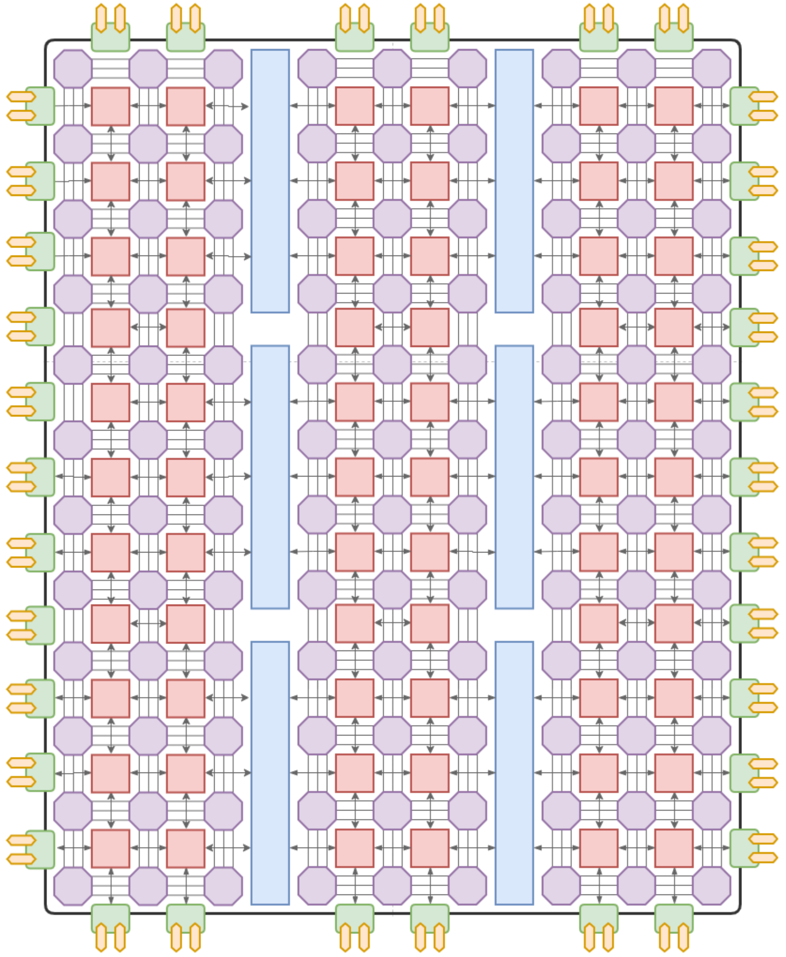
A Scientist's Guide to FPGAs

iCSC 2019

Alexander Ruede
(CERN/KIT-IPE)

Content

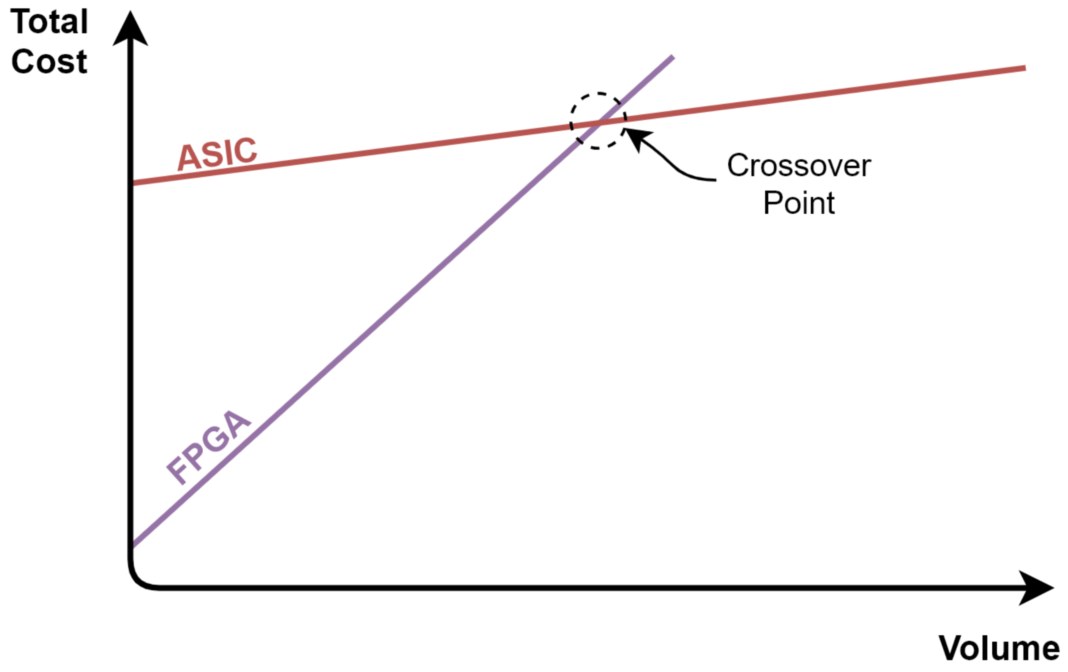
1. Introduction
2. The Emergence of FPGAs
3. Digital Design
4. Anatomy of FPGAs
5. Classical Design Flow
6. CPU / GPU vs. FPGA
7. Pros & Cons
8. Applications
9. Examples
10. Getting Started



