4th COMCHA School

Zaragoza, 8 – 15 April 2026

FPGA programming

Álvaro Navarro Tobar (Ciemat)

Proyecto PID2023-148896NB-I00 financiado por:

MINISTERIO DE CIENCIA, INNOVACIÓN Y UNIVERSIDADES

Cofinanciado por la Unión Europea

AGENCIA ESTATAL DE INVESTIGACIÓN

FPGA:
what it is and what it is not

A zoo-logic metaphor

What is an FPGA?

FPGA: **F**ield-**P**rogrammable **G**ate **A**rray

- Array of Logic Gates: AND, OR, NOT…
- Field-programmable: its behavior can be modified after it leaves the factory

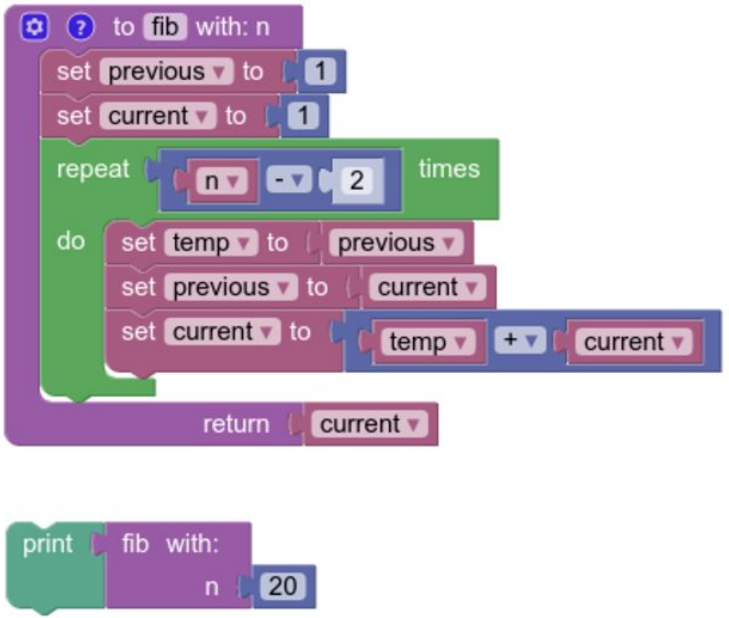
...???

It is a programmable logic device whose architecture is radically different from that of a traditional processor

Let’s continue…

What is an FPGA not?

A linear or sequential program: a sequence of instructions executed one after another on a processor. It may include loops (for) or conditional branches (if), but there is always a “pointer” that marks which line of the program is being executed at any given moment.



What is an FPGA not?

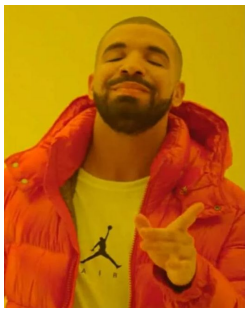
Arduino

Multi-core CPU

GPU

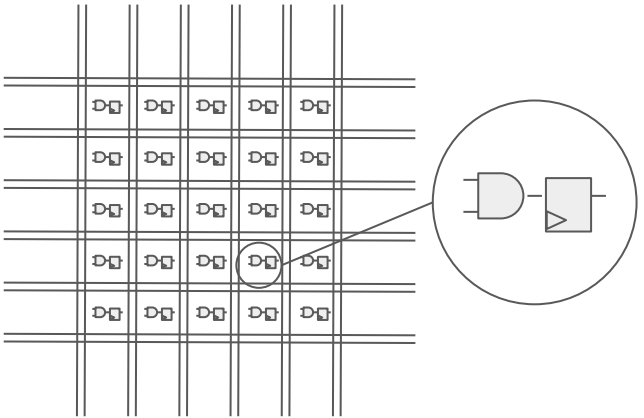
What is an FPGA not?

- An array of configurable elementary digital components (LUTs, FFs)
- With interconnection resources that are also configurable
- And with other application-specific resources that are also configurable (RAM, DSP, IO blocks, Gigabit Transceivers, clocking resources)



It is a toolbox full of hardware resources that can be configured to perform logical tasks

- Lego® box
- Assembly line



And you, which one are you more like?

To program FPGAs successfully, it is not enough to (just) learn a new programming language. The key is to internalize an architecture that is different from the one we are used to.

Note 1: GPUs (graphics cards)

The leap in parallelism between a CPU and a GPU is poorly represented by the animal metaphor (metaphors are powerful, but limited).

Moving from a processor to a GPU requires a major mindset shift, in the same direction as moving to an FPGA: unfolding sequential loops across multiple processors that run in parallel.

Each one has use cases in which it excels.

AMD: Ryzen, Radeon, Xilinx

Note 2: ASIC

There is customization beyond FPGAs

ASIC: **A**pplication-**S**pecific **I**ntegrated **C**ircuit

Has the universe not yet produced a beast that perfectly suits your needs? You can create it yourself!

A horse that flies? A venomous, duck-billed mammal that swims and lays eggs? No problem!

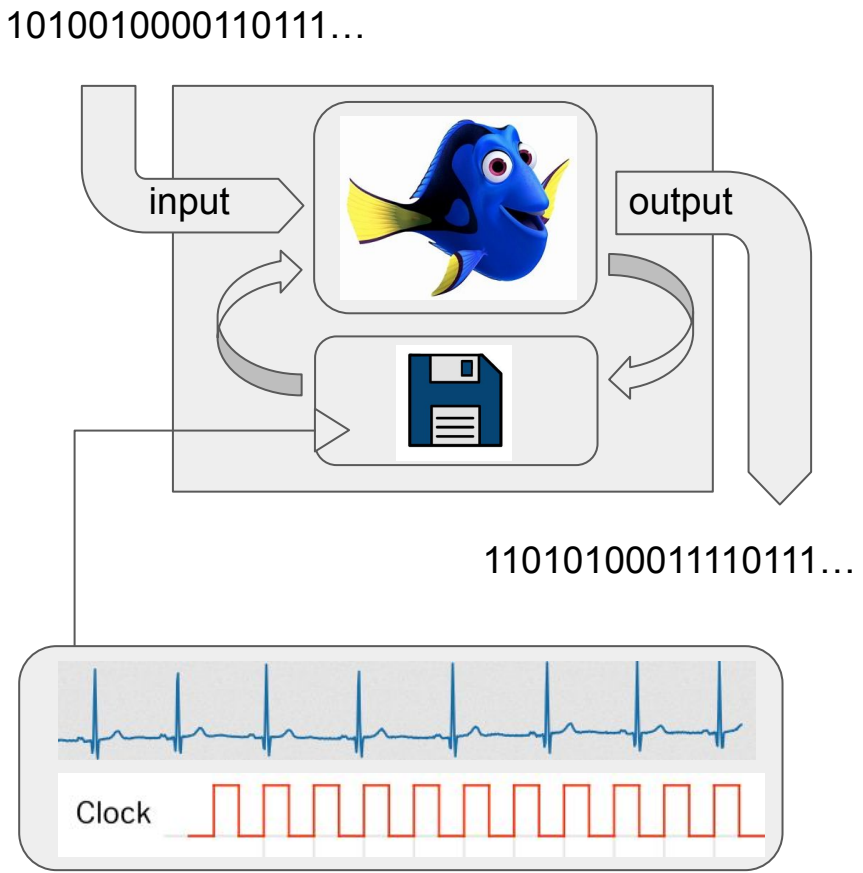
But beware… the greater the customization, the greater the development cost and risk

Fundamentals of Digital Logic

From digital logic to FPGAs

Digital Systems 101

- Digital data: 0s and 1s
- Inputs and outputs: the outputs depend on the inputs
- **Combinational logic** → only on the current inputs
  - There is no memory/storage
- **Sequential logic** → also on past inputs
  - They store a "**state**" that contains the relevant information from the past
- A change of state does not occur at any arbitrary moment → it is controlled by the **clock** signal
  - The clock triggers information storage on its rising edges (change 0 → 1)

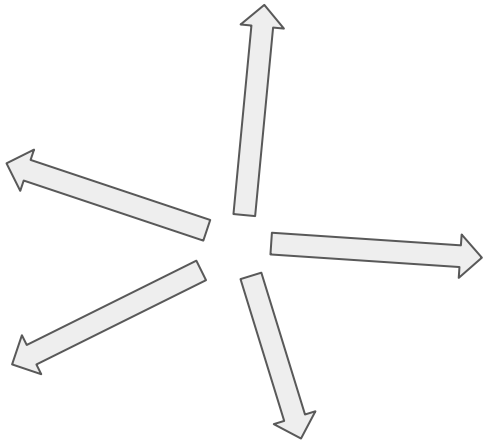


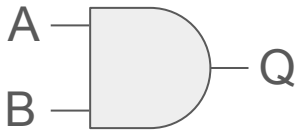
Look-Up Tables (LUT)

