Tampere University

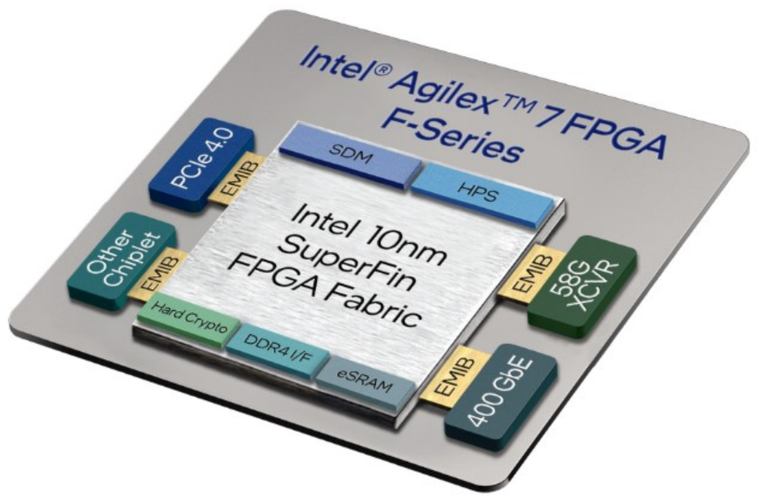
# Field-Programmable Gate Arrays

COMP.CE.250 System-on-Chip Design

Arto Oinonen

Tampere University

2026



## Outline

### FPGA Architectures

- Logic, interconnects, clocking, integrated macros
- Selection criteria

### Modern FPGA use cases

- Examples from the last 10 years

# History

First simple **PLA** (Programmable Logic Array) components in 1970s

First **FPGAs** in mid-80s

- Altera Classic series 1984
- Xilinx 2000 series 1985

In 2000s, complete systems are implementable with FPGAs

First **SoC FPGAs** in 2010s

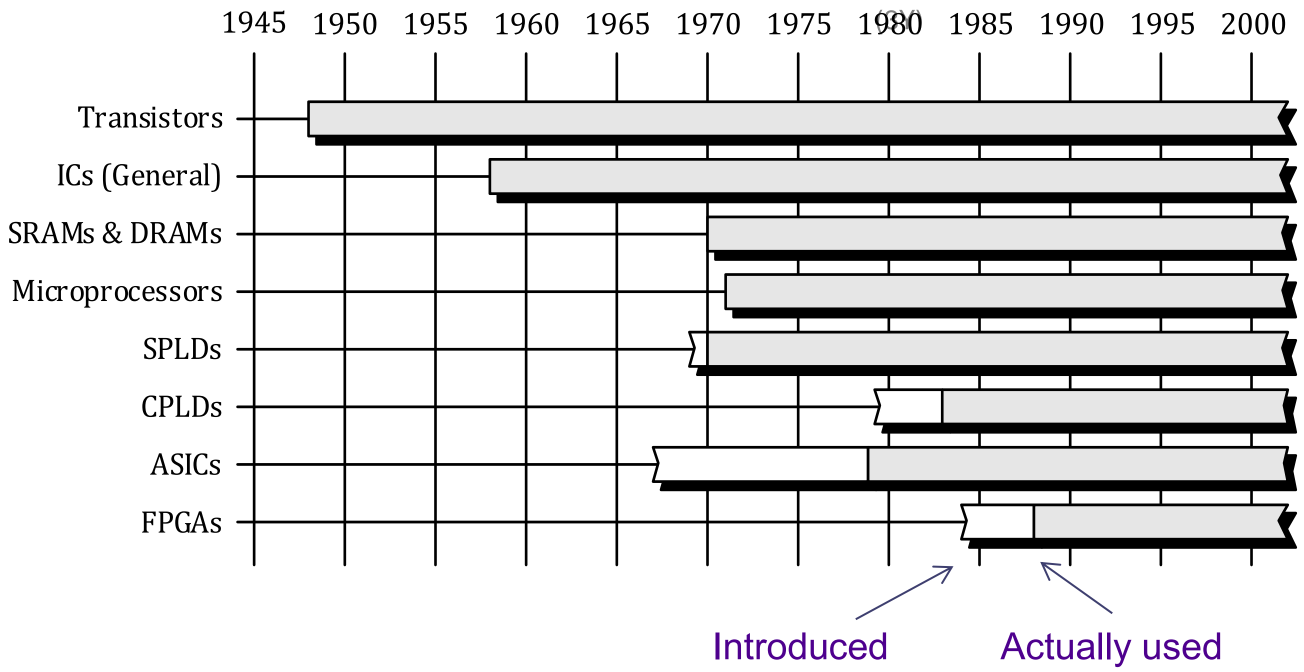
- FPGAs that integrate hard CPU cores
- Xilinx ZYNQ-7000 in 2011
- Altera Cyclone V in 2012

2018: First server grade CPU to include FPGA fabric – Intel Xeon Scalable 6138P

- 20 Intel Skylake CPU cores and Intel Arria 10 GX 1150 FPGA

Trends in 2020s:

- AI applications: specialized function units, more internal memory
- Data center acceleration: low latency / high bandwidth connectivity, “SmartNICs”



## FPGA manufacturers

- Altera acquired by **Intel** in 2015
  - **Altera** independent again in 2025
- Xilinx acquired by **AMD** in 2022
- Actel acquired by **Microsemi** in 2010 -> **Microchip Technology** 2018
- Also Lattice, Achronix, QuickLogic, ...

These slides still refer to Intel/Xilinx for old products, but prefer AMD/Altera

# SPLDs

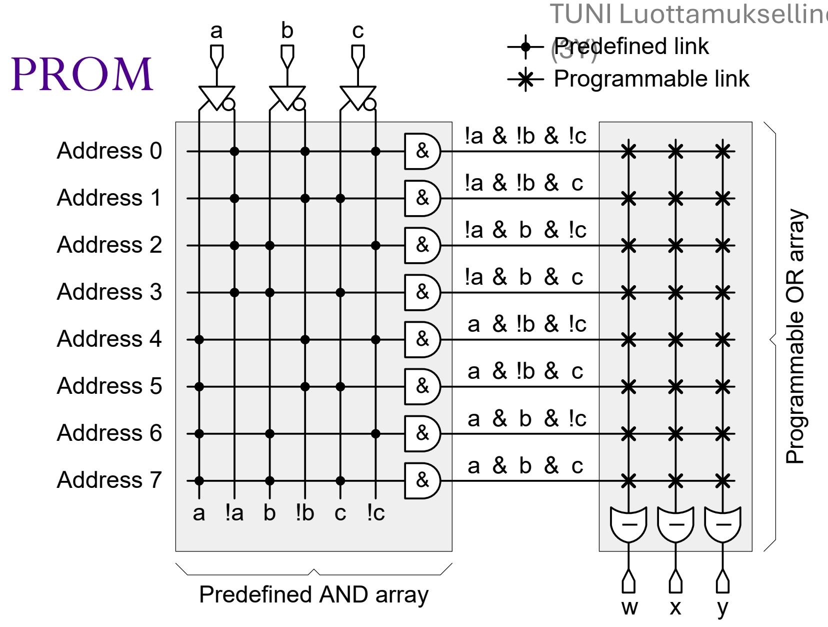
First Programmable Logic Devices (PLD) were PROMs in 1970

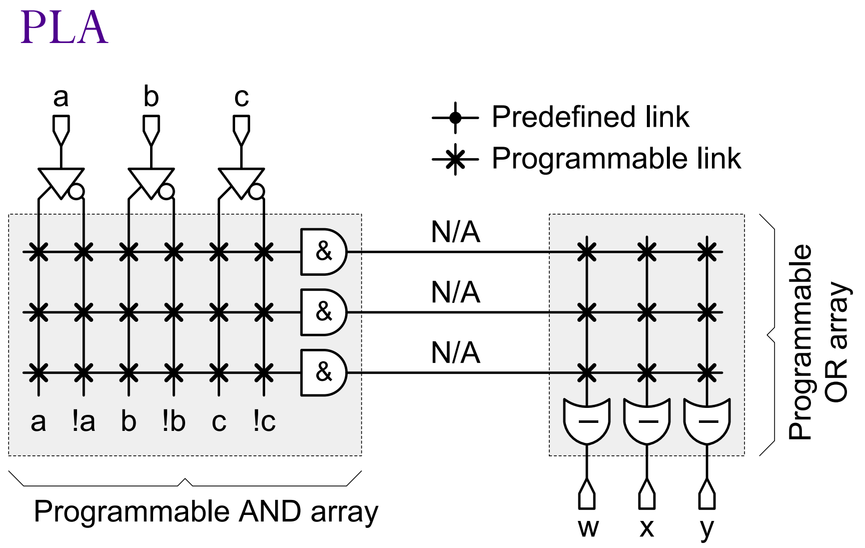
- OR gates were programmable

Evolution led to Programmable Logic Arrays (PLA) in the 1970s

- Both ANDs and ORs programmable

These are classified as Simple Programmable Logic Devices





# CPLDs

Complex Programmable Logic Devices were introduced circa 1980

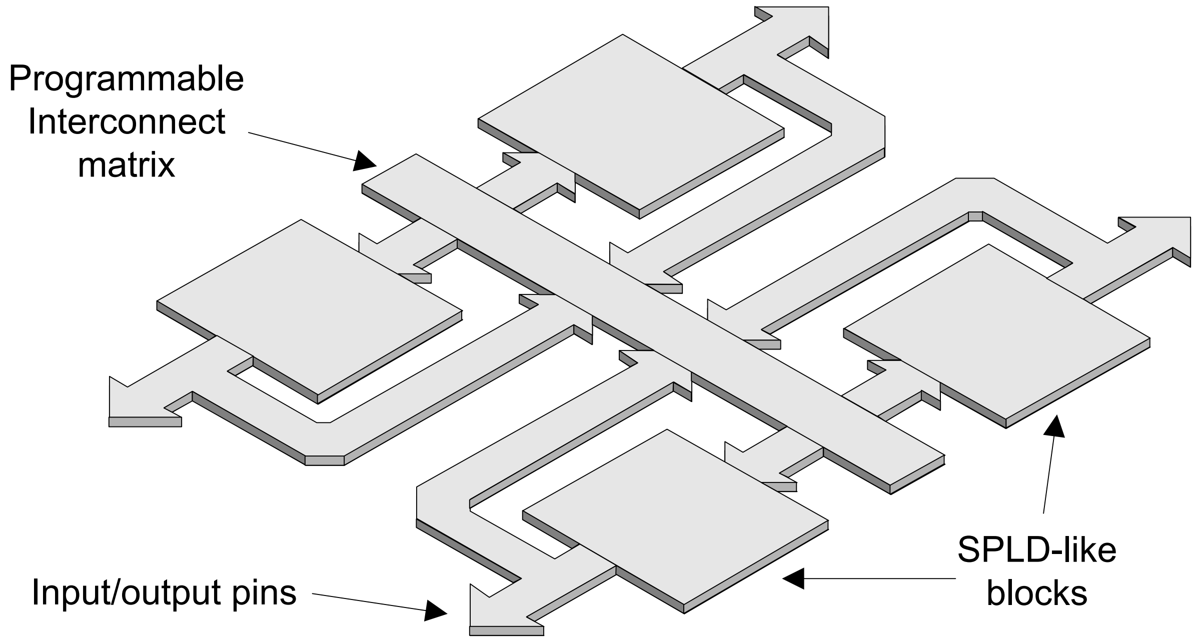
Main idea was that majority of the building blocks were not supposed (or could not be) connected to each other

- Usually every link is not required, some pins are unidirectional
- Significant save in interconnection area

=> Programmable interconnections nonetheless

Often non-volatile

None/few hard-macros



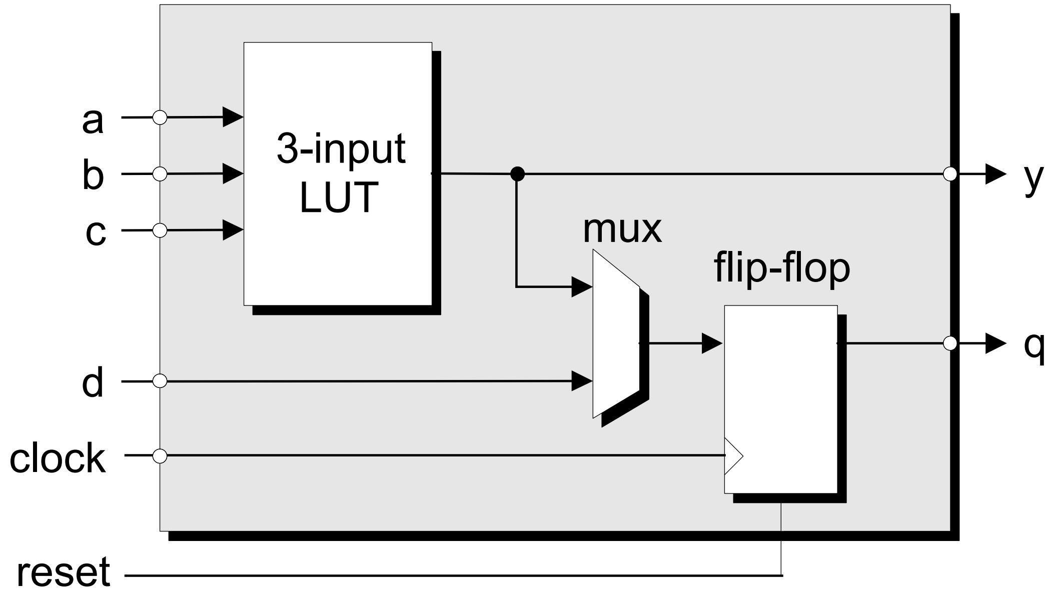
# FPGAs

## Field-programmable Gate Array

- Xilinx developed the first FPGA in 1984

## The AND and OR arrays are replaced by Programmable Logic Blocks

- Contains essentially a LUT and a flip-flop
- Look-up Table (LUT) implements a truth table
  - For example, a 4-input LUT can implement *any* function that has four inputs and one output



A very simple programmable logic block

# FPGAs #2

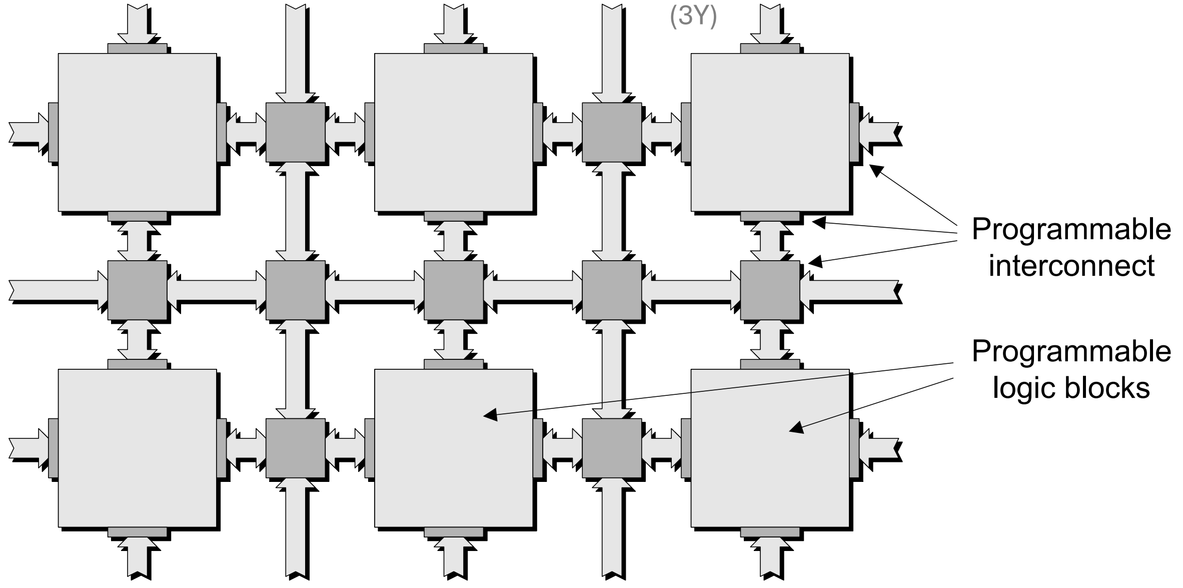
**FPGAs can implement vastly more complex functions than CPLDs**

