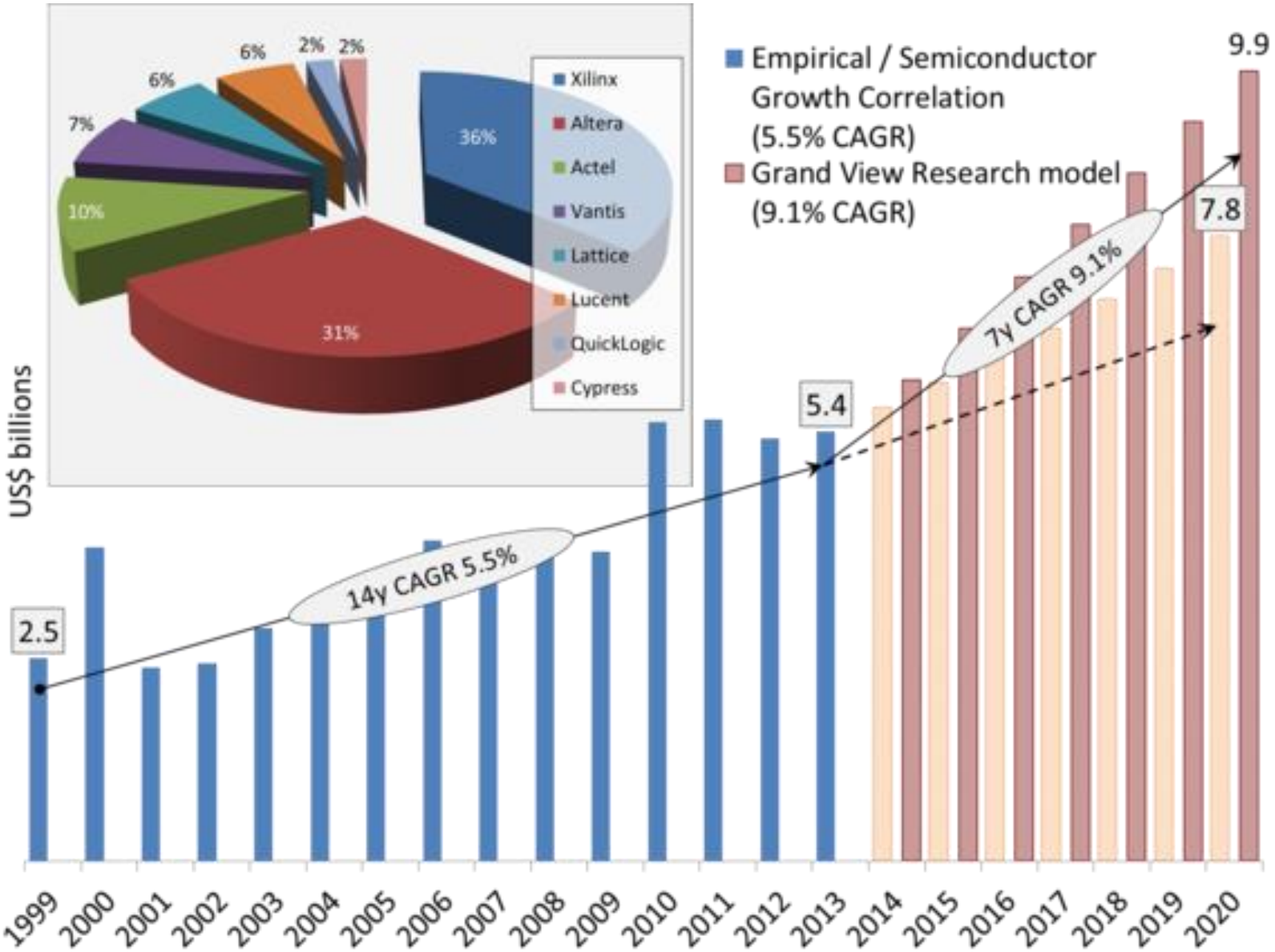
Let's go to https://kahoot.it/



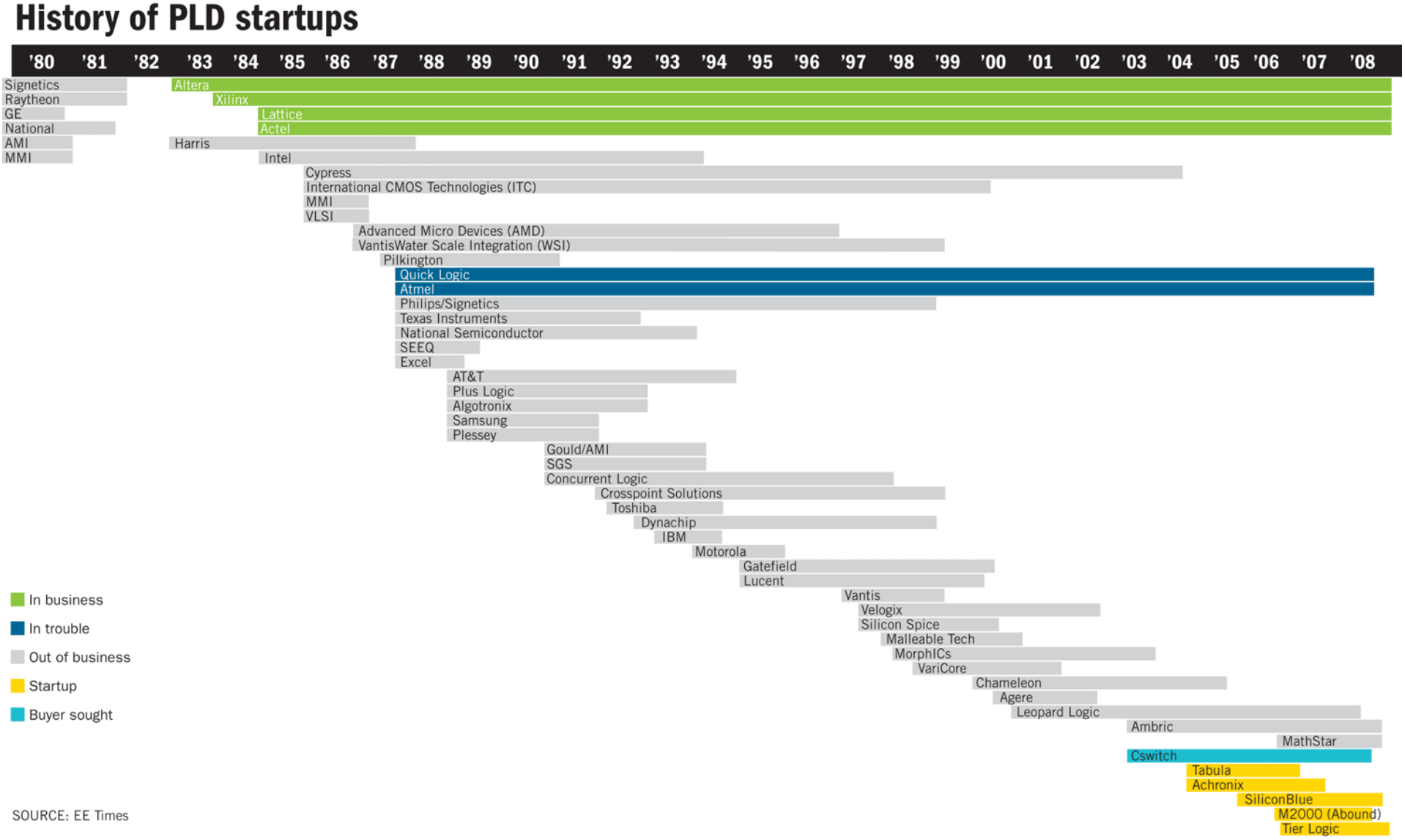
FPGA market


300
250
200
150
100
50
0
Millions of Units
$6
$5
$4
$3
$2
$1
$0
Billions of Dollars
2005
2006
2007
2008
2009
2010
2011
2012
2013*
2014*
2015*
>1 Million Gates
≤1 Million Gates
≤200K Gates
Total Revenue
Source: Semico Research Corp

FPGA market


6%
2%
2%
6%
7%
10%
36%
31%
Xilinx
Altera
Actel
Vantis
Lattice
Lucent
QuickLogic
Cypress
Empirical / Semiconductor Growth Correlation (5.5% CAGR)
Grand View Research model (9.1% CAGR)
US$ billions
9.9
7.8
5.4
2.5
14y CAGR 5.5%
7y CAGR 9.1%
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020

FPGA market


History of PLD startups
'80 '81 '82 '83 '84 '85 '86 '87 '88 '89 '90 '91 '92 '93 '94 '95 '96 '97 '98 '99 '00 '01 '02 '03 '04 '05 '06 '07 '08
Signetics
Raytheon
GE
National
AMI
MMI
Altera
Xilinx
Lattice
Actel
Harris
Intel
Cypress
International CMOS Technologies (ITC)
MMI
VLSI
Advanced Micro Devices (AMD)
VantisWater Scale Integration (WSI)
Pilkington
Quick Logic
Atmel
Philips/Signetics
Texas Instruments
National Semiconductor
SEEQ
Excel
AT&T
Plus Logic
Algotronix
Samsung
Plessey
Gould/AMI
SGS
Concurrent Logic
Crosspoint Solutions
Toshiba
Dynachip
IBM
Motorola
Gatefield
Lucent
Vantis
Velogix
Silicon Spice
Malleable Tech
MorphICs
VariCore
Chameleon
Agere
Leopard Logic
Ambric
MathStar
Cswitch
Tabula
Achronix
SiliconBlue
M2000 (Abound)
Tier Logic
In business
In trouble
Out of business
Startup
Buyer sought
SOURCE: EE Times

FPGA market

Main actors on the market:

Spartan, Virtex, Kintex, Artix

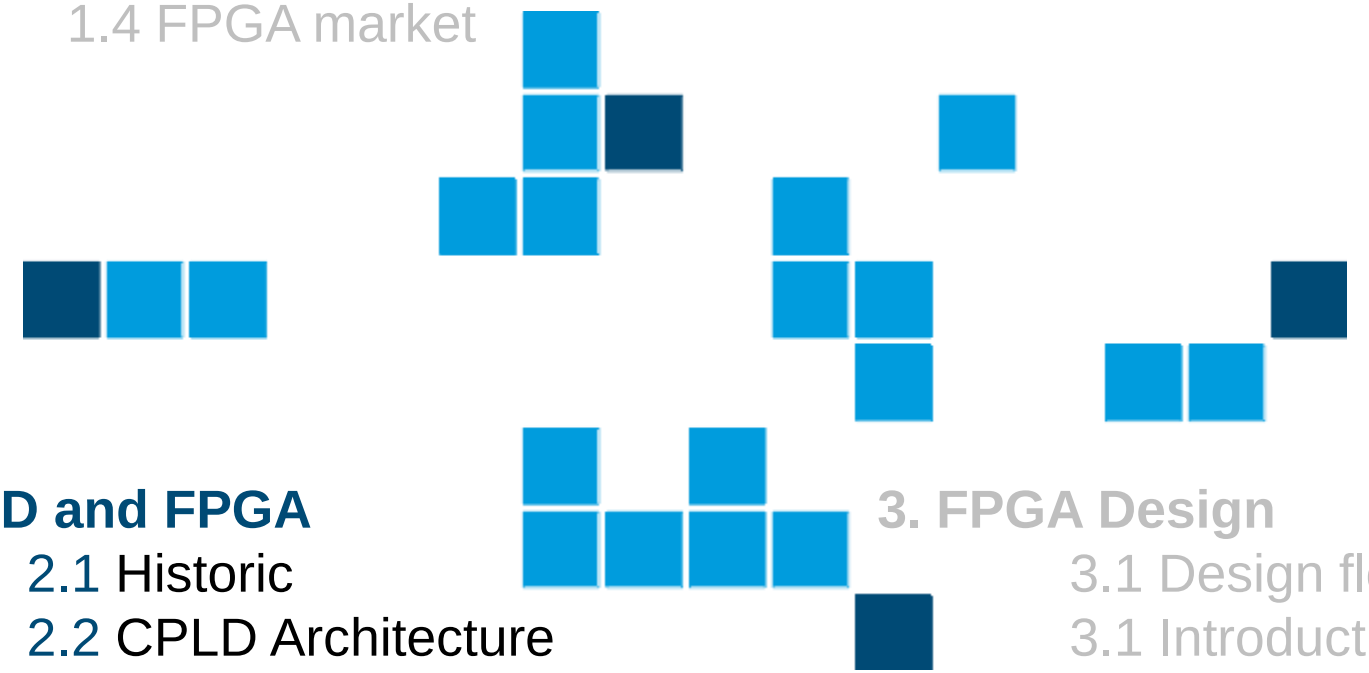
Stratix, Arria, Cyclone

Igloo, ProASIC3

iCE, MachXO, ECP

Outline

1. Introduction
    - 1.1 Different types of digital components
    - 1.2 Advantages and drawbacks of each type
    - 1.3 Selected applications
    - 1.4 FPGA market



2. CPLD and FPGA
    - 2.1 Historic
    - 2.2 CPLD Architecture
    - 2.3 FPGA Architecture
    - 2.3 Clock and timing management
    - 2.4 Other embedded ressources

3. FPGA Design
    - 3.1 Design flow
    - 3.1 Introduction to VHDL
    - 3.2 Simulations
    - 3.3 Finite State Machines

Historic

PAL (Programmable Array Logic) - **SPLD** (Simple Programmable Logic Device)

- oldest familly – Obsolet (-> 2000's)
- *Technology*:
  - Fuse or Anti-fuse (OTP: One Time Programmable)
  - EPROM (Erasable Programmable Read Only Memory) (UV erasable, 2005)
  - EEPROM (Electrically-Erasable Programmable Read-Only Memory)

CPLD (Complex Programmable Logic Device)

- *Technology:* EPROM or EEPROM

FPGA (Field Programmable Gate Array)

- *Technology: mainly* SRAM (*Random Acces Memory*)

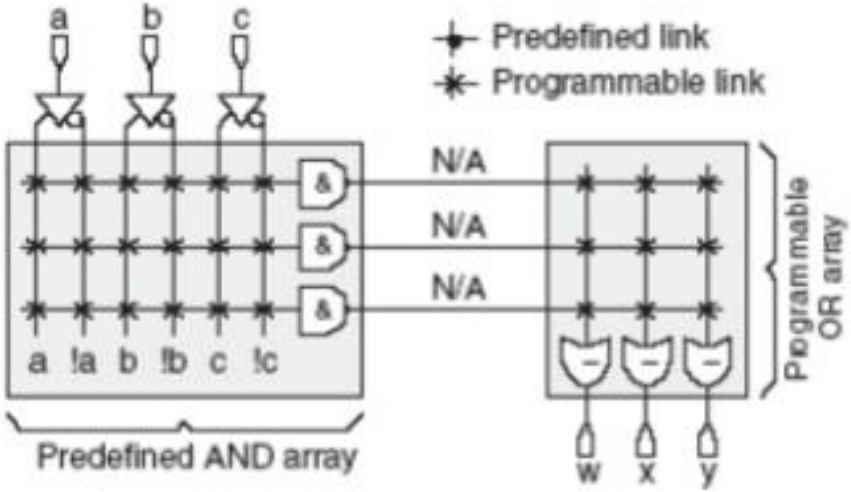
(Re)configuration ?

Time of apparition

(Re)configuration

Configuration	Reprogrammable?	Volatile?	Technology
Fuse	No	No	Bipolar
Anti-fuse	No	No	CMOS
EPROM	Yes (UV erasable)	No	UVCMOS
EEPROM	Yes (on circuit)	No	EECMOS
SRAM	Yes	yes	CMOS

SPLD architecture

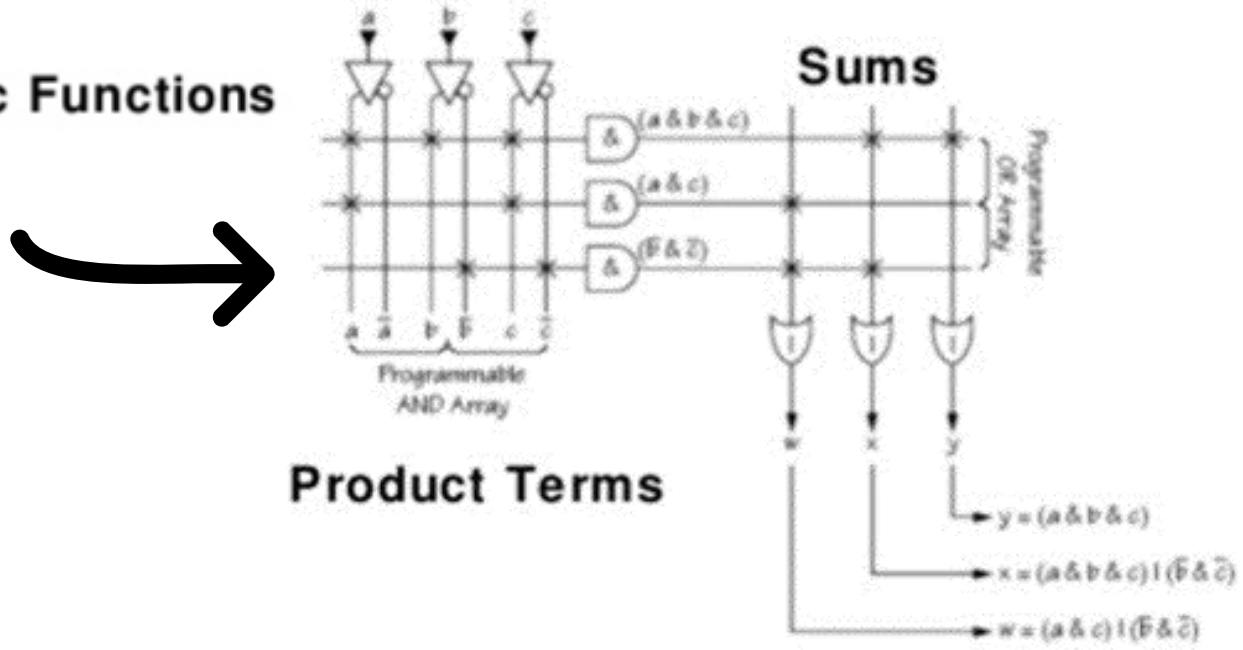


$y = (a \,\&\, b \,\&\, c)$

$x = (a \,\&\, b \,\&\, c) \mid (\bar{b} \,\&\, \bar{c})$

$w = (a \,\&\, c) \mid (\bar{b} \,\&\, \bar{c})$

Logic Functions



SPLD pros and cons

Pros:

Very effective approach to replace TTL logic

Can offer very high frequency performance

Cons:

OBSOLET

Lowest Density of all programmable devices

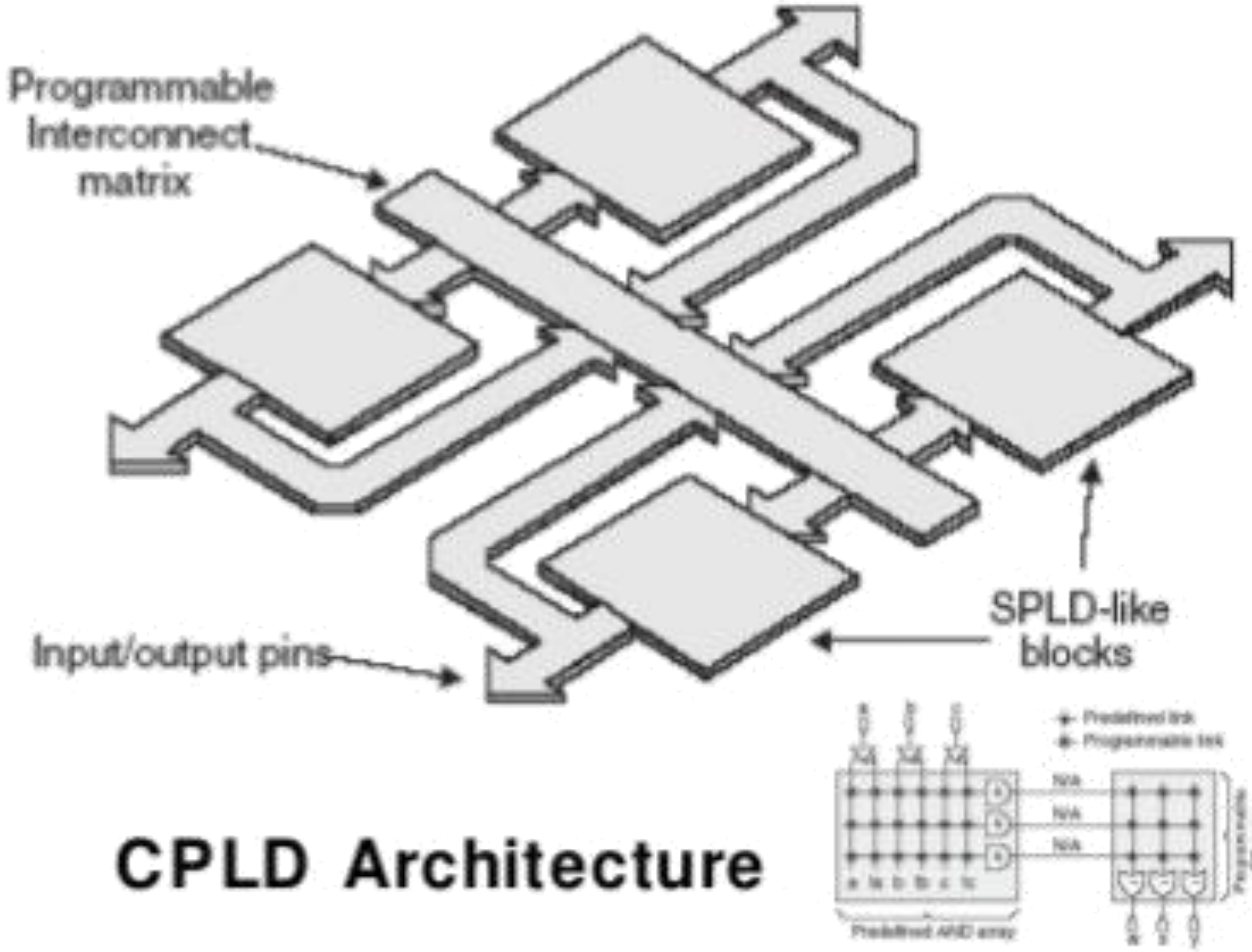
Not optimized:

multiply by n the number of I/O => multiply by n^2 the hardware ressources needed

Only a few I/O pins

CPLD architecture

CPLD = ~Multiple SPLD with a relatively small programmable interconnect

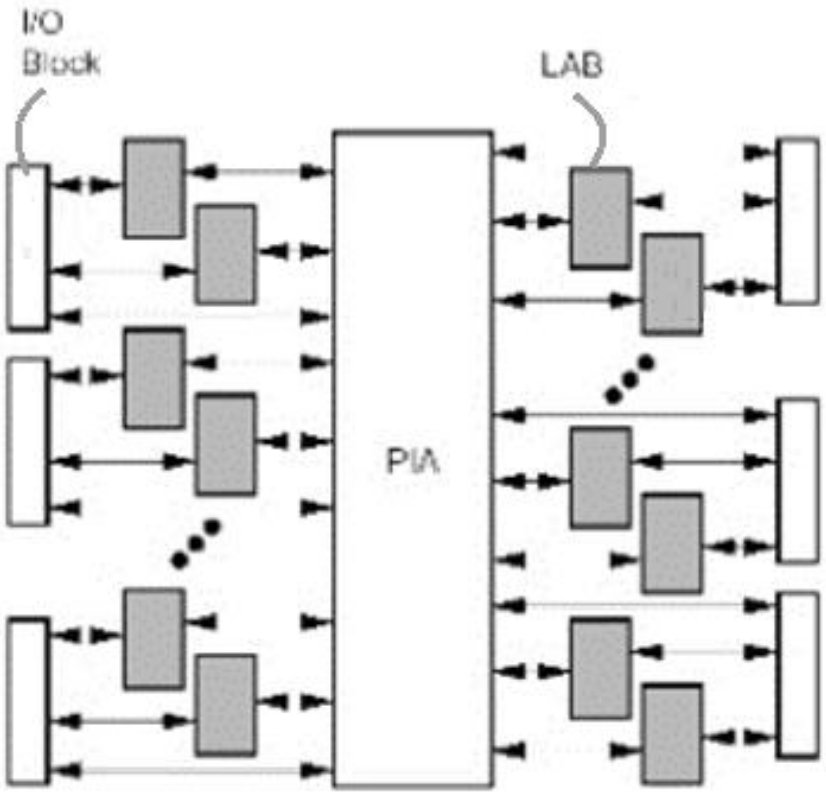


CPLD Architecture

CPLD architecture

Logic Array Blocks (LAB):

- Each LAB contains Macrocells (equivalent to 1 SPLD)
- Generally from 2 to 64 LABs par CPLD



Programmable Interconnect Array (PIA):

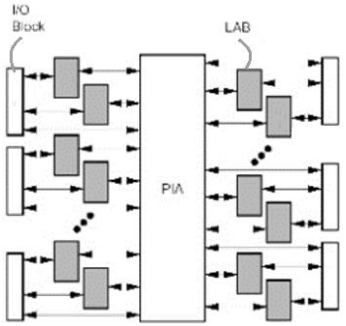
- Allows complex function to be dispatched into multiple LABs

I/O blocks:

- Large number of pins (up to 200 I/O pins)

CPLD architecture

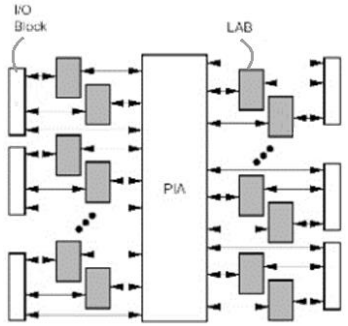
LAB (Logic Array Block): example of macrocell



Logic Array
Global Clear
Global Clock
Parallel Logic Expanders (from other macrocells)
Programmable Register
Register Bypass
to I/O Control Block
PRN
D/T
Q
ENA
CLRN
Clock/ Enable Select
VCC
Product-Term Select Matrix
Clear Select
to PIA
Shared Logic Expanders
36 Signals from PIA
16 Expander Product Terms

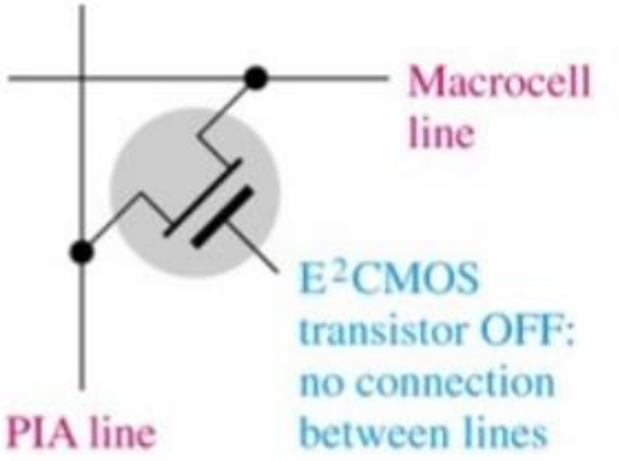
Output may be registered

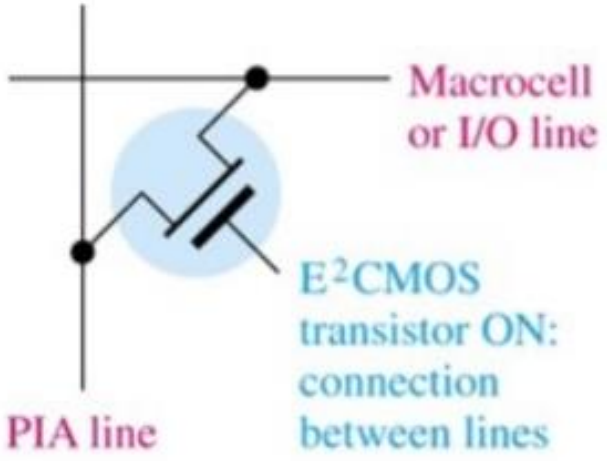
CPLD architecture



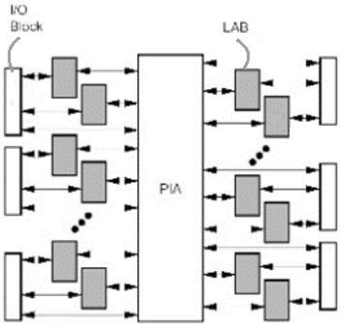
PIA (Programmable Interconnect Array)

- Allows complex function to be dispatched into multiple LABs





CPLD architecture



I/O blocks

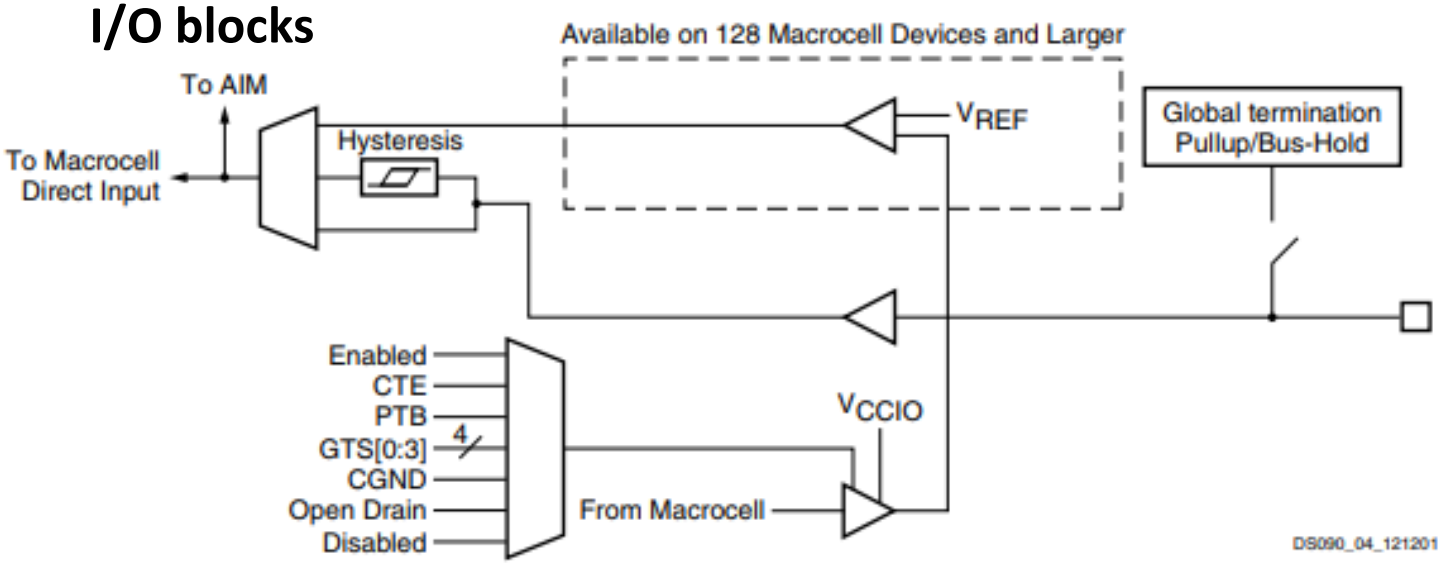


Figure 4: **CoolRunner-II CPLD I/O Block Diagram**

Table 5: **CoolRunner-II CPLD I/O Standard Summary**

IOSTANDARD Attribute	V_{CCIO}	Input V_{REF}	Board Termination Voltage (V_{TT})	Schmitt-trigger Support
LVTTL	3.3	N/A	N/A	Optional
LVCMOS33	3.3	N/A	N/A	Optional
LVCMOS25	2.5	N/A	N/A	Optional
LVCMOS18	1.8	N/A	N/A	Optional
LVCMOS15	1.5	N/A	N/A	Not optional
HSTL_1	1.5	0.75	0.75	Not optional
SSTL2_1	2.5	1.25	1.25	Not optional
SSTL3_1	3.3	1.5	1.5	Not optional

CPLD example

Example: Xilinx CPLD families

Features	CoolRunner-II	XC9500XL
Core Voltage	1.8	3.3/2.5
Macrocells	32-512	36-288
I/Os	21-270	34-192
I/O Tolerance	1.5V, 1.8V, 2.5V, 3.3V	5.0V (XL), 3.3V, 2.5V
TPD / f max (fastest)	3.8/323	5/222
Ultra Low Standby Power	28.8µW*	Low power mode
I/O Standards	LVTTL, LVCMOS, HSTL, SSTL	LVTTL, LVCMOS

CPLD pros and cons

Advantages and drawbacks?