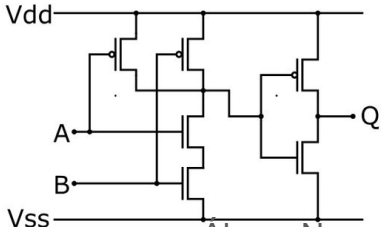
Quentin will come **if** **A**lice **and** **B**ob come

$Q = A \wedge B$

Q <= A and B;

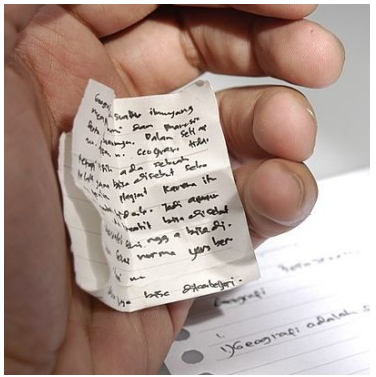






Look-Up Table

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1



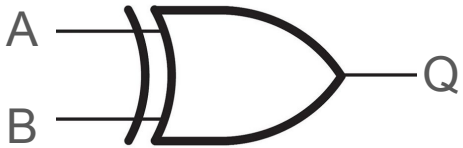
It is a list with the answers to all possible questions

It can represent any logical function

The number of rows is $2^{\text{number of inputs}}$

In FPGAs, typically 4–6 inputs

Exercise: XOR gate truth table

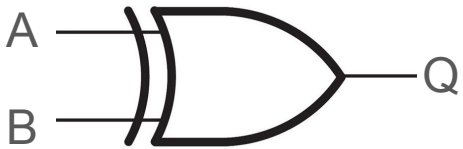


A	B	Q
0	0	
0	1	
1	0	
1	1	

Quentin wants to go to a party, but he cannot drive and does not have a vehicle. Alicia and Bob, who are a couple, can drive and have a motorbike. Quentin could go if Alicia or Bob goes to the party, but not if both go, because all three cannot ride on the motorbike.

Quentin wil come **if only** **A**lice **or** **B**ob comes

Exercise: XOR gate truth table

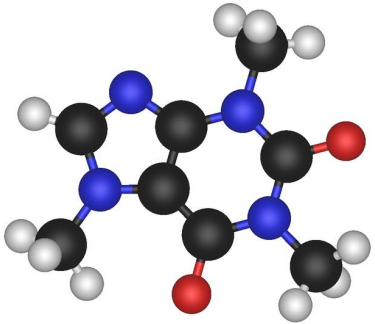


A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

Quentin wants to go to a party, but he cannot drive and does not have a vehicle. Alicia and Bob, who are a couple, can drive and have a motorbike. Quentin could go if Alicia or Bob goes to the party, but not if both go, because all three cannot ride on the motorbike.

Quentin wil come **if only** **A**lice **or** **B**ob comes

Look-Up Tables (LUT)



A and B

A xor B

A	B	Q_1	Q_0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

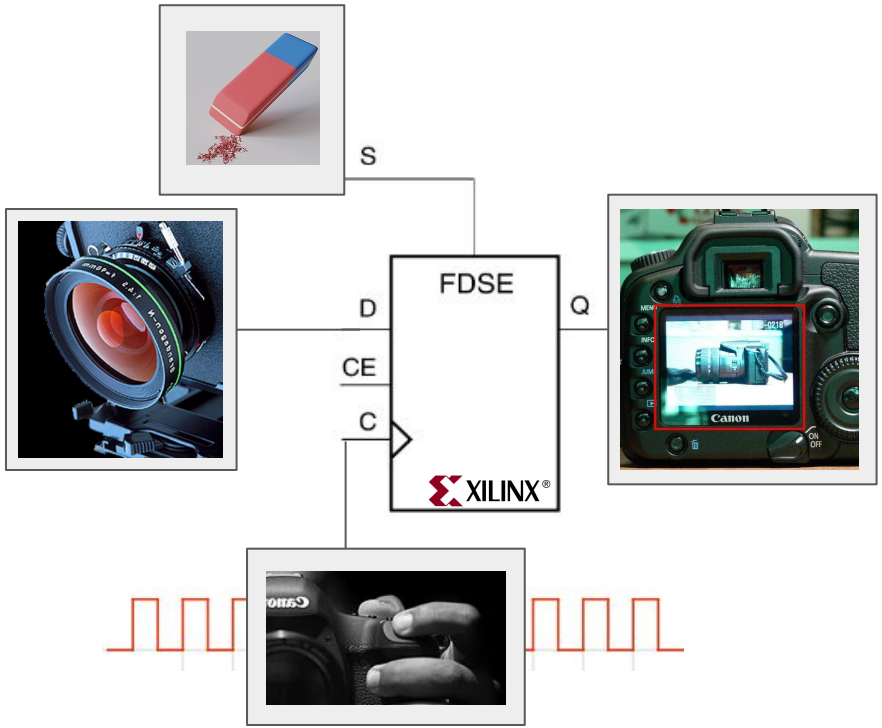
A + B

- By combining LUTs, more complex logical functions can be built (~atoms → molecules)
- Not only logic, but also arithmetic functions

  Example: $(Q_1Q_0) <= A + B$

- LUTs are defined by the code ⇒ they are fixed when the firmware is loaded, and they do not change during operation
  - You cannot turn an AND into an OR, but… you can program a LUT that behaves as an AND or as an OR depending on a third input
- The outputs are determined by the current value of the inputs… **after a small propagation delay**
- **Some manufacturers have LUTs with special functions **

Flip-Flops (FFs)



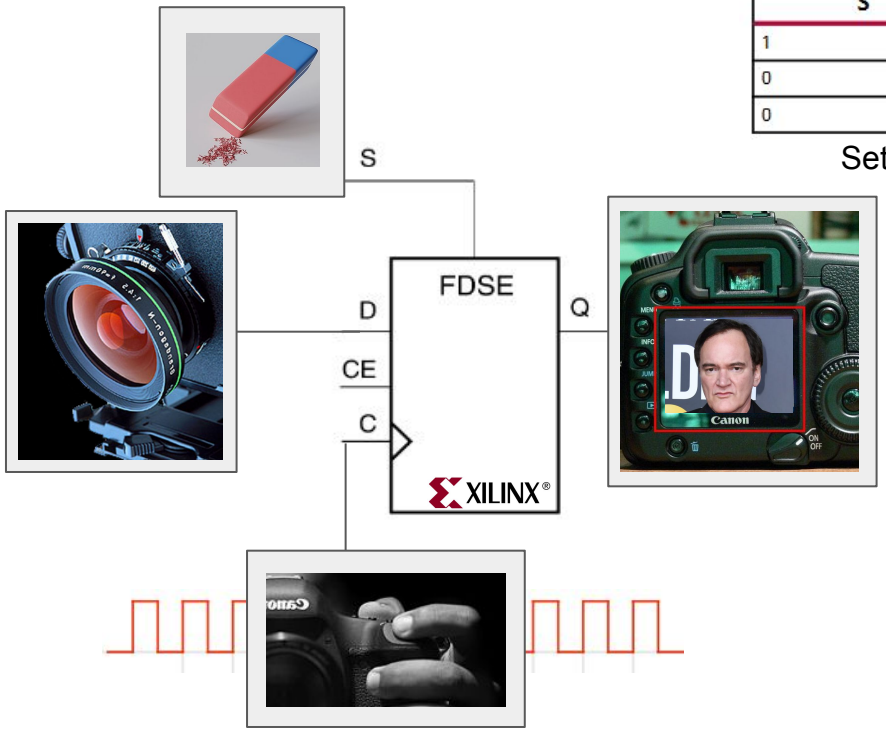
- It stores 1 bit of information received at its data input (D) and keeps it at its output (Q), updating its value only when a rising edge occurs at its clock input (C).
  - A set/reset port (S) can “clear” the information, restoring a known value
  - The “clock enable” (CE) signal can temporarily disable the storage of new information
- This is what is called a D-type Flip-Flop (D → data)
- If you come from electronics, you will know other types of FF (JK, SR, T). In FPGAs, FFs are D-type

Flip-Flops (FFs)

Logic Table

Inputs				Outputs
S	CE	D	C	Q
1	X	X	↑	1
0	0	X	X	No Change
0	1	D	↑	D

Set Clock Enable Data Clock



They have a truth table too!

The memory elements in an FPGA can be configured as 4 different types of FF, depending on whether they have a set or a reset (store 1 or 0) and whether it operates synchronously or asynchronously (whether it needs a clock edge or not to take effect)

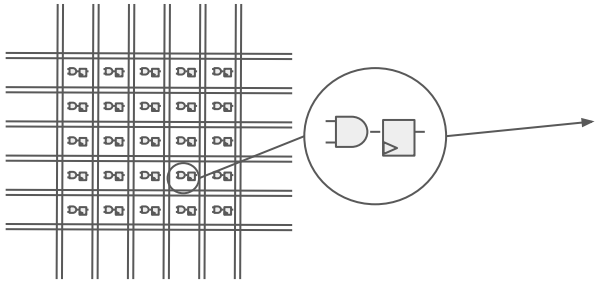
Each LUT y FF is continuously switching!