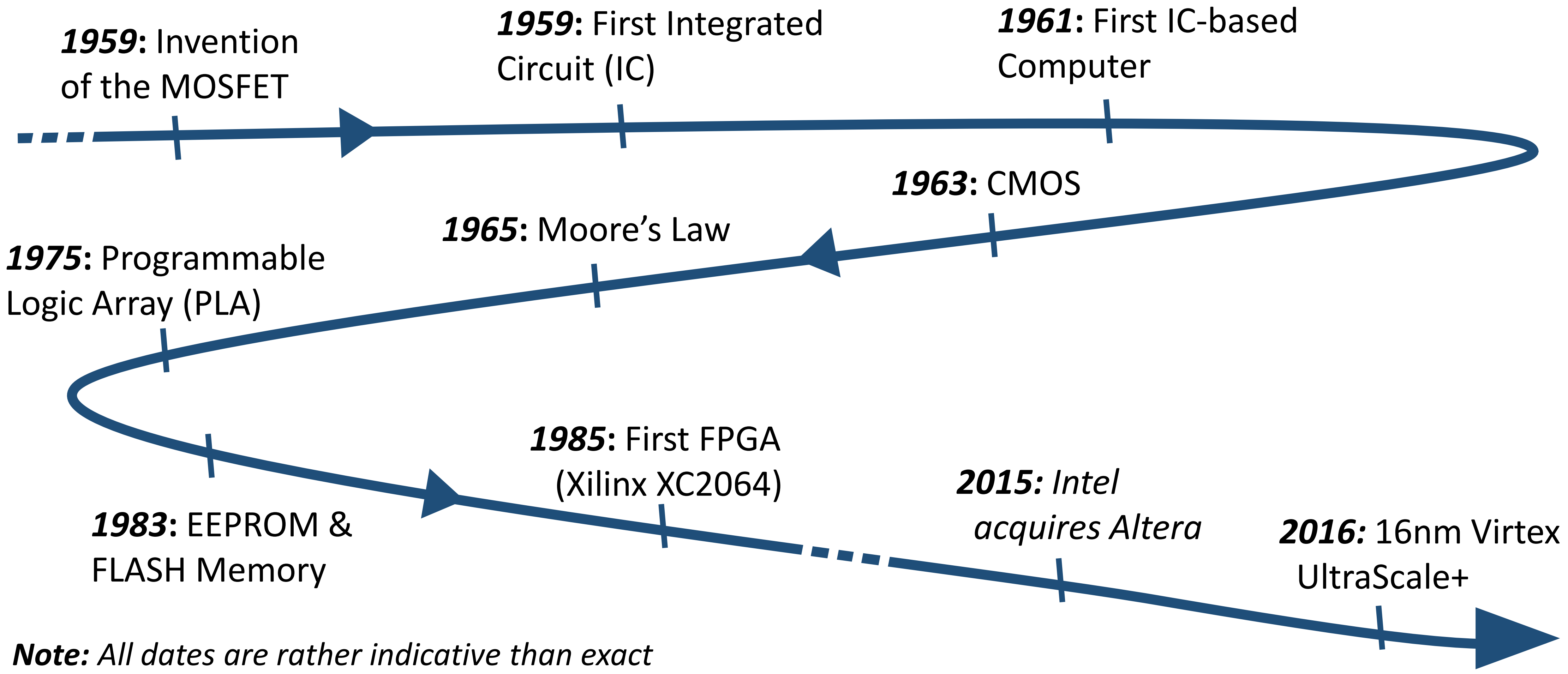
1. Introduction

FPGA: Field-Programmable Gate Array

- Growing $\left\{\begin{array}{c}\text{market}\\ \text{applications}\\ \text{popularity}\end{array}\right\}$ of FPGAs
- Well established in HEP experiments
- Finding the way into data centers
- Can be substitution for:
  - ASICs (traditionally)
  - Processors (recently)



2. The Emergence of FPGAs


1959: Invention of the MOSFET
1959: First Integrated Circuit (IC)
1961: First IC-based Computer
1963: CMOS
1965: Moore's Law
1975: Programmable Logic Array (PLA)
1983: EEPROM & FLASH Memory
1985: First FPGA (Xilinx XC2064)
2015: *Intel acquires Altera*
2016: 16nm Virtex UltraScale+

Note: *All dates are rather indicative than exact*

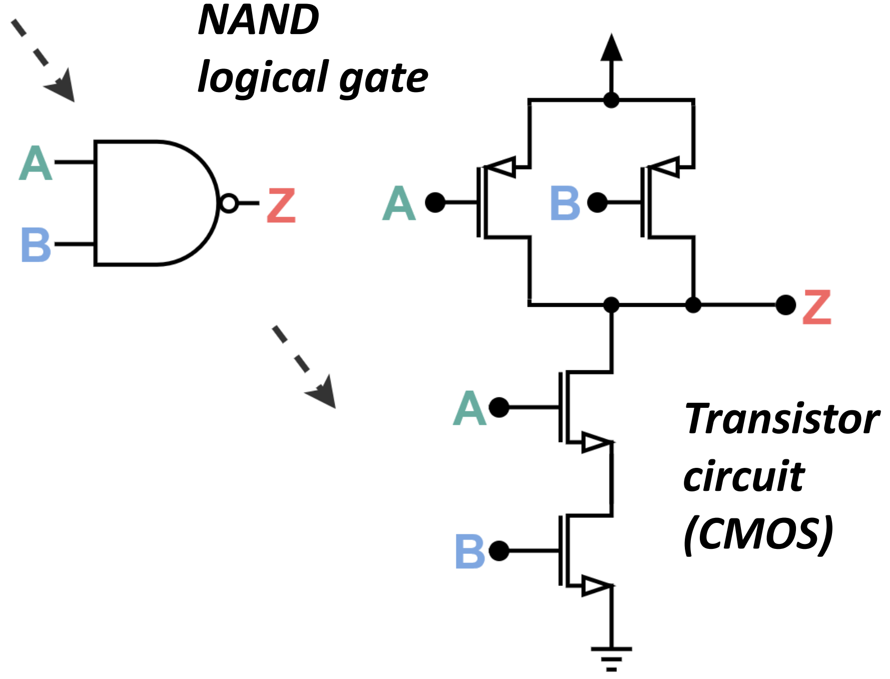
3. Digital Design

Digital Logic

- Digital information is processed and stored in *binary* form
- *Boolean algebra* and *truth tables* are used to express combinatorial logic circuits
- Basic logical function (AND, OR, etc.) are abstracted in *logical gates*
- Gates can efficiently be implemented in transistor circuits (e.g. CMOS)
- Every logical function can be implemented by using gates

