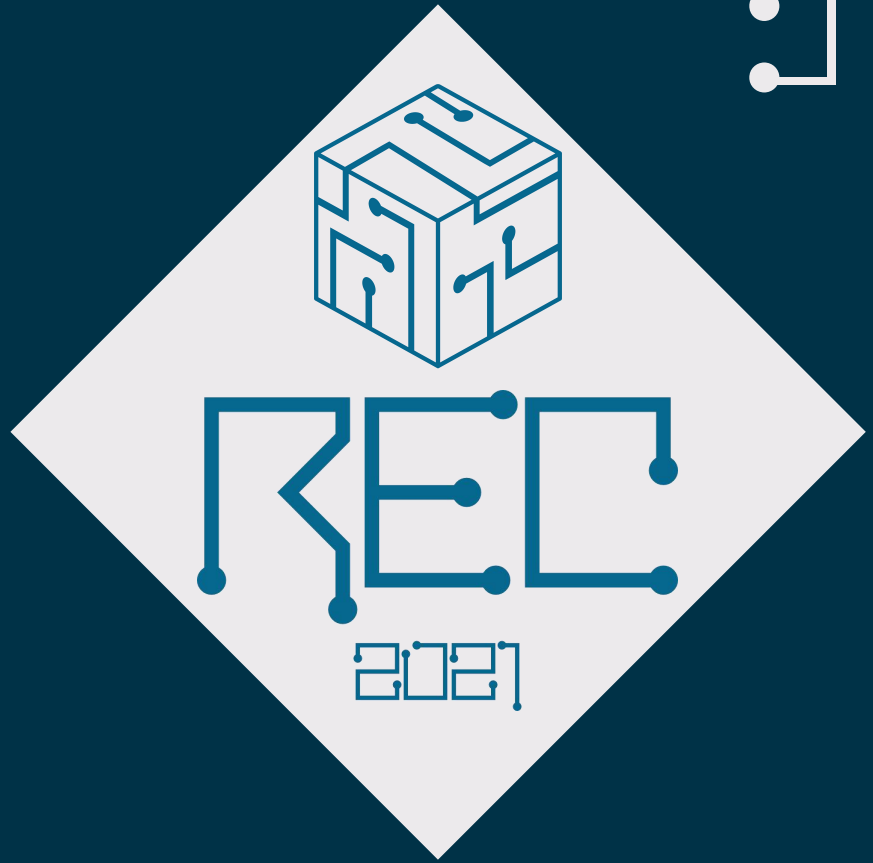
FPGA *101* for Software Engineers



Nuno Paulino
INESC TEC
nuno.m.paulino@inesctec.pt

01

Introduction

What are FPGAs
Early FPGAs
FPGA Architecture
FPGA Growth

02

Learning Curve

Where to Learn?
Hardware “Programming”Languages
Compilation

03

FPGA Spaces

Server
Embedded
Hobby
Edge & AI (?)

04

High-Level Synthesis

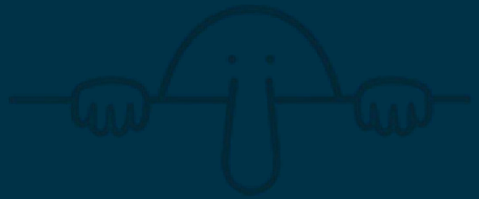
Re-Targeting Old Languages
A Device to Rival GPUs?

05

Witness Testemonies

Pedro Silva
Tiago Santos

01. Introduction



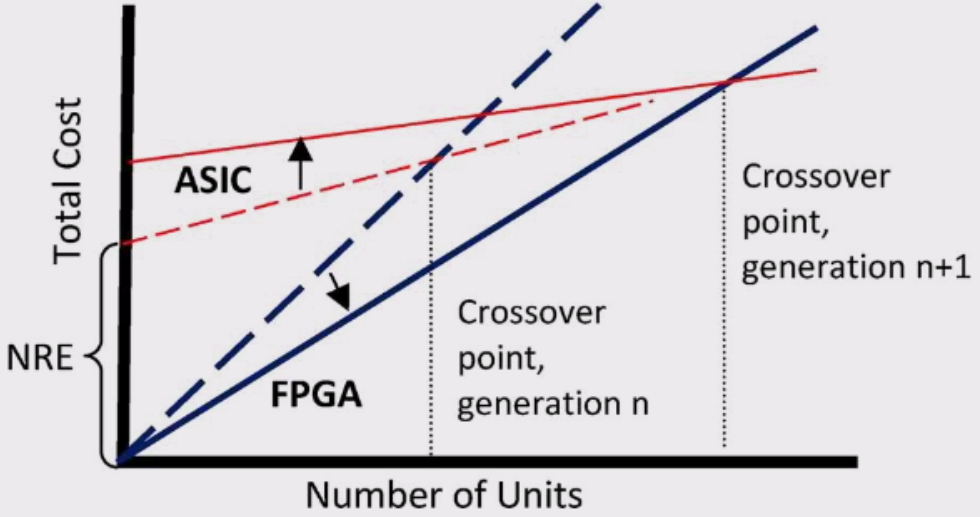
What are they?

- A type of integrated circuit (IC)
  - Reconfigurable functionality by changing connections between logic blocks
  - Like a microscopic breadboard capable of firmware updates
  - You can build anything!

- CPUs/GPUs are programmable too!
  - Yes, but you're stuck with their respective **models** (e.g., von Neumann) -> They are **ASICs**
  - They're also expensive to make (tradeoff at high volume), and "impossible" to bugfix (re-spin)

FPGA vs. ASIC

- Non-Recurring Engineering
  - Initial masking and fabrication cost of ASICs (high)
- They made/make sense versus ASICs depending on **volume**, and on NRE
  - Despite long “compile” times, they’re still orders of magnitude ahead of ASICs on “bug fixes”



Steve Trimberger, “*Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology*”, Proceedings of the IEEE, 2015

The first FPGA and its Father (circa 1984)

- Ross Freeman (1948-1989)
  - Peace Corps Volunteer
  - Inventor of the “FPGA”
  - Founder of Xilinx Inc.
- XC2000 Family
  - Up to 100 4-Input LUTs!
  - Up to 100Mhz! @ 1µm

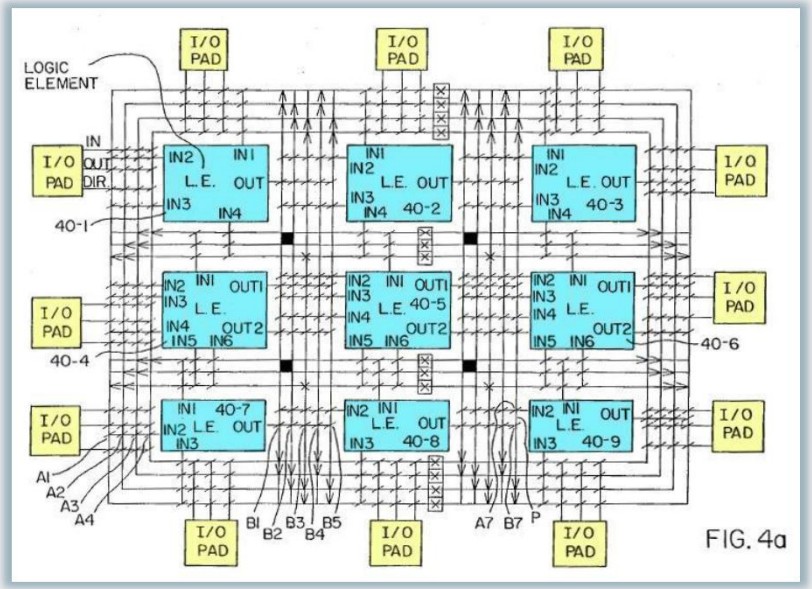
Xilinx XC2000
First family of SRAM re-configurable devices

Ross Freeman
Founder of Xilinx Inc.
(colorized)

More on Ross Freeman: *https://www.autodesk.com/products/eagle/blog/ross-freeman/*

Looking inside...