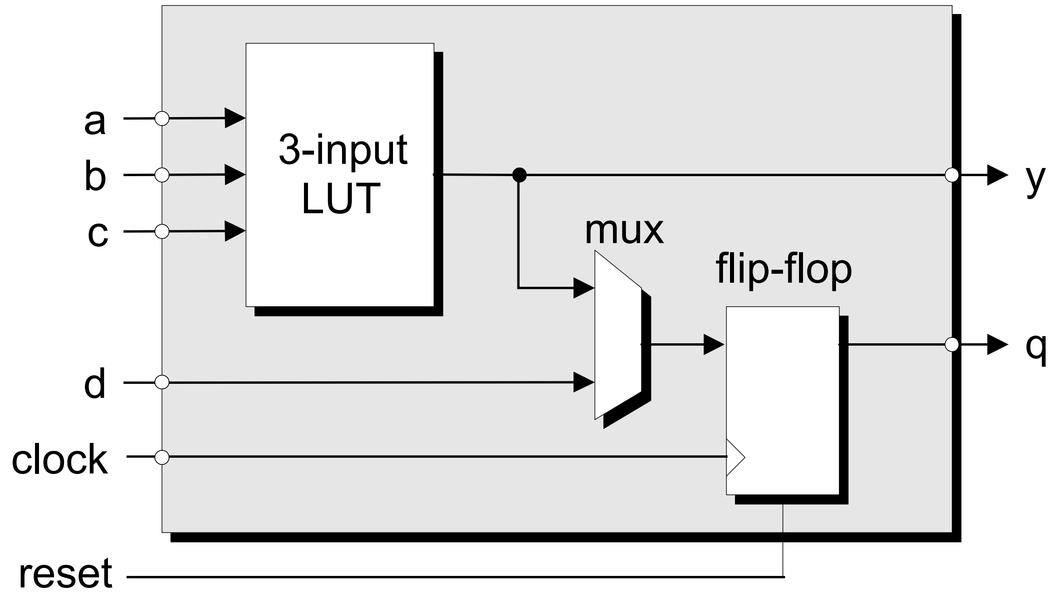
- In addition to programmable logic blocks, FPGAs can include additional primitives such as block RAM, multipliers (DSP), PLLs and transceivers

**Since the introduction of CPLDs, EDA tools started to emerge**

- Optimal placement of logic functions to the chip when having only limited number of links between the functions is far from trivial

**FPGA devices differ in their reconfiguration style (we'll return to this)**





# FPGA ARCHITECTURES

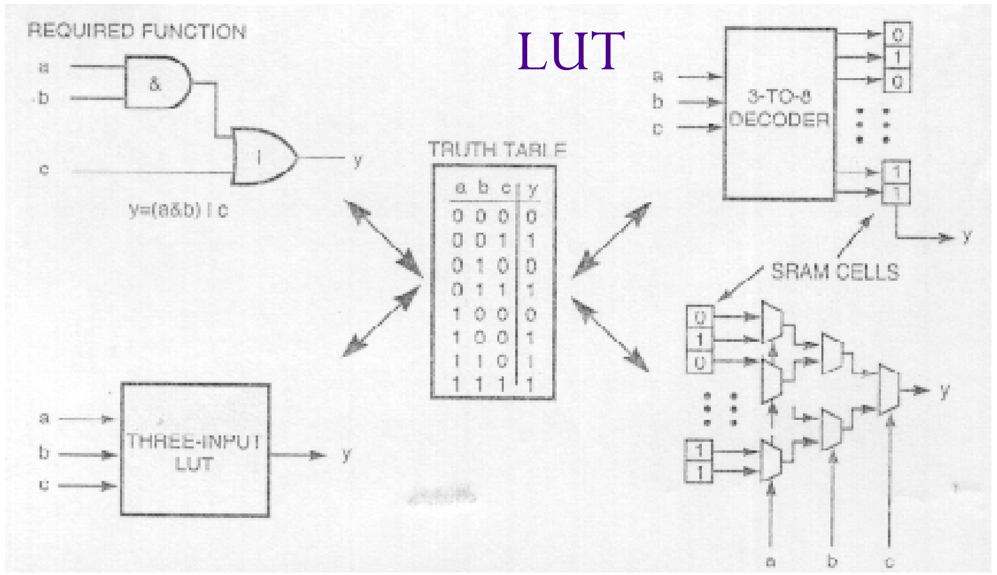
# FPGA Basic Logic Cells

**Include *fixed* amount of combinational logic and registers**

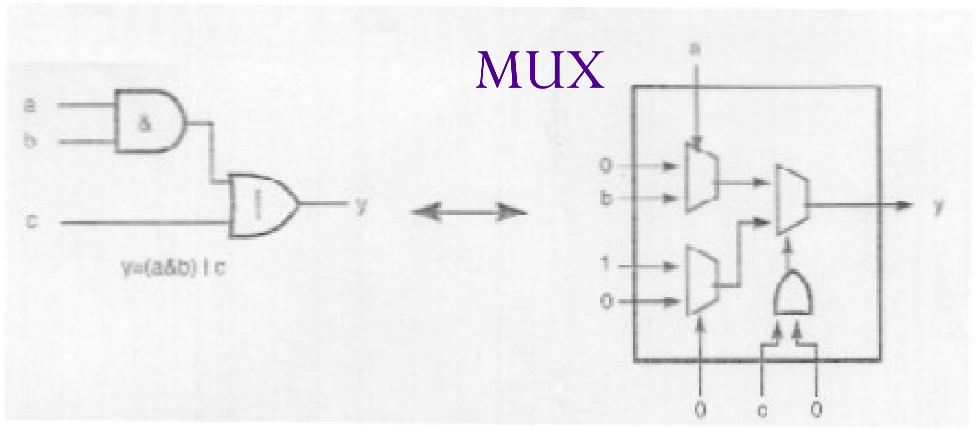
- LUT is the prevailing. Flexible.
- MUX-based structures could also do the trick

**Usually FPGAs contain 1-4 programmable registers per logic cell**

- In some architectures, the LUTs can also be used as tiny memory banks



| a | b | c | y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |



# FPGA Architecture

The logic cells are typically grouped into larger arrays of logic blocks

- Intel: Logic Array Block (LAB)
  - Consists of ALMs (Adaptive logic modules)
- Xilinx Configurable Logic Block (CLB)
  - Consists of *Slices*

And what's best, these names tend to change with every new device and also new terms are introduced…

- ALMs/Slices can be configured in multiple modes
  - For example: 1x 6-input function, 2x 5-input functions, memory, shift register modes, ...
- Check the layout for your device

FPGA architectures differ more and more which makes the direct comparison a bit harder

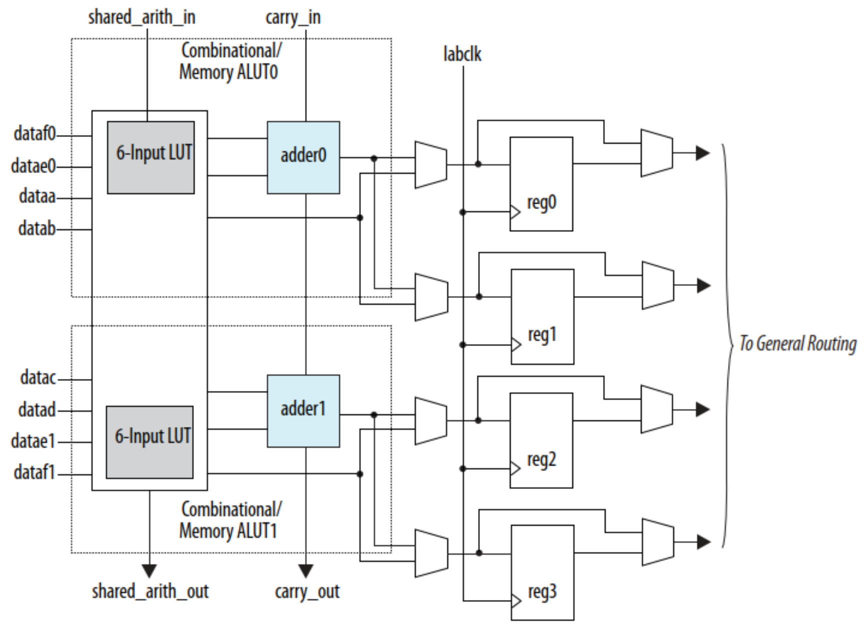
- Always report #LUT, #FFs, memory bits, #MUL from your own design
- Logic Elements (LE) used in marketing, but only a rough estimate

# Altera ALM and AMD Slice

## Intel Arria 10 ALM

- LAB = 10 ALMs

**Figure 1-6: ALM High-Level Block Diagram for Arria 10 Devices**



## AMD Virtex Ultrascale+ simplified Slice

- Slice = 8 of these in parallel
- CLB = 1 slice in Ultrascale

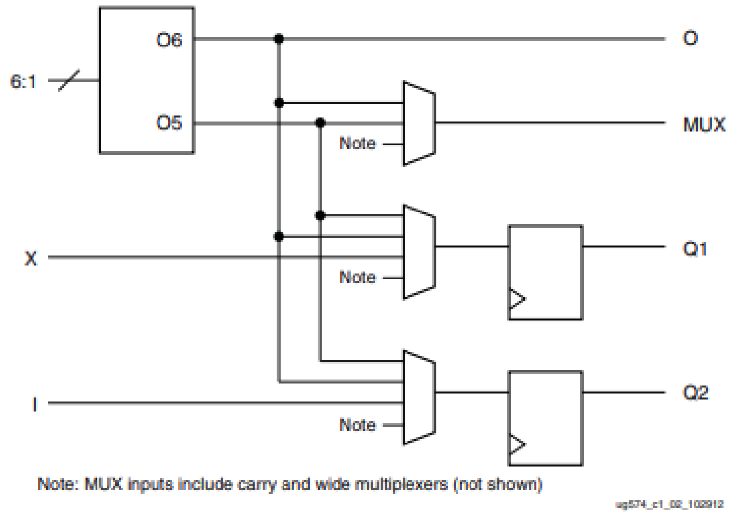
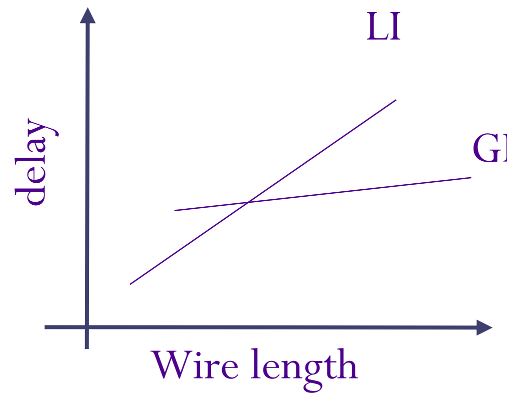

6:1
O6
O5
O
MUX
Note
X
Note
Q1
I
Note
Q2
Note: MUX inputs include carry and wide multiplexers (not shown)
ug574_c1_02_102912

*Figure 1-2:* **A Simplified View of Slice I/O Connecting to a LUT and Storage Elements**

Similar elements, but don’t expect apples to apples comparison in specs

# FPGA Interconnects

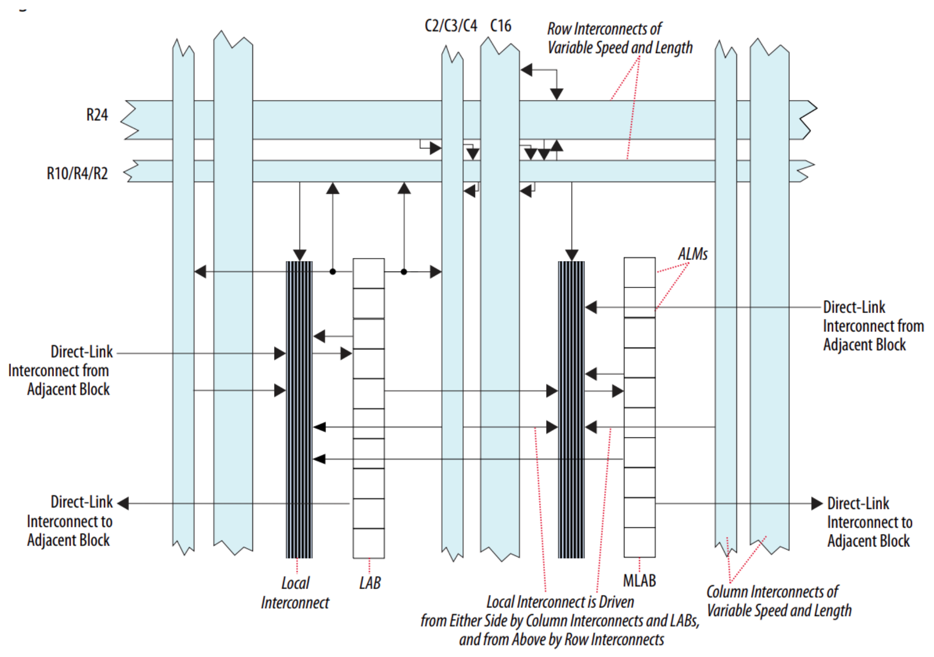


## Hierarchical row+column interconnects

- Local interconnections (LI)
- Global interconnections (GI) (#GI << #LI)
- The different levels are connected with switches

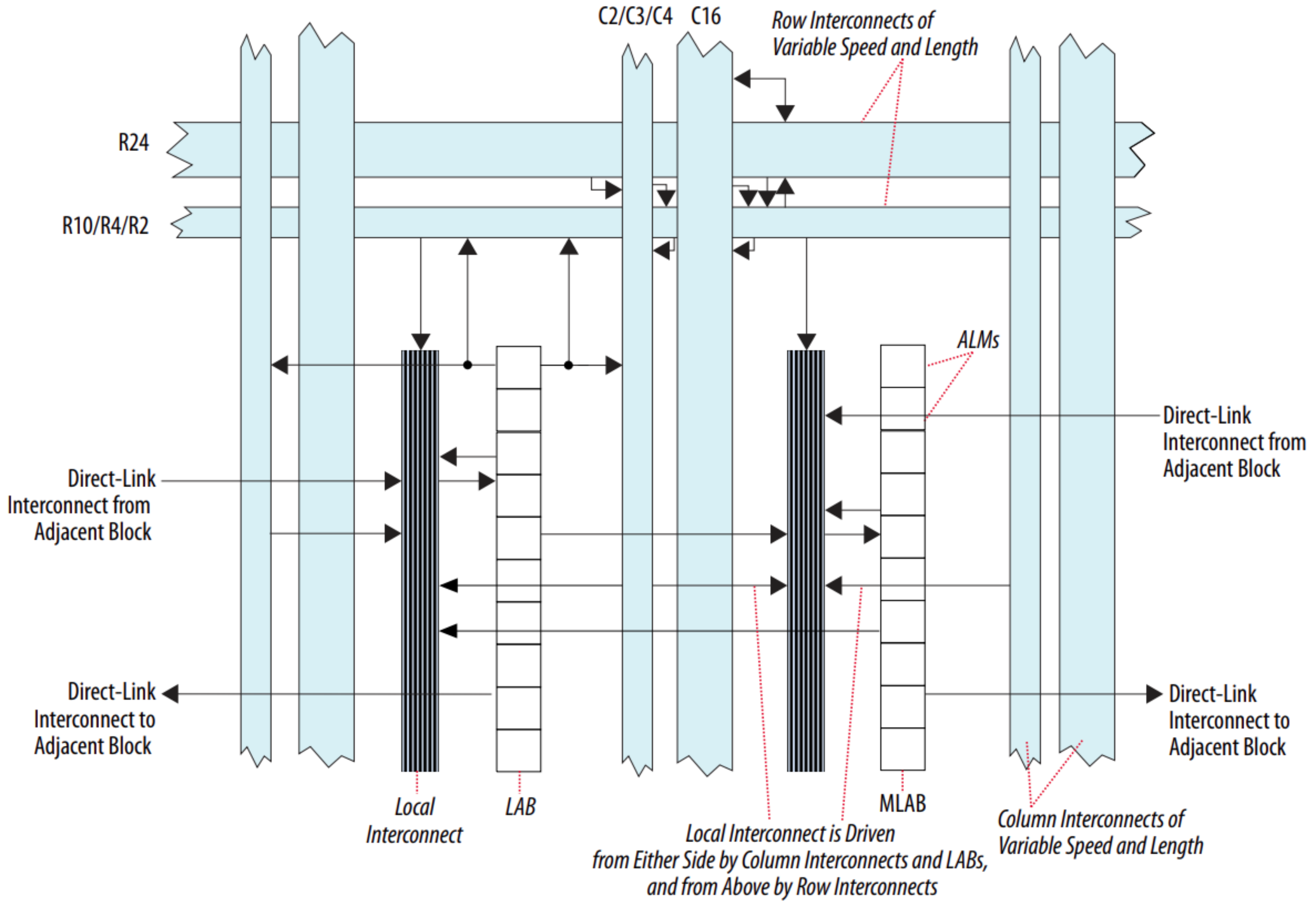
## Local interconnects are shorter than global

- Shorter implies better speed, but GI links are made broader and with more repeaters => faster with fixed length line than LI


C2/C3/C4
C16
Row Interconnects of Variable Speed and Length
R24
R10/R4/R2
ALMs
Direct-Link Interconnect from Adjacent Block
Direct-Link Interconnect from Adjacent Block
Direct-Link Interconnect to Adjacent Block
Direct-Link Interconnect to Adjacent Block
Local Interconnect
LAB
MLAB
Local Interconnect is Driven from Either Side by Column Interconnects and LABs, and from Above by Row Interconnects
Column Interconnects of Variable Speed and Length

## Figure 1. Intel Stratix 10 LAB Structure and Interconnects Overview

This figure shows an overview of the Intel Stratix 10 LAB and MLAB structure with the LAB interconnects.