A	B	Z
0	0	1
0	1	1
1	0	1
1	1	0

Truth table of logical function

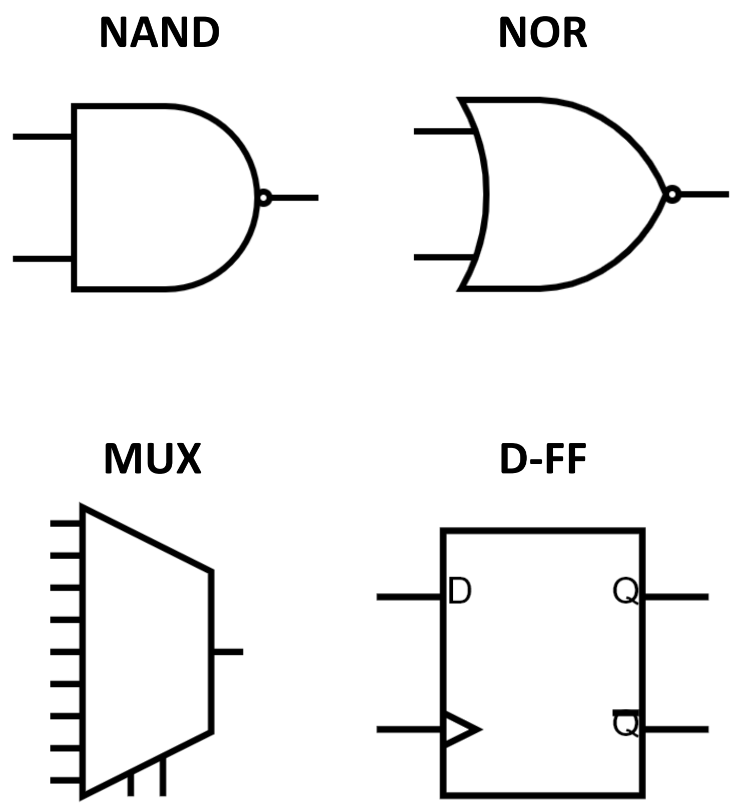


3. Digital Design

Digital Building Blocks and Processes

There are two kinds of processes with different building blocks:

1. **Combinatorial**
   → *"Instant" state changes, e.g.:*
   - Classical gates (especially NAND & NOR)
   - Multiplexer (MUX)
2. **Synchronous**
   → *"Clocked" state changes, e.g.:*
   - Flip Flop (e.g. D-FF)
   - FIFO (First-In First-Out)

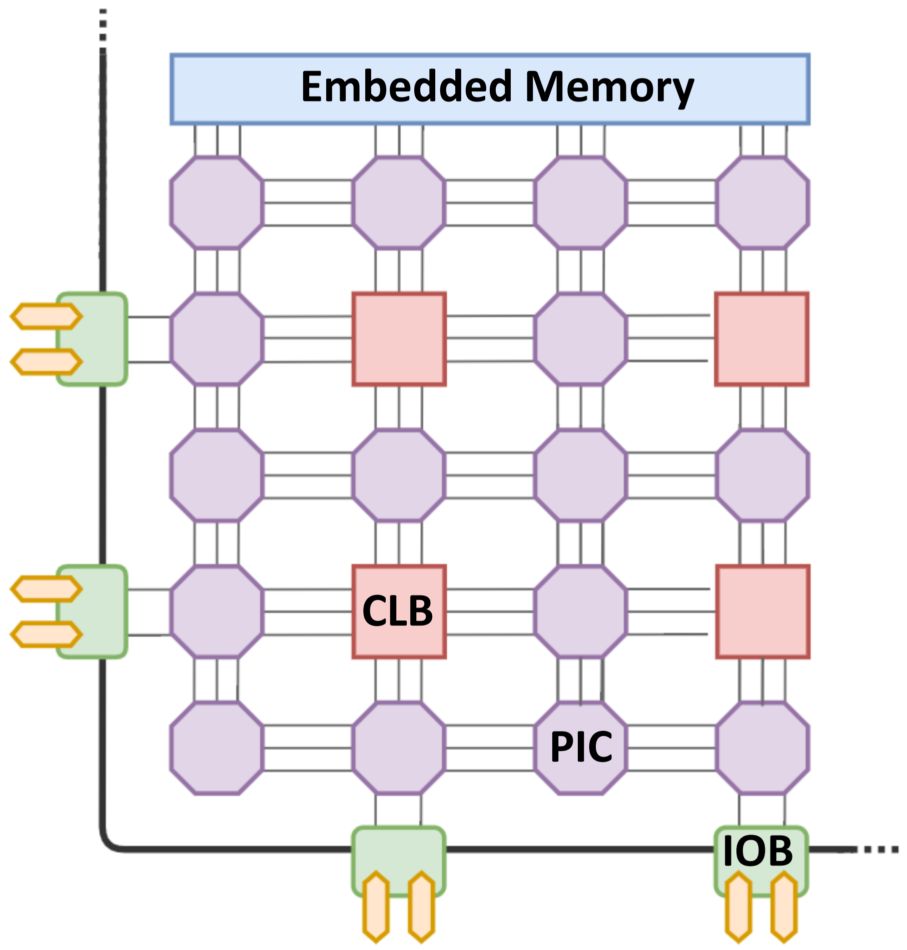


4. Anatomy of FPGAs

Architecture

Configurable logic blocks, interconnected by a switch matrix and surrounded by I/Os.