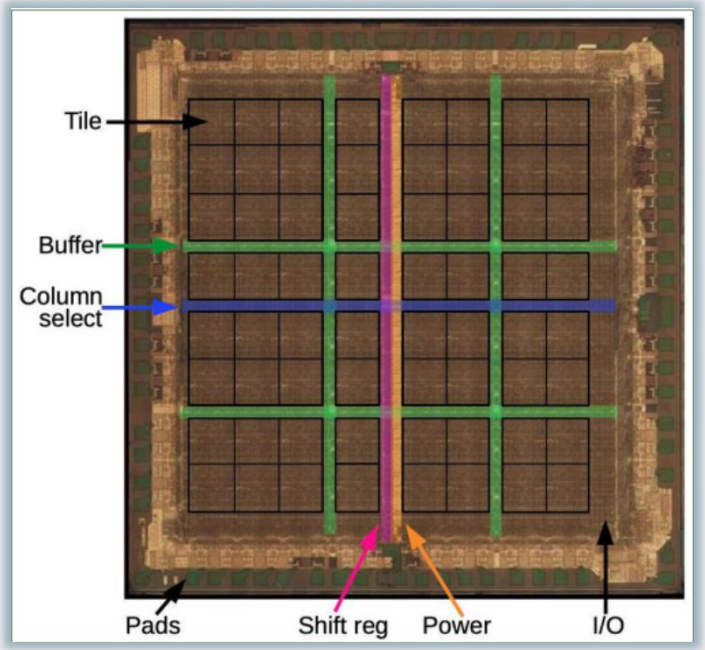
- Configurable Blocks
  - Re-programmable with arbitrary logic functions, + data storage (registers)
- Interconnections
  - Short and long connections between blocks, + connections to the outside
- Programmed with **XACT**
  - MS-DOS *"GUI"*
  - For only **$12.000 in 1984!**



From the XC2000 Patent
Simplified 3x3 diagram
(US4870302A)

XC2000 Under the microscope

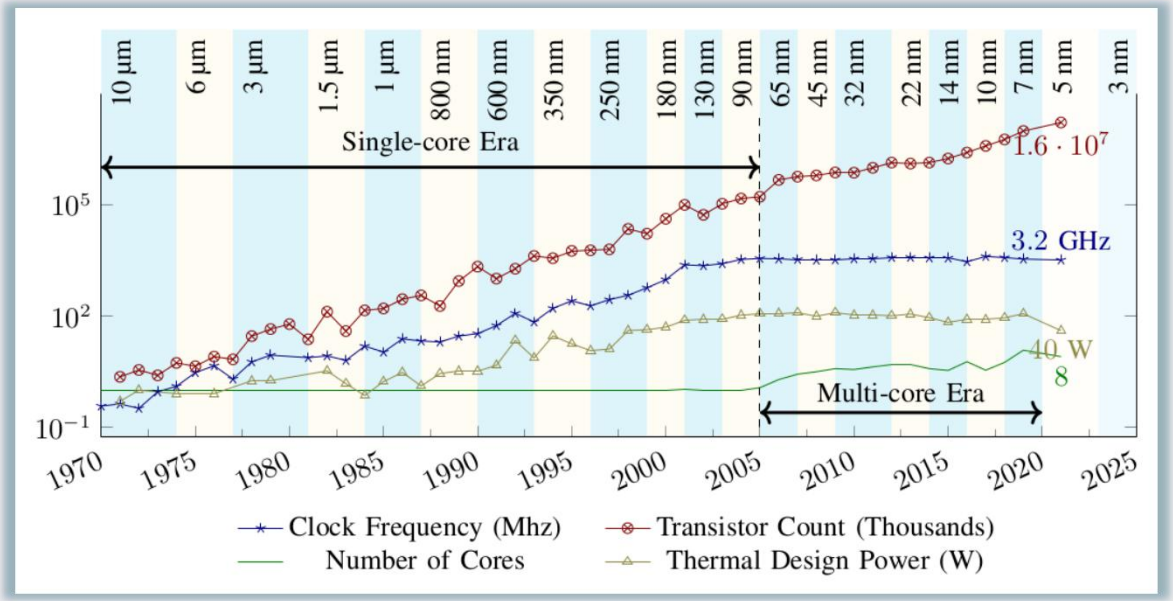
- Tiles in 8x8 arrangement
  - Includes the CLBs and the interconnets
- By todays standards, this is:
  - Small in resources
  - Huge in required size
- Where are we now?



More on the XC2000:
http://www.righto.com/2020/09/reverse-engineering-first-fpga-chip.html

50 Years of CPU Evolution

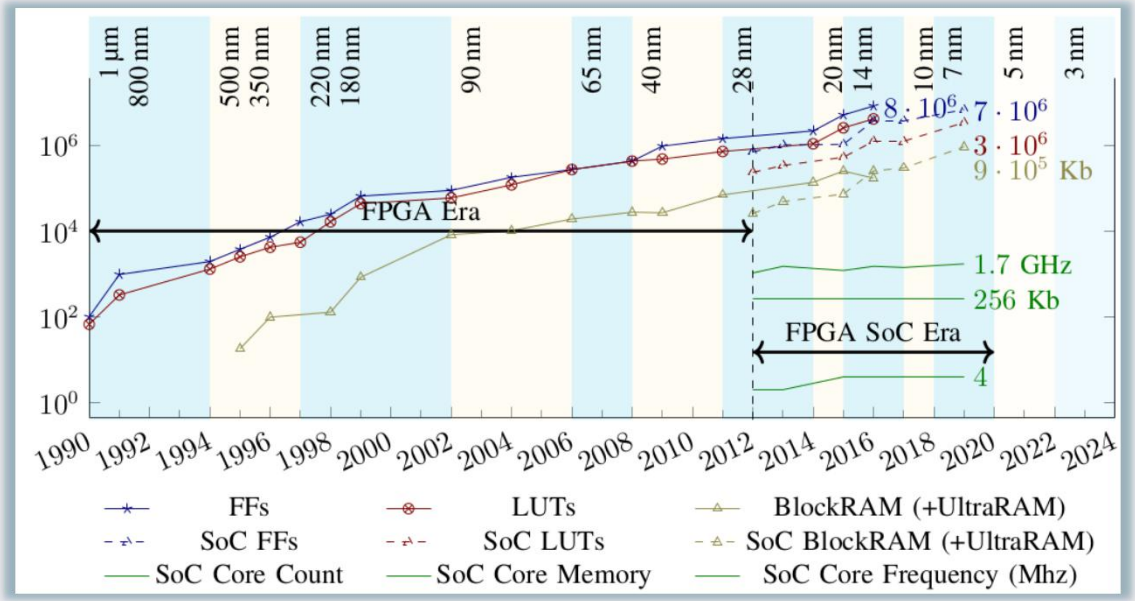
- Average for top 30 devices per year
- Stagnation >2005
  - Start of multi-core era
- **Breakdown** of Dennard Scaling and Moores Law



N. Paulino, J. Bispo, J. C. Ferreira and J. Cardoso,
"A Binary Translation Framework for Automated Hardware Generation," in IEEE Micro

30 Years of FPGA Evolution

- Since ~1990
  - Capacity x10000
  - Performance x100
  - From 1µm to 14nm
  - Many dedicated components (e.g., DSPs)
- After 2012
  - The SoC FPGA Era

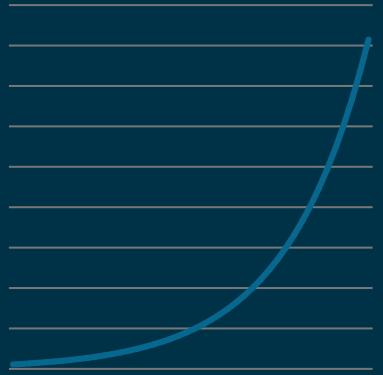


N. Paulino, J. Bispo, J. C. Ferreira and J. Cardoso,
"A Binary Translation Framework for Automated Hardware Generation," in IEEE Micro

Are they really that relevant? Who's involved?

- Intel Corporation
  - Purchases Altera for **$16.6 billion** in 2015
- Advance Micro Devices (AMD)
  - Purchases Xilinx Inc. for **$35 billion** in 2021 (sale just became final)
- Some users: Amazon, Microsoft, Google, Ali Baba
- You may have heard about FPGAs associated with Machine Learning, Deep Learning, AI, Computer Vision, Data centers, etc

02. Learning Curve



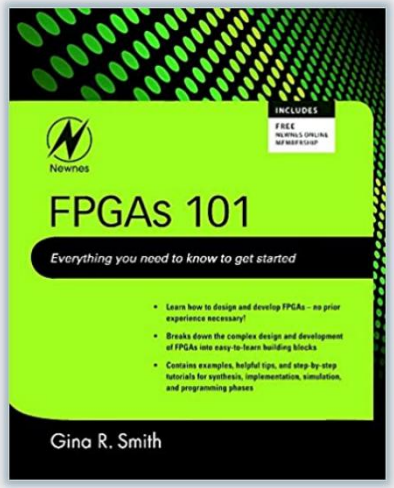
Where to learn?

I already know how to program!