Pros :

Density - CPLD replaces 100.000 individual gates

Large number of I/O pins

High frequency - Predictible timing

Ideal for interfacing or scheduling

Cons :

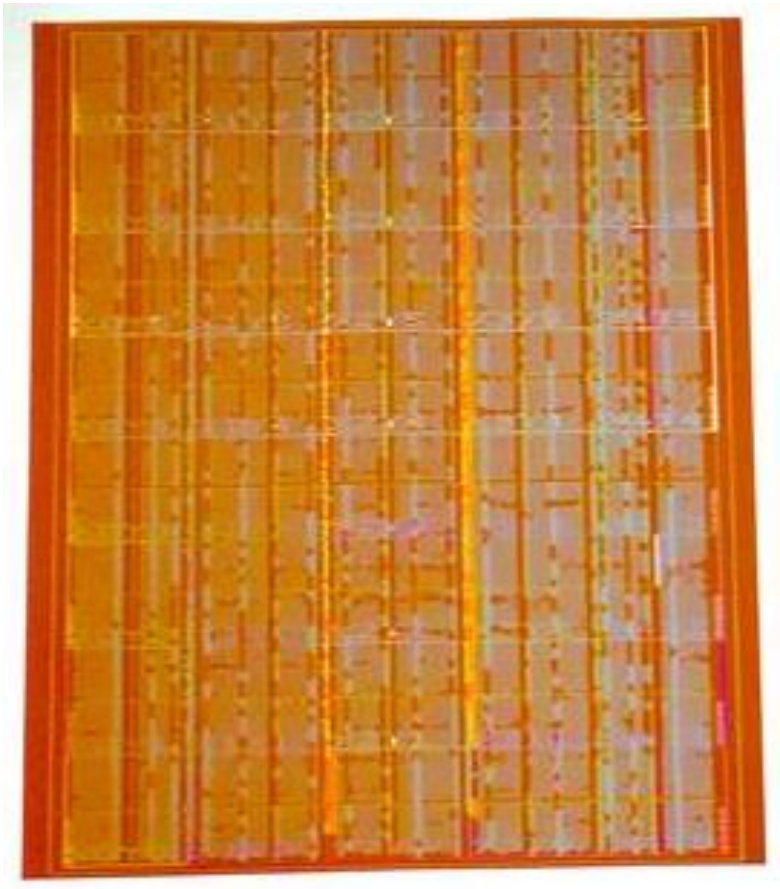
Not enough logic ressources for computation

Cost exponentially increase with complexity

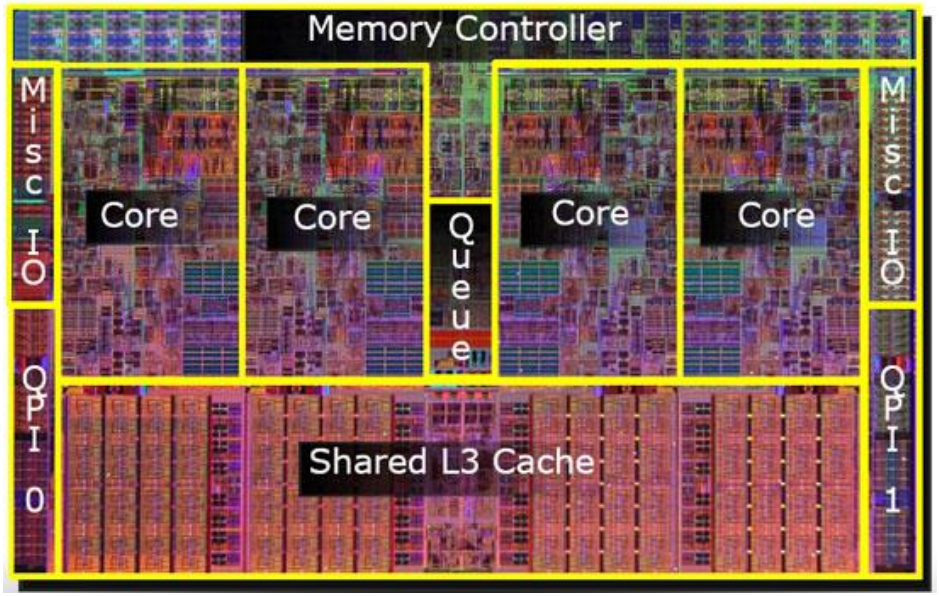
FPGA architecture

1. Global presentation
2. Logic block
3. Interconnexion
4. I/O
5. Clock and timing management
6. Other embedded ressources
7. Miscellanous

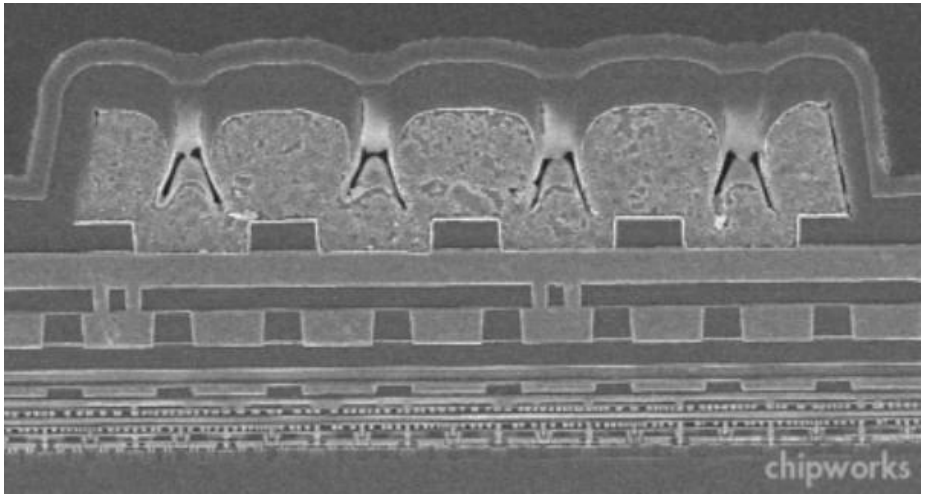
FPGA architecture



Xilinx UltraScale VU095

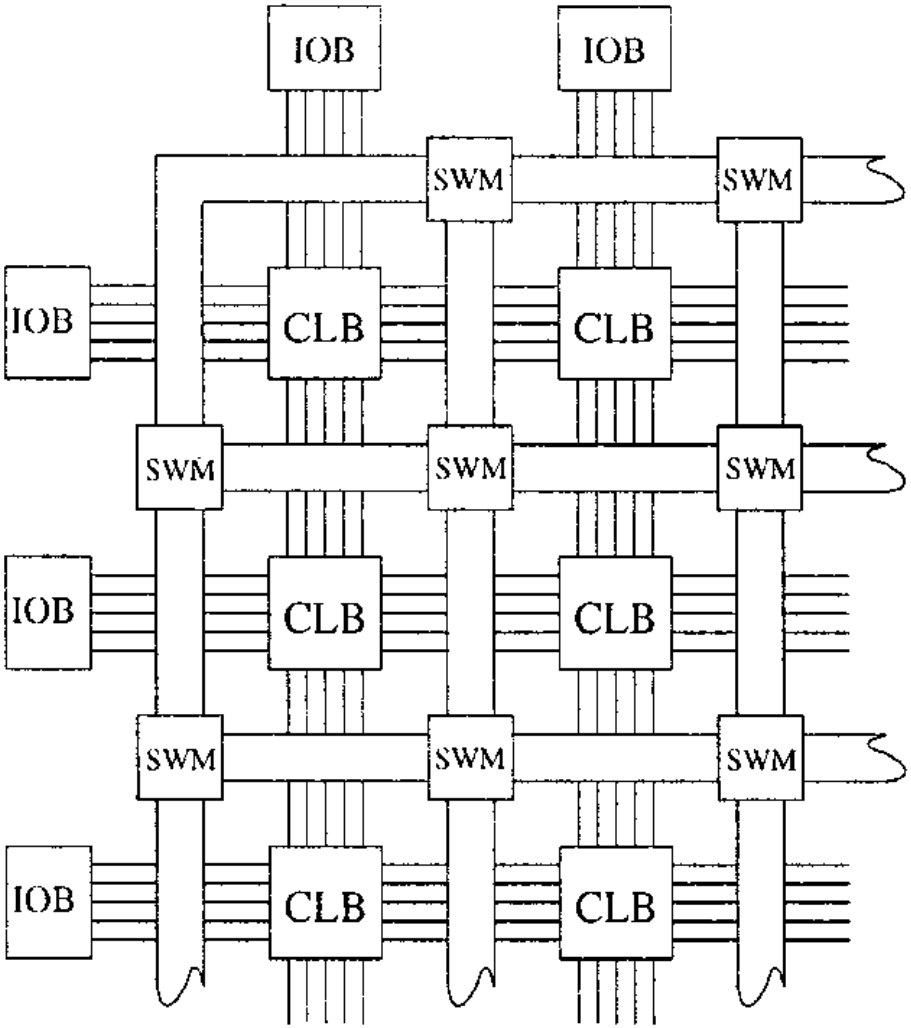


Intel Core i7



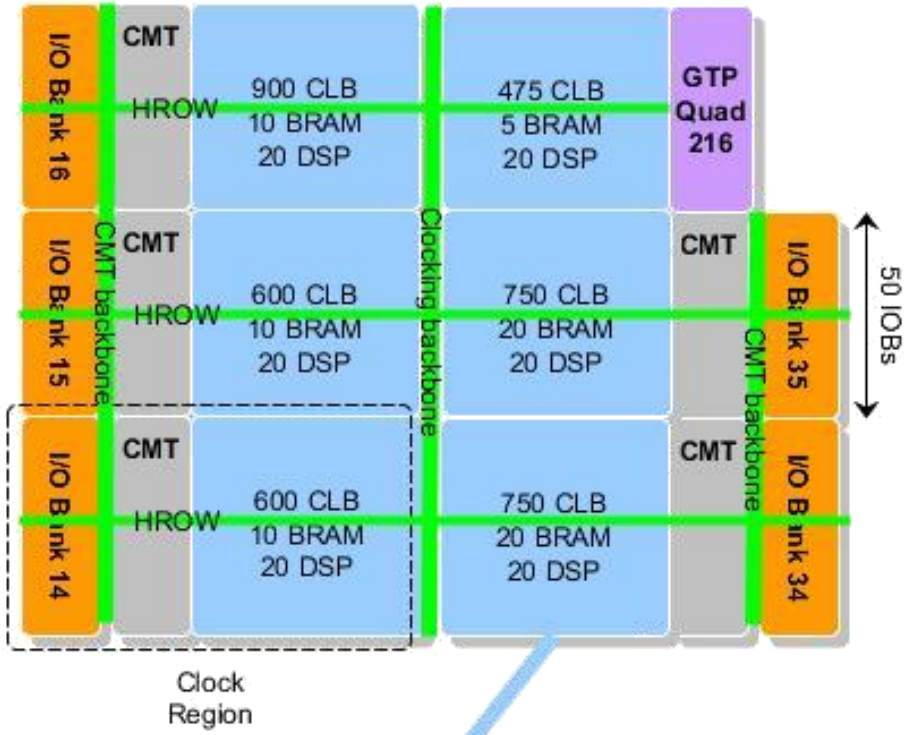
TMSC 28 nm

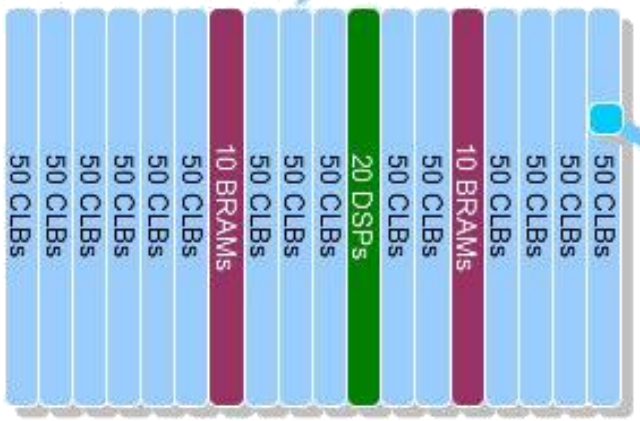
FPGA architecture

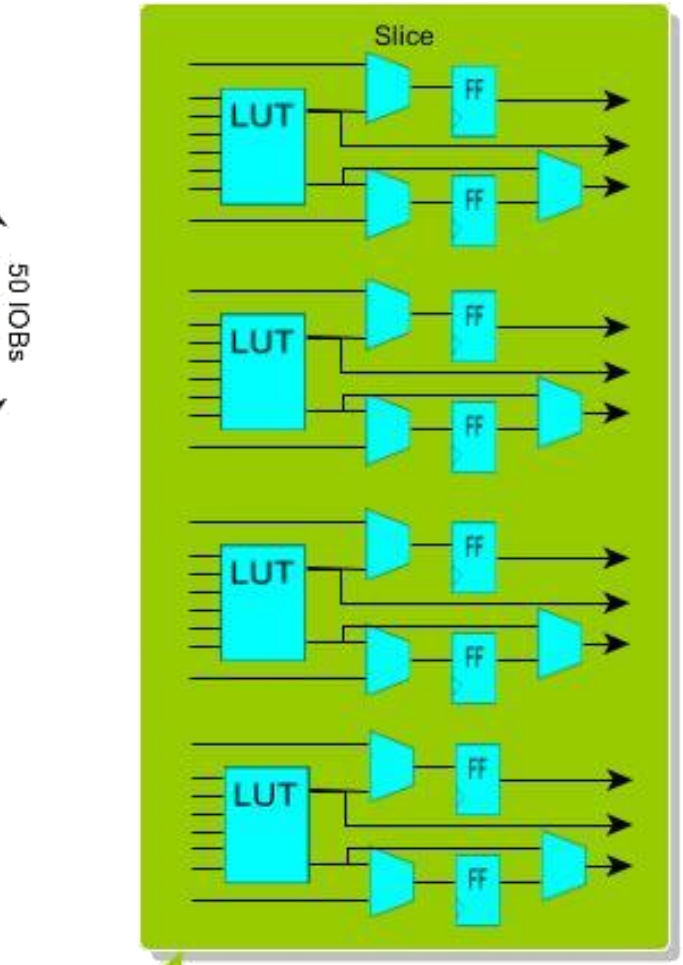


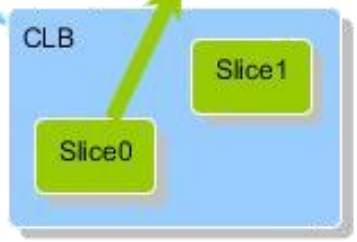
Lowest granularity of logic cell compared to CPLD

FPGA architecture







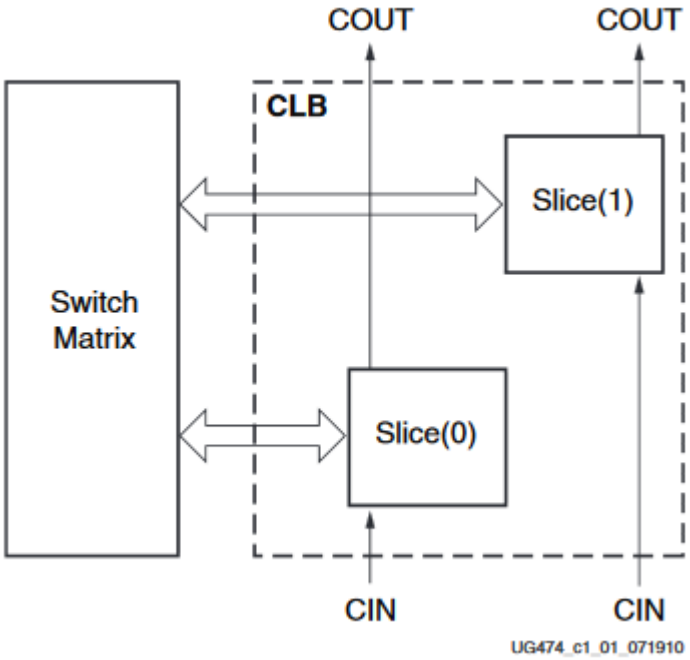


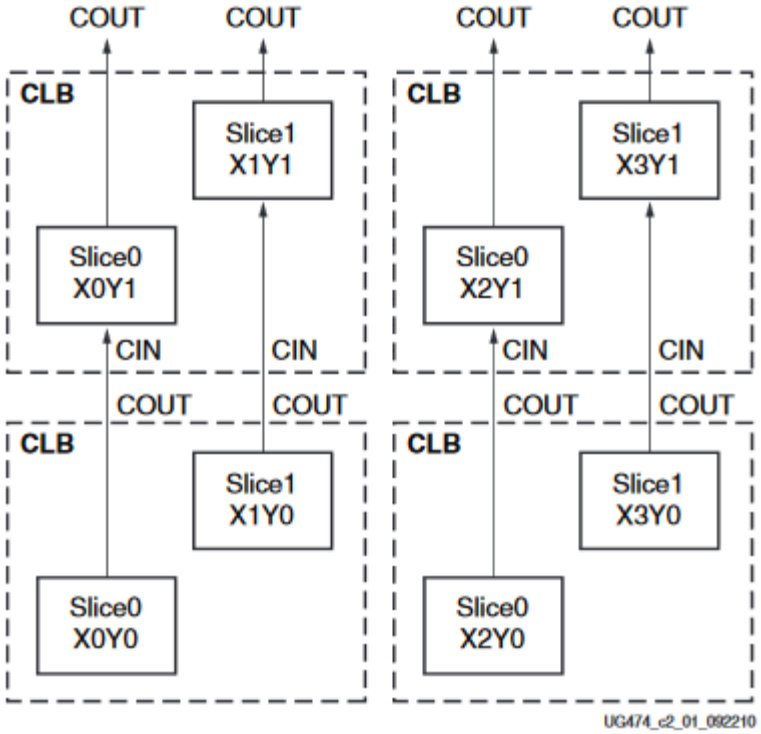
FPGA architecture

1. Global presentation
2. Logic block
3. Interconnexion
4. I/O
5. Clock and timing management
6. Other embedded ressources
7. Miscellanous

FPGA - Logic block

CLB: Configurable Logic Block





Every slice is connected to the switch matrix

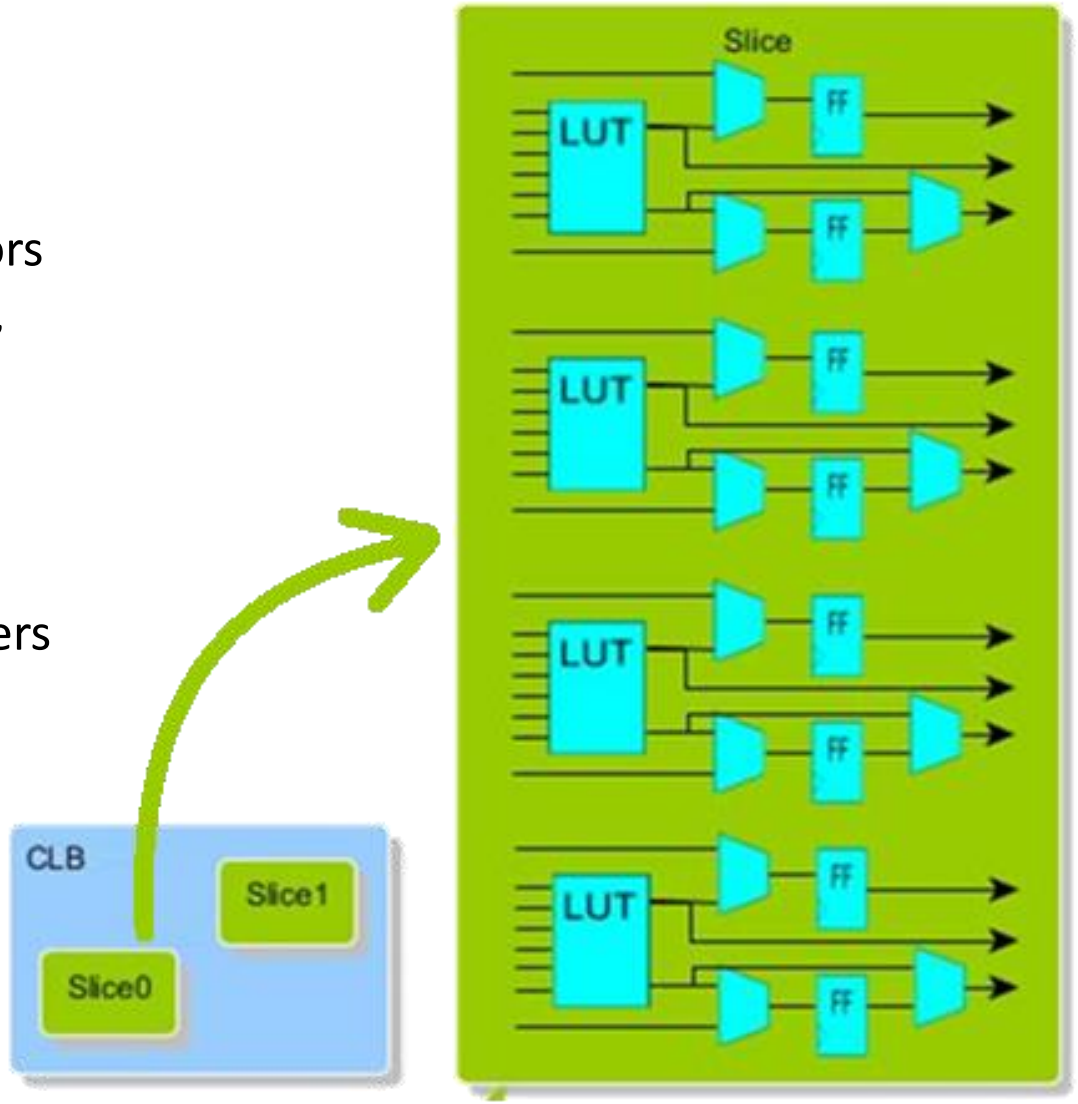
Carry chain between slices of adjacent CLBs

FPGA – Logic block

Every CLB contains 2 slices

Every slice contains:

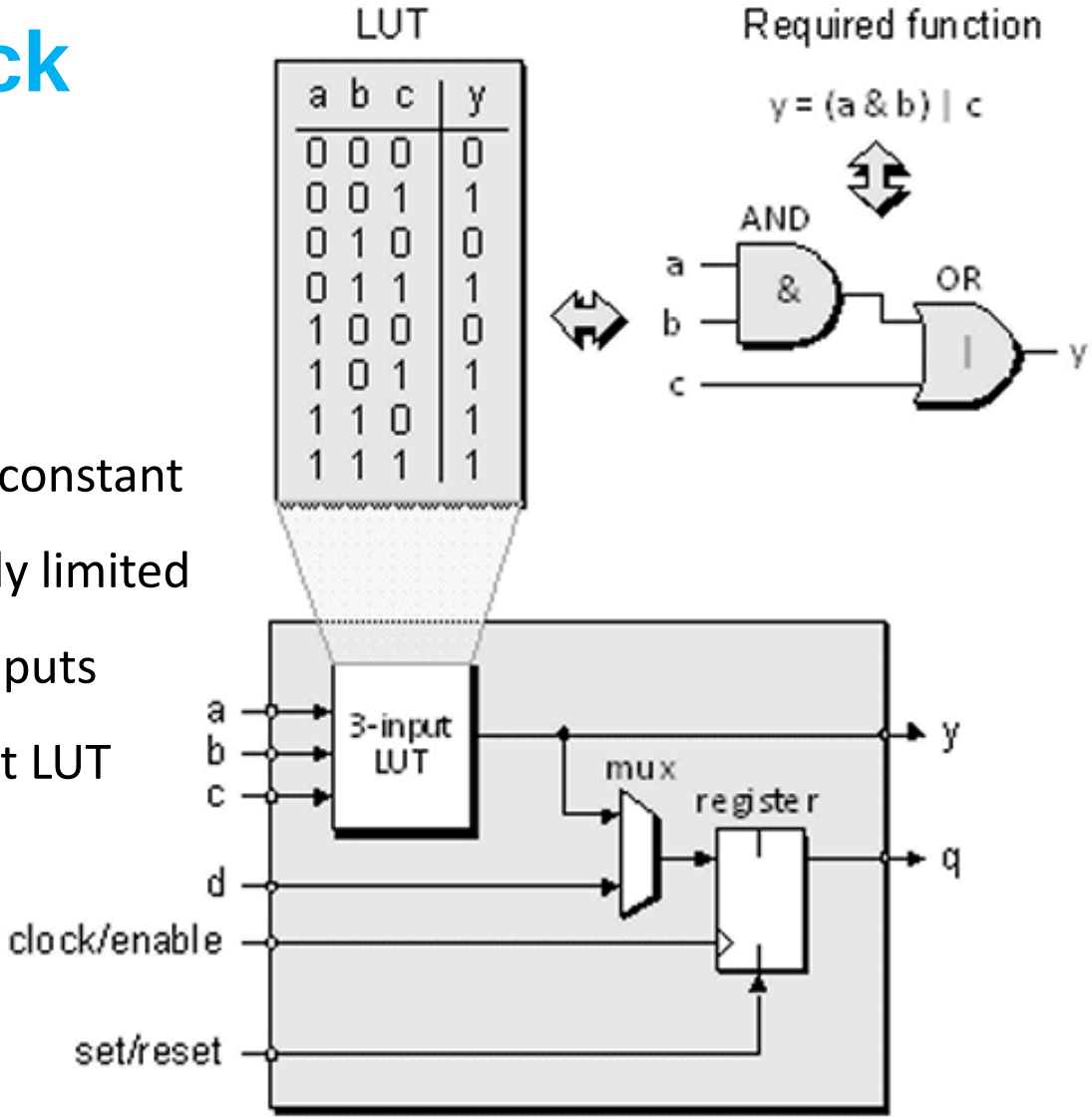
- 4 logic-function generators
  - look-up tables “LUT”
- Eight storage elements
  - Flip-flop “FF”
- Wide-function multiplexers
- Carry logic



FPGA – Logic block

Look-Up Tables (LUT):

- ✓ All logic is done here
- ✓ SRAM Memory
- ✓ Propagation delay is constant
- ✓ The complexity is only limited by the number of inputs
- ✓ Generally 4 to 6 input LUT



- ✓ Can be used as multiplexer
- ✓ Or as distributed RAM

FPGA architecture

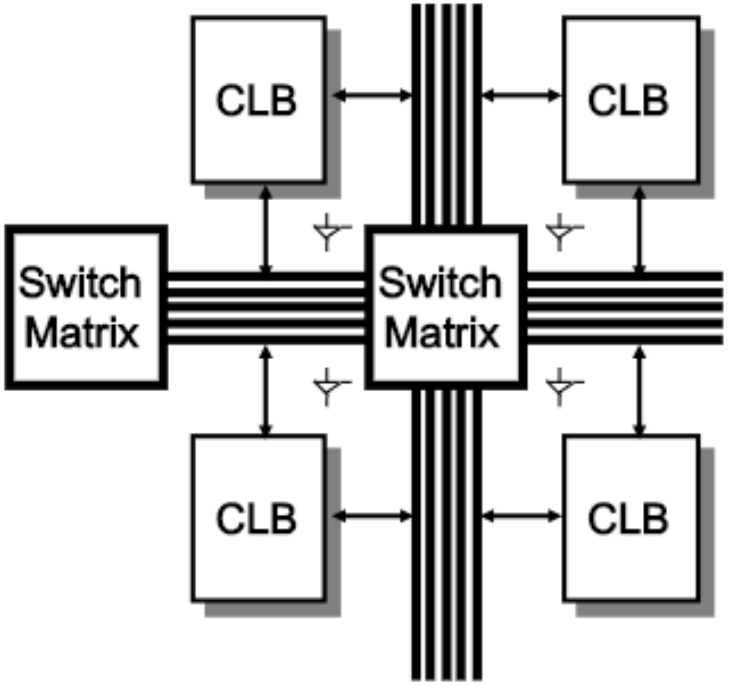
1. Global presentation
2. Logic block
3. Interconnexion
4. I/O
5. Clock and timing management
6. Other embedded ressources
7. Miscellanous

FPGA - Interconnexion

Interconnexion

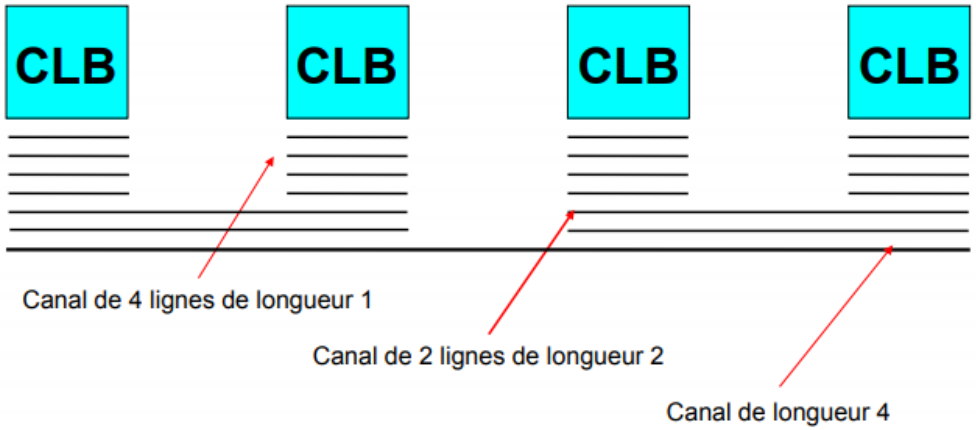
- ✓ Between CLB
- χ Introduce delay ;

  => more delay if CLB are distant



Different types of interconnexion

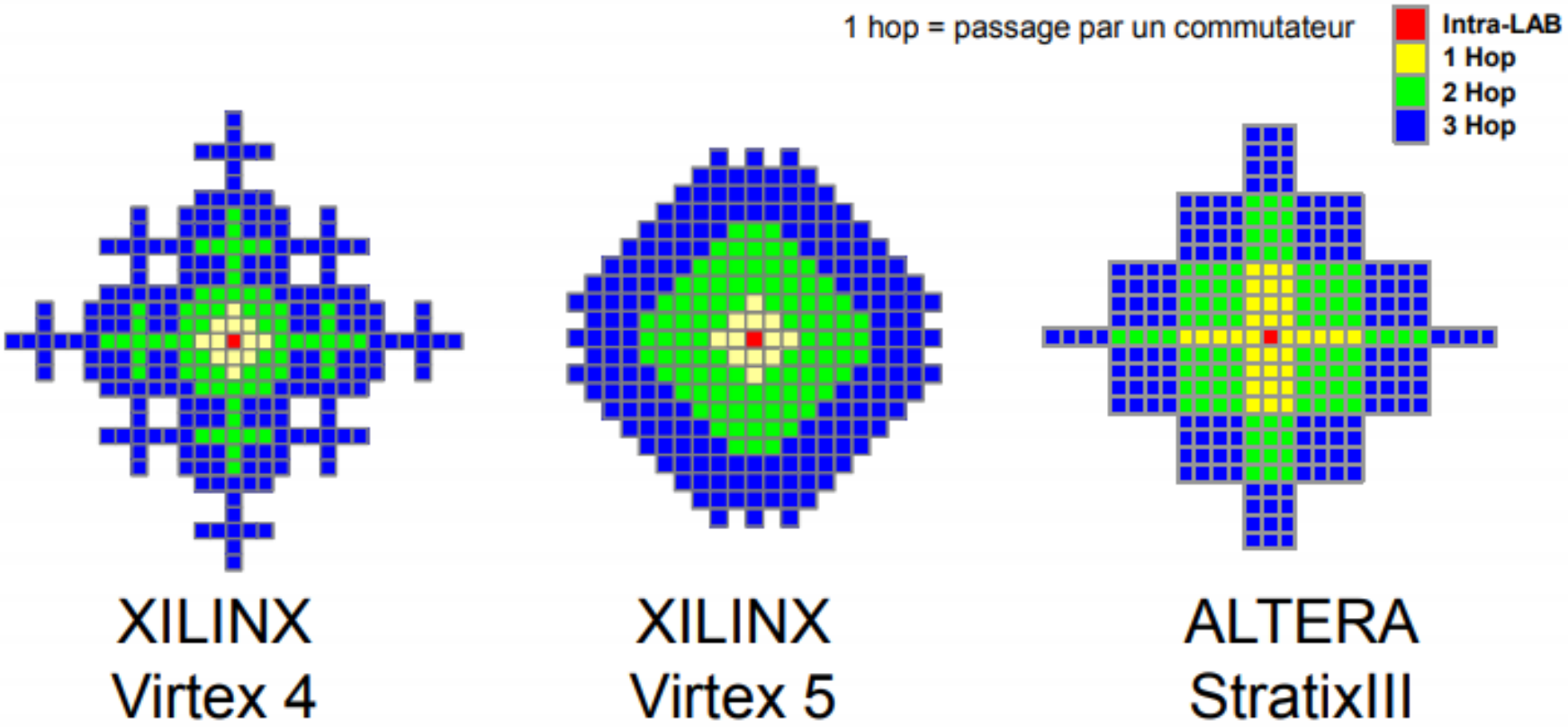
- Fast direct interconnexion: CLB to CLB
- General Purpose Interconnexion: uses switch matrix
- Long lines
- Global clock distribution
- Horizontal clock distribution



FPGA - Interconnection

Interconnexion

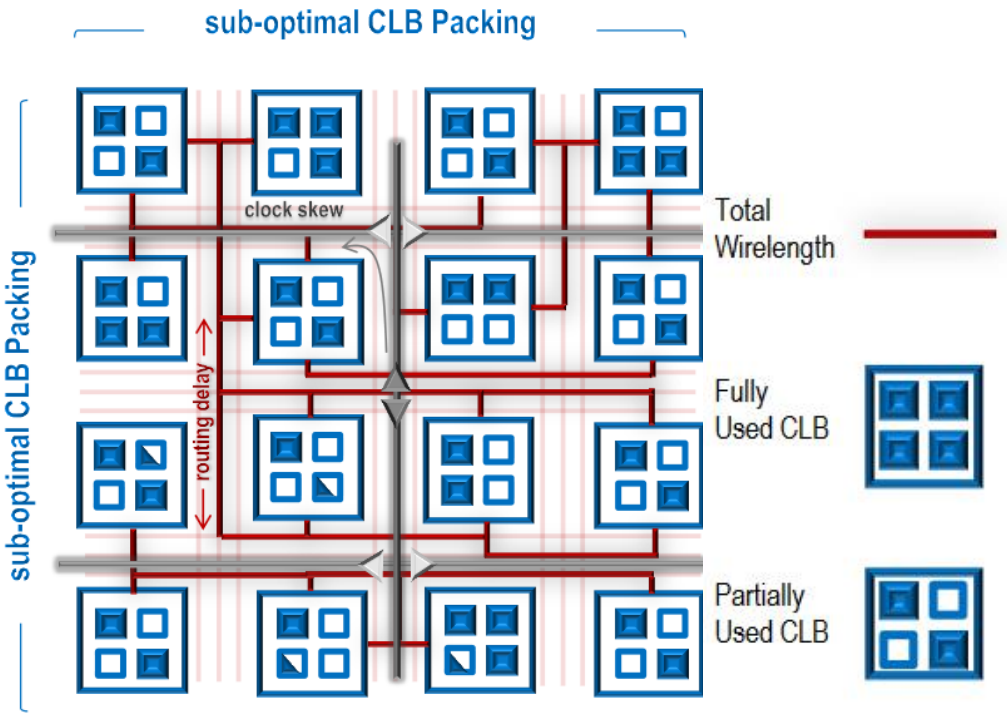
More and more interconnection to reduce overall length

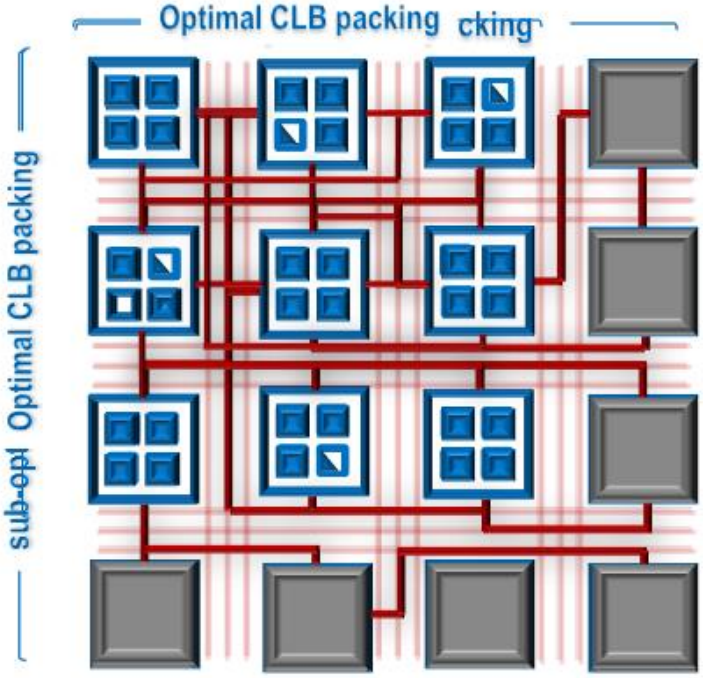


FPGA - Interconnexion

Interconnexion

- CLB packing should be optimized

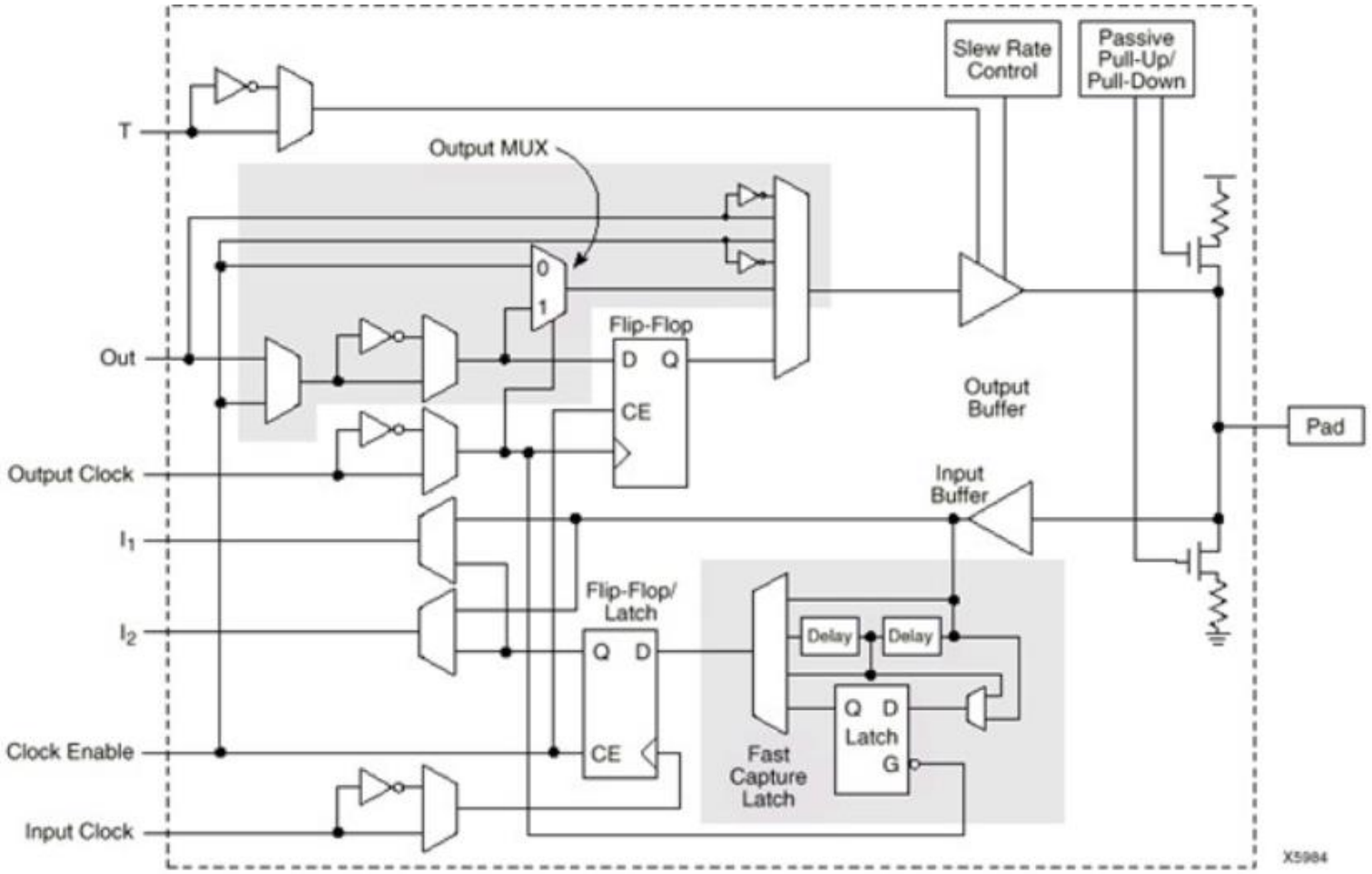




FPGA architecture

1. Global presentation
2. Logic block
3. Interconnexion
4. I/O
    a. I/O blocks
    b. Parallel and serial link
    c. Analogic IO
5. Clock and timing management
6. Other embedded ressources
7. Miscellanous

FPGA – I/O block

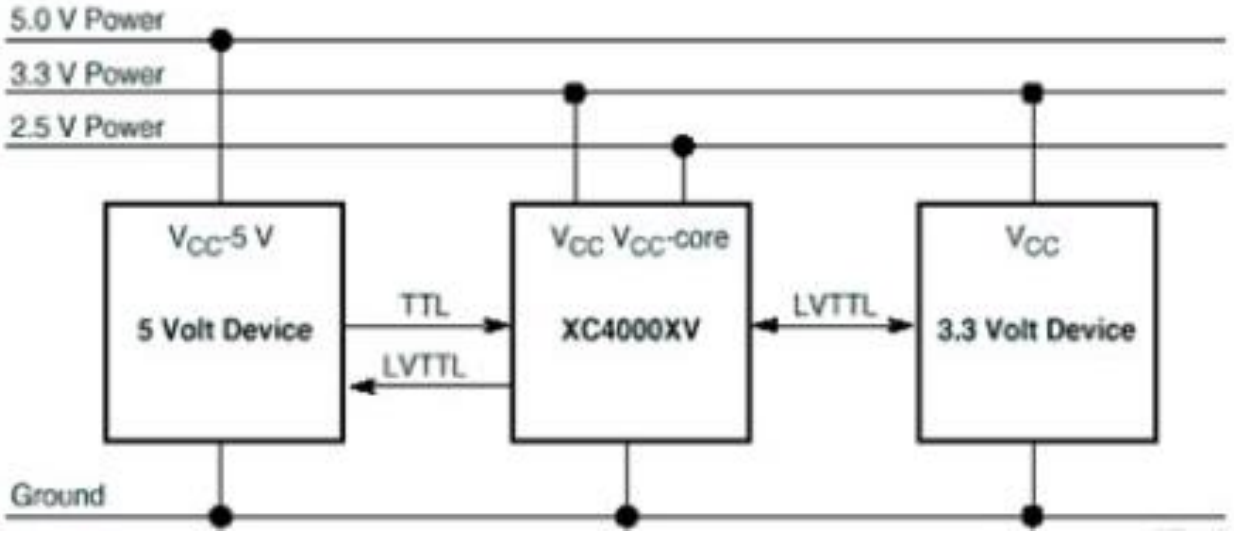


Can be configured as input or output

Single ended configuration (image) or can be used as a differential port

FPGA - I/O block

I/O block: Multiple standard supported



	V_{OUT}_max	V_{IH}	V_{IL}	V_{OH}	V_{OL}
TTL	5.5	2.0	0.8	2.4	0.4
LVTTL	3.6	2.0	0.8	2.4	0.4
LVCMOS	3.6	Vcc*0.5	Vcc*0.3	Vcc*0.9	Vcc*0.1