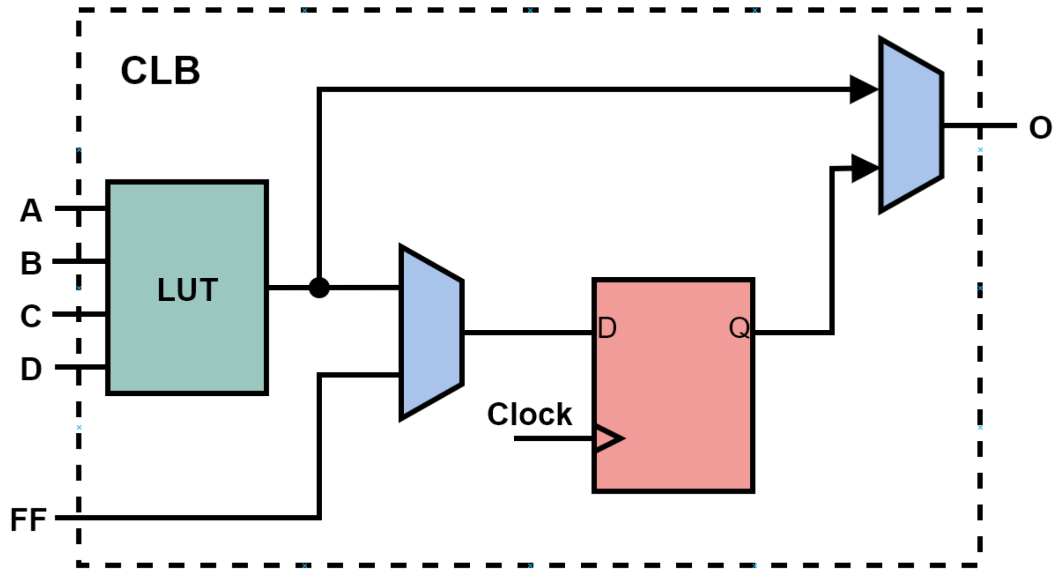
- CLB: Configurable Logic Block
- IOB: Input-Output Block
- PIC: Programmable Interconnect
- Clock Management
- Memory
- Hardened Cores



4. Anatomy of FPGAs

Configurable Logic Block (CLB)
(Also “logic cell” or “logic element”)

- Logic functions are implemented in Look-Up-Tables (LUTs)
- LUTs can implement any arbitrarily defined n-input Boolean function
- D-type flip-flops (storage elements) can be triggered on either clock edge
- Flip-flops can take input from outside the CLB or from the LUT



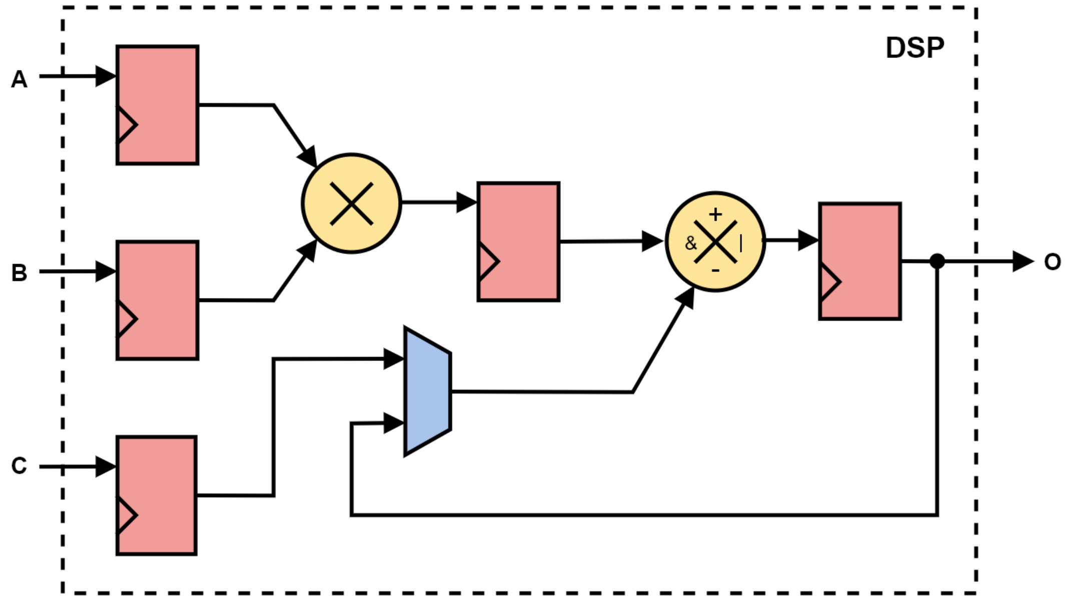
Simplified example CLB with one 4-input LUT and one flip-flop

4. Anatomy of FPGAs

Hardened Cores (Also called “IP cores”)
Complicated tasks (e.g. multiplication) take up a lot of logic cells
→ *Hardened cores in silicon* for more effective use of resources

Typical cores found in modern FPGAs:

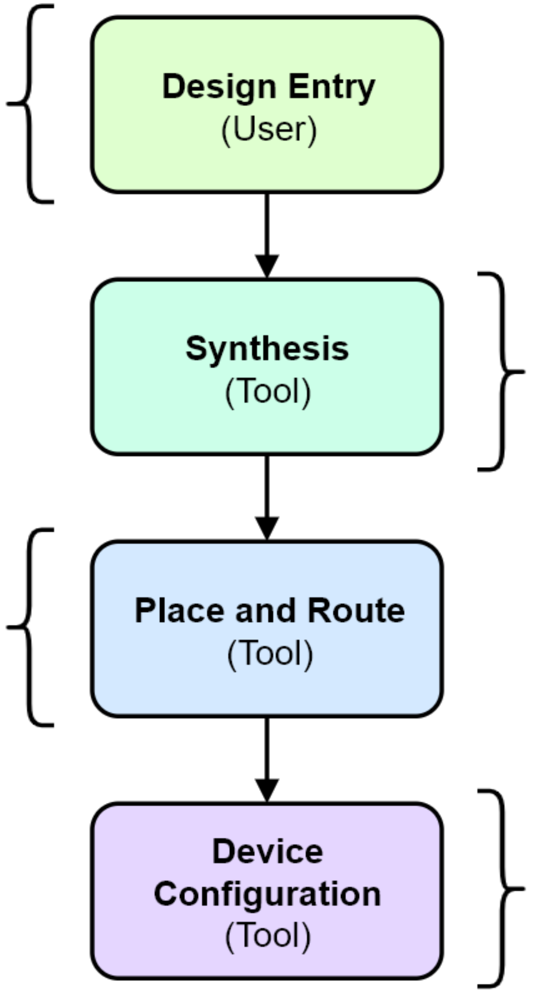
- Memory (Block RAM)
  - → FIFO
  - → Shift Register
- DSP blocks
- Clocking (Programmable PLL)
- Communication interfaces (e.g. PCIe)
- Serializer/Deserializer (SerDes)
- CPU



Exemplary DSP block with multiplier, accumulator and pipeline stages

5. Classical Design Flow

- First: Sketch design on paper!
- Write design in HDL (hardware description language), e.g. VHDL or Verilog, or use schematic entry
- Behavioral Simulation



- Synthesis tool converts hardware description into netlist
- Can perform logic optimization, register load balancing, etc.