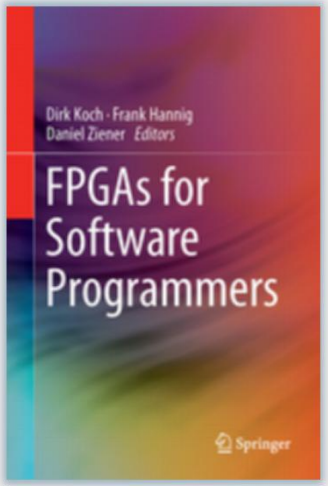
- Googling *"FPGA code Hello World example"* won't get you far...
- What might you need to start?
  - What Languages?
  - How do I compile?
  - What *can* I compile?
  - Where do I run my code?

Where can I learn?

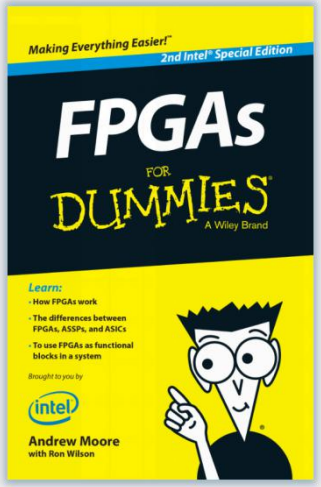
- Books?
  - Big
  - Some expensive
- There are C books too...
  - But honestly, I learned from the Internet
  - Ctrl-C, Ctrl-V, compile, modify and try!



33€
(229 pages)
2010



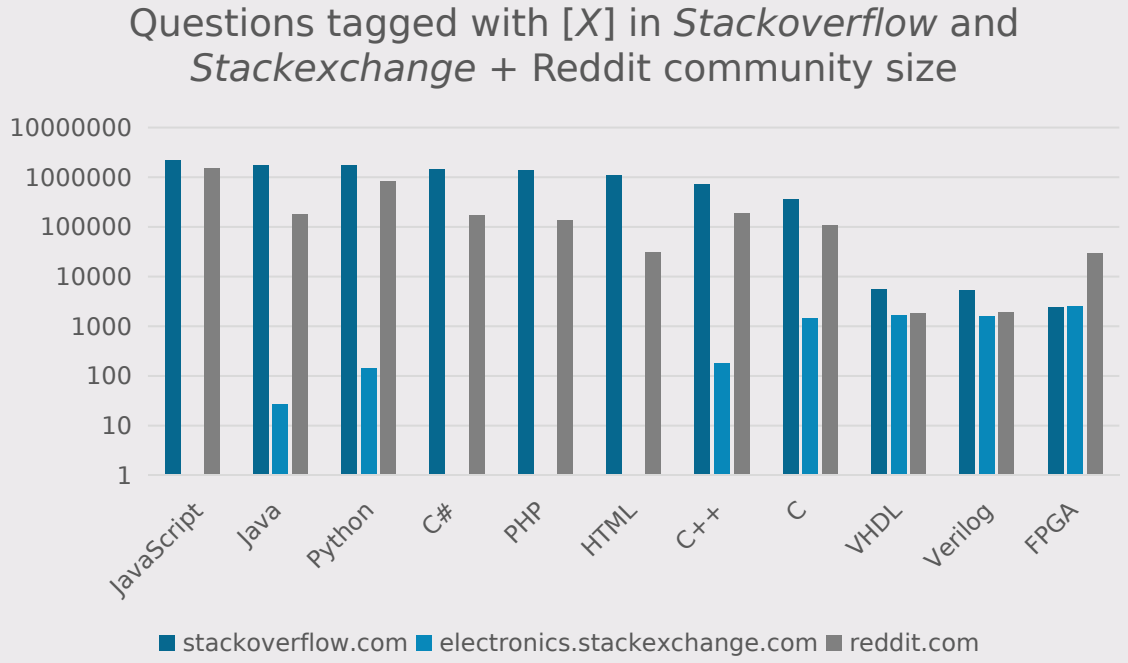
85€
(327 pages)
2016



Free!
(53 pages)
2017

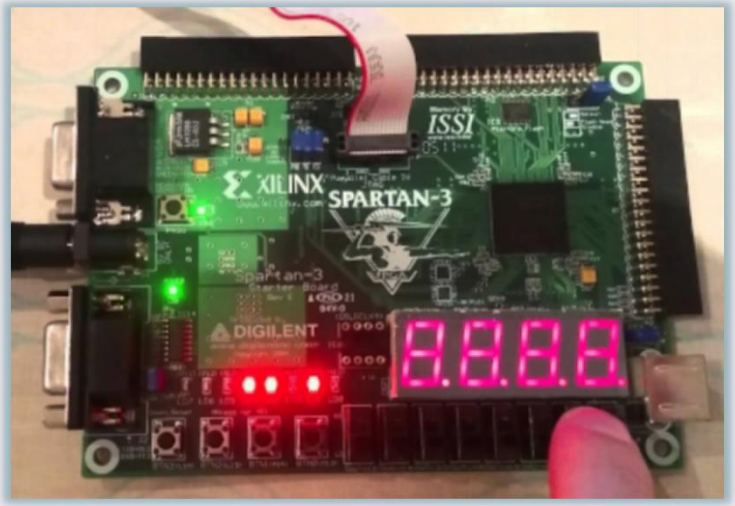
Maybe online?

- ~2.2 million hits for languages like Javascript, Java, etc
- ~1000 hits for FPGAs and related languages...
- There isn't much of a community... **yet!**



Electric Engineers must learn plenty about FPGAs (?)

- Lets look...
  - MIEEC@FEUP
    - ~90 subjects... 2 or 3 on FPGAs?
  - MIEEC@Nova
    - ~130 subjects... 1 on HDLs? (not sure)
- How much time to be a good digital circuit design engineer on FPGA?
  - Opinions range from 2 to 5 years, full time.
  - But let's try...



We learn on this (Xilinx Spartan-3 Development Kit)

Once the LEDs blink, it's a great sucess!

Hello World?

- In C

```c
#include <stdio.h>
int main() {
   printf("Hello, World!");
   return 0;
}
```

```
~$ gcc hello.c -o hello
~$ ./hello
“Hello World!”
```

  - I can edit compile and run in **seconds (!)**, and debug with *printfs (!!!)*

- On FPGAs, lets Google it...

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity hello_world is
   port(
   clk : in std_logic;
   led : out std_logic);
end hello_world;

architecture rtl of hello_world is

constant CFREQ : integer := 20000000;
constant BFREQ : integer := 2000000;
constant CMAX : integer :=
CFREQ/BFREQ/2-1;

signal cnt : unsigned(24 downto 0)
  := to_unsigned(0, 25);
```

```
signal blink
  : std_logic := ‘0’;

begin
process(clk)
   begin
   if rising_edge(clk) then
    if cnt = CMAX then
      cnt <= (others => ’0’);
      blink <= not blink;
    else
      cnt <= cnt + 1;
    end if;
   end if;
end process;
led <= blink;
end rtl;
```

...where’s the output?

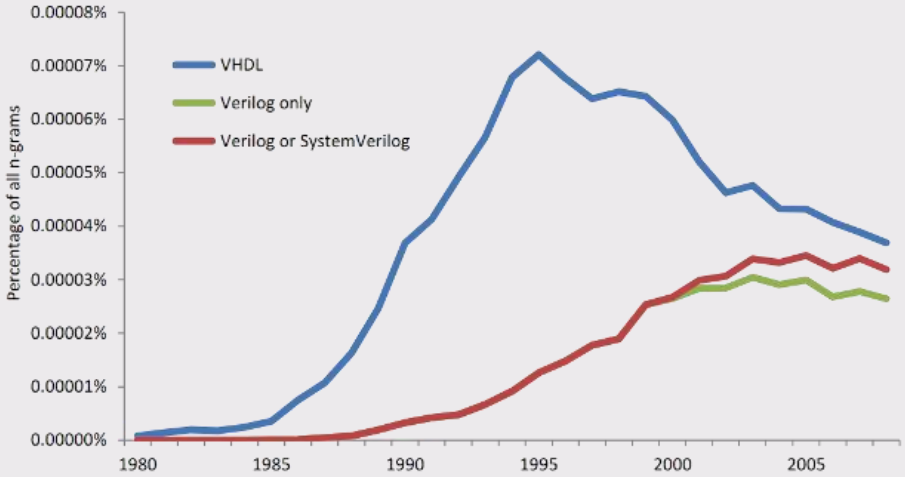
Languages

How to design hardware? Hardware Description Languages

- Fundamentally
  - Statements are concurrent
  - Scopes express modules (blocks)
  - There's no functions, stack, heap, memory, *stdio*, *etc*
- HDLs simultaneously express structure (space) and control (time)

The usual suspects...

- Verilog (since 1984)
  - Weak typing
  - Less verbose (than VHDL)
- VHDL (since ~1980)
  - Strong typing
  - More verbose (than anything else)
- Mixed Design (both!)



Steve Golson, Leah Clark, "Language Wars in the 21st Century: Verilog versus VHDL–Revisited", 2016, Synopsys Users Group (SNUG)

"Europe used to be a huge VHDL supporter, but this is a legacy issue now and there is very little new VHDL being written."
- Steve Holson and Leah Clark

Emerging Object Oriented Languages (and IRs...)

- **Chisel3** (since 2012) and **SpinalHDL** (since 2014), others (DFiant, Gemini, ...)
  - **Both** based on Scala (i.e., inner DSL)
  - Generate HDL from OO design (inheritance, overloading)
    - A lot of boilerplate is removed (e.g., clock declarations, process blocks, enables, resets)
  - Online Jupyter bootcamps available!

Calyx

https://capra.cs.cornell.edu/calyx/
DOI:10.1145/3445814.3446712

CHISEL

https://github.com/chipsalliance/chisel3
https://fires.im/micro19-slides-pdf/02_chipyard_basics.pdf

https://github.com/SpinalHDL
https://spinalhdl.github.io/SpinalDoc-RTD/

Emerging Object Oriented Languages (and IRs...)

- Chisel3