Data pass through different processing stages, but when they move from one stage to the next, that does not mean the first one pauses and the second one starts; the first one simply starts working on new data

Is it possible to “pause” part of the FPGA? More or less → using Clock Enables, Resets, or other techniques

Elements of an FPGA

Configurable Logic Block → FPGA bricks



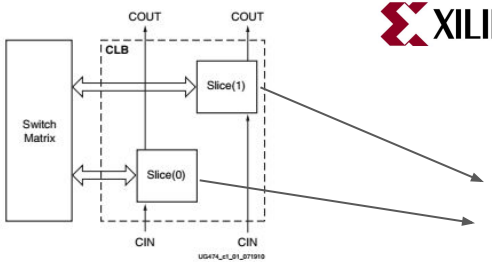


Figure 1-1: **Arrangement of Slices within the CLB**

XILINX®

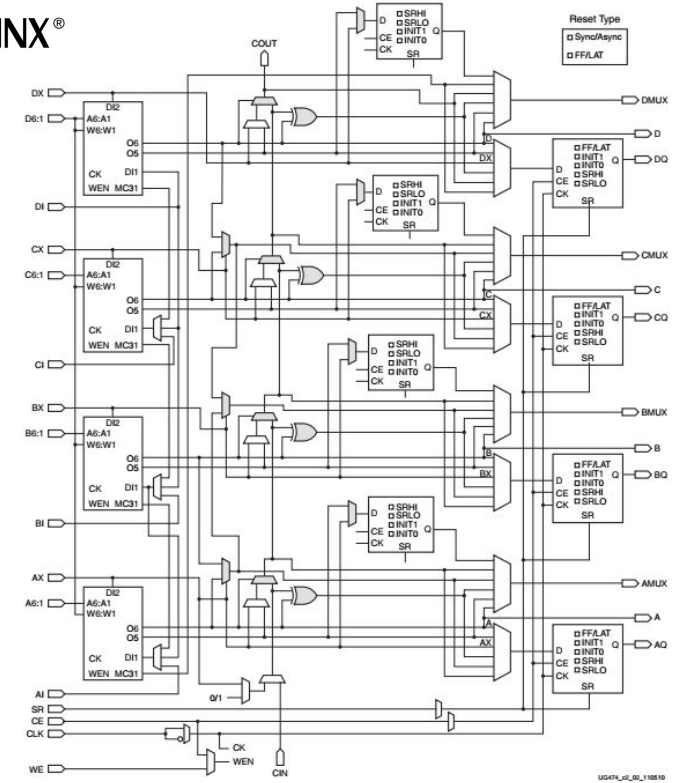
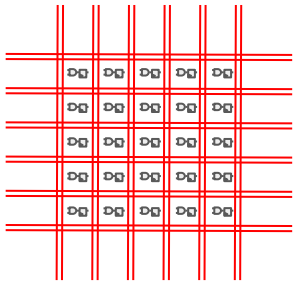
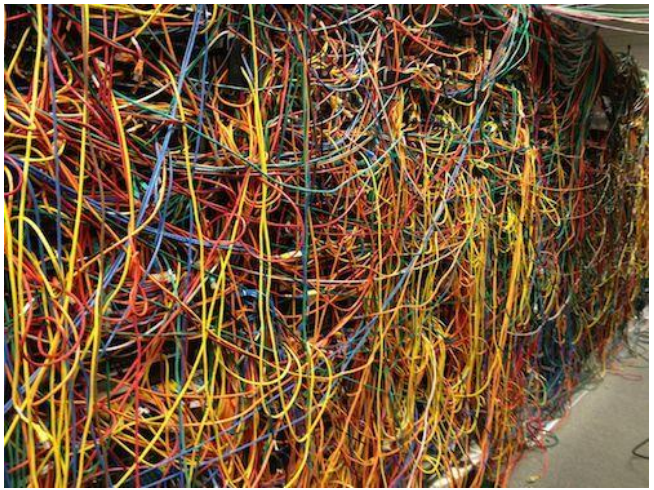


Figure 2-3: **Diagram of SLICEM**

- The basic element that is repeated over and over to provide the bulk of the logic resources
- It contains LUTs and FFs connected in a versatile way
- Multiplexing resources
- Carry logic for arithmetic

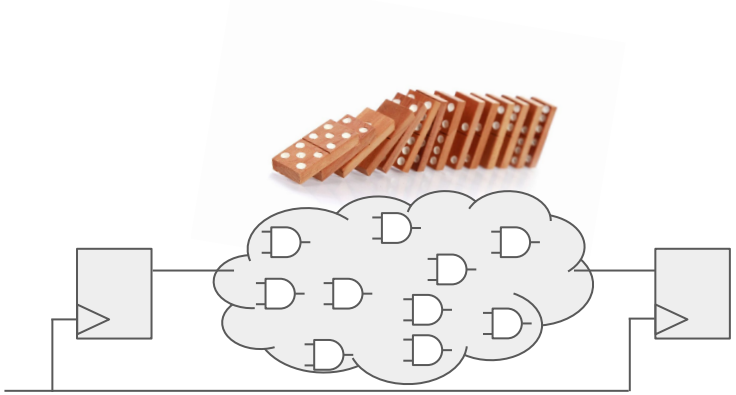
Routing resources



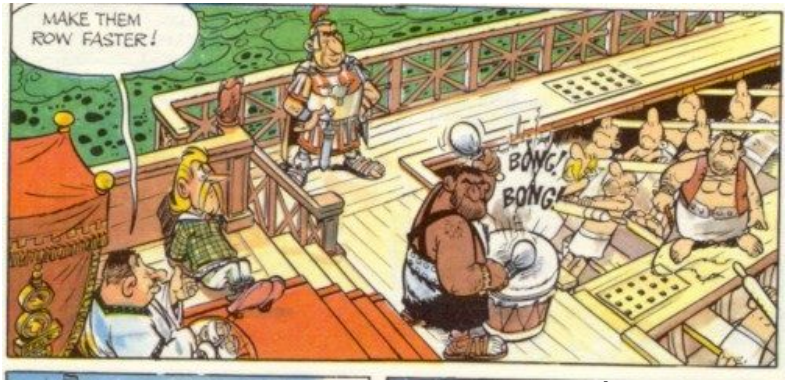


- They allow the inputs and outputs of the different logic resources to be connected together
- There are special high-speed lines for routing clocks or critical signals (high fan-out, resets…)
- The wiring is completely determined during programming
- It is abundant but not unlimited ⇒ congestion problems
- The transmission of information is not instantaneous (duh!): it has a delay (on the order of picoseconds or a few nanoseconds)
- The importance of *placement*

Clocks and timing



- When an FF changes its value, the change propagates through the combinational logic (LUTs and routing)
- The propagation of the change must settle at the destination FF input before the next clock edge
- This check is performed automatically by the design tool: “timing verification/closure”
- The “compiler” can make “efforts” to close timing, but ultimately it depends on the code



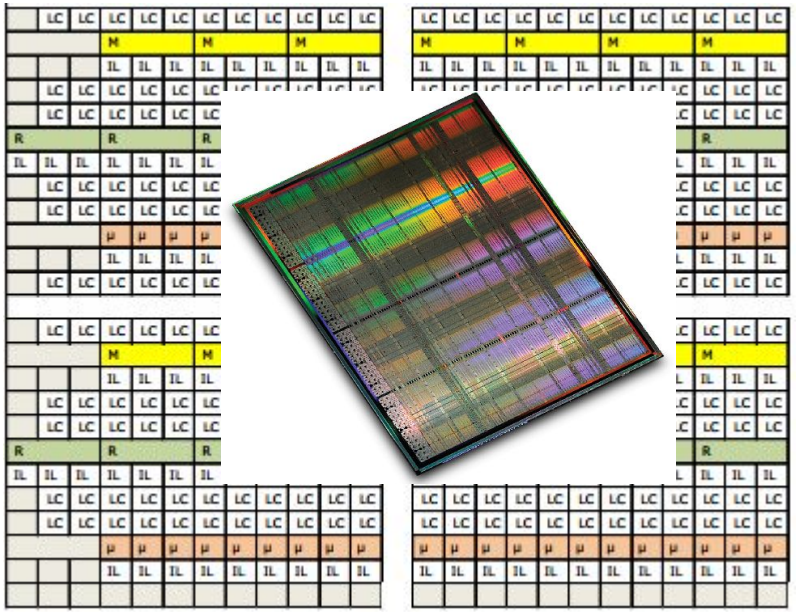
Design Timing Summary

Setup

Worst Negative Slack (WNS):	0.004 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	256

All user specified timing constraints are met.