- Find best location of primitives for all elements in netlist
- Programming interconnects, respecting all timing information
- Placement can be constrained and defined by user

- Generating bit stream for direct FPGA programming or for external memory configuration

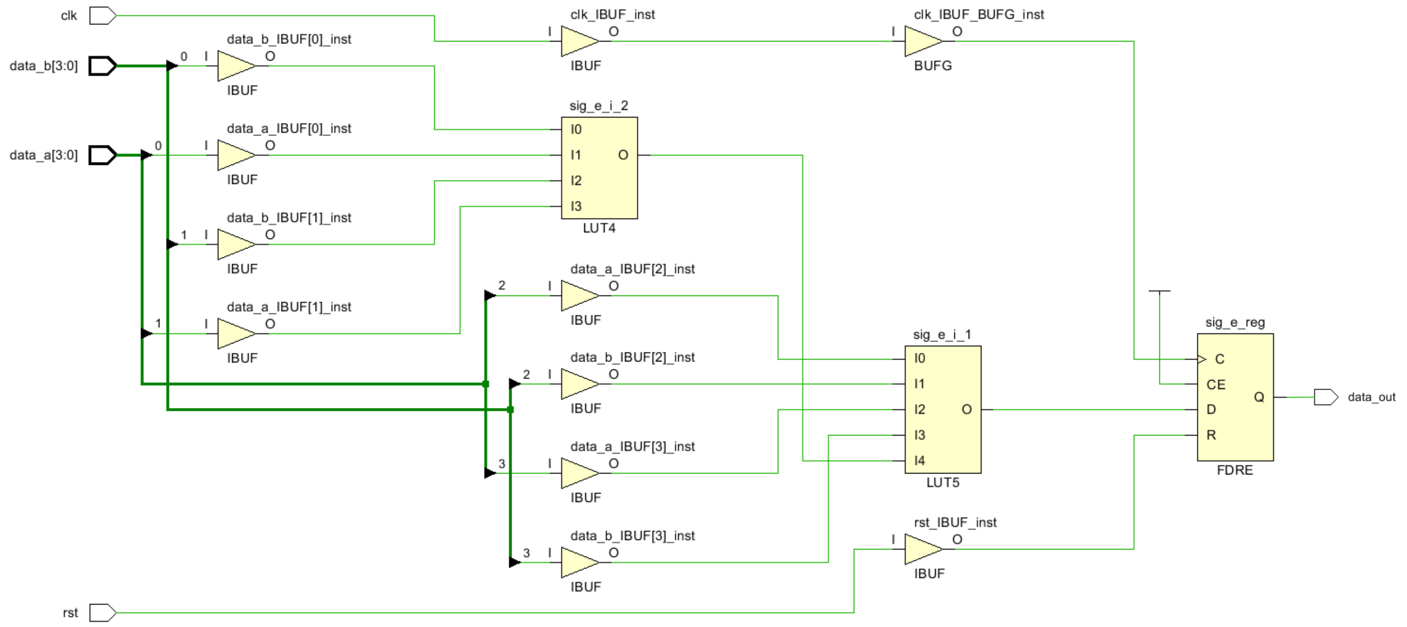
5. Classical Design Flow

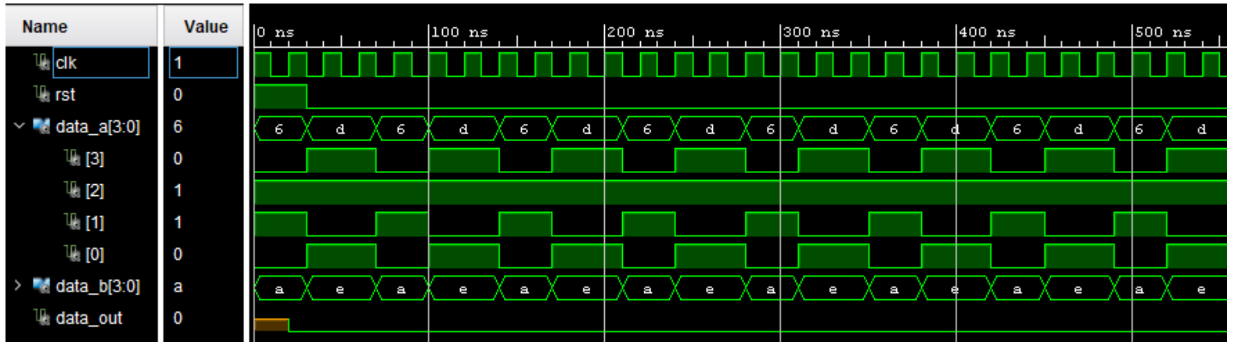
How it looks like…

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

entity test is
    Port(
        clk : in std_logic;
        rst : in std_logic;
        data_a : in std_logic_vector( 3 downto 0 );
        data_b : in std_logic_vector( 3 downto 0 );
        data_out : out std_logic
    );
end test;