FPGA – IOs

Table 1-1: **Supported Features in the HR and HP I/O Banks**

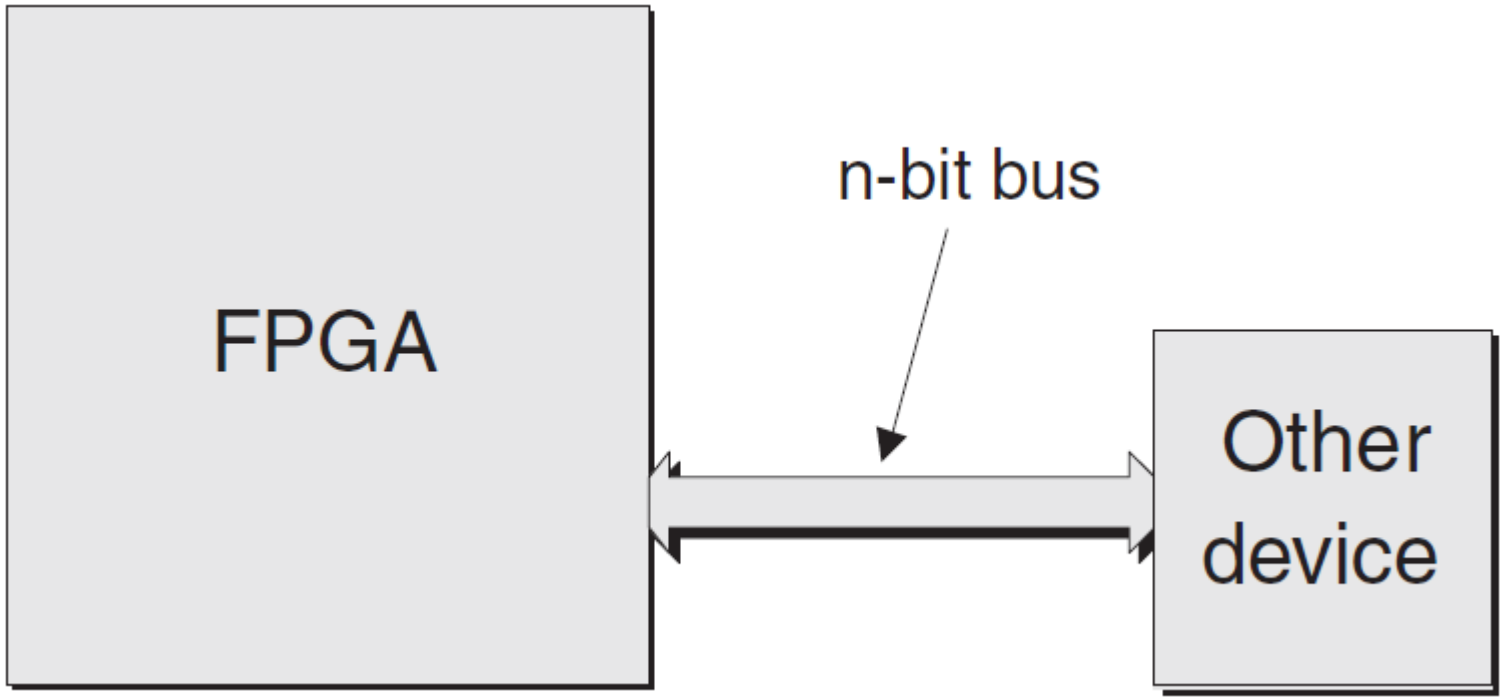
Feature	HP I/O Banks	HR I/O Banks
3.3V I/O standards(1)	N/A	Supported
2.5V I/O standards(1)	N/A	Supported
1.8V I/O standards(1)	Supported	Supported
1.5V I/O standards(1)	Supported	Supported
1.35V I/O standards(1)	Supported	Supported
1.2V I/O standards(1)	Supported	Supported
LVDS signaling	Supported(2)	Supported
24 mA drive option for LVCMOS18 and LVTTL outputs	N/A	Supported
V_{CCAUX_IO} supply rail	Supported	N/A
Digitally-controlled impedance (DCI) and DCI cascading	Supported	N/A
Internal V_{REF}	Supported	Supported

HP: High Performance / HR: High Range

FPGA architecture

1. Global presentation
2. Logic block
3. Interconnexion
4. I/O
    a. I/O blocks
    b. Parallel and serial link
    c. Analogic IO
5. Clock and timing management
6. Other embedded ressources
7. Miscellanous

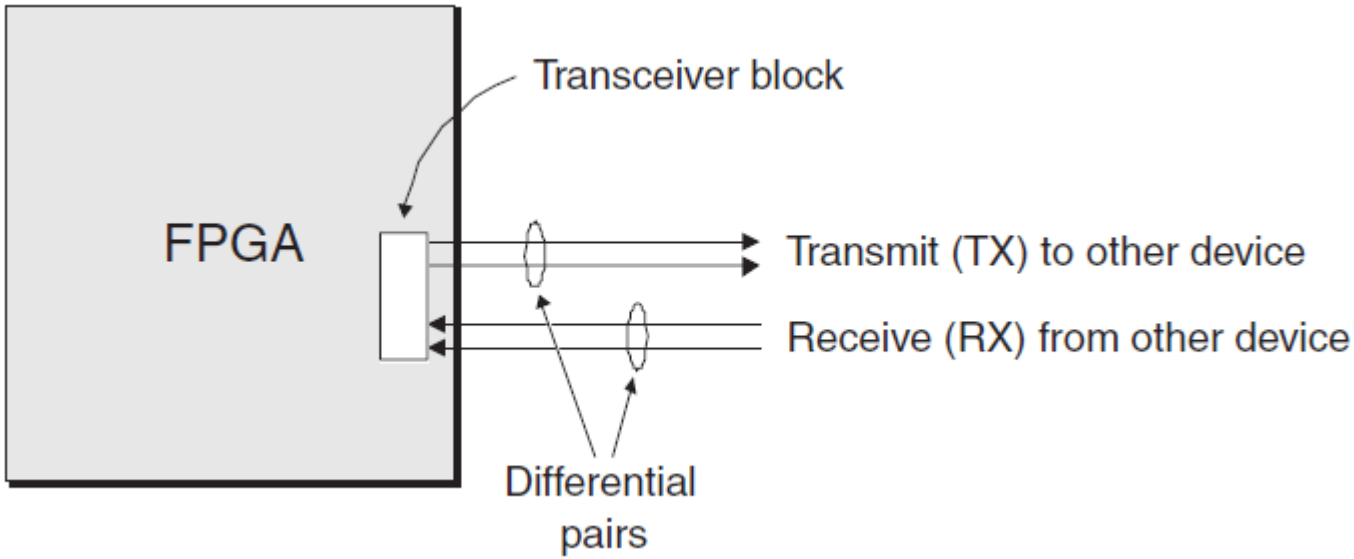
FPGA – IOs



Parallel communication bus

Advantages and drawbacks ?

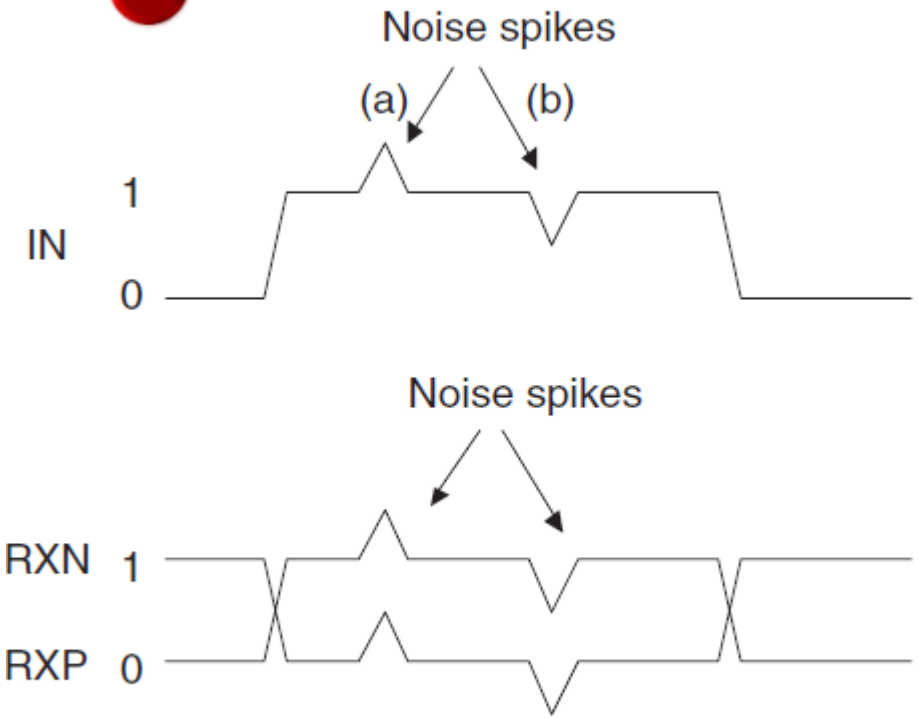
FPGA – IOs

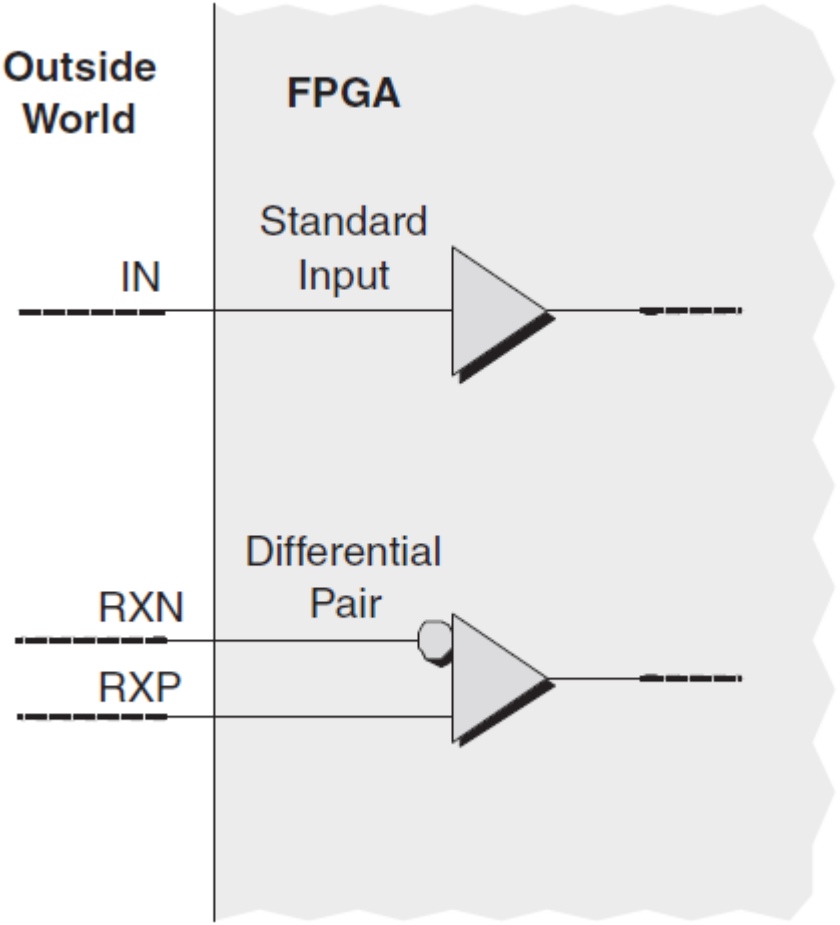


Modern serial buses

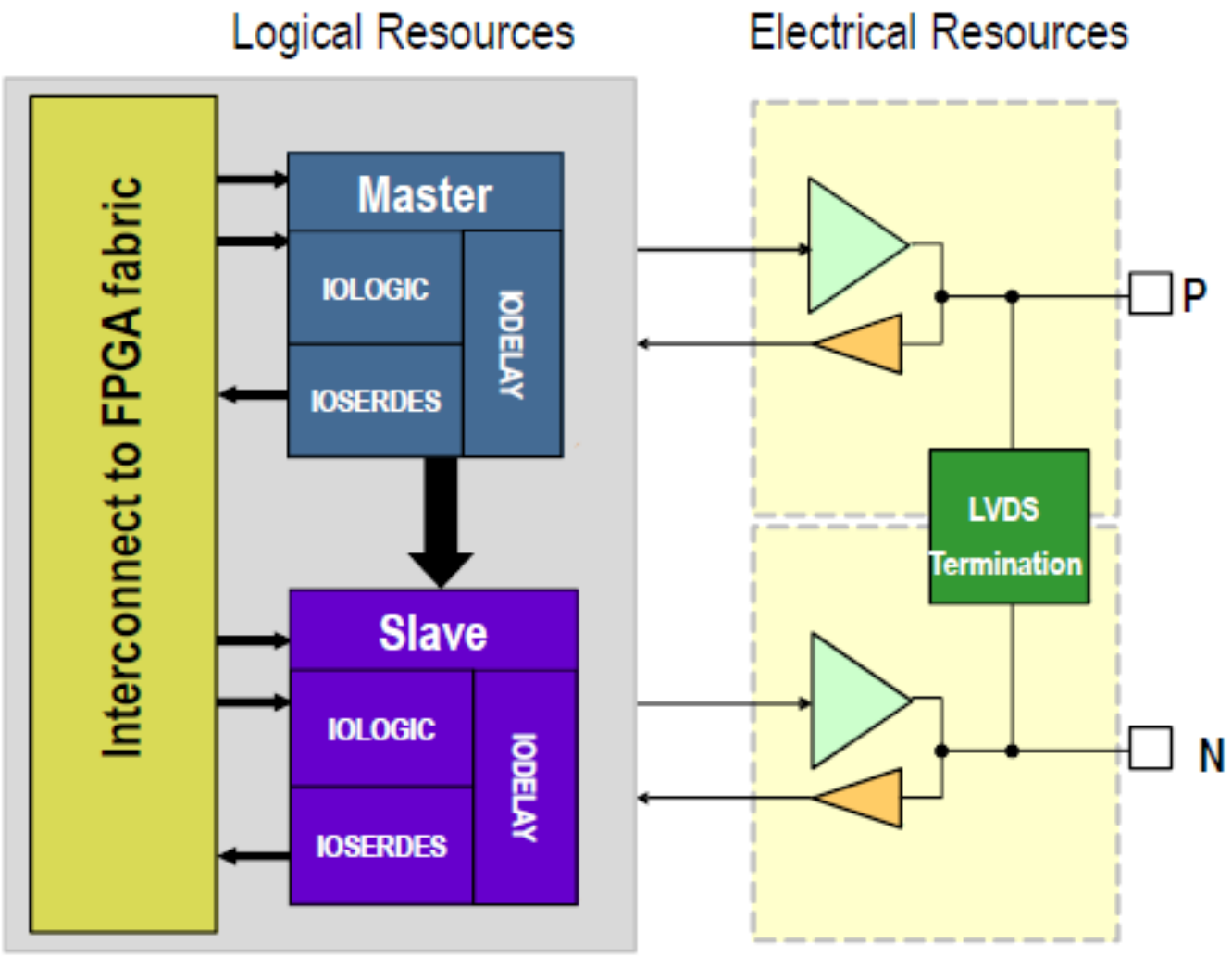
FPGA – IOs

Why serial buses ?



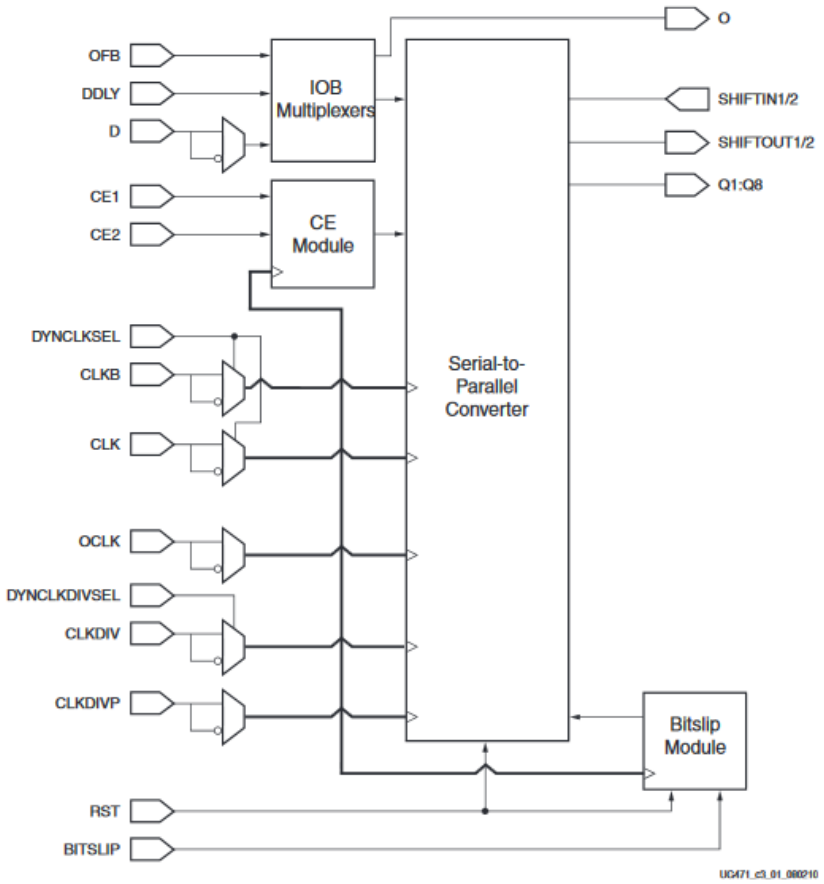


FPGA – IOs


Logical Resources
Electrical Resources
Interconnect to FPGA fabric
Master
IOLOGIC
IODELAY
IOSERDES
Slave
IOLOGIC
IODELAY
IOSERDES
LVDS
Termination
P
N

FPGA – IOs

Artix-7 example: ISERDESE2 and OSERDESE2 primitives



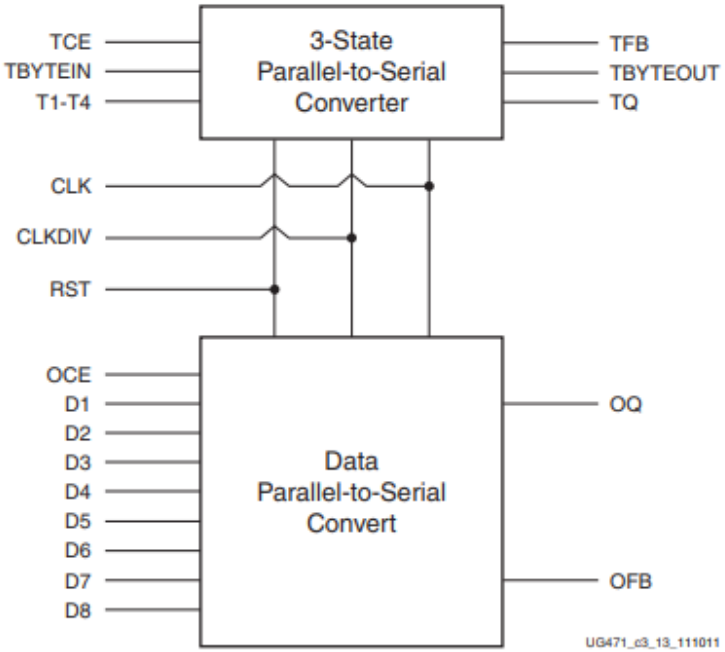


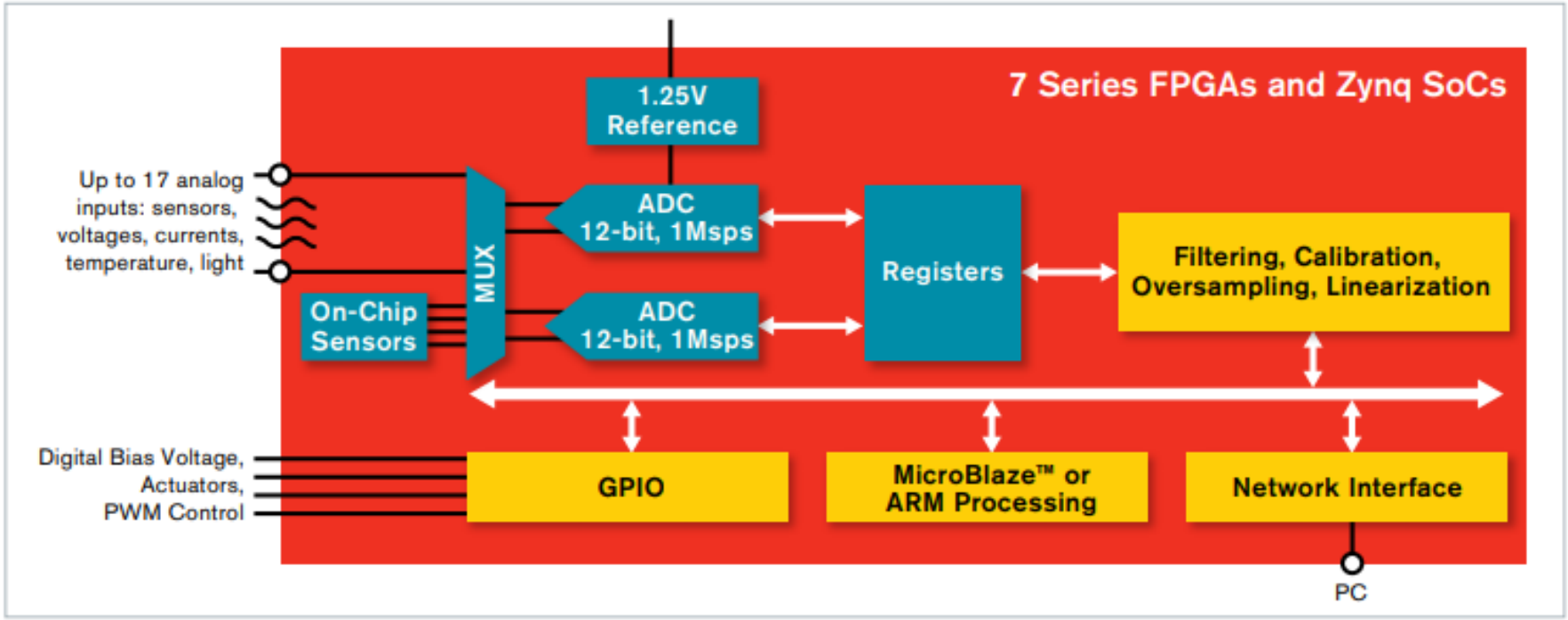
Figure 3-13: **OSERDESE2 Block Diagram**

FPGA architecture

1. Global presentation
2. Logic block
3. Interconnexion
4. I/O
    a. I/O blocks
    b. Parallel and serial link
    c. Analogic IO
5. Clock and timing management
6. Other embedded ressources
7. Miscellanous

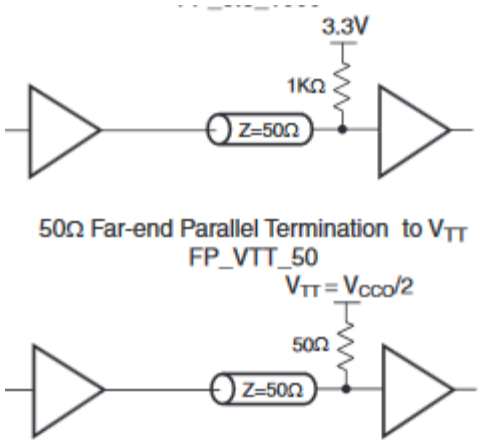
FPGA – IOs

ANALOG MIXED SIGNAL BLOCK DIAGRAM

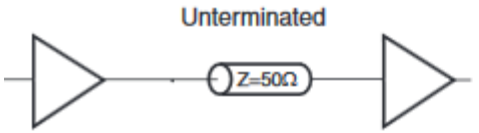


FPGA – IOs

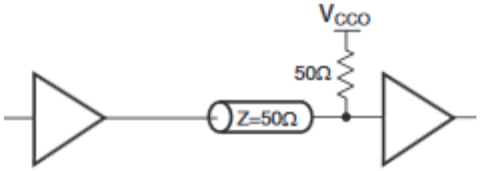
Termination (maintain signal integrity)



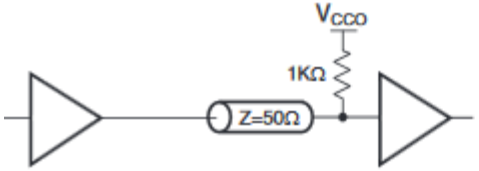
50Ω Far-end Parallel Termination to V_{TT}
FP_VTT_50



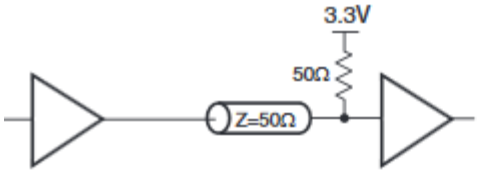
50Ω Far-end Parallel Termination to V_{CCO}
FP_VCCO_50



1KΩ Far-end Parallel Termination to V_{CCO}
FP_VCCO_1000

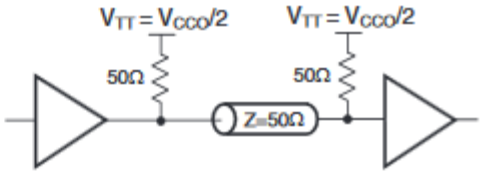


50Ω Far-end Parallel Termination to 3.3V
FP_3.3_50



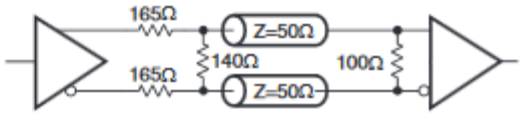
1KΩ Far-end Parallel Termination to 3.3V
FP_3.3_1000

50Ω Near-end Parallel Termination to V_{TT}
50Ω Far-end Parallel Termination to V_{TT}
NP_VTT_50_FP_VTT_50

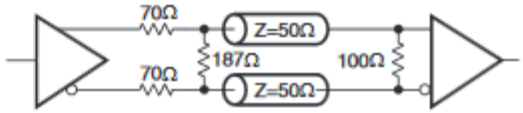


100Ω Far-end Differential Termination
FD_100

165Ω Near Series, 140Ω Near Differential,
100Ω Far Differential
NS_165_ND_140_FD_100



70Ω Near Series, 187Ω Near Differential,
100Ω Far Differential
NS_70_ND_187_FD_100



FPGA – Family example

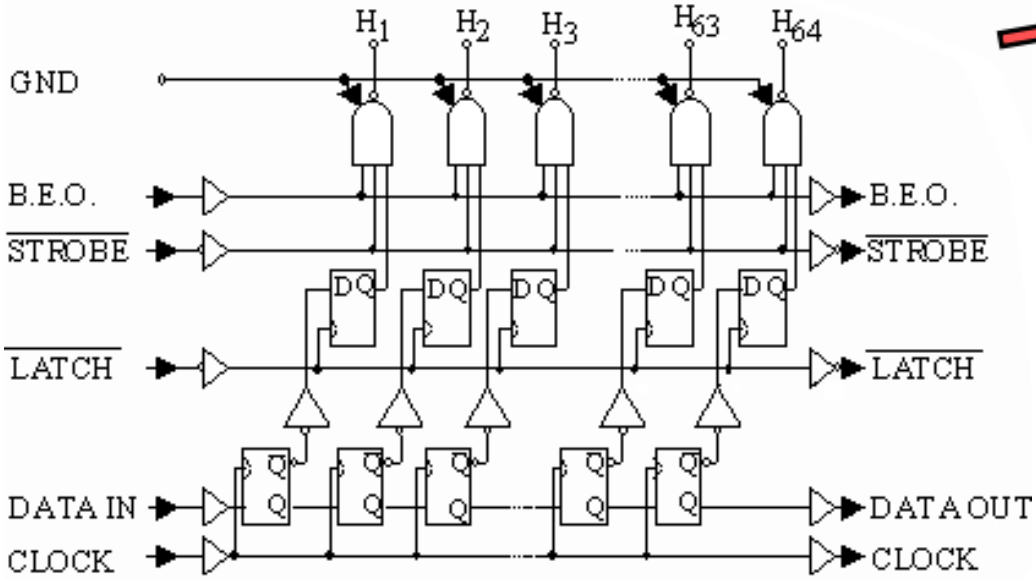
Features	Artix™-7	Kintex™-7	Virtex®-7	Spartan®-6	Virtex-6
Logic Cells	215,000	480,000	2,000,000	150,000	760,000
BlockRAM	13Mb	34Mb	68Mb	4.8Mb	38Mb
DSP Slices	740	1,920	3,600	180	2,016
DSP Performance (symmetric FIR)	930GMACS	2,845GMACS	5,335GMACS	140GMACS	2,419GMACS
Transceiver Count	16	32	96	8	72
Transceiver Speed	6.6Gb/s	12.5Gb/s	28.05Gb/s	3.2Gb/s	11.18Gb/s
Total Transceiver Bandwidth (full duplex)	211Gb/s	800Gb/s	2,784Gb/s	50Gb/s	536Gb/s
Memory Interface (DDR3)	1,066Mb/s	1,866Mb/s	1,866Mb/s	800Mb/s	1,066Mb/s
PCI Express® Interface	x4 Gen2	Gen2x8	Gen3x8	Gen1x1	Gen2x8
Analog Mixed Signal (AMS)/XADC	Yes	Yes	Yes	-	Yes
Configuration AES	Yes	Yes	Yes	Yes	Yes
I/O Pins	500	500	1,200	576	1,200
I/O Voltage	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.5V, 1.8V, 2.5V
EasyPath™ Cost Reduction Solution	-	Yes	Yes	-	Yes

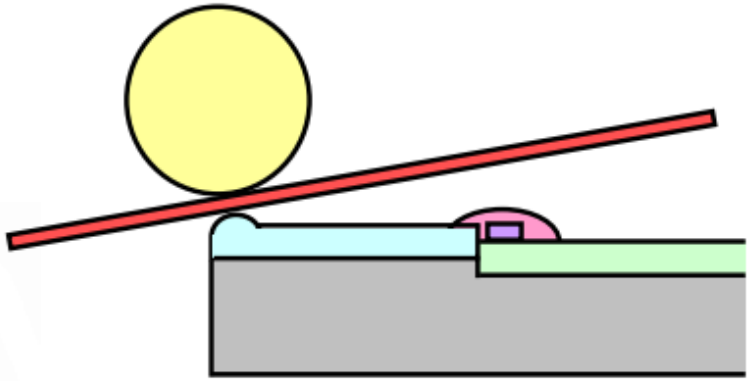
CPLD vs FPGA

	CPLD	FPGA
Configuration	Non volatile configuration	SRAM-reconfiguration: PROM required
Architecture	Combination of SPLD – more combinatorial	Gate array-like – more registers and RAM
Timing	Predictible timing	Application dependant
Speed	Up to ~200MHz	Up to ~150MHz (application dependant)
Density	Low to medium (up to ~10k gates)	Medium to High (up to millions of gates)
Typical application	Combinatorial logic: Fast counters, state machine, interfacing or scheduling	Registered logic: computation

CPLD vs FPGA

Thermal printhead example:





Résolution verticale: 8pts /mm, avance papier 500mm/s
Résolution horizontale: 8pts/mm, largeur impression 100mm
Temps de chauffe/ligne: 100µs

- Data feeding between each line
- Synchronize heating with paper movement

FPGA or CPLD?

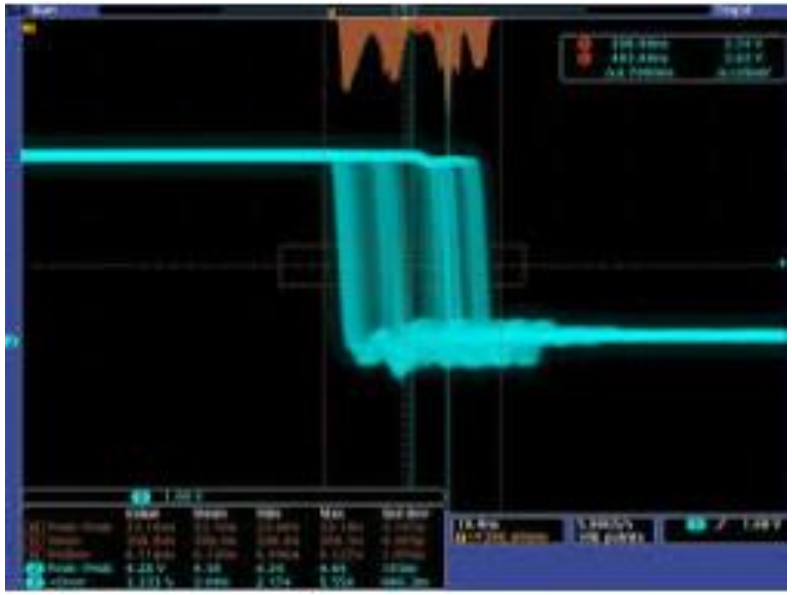
FPGA architecture

1. Global presentation
2. Logic block
3. Interconnexion
4. I/O
5. Clock and timing management
6. Other embedded ressources
7. Miscellanous

FPGA - Clock

- ✓ FPGA is a synchronous device: use of a **global clock** with **dedicated routing** capabilities and **buffers**.

Why is it important to manage Clock distribution into a FPGA?