RAM & DSP

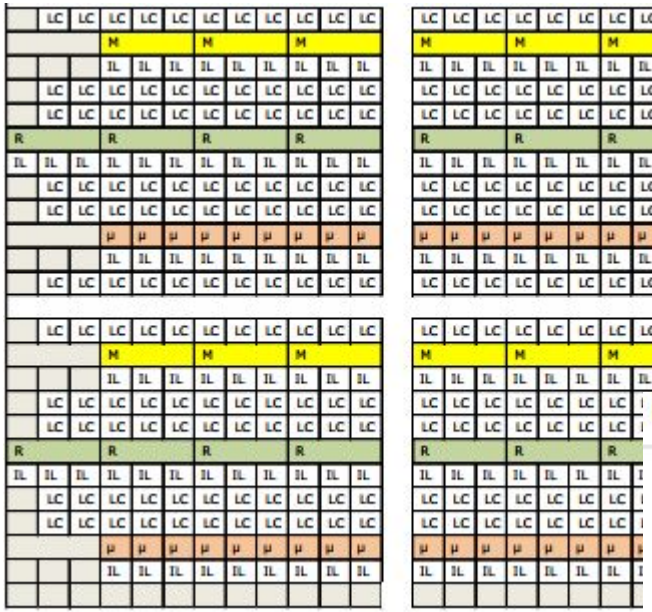


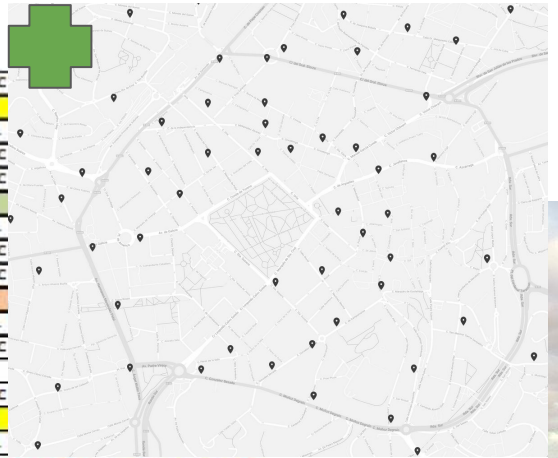
Some tasks are very common across many applications, and they are implemented inefficiently using the FPGA logic described above (LUTs+FFs).

Manufacturers add blocks specialized for these tasks and distribute them throughout the FPGA fabric

- RAM: Random Access Memory
  - Data storage; more integrated capacity than FFs (e.g., Spartan7-50: 2.7 Mb RAM vs 700 Kb LUTs+FFs).
  - Low-latency access (vs. external RAM)
- DSP: Digital Signal Processor
  - Integer multiplication and addition
  - (25bits)x(18bits) → ~500 LUTs, 10 depth levels, vs. 1 DSP

Why scattered?





Map of pharmacies in Oviedo

Design Timing Summary

Setup

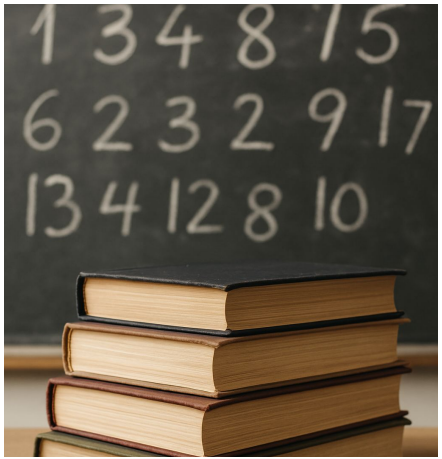
Worst Negative Slack (WNS):	0.004 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	256

All user specified timing constraints are met.

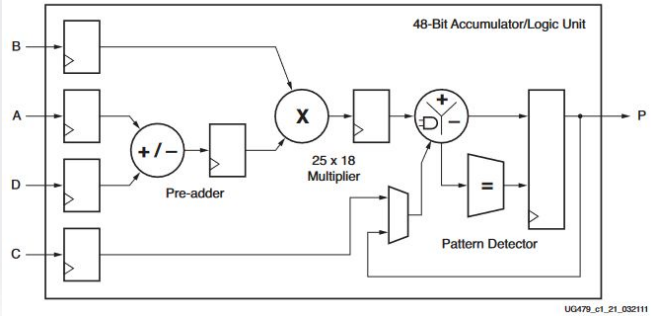
When **time** is **critical**…

You do not want to waste time on route

RAM



DSP



- Blocks dedicated to the “massive” storage of data, while adapting to the flexibility required in FPGAs
- RAM: large “lists” of values that can be written and read very quickly.

- DSP is a hardware block dedicated specifically to performing arithmetic operations quickly and efficiently
- It can perform multiple additions, subtractions, multiplications, and comparisons against a pattern

What about divisions?

- Division in an FPGA is not a basic operation: it is expensive and slow
- Before dividing, ask yourself: do I really have to divide, or can I reformulate the problem?
  - Examples: DT, AM, t ~ x, slope
- There is no dedicated HW and the synthesizer does not generate optimal code
- Divisions by constants:
  - Power of 2: free
  - Others: multiplication by the reciprocal
- Division by signal: expensive, try to avoid it

Extreme connectivity (= many input/output pins)

Thanks to its internal architecture, an FPGA can handle abundant external connectivity. Its resources can be organized to serve a high number of input/output streams or channels.
(you can very easily highlight the number of ants needed at each anthill entrance)

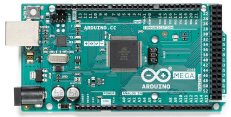
General Purpose I/O (GPIO)

- General purpose: a wide variety of electrical standards (voltage levels, impedance, differential/single-ended…), versatile
- Available in large numbers
  - Spartan7-50: 100–250 pins
  - Virtex UltraScale Plus 19P: 2000 pins
- Relatively fast (~1 Gbps)

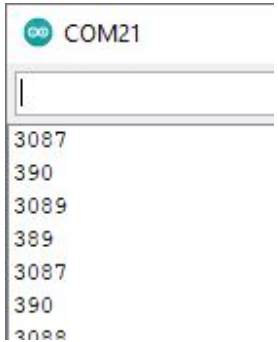
Gigabit transceivers

- For serial digital communications (PCIe, optical fiber, …)
- SerDes: serializer-deserializer
- Available in low-to-medium numbers
  - Spartan7/Artix7: 0/16 links max
  - Virtex UltraScale Plus 13P: 128 links
- Very fast (~10–40 Gbps)

Face to face: CPU vs FPGA

Processor  …vs… FPGA

```
unsigned short datain;
unsigned short dataout;
byte counters[16];

unsigned long serialtimer = 0;
unsigned long t0 = 0;
bool do_main_processing = true;

void setup() {
  Serial.begin(9600);
  DDRF = B00000000;
  DDRK = B00000000;
  DDRA = B11111111;
  DDRC = B11111111;
  for (int i= 0; i<16; i++) counters[i] = 128;
}

void loop(){
  if(do_main_processing) {
    datain = (PINF << 8) + PINK;
    dataout = 0;
    for (int i= 0; i<16; i++) {
      if (datain&1) { if(counters[i]!=255) counters[i]++; }
      else          { if(counters[i]!=0  ) counters[i]--; }
      if(counters[i]>=128) dataout |= (1<<i);
      datain = datain >> 1;
    }
    PORTA = (byte) (dataout & 0xFF);
    dataout = dataout >> 8;
    PORTC = (byte) (dataout & 0xFF);
  }

  if(serialtimer) serialtimer--;
  else{
    Serial.println(millis()-t0);
    serialtimer = 100000;
    do_main_processing = not do_main_processing;
    t0 = millis();
  }
}
```



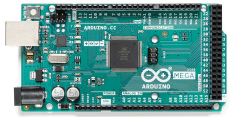
Task

- According to the value of each of the 16 input pins
  → increment/decrement the corresponding counters
  → according to the value of each counter
  → the 16 output pins are 1 or 0
- Task chosen so that the FPGA wins 🤷
  - **Other tasks are better suited to a CPU or GPU**

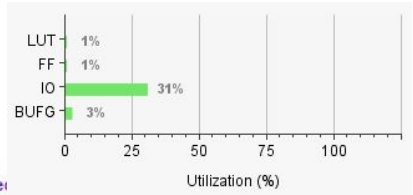
Arduino Mega 2650

- AVR ATmega2560, ~ 15 € → 8 bit, 16 MHz
- f_{max} **= 37 kHz**
  - 100% CPU use
- There are many low-cost processors with a higher clock frequency (~500 MHz) → f_{max} **≃ 1 MHz**