architecture Behavioral of test is
    signal sig_d : std_logic_vector( 3 downto 0 );
    signal sig_e : std_logic;
begin
    sig_d <=  data_a xor data_b;
    sync_proc : process( clk )
    begin
        if rising_edge( clk ) then
            if rst = '1' then
                sig_e <= '0';
            else
                sig_e <= sig_d(0) and sig_d(1);
            end if;
        end if;
    end process;
    data_out <= sig_e;
end Behavioral;
```





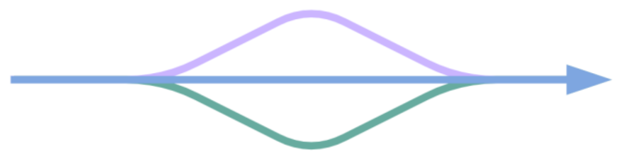
Name	Value
clk	1
rst	0
data_a[3:0]	6
[3]	0
[2]	1
[1]	1
[0]	0
data_b[3:0]	a
data_out	0

Recapture

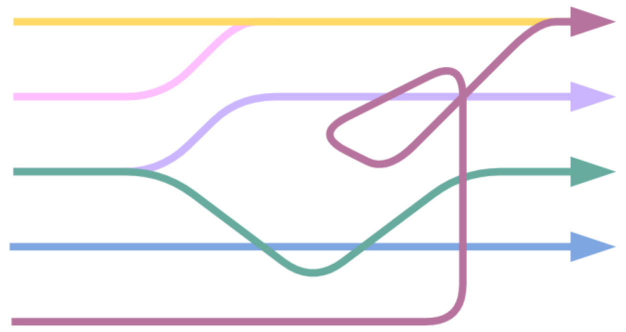
- **Digital Design**
  - Digital information is *binary*
  - We can express logical function in *Boolean algebra* or *truth tables* and model them in circuits of *logical gates*
  - Processes can be *combinatorial* (instant) or *synchronous* (sequential)
- **Anatomy of FPGAs**
  - *Configurable logic blocks*, surrounded by *I/Os* and interconnected by a *switch matrix*
  - Programming the FPGA: "Writing values into LUTs and configuring switches"
  - Dedicated blocks for *memory, clocks, signal processing, communication*, etc.
- **Design Flow**
  - The *hardware description* is translated into a *netlist* by the *synthesizer*
  - The *placing and routing* finds the best locations for the primitives and *interconnects* the components

6.1 Software vs. HDL

- **Software / CPU**
  - *Specifying a sequence of instructions*
  - Implicit sequential processes
  - Explicit concurrency
  - Fixed memory hierarchy



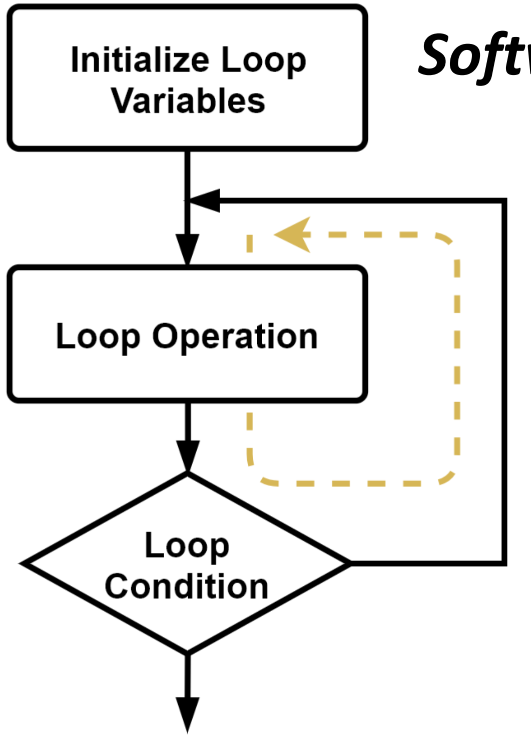
- **HDL / FPGA**
  - *Describing structure and behavior of digital components*
  - (Implicit) Arbitrary concurrency
  - Synchronous and/or purely combinatorial processes
  - Notions to explicitly express time (in simulation)