FPGA - Clock

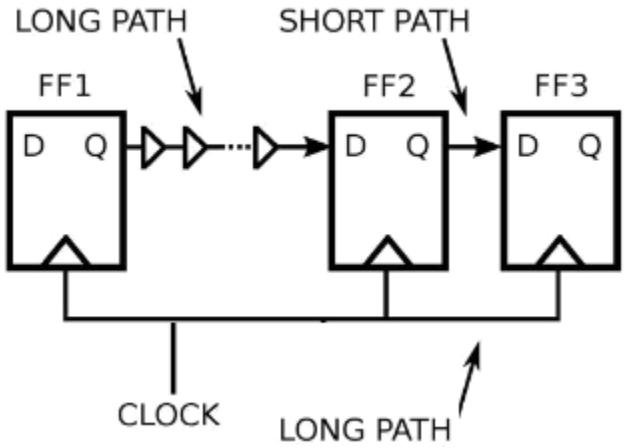
✓ Clock skew :

- the same sourced clock signal arrives at different components at different times
- Could be caused by wire-interconnect length, capacitive coupling,…

✓ Skew and jitter could provide 2 types of faults:

- **Setup violation:**
  Destination flip-flop FF2 receives the clock tick before the data coming from FF1
  => the data doesn't pass through FF2

- **Hold violation:**
  clock travels slower than the data path from FF2 to FF3
  => the data penetrates two registers in the same clock tick



FPGA - Clock

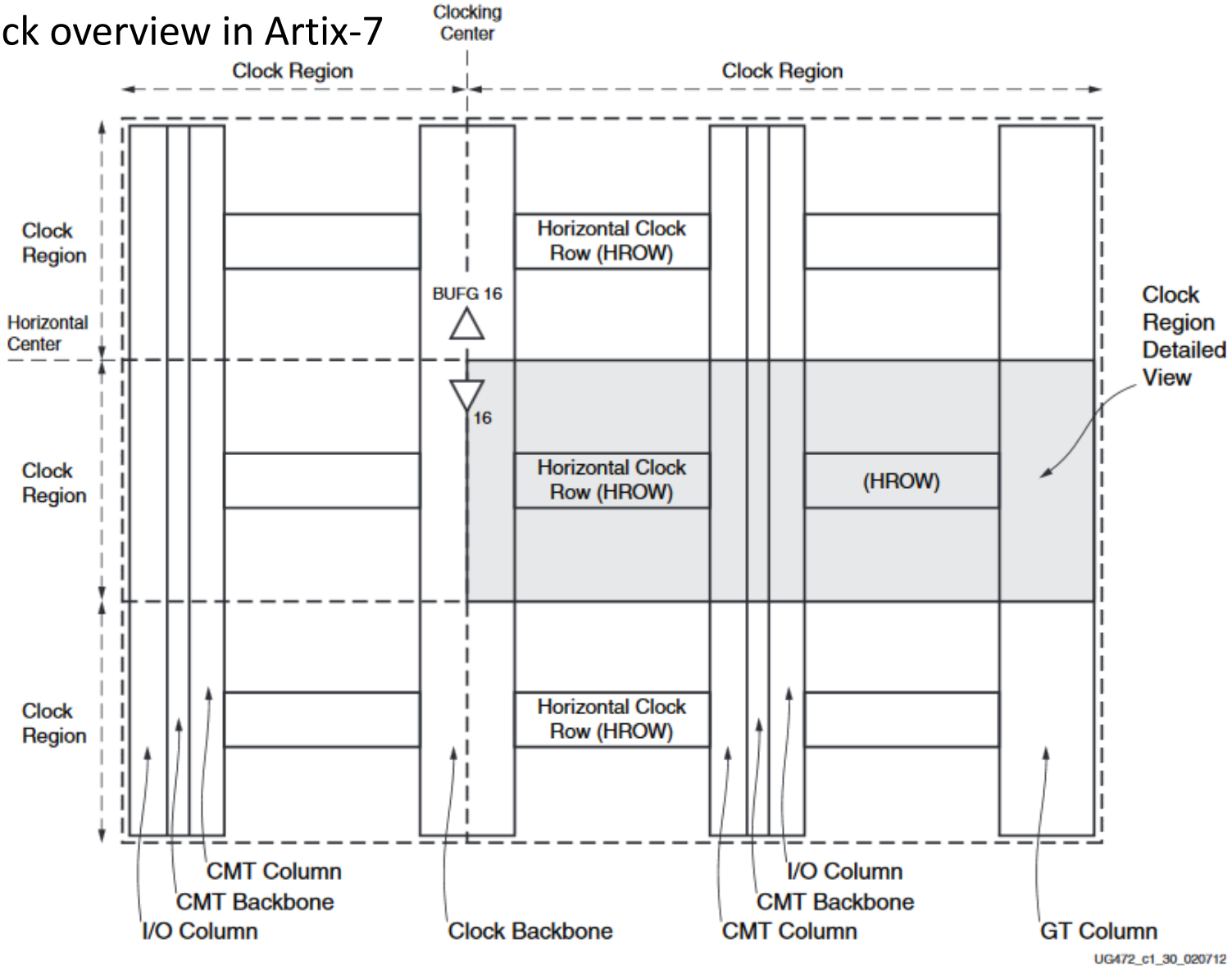
✓ Use Clock dedicated resources to optimize clock distribution:

- **Global clock lines**
  - Distribute the clock into the device with optimized performance (low skew, propagation delay,…)
  - Often separated between global clock lines and regional clock trees

- **Global clock buffers**
  - Drives the global clock lines
  - Can be used as a clock enable circuit to enable or disable clocks that span multiple clock regions
  - Can be used as a glitch-free multiplexer to:
    - select between two clock sources
    - switch away from a failed clock source

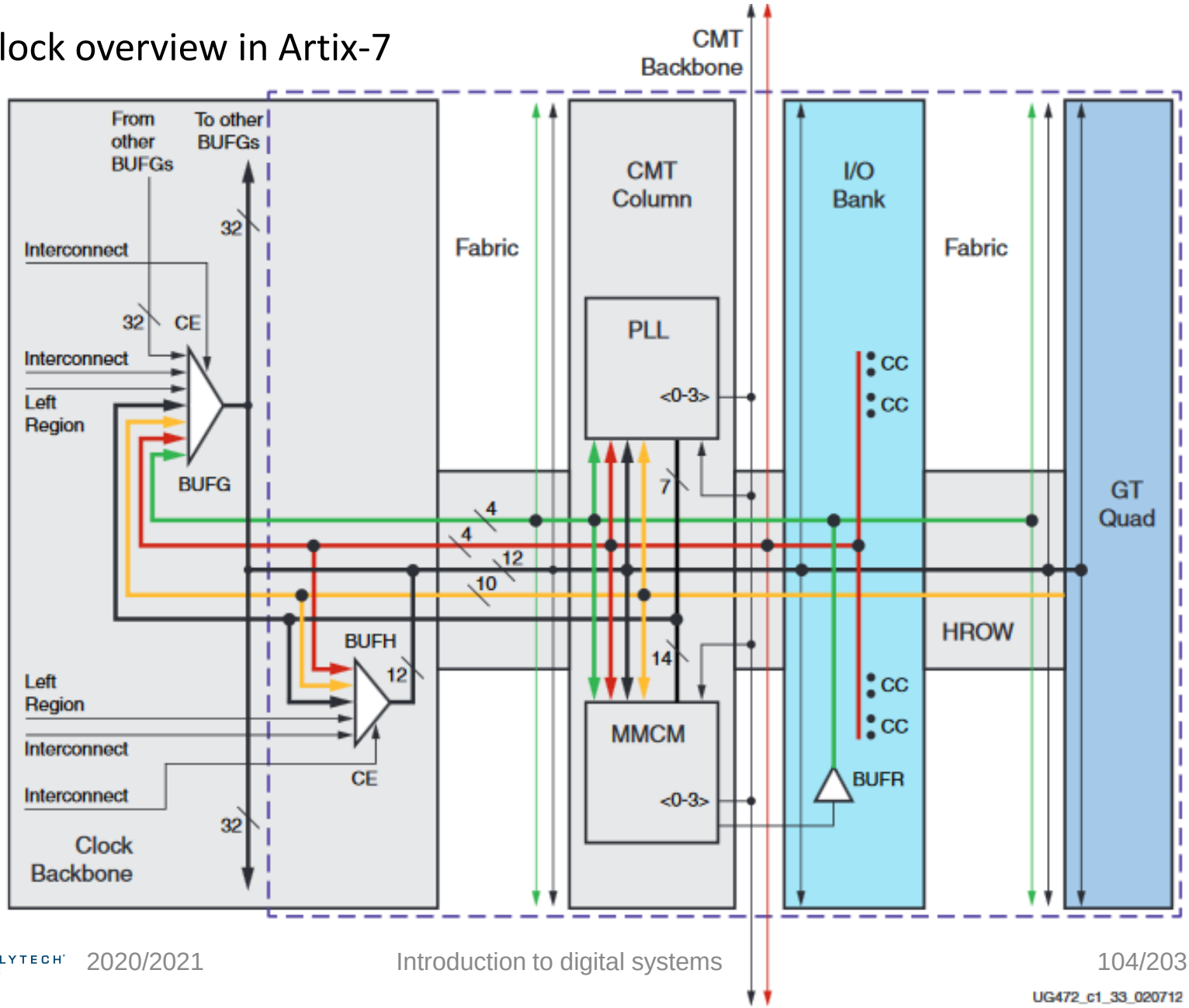
FPGA - Clock

- ✓ Clock overview in Artix-7



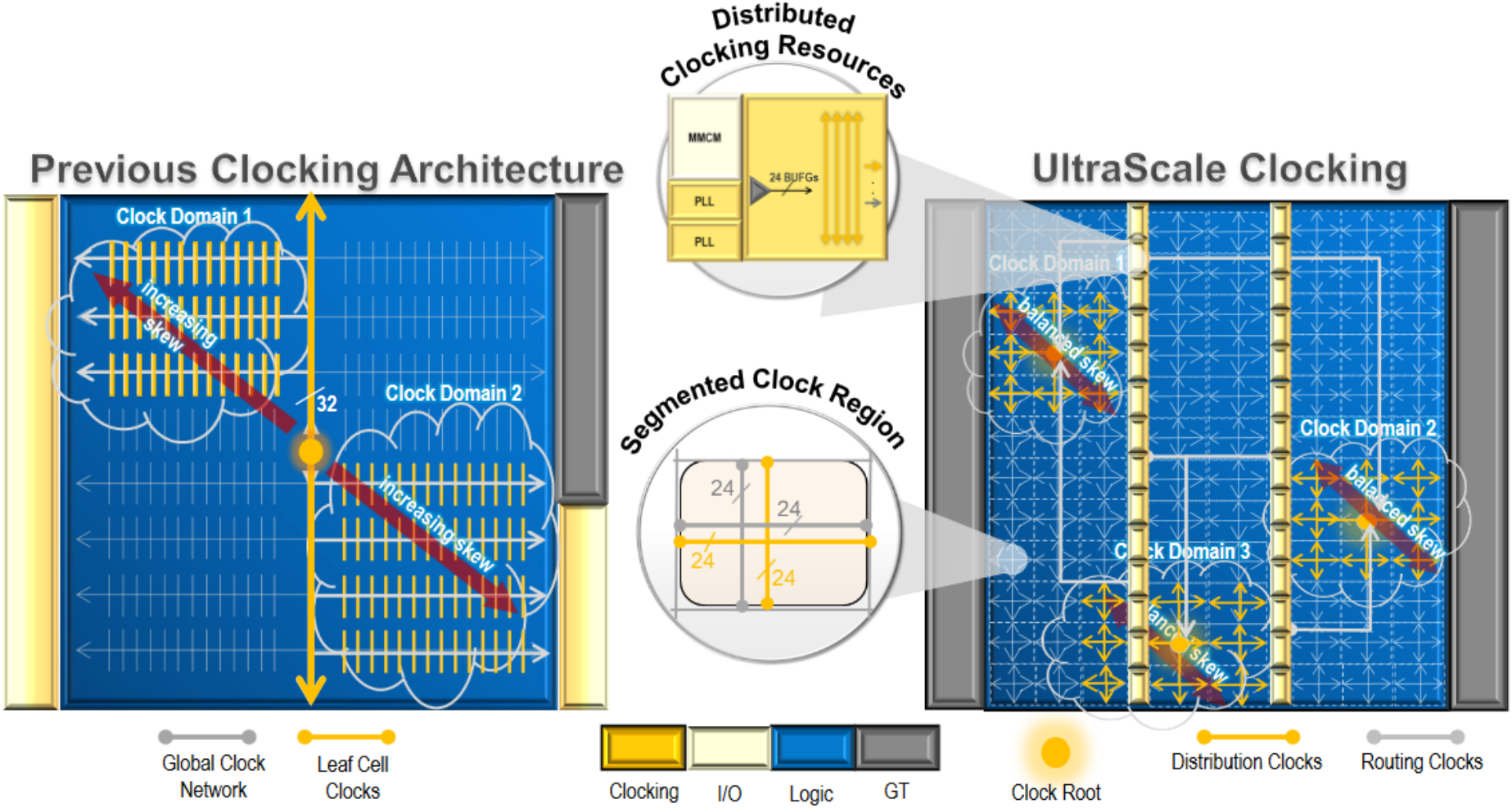
FPGA - Clock

- ✓ Clock overview in Artix-7



FPGA - Clock

- ✓ Ultrascale clocking



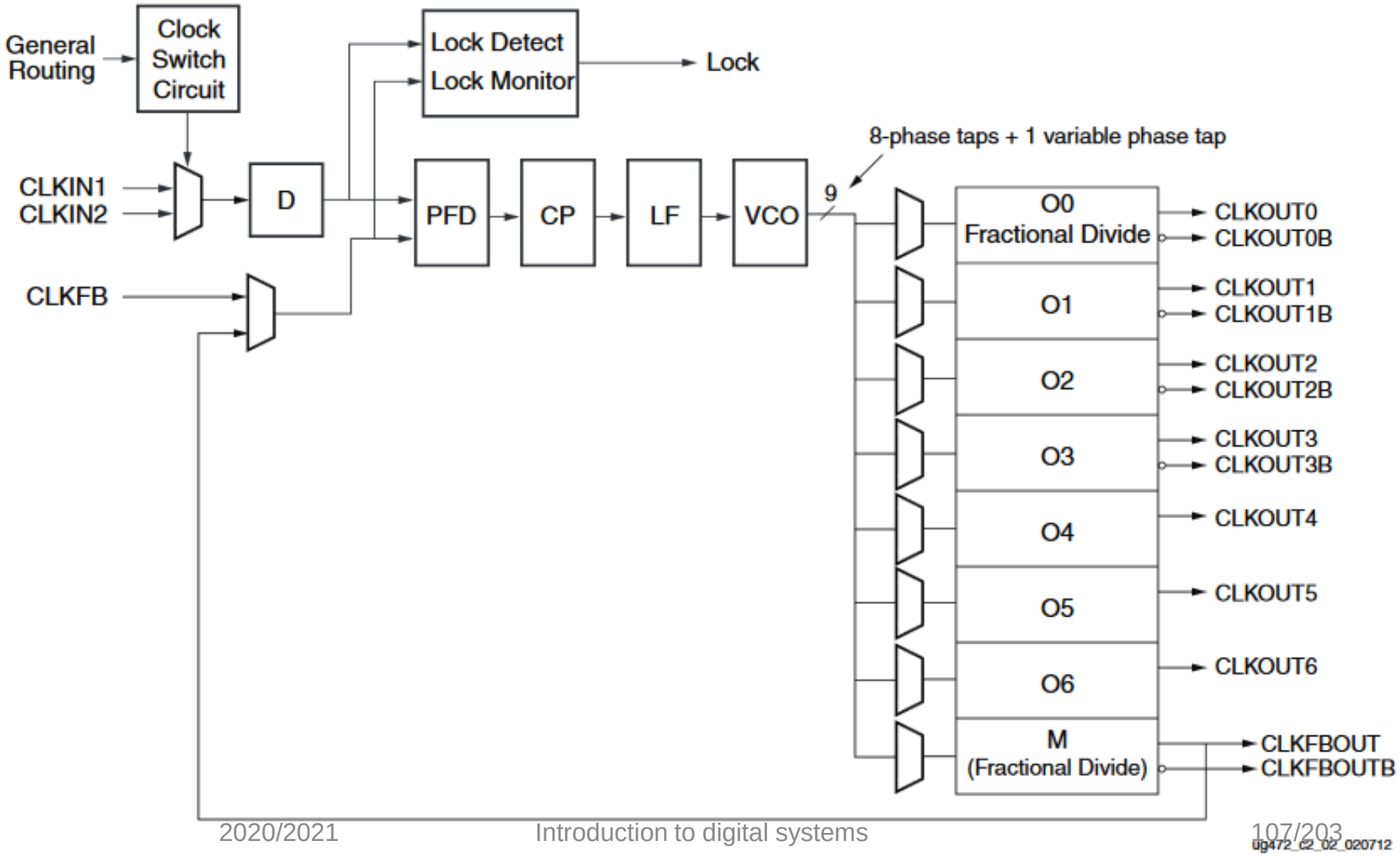
FPGA - Clock

- **Clock Management Tiles (CMT)**
  - Serve as frequency synthesizers for a wide range of frequencies
  - Serve as a jitter filters for either external or internal clocks
  - Deskew clocks

  Includes Phase locked loop

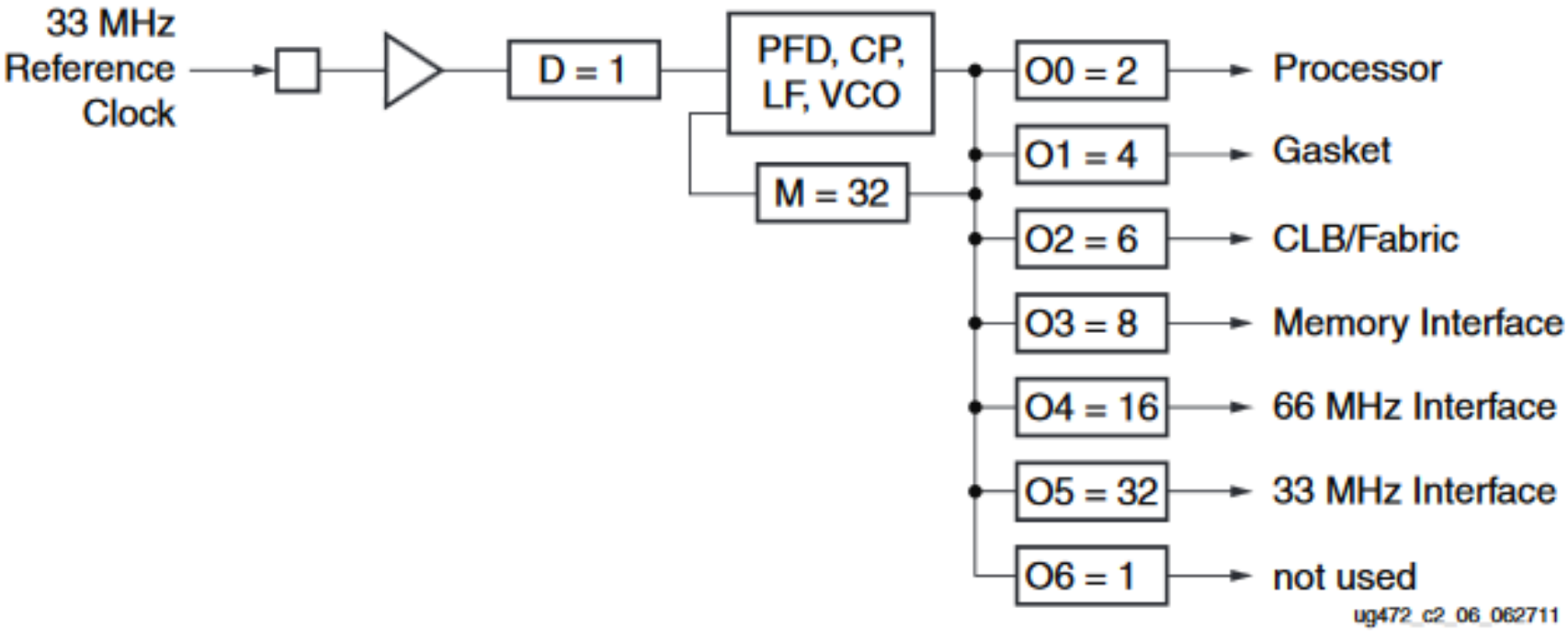
FPGA - Clock

- ✓ Clock Management Tiles :



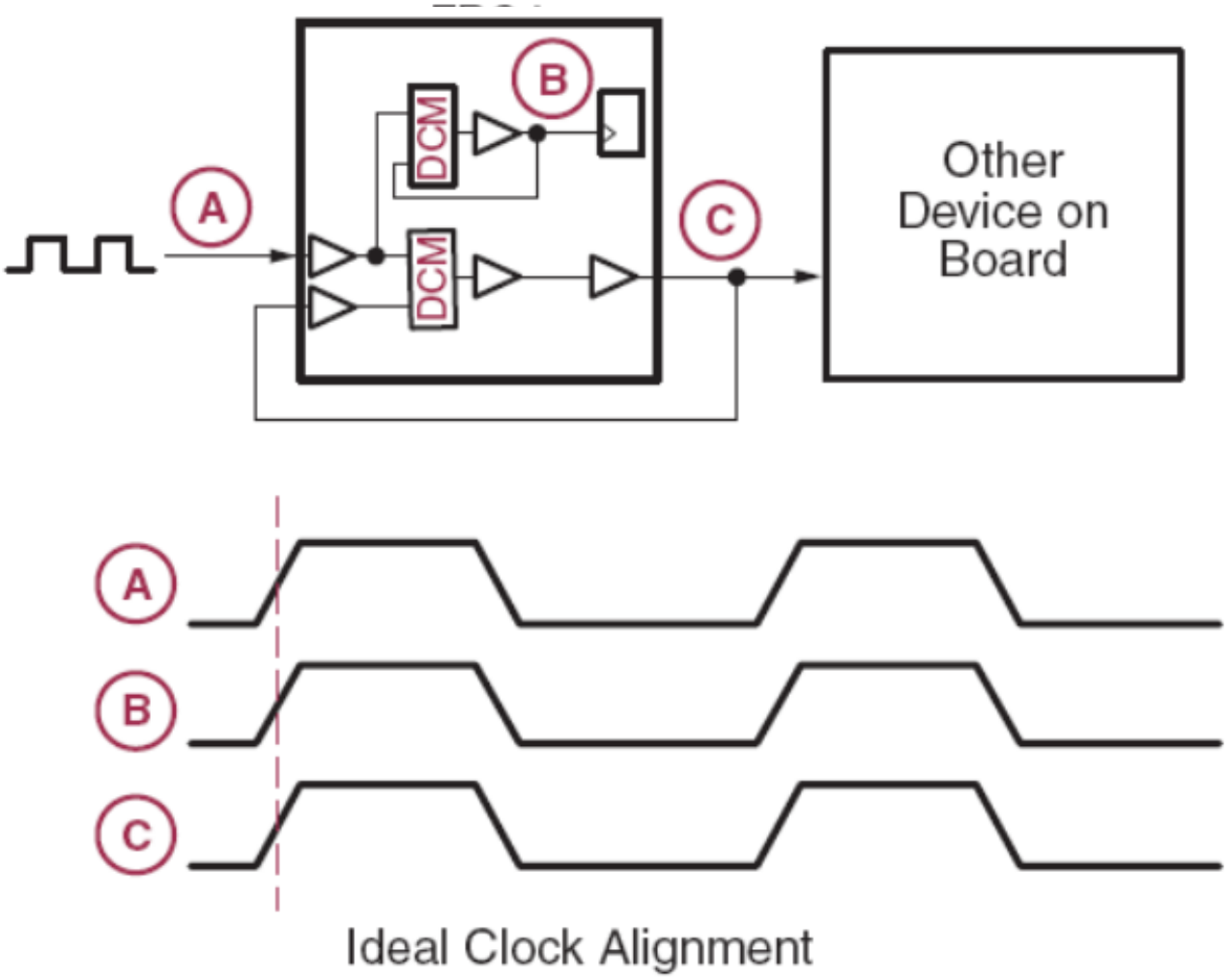
FPGA - Clock

- ✓ Clock Management Tiles example:



FPGA - Clock

- ✓ Clock alignment:



FPGA – Propagation delay

Timing constraint management

- Combinatorial logic introduces Propagation delay

⇒ Numbers of CLBs between 2 registers shall be compliant with the clock frequency

Example for 50 MHz clock frequency in XC4000XL-3:

Clock period 20 ns

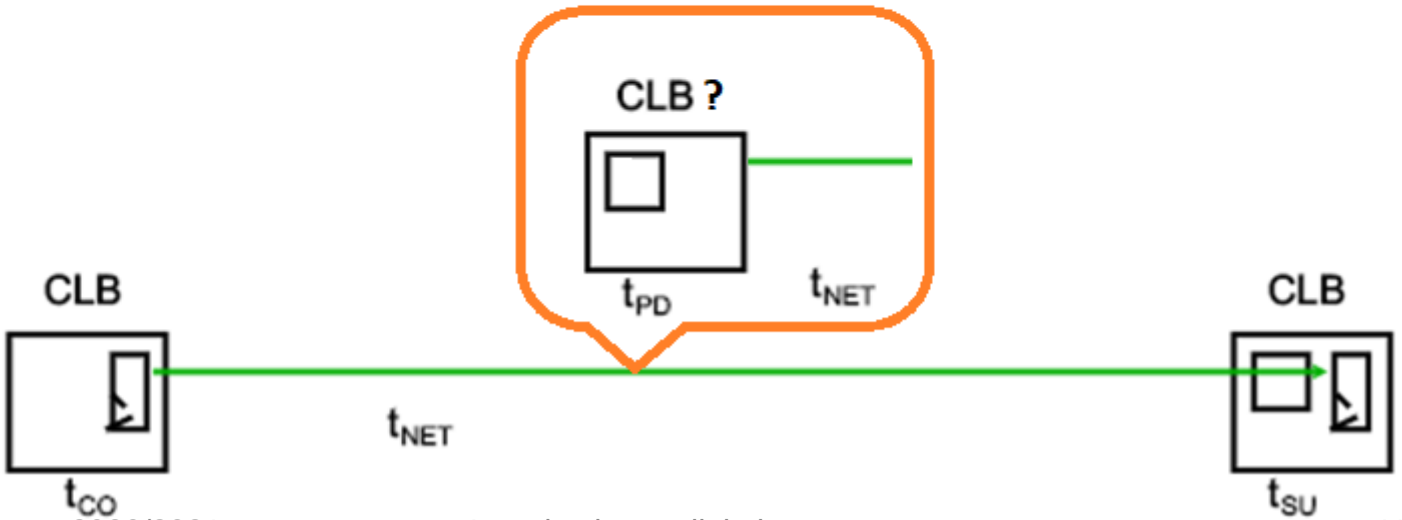
t_{NET} = 3 ns

t_{SU} = 2 ns

t_{CO} = 3 ns

t_{PD} = 3 ns

What is the maximum number of CLB that can be added between these 2 registers?



FPGA – Propagation delay

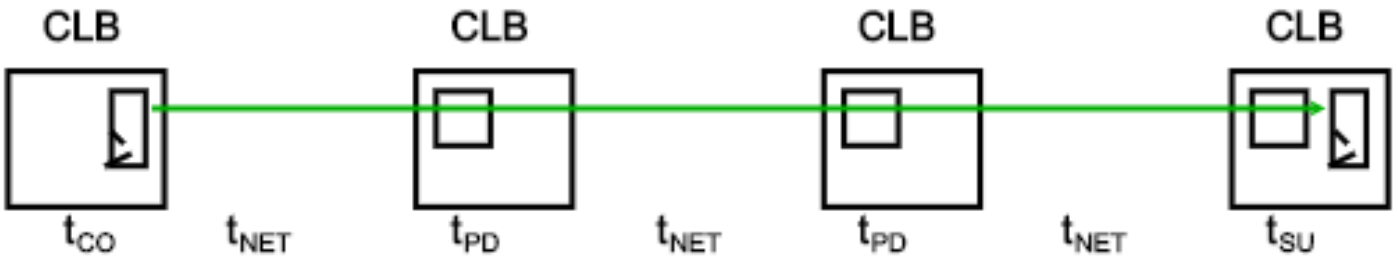
Timing constraint management

- Combinatorial logic introduces Propagation delay

⇒ Numbers of CLBs between 2 registers shall be compliant with the clock frequency

Example for 50 MHz clock frequency in XC4000XL-3:

Clock period	20 ns	
One level		- 8 ns ($t_{CO} + t_{NET} + t_{SU}$)
Delay allowance		12 ns
Each added level		% 6 ns ($t_{PD} + t_{NET}$)
Added levels of logic allowed	2 CLBs	

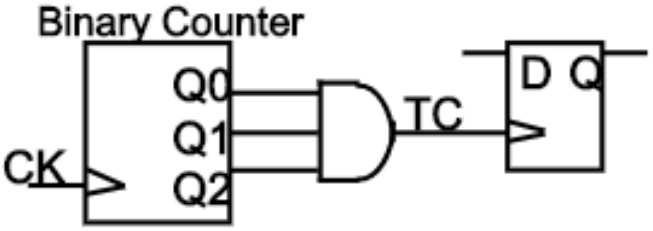


FPGA – Propagation delay

Avoid Gated-Clock or Asynch. Reset

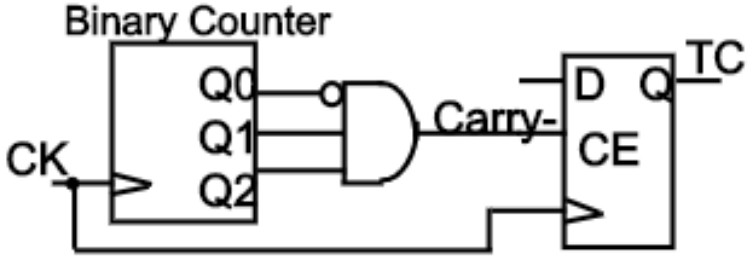
- Move gating from clock pin to prevent glitch from affecting logic.

Poor Design:



TC and Q may glitch during the transition of Q<0:2> from 011 to 100

Improved Designs:



TC will not glitch during the transition of Q<0:2> from 011 to 100

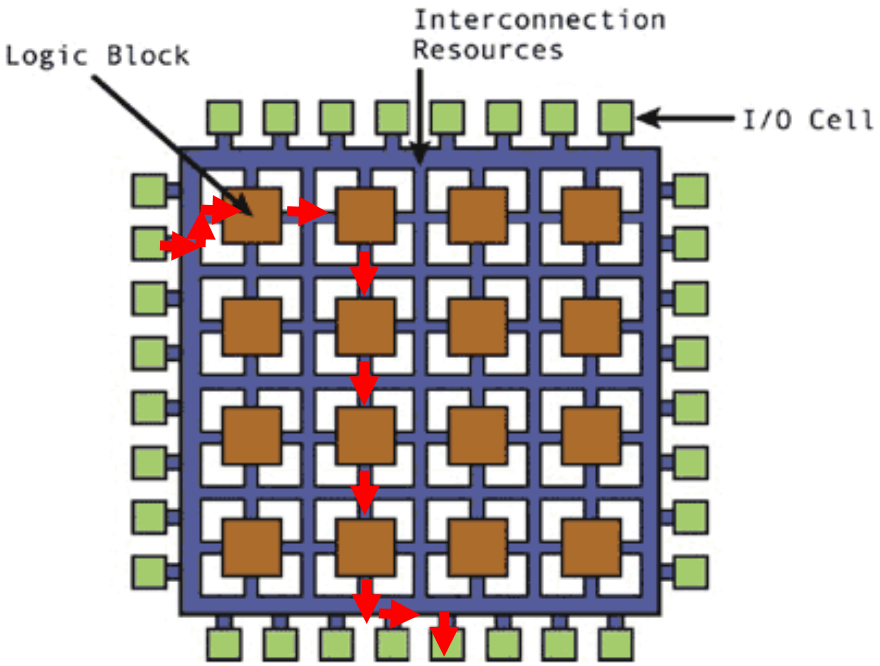
FPGA – Performances

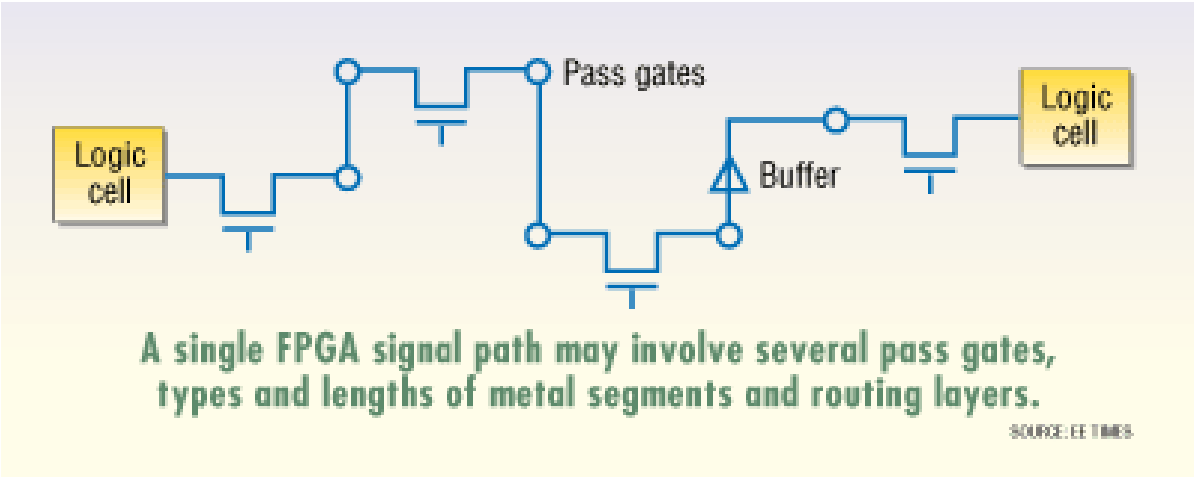
✓ Minimal latency = highest clock speed

BUT:

- CLB delay
- Interconnect matrix
- IOs

=> Hold or Setup violation





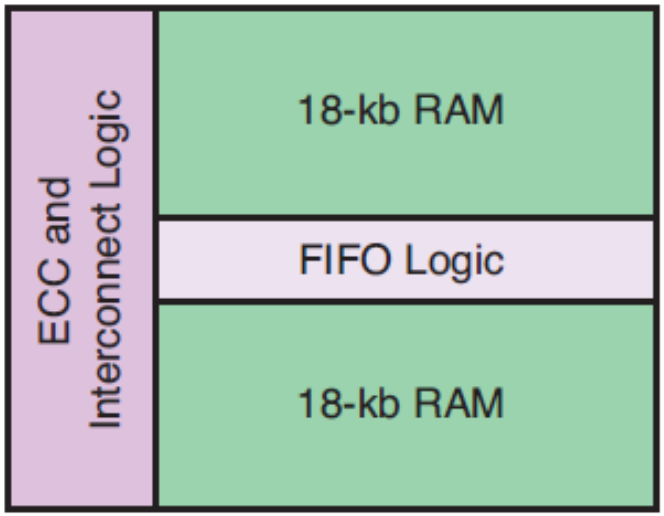
FPGA architecture

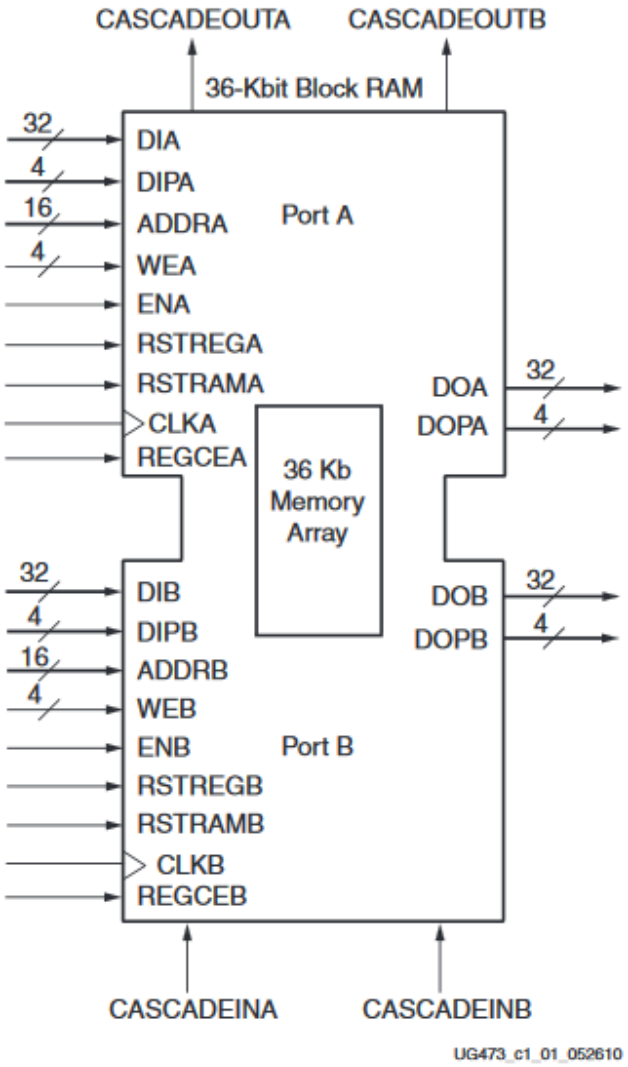
1. Global presentation
2. Logic block
3. Interconnexion
4. I/O
5. Clock and timing management
6. Other embedded ressources
7. Miscellanous

FPGA – Embedded memory

Memory circuits.

- Up to ~68Mbits, dispatched into small block of some kbits



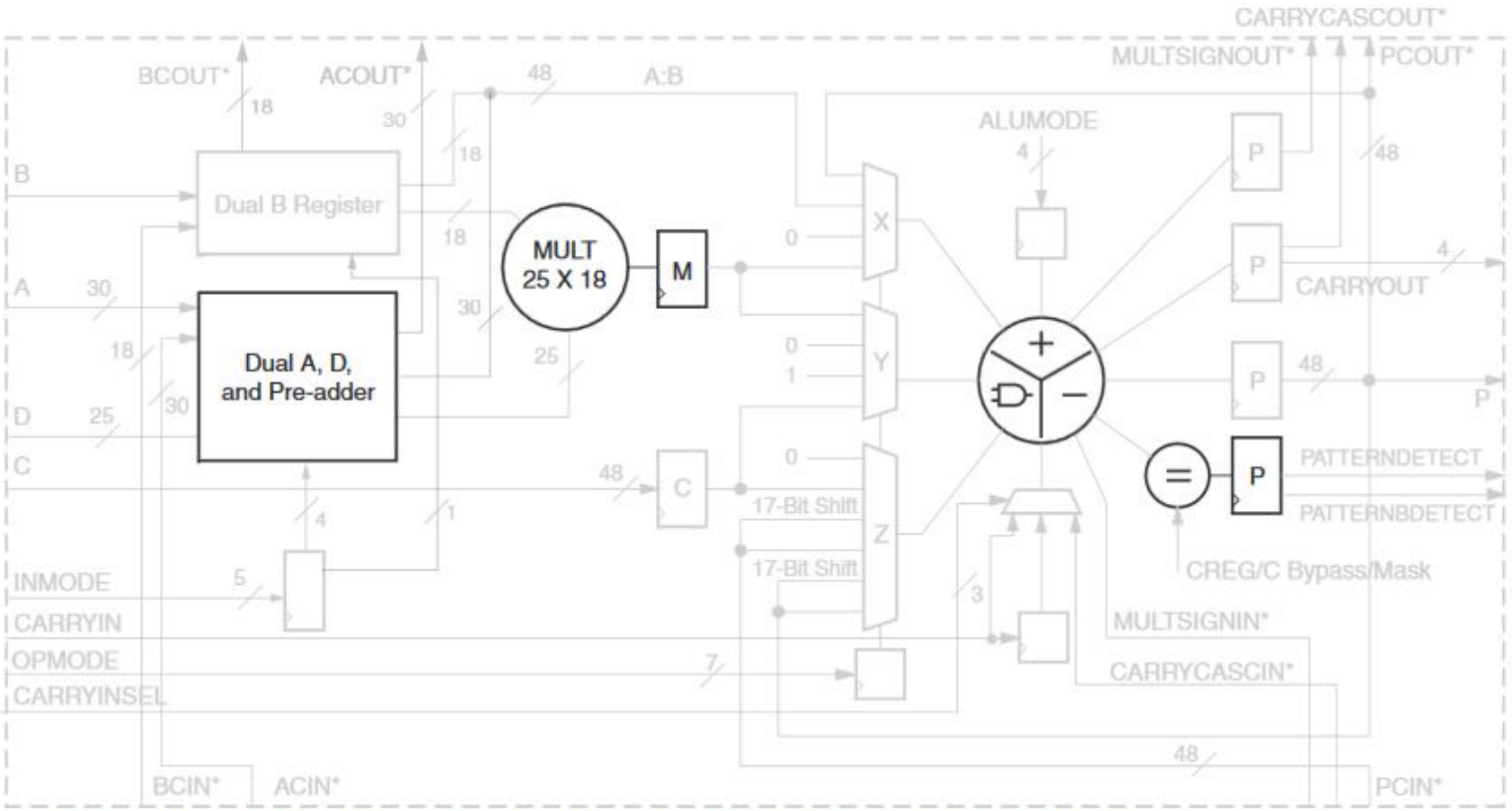


FPGA – DSP circuits

DSP circuits.