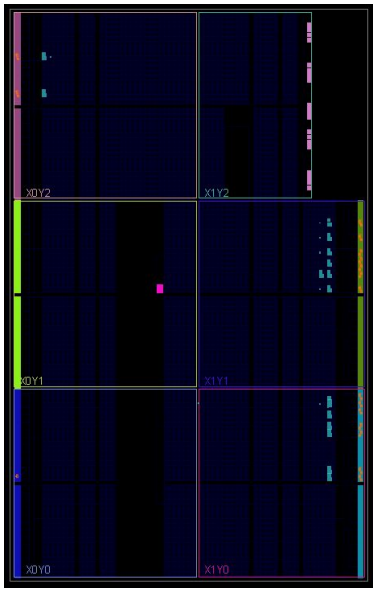
Processor



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity top is
  Port (
    clk     : in  std_logic;
    datai   : in  std_logic_ve
    datao   : out std_logic_vector(15 downto 0)
  );
end top;

architecture Behavioral of top is
begin
  chans_gen: for i in 0 to 15 generate
    signal counter : unsigned(7 downto 0) := to_unsigned(128,8);
  begin
    datao(i) <= counter(7);

    process(clk)
    begin
      if rising_edge(clk) then
        if    datai(i) = '1' and counter /= to_unsigned(255,8) then
          counter <= counter + 1;
        elsif datai(i) = '0' and counter /= to_unsigned(0,8) then
          counter <= counter - 1;
        end if;
      end if;
    end process;

  end generate;
end Behavioral;
```



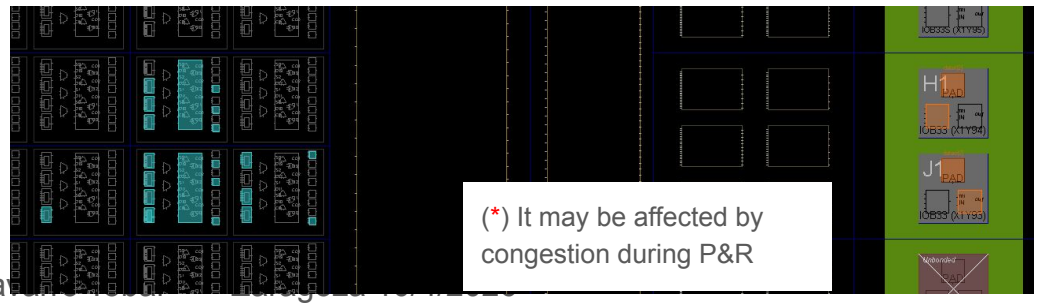
…vs…



FPGA

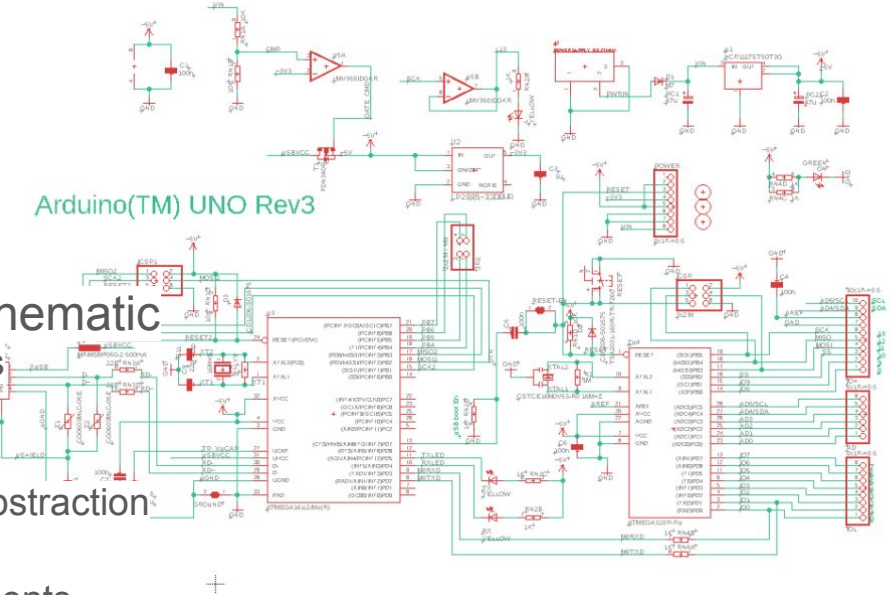
Digilent CMOD-A7

- Artix7-35T, ~40€
- f_{max} **= 325 MHz**
  - Slack = 4 ps (0.1% period)
  - Most of the silicon unused
- It could increase to ~600 MHz using DSPs
- Adding more tasks does not harm the existing ones (*) → real time



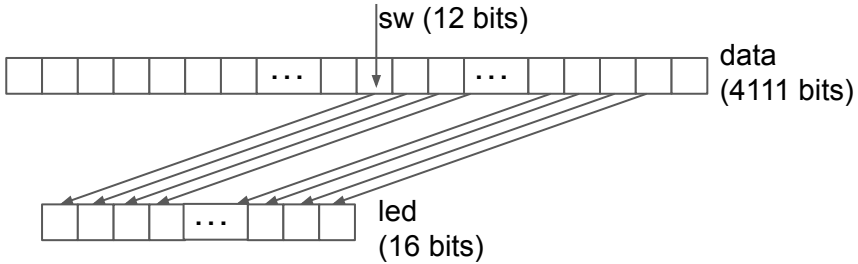
(*) It may be affected by congestion during P&R

How is an FPGA programmed?



- Non-linear/non-sequential code
  - It requires a mindset shift
- Sometimes it looks more like a PCB schematic
  - It defines modules and their interconnections
- HDL: Hardware Description Language
  - Traditionally VHDL and Verilog
  - Both allow you to work at several levels of abstraction
- Three steps
  - Synthesis: HDL → interconnected logic elements
  - Implementation (Place & Route): assignment of the logic elements to specific positions in the FPGA, and of the routing resources to connect them
  - Binary generation (bitfile)
- HLS: High-Level Synthesis
  - Before synthesis, a higher abstraction level: C → HDL
  - It makes some tasks that are tedious in HDL easier
  - It does not let you program an FPGA as if it were a processor. It will not save you from having to adapt your mental models to FPGA architecture (that is the fun part!).

It is necessary to change the way of thinking…



Problem to solve: from an input vector of ~4000 bits, extract 16 consecutive bits starting from a provided position

Approaches:

1. Direct:
   a. 16 multiplexers 4096:1
   b. 8500 LUTs (**26%** Spartan7-50)
2. "Zoom" in two steps
   a. 79 multiplexers 64:1
   b. 16 multiplexers 64:1
   c. 1400 LUTs (**4.5%** Spartan7-50)

÷6!!

```
led <= data(to_integer(sw)+15 downto to_integer(sw));
```

```
msb <= sw(11 downto 6);
lsb <= sw( 5 downto 0);
zoom <= data(to_integer(msb)*64 + 78 downto to_integer(msb)*64);
led  <= zoom(to_integer(lsb)    + 15 downto to_integer(lsb)    );
```

Concurrency

```
ctr <= current_ctr;

current_ctr <= next_ctr when rising_edge(clk);

next_ctr <= current_ctr + 1 when incr = '1' else
            current_ctr - 1 when decr = '1' else
            current_ctr;
```

Do you agree?

- The order in which the statements are written seems counterintuitive; it does not follow the logical reasoning process
- In linear or sequential programming this would not work: you have to update the values before using them
- In an architecture, all (top-level) statements are concurrent: they happen at the same time; you simply write some before others because writing them all on the same line would not be very readable

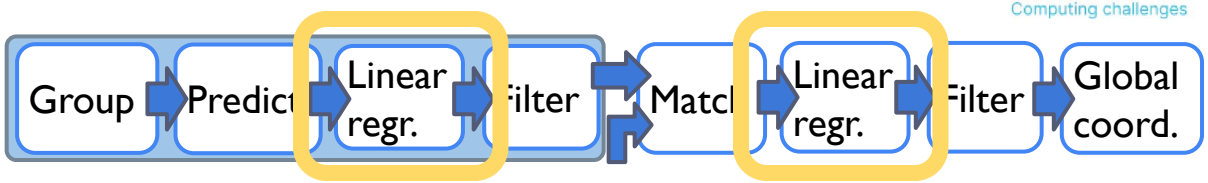
Parallelism: resource-latency tradeoff

- In an FPGA, the resource-latency tradeoff is much more evident than in a CPU. When you have a CPU, doing more operations usually just means spending more time, until that is not enough and you buy a new CPU. In an FPGA, you can constantly choose between reusing resources and spending time, or instantiating more resources.
- For example, suppose we already have a 32-number adder capable of 400 MHz throughput. What should we do if we are asked to add 128 numbers at 100 MHz?
  - The sensible thing is to run our adder 4 times and accumulate the result
- As in life, you choose… what can you afford to spend: time or resources?

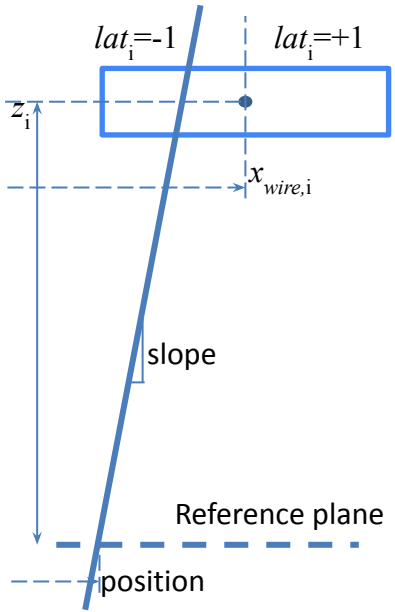
Algorithms vs FPGAs

Compute modules vs data flow

Linear regression



$$y \equiv \begin{bmatrix} x_{wire,1} + lat_1 \cdot t_1 \\ x_{wire,2} + lat_2 \cdot t_2 \\ \cdots \\ x_{wire,8} + lat_8 \cdot t_8 \end{bmatrix}, X \equiv \begin{bmatrix} lat_1 & 1 & z_1 \\ lat_2 & 1 & z_2 \\ & \cdots & \\ lat_8 & 1 & z_8 \end{bmatrix}, b \equiv \begin{bmatrix} t_0 \\ pos \\ slope \end{bmatrix}$$



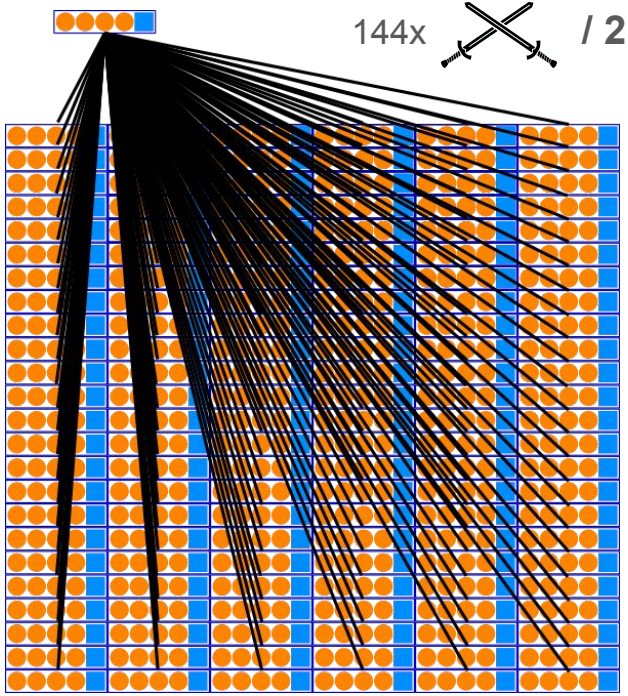
$$y = X \cdot b + \epsilon$$
$$b = (X^T \cdot X)^{-1} \cdot X^T \cdot y$$
$$C \equiv (X^T \cdot X)^{-1} \cdot X^T$$
$$b = C \cdot y$$

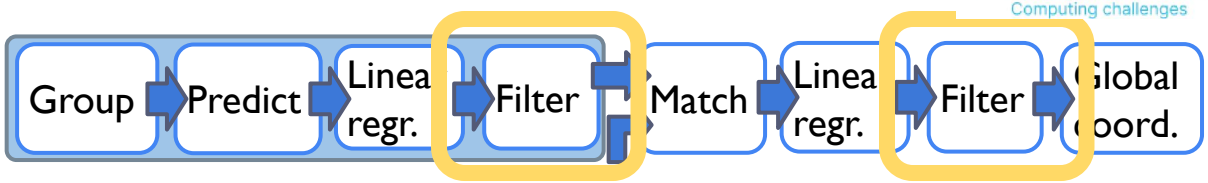
ROM LUT DSP

- Inputs: horizontal position of the wires, TDC value, laterality hypothesis
- Outputs: track parameters (T0, position, slope) and chi squared
- Ordinary Least Squares, matrix solution:
  - C matrix super expensive computationally
  - y matrix: much simpler, unavoidable (hit timestamps)
- We compute C matrix offline, store it in ROM
  - For 1 SL, ~50 rows, 200 bits each → distributed RAM
  - For 2 SL, ~1500 rows, 360 bits each → 20 Block RAM (UltraRAM doesn't allow ROM initialization)
- Computational load reduced to bare minimum
- Latency: 15 clock cycles
- 8 layers: 3k LUT, 4.5k FFs, 20 BRAM, 25 DSP

Filtering

1 segment
= 35 bits (*)





- Hits must be used once, in the best-quality segment
- Arriving segments are compared with 144 previous segments (24-BX deep history x 6 segments width):
  - May duel any (only if they share any hit)
  - May be killed by any
  - If it survives, it may kill any
- All must happen before the next segment arrives (2 ns!)
- Pipeline the input stage: before being written, each incoming segment spends 4 cycles calculating its “duel” result with all 144 pre-stored segments
- **But…** when a hit in the pipeline reaches the table, the table may have been changed already… Incoming hits “shoot bullets” at each other, these “bullets” also traverse the pipeline and reach its target in the required moment
- 6k LUT, 7k FFs, 4 BRAM

(*) Actually close to 300 bits, but unused are stored to RAM

The FPGA use case

CPU

- Generic, sequential, versatile
- Calculations that are not (very) parallel
- High numerical precision

GPU

- Large homogeneous parallelism: the same operation on many data items
- High numerical precision

FPGA

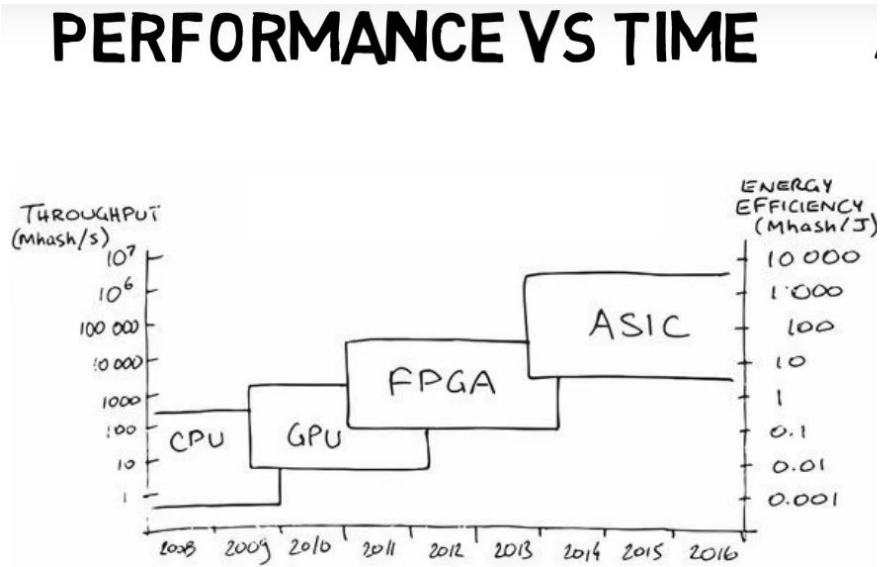
- Numerical **computation**: cumbersome
  - Inefficient on 64-bit data
  - But… do you really need 64 bits? With small numbers (<32 b?), things change
- Massive **parallelism**: ideal for data-flow-style modules
- Very **heterogeneous** algorithms: each little piece of the FPGA can execute a completely different task. For some, it is the only viable option; for others, it is good enough
- Ultra-low, deterministic **latency**. Real time → Trigger!
- Extreme **throughput**
- Extreme **connectivity**