C2/C3/C4
C16
Row Interconnects of Variable Speed and Length
R24
R10/R4/R2
ALMs
Direct-Link Interconnect from Adjacent Block
Direct-Link Interconnect from Adjacent Block
Direct-Link Interconnect to Adjacent Block
Direct-Link Interconnect to Adjacent Block
Local Interconnect
LAB
MLAB
Local Interconnect is Driven from Either Side by Column Interconnects and LABs, and from Above by Row Interconnects
Column Interconnects of Variable Speed and Length

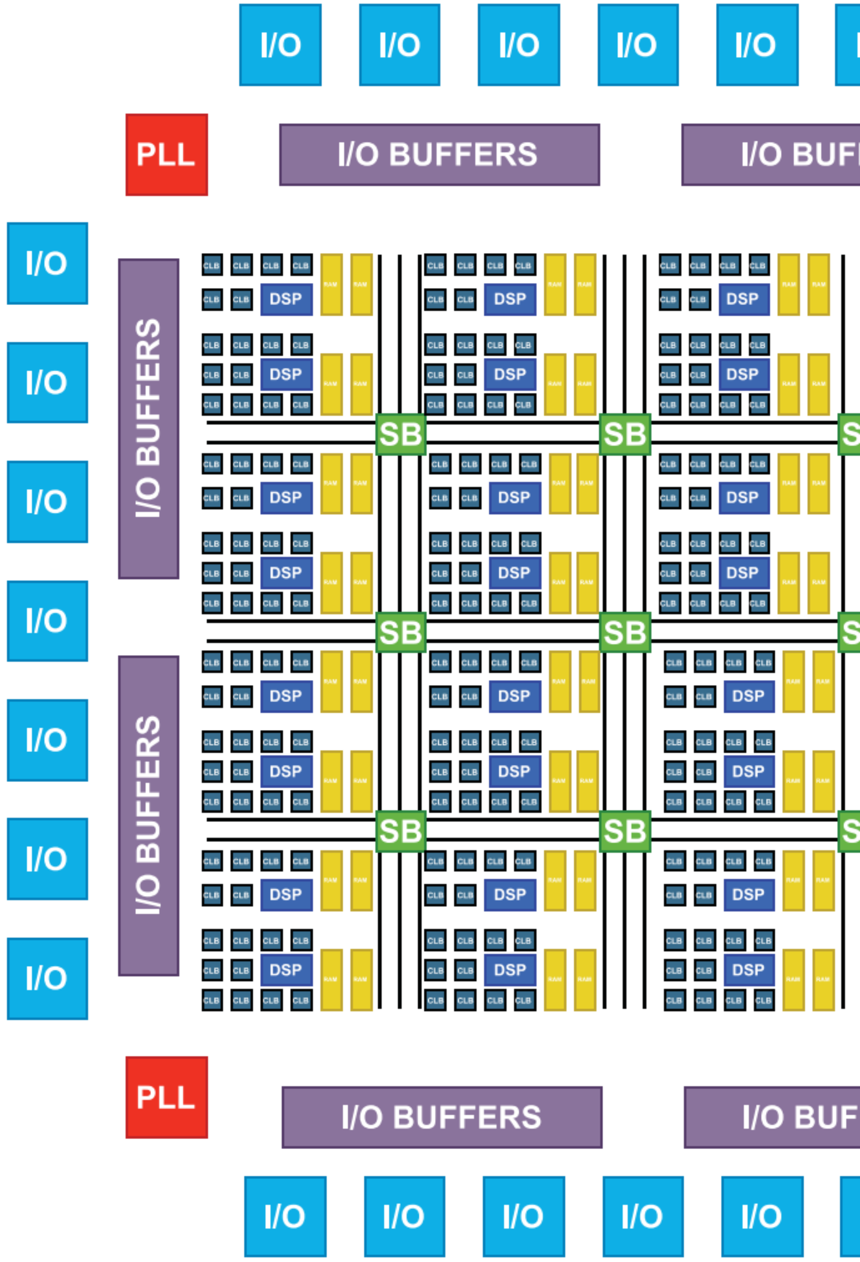
# FPGA I/O

## Most IO pins are programmable

- Direction: input, output, inout
- Other parameters: drive strength, delay…
- Some are dedicated to certain functionality, e.g., clock input, DRAM address line…
- Usually hundreds of pins available

## Divided into banks

- E.g., two banks on each side (topA, topB, leftA, leftB…)
- Banks may have different voltage
- Sometimes restrictions to routing, e.g., each bit of std_logic_vector must be connected to same bank



# Clock Networks in FPGAs

FPGAs are designed for synchronous logic

- This is the case with 99% of FPGAs even if some exotic devices exist

FPGAs include clock networks and support different clock domains within the device

- Clock networks are hierarchical

Global clocks (Gclk)

- Gclks provide a *zero-skew clock* network spanned over the whole chip

Regional clocks (Rclk) (may have several)

- Rclks provide a zero-skew network within some portion of the chip
  - Naturally, all the blocks using given Rclk must reside in same portion

Number of Rclk >> Gclk (e.g., 100 Rclks and 16 Gclks)

# Generating Clocks

The mystical ”clk” signal is generated by:

Input (crystal) oscillator

- Input to the FPGA device (dedicated pins) which buffers the signal
- This can be directly used

DLL/PLL circuitry that multiplies/divides the input clock

- Locks to the required frequency, may provide phase shift
- E.g., create a stable 200 MHz clock from 50 MHz input clock

Internal feedback-loop clocks or clock dividers

- The most hazardous way
- Doable, but **don’t use this**
  - Prone to variations on process, voltage, and temperature
  - Static timing analysis often cannot be used (verification very difficult)
  - Place-and-route may change timing (exact timing cannot be set by tools)
  - The pulse width will change if you migrate to a different device

There are 3 main types of devices

## FPGA CONFIGURATION

# FPGA Devices 1: SRAM-Based

SRAM is used to configure the interconnection switches and LUTs

- Majority of FPGAs nowadays
- Usually implemented with leading-edge technology
- Can be re-programmed arbitrarily many times
  - Ideal for prototyping and rapid development

Since SRAMs lose their contents when powered off, an external device (+non-volatile memory) is required to program them during boot-up

- One concern is security: the device configuration bitstream can be copied during the programming
- Bitstream encryption can prevent this

Manufacturers include AMD, Altera, Lattice, Achronix, QuickLogic, Microchip, ...

# FPGA Devices 2: Antifuse

Need a special in-chip programmer circuitry (may be big), but retain the program during shut-down (non-volatile)

- Fast boot, good security, low power
- One-time programmable (OTP) only

No need for external circuitry

They are *rad-hard* (quite immune to radiation effects)

- Good for, e.g., space applications

Compared to the SRAM-based with same technology, antifuses have

- Better density (logic gates/mm$^2$)
- Lower interconnect delay – Faster
- BUT! Usually available chips are even several technology generations behind SRAM counterparts due to extra processing steps required
  - Cancels some of the benefits

Manufacturers include, e.g., Microchip, QuickLogic

# FPGA Devices 3: EEPROM/FLASH

Programming is similar to SRAM-based, but non-volatile

- Both re-programmable and fast boot

Good security

EEPROM and FLASH 1-bit cells need two (special) transistors

- Typical 1-bit SRAM implementation requires 6 transistors
- => Smaller cells than in SRAM devices
- Faster, more density

BUT! Also few generations behind the leading edge

Some devices integrate small Flash memory but LUT and wire configuration is done with SRAM

Manufacturers include, e.g., Microchip, AMD

## FPGA Device Technologies: Summary

| Feature | SRAM | Antifuse | E2PROM / FLASH |
|---|---|---|---|
| **Technology node** | State-of-the-art | One or more generations behind | One or more generations behind |
| **Reprogrammable** | Yes (in system) | No | Yes (in-system or offline) |
| **Reprogramming speed (inc. erasing)** | Fast | ---- | 3x slower than SRAM |
| **Volatile (must be programmed on power-up)** | Yes | No | No (but can be if required) |
| **Requires external configuration file** | Yes | No | No |
| **Good for prototyping** | Yes (very good) | No | Yes (reasonable) |
| **Instant-on** | No | Yes | Yes |
| **IP Security** | Acceptable (especially when using bitstream encryption) | Very Good | Very Good |
| **Size of configuration cell** | Large (six transistors) | Very small | Medium-small (two transistors) |
| **Power consumption** | Medium | Low | Medium |
| **Rad Hard** | No | Yes | Not really |

# Configuring a SRAM-Based FPGA

The device needs a *programming file* (also called as bit file or bitstream)

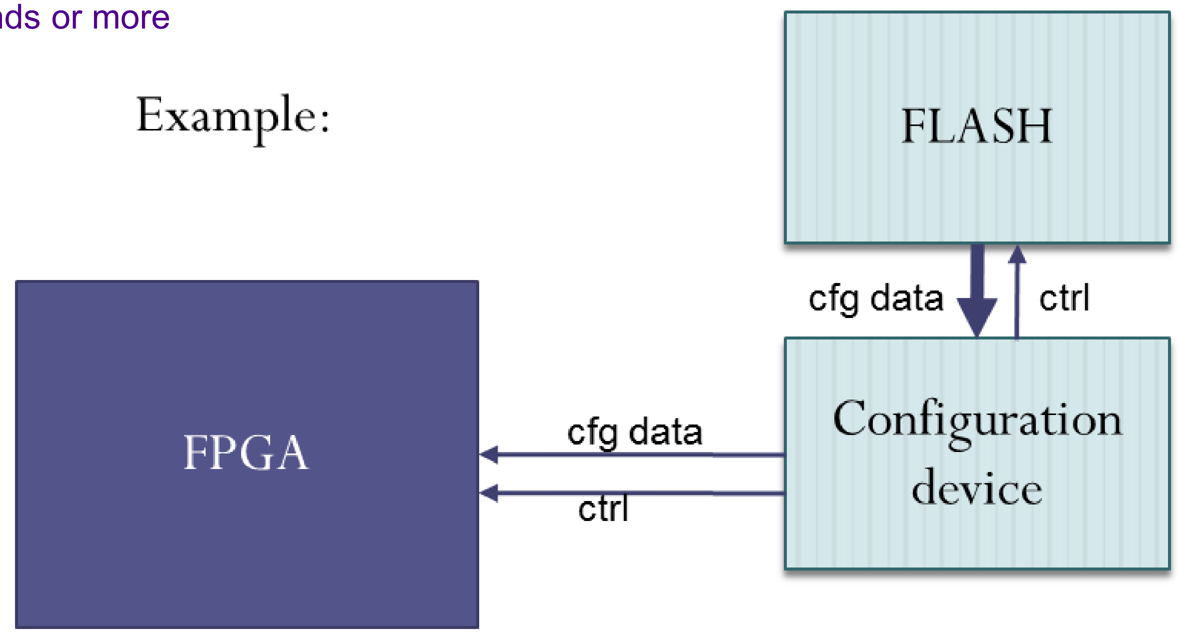
- Includes the programming into for each cell of the FPGA
- Usually proprietary format

Again, a lot of variation across manufacturers and devices

Each cell and interconnection needs to be configured at start-up

- Programming file size from several kilobytes to megabytes
- Takes time in the order of milliseconds or more

Example:



- Stores the bitstream

–Reads the bitstream from the flash
–Writes the bitstream to the FPGA

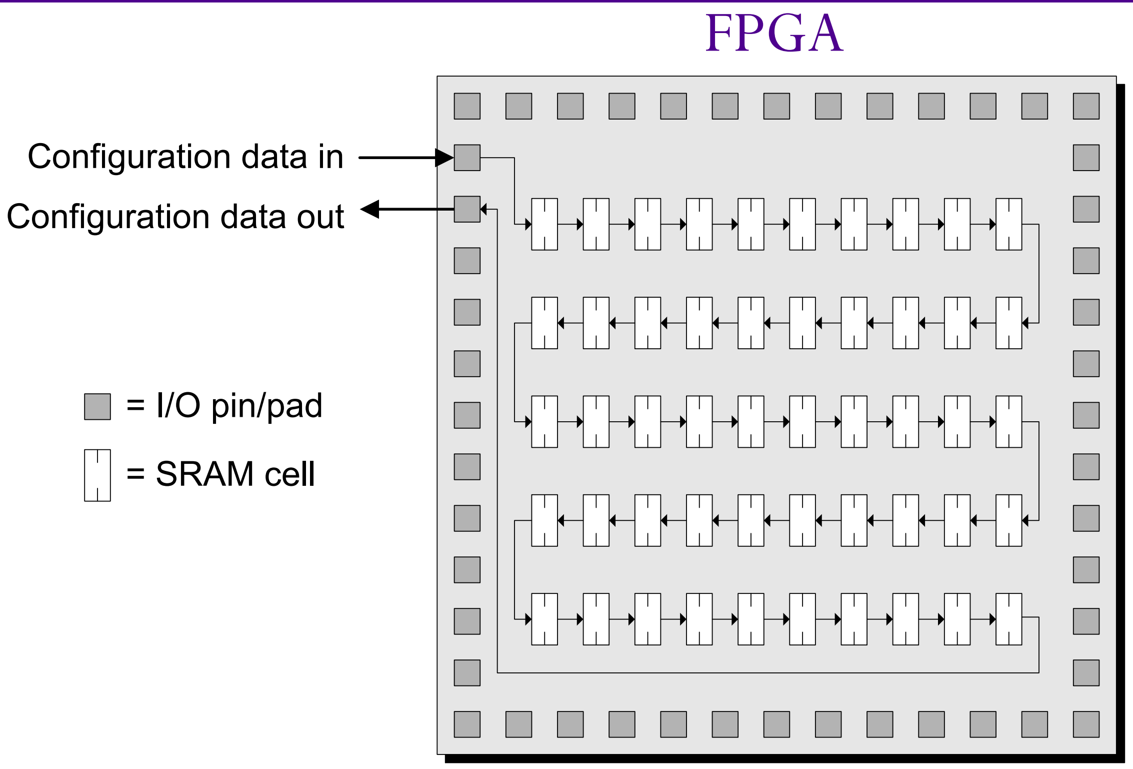
# SRAM-Based FPGA Configuration (2)

Common procedure is to use serial configuration circuit in order to save the PCB area and precious I/O pins

The process can be visualized as a shift register chain of cells and on every clock tick one cell is programmed

- Millions of cells, slow, similar idea as in scan chain

The internal implementation of the ”register chain” varies



## SRAM-Based FPGA Configuration (3)

FPGAs may support also *master*-mode in which the FPGA may directly connect to a memory to obtain its configuration bits

- Does not need an external configuration device

Parallel programmers are often used in order to increase the programming speed

- Byte-wide ports are common

JTAG port is also supported

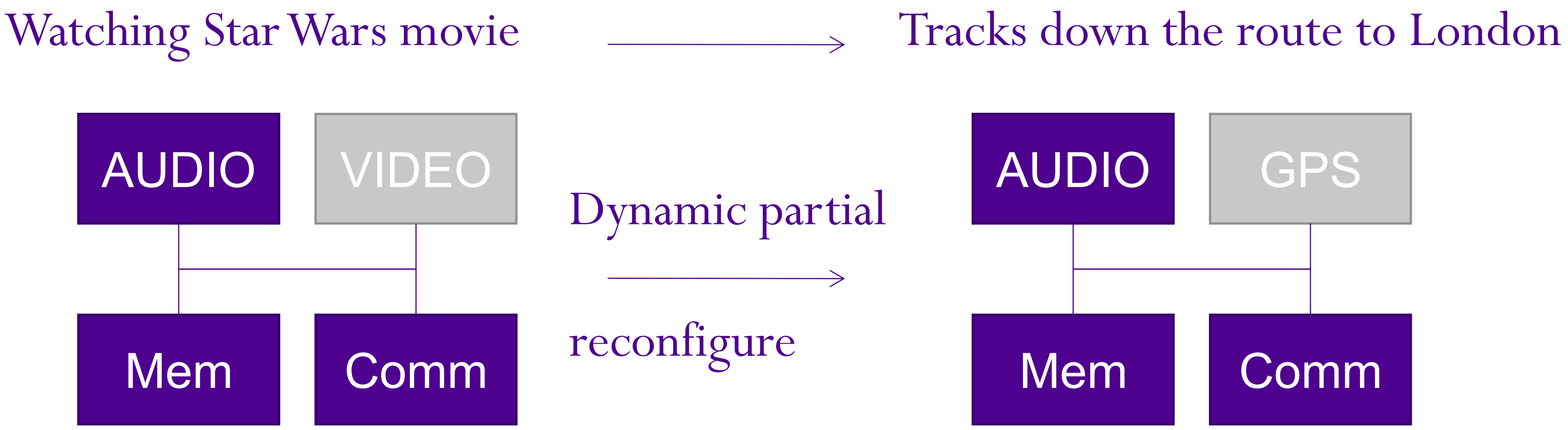
- Standard connection originally for testing
- Has room for special commands
- Widely used in prototyping phase as the FPGA may be directly programmed with JTAG, instead of programming the flash and then reseting the device