Up to ~4k multiplier blocks

In Artix 7 series: 25 x 18 multiplier, adder/subtracter

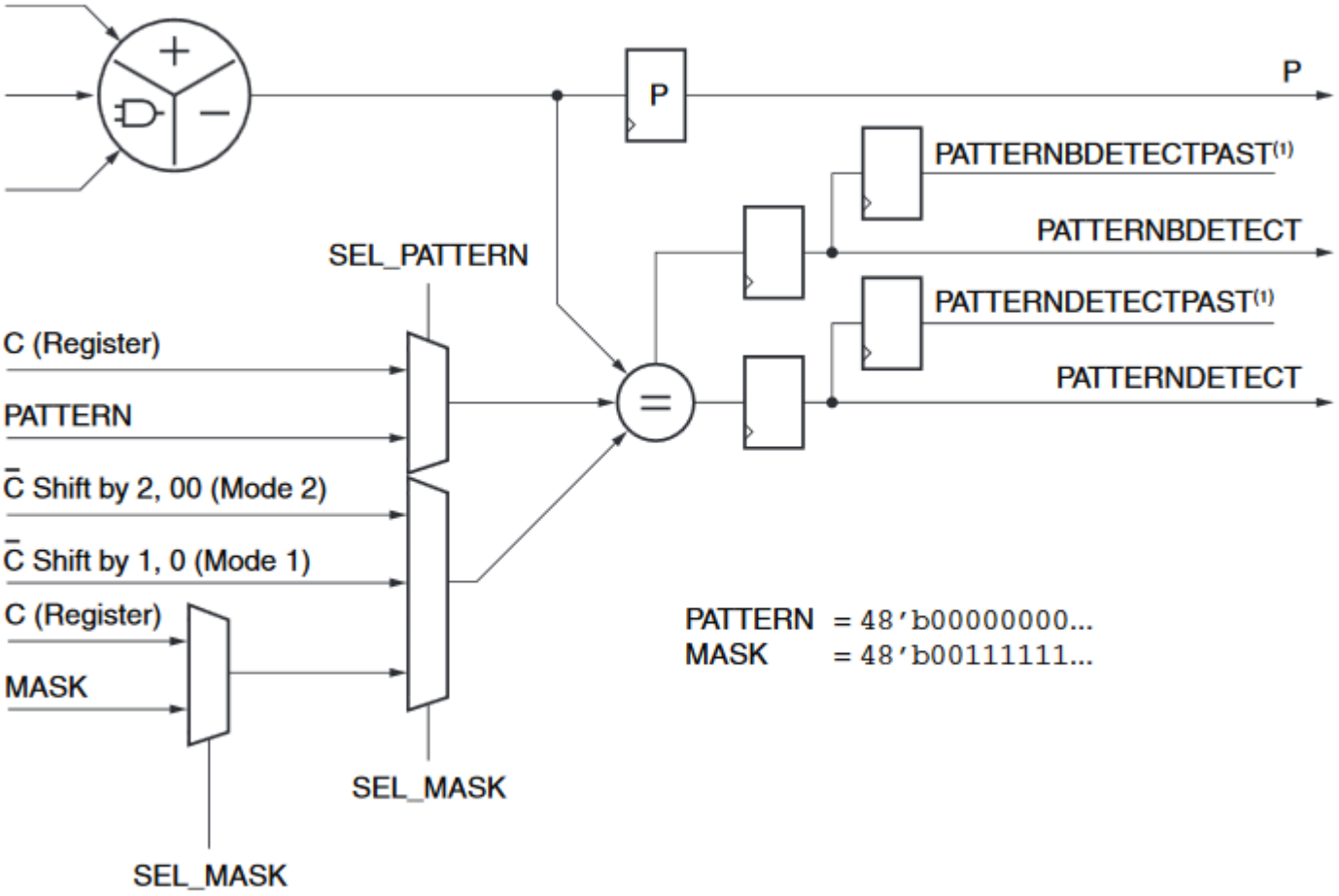


*These signals are dedicated routing paths internal to the DSP48E1 column. They are not accessible via fabric routing resources.

UG369_c1_14_082109

FPGA – Pattern recognition

Pattern recognition circuits.



Notes:
1. Denotes an internal signal.

FPGA – IP

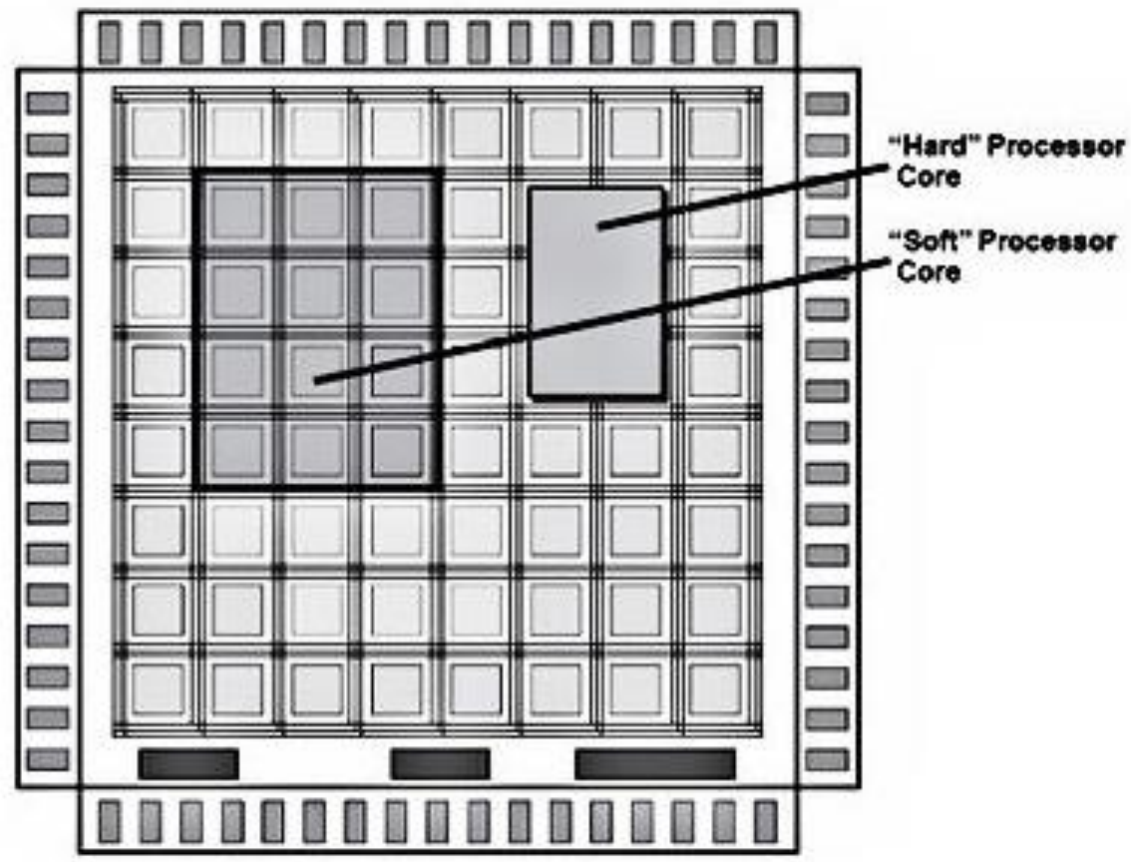
- ✓ An IP (intellectual property) core is a block of logic or data that is used in making a FPGA product.

- ✓ IP cores fall into one of two main categories: hard cores or soft cores:

  Hard cores are physical implementation (placed and routed).

  Soft cores exist either as a netlist or HDL code.

FPGA – IP



FPGA – soft IP

- Arithmetic core
- Memory core
- Prototype board
- Other
- Communication controller
- Processor
- Coprocessor
- System on Chip
- Crypto core
- System on Module
- DSP core
- System controller
- ECC core
- Testing / Verification
- Library
- Video controller

http://opencores.org/

FPGA – Hard IP

External memory management (DDR, DDR2, DDR3, …)

Hard processor

Analog blocks (ADC, DAC, …)

CRC (Cyclic Redundancy Check)

Crypto cores (AES, Advanced Encryption Standard)

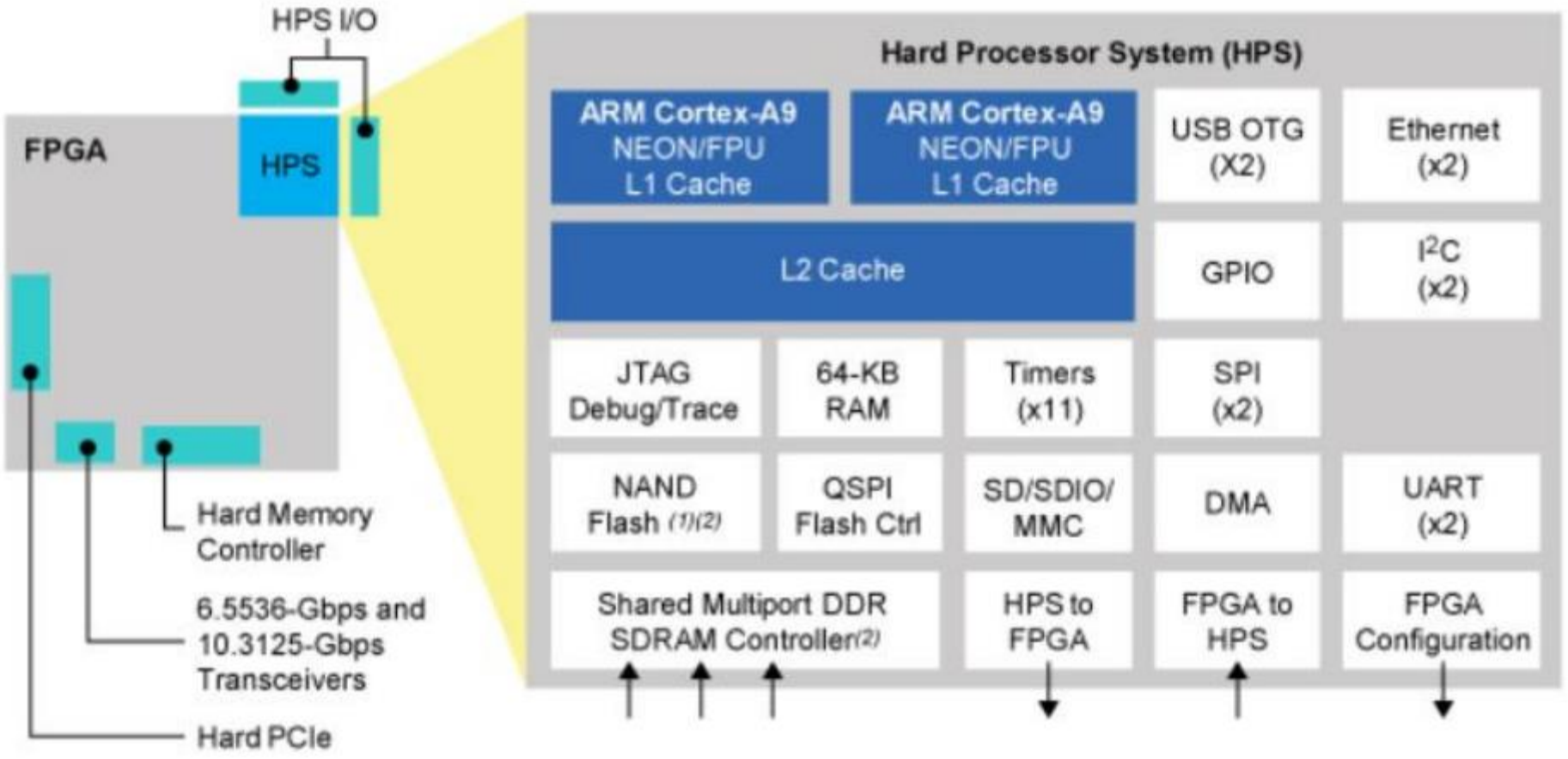
Secured storage of keys

…

$\Rightarrow$ Not configurable : Dedicated I/O

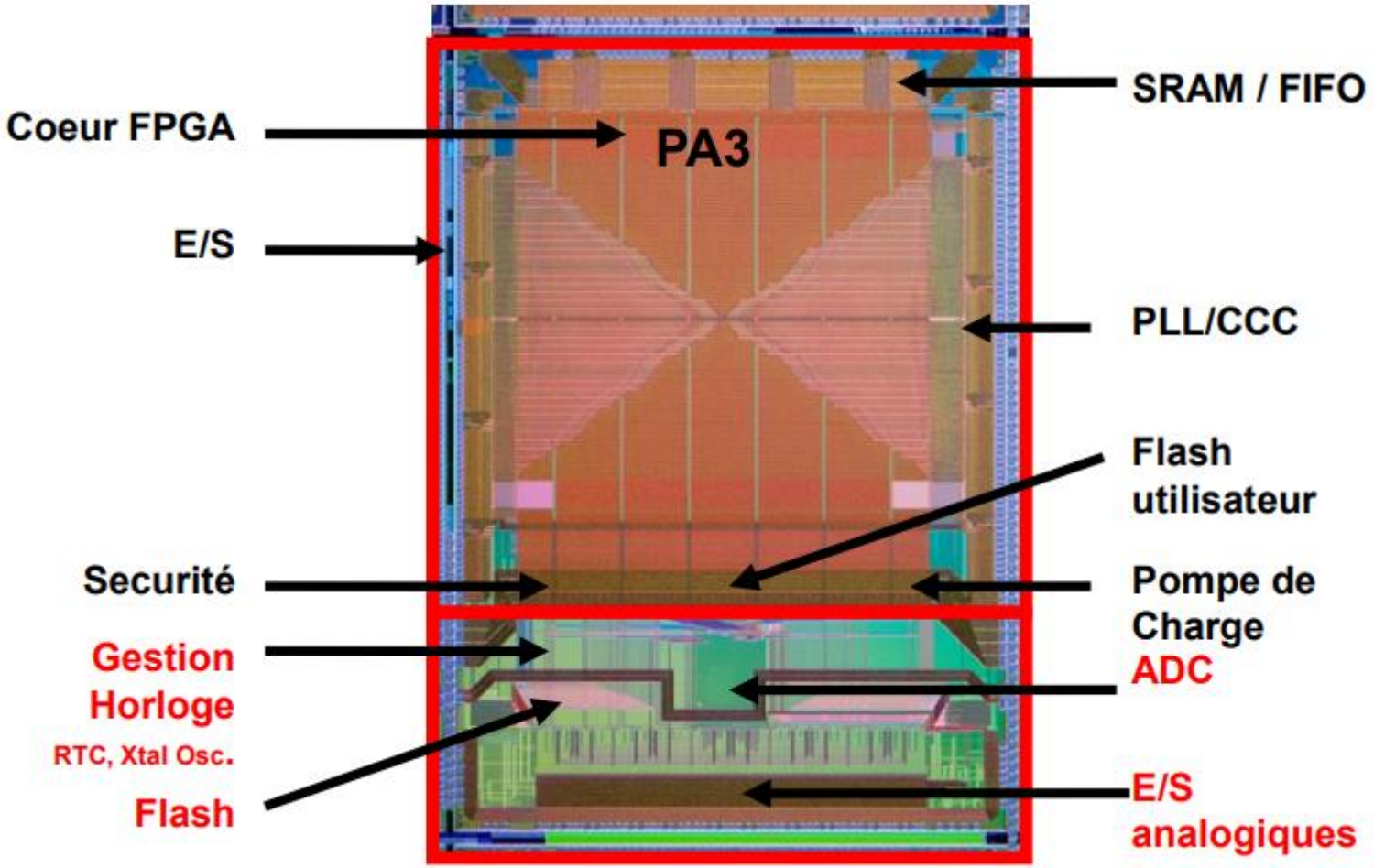
FPGA – Hard IP

Example: Intel ARRIA V with dual-ARM Cortex A9, Memory Controller, and PCIe hard IP



FPGA – Hard IP

Example: Actel Fusion with analogic blocks



FPGA – IP

SOFT CORES

Advantages and drawbacks ?

Advantages

- ✓ Much higher level of portability
- ✓ Generally most affordable
- ✓ More low-cost/free sources due to easier implementation
- ✓ Relatively easy to target to specific architectures
- ✓ Relatively easy to modify
- ✓ Possibility of portability between architectures

Drawbacks

- χ Much lower level of optimization: lower performance levels, higher resource utilization
- χ May require more design effort
- χ Differences in tool sets used for design implementation can affect results significantly and are not always predictable

FPGA – IP

HARD CORES

Advantages and drawbacks ?

Advantages

- ✓ Well documented, highly-optimized, high-performance, reliable fixed implementation
- ✓ Similar to purchase of a standard IC component
- ✓ High level of confidence in functionality, measured and characterized functionality
- ✓ Performance
- ✓ Power consumption

Drawbacks

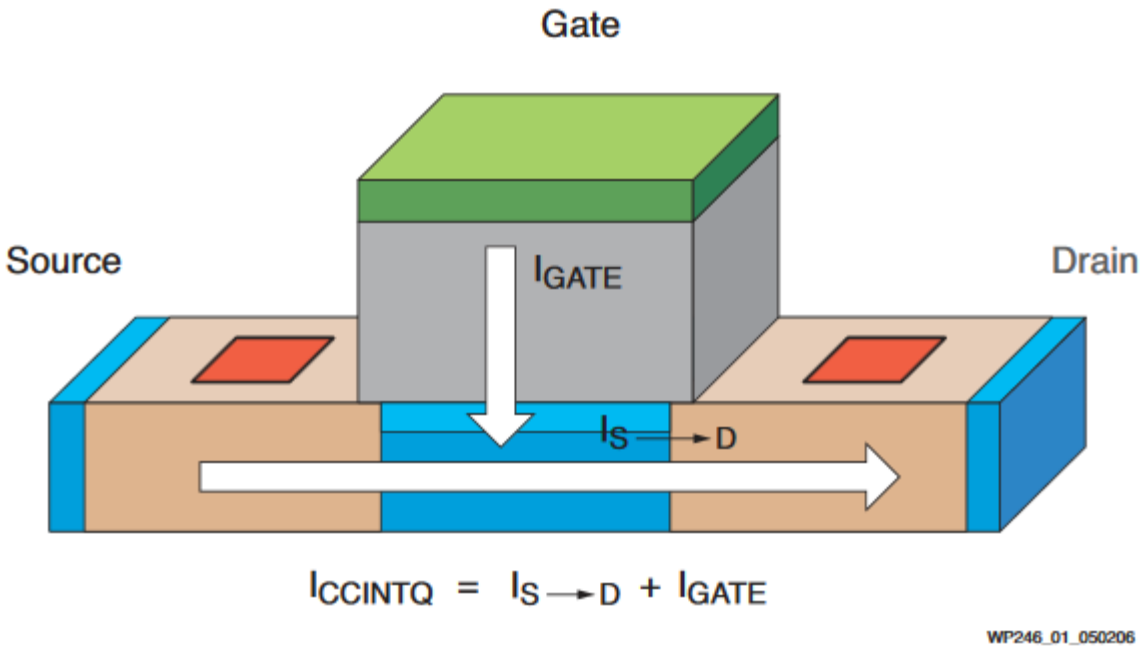
- χ Highly-optimized and fine tuned: difficult to port to other targets with equivalent performance
- χ Strong incentive for provider to try to strongly tie to their architecture and make it less attractive to port to alternative architectures
- χ Fixed implementation, unable to modify core implementation or add additional instances if required without switching devices within the family

FPGA architecture

1. Global presentation
2. Logic block
3. Interconnexion
4. I/O
5. Clock and timing management
6. Other embedded ressources
7. Miscellanous

FPGA – Consumption

- ✓ Static consumption:
  - due to leakage (very low)
  - Increase with decreasing the gate length



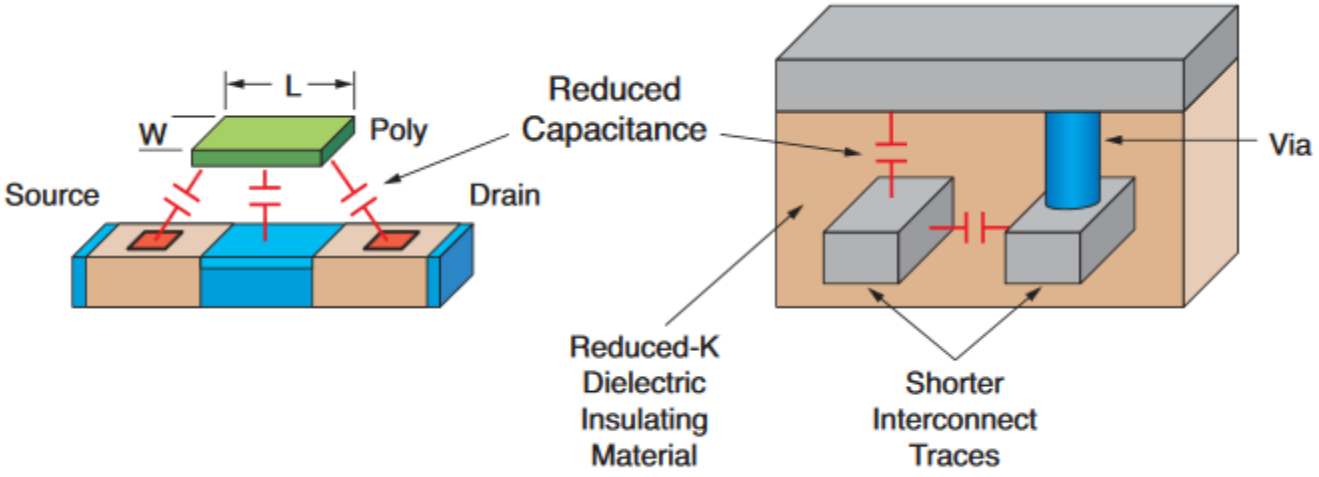
FPGA – Consumption

- ✓ Dynamic consumption:
  - due to switching $P_{Dynamic} = \frac{1}{2}\sum_i C_i f_i V^2$

Where: C_i: capacitance

f_i: switching frequency

V: Bias voltage

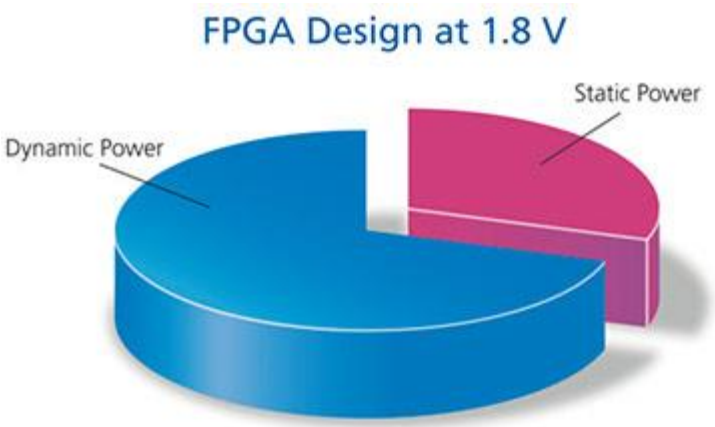


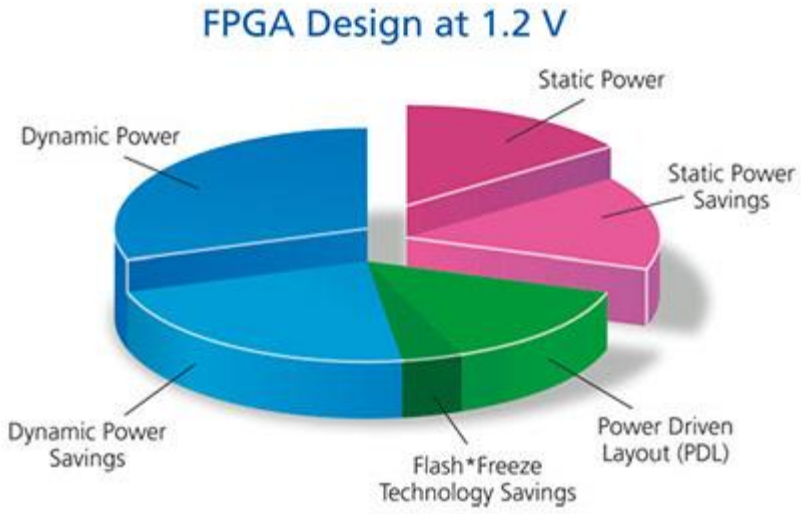
FPGA – Consumption

- ✓ Power optimization at equal performance:

  Use lower voltage

$$P_{Dynamic} = \frac{1}{2}\sum_{i} C_i f_i V^2$$





FPGA – Consumption

- ✓ Dependency to temperature and voltage

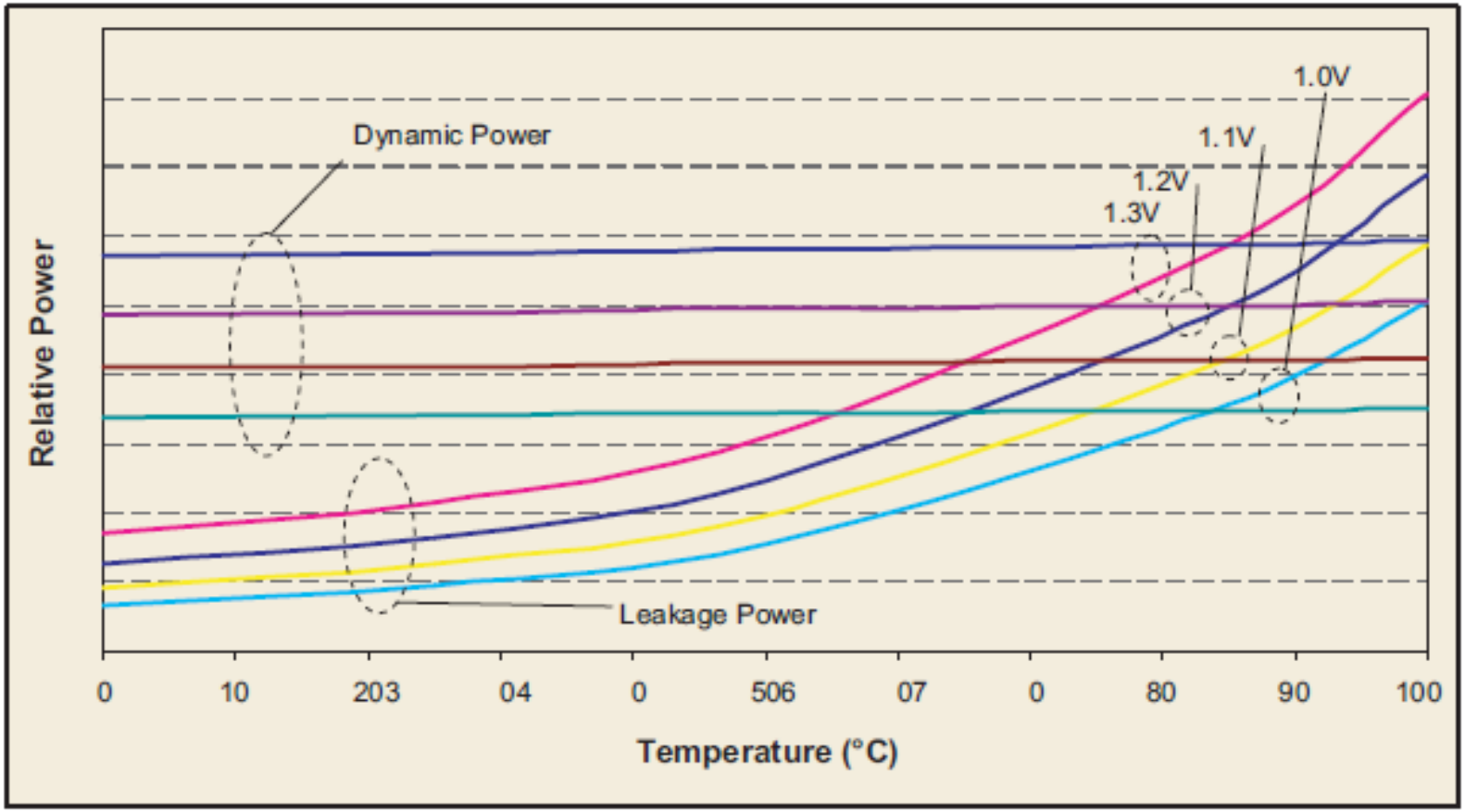
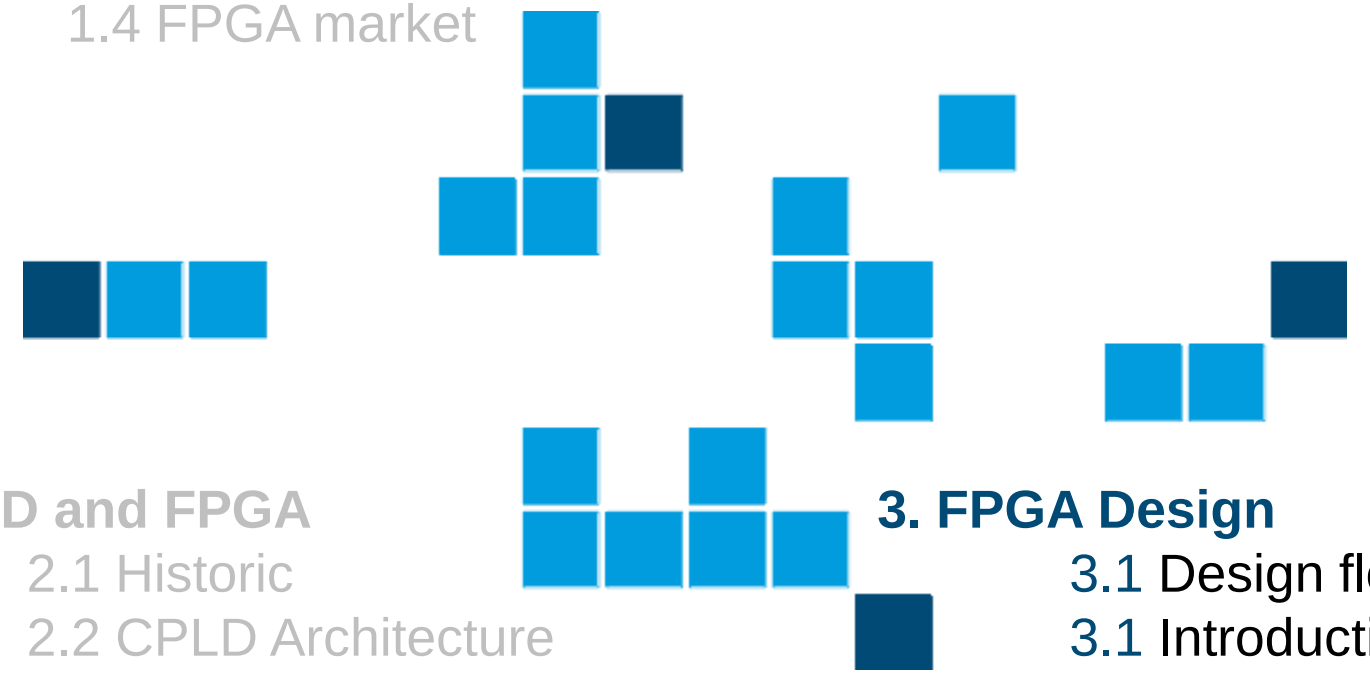


Figure 1 – Power dependency on voltage and temperature

Outline

1. Introduction
    - 1.1 Different types of digital components
    - 1.2 Advantages and drawbacks of each type
    - 1.3 Selected applications
    - 1.4 FPGA market



2. CPLD and FPGA
    - 2.1 Historic
    - 2.2 CPLD Architecture
    - 2.3 FPGA Architecture
    - 2.3 Clock and timing management
    - 2.4 Other embedded ressources

3. FPGA Design
    - 3.1 Design flow
    - 3.1 Introduction to VHDL
    - 3.2 Simulations
    - 3.3 Finite State Machines

Avant de commencer….

- ✓ Essayer de démarrer Vivado sur tous les PC

- ✓ Envoyer le projet Exo1

FPGA - design

Circuit design using FPGA: need EDA (Electronic Design Automation) software

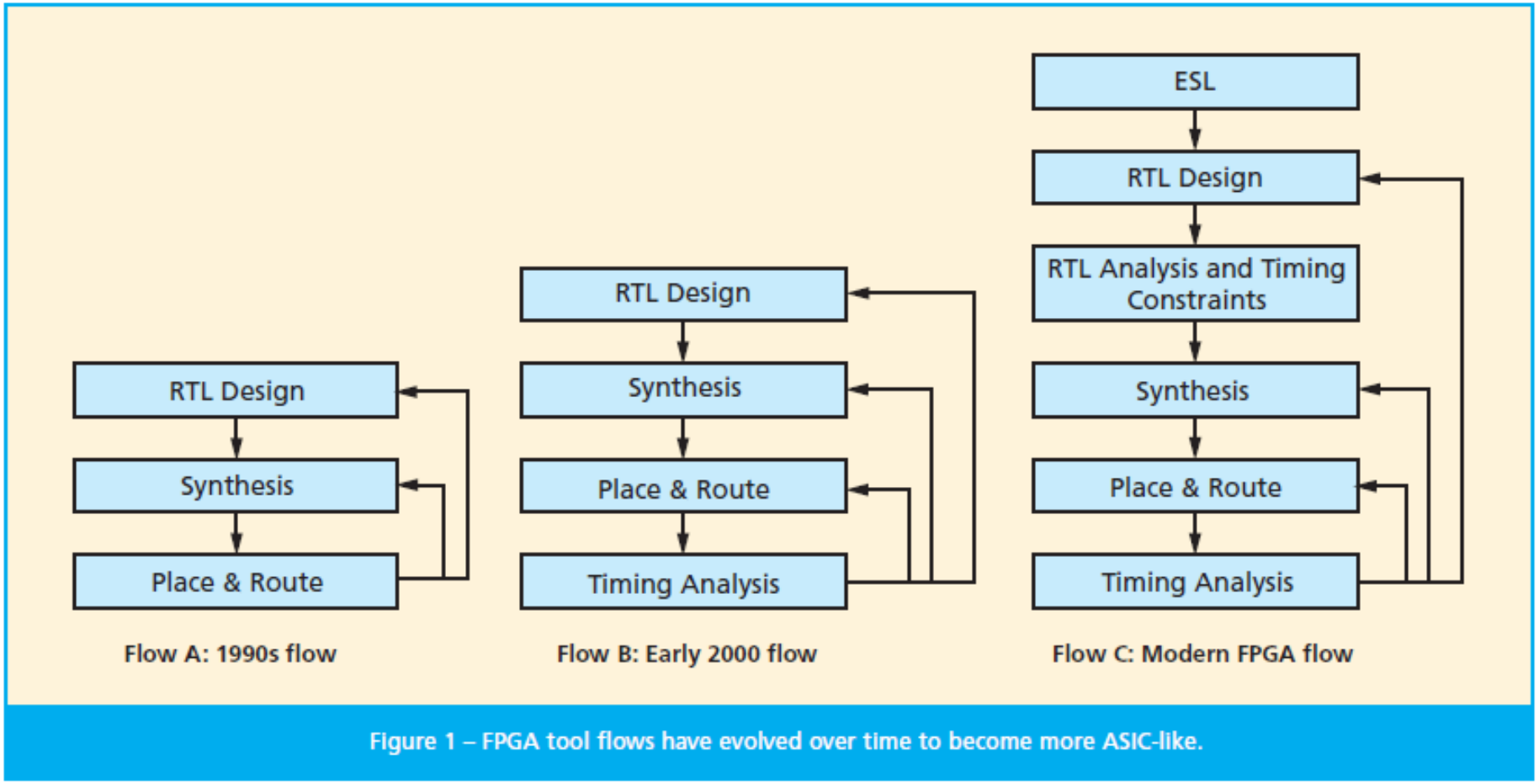
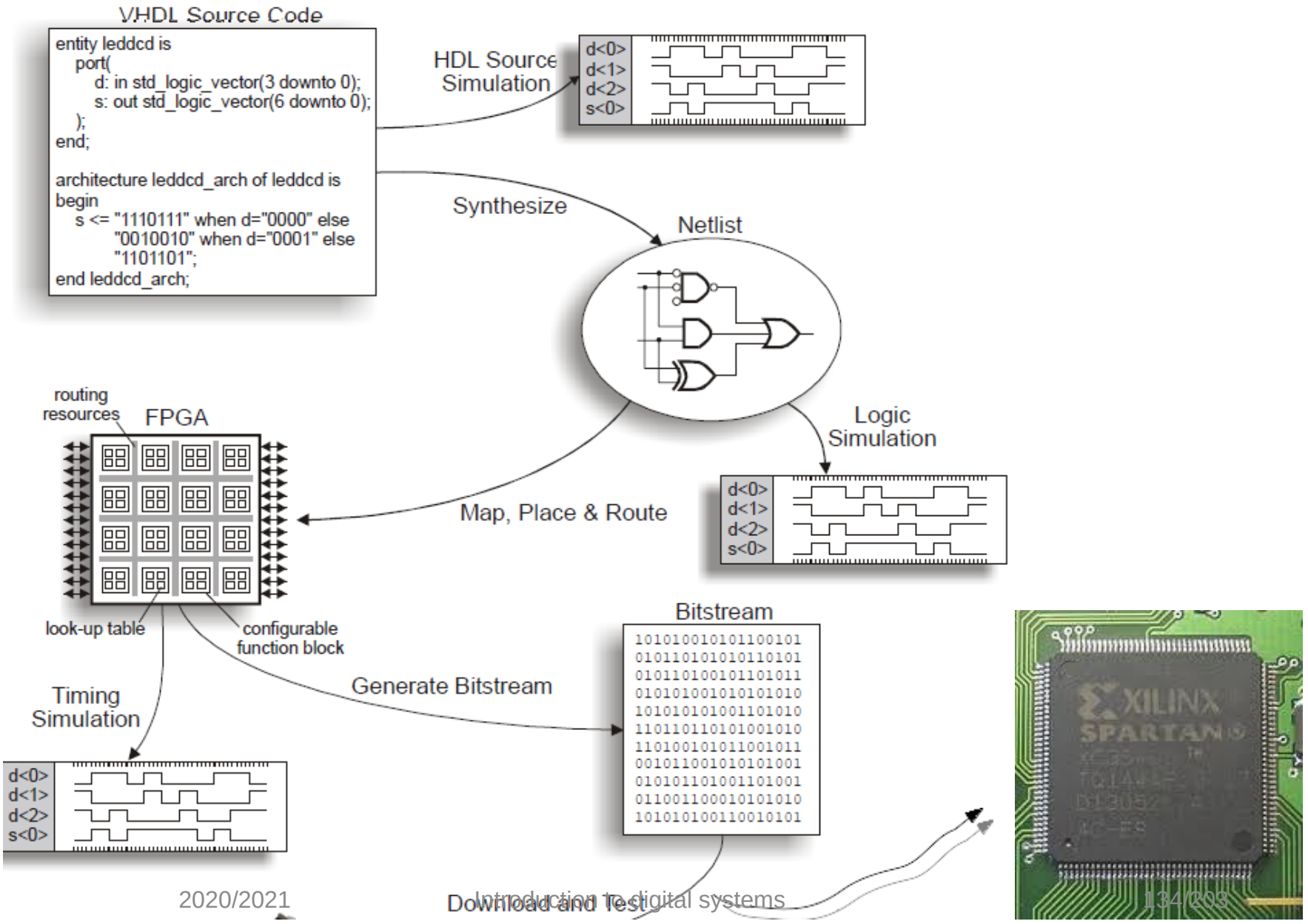


Figure 1 – FPGA tool flows have evolved over time to become more ASIC-like.

FPGA - design


VHDL Source Code
entity leddcd is
port(
d: in std_logic_vector(3 downto 0);
s: out std_logic_vector(6 downto 0);
);
end;
architecture leddcd_arch of leddcd is
begin
s <= "1110111" when d="0000" else
"0010010" when d="0001" else
"1101101";
end leddcd_arch;
HDL Source Simulation
d<0>
d<1>
d<2>
s<0>
Synthesize
Netlist
routing resources
FPGA
look-up table
configurable function block
Map, Place & Route
Logic Simulation
d<0>
d<1>
d<2>
s<0>
Bitstream
101010010101100101
010110101010110101
010110100101101011
010101001010101010
101010101001101010
110110110101001010
110100101011001011
001011001010101001
010101101001101001
011001100010101010
101010100110010101
Generate Bitstream
Timing Simulation
d<0>
d<1>
d<2>
s<0>
Download and Test
XILINX
SPARTAN

FPGA - design

SYNTHESIS:

- Analyses the HDL code and infers specific design blocks or macros. Provides a netlist and the RTL schematics

TRANSLATE:

- Merges all netlists and design constraints to describe the logical design (reduced to the manufacturer primitives)

MAP, PLACE AND ROUTE :

- maps the translated files into FPGA elements (CLB, IOs, …)
- place the FPGA elements and route (configure links between CLB, IOs, …)

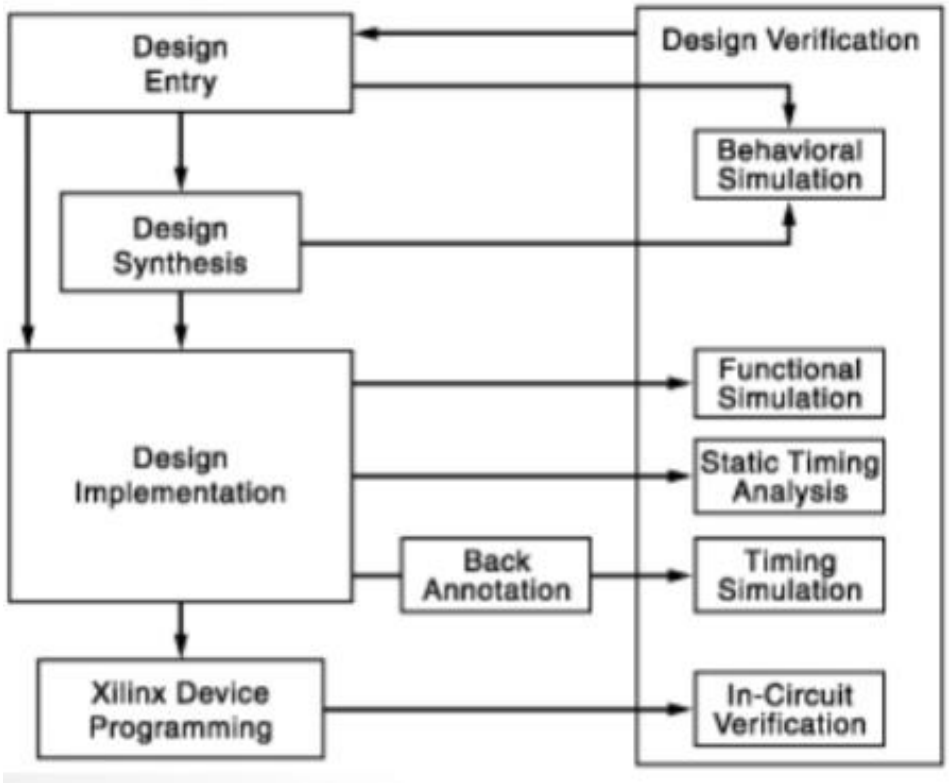
BITSREAM GENERATION:

- generates the programming file to be downloaded to FPGA memory

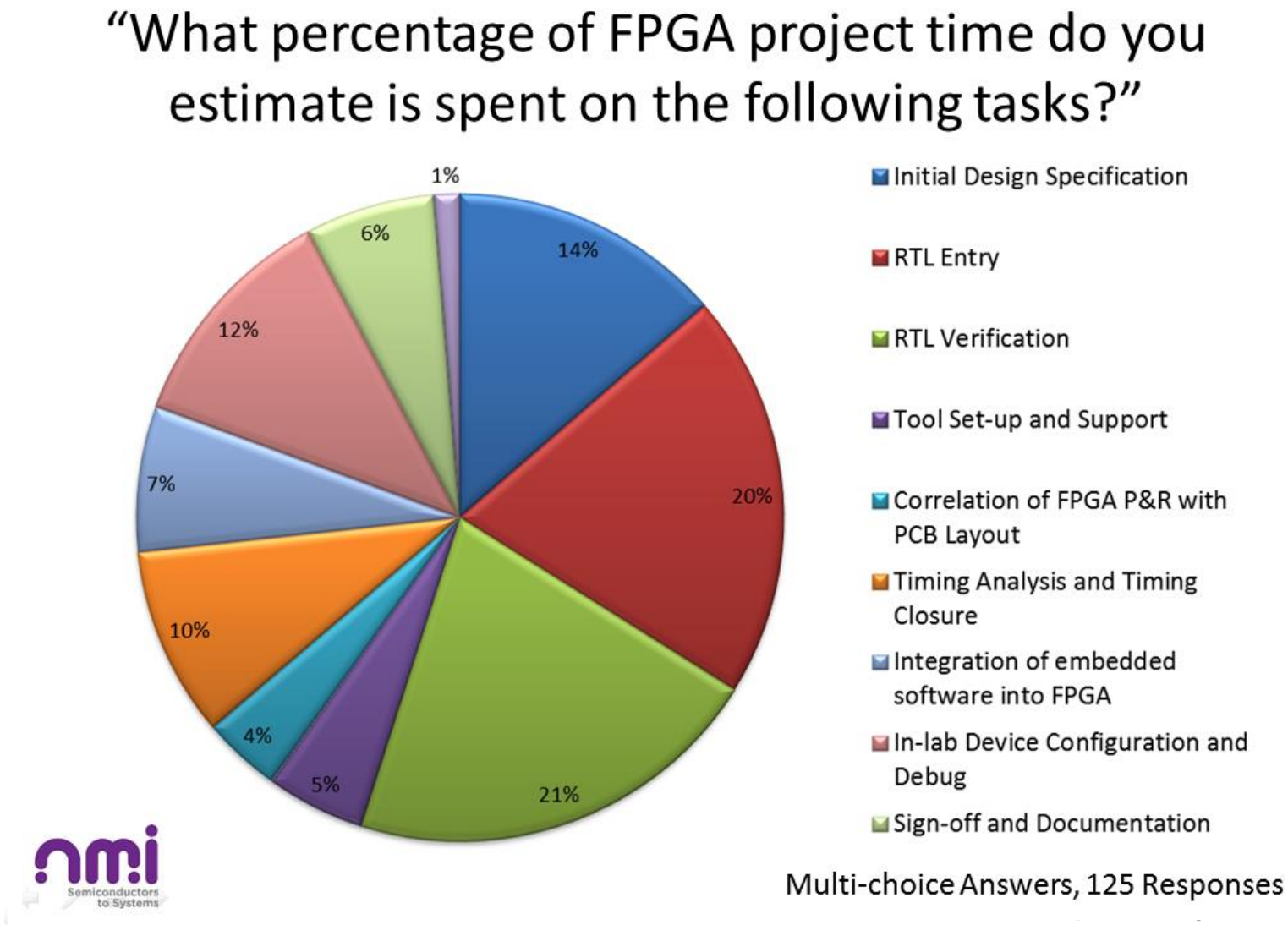
FPGA - design

- ✓ Each step of each HDL module could be validated by simulation

Strongly recommended!



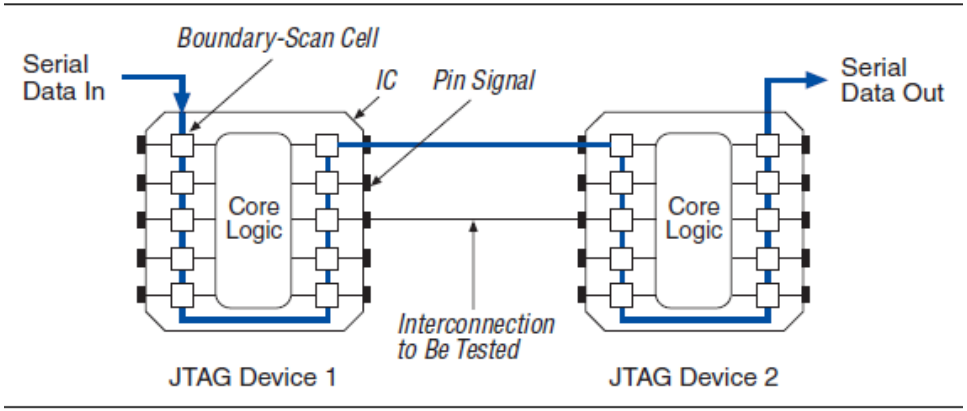
FPGA - design



http://www.testandverification.com/thought-leadership/fpgas-really-get-verified-enough-22-january-2015/

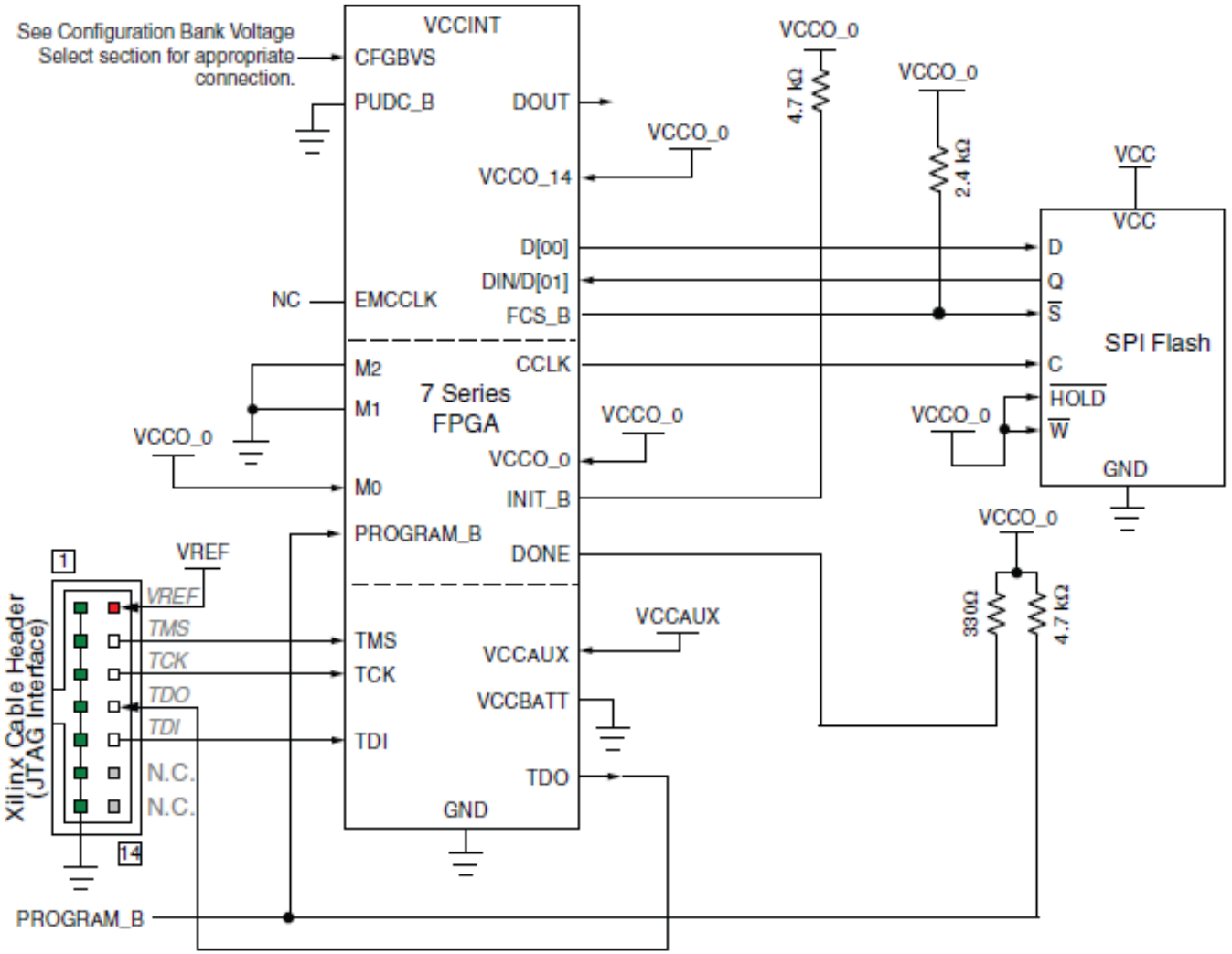
FPGA - design

- ✓ Programming: JTAG (Joint Test Action Group), IEEE 1149.1



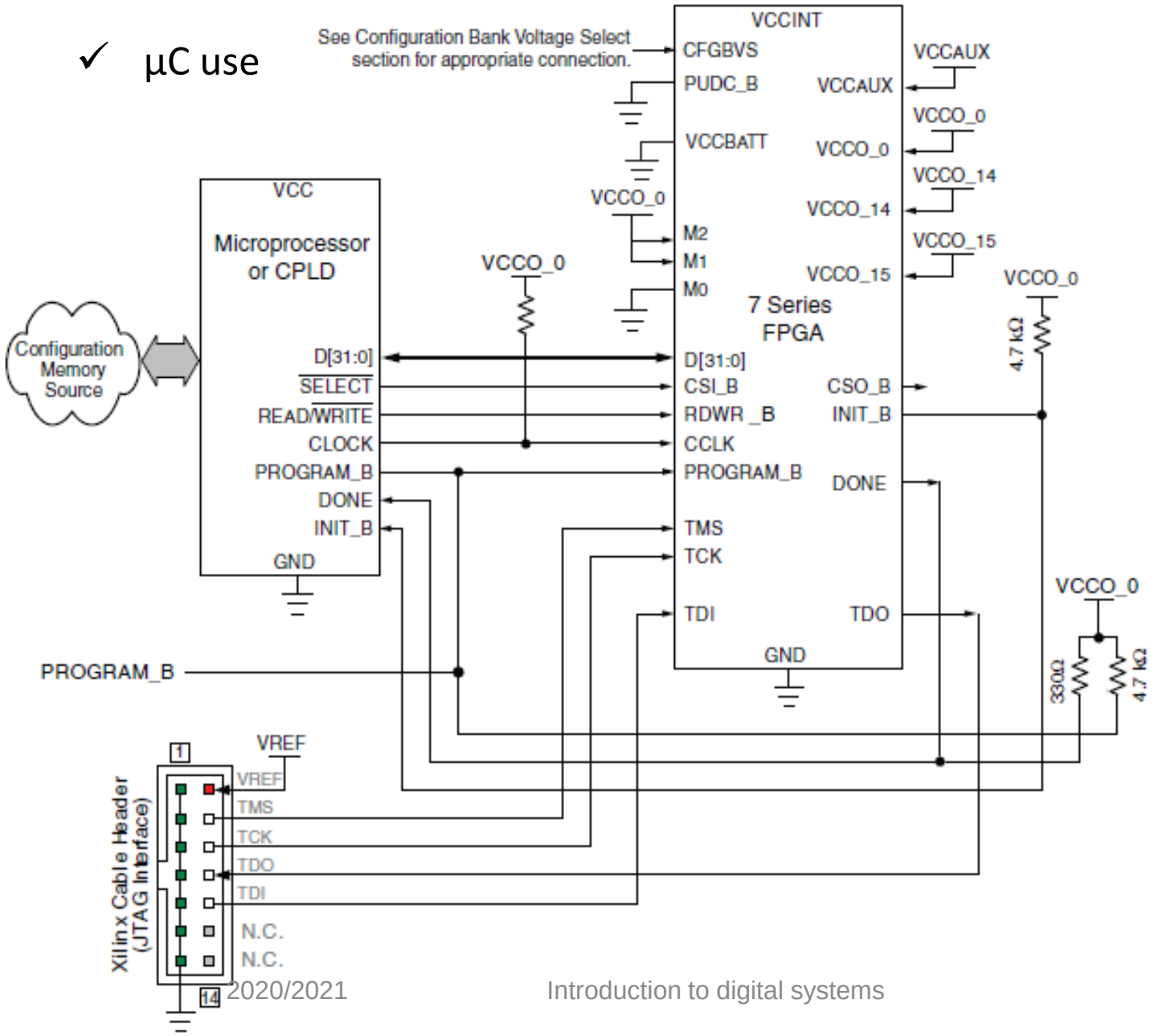
FPGA - design

- ✓ Flash SPI use



FPGA - design

- ✓ µC use



Advantages and drawbacks ?

QUIZ

Let's go to https://kahoot.it/

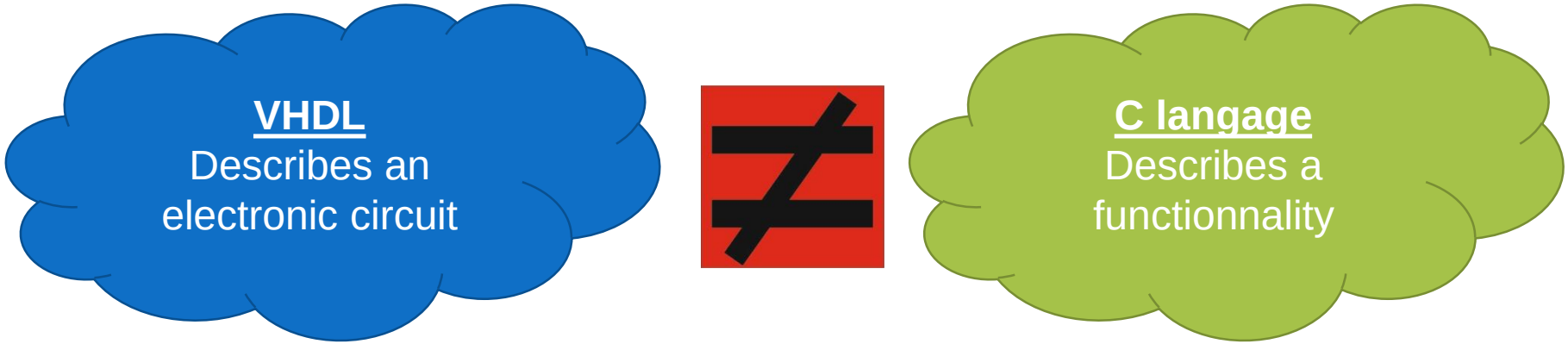
VHDL - basics

VHDL = Very high speed integrated circuit **Hardware Description** Language

- ✓ IEEE Standard
- ✓ Used to design ASICs, FPGA and CPLD, model and simulate
- ✓ High level language

- χ Strongly typed
- χ Semantic close to ‘C’ language but **TOTALLY DIFFERENT**

VHDL - basics



Basic rules of **VHDL:**

Case insensitive

Free formatting

Strongly typed

Identifiers = letters, digits, underscore and must start with a letter

Comments begin with --

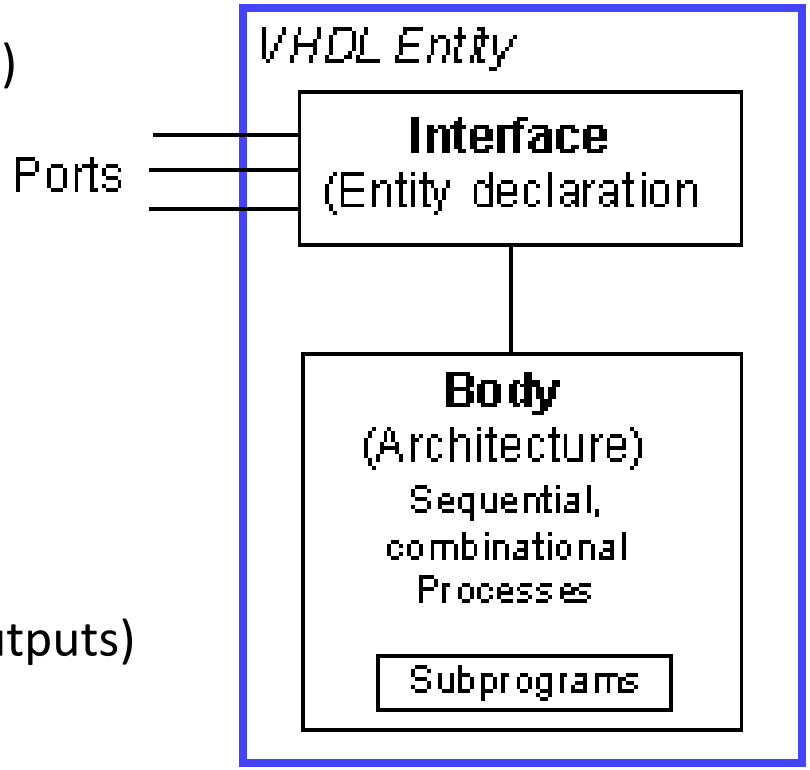
VHDL – entity and architecture

Basic structure: Contains at least an entity declaration and its architecture

✓ **Entity declaration**

- Describes the interface with the external circuit (i.e. input and output signals)
- Signal flow: in, out, inout

⇒ *Block seen from outside*



✓ **Architecture**

- describes the internal working (i.e. relationship between inputs and outputs)

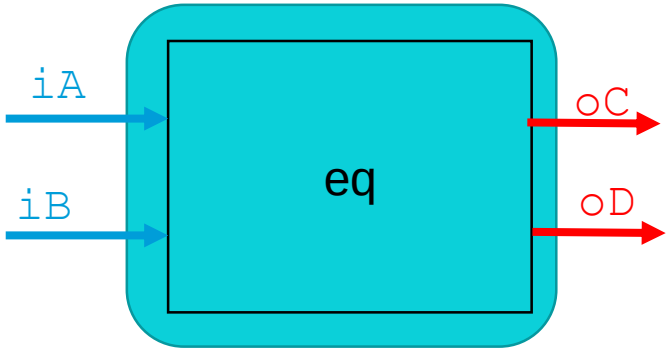
⇒ *Inside the block*

VHDL - entity and architecture

Basic structure example:

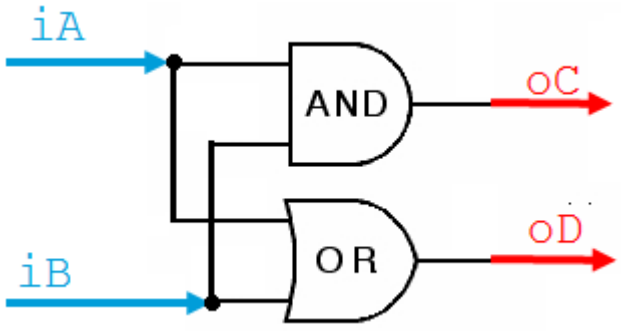
✓ **Entity declaration**

```
entity eq is
  port (
    iA, iB : in std_logic;
    oC, oD : out std_logic
  );
end eq;
```



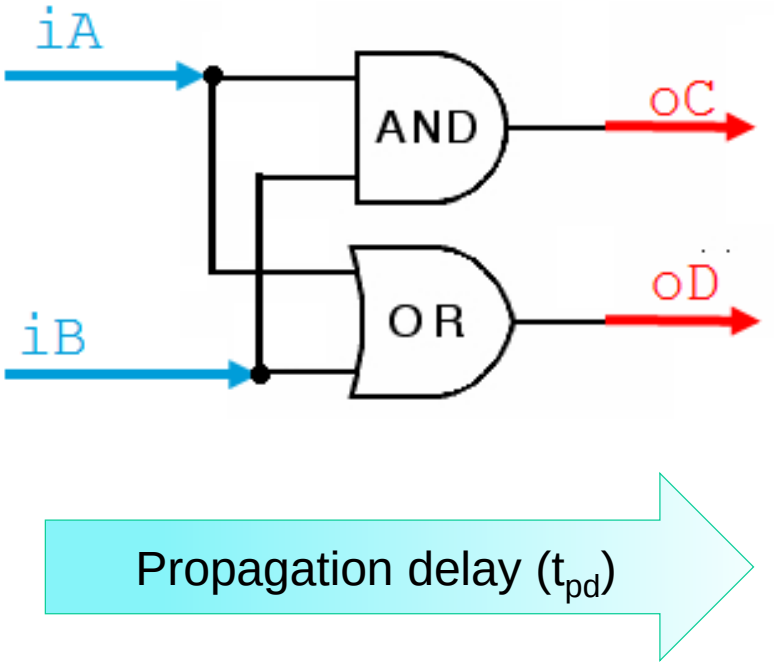
✓ **Architecture declaration**

```
Architecture arch of eq is
begin
  oC <= iA and iB;
  oD <= iA or iB;
end eq;
```



VHDL - entity and architecture

- ✓ All statements are **concurrent (= in parallel)**
- ✓ The outputs (oC and oD) are settled after a **propagation delay** (LUT)



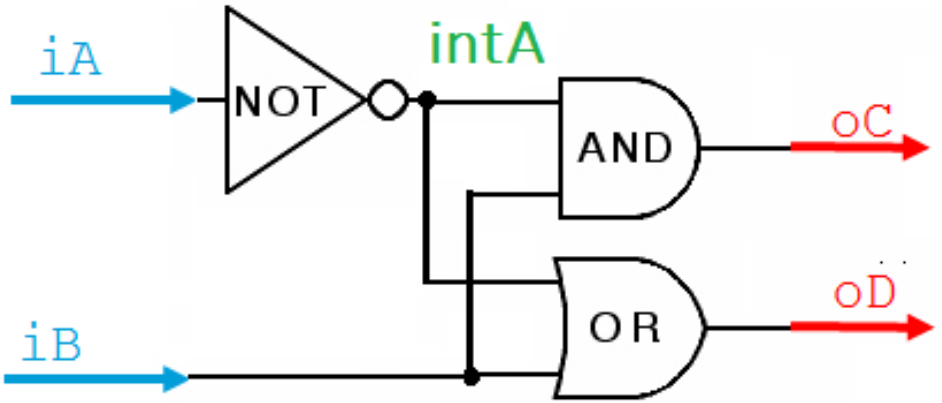
VHDL - entity and architecture

Architecture can include a declaration section

- constant, internal signal, …

✓ Example of a signal:

```
Architecture arch of eq is

   -- declaration section
   signal intA : std_logic;
   -- end declaration section

begin
   intA <= not iA;
   oC <= intA and iB;
   oD <= intA or iB;
end eq;
```



VHDL - entity and architecture

- ✓ An **output** can **only be used as the destination** of an expression

```
entity toto is
port ( a, b : in std_logic;
       z, no_z : out std_logic);
end toto;

architecture wrong of toto is
begin
       z <= a and b;
        no_z <= not z;
end wrong;

architecture good of toto is
       signal resultat : std_logic;
begin
       resultat <= a and b;
       z <= resultat;
        no_z <= not resultat;
end good;
```

VHDL – data formats

✓ VHDL was designed for different tasks (?) => lots of different data types

χ But only a small part can be *synthetized* (i.e. can be used for FPGA design)

- *std_logic*: 9 possible values ; only 3 are suitable for synthesis
  '0', '1', and 'Z' (high impedance)
- *std_logic_vector*: array of std_logic

```
std_logic_vector(MSB downto LSB)
std_logic_vector(LSB to MSB)
```

- integer, boolean, natural, positive
- enumerated format (see FSM)

VHDL - Operators

Operators grouped according to priority level, highest priority first

- expressions with the same operator priority are evaluated form left to right

➢ Use parenthesis to clearly specify the evaluation order

miscellaneous operators	** | abs | not
multiplying operators	* | / | mod | rem
sign operators	+ | -
adding operators	+ | - | &
shift operators	sll | srl | sla | sra | rol | ror
relational operators	= | /= | < | <= | > | >=
logical operators	and | or | nand | nor | xor | xnor

VHDL - Operators

- ✓ Logical operators: or, and, not, xor …

```
signal a, b, c, r : std_logic := '1';
r <= a and b xnor a nor c;
```

r?

- ✓ Comparison operators: =, <, >, /=
  - Boolean operation (bit to bit comparison, take care of size)
  - Both operands must be of the same type
  - Return a Boolean (true or false)

- ✓ Concatenation: &

```
signal A : std_logic_vector(2 downto 0);
A <= '0' & '1' & '0';
```

VHDL - Operators

Operators:

- ✓ Arithmetic operators +, -, *, /, **, mod, rem, abs
  - mod and rem only defined for integer

```
variable A, B : integer;
A := 2**3 + 5*3;
B := A mod 2;
```

B?

- ✓ Arithmetic operators:

sll	Shift left logical	(filled with 0)
srl	Shift right logical	(filled with 0)
sla	Shift left arithmetic	(keep the original sign)
sra	Shift right arithmetic	(keep the original sign)
rol	Rotate left logical	
ror	Rotate right logical	

  - ➢ prefer « shift_left » and « shift_right » (ieee.numeric_std)

```
r_Unsigned_L <= shift_left(unsigned(r_Shift1), 1);
```

VHDL – 3 types of modeling

- Dataflow modeling

Direct assignements,

Mostly used for combinatorial logic

```
m <= (x and (not s)) or y;
```

- Structural modeling

Description of the connections between predefined blocks or primitives

```
and_comp_1 : and2 port map (
        i0 => a,
        i1 => b,
        o => a_int          );
```

- Behavioral modeling

Describes the functionnality behaviorally

Mostly used if sequential operation to be executed when a specific signal toogles

```
process (c, d, e)
begin
        <behavioral code here>
end process;
```

VHDL – Introduction to Vivado

Launch XILINX VIVADO 2018.3

(or older version … or *Xilinx ISE 12.1… (/usr/local/ISE_DS/ISE/bin/lin64/ise*))

- ✓ Complex electronics circuit modelisation
- ✓ Behavioral simulation
- ✓ Logic synthesis

top.vhd

Copy and open the project Ex1 (/usr/local/public)

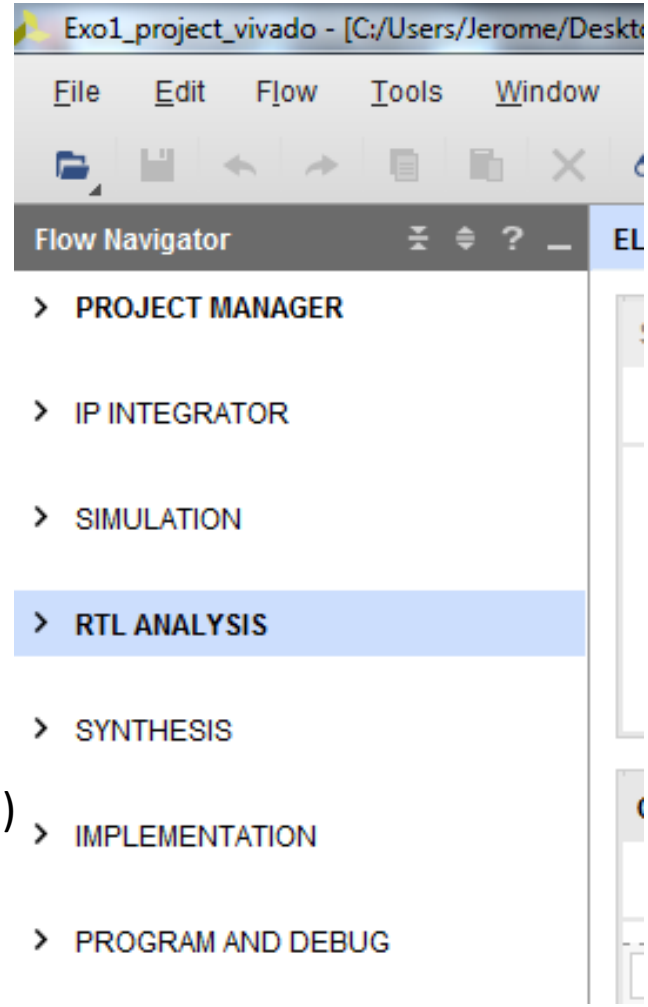
Have look and first conclusions to VHDL syntax

Try to guess the functionality of this module?

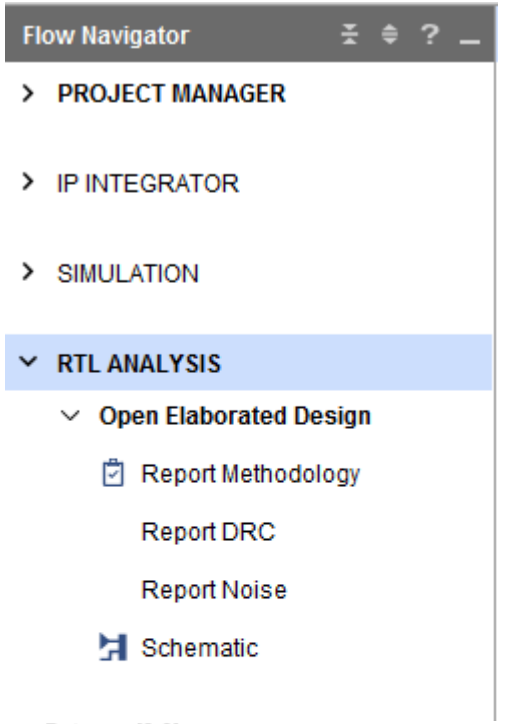
VHDL - Introduction to Vivado

FLOW NAVIGATOR control panel:

- Contains all steps of the design:
  - RTL Analysis
  - Synthesis
  - Implementation
  - Program and Debug
- Simulation:
  - Usefull at each step of design
- Project manager:
  - Main settings (FPGA target choice, language,…)
  - IP catalog
  - Language templates



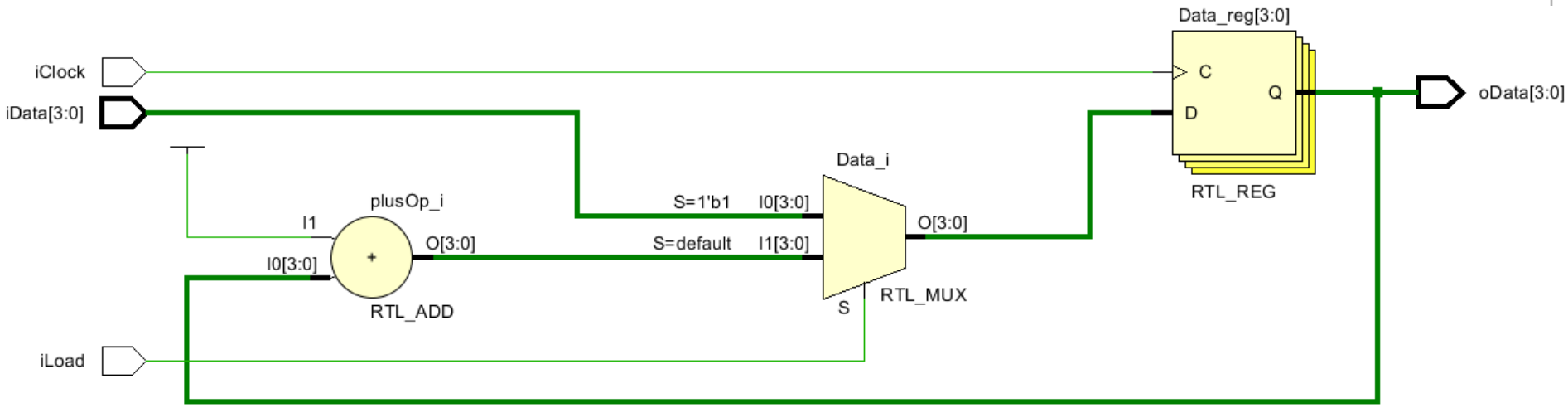
VHDL - Introduction to Vivado



RTL analysis -> Open Elaborated Design

- Check the code syntax
- Implement code into macro functions

➢ RTL schematic (primitive level)

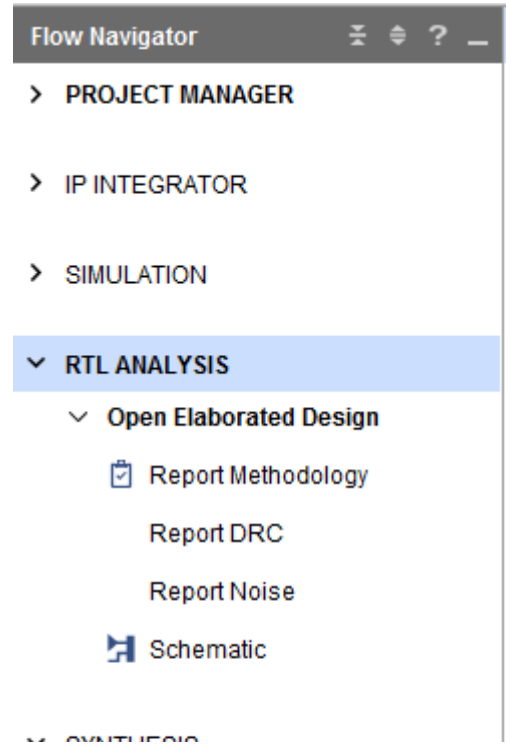


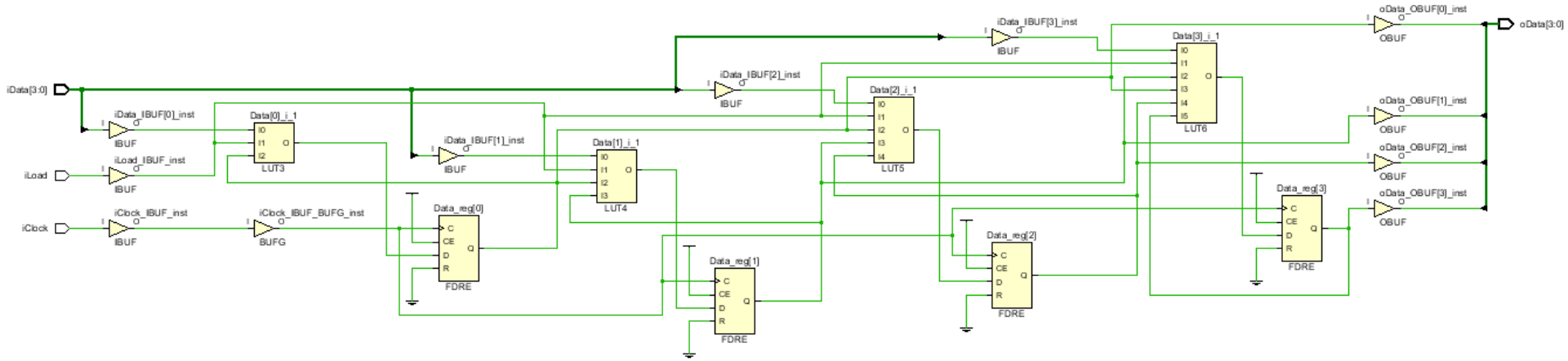
VHDL - Introduction to Vivado

Synthesis -> Run synthesis

- Infers specific design blocks or macros.
- Provide a netlist and schematics

➢ Post-Synthesis schematic: LUT and FF level

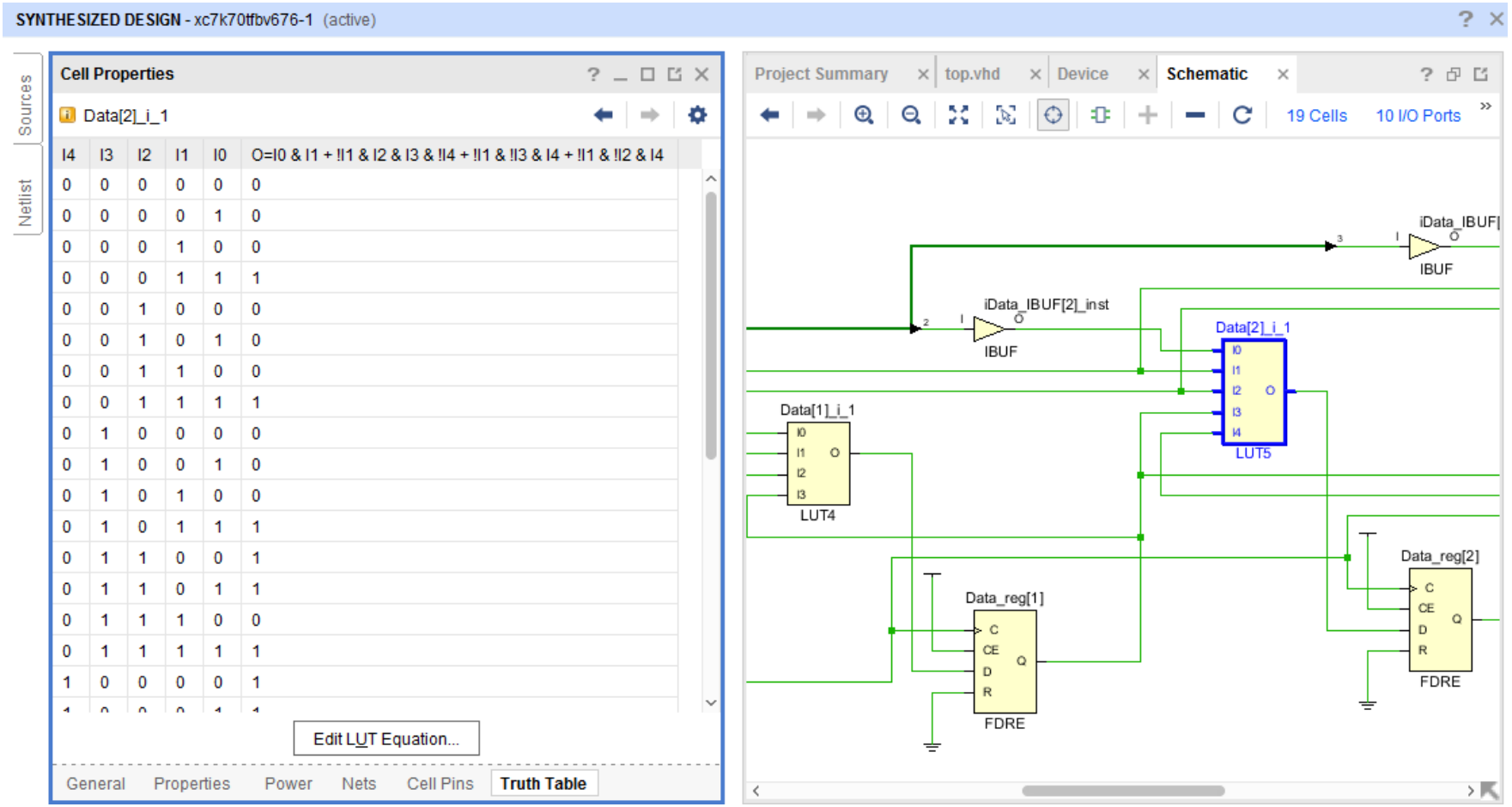




VHDL - Introduction to Vivado

Synthetis -> Run synthesis

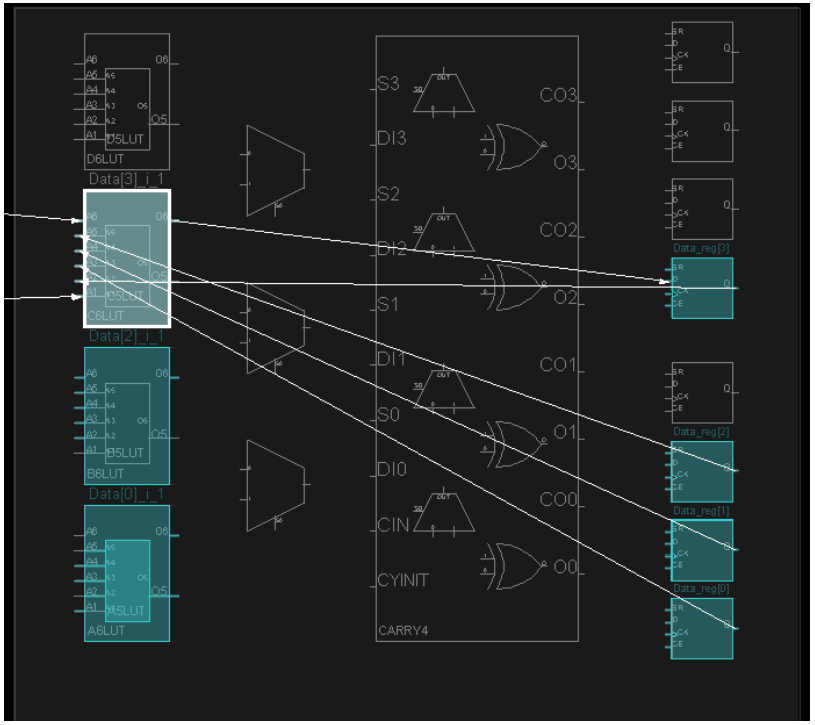
- Truth table of each LUT

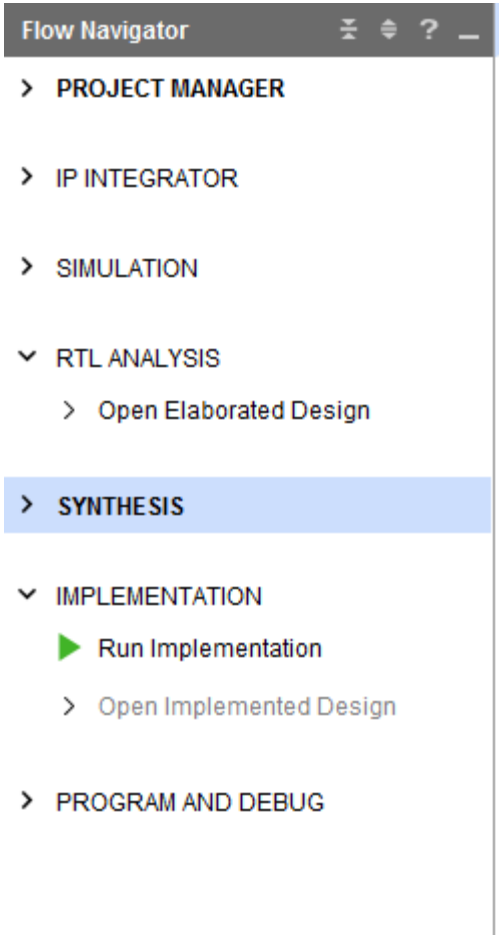


VHDL - Introduction to Vivado

Implementation -> Run implementation

- maps the translated files into FPGA elements (CLB, IOs, …)
- place the FPGA elements and route (configure links between CLB, IOs, …)



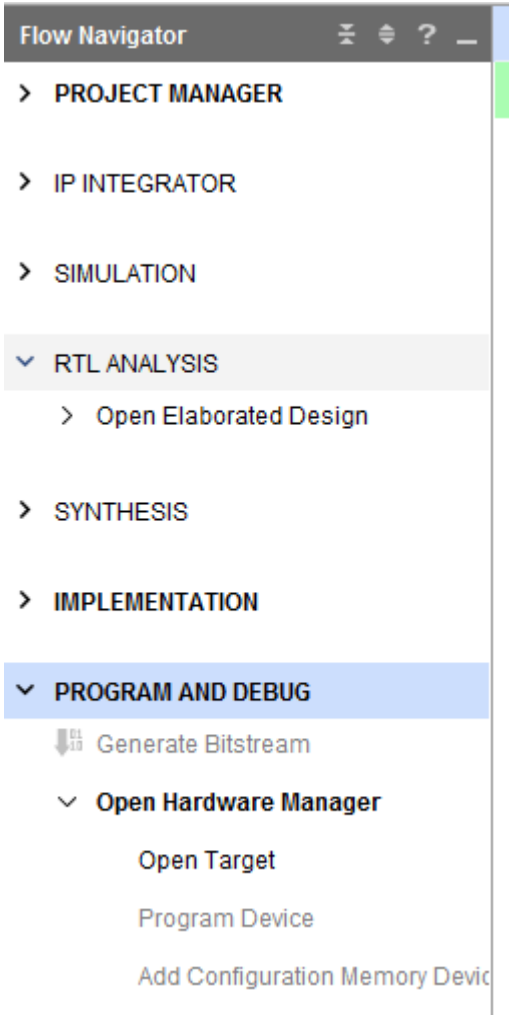


VHDL - Introduction to Vivado

Program and debug -> Generate Bitstream

- generates the programming file to be downloaded to FPGA memory

➢ Ready to program FPGA!



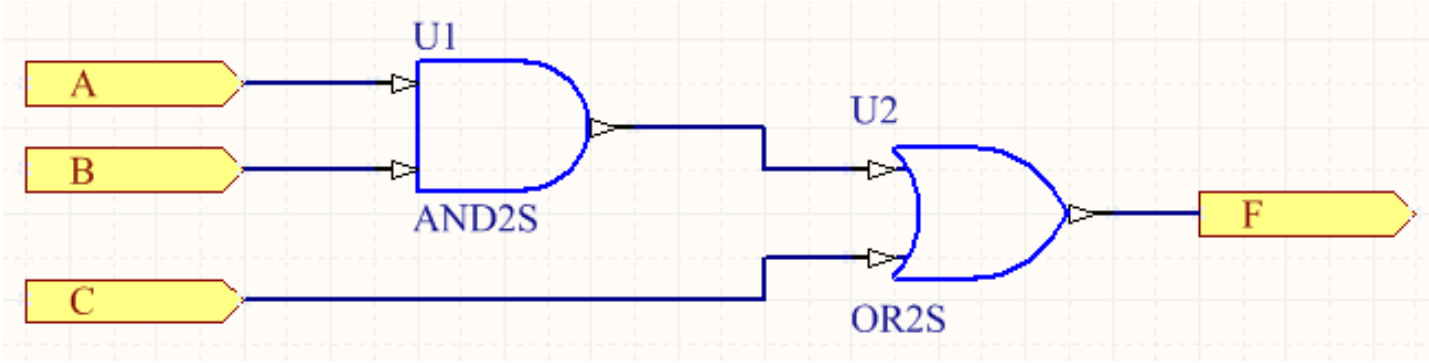
VHDL - basics

Gate-level combinatorial circuit /dataflow modeling

Simple logic gates

=> Exercise 2a

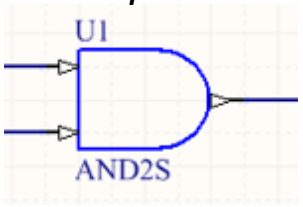
Write a VHDL program to perform the following function, using dataflow modeling:



VHDL - basics

Gate-level combinatorial circuit / structural modeling

example: 2-input AND gate



1- **Declaration of module** (or primitive)

- in the declaration section

```
component and2 port (
        i0, i1: in std_logic;
        o : out std_logic);
end component;
```

2- **Instanciation and mapping**

- in the body of the architecture

```
instance_name : and2 port map (
        i0 => x(1),
        i1 => s_bar,
        o => x_int(1)    );
```

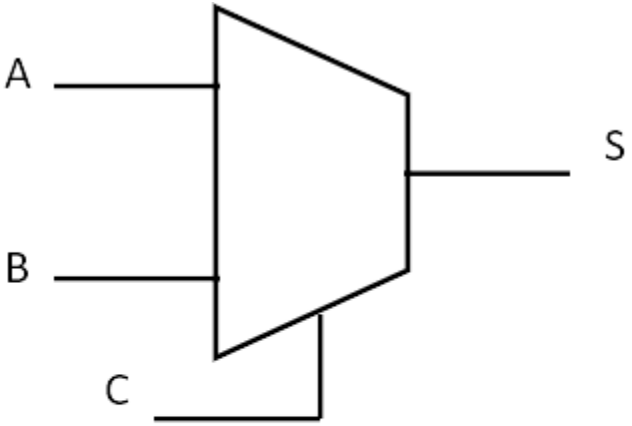
=> Exercise 2b: Rewrite the Ex 2a using structural modeling

top_structural.vhd

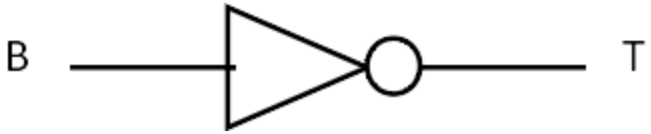
VHDL - basics

RT-level / Dataflow modeling with concurrent assignments

=> Exercise 3. Write a VHDL program to describe the following circuit:



top.vhd



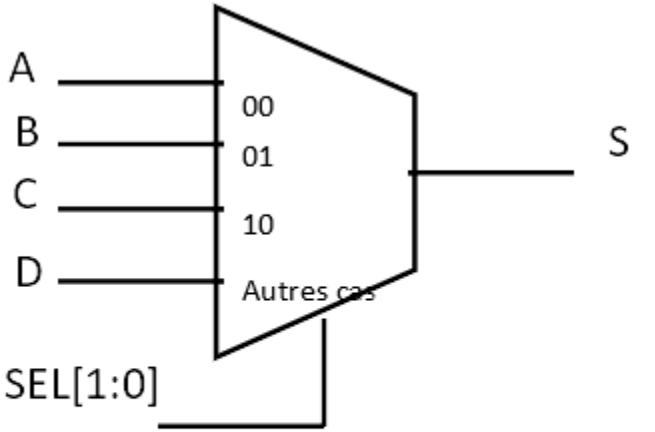
Open the RTL schematics and compare!

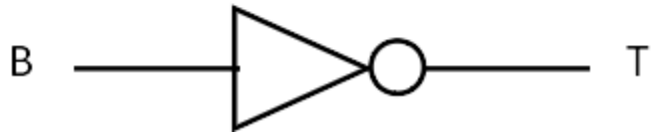
```
signal_x <=     value_expr_1    when boolean_expr_1 else
                value_expr_2    when boolean_expr_2
                                else valeur_z;
```

VHDL - basics

RT-level / Dataflow modeling with concurrent assignments

=> Exercise 4. Write a VHDL program to describe the following circuit:





top.vhd

```
With    sel select
        signal <=     value_expr_1 when val1,
                      value_expr_2 when val2,
                      …
                      value_expr_n when others;
```

VHDL - basics

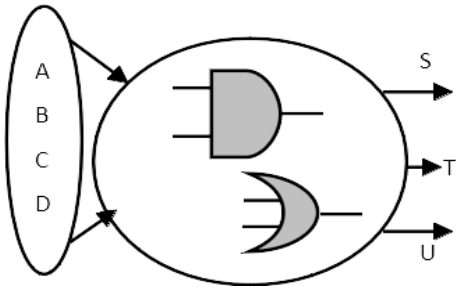
RT-level with a process / Behavioral modeling

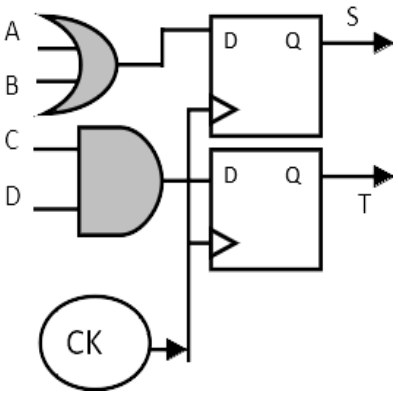
- ✓ Sequential statements, executed in sequence
- ✓ A process itself is a concurrent statement
- χ Pay attention to coding, implying complex implementations
- ✓ Different semantic

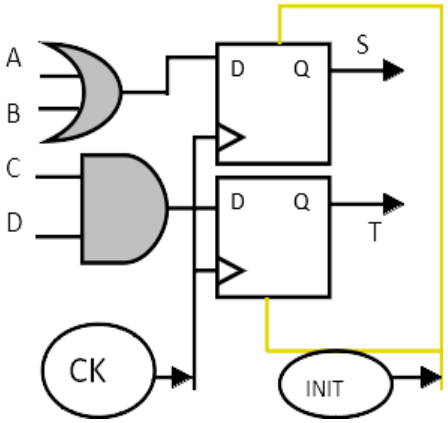
```
[Label] : process (sensitivity list)
        --  variable or signal declaration

begin
        -- sequential statements
end process;
```

VHDL - basics

RT-level with a process







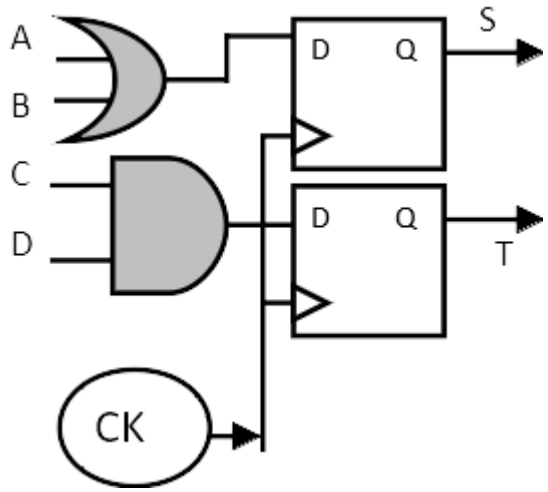

```
process(A,B,C,D)
begin
  if(A='1'and B='0')then
     ...
  end if;
end process;
```

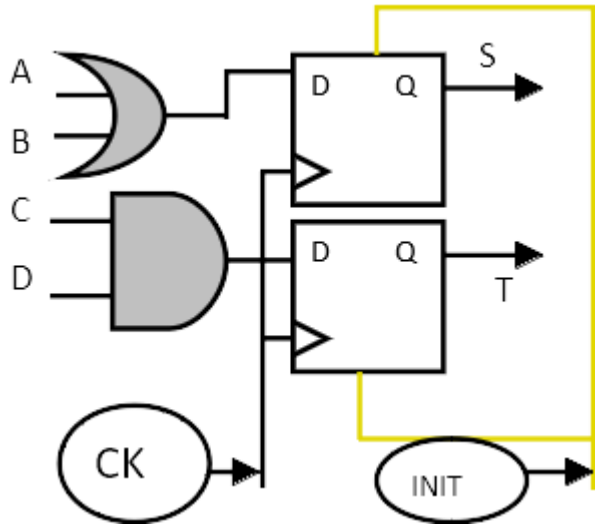
```
process(CK)
begin
  if (CK'event and CK='1') then
    if (A='1' and B='0') then
       ...
    end if;
end process;
```

```
process(CK, INIT)
begin
  if INIT='1' then
     S <='1'; T<= '0';
  elsif (CK'event and CK='1') then
       if (A='1' and B='0') then
       ...
      end if;
  end if;
end process;
```

VHDL - basics

RT-level with a process – **not to be done**






```
process(CK, A, B, C, D)
begin
  if (CK'event and CK='1') then
    if (A='1' and B='0') then
       ...
    end if;
end process;
```

```
process(CK, INIT, A, B, C, D)
begin
  if INIT='1' then
     S <='1'; T<= '0';
  elsif (CK'event and CK='1') then
      if (A='1' and B='0') then
       ...
    end if;
  end if;
end process;
```

VHDL - basics

A VHDL program is a collection of concurrent processes.

- A signal assignement is considered as a processus

A process is executed in an infinite loop, unless

- a **sensitivity list** is defined

OR

- « **wait**; » command at the end of the process (refer to simulation below)

Note: *sensistivity list AND wait command could not be used in the same process*

All signals in a process are **updated simultaneously**:

- at the end of the process activated by a sensitivity list
- At the next « wait » command

- All calculus are done with initial value and then updated

```
Consider a = 4 and b = 1, and the following command in a process
b <= a;
c <= b;        The results are: b = 4
                                C = 1
```

VHDL - basics

If statement

Boolean expression are evaluated sequentially until an expression is evaluated as true or else statement is reached
The corresponding branch is executed

```
if boolean_expr_1 then
        -- sequential statements
elsif boolean_expr_2 then
        -- sequential statements
…
elsif boolean_expr_3 then
        -- sequential statements
else
        -- sequential statements
end if;
```

VHDL - basics

Case statement

- ✓ Use the ‘sel’ signal to select the execution of statements
- ✓ Choices must be exclusive
- ✓ ‘Others’ covers all other cases

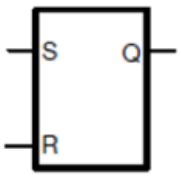
```
case sel is
        when choice_1 =>
                -- sequential statements
        when choice_2 =>
                -- sequential statements
        …
        when others =>
                -- sequential statements
end case;
```

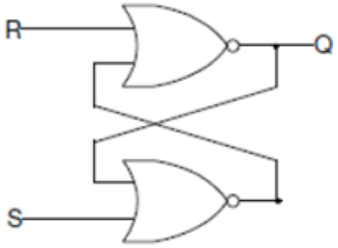
VHDL - basics

RT-level with a process / Behavioral modeling

=> Exercise 9. Basic structures with process statements

1- Write a VHDL program to describe a RS flip-flop

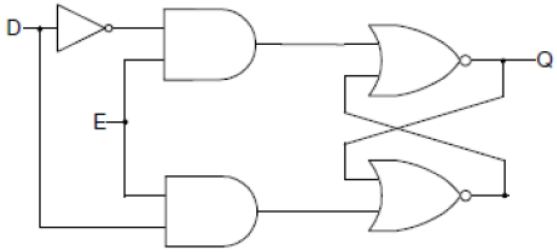




S	R	Q
0	0	Latch
0	1	0
1	0	1
1	1	Metastable


RS2.vhd

2- Write a VHDL program to describe a D latch



Enable	D	Q
0	0	Latch
0	1	Latch
1	0	0
1	1	1


latch.vhd

VHDL - testbench

Simulation purpose: ensure that the hardware design meets the requirements

Use of testbench to provide stimulii close to real application

Use the same environment

```
entity test is
end test;

architecture behavior of test is

   -- component declaration for the unit under test (uut)
    component name_of_component_to_test
       port(
          iA, iB : in  std_logic;
          oC, oD : out  std_logic
       );
    end component;
```

VHDL - testbench

```
--declare inputs and outputs and initialize them
   signal …

-- clock period definitions
   constant clk_period : time := 1 ns;

begin

-- instantiate the unit under test (uut)
   uut: name_of_component_to_test port map (
             iA => iA,
             …);

-- clock process
   clk_process : process
   begin
       clk <= '0';
       wait for clk_period/2;
       clk <= '1';
       wait for clk_period/2;
   end process;
```

VHDL - testbench

```
-- stimulus process
  stim_proc : process
   begin
        wait for 10 ns;
        iA <='1';
        wait for 11 ns;
        iB <='0';
        …
        wait;
  end process;

end;
```

Example of VHDL module and associated testbench of the previous exercises:

VHDL - basics


flip_flop.vhd

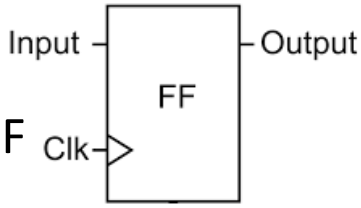

flip_flop_tb.vhd

RT-level with a process / Behavioral modeling

=> Exercise 9. Basic structures with process statements

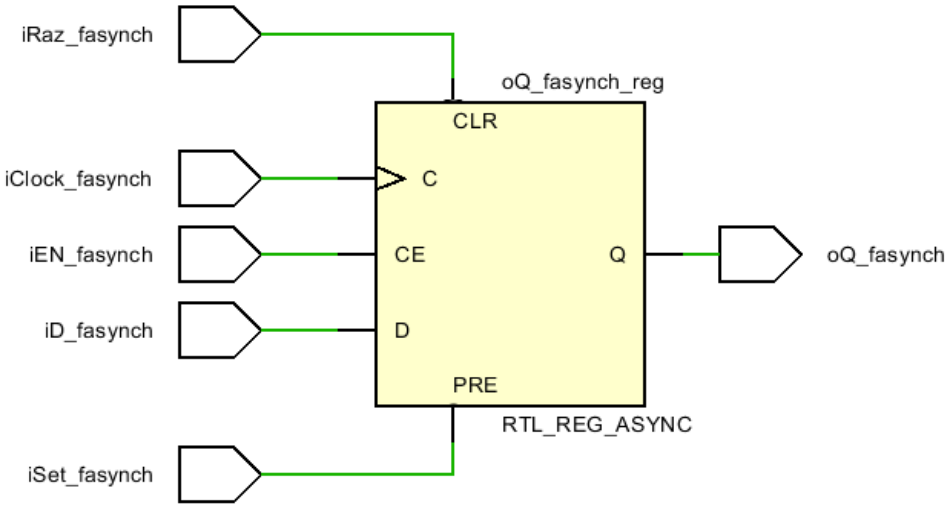
3- Write a VHDL program to describe a simple Flip-Flop.

Create a test bench, and validate the functionality of this FF



4- Write a VHDL program to describe a Flip-Flop called « f_async »
with asynchronous set and reset, and enable input

Create a test bench, and validate the functionality of this FF.




f_async.vhd


f_async_tb.vhd

VHDL - reuse

Library and package

Allow to reuse previous definitions (data types for example) or previous code

A library is a collection of related packages.

Example: IEEE library

```
library IEEE;

use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use IEEE.std_logic_arith.all;
use IEEE.numeric_bit.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;
use IEEE.math_complex.all;
```

VHDL - reuse

Library and package

The package ***std_logic_1164*** provides enhanced signal types. Types defined include: std_logic std_logic_vector

The package ***std_logic_arith*** provides:

- Signed and unsigned type definition
- Arithmetic (+, -, *) and comparison function (<, <=, >, >=, =, /=) on these types

The package ***std_logic_signed*** provides signed numerical computation on type std_logic_vector

The package ***std_logic_unsigned*** provides unsigned numerical computation on type std_logic_vector

VHDL - reuse

Library and package

The package ***numeric_std*** provides:

- Signed and unsigned type definition
- Arithmetic (+, -, *) and comparison function (<, <=, >, >=, =, /=) on this types

  ⇒ Main differences with previous std_logic_xxxx:

  ✓ Supported by all synthesis tool.

  ❖ Type casting required.

The package ***numeric_bit*** provides numerical computation. Types defined include: unsigned signed arrays of type bit for signals

The package ***math_real*** provides numerical computation on type real

The package ***math_complex*** provides numerical computation. Types defined include: complex, complex_vector, complex_polar

The package ***std_logic_textio*** provides input/output for 1164 types

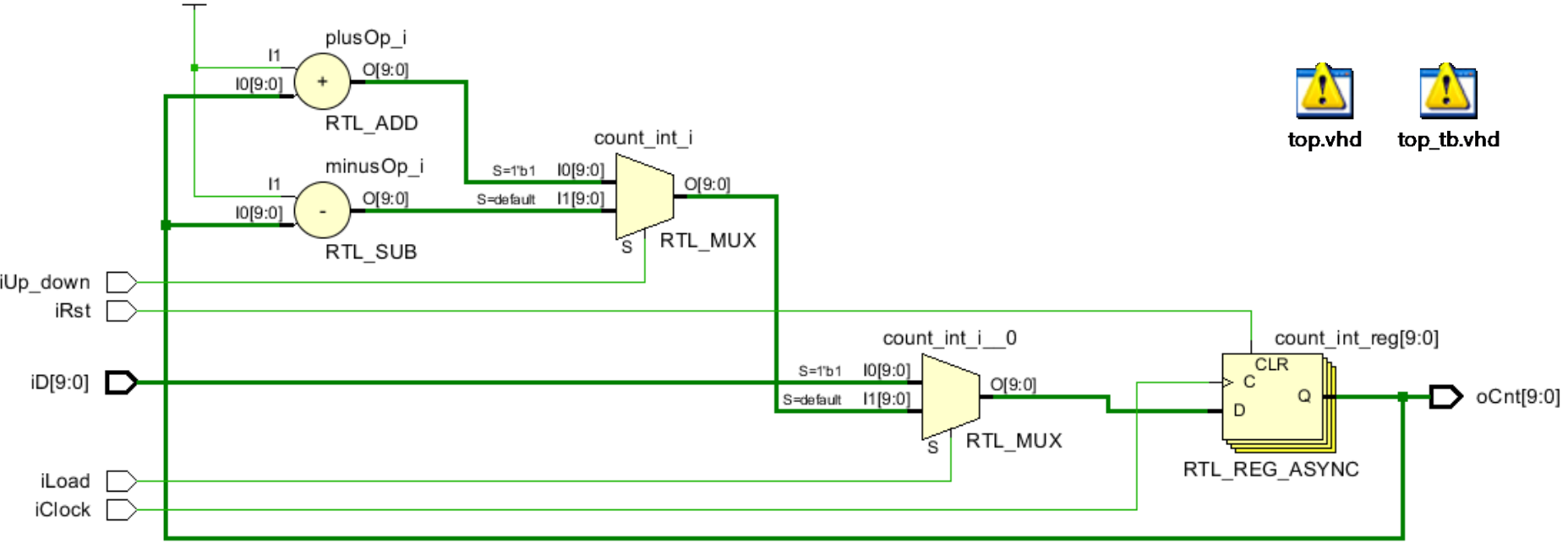
VHDL - basics

RT-level with a process / Behavioral modeling

=> Exercise 10. Summary

Write a VHDL program to describe a loadable up/down 10 bits counter with asynchronous reset

Create a test bench, and validate the functionality of this counter



VHDL - reuse

How to create your own package?

```
package package_name is
      -- Declaration of
            -- types and subtypes
            -- functions
            -- constants, signals etc.
end package_name;
```


```
package body package_name is
      -- Definition of previously declared
         -- constants
         -- functions
      -- Declaration/definition of additional
         -- types and subtypes
         -- functions
         -- constants, signals and shared variables
end package_name;
```

VHDL - reuse

How to use your own package?

The package is compiled to the library “work” by default.

Then, include the particular package of this library “work”.

```
library work;
use work.package_name.all;
```

Example:


test_pkg.vhd


test.vhd

VHDL - reuse

Constants and generics

Constant are used to help code to be clear and maintainable

```
constant const_name: data_type := const_value;
```

Generics are a means of passing specific information into an entity.
Very useful to instantiate a module multiple times with slight changes (depth of registers,…)

```
entity entity_name is
        generic( name: data_type := default_value);
        port(
        …
        );
end entity_name;
```

generic example_2.vhd

VHDL - reuse

Component and port map

Used to define basic components and reuse them latter in VHDL code (~ buy circuits and solder them to make a bigger function)

1- Declaration of the component in the declaration section

```
component component_name port (
                    i0, i1: in std_logic;
                    …
                    r: out std_logic  );
end component;
```

2- Instantiation and mapping in the body of the architecture with **port map**

```
component_name port map (
                pin_on_module => signal_name,
                pin_on_module => signal_name,
                …
                pin_on_module => signal_name
);
```

VHDL - reuse

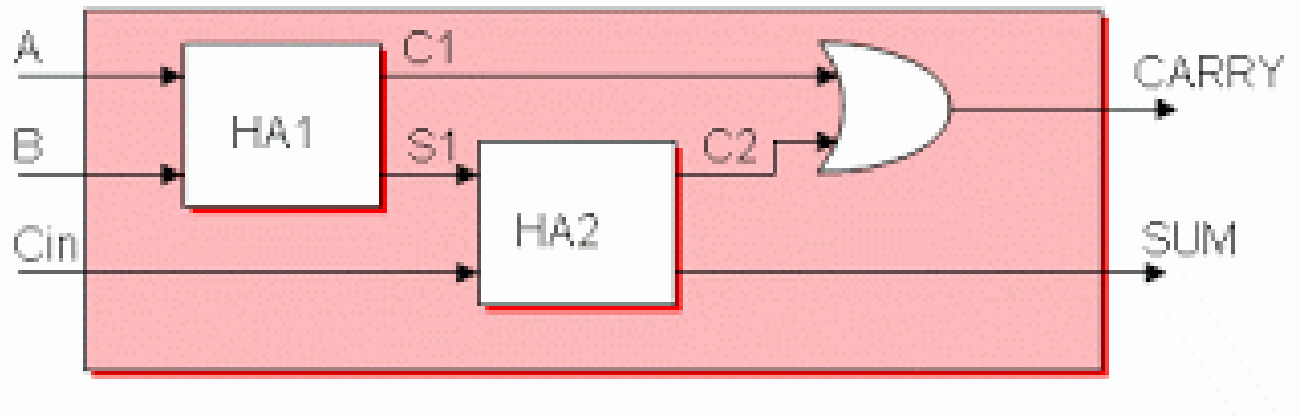
Component and port map

⇒ **Exercise 13:**

1- Describe in VDHL a half-adder

demi_add.vhd

2- Using this half-adder and an or-gate, describe a full-adder using component instantiation



full_add.vhd

3- Create a testbench for the full adder, and verify the functionality of this module

fulladder_sim.vhd

VHDL – reuse

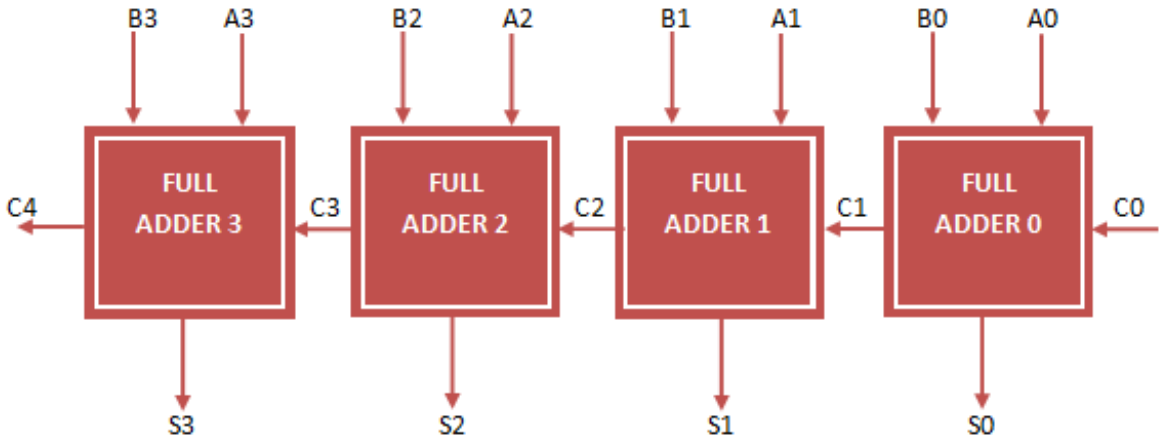
RT-level with concurrent assigment statement

⇒ **Exercise 14**

1- Using the full adder of Exercise 13, write a VHDL program to generate a 4-bit adder

top.vhd

```
LABEL: for I in int_a to int_b generate
        -- concurrent instructions
end generate;
```



2- Write a testbench for the 4-bit adder of Ex 14, and verify the functionality of this module

top_tb.vhd

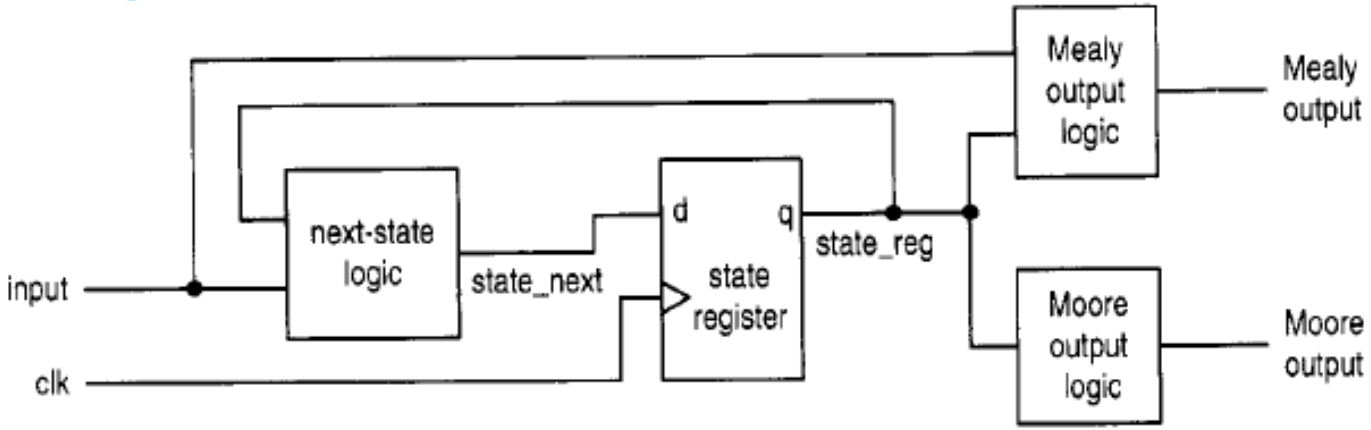
VHDL - FSM

- ✓ FSM ? mathematical model of computation used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of states. The machine is in only one state at a time; the state it is in at any given time is called the current state. It can change from one state to another when initiated by a triggering event or condition; this is called a transition. A particular FSM is defined by a list of its states, its initial state, and the triggering condition for each transition.