- **Steep learning curve…** it may still pay off because of technical requirements or economies of scale. It is improving (HLS, acceleration platforms)

FPGA for computing: we are not the first to walk this path

- The greatest possible energy efficiency is achieved with silicon designed perfectly for your specific problem → **ASIC**. But: bugs, investment, zero versatility
- Algorithm with little parallelism and very sequential → **CPU**
- Identical operations on large vectors of high-numerical-precision data → **GPU**
- (Almost) anything else → **FPGA**. Custom circuits (**~ASIC**) for heterogeneous algorithms, but without “freezing” the silicon

Criptomining

Interested in efficiency ฿/€, ฿/kWh



Practical info for tomorrow's hands-on session

- We will be running some simulations and programming VHDL code. There are two options:
  - Web-based in edaplayground.com. You need to create an account in the page in advance, either with SSO (Google) or with email/pwd (slower?)
  - Locally in your PCs with open software stack (GHDL+GTKWave). I will provide upload download links and instructions for Windows, Linux and/or Mac (**which ones do you need?**) this evening.
- I'll demo the flow of programming and loading a design in a AMD/Xilinx FPGA with Vivado and illustrate some more concepts

Epilogue

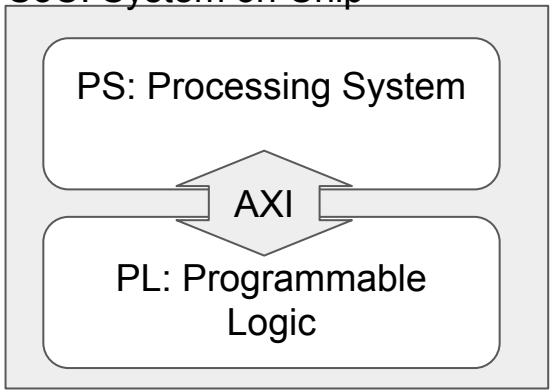
FPGA ♡ CPU

- If your task requires it, you can implement a processor in the FPGA fabric: these are called "**soft processors**". Manufacturers usually make this task easier.
- In fact, industry uses FPGAs for pre-production testing of all kinds of ASICs, including processors (although obviously it is not the fastest path to having a processor)
- FPGAs can be combined with "**hard**" processors. This is called an SoC (System on Chip); in Xilinx, this is the Zynq family.
- Connecting PS and PL is an interesting challenge: it is normally done through a family of protocols called AXI
-

ARM White Paper

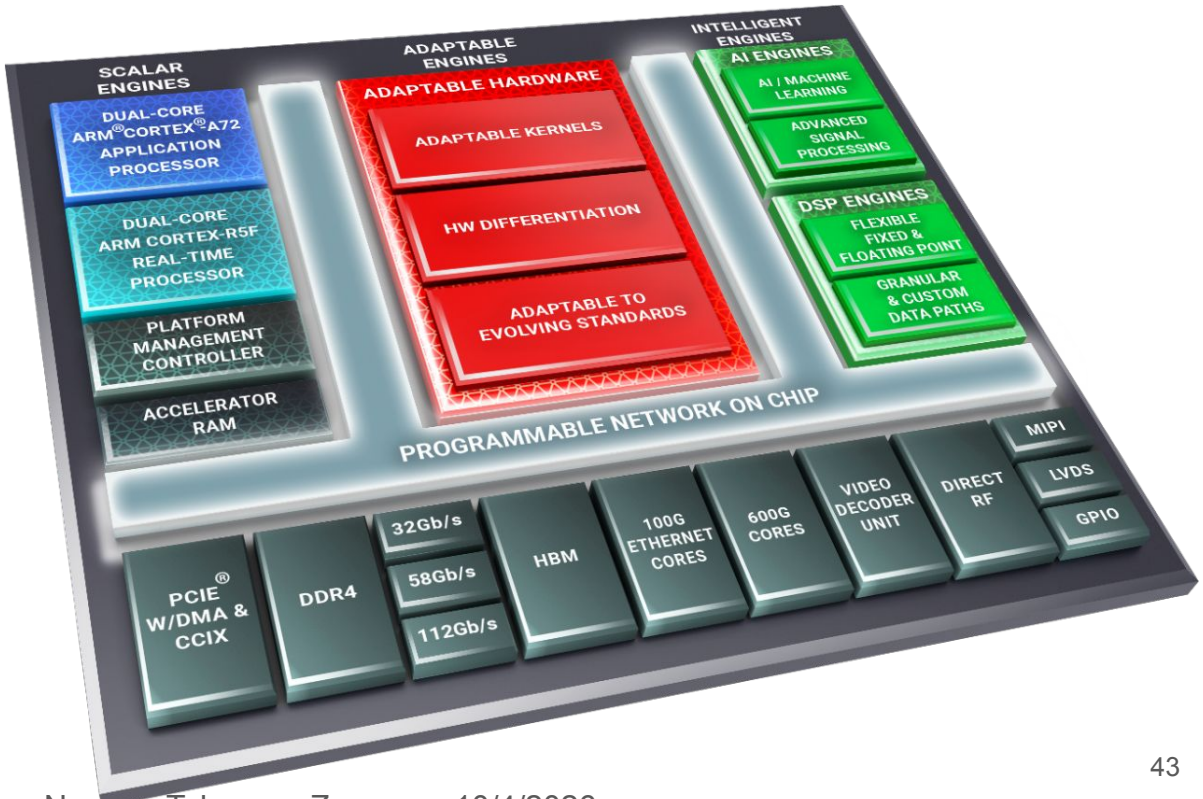
Prototyping ARM® Cortex®-A Processors using FPGA platforms

SoC: System on Chip



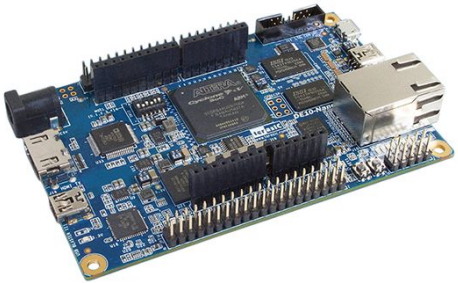
Beyond Zynq (CPU-FPGA SoCs): Versal

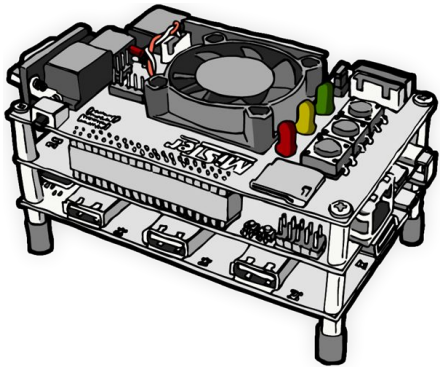
- In Versal, Xilinx's latest FPGA family, FPGAs have been abandoned: it now only sells SoCs
- They have processors, CPUs, and in some devices AI engines and DSP engines



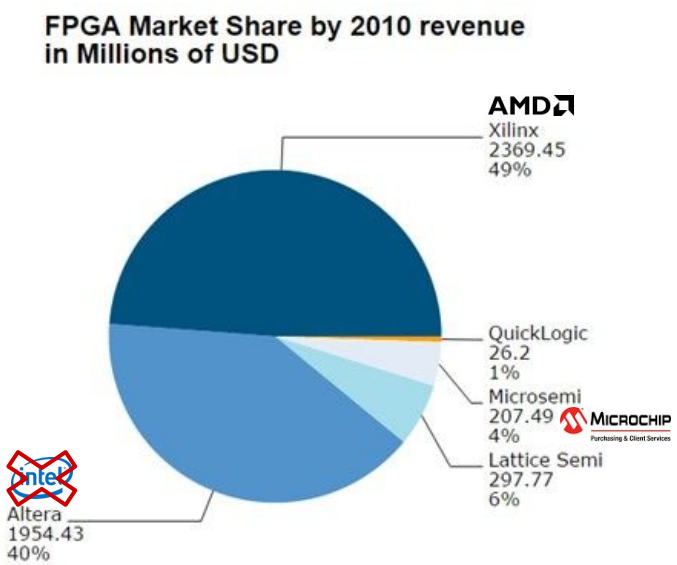
FPGA for ASIC prototyping… and for emulation!

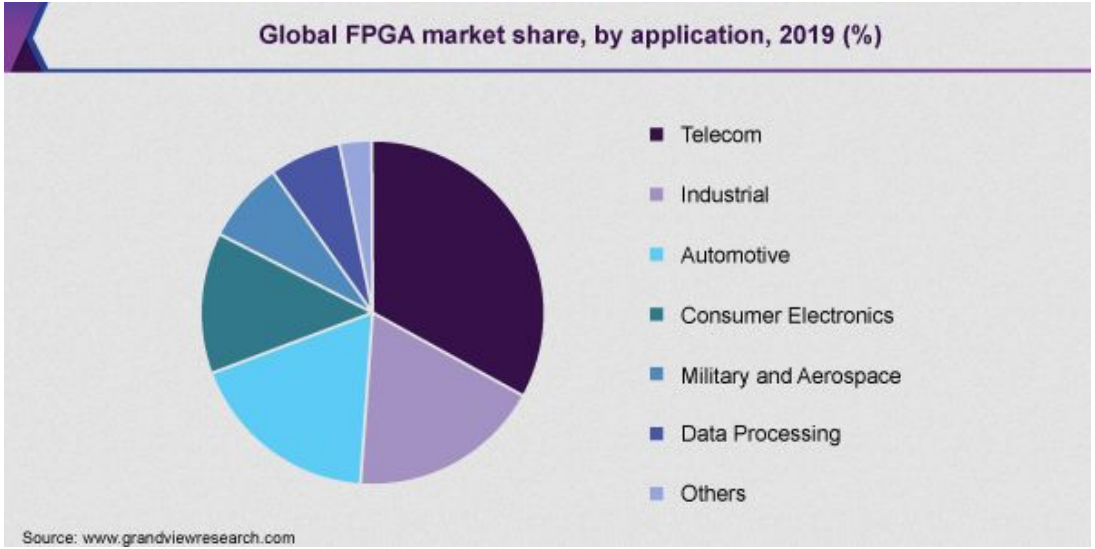
- Able to run video game console games from more than 20 years ago directly from their original cartridges
- It is not magic; they have simply emulated the complete chip of that video game console in an FPGA
- Altera Cyclone V





FPGA market

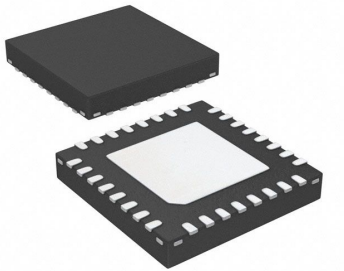

FPGA Market Share by 2010 revenue in Millions of USD
AMD
Xilinx
2369.45
49%
QuickLogic
26.2
1%
Microsemi
207.49
4%
MICROCHIP
Lattice Semi
297.77
6%
Altera
1954.43
40%


Global FPGA market share, by application, 2019 (%)
Telecom
Industrial
Automotive
Consumer Electronics
Military and Aerospace
Data Processing
Others
Source: www.grandviewresearch.com

FPGA range

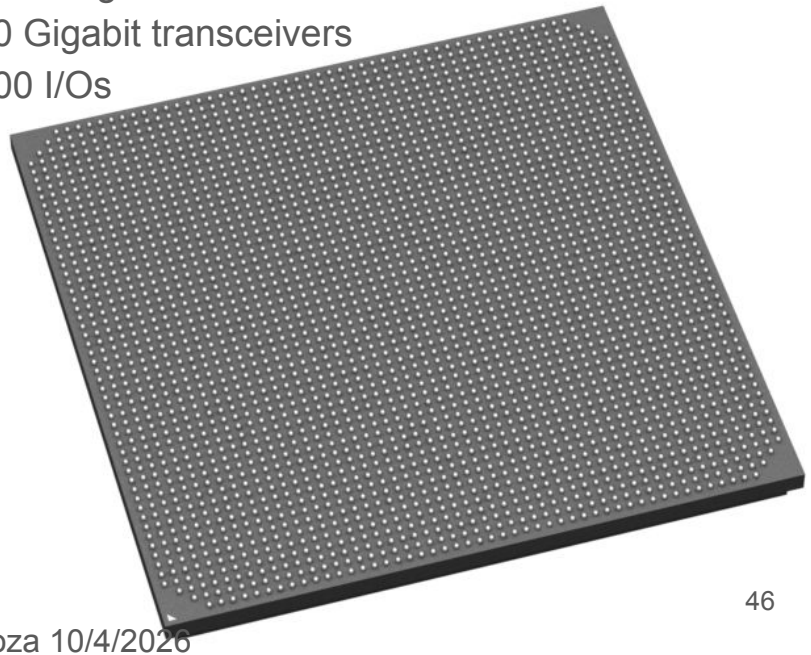
Lattice Semiconductor ICE40LP384

- ~2 €
- ~400 Logic Elements
- 21 I/Os



AMD VP1902

- ~30,000 € 😱😱
- ~20M Logic Elements
- 160 Gigabit transceivers
- 2000 I/Os



4th COMCHA School

Zaragoza, 8 – 15 April 2026

FPGA hands-on

Álvaro Navarro Tobar (Ciemat)

Proyecto PID2023-148896NB-I00 financiado por:

MINISTERIO DE CIENCIA, INNOVACIÓN Y UNIVERSIDADES

Cofinanciado por la Unión Europea

AGENCIA ESTATAL DE INVESTIGACIÓN

Part 1: Vivado walkthrough

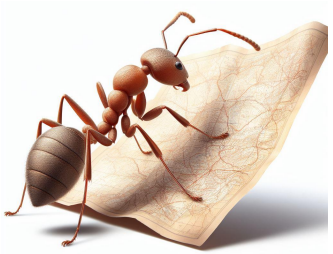
Let's look at the sources

top.vhd

- Comments
- Import libraries… IEEE!!
  - std_logic_1164
  - numeric_std
  - Don't use other non-standard libraries!
- Entity declaration: "ok so now I'm going to start describing an entity (= module = "chip") that has these interfaces to the outside world"
- Architecture: "and now I'm going to describe how this entity works"
  - Declarative preamble
  - begin… here starts the fun

boolean.xdc

- Comments
- Pin assignments (with information about electrical standards)
- Timing clock constraints: 100 MHz!
- Also cound have:
  - Timing constraints (pathsthat aren't critical, etc.)
  - Floorplanning (instructions for placement)
  - …



Let's run the workflow

- In addition to the top module, there are more sub-modules
- top.vhd instantiates several other entities
  - Each is defined in a separate vhd file in src/
  - top simply instantiates and interconnects them; it's not describing any logic → HDL
  - Look at the ports and signals assignments
- RTL schematic is a representation of what is written in the code
  - You can zoom into the interior of some modules. For example edge detector contains a FF + AND gate
- Synthethise the project
  - Open the synthesis results (log and design runs)
  - View the synthesized schematic
- Run implementation
  - Resource utilization can vary
  - Timing closure summary
- Open Implemented design
  - Zoom on the design and observe some of the logical elements, "hover" on some of the boxes
  - Right click, show connectivity
  - Click routing resources
- Run Generate bitstream, open hardware manager, load design

Lessons learned so far

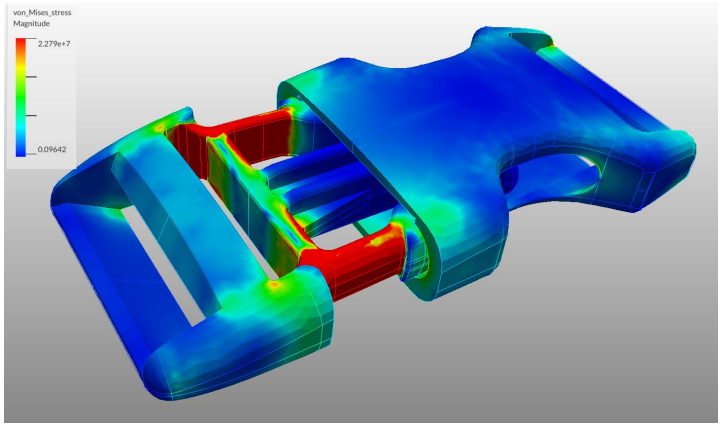
- FW projects consist in hierarchy of modules that instantiate (call) other modules to get the desired behaviour
- Just as in SW projects, it is better to keep independent functionality separated in different files so that you don’t have to rewrite many times your code
- HDL as a description of what modules we’re using and how they interconnect
- Synthesis interprets HDL and generates an shematic of interconnected FPGA primitives (not logical functions)
- Implementation can affect resource utilization.
- Timing is evaluated after implementation (when all components are placed and the nets are routed).
- The implemented design consists of the FPGA elements placed in their positions and interconnected.

Part 2: Coding time!

Simulation: run your circuit-program in a computer

In addition to the vhdl files that define our code, we need a new special file called **testbench**:

- It instantiates the module you want to test and connects signals to its ports
- It assigns values to the input ports of your module
- It uses special statements that are not intended for synthesis: after, wait, assert



Simulation option 1: edaplayground.com

1	https://www.edaplayground.com/x/9kBY	AND Gate
2	https://www.edaplayground.com/x/EwXR	Majority Gate
3	https://www.edaplayground.com/x/Y_GF	Unanimity Gate
4	https://www.edaplayground.com/x/iK92	Noise filter
5	https://www.edaplayground.com/x/rLG3	AND Gate with recent memory
99	https://www.edaplayground.com/x/QKUd	Coding styles

Simulation option 2: local vhdl simulator

Plan B if edaplayground.com is down or misbehaves (happens sometimes)… We can try to run the simulation locally

- Download the vhdl_simulator.zip file from indico and unpack
- Cross fingers if you're in linux or mac
- Run the get_binaries script for your system:
  - Windows: downloads binaries
  - Linux: Mix of install packages and download binaries
  - Mac: installs packages, requires Homebrew
- Run the launch script (run_xxxx) in order to simulate the examples and exercises

For the savvy

My sons won't stop saying "six seven". It's maddening. It's not like they say it 6 or 7 times a day — peak rates are closer to 67 "six-sevens" per minute. It's driving me crazy. They say I'm exaggerating, so I want to prove them wrong. As proper sons of an FPGA developer, they speak in a binary stream, with each digit encoded in 4 bits (they only know how to count up to F — cute little creatures). So, 6 is `0110` and 7 is `0111`. Please design a module that receives a clock signal `clk`, a serial input bit stream `brainrot` and produces: an 8-bit unsigned output `counter : out unsigned(7 downto 0)`

The counter must increment by 1 each time the pattern **six-seven** (6→7) is detected. However, my children sometimes invert it, so you must also count **seven-six** (7→6). Do **not** count six-six (6→6) seven-seven (7→7). Additionally, overlapping patterns must only be counted once. For example the sequence six-seven-six should count as **one occurrence**, not two.

Assume the input stream is continuous and digits are received serially, 4 bits per digit. Try to make your design elegant.