  - Flexible memory hierarchy



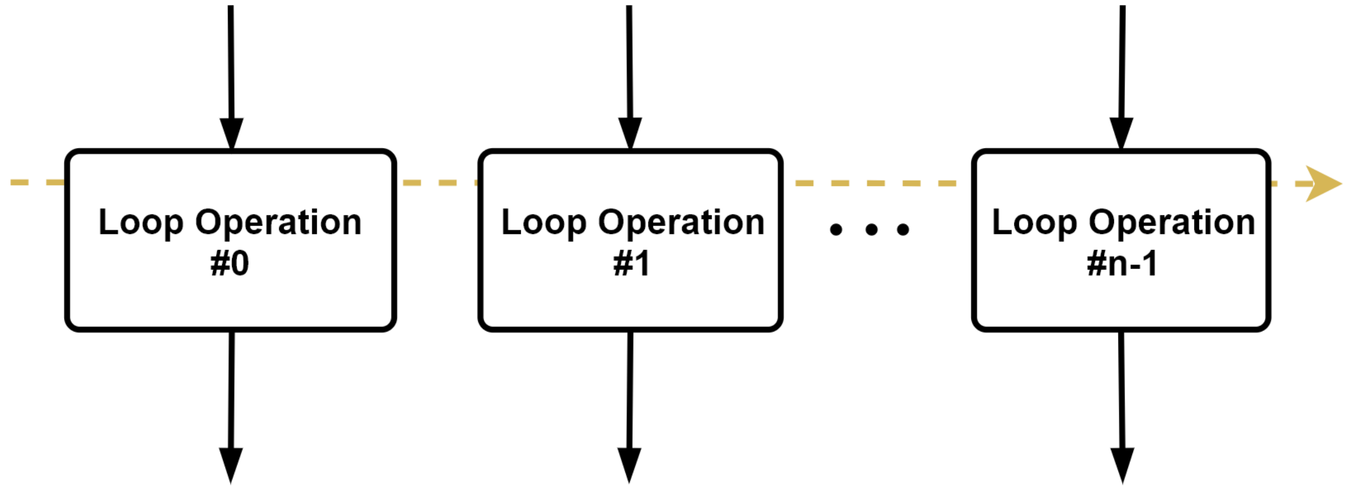
6.1 Software vs. HDL

Example: For-Loop

Software



HDL



6.2 GPU vs. FPGA

GPU: Large array of ALU cores, including cache memory and thread interfaces
FPGA: Large array of logic blocks, surrounded by I/Os, connected by a switch matrix

	GPU		FPGA
• **Data path and data types:**	Fixed	X	Fully customizable
• **Memory:**	complex hierarchies	X	Flexible
• **Floating Point:**	Native support	X	Limited support
• **Power Efficiency*:**	Low to Medium	X	Very High

* “GPU vs FPGA Performance Comparison”- *BERTEN DSP S.L., BWP001 v1.0, 2016*

7. Pros & Cons

Pros

- High flexibility
- Customizable data types
- Arbitrary concurrency
- Connectivity
- Power efficiency
- Real time suitability
- Suitable for safety critical applications

Cons

- High complexity
- Limited “math support”
- Effort for floating point implementation
- It’s **hardware** design - software knowledge does not apply
- Cost (for large devices)

8. Application Areas

Some of *many* application areas:

- **Networking/Telecommunication**
  → High throughput, I/O density
- **Space**
  → Radiation hardness, reconfigurability
- **Medical/Scientific Instrumentation**
  → Connectivity and customizability
- **Image Processing**
  → High throughput, parallelism, interfacing
- **Machine Learning**
  → Flexible data types, power efficiency

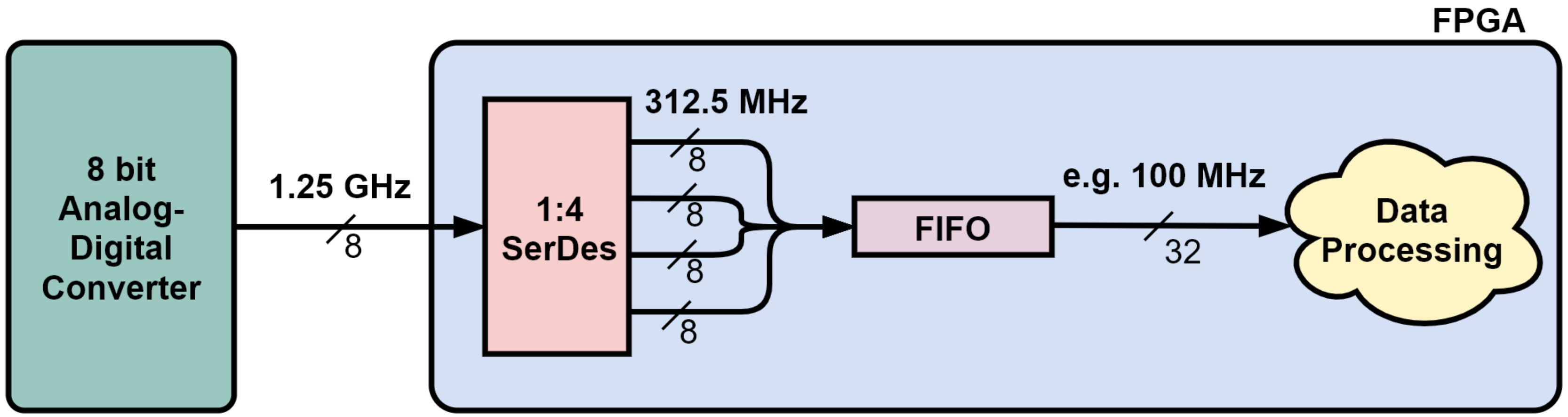
9.1 Example: High Speed Data Converter

Typical challenge in experimental setup:
Interfacing to high speed analog-digital converter (ADC)

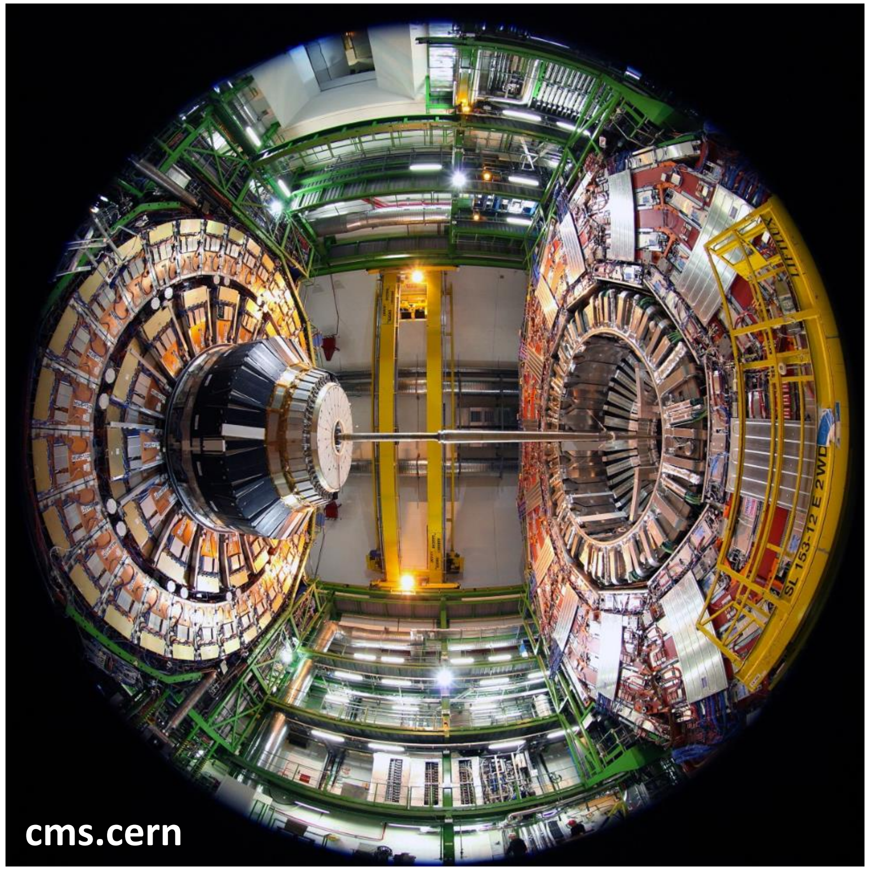
Example: 8 bit ADC with 1.25 GHz sampling rate

Challenge: Interface between high frequency sampling rate and lower FPGA-internal processing frequency

→ Deserialize data!

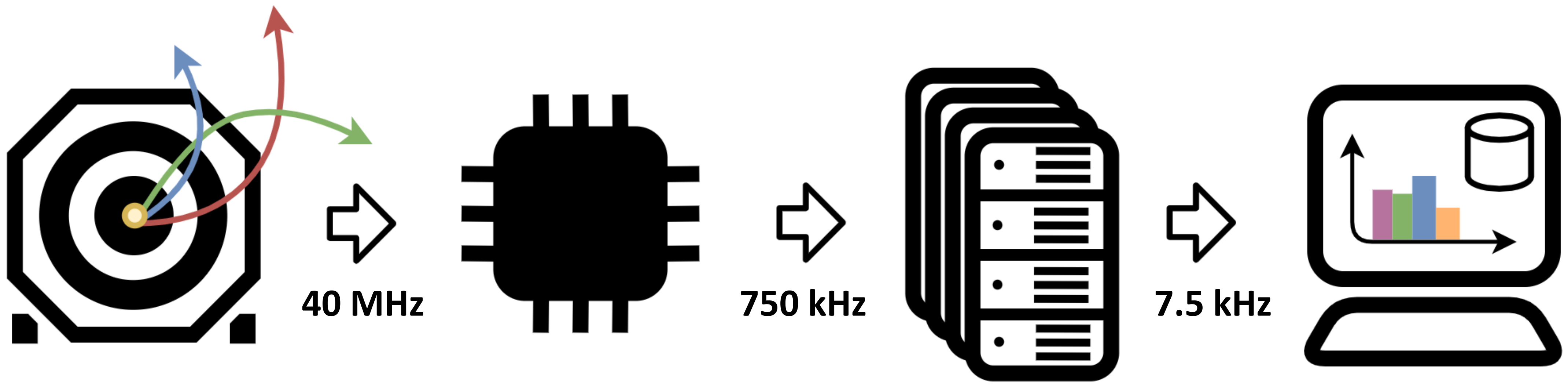