```
class Add extends Module {
  val io = IO(new Bundle {
    val a = Input(UInt(8.W))
    val b = Input(UInt(8.W))
    val y = Output(UInt(8.W))
  })

  io.y := io.a + io.b
}
```

- SpinalHDL

```
class MyComponent extends Component {
  val io = new Bundle {
    val a = in Bool
    val b = in Bool
    val c = in Bool
    val result = out Bool
  }

  io.result := (io.a & io.b) | (!io.c)
}
```

Both very similar, and allow for functional programming for hardware! e.g., (Chisel3):

```
val delayFilter = Module(new FirFilter(8, Seq(0.U, 1.U))) // functional module decl.
```

Emerging Object Oriented Languages (and IRs...)

- More on Chisel3
  - Developed at UC Berkley
  - Uses **FIRRTL** intermediate representation (LLVM of hardware?)
  - Integral part of Berkeley's Chipyard
    - BOOM (Berkely Out-of-Order Machine), Rocket Chip (In-Order Core), etc
  - Used in **Sifive**!
    - *"At SiFive, all RTL development is done in Chisel (...)"*
      - Krste Asanović, RISC-V Foundation
  - Some already teach it (e.g., University of Denmark)

https://chipyard.readthedocs.io/en/latest/index.html

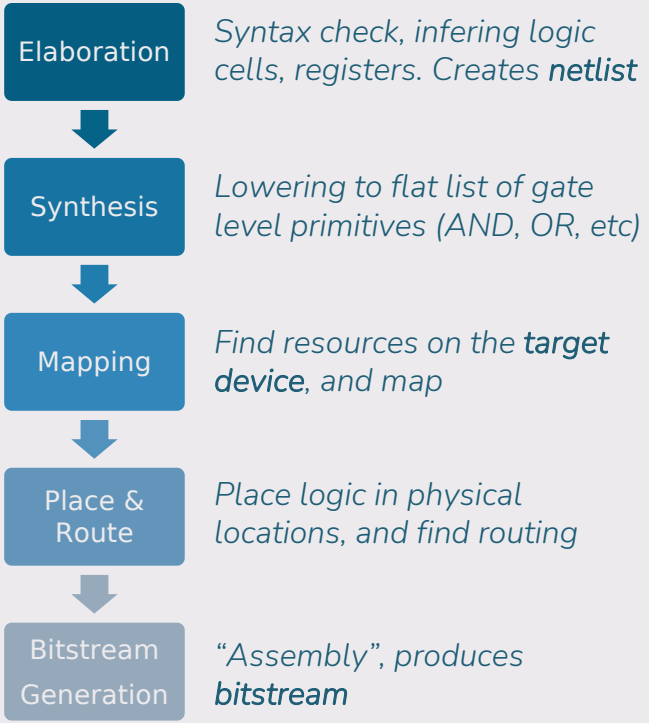
Tools

Compilation Flow and IDEs

- Compilation -> Synthesis
- Some giants
  - Xilinx Vitis / Vivado Suites
  - Intel Quartus
  - Synopsys Design Compiler
- Some free tools/projects exist, like Verilator, SymbiFlow, Yosys, Rapidsmith

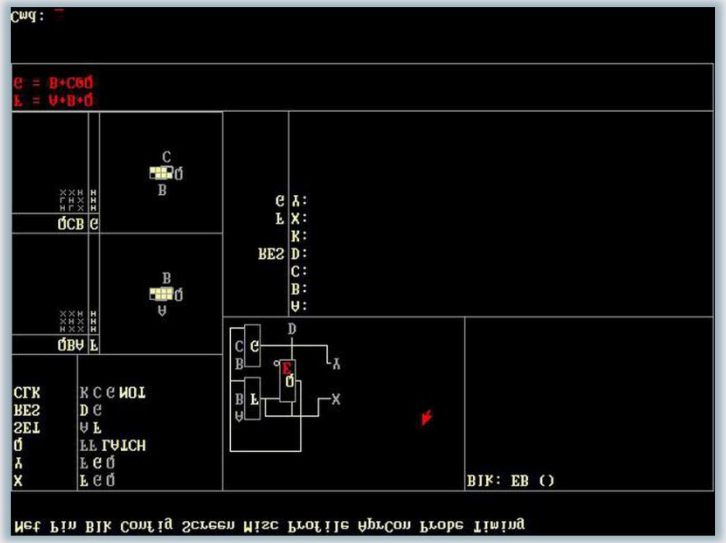
From HDLs to Circuits

- When you compile C code, you generate control for your architecture
- But here, the code **is the architecture**
- **Place & Route** is one of the **major** outstanding issues in FPGA design
  - Design dependant, but can be up to **dozens of hours**



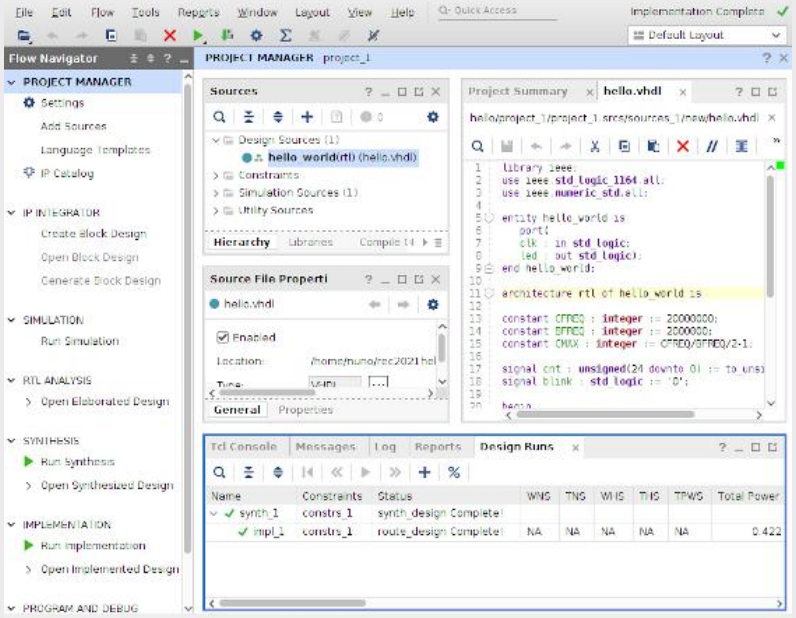
XACT

- Locked away somewhere in FEUP, this software remains...



Xilinx Vitis + Vivado

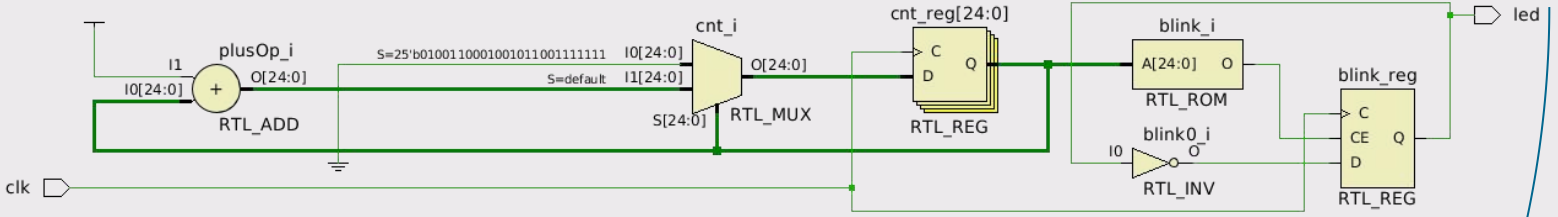
- Sucessor to many other tools..
  - Xilinx ISE (defunct)
  - Xilinx EDK (defunct)
  - Xilinx SDx (defunct?)
- Vitis
  - Software perspective (host code + HLS)
- Vivado
  - Hardware perspective (HDL, block designs, IP blocks)



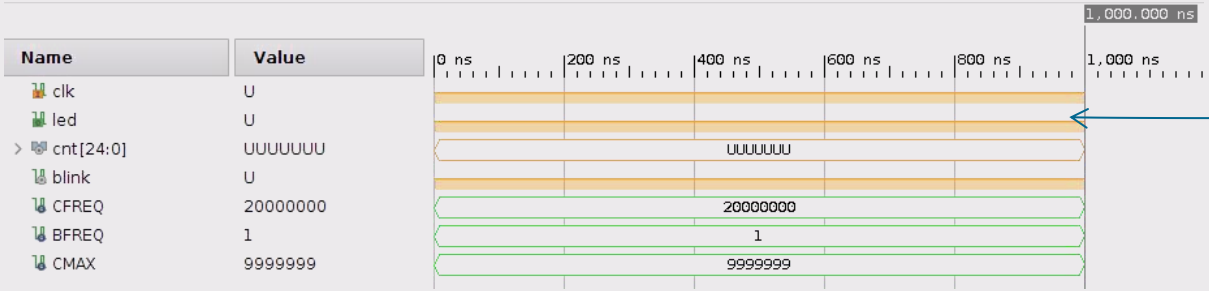
Our "Hello" in Vivado

Ok. Back to our Hello World. Where's the output?

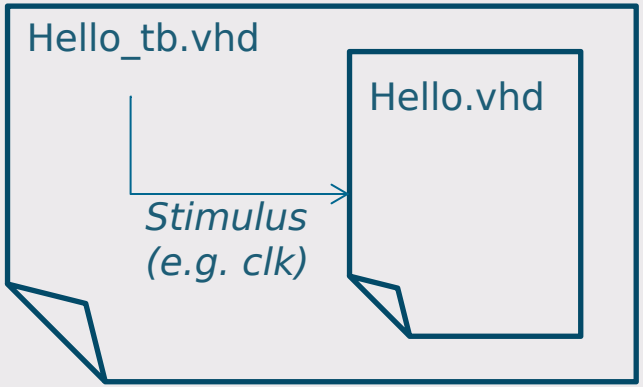
- Elaborated Design



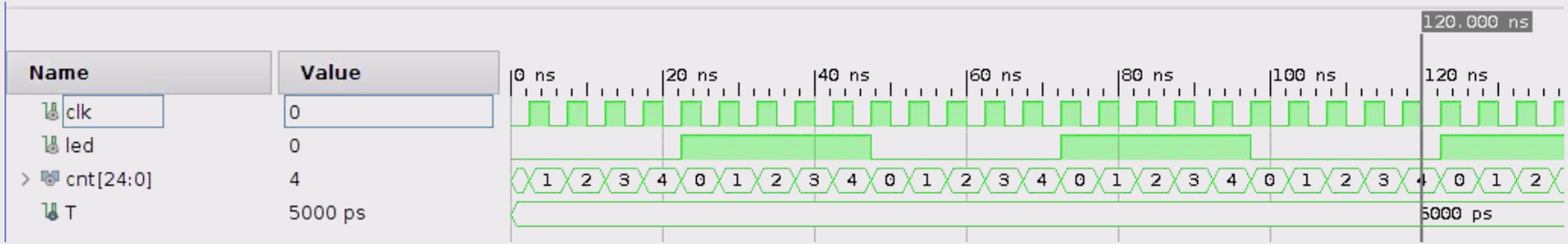
- I guess I'll click Run Simulation?



I can’t compile and run?... No, you need a testbench!



- Verification is most of the job
  - For very large designs, its not trivial
  - Worse if you have components external to the FPGA (i.e., DDR)
  - But its 100% deterministic!



On the chip! (Synthezising for a Xilinx Virtex-7)

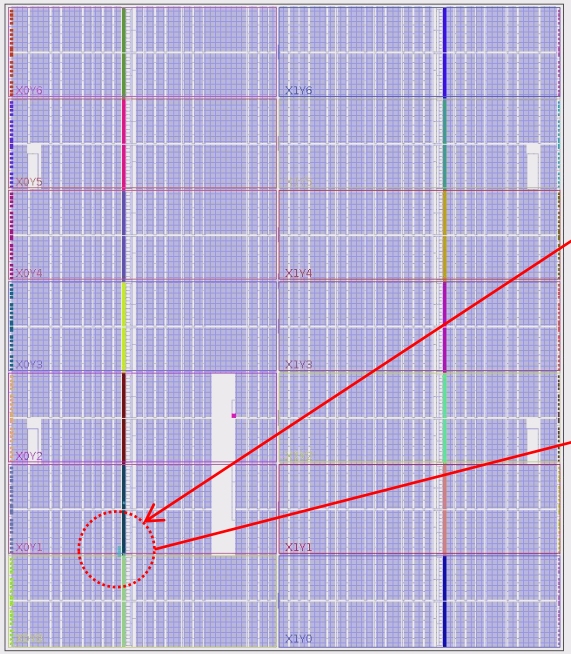


Diagram of the Virtex-7 FPGA in Vivado

Where is it?...
There!

Zoom!

- That's the (ugly) core of it!
- Do I have to create everything from scratch? No.
  - Many common components are embedded into the FPGA
  - Lots of pre-made "soft-core" designs (e.g., RISC-V soft-cores for your FPGA)
  - Progress has been significant towards higher abstractions (away from HDL)
- Libraries, abstractions, and form-factors place the FPGA in many **spaces**

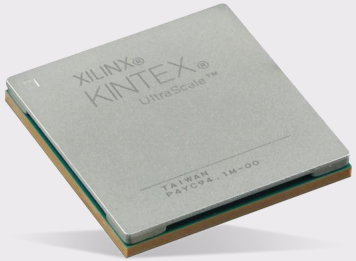
03. FPGA Spaces

The FPGA and where you put it

- The FPGA is only the IC
- Where to put CPUs and GPUs
  - CPU --> Motherboard socket
  - GPU --> Motherboard PCI-e Slot
- FPGAs I can place on
  - PCI-e boards -> Server Racks -> **Server Space**
  - Custom PCBs
    - **Edge/Embedded Space**
    - **Hobby/Educational Space**
  - Development Kits --> May cover all of the above

Stratix 10
SX2800K SoC
(High-end, 14nm)

I cost about
$24.800 (!)



Kintex UltraScale
XCKU115 SoC
(Mid-range, 20nm)

I cost about $7,800

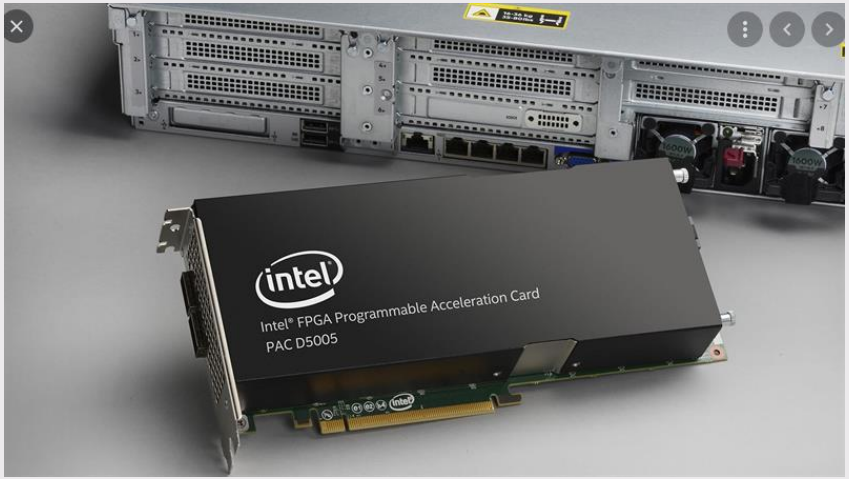
Server Space

Powering big data workloads

- Applications
  - Cloud Computing (search engines)
  - Cutting-edge AI Applications
  - SmartNICs
  - Accessibility (for education)
- Hardware updates can lead to server performance improvements at low cost

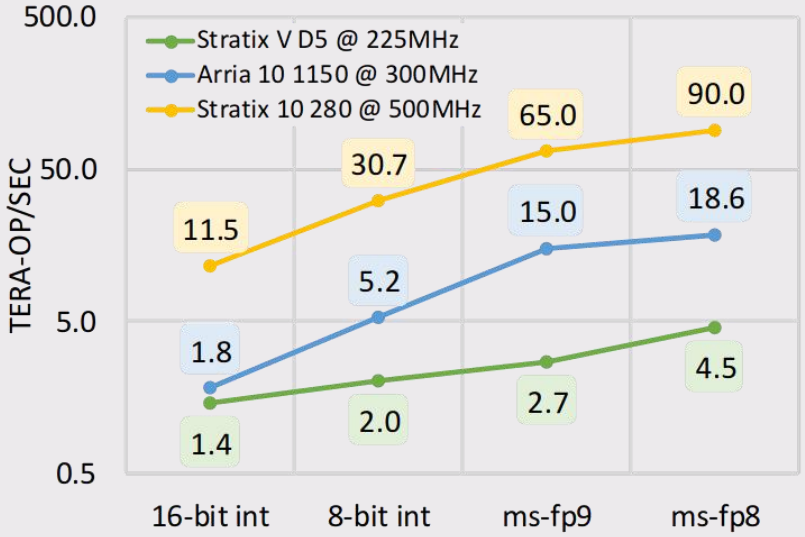
- *“30,000 Images/Second”*
  - Two AMD EPYC 7551 CPUs
  - **Eight** Alveo U250 PCI-e Cards
    - ~$7.500 (each!)
  - AI Inference Record (?)
    - GoogLeNet CNN

AI Inference Record:
https://www.enterpriseai.news/2018/10/03/30000-images-second-xilinx-and-amd-claim-ai-inferencing-record/



- Intel Programmable Acceleration Cards (PAC)
  - ~$7.000
  - Based on Stratix 10 FPGAs (Altera)
  - Example:
    - Intel OpenVINO Toolkit
    - ~20x over GPU based solutions

OpenVINO™ Toolkit and FPGAs
https://techdecoded.intel.io/resources/openvino-toolkit-and-fpgas/



"NPU Peak performance of the Brainwave DPU across three generations of Intel FPGAs. The use of ms-fp8 narrow precision improves performance by 3.2X-7.8X over a conventional 16-bit fixed point."

- Microsoft Project Brainwave
  - Models used in Bing and Azure
    - *"(...) the world's largest cloud investment in FPGAs"*
  - On a Intel Stratix 10 (280k) FPGA
    - 39 TFLOPS @ 300MHz, 125W
    - (RTX 2080: 89 TFLOPS, 250W)

Project Brainwave:
https://www.microsoft.com/en-us/research/project/project-brainwave/
https://ieeexplore.ieee.org/document/8344479

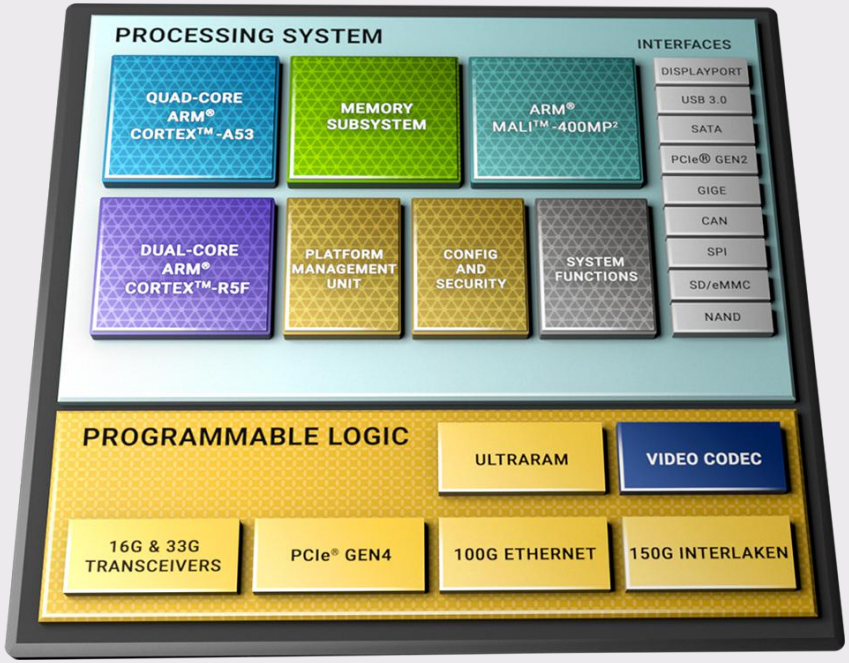
- Versal AI VC6190 Kit
  - Only **$12.000!**
  - The first “ACAP” type device
    - Dedicated AI engines + CPU + FPGA
  - *“(...) x100 greater compute efficiency over server-grade CPUs (...)“*
  - *“(...) x20 over other FPGAs (...)“*

Versal ACAP White Paper: *https://www.xilinx.com/support/documentation/white_papers/wp505-versal-acap.pdf*

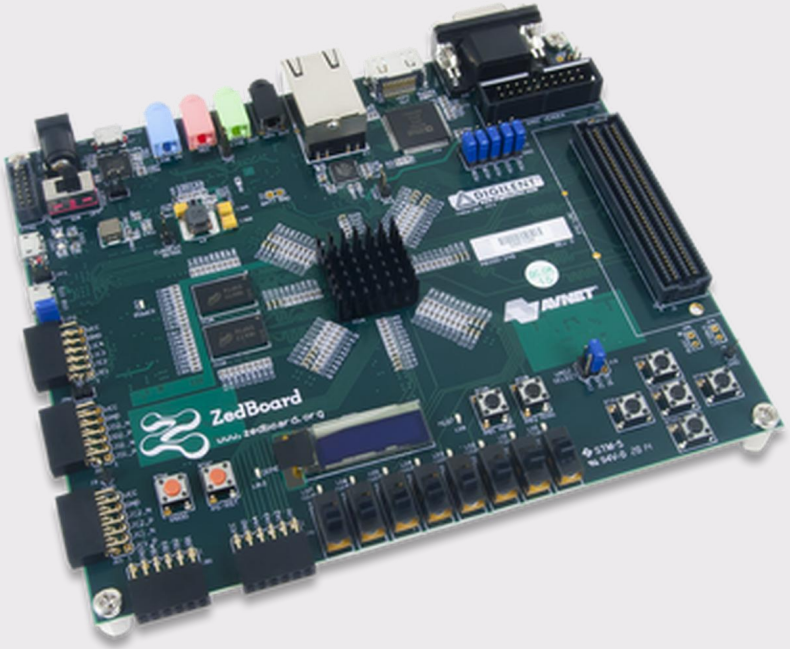
Embedded Space

Gaining more traction with the FPGA based MPSoC

- Sub-Spaces
  - Consumer Electronics
  - Telecommunications
  - Automotive
  - Medical
  - Space & Defence



- Xilinx UltraScale+ Family
  - Hard ARM cores
  - Common function cores built-in
  - Fast interface between PS and PL
  - Operating Systems (!)
- Single-board projects away from barebones gate level design



- Xilinx Zedboard
  - $450 (!)
  - Zynq-7000 SoC FPGA
  - Lots of peripheral interfaces
  - Single board computer
  - (Sucessor to the Spartan-3 for EE classes?...)

Some Products!

- Waymo (Google's self driving car)
- HTC Vive (VR)
- NVIDIA G-Sync
- Smartphones
  - Good for fixing "bugs" after product launch
- Apple MacBook Pro "Afterburner"
  - PCI-e card for video codecs / streaming / editing

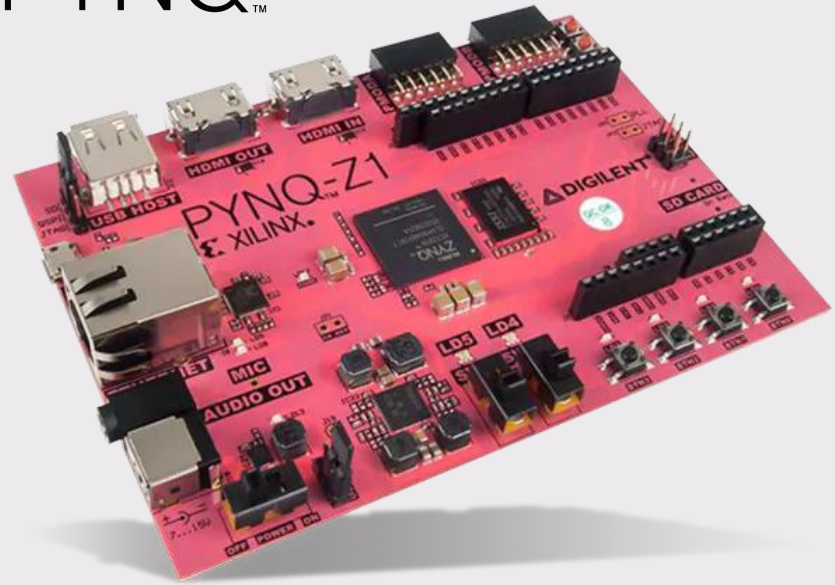
Examples from presentation from VHDLwhiz:
"An Introduction to FPGAs & Programmable Logic"
https://www.youtube.com/watch?v=lmvdPQQAehQ

Hobby Space!

Dude, where's my weekend project?

- Wheres the Arduino or Raspberry Pi of FPGAs?
  - Used to be difficult due to complexity of tools (>20GBs), licenses required, cost of boards, learning curve, HDLs, *etc*
    - *Hard to create a community "backbone"*
  - Now there are a few alternatives!

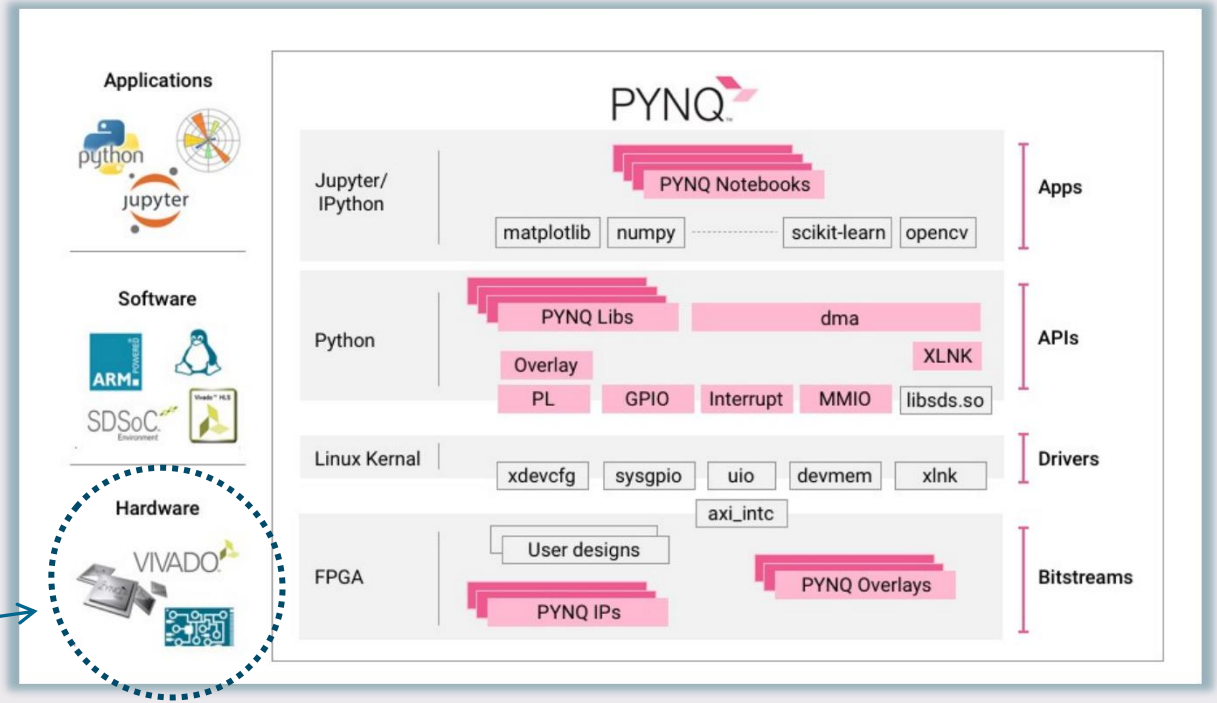
PYNQ



- Xilinx PYNQ-Z1
  - 8 cm x 12 cm
  - $199 (+ accessories…)
  - Xilinx Zynq®-7020 SoC
  - Python + Zynq = PYNQ
  - With Operating System
  - https://github.com/Xilinx/PYNQ_Workshop

Xilinx PYNQ Abstraction Stack

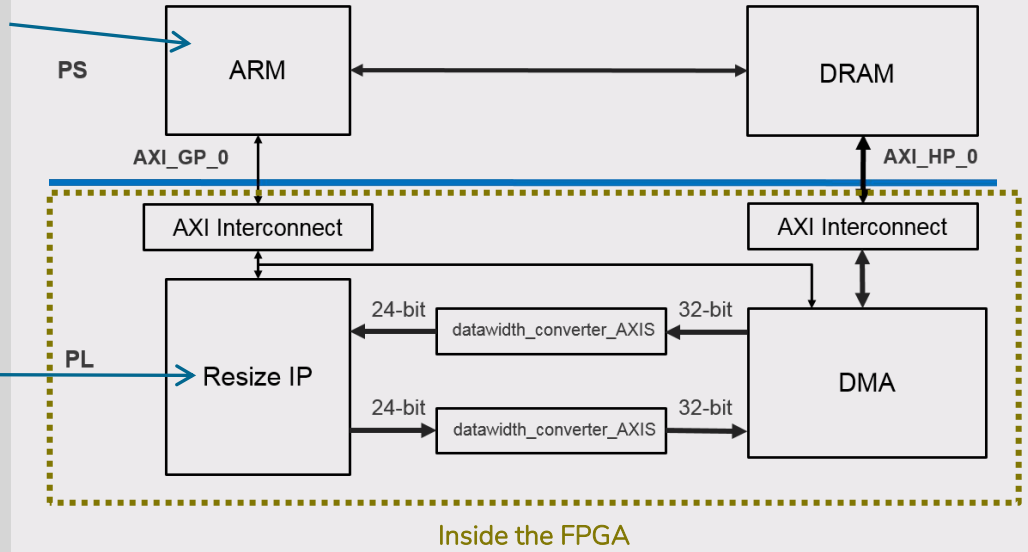
- Run an OS
- Use familar languages to integrate *sw + hw*
- Need custom modules?
  - Still need to design the hardware



Example on the PYNQ-Z1 (PYNQ-HelloWorld)

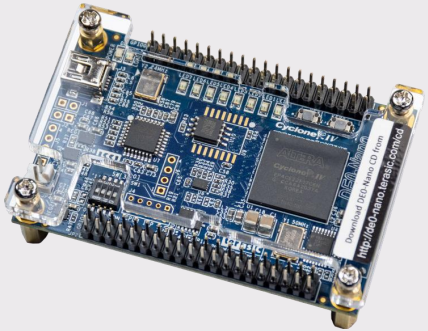
- *https://github.com/Xilinx/PYNQ-HelloWorld*

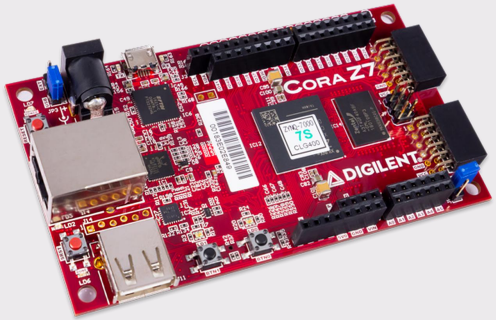
```
%% (...) imports

resize_design = Overlay("resizer.bit")
dma = resize_design.axi_dma_0
resizer = resize_design.resize_accel_0
original_image = Image.open("images/sahara.jpg")

%% (...) boilerplate (...)

resizer.write(0x10, old_height)
resizer.write(0x18, old_width)
resizer.write(0x20, new_height)
resizer.write(0x28, new_width)

dma.sendchannel.transfer(in_buffer)
dma.recvchannel.transfer(out_buffer)
resizer.write(0x00,0x81) # start
dma.sendchannel.wait()
dma.recvchannel.wait()

resized_image = Image.fromarray(out_buffer)
_ = plt.imshow(resized_image)

del in_buffer
del out_buffer
```

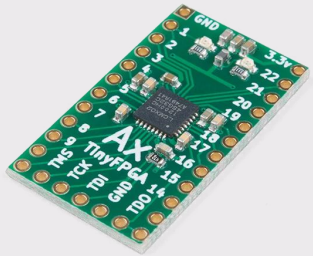


(Some) Other Boards

- Terasic DE0-Nano
  - 20cm x 13cm
  - Altera Cyclone IV
  - $93
- Digilent Cora Z7
  - 15cm x 7cm
  - Xilinx Zynq-7010
  - $99
- TinyFPGA
  - 3.0cm x 1.7cm
  - Lattice FPGA
  - As low as **$12**!

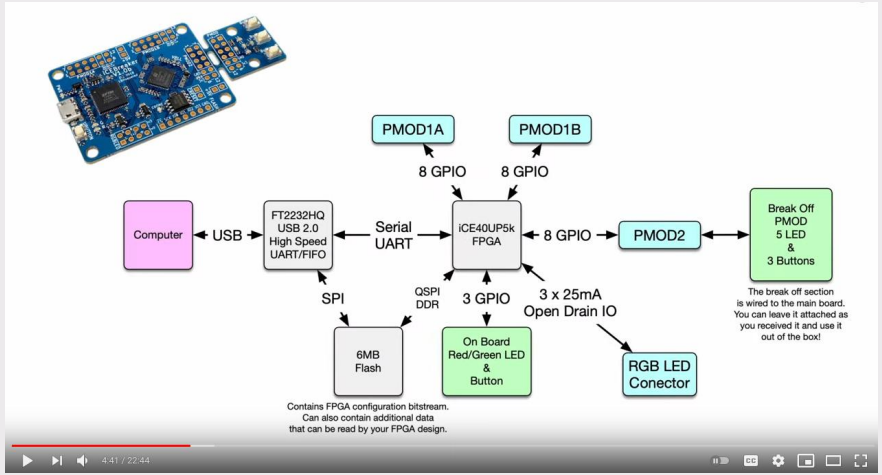






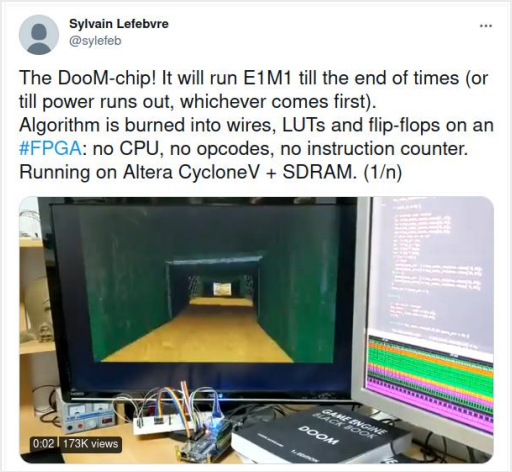
But does it run DOOM?

- Yes.
  - Using a RISC-V main processor
  - On a Lattice ICE40 FPGA



https://www.youtube.com/watch?v=3ZBAZ5QoCAk
https://hackaday.com/2021/02/07/ice40-runs-doom/

- Twice.
  - Using entirely custom logic (no insts.)
  - On a Intel Cyclone V FPGA



https://twitter.com/sylefeb/status/1258808333265514497
https://www.engadget.com/doom-chip-fpga-173503758.html

Edge

The best device for power efficiency at the edge?

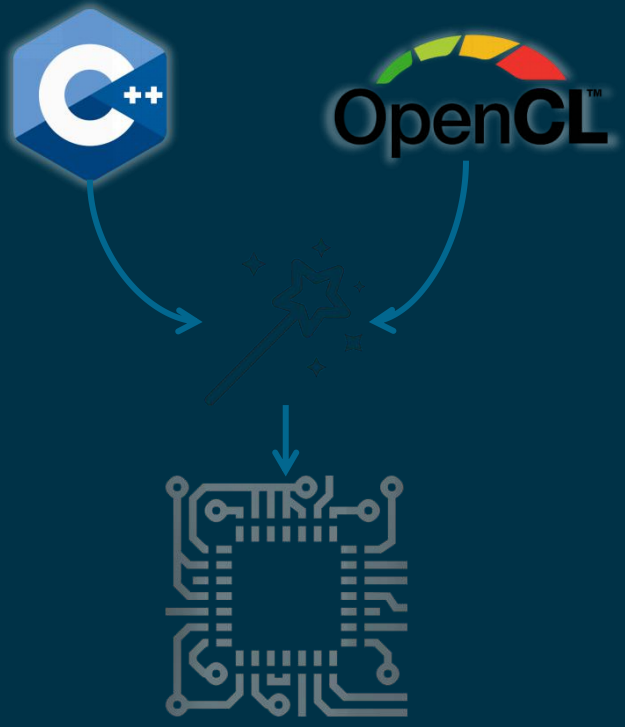
- The domain of excellence?
  - Adaptive
  - Low NRE
  - Updates Over-the-Air
  - Hardware accelerated radio for Internet-of-Things
  - Good performance to energy tradeoff (for battery devices)

Examples of recent platforms for Edge Applications

- FPGA Based
  - Xilinx Kria Family (and others)
  - ~$250 for this module

- A challenger appears! GPUs (?)
  - NVIDIA Jetson Family
  - ~$479 for the standalone module

04. High-Level Synthesis



C/C++ or OpenCL into RTL

- Xilinx Vitis HLS (Vivado HLS)
  - *“Vitis™ HLS is a high-level synthesis tool that allows C, C++, and OpenCL functions to become hard wired onto the device logic fabric and RAM/DSP blocks.“*
- Intel Quartus HLS
  - *“(...) is a high-level synthesis tool that takes in untimed C++ as input and generates production-quality register transfer level (RTL) code (...)“*

- Example

```
extern "C" {
void krnl_vadd( const unsigned int* in1, const unsigned int*
in2, unsigned int* out_r, int size) {

  unsigned int v1_buffer[BUFFER_SIZE];

    for (int i = 0; i < size; i += BUFFER_SIZE) {
#pragma HLS LOOP_TRIPCOUNT min = c_len max = c_len
        int chunk_size = BUFFER_SIZE;
        if ((i + BUFFER_SIZE) > size) chunk_size = size - i;

        for (int j = 0; j < chunk_size; j++) {
#pragma HLS LOOP_TRIPCOUNT min = c_size max = c_size
            v1_buffer[j] = in1[i + j];
        }
        for (int j = 0; j < chunk_size; j++) {
#pragma HLS LOOP_TRIPCOUNT min = c_size max = c_size
            out_r[i + j] = v1_buffer[j] + in2[i + j];
        }
    }
}}
```

https://github.com/Xilinx/Vitis_Accel_Examples

Example: OpenCL (1/3)

- Accelerating k-means via HLS
  - Alpha Data PCI-e FPGA Card (Kintex)
  - Task kernels (single-thread)
    - Loop pipelining
  - NDRange kernels
    - OpenCL model of workgroups

```
__kernel void s1kmeans1(global uint *data, int n, int m,
int k, int t, global uint *centr, global int *labels,
global uint *cl, global int *counts, global int *itcount)
{
  ulong old_error, error = INT_MAX;
  uint i = 0, j = 0; itcount[0] = 0;

  do {                                                    A
    old_error = error, error = 0; // save error
    for (i = 0; i < k; i++) {
      counts[i] = 0;  // clear tmp counts
      for (j = 0; j < m; j++) cl[i*m+j] = 0;
    }

    for (int h = 0; h < n; h++) {                         B
      uint mindist = INT_MAX;                             C
      for (i = 0; i < k; i++) {
        ulong dist = 0, diff = 0;                         D
        for (j = 0; j < m; j++) {
          diff = data[h*m+j] - centr[i*m+j];
          dist += diff*diff;
        }
        if((int)(dist/2) < (int)(mindist/2)) {
          labels[h] = i;
          mindist = dist;
        }
      }
      counts[labels[h]]++;
      for (j = 0; j < m; j++) // new aux sum
        cl[labels[h]*m+j] += data[h*m+j];
      error += mindist; // update error
    }
    itcount[0]++;
    for (i = 0; i < k; i++) // new centroids
      for (j = 0; j < m) && (counts[i] > 0); j++)
        centr[i*m+j] = cl[i*m+j] / counts[i];
  } while(abs((error - old_error)) > t);
}
```

Baseline OpenCL version of k-means clustering

Nuno Paulino, J. C. Ferreira and J. M. P. Cardoso, "Optimizing OpenCL Code for Performance on FPGA: k-Means Case Study With Integer Data Sets," in IEEE Access, vol. 8, 2020

http://pepcc.inesctec.pt

Example: OpenCL (2/3)

- Changes?
  - No outtermost loop
  - Some work moved to the host (scope A)
  - OpenCL vector types
    - **uint16**
  - Scopes E1 to E4
    - Burst reads/writes

```
__kernel void s1kmeans5b8(
  global uint16 *data, int n, int m,
  int k, float t, global uint16 *centroids,
  global uint16 *labels, global uint16 *min_dist)
{
  size_t gsz0 = get_global_size(0U);
  size_t gid0 = get_group_id(0U);
  int npoints = n/gsz0;
  int offset = gid0 * npoints;

  uint tmplabels[MAXPTS], tmpdist[MAXPTS]                    E1
  __attribute__(xcl_array_partition(cyclic,16,1));
  uint8 tmppts[TMPPTS], tmpcentr[8 * MAXK]
  __attribute__(xcl_array_partition(cyclic,2,1));

  for(int i = 0; i < k/2; i++) {                             E2
    // All burst reads and writes
    // are performed with the maximum data
    // width, regardless of N, D, or K
    uint16 tmpread = centroids[i];
    tmpcentr[(i*2)+0] = tmpread.lo;
    tmpcentr[(i*2)+1] = tmpread.hi;
  }
```

(to be continued...)