# Dynamically Reconfigurable Logic

A design that can be reconfigured on the fly while remaining resident in the system

- For example, during startup FPGA first performs self-testing
- Then, the real configuration is loaded

It would be superb that, e.g., a GPS hardware could be reconfigured to video codec on-the-fly as the user selects that



## Problems in Dynamic Reconfiguration…

The time it takes to reconfigure the logic and interconnects

- They are typically programmed using a serial data stream (or a parallel stream only 8 bits wide)
- Usually at least several milliseconds for partial reconfiguration
- Only some devices support partial reconfiguration (coarse-grain, column-wise)

Design considerations

- Requires manual partitioning and a more complex synthesis flow
- Layouts of reconfigurable parts must be compatible
- The inputs to the PR region have to be halted during configuration
- More non-volatile memory required for bitstream

Reconfigurability was a big fuzz in 1990s and it’s again rising its head

- Still not very common

*Transceiver is a device that has both a* ***trans****mitter and a re****ceiver*** *which are combined and share common circuitry*

# HARD MACROS AND GIGABIT TRANSCEIVERS

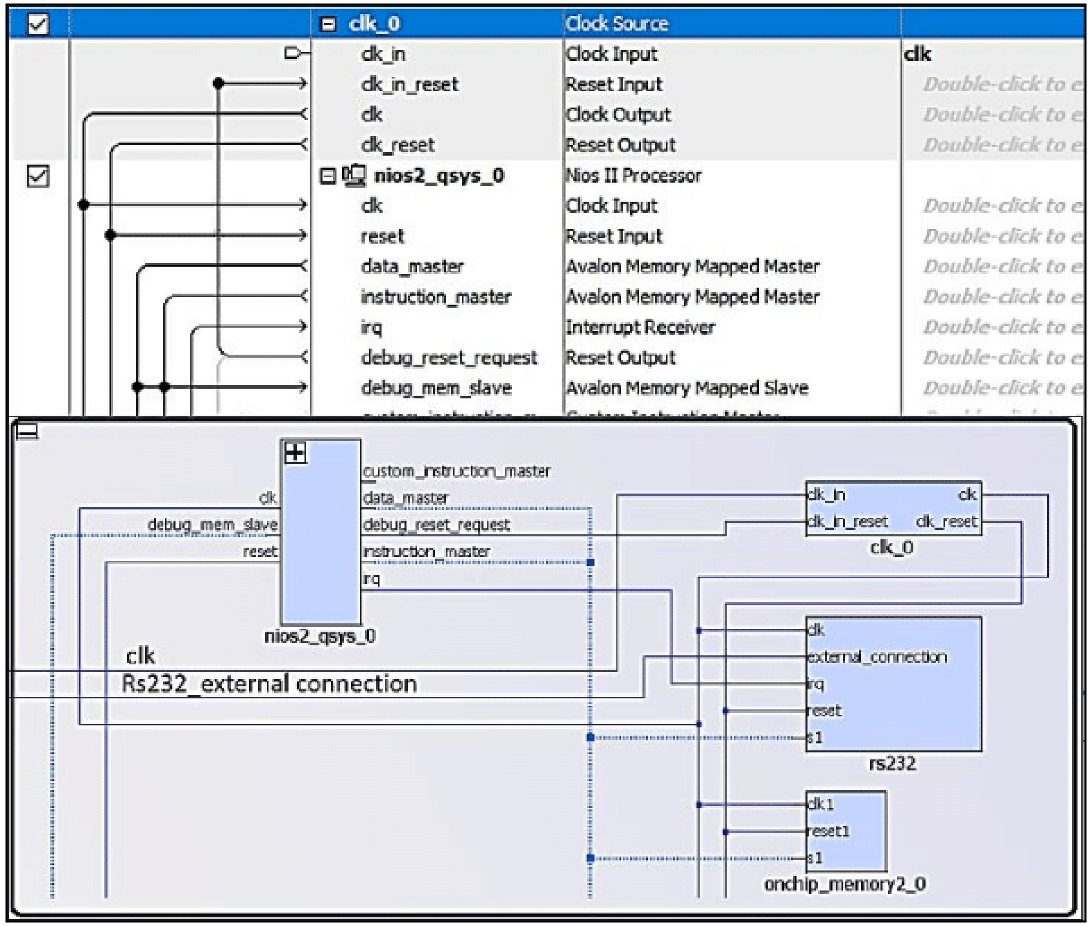
# Soft Macros

## IP that can be placed on the FPGA fabric

- IP catalogs provided by the vendor
  - Configurators included in the synthesis tools
- Commercial/open-source IP

## Soft CPU cores

- AMD/Xilinx Microblaze
- Altera Nios II
  - NIOS V: Altera's custom RISC-V soft core



| | Name | Description | Export |
|---|---|---|---|
| ☑ | ⊟ clk_0 | Clock Source | |
| | clk_in | Clock Input | clk |
| | clk_in_reset | Reset Input | Double-click to e |
| | clk | Clock Output | Double-click to e |
| | clk_reset | Reset Output | Double-click to e |
| ☑ | ⊟ nios2_qsys_0 | Nios II Processor | |
| | clk | Clock Input | Double-click to e |
| | reset | Reset Input | Double-click to e |
| | data_master | Avalon Memory Mapped Master | Double-click to e |
| | instruction_master | Avalon Memory Mapped Master | Double-click to e |
| | irq | Interrupt Receiver | Double-click to e |
| | debug_reset_request | Reset Output | Double-click to e |
| | debug_mem_slave | Avalon Memory Mapped Slave | Double-click to e |

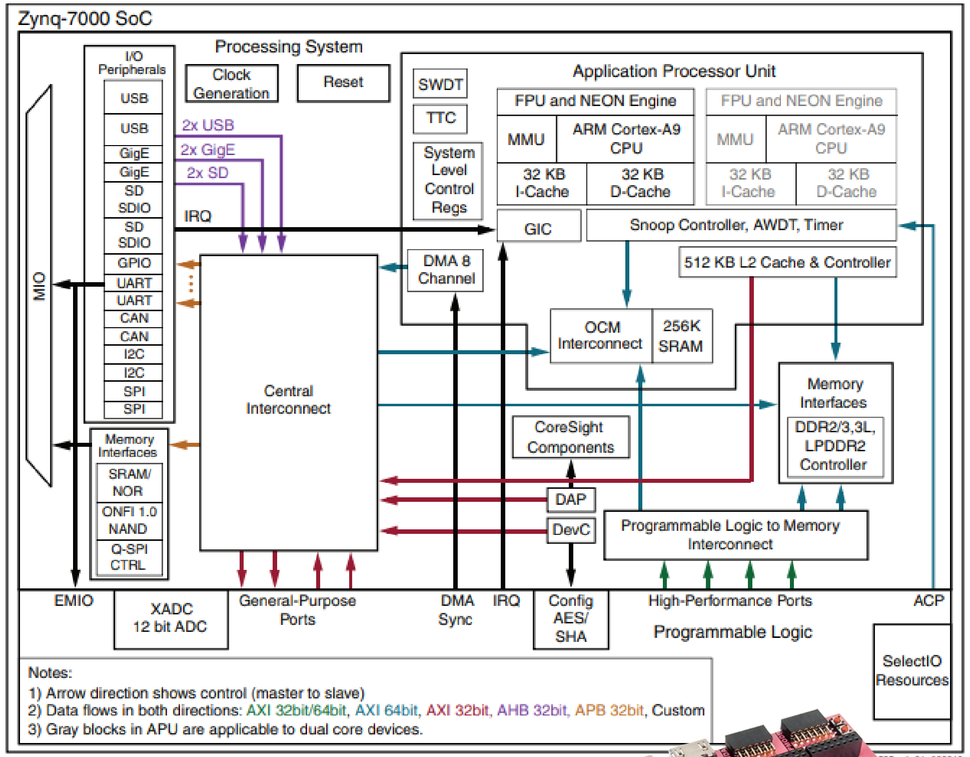
# Integrated Hard Macros

## FPGA devices have an increasing number of integrated hard macros

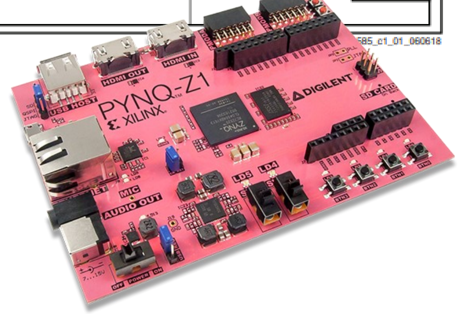
- Not built from LUTs (much faster and smaller)
- Included in each device despite of usage ➔ everyone pays
- Includes the most common functions
- Allow some configuration even if they are “hard”

## Usual hard macros

- PLL/DLLs for clock manipulation
- Memories
- High-speed multipliers with accumulate (MAC) (DSP)
- High speed I/O link controllers
- Integrated microprocessors (e.g., ARM, RISC-V) -> “SoC FPGA”

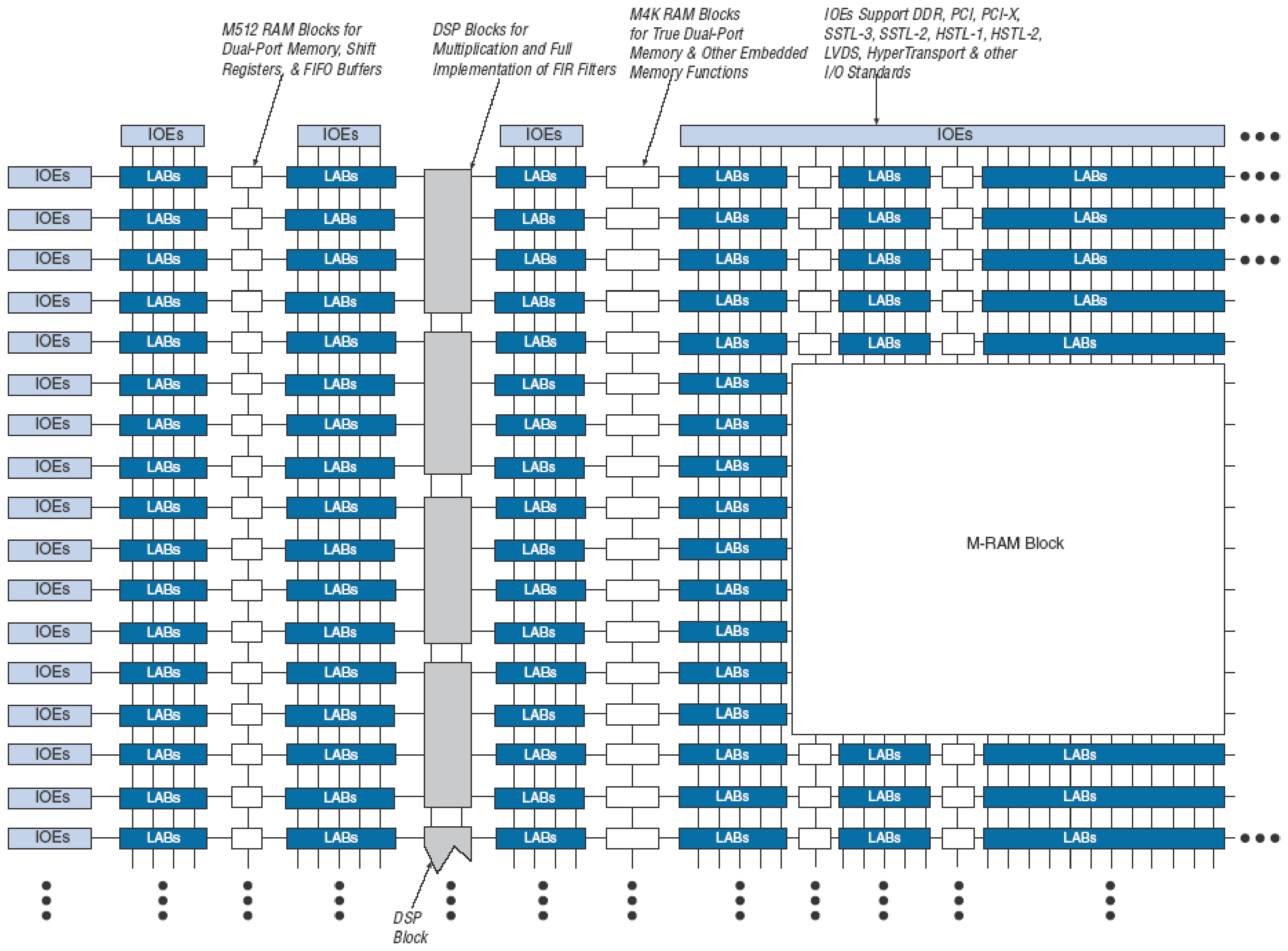


Notes:
1) Arrow direction shows control (master to slave)
2) Data flows in both directions: AXI 32bit/64bit, AXI 64bit, AXI 32bit, AHB 32bit, APB 32bit, Custom
3) Gray blocks in APU are applicable to dual core devices.



## Integrated Hard Macros (2)

*Figure 2–1. Stratix II Block Diagram*


M512 RAM Blocks for Dual-Port Memory, Shift Registers, & FIFO Buffers
DSP Blocks for Multiplication and Full Implementation of FIR Filters
M4K RAM Blocks for True Dual-Port Memory & Other Embedded Memory Functions
IOEs Support DDR, PCI, PCI-X, SSTL-3, SSTL-2, HSTL-1, HSTL-2, LVDS, HyperTransport & other I/O Standards
IOEs
LABs
M-RAM Block
DSP Block

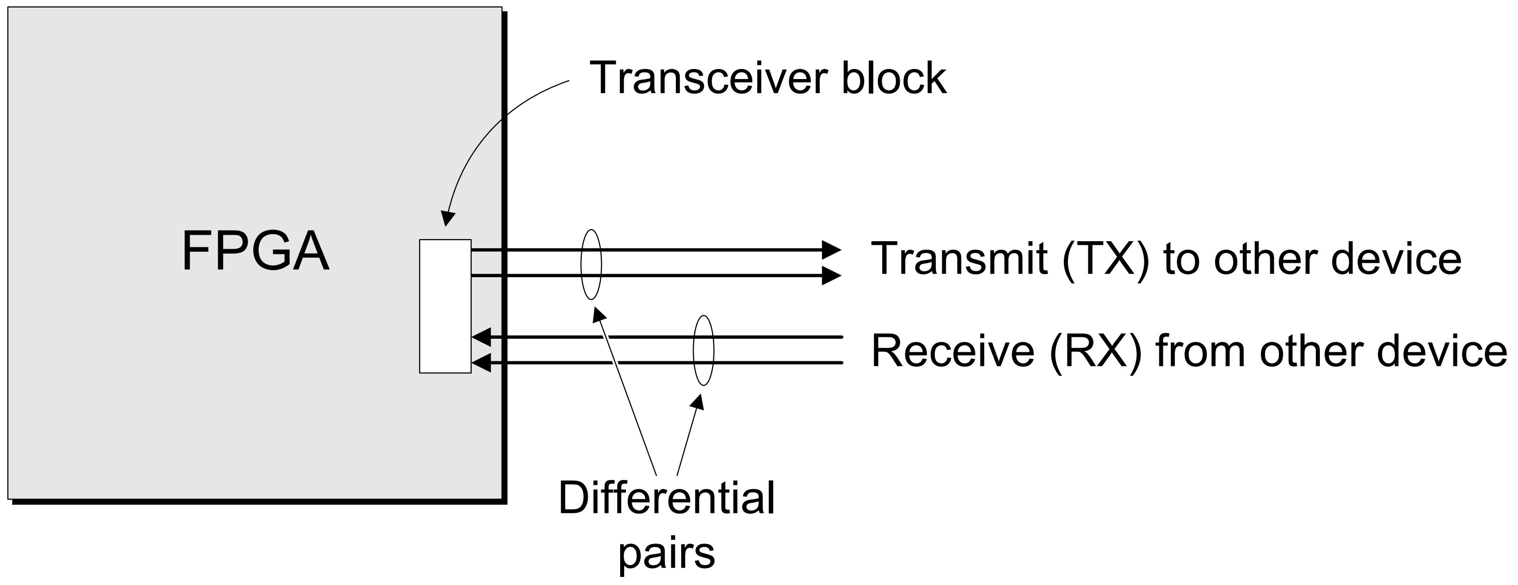
# Basics for I/O Transceivers

**Parallel buses were the prevailing data transmission type, but high-speed parallel wiring is very hard to manage**

- Signal integrity issues (crosstalk, susceptibility to noise etc., track length on PCB)

**Serial communication simplifies many things**

- Unidirectional point-to-point links, only two devices instead of multi-master (compare to shared bus)
- Necessitates higher frequency than parallel communication