9.2 Example: CMS Phase II DAQ



9.2 Example: CMS Phase II DAQ

- **Phase II Trigger & DAQ**



CMS

- ~50 k High-speed links from detector
- ~50 Tb/s throughput

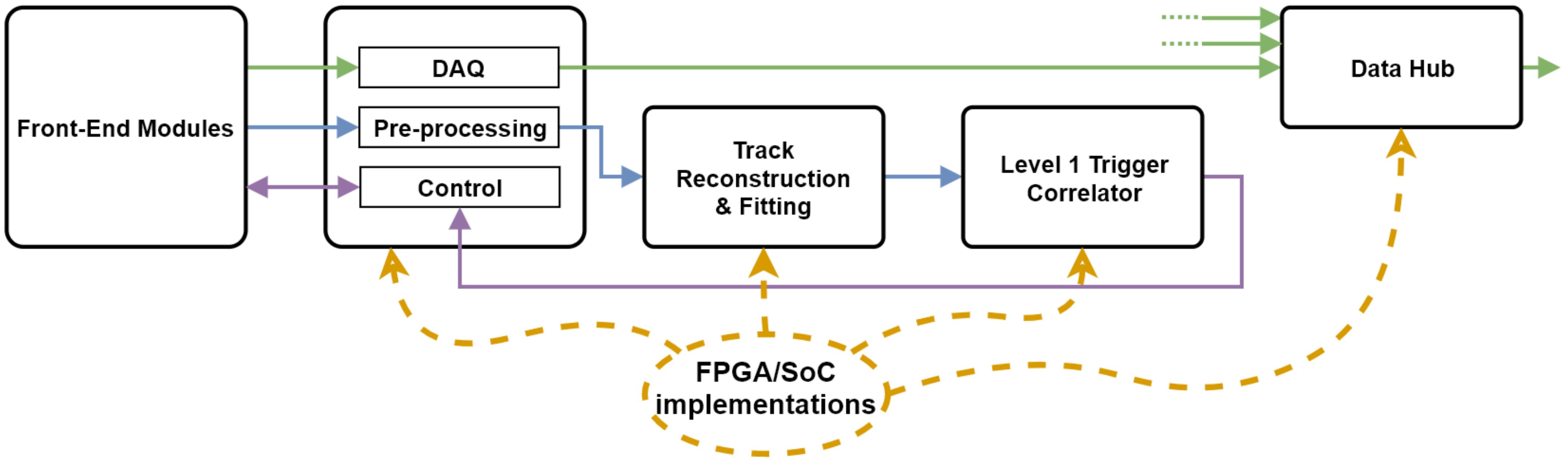
Level-1 Trigger

- FPGA implementation
- 12.5 µs decision time
- 98.125% rejection

High-Level Trigger

- Standard processing nodes
- < 1 s decision time
- 99% rejection

9.2 Example: CMS Phase II Tracker



- Hardcore real-time requirements
- High I/O density
- Custom protocols, (partially) custom interfaces
- Power and space requirements (underground cavern)

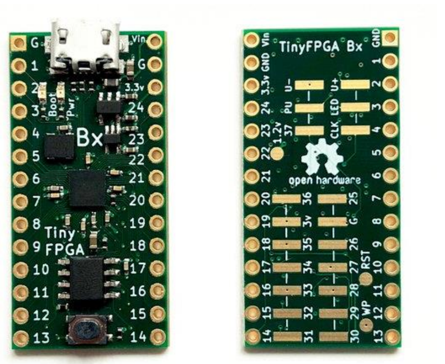
$\Bigg\}$ **Predestined for FPGAs!**

10. Getting Started

- Visit Giorgios lecture!
- Low-cost development boards:
  - TinyFPGA (open source) – Lattice XO2/iCE40 FPGA
  - Digilent Basys3 – Xilinx Artix-7 FPGA
- Most Vendors tool chains are for free,
  open source chains exist for some FPGAs
  (Project IceStorm for Lattice iCE40 FPGAs)
- Try out different languages:

  VHDL, Verilog, *Migen,* MyHDL,…
- Start with blinking LED (the "Hello World" of hardware)
- Trial and error + online education
- Realize your project!

TinyFPGA



Basys3

Backup: Languages

- Languages are often divided into two subsets:
  - *Synthesizable* → can be translated into a physical design
  - *Non-synthesizable* → only used for simulation
- Hardware Description Languages (HDLs)
  → *Good for advanced design optimization*
  - <u>Describe</u> and <u>simulate</u> hardware on behavioral/RTL level
  - VHDL, Verilog (the “classical” HDLs)
  - SystemVerilog (enhanced Verilog)
  - Migen, Chisel, Clash, Spinal … (unconventional/experimental languages)
- High Level Synthesis (HLS)
  → *Good for fast development cycles*
  - Higher level abstraction, based on C/C++
  - HDL as output product
  - OpenCL: Cross-Platform parallel language (good for SoCs)