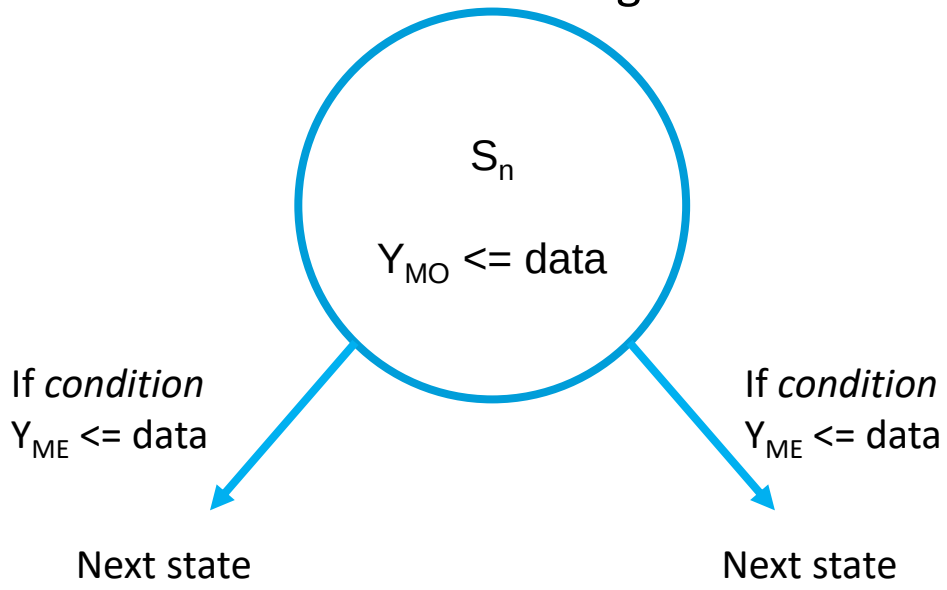
  (*wikipedia.org*)

- ✓ Tool to represent sequential system, with different states
- ✓ State transition if conditions are verified
- ✓ Outputs are dependent on:
  - Current state only (Moore), state changes after clock
  - Current state and inputs (Mealy), less states than Moore
  - Both (Moore + Mealy)

VHDL - FSM



- ✓ Easy to implement in VHDL
- ✓ Representation: ASM chart or state diagram

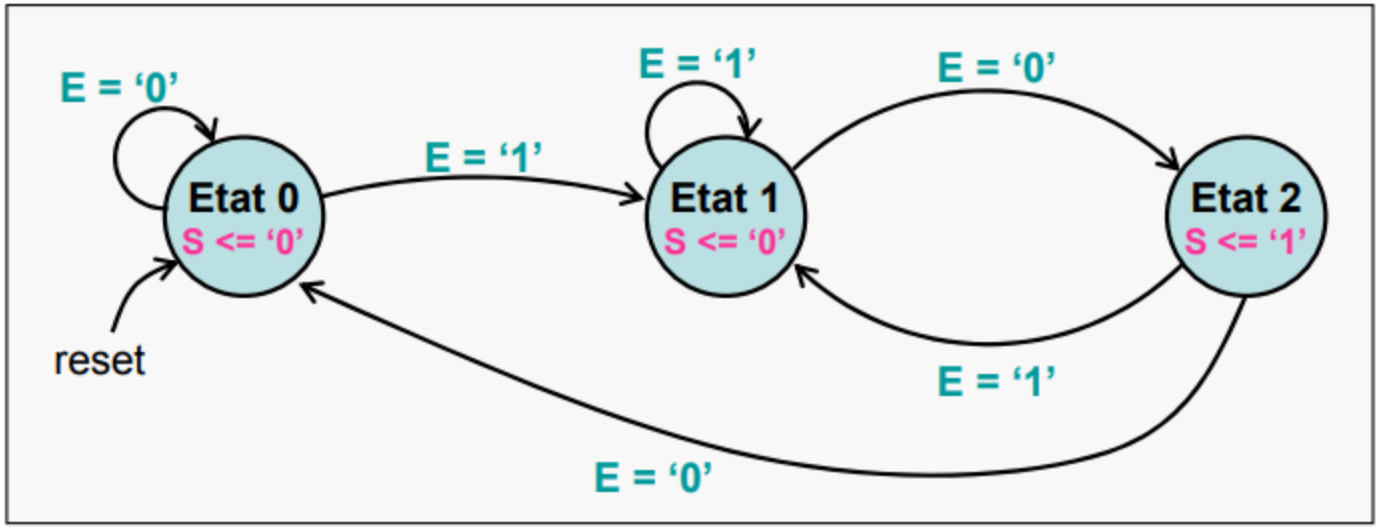


MO = Moore
ME = Mealy

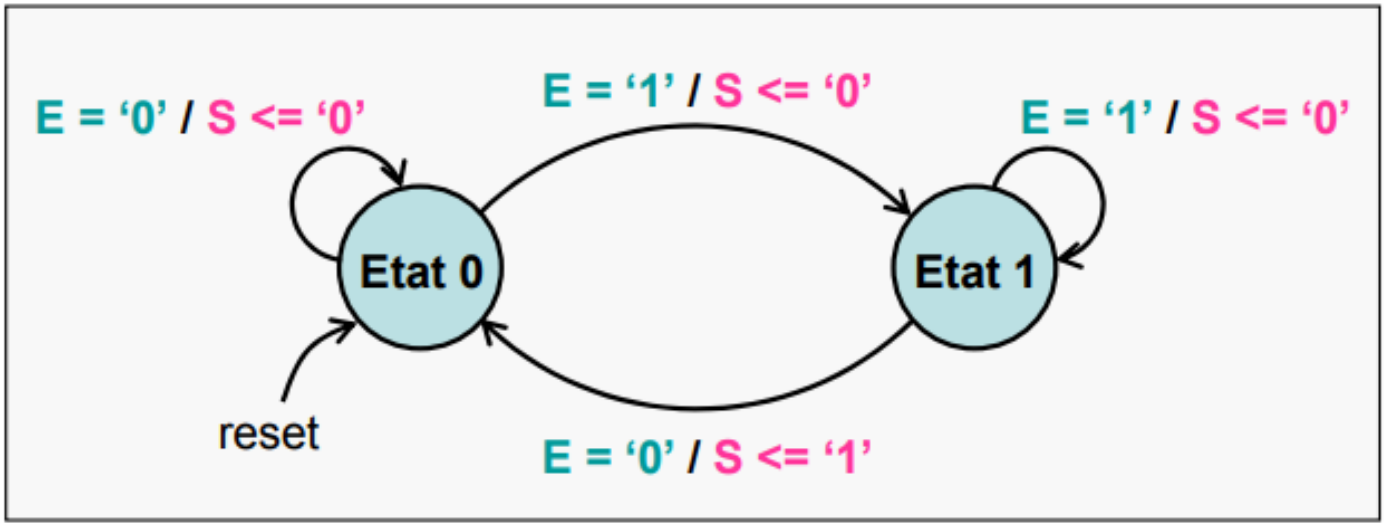
VHDL - FSM

✓ Example of Moore and Mealy FSM: detection of input toggle from 1 to 0

MOORE:

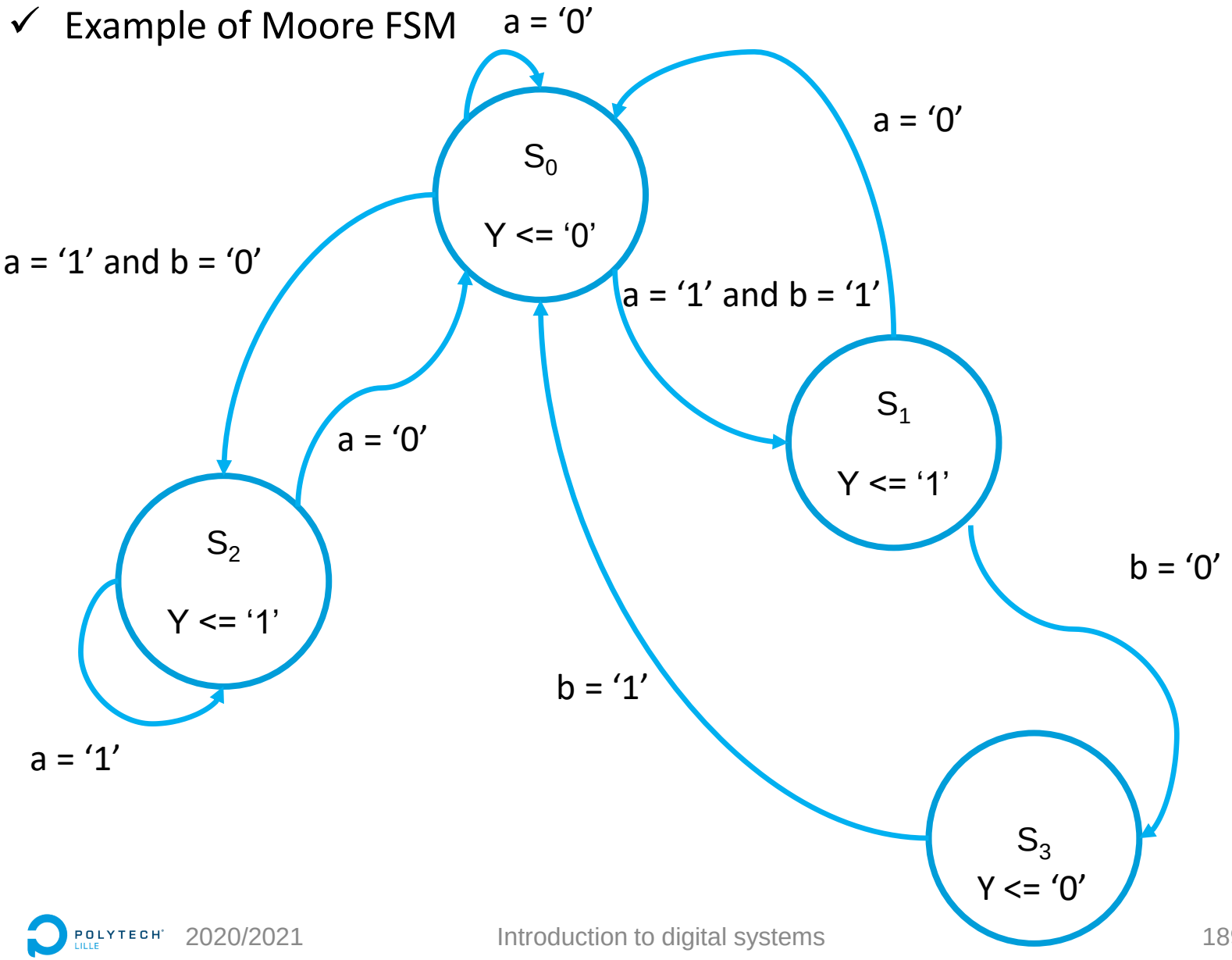


MEALY:



VHDL - FSM

✓ Example of Moore FSM



VHDL - FSM

How to implement it in VHDL?

- ✓ States are defined using enumerated type

```
type state_type is (S0, S1, S2);
signal state : state_type ;
```

- ✓ FSM description using the 'case' statement

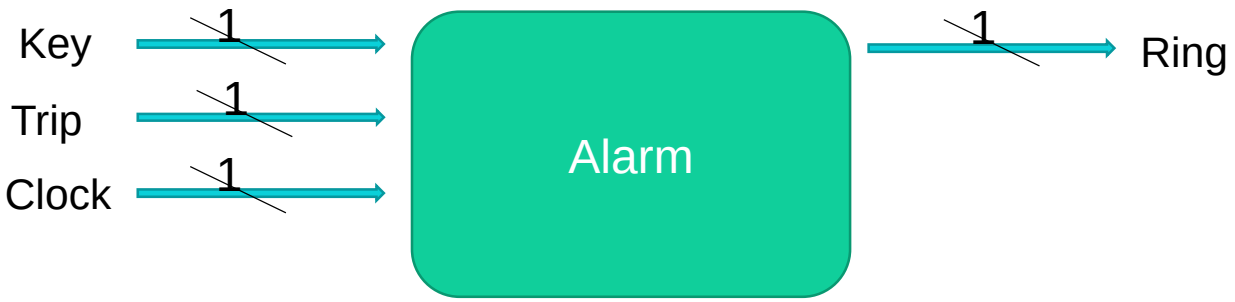
```
MyFSM : process(clock) begin
   if clock'event and clock='1' then
      case state is
      when S0 => if i1 ='1' then state <= S1;
                 else state <= S0;
                 end if;
      when S1 => if i1 ='0' then state <= S2;
                 else state <= S1;
                 end if;
      ....other states and transition conditions....
      when others => --what happen in non defined state?
      end case;
   end if;
end process;
```

VHDL - FSM

Example: Clock alarm

- **inputs:**
  - Key: activates (when ‘1’) or disables (when ‘0) the alarm function
  - Trip: Becomes ‘1’ when present time is equal to alarm time
  - clk
- **output:**
  - Ring : activate the ring

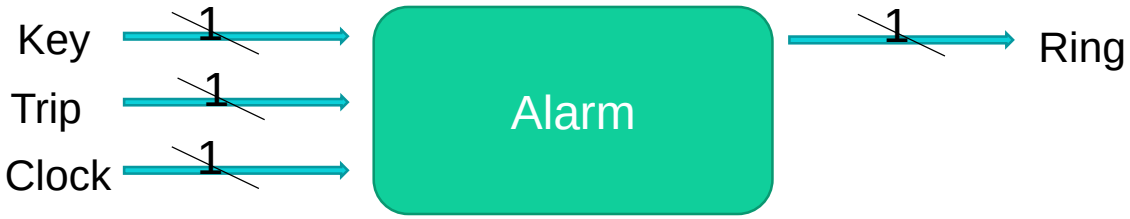
⇒ The alarm shall ring until it is disabled (i.e. until Key becomes ‘0’)



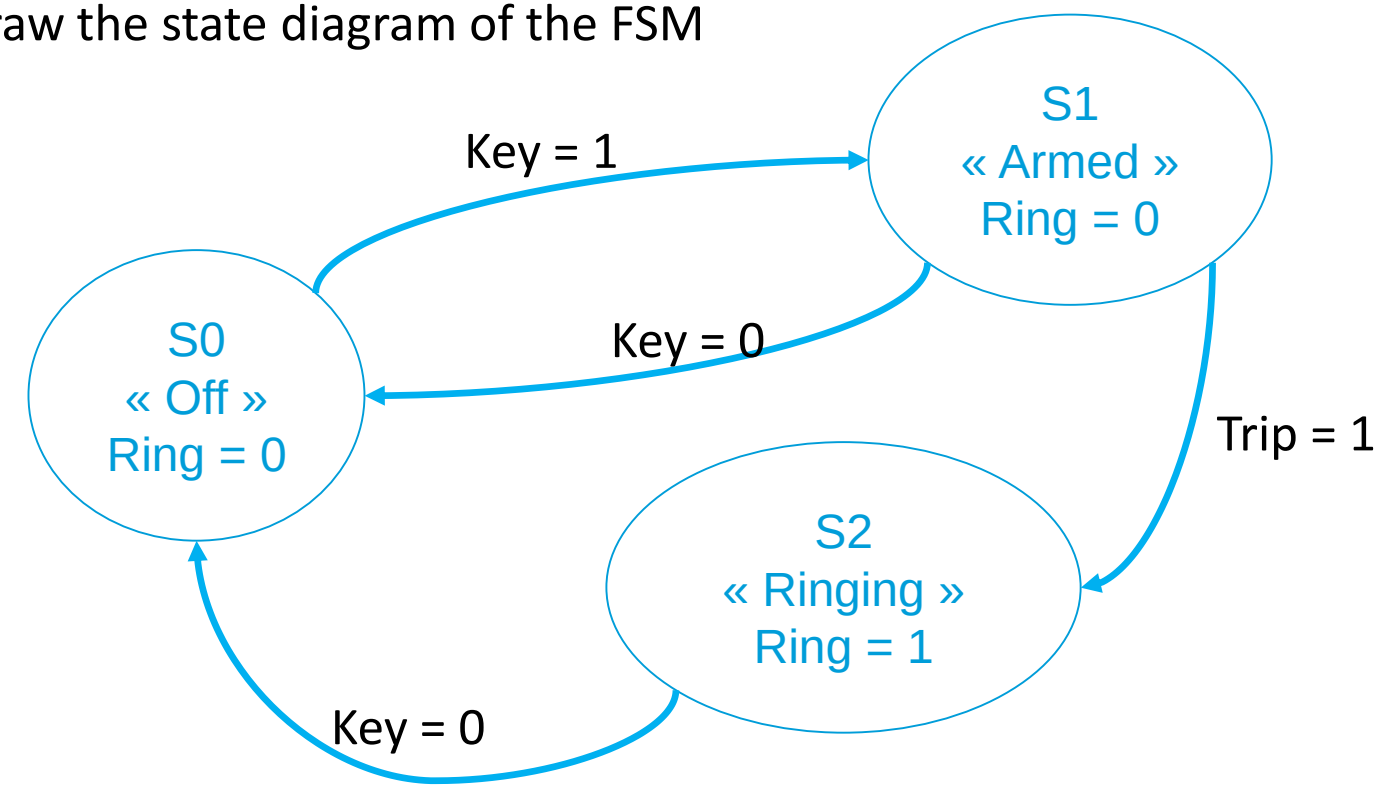
VHDL - FSM

Example: Clock alarm

⇒ The alarm shall ring until it is disabled (i.e. until Key becomes '0')



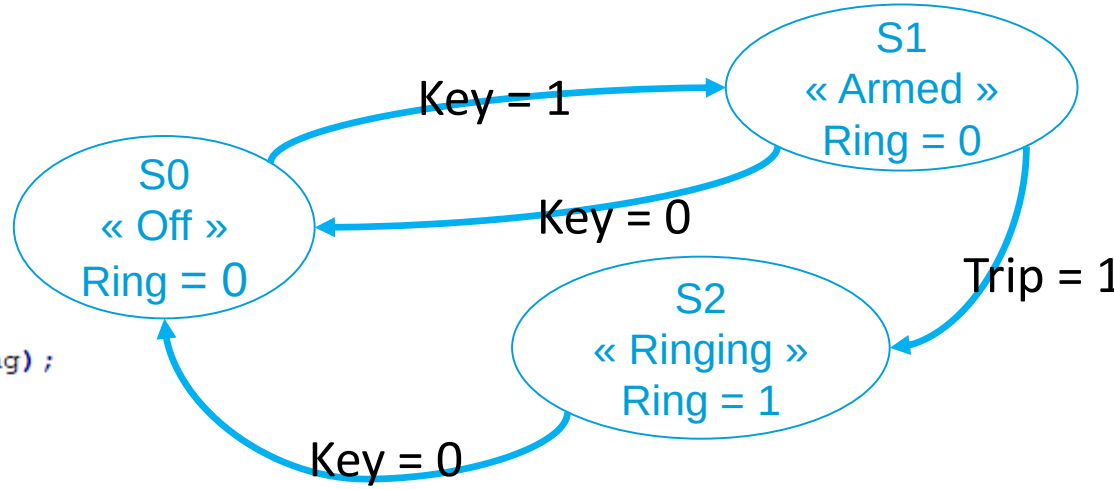
➢ Draw the state diagram of the FSM



VHDL - FSM

Example: Clock alarm

- Code this FSM in VHDL




```
ARCHITECTURE ar  OF Alarm IS

  TYPE typetat IS (Armed, Off, Ringing);
  SIGNAL etat : typetat;

  BEGIN
    -- partie séquentielle
    PROCESS (clock) BEGIN
     IF Clock ='1' AND Clock'EVENT THEN
            CASE etat IS
               WHEN Off => IF key ='1' THEN etat <= Armed;
                           ELSE etat <= Off;
                           END IF;
               WHEN Armed => IF Key = '0' THEN
                               etat <= Off;
                             ELSIF Trip ='1' THEN
                                 etat <= Ringing;
                              ELSE etat <= Armed;
                             END IF;
               WHEN Ringing => IF Key ='0' THEN
                                 etat <= Off;
                               ELSE etat <= Ringing;
                               END IF;
            END CASE;
          END IF;
     END PROCESS;

-- partie combinatoire
   Ring <= '1' WHEN etat=Ringing ELSE
           '0';
END ar;
```

VHDL - FSM

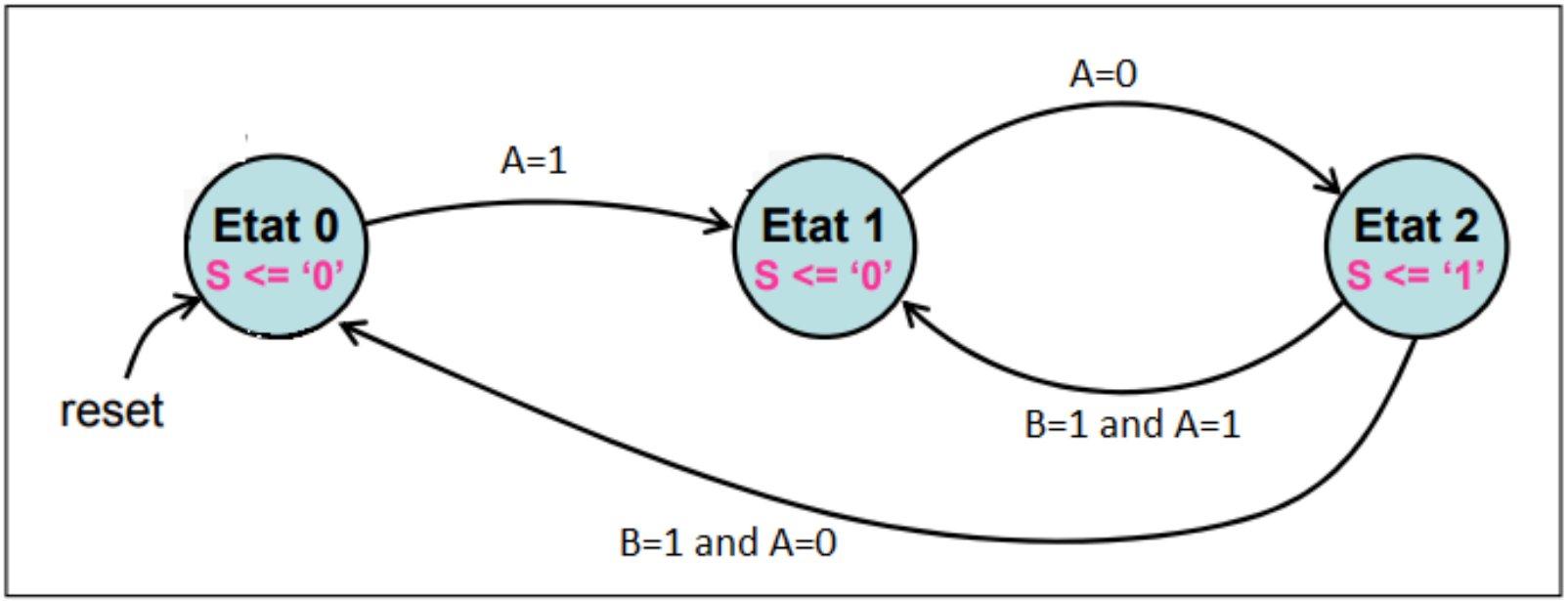
✓ Exercise 15 Consider the following functionality :

- Detection of input A toggle from 1 to 0, and then put output S to 1.
- Latch output S to 1 while input B is low.

- ➢ Draw the state diagram of the FSM
- ➢ Implement it in VHDL
- ➢ Simulate the behavior of this FSM


top.vhd


top_tb.vhd

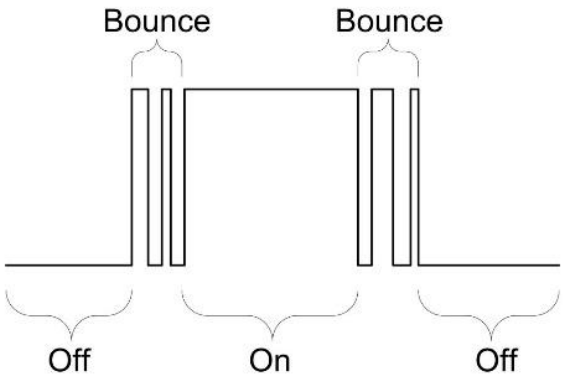


VHDL - FSM

✓ Exercise 16. Debouncing circuit

- When a mechanical button is pressed, it can have bounce, i.e. oscillations between ‘1’ and ‘0’ state before stabilization
- Bounce ~ 20 ms

- ➢ Draw a FSM to debounce the off-on and on-off changes
- ➢ Describe the FSM in VHDL
- ➢ Write a testbench for the FSM

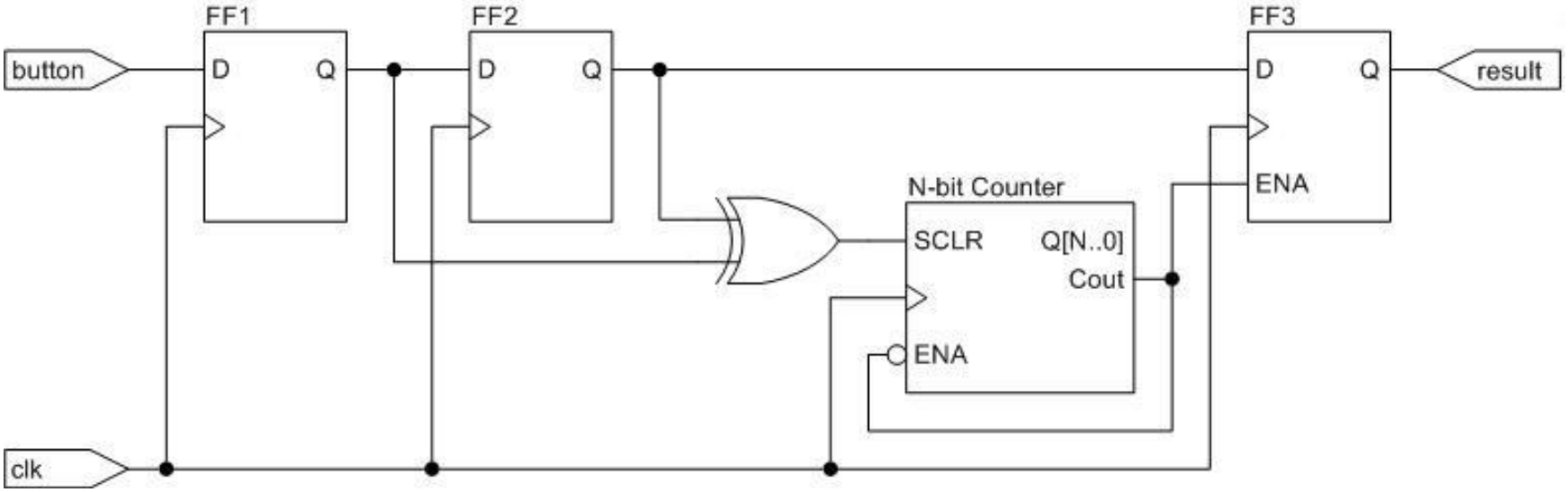



debounceFSM.vhd


debounceFSM_tb.vhd

VHDL - FSM

- ✓ Alternative solution for debouncing



https://eewiki.net/pages/viewpage.action?pageId=4980758

VHDL - FSM

- ✓ Exercise 17. Binary to BCD decoder
  - ➢ Draw a FSM to decode a 8 bit-binary code to a BCD
  - ➢ Describe the FSM in VHDL
  - ➢ Write a testbench for this FSM


binary_to_bcd_4bits.vhd


binary_to_bcd_tb.vhd


binary_to_bcd.vhd


binary_to_bcd_tb.vhd

Shift and Add-3 Algorithm

1. Shift the binary number left one bit.
2. If, after a shift, the binary value in any of the BCD columns is 5 or greater, add 3 to that value in that BCD column.
3. If less than 8 shifts have taken place, go to 1.
4. The BCD number is in the Hundreds, Tens, and Units column.

VHDL - FSM

Exercise 17. Shift and Add-3 Algorithm

1. Shift the binary number left one bit.
2. If, after a shift, the binary value in any of the BCD columns is 5 or greater, add 3 to that value in that BCD column.
3. If less than 8 shifts have taken place, go to 1.
4. The BCD number is in the Hundreds, Tens, and Units column.

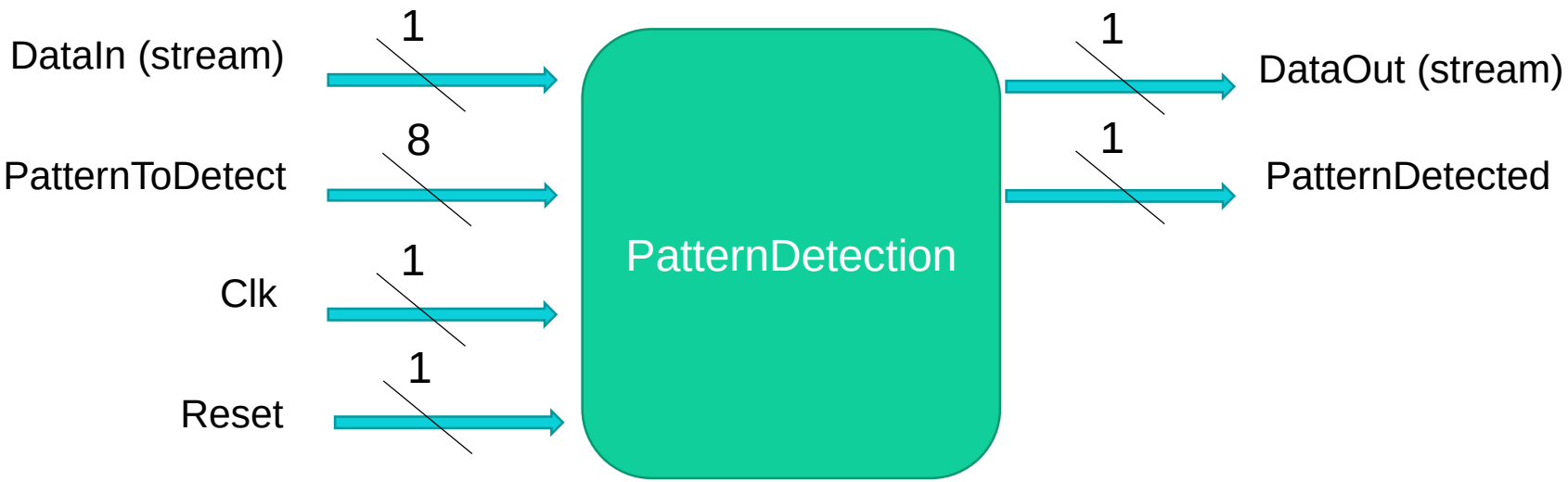
Operation	Hundreds	Tens	Units	Binary
Start				1 1 1 1 | 1 1 1 1
Shift 1			1	1 1 1 1 | 1 1 1
Shift 2			1 1	1 1 1 1 | 1 1
Shift 3			1 1 1	1 1 1 1 | 1
Add 3			1 0 1 0	1 1 1 1 | 1
Shift 4		1	0 1 0 1	1 1 1 1 |
Add 3		1	1 0 0 0	1 1 1 1 |
Shift 5		1 1	0 0 0 1	1 1 1 |
Shift 6		1 1 0	0 0 1 1	1 1 |
Add 3		1 0 0 1	0 0 1 1	1 1 |
Shift 7	1	0 0 1 0	0 1 1 1	1 |
Add 3	1	0 0 1 0	1 0 1 0	1 |
Shift 8	1 0	0 1 0 1	0 1 0 1	|
BCD	2	5	5	|

VHDL – Operations on data stream

✓ Exercise 18. Big brother is listening to your digital life!

Write a VHDL program to perform a pattern detection on a continuous flow of bits. More details on next slide…

Then, simulate it!



VHDL – Operations on data stream

✓ Ex18.

Detect in **DataIn** stream (LSB 1st) a specific pattern **PatternToDetect** (8 bits)

Transmit the flow DataIn on DataOut output, synchronous with the **PatternDetected** output:

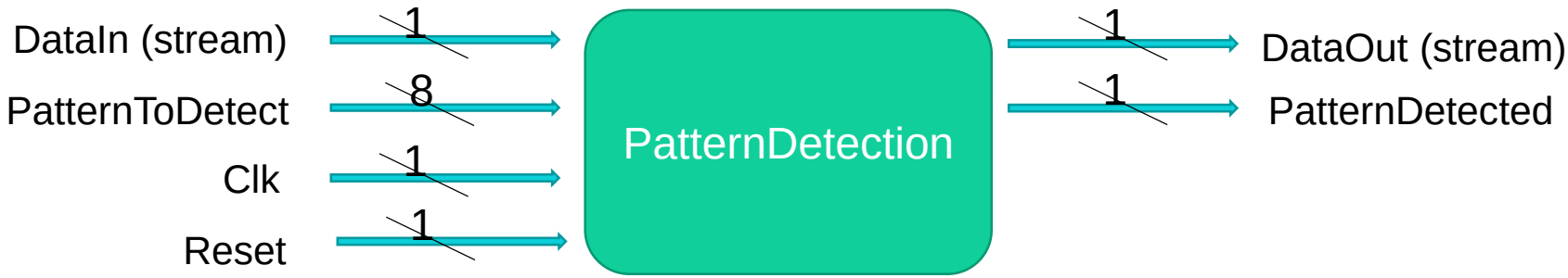
- PatternDetected stays Low while the LSB of the detected pattern is not on DataOut
- PatternDetected is High as soon as the LSB of the detected pattern outcomes on DataOut
- PatternDetected returns low just after the MSB goes out from DataOut

Implementation in 2 processes:

- One process to buffer the incoming stream DataIn
- One process to detect the pattern

Synchronous reset

Maximum latency: 9 clk



FPGA – Conclusion

- ✓ Choose a FPGA for application ?

- ✓ Choose the right architecture ?
- ✓ Choose the right family / device ?
- ✓ IP integration ?
- ✓ Soft or Hard processor core ?
- ✓ VHDL design ?

Bibliography

- ✓ The design Warrior's Guide to FPGAs, Clive « Max » Maxfield, Elsevier
- ✓ Digital Design principles & practices, John F. Wakerly, Prentice Hall
- ✓ FPGA Prototyping by VHDL Examples, Pong P. Chu, John Wiley & Sons
- ✓ Systèmes électroniques numériques complexes, A. Nketsa et D. Delauzin, Ellipses.
- ✓ Techniques de l'ingénieur, n° E2468v2, E2455, E2452, E2461, E2460, H8000, E3565, E2492, H1196.
- ✓ Introductions aux circuits FPGA, A. Tisserand, ENS Lyon.
- ✓ Doc. Xilinx : www.xilinx.com, Xcell journal.
- ✓ www.altera.com