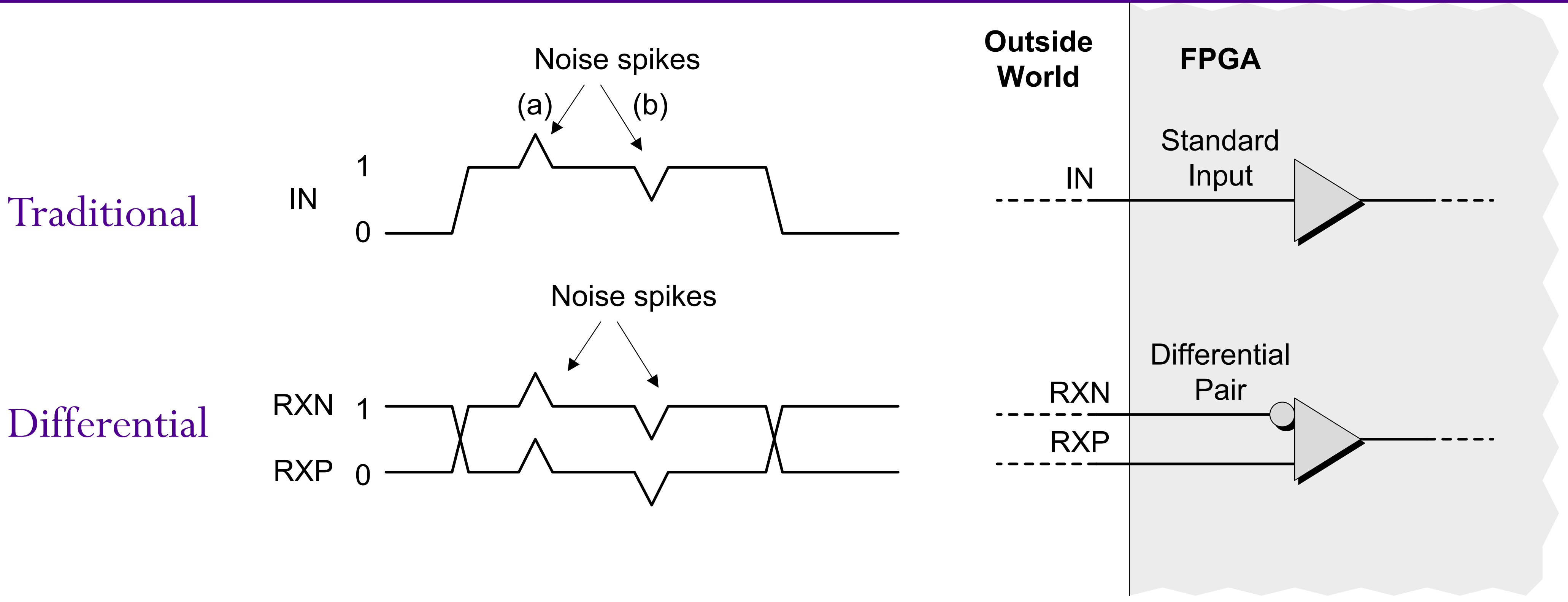


# Differential Signaling

Only the difference between the signal levels matter

- Always carry complementary values

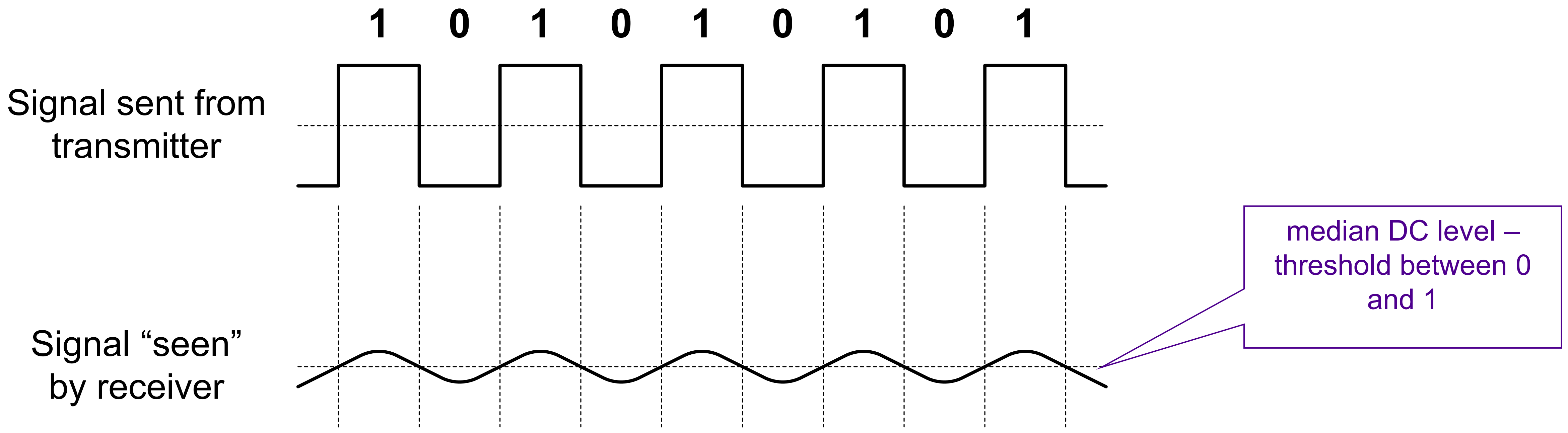
If the tracks are close to each other, noise will affect both lines similarly → the difference stays the same



# Data Transmission

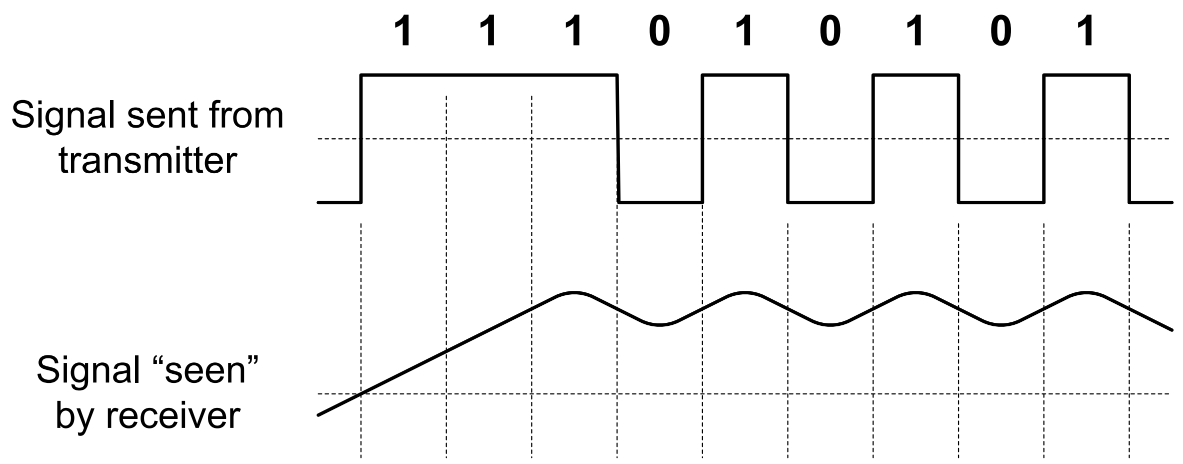
When transferring signals with data rates of gigabits per second, the circuit board and its tracks absorb a lot of the high-frequency content of the signal

The receiver only gets to see a drastically attenuated version of that signal


1 0 1 0 1 0 1 0 1
Signal sent from transmitter
Signal "seen" by receiver
median DC level – threshold between 0 and 1

# Data Transmission (2)

Problems arise if you send data that have, e.g., multiple 1’s in the beginning (overly pessimistic example)



- Received signal level constantly over the threshold
- Receiver only sees a chain of 1’s!
- => encoding standards, e.g., 8/10, we send 2 extra bits for each 8 bits to ensure that there is no more than five 0’s or 1’s in a row
- 1/5 of the bandwidth is wasted but DC level will be correct
- Another choice would be lower frequency

## Signaling Standards

Electronics wouldn’t be electronics if there weren’t variety of standards for this sort of thing:

- Fibre Channel
- InfiniBand
- PCI Express
- 10-gigabit Ethernet
- and others…

An embedded FPGA transceiver can typically be configured to support some of these (but not all)

## Integrated Gigabit Transceivers

E.g., Stratix 10 (released 2016) supports speeds up to 28.3 Gbps

- AMD Versal ACAP GTM (2022): 112 Gbps

However, we can group a set of transceivers so we can further improve the data rate

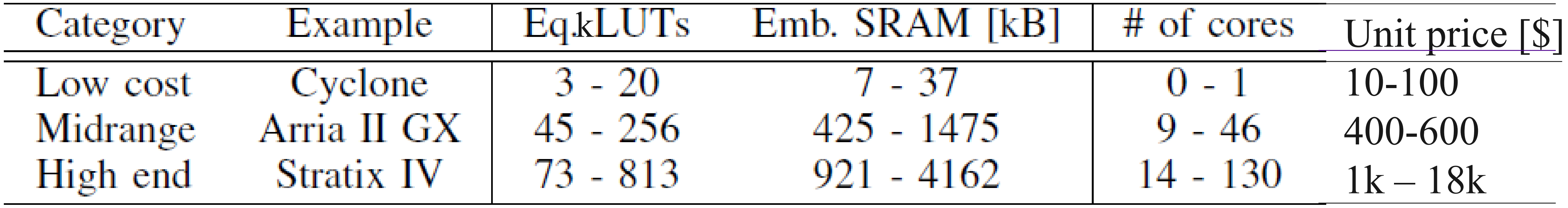
- Using 4 transceivers would result in, e.g., 100 Gbps speed
- Extra logic required to pack and unpack the data being send from device to device

One should try to utilize the FPGA board's capabilities as much as possible instead of developing own proprietary solutions

Sidenote: e.g., 10 Gbps serial link => data transfer rate 10 GHz = 0.1 ns period

- Speed of light is 299,792,458 m/s. Light traverses **3 cm** during one period, electrons somewhat less…

TABLE I: Approximation on how many $\overbrace{\text{cores}}^{\text{TTA}}$ could fit on a single FPGA chip.



| Category | Example | Eq.kLUTs | Emb. SRAM [kB] | # of cores | Unit price [$] |
|---|---|---|---|---|---|
| Low cost | Cyclone | 3 - 20 | 7 - 37 | 0 - 1 | 10-100 |
| Midrange | Arria II GX | 45 - 256 | 425 - 1475 | 9 - 46 | 400-600 |
| High end | Stratix IV | 73 - 813 | 921 - 4162 | 14 - 130 | 1k – 18k |

Orig. table: [P. Jääskeläinen,et al. "TCEMC: A Co-Design Flow for Application-Specific Multicores", SAMOS XI, July 2011, pp. 85-92]

## FPGA PERFORMANCE AND SELECTION CRITERIA

# Typical Application Domains

➢ASIC:
- Mass products, consumer electronics
- Mobile phones
- Computers
- Smart devices, wearables

➢FPGA:
- Industrial (/military) electronics
- Some consumer products
- Cell phone base stations
- Factory automation
- Internet routers
- Data centers
- ”Glue logic”

Mars rover project used Actel and Xilinx FPGAs

F-16 AN/APG-68 Programmable Signal Radar Processor uses Altera Stratix II

# Tools

You need a simulator, synthesizer, place-and-route, timing analyzer, and programmer

- In practice, also virtual logic analyzer and design viewers (schematic, RTL, technology, chip level) are invaluable

The basic set of tools is provided by the FPGA vendor

- Typically these have sufficient features and are good enough
- Most of all, they're cheap!
- Targeted to the specific FPGA devices
- Also include IP libraries and configuration for hard and soft macros

Development boards can be obtained fairly cheaply (~few hundred to few thousand $)

Open-source alternatives exist

- Project IceStorm: Lattice iCE40 FPGA bitstream reverse-engineered

## Design Performance: Speed

Total delay in an FPGA is sum of three factors:

- Delay from FF clock to FF Q (constant)
- Interconnect delay
- Logic cell delay (LUT)

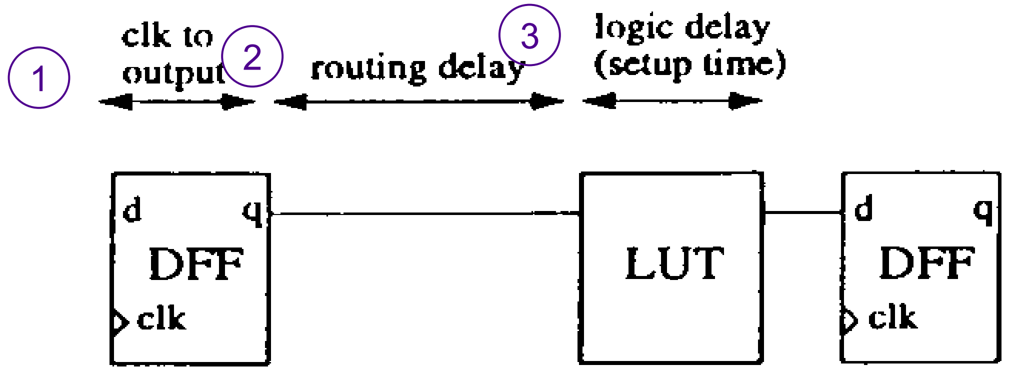
Interconnect delay and #LUTs in path vary depending on logic function

Interconnect delay depends on the number of switches in the path (which form the path from source to destination) and the route length

**Typically, routing delay is 60-80% of total delay of critical path!**

Maximum operating frequency of the FPGA (generally)

- Big designs ~100 MHz
- Small designs ~up to 200 MHz
- Note that most SoCs operate above 1 GHz



## Design Performance: Area

- Very dependent on the application
  - FPGA is good for register-heavy designs
- The more area the design takes, more difficult it is to route
  - ➔ smaller clock frequency
- Largest high-end FPGAs can hold very complex architectures, comprising several soft RISC processors and other hardware
  - ”Multi-million ASIC gates”
- Design with small area can be fitted into cheaper FPGA
- 3rd basic measure, power, getting more important

## Separation of Targets

Strong separation between high-end and low-end FPGA devices

### Low-end

- Low cost, lower logic capacity, less memory, less integrated hard macros
- Target is traditional cost-sensitive consumer products and glue-logic domain
- Price from few to tens of euros, cheaper for high quantities

### High-end

- Highly optimized, for:
  - Speed and large capacity – "Traditional" applications
  - Transceivers – RF applications, communications
  - DSPs and Memory – Machine learning, high-performance computing
- Bleeding-edge hard macros: PCIe 5, GDDR6, HBM memories, AES cryptography blocks, Terasample-scale AD/DA converters
- Pricing thousands of euros/device, up to 10k-range for the best

# Characteristics of Few FPGA Families

| Device family | Model | Logic cells (k) | Memory (Mb) | DSP Blocks/Slices | Transceivers | XCVR bandwidth (Gbit/s, MAX) | Node | Year | Extras |
|---|---|---|---|---|---|---|---|---|---|
| Xilinx Artix Ultrascale+ | AU25P | 308 | 10,5 | 1 200 | 12 | 16,3 | 16nm | 2021 | |
| Xilinx Kintex Ultrascale+ | KU19P | 1 843 | 141,8 | 1 080 | 32 | 32,75 | 16nm | 2020 | |
| Xilinx Virtex Ultrascale+ | VU19P | 8 938 | 165,9 | 3 840 | 80 | 32,75 | 16nm | 2016 | |
| Xilinx Virtex Ultrascale+ 58G | VU29P | 3 780 | 454,5 | 12 288 | 80 | 58 | 16nm | 2018 | |
| Xilinx Virtex Ultrascale+ HBM | VU37P | 2 852 | 340,9 | 9 024 | 96 | 32,75 | 16nm | 2018 | 8GB HBM memory |
| Xilinx Spartan 6 | XC6SLX150 | 150 | 4,7 | 180 | 8 | 3,2 | 45nm | 2009 | |
| Xilinx Virtex 7 | XC7V2000T | 1 955 | 46,0 | 2 160 | 36 | 12,5 | 28nm | 2010 | |
| Intel Arria 10 | GX 1150 | 1 150 | 65,7 | 1 518 | 96 | 17,4 | 20nm | 2013 | |
| Intel Stratix 10 | SX 2800 | 2 753 | 244,0 | 5 760 | 96 | 28,3 | 14nm | 2013 | |
| Intel Stratix 10 | GX 10M | 10 200 | 308,0 | 3 456 | 48 | 17,4 | 14nm | 2019 | |
| Intel Agilex 9 | AGRM027 | 2 693 | 190,0 | 8 528 | 32 | 58 | 10nm | 2023 | 64 Gsps ADC/DAC |
| Intel Agilex 7 F-series | AGF027 | 2 693 | 287,0 | 8 528 | 12 | 58 | 10nm | 2019 | |
| Intel Agilex 7 M-series | 039 | 3 851 | 370,00 | 12 300 | 116 | 116 | Intel 7 | 2022 | 16GB HBM memory |
| Achronix Speedster7t | 7t6000 | 2 600 | 384,00 | 1 760 | 64 | 112 | 7nm | 2021 | |

Note: Comparison of LEs and DSPs between vendors not feasible

- General trends and product families can be seen here

# FPGA Device Selection Criteria #1

## Circuit capacity

- Amount of logic elements and registers, logic element size, (routing resources)
- Amount of RAM, types of RAM
- Required hard macros
- I/O signal routing (How the location of an I/O pin affects the routing)

## Number of I/O signals and supported standards

## Pricing

- Unit price in volume production
- Development cost
- Ranges a lot depending on the amount, specific device and package (and the client)
- Prices are subject to rapid changes → long term contracts should be carefully considered
- FPGAs are rather expensive, e.g., 5-150 euros, and cheapest microcontrollers are ~0.95-5 euros

## Temperature range, radiation-hardness

## Power consumption

# FPGA Device Selection Criteria #2

**Programming style**
- Re-programming, flexibility vs. security
- External components required and their price

**Future**
- Availability of the chips in volume and in time
- Compatible pin/package mapping between different flavors of the device

**Voltage levels, inside the chip and for I/O**
- Compatibility with PCB and adequate noise margins

**Circuit speed**
- Basic cell speed, routing speed, routing delay predictability
- Affects only the most high-performance designs

**Global signals – signals that fo to every cell (clk, reset)**
- Clock networks, clock generation inside the chip, dedicated clock I/O pins
- Dedicated global reset pin

**Development environment**
- CAD tools, usability, support

**Packaging (suitability for chosen PCB assembly etc.)**

# Availability and Life Span

The digital CMOS technology develops rapidly

- New devices are introduced faster and faster

The life span of certain device is dictated by its demand

- Widely used devices are more certain to stick around for years
- Very widely used devices may life quite long (even 10-20 years, e.g., Xilinx 7 series)
- The manufacturer may give some guarantees of life span

The old device may be convertible to a new device without modifications

- Package, pins, operating voltage, configuration
- Operating voltage tends to change between technology generations and that causes most of the problems with compatibility

Choosing between different vendors may be complicated. The experience with certain manufacturers devices may be the dominant factor.

- Relying purely on soft, FPGA-vendor-independent, IP cores helps in porting the system to another device

## MODERN FPGA USE CASES

# Microsoft Cloud Acceleration Architecture (2016)

## Cloud-scale, FPGA-based acceleration architecture

- FPGA connected to each server in the datacenter
  - PCIe
  - 40Gbps ethernet

## Each FPGA can provide

- Local acceleration (PCIe)
- Network Acceleration ("Bump-in-the-wire")
- Global acceleration (Ethernet)

## Applications

- **Bing search page ranking**
- Network acceleration
  - Intrusion detection
  - Deep packet inspection
  - Encryption
- Machine learning

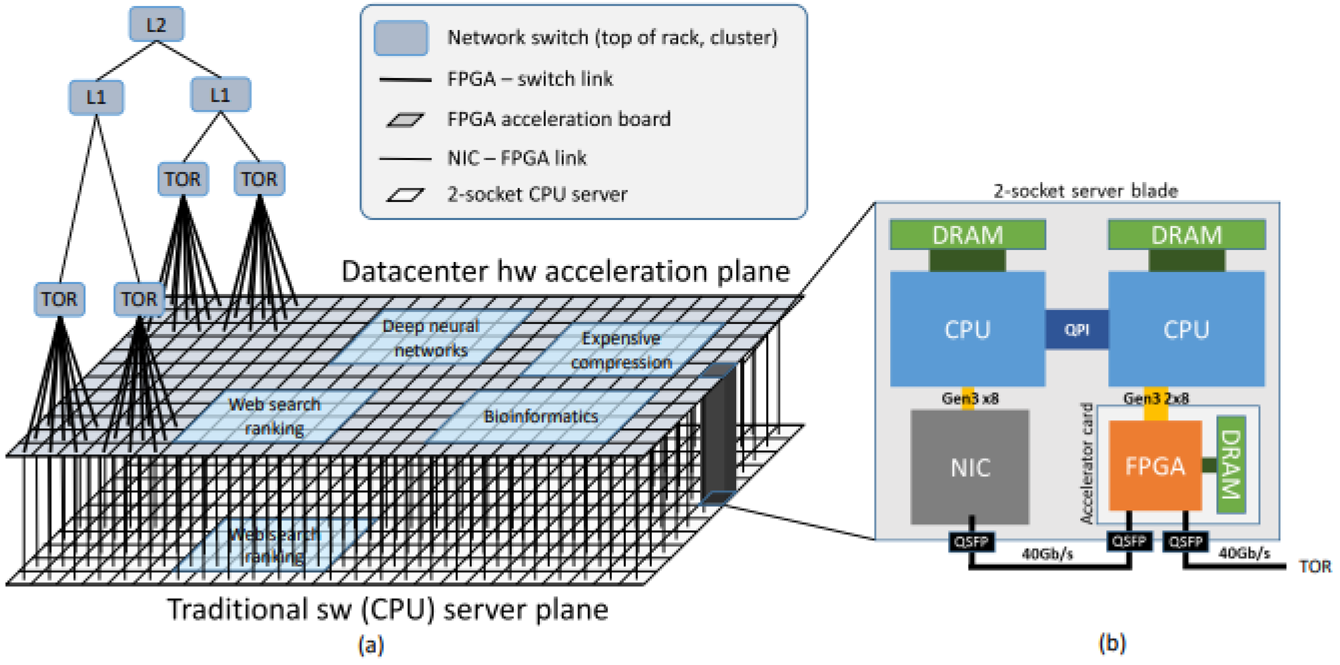


Fig. 1. (a) Decoupled Programmable Hardware Plane, (b) Server + FPGA schematic.

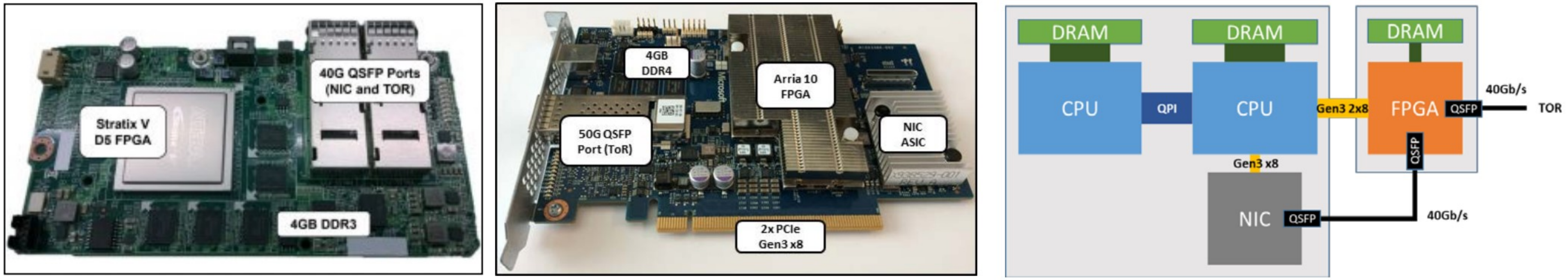
A. M. Caulfield et al. A cloud-scale acceleration architecture. In 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 1–13, Oct 2016.

# Microsoft Azure SmartNICs (2018)

## Microsoft's solution for offloading host networking to hardware

- •Custom FPGA-based SmartNICs
- •"Bump-in-the-wire" architecture

*"We show that FPGAs are the best current platform for offloading our networking stack as ASICs do not provide sufficient programmability, and embedded CPU cores do not provide scalable performance"*



(a) Azure SmartNIC Gen1, 40GbE w/ external NIC (b) Azure SmartNIC Gen2, 50GbE w/ on-board NIC (c) *bump-in-the-wire* architecture

Figure 2: Azure SmartNIC boards with Bump-in-the-Wire Architecture

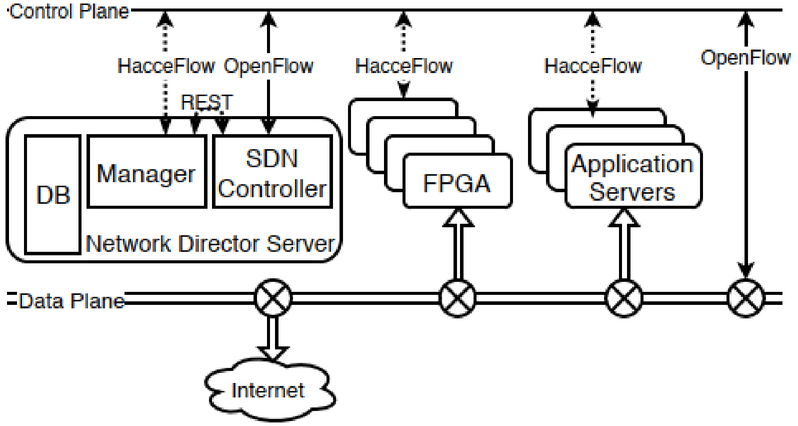
D. Firestone, A. Putnam, et al, "Azure accelerated networking: SmartNICs in the Public Cloud," in 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18). Renton, WA: USENIX Association, 2018, pp. 51–66.

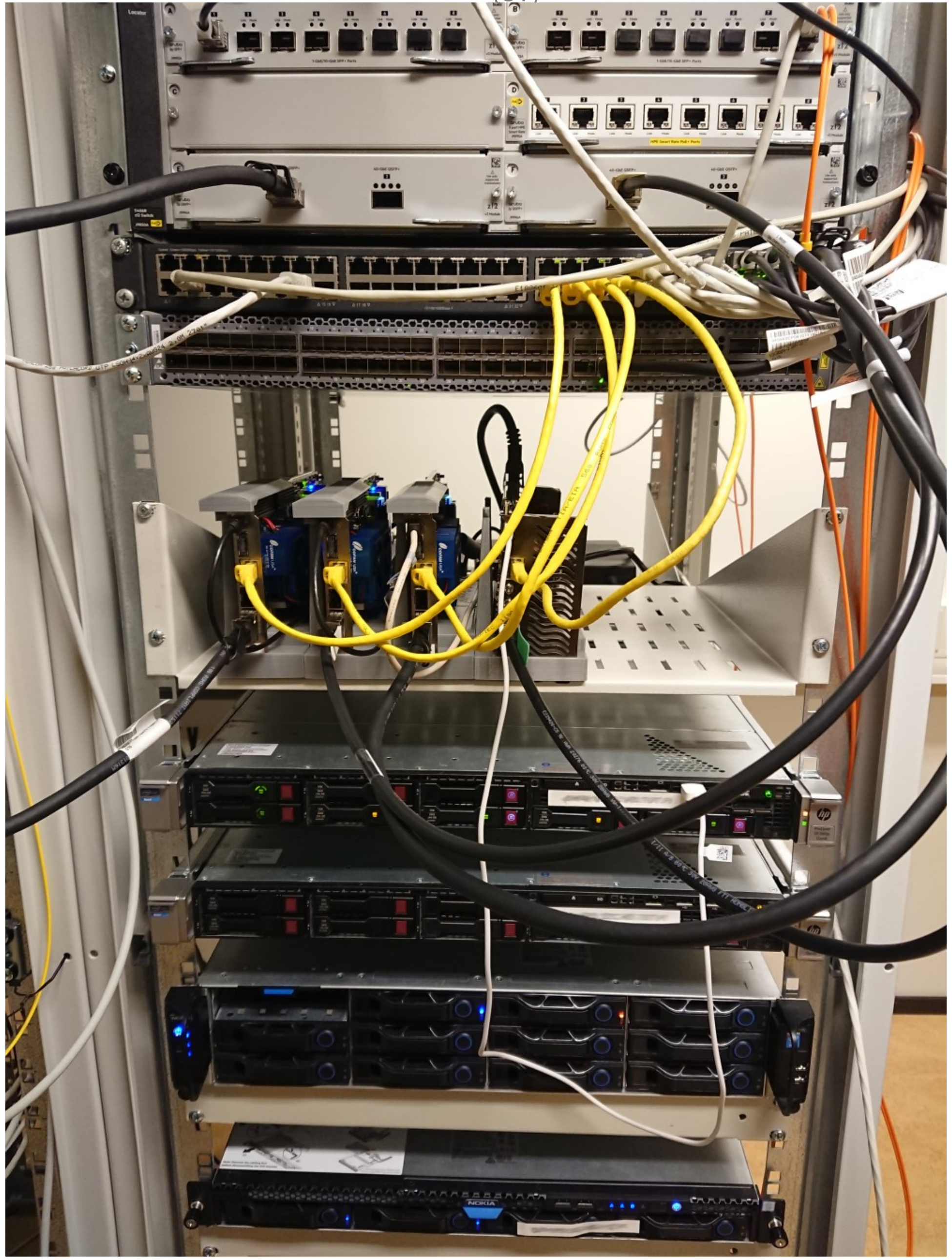
# TUNI FPGA cloud (2017->)

**Target: FPGA accelerated microservices**

- FPGA accelerators as standalone devices in the network
- ➢Scalable acceleration platform

- 40Gbps data plane (Fiber network)
- Software-Defined Networking (SDN)
- Standard servers
- Intel Arria 10 and Stratix 10 FPGAs





# TUNI FPGA cloud (2017->)

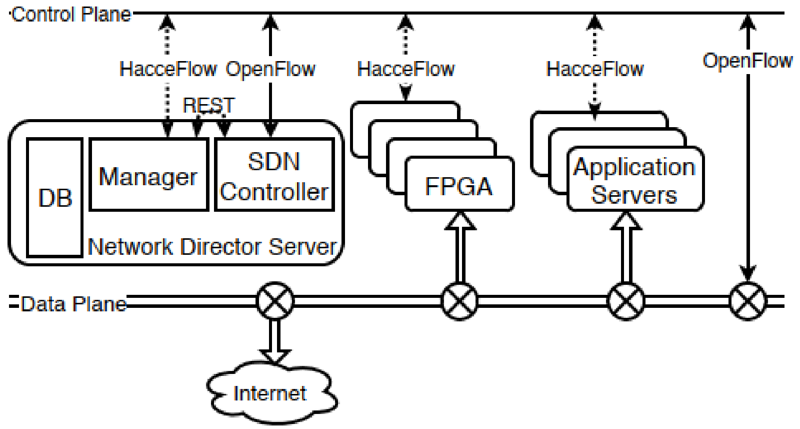
## Target: FPGA accelerated microservices

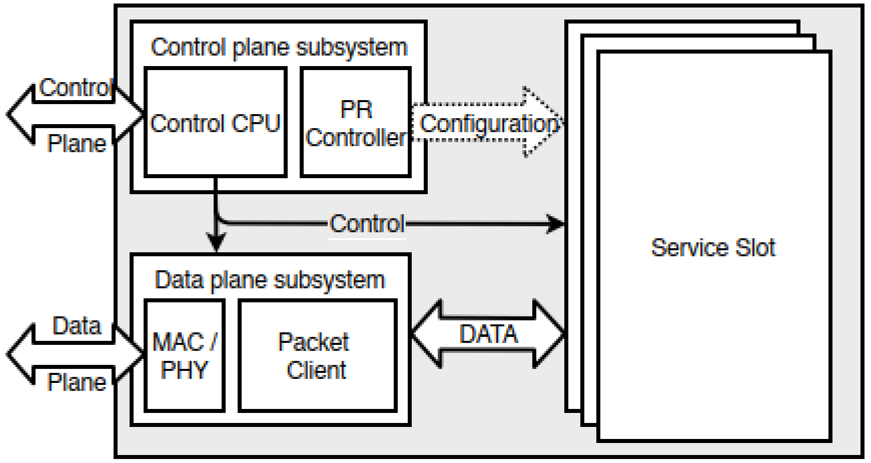
- FPGA accelerators as standalone devices in the network
  ➢Scalable acceleration platform

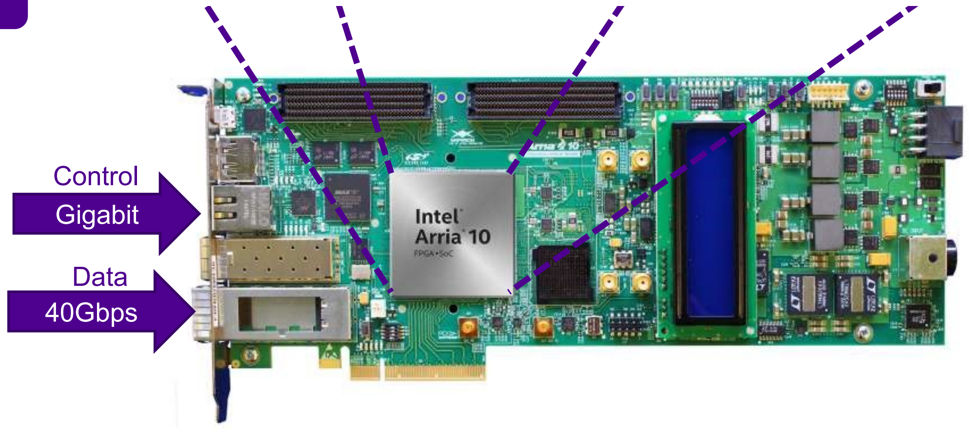
- 40Gbps data plane (Fiber network)
- Software-Defined Networking (SDN)
- Standard servers
- Intel Arria 10 and Stratix 10 FPGAs

## Partial Reconfiguration allows service remapping

- IPsec VPN accelerator
- HEVC video encoder
- AI
- Scientific calculation
- ...







# FPGA Accelerated IPsec

## Feasibility study for IPsec AES algorithm acceleration

- Software-Defined Networking (SDN) reroutes IPsec packets to FPGAs
- AES performed on FPGA instead of the IPsec server
- A standard server can achieve **1-2Gbps** with latency of **200µs**

## Our implementation

- Architecture capable of hosting 1000 VPN tunnels
- One FPGA capable of **10Gbps** with latency below **10µs**

**Feasibility of FPGA accelerated IPsec on cloud**
M. Vajaranta, A. Oinonen, T. D. Hämäläinen, V. Viitamäki, J. Markunmäki and A. Kulmala

*Microprocessors and Microsystems - Embedded Hardware Design.* Ed. by P. Kitsos. 2019, 102861
DOI: 10.1016/j.micpro.2019.102861

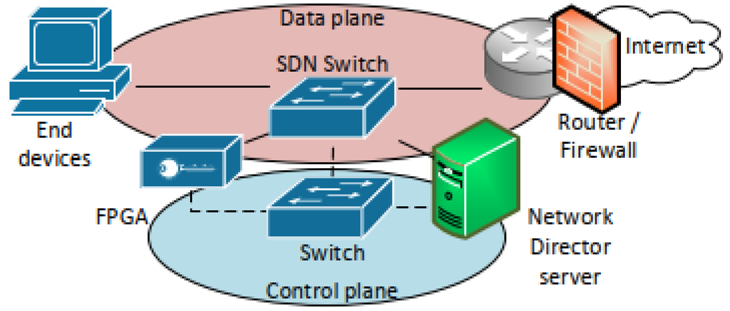


Figure 1: The overall network architecture.

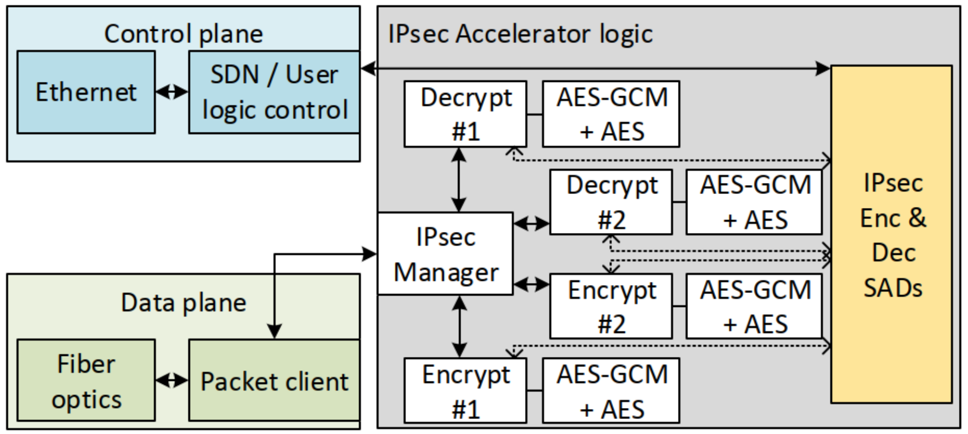


Figure 5: Architecture of the proposed IPsec FPGA implementation.

# Cloud HEVC Encoder

## FPGA-accelerated HEVC

- A cloud setup for video encoding
- The encoder software runs on a server that finds an available FPGA for acceleration
- Dynamic device allocation

- Kvazaar HEVC encoder

➢High-level synthesis

## Performance

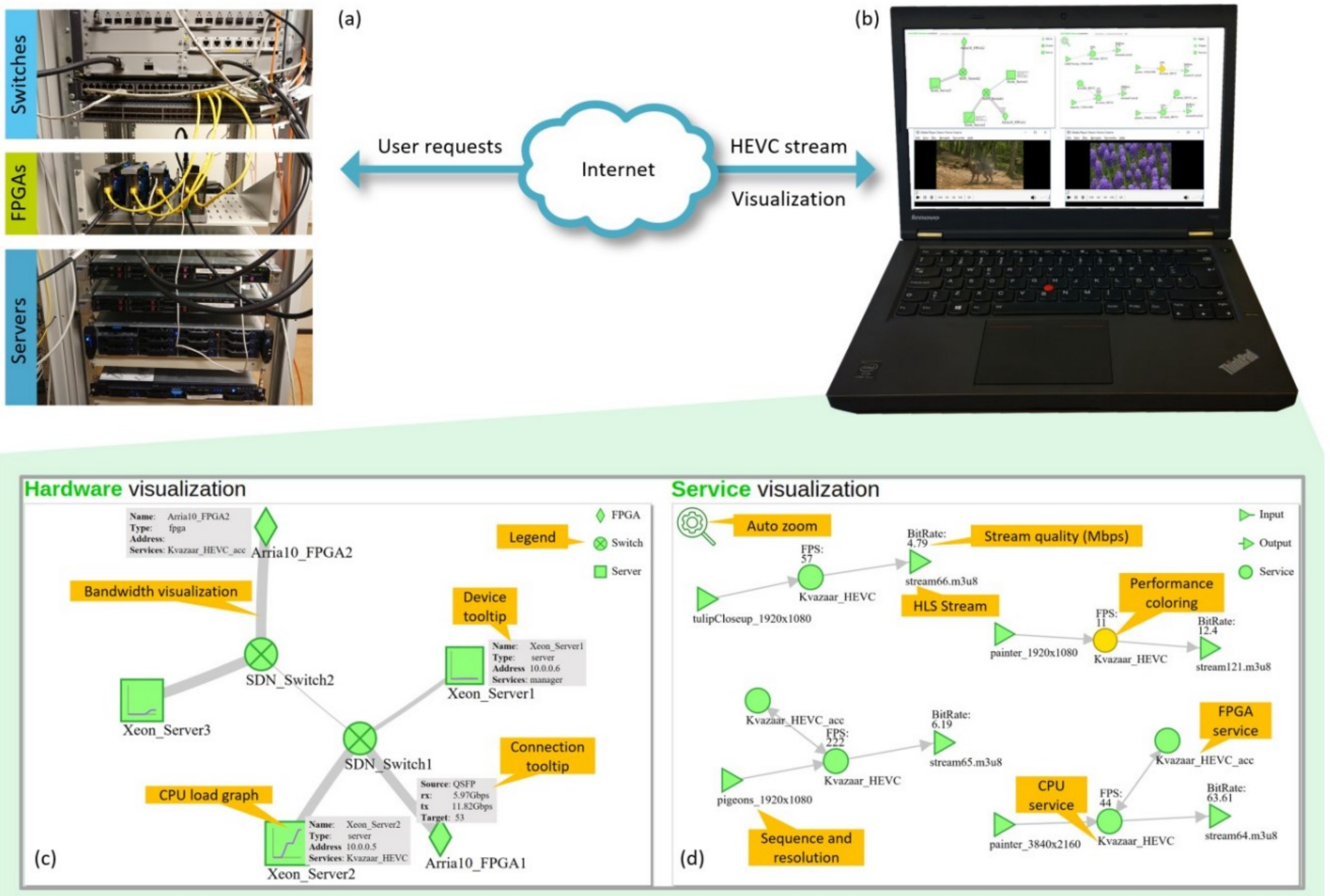
- One server + 2 FPGAs capable of 4k @ 90 fps
- Bottleneck: 2x10Gbps network card in the server

Dynamic resource allocation for HEVC encoding in FPGA-accelerated SDN cloud

P. Sjövall, A. Oinonen, M. Teuho, J. Vanne, and T. D. Hämäläinen

In *Proceedings of IEEE Nordic Circuits and Systems Conference*, Helsinki, Finland, Oct. 2019

DOI: 10.1109/NORCHIP.2019.8906940



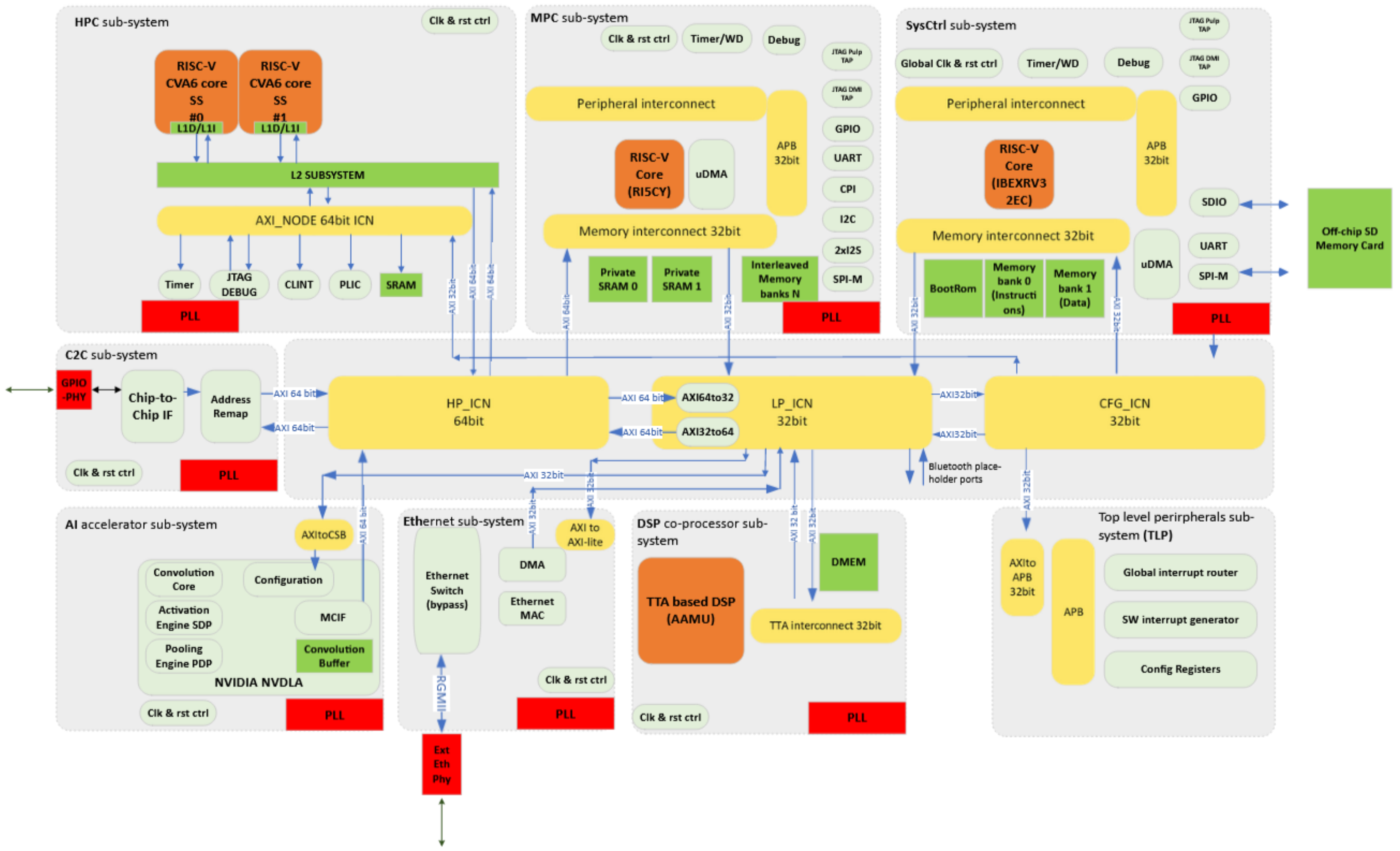
# Ballast FPGA Prototyping (SoC Hub)

## The verification problem

- Verifying the whole SoC through simulation is not very efficient
  - ➢Accuracy of external peripheral simulation models?
  - ➢The runtime for CPU boot process
    - If the CPU boot from SD card takes seconds and the simulator can run 10ms / hour...

## Early SW development

- Software developers don't want to spend their time with simulators



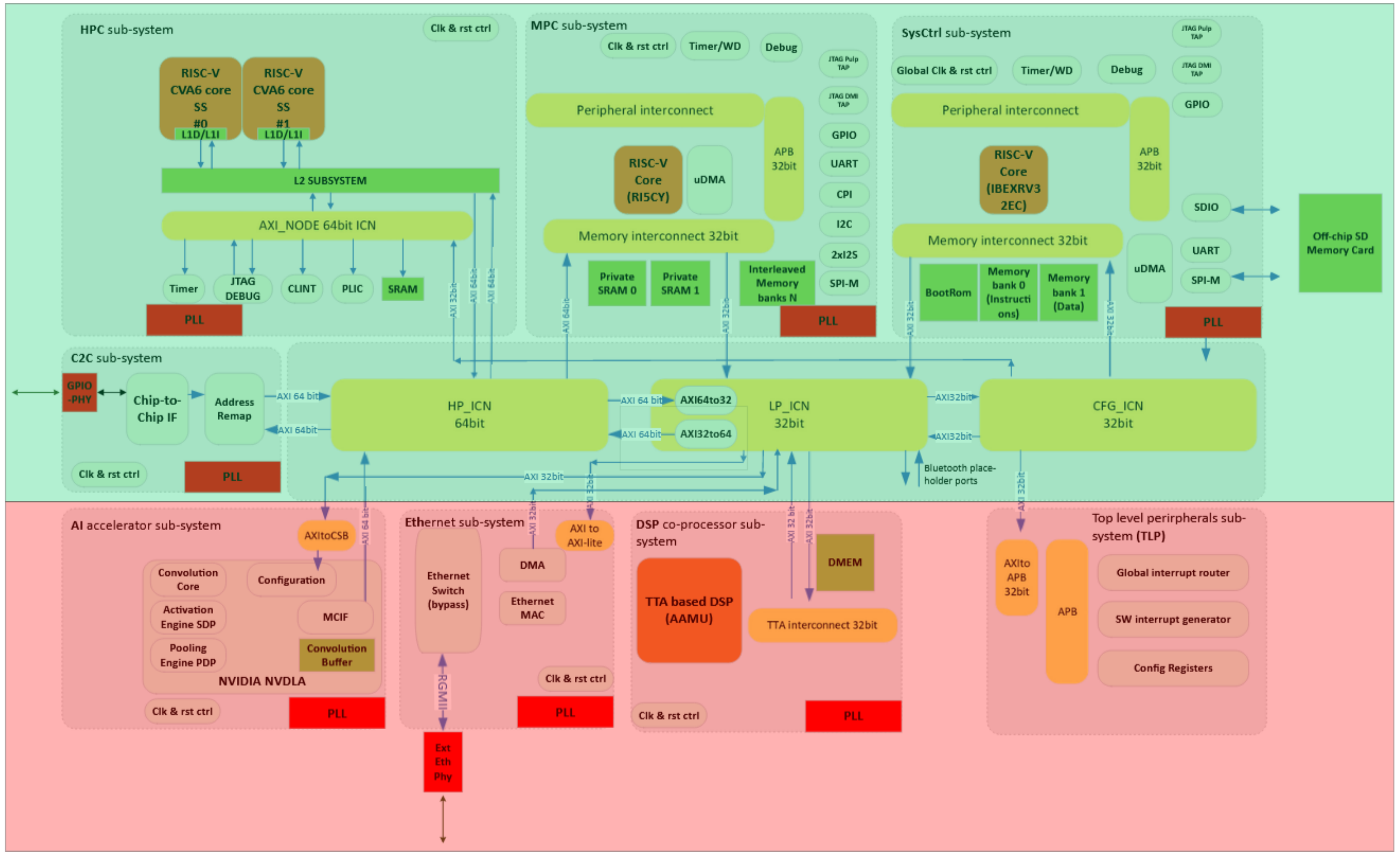
# Ballast FPGA Prototyping (SoC Hub)

## Ballast prototyping targets

- Validate boot design
- Validate the debugging interface
- Validate peripheral functionality
  - ➢Camera
  - ➢SPI
  - ➢I2C
  - ➢C2C subsystem
- Provide a platform for SW development
  - ➢BSP development as early as possible
  - ➢Prepare for wake-up activities

## Design partitioning

- Validate boot design

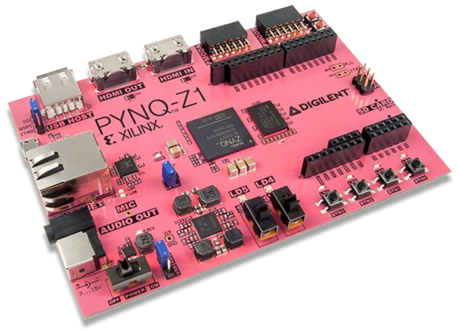


# Prototyping Platforms

Important to pick a platform with sufficient capacity and interfaces for prototyping.

Three Platforms chosen

- Xilinx VCU118
  - Virtex Ultrascale+
  - Large capacity (2.5M Logic Cells).
- Xilinx ZCU104 MPSoC
  - Zynq Ultrascale+
  - Medium capacity (500K Logic Cells).
- Contains embedded quad-core Arm Cortex A53
- Xilinx Pynq-Z1
  - Zynq-7000
  - Small capacity (13.3K Logic Cells).
  - Contains embedded dual-core Arm Cortex A9
  - A lot of I/O



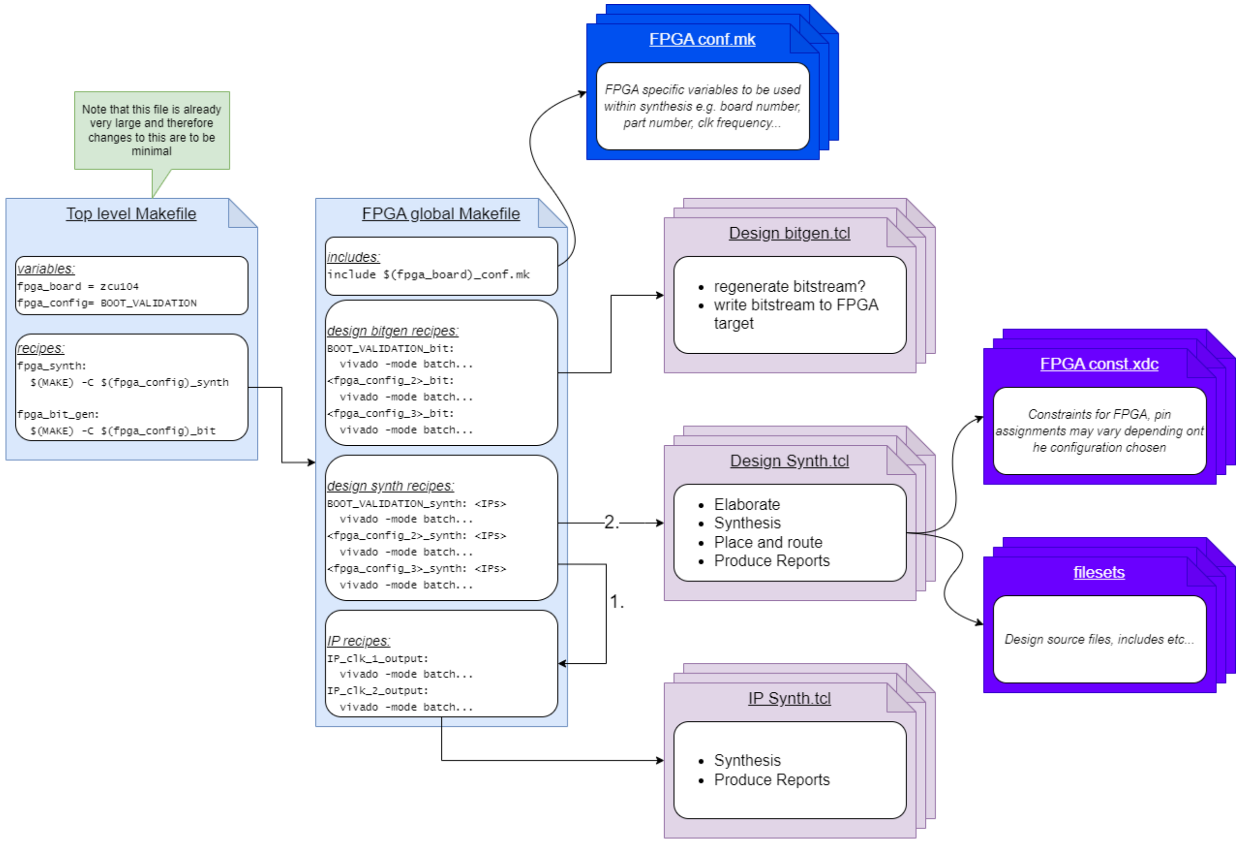
# Prototyping Project Build Flow

## A build system that is modular, extensible and flexible

- Typical tools used i.e. Makefiles + TCL scripts, to ensure compatibility in multiple environments.

## Design refactoring

- ASIC design does not map 1-to-1 on FPGA, but some refactoring is required
  - ➢Clock and reset
  - ➢Clock gating in registers
  - ➢On-chip memories
  - ➢Test logic
  - ➢Pin placement and timing constraints


Note that this file is already very large and therefore changes to this are to be minimal
Top level Makefile
FPGA global Makefile
FPGA conf.mk
FPGA specific variables to be used within synthesis e.g. board number, part number, clk frequency...
Design bitgen.tcl
• regenerate bitstream?
• write bitstream to FPGA target
Design Synth.tcl
• Elaborate
• Synthesis
• Place and route
• Produce Reports
IP Synth.tcl
• Synthesis
• Produce Reports
FPGA const.xdc
Constraints for FPGA, pin assignments may vary depending ont he configuration chosen
filesets
Design source files, includes etc...
2.
1.

*variables:*
```
fpga_board = zcu104
fpga_config= BOOT_VALIDATION
```

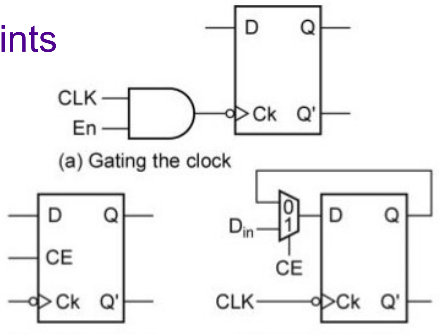
*recipes:*
```
fpga_synth:
  $(MAKE) -C $(fpga_config)_synth

fpga_bit_gen:
  $(MAKE) -C $(fpga_config)_bit
```

*includes:*
```
include $(fpga_board)_conf.mk
```

*design bitgen recipes:*
```
BOOT_VALIDATION_bit:
  vivado -mode batch...
<fpga_config_2>_bit:
  vivado -mode batch...
<fpga_config_3>_bit:
  vivado -mode batch...
```

*design synth recipes:*
```
BOOT_VALIDATION_synth: <IPs>
  vivado -mode batch...
<fpga_config_2>_synth: <IPs>
  vivado -mode batch...
<fpga_config_3>_synth: <IPs>
  vivado -mode batch...
```

*IP recipes:*
```
IP_clk_1_output:
  vivado -mode batch...
IP_clk_2_output:
  vivado -mode batch...
```



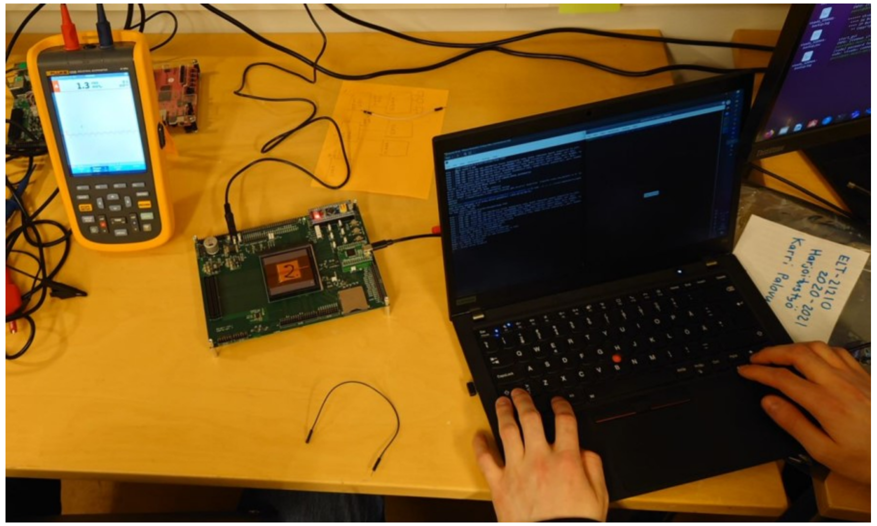
(a) Gating the clock
(b) D-CE symbol
(c) Implementation

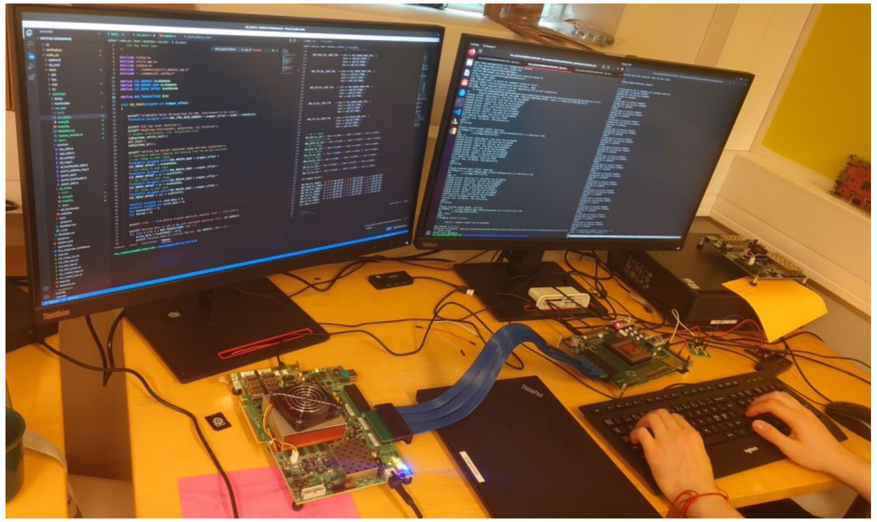
# Ballast Prototyping Results

- **Bug identified in boot design.**
- Debugging design successfully validated.
- Debugging tools and SDK developed.

- When taped-out Ballast sample chips arrived, the wake-up activities were performed using artefacts generated by prototyping activities.

- C2C prototype was extended to be used as off-chip memory bridge for Ballast (Silta).





*Not entirely sure who did the predictions back then, but let's end today with checking how close they were*

*My predictions: Nope.*

# PREDICTIONS FROM 2007(?)

## Possible FPGA Developments

- **The FPGA capacity will continue to grow with processing technology improvements**
  - 90nm devices at 2004, 65nm at 2006, 2008-2009 45nm, 2010 28nm, 2013 14nm
    - Note that this is the first introduction, not volume production
- **Partial reconfiguration / faster configuration?**
- **Widening separation to low-power and high-performance devices**
- **More hard macros**
  - A/D and D/A converters are a good guess for next
- FPGA using non-volatile MRAM configuration
  - Potential to comprise the speed of SRAM and non-volatility of FLASH

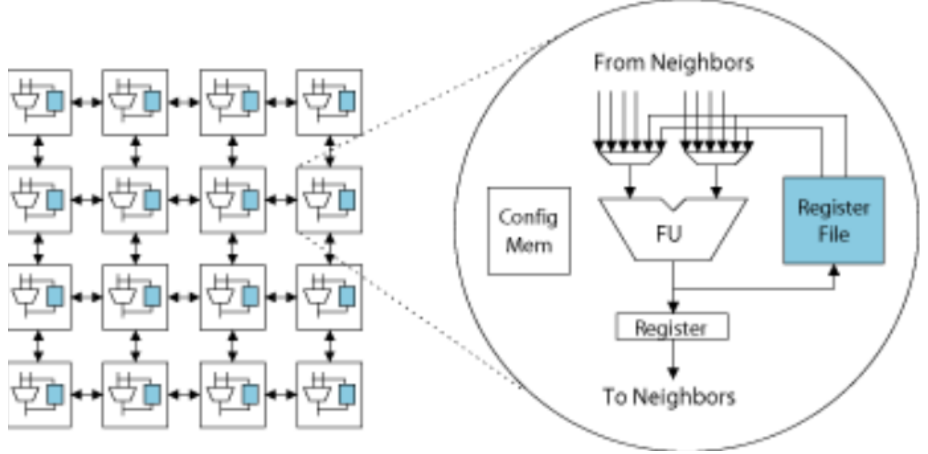
## Possible FPGA Developments (2)

- **We might even see considerable changes in the FPGA device architecture of LEs and interconnects**
  - > 10 years the basic LE contained 4-input LUT and flip-flop
  - This has changed in 2005 to 6-8-input LUT and 2 flip-flops
  - Solution to reduce routing overhead vs. logic delay is increasingly important
- ”Field Programmable Node Arrays”, Coarse grain reconfiguration?

  Wait for it…
  - The FPGAs would consist of *nodes* that are large programmable functional blocks such as algorithmical accelerators or CPUs (the mode can be configured)
  - Adder, MUL, register file, small ALU… instead of LUT+FF
  - These nodes are interconnected with hierarchical networks

# Coarse-Grain Reconfigurable Arrays

- Hardwired ALU is more efficient than ALU built from LUTs
  - Suites DSP algorithms, such as filtering, image manipulation etc.
  - However, less efficient, for example in bit manipulation or exotic bit widths etc.
- Coarse-grain cell need less configuration bits
- Research seems to suffer from *re-inventing the wheel* syndrome
  - ”Hey, let’s start from scratch and do not reuse what others have done!”
  - Rather easy to design (yet another) coarse-grain reconfigurable HW architecture
  - Usually it is much harder to develop design tools than reconfigurable HW
  - Reasonable evaluation against ASIC, FPGA, and CPU seems to be nearly non-existent
- Some commercial products available, but lots of research to be done…



http://cccp.eecs.umich.edu/research/cgra_arch.gif

## WHAT'S A VERSAL ADAPTIVE SOC

### Heterogeneous Acceleration

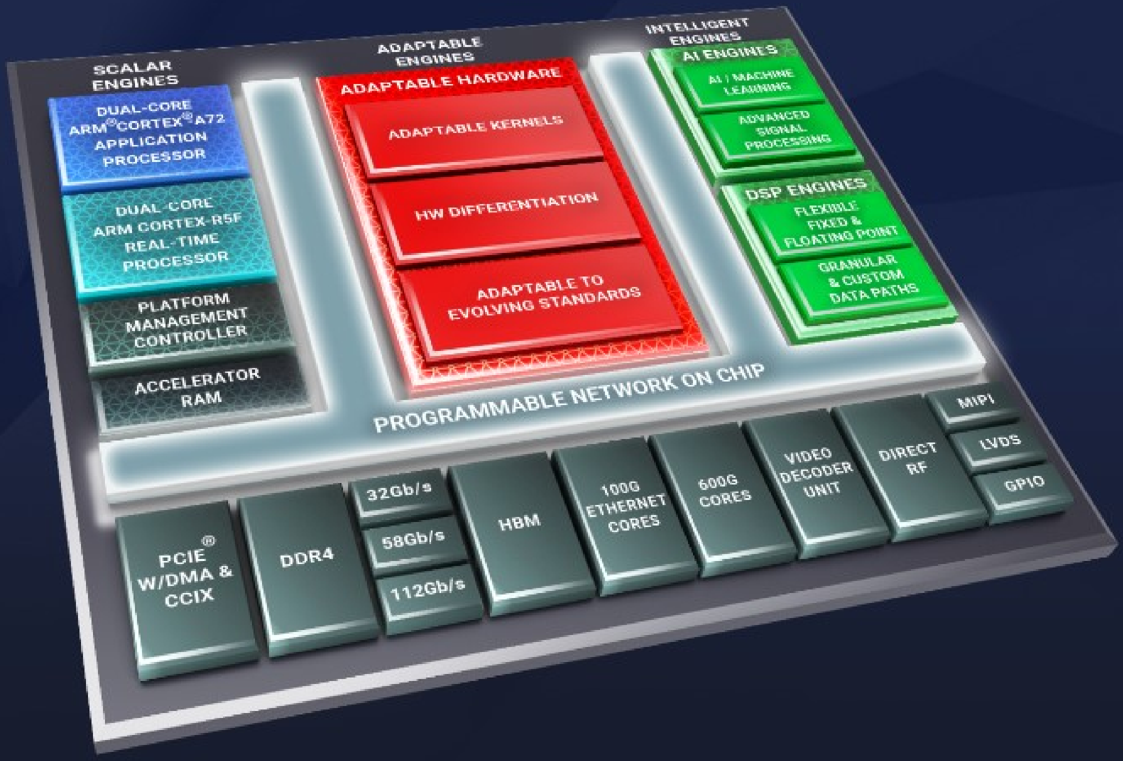
Highly integrated, multicore compute platform that can adapt with evolving and diverse algorithms

### Any Application

Dynamically customizable at hardware and software levels to fit a wide range of applications and workloads

### Any Developer

Architected around a programmable network on chip (NoC), the AMD Versal™ adaptive SoCs are easily programmed by software developers and hardware programmers alike



Versal adaptive SoCs deliver unparalleled application- and system-level value for cloud, network, and edge applications. The disruptive 7nm architecture combines heterogeneous compute engines with a breadth of hardened memory and interfacing technologies for superior performance/watt over competing 10nm FPGAs.

Learn More

The new white paper evaluates system-level performance of the Versal architecture across a set of domain applications as compared to competing programmable-logic based devices. It provides both a qualitative and quantitative analysis of real-world benchmarks.

Read White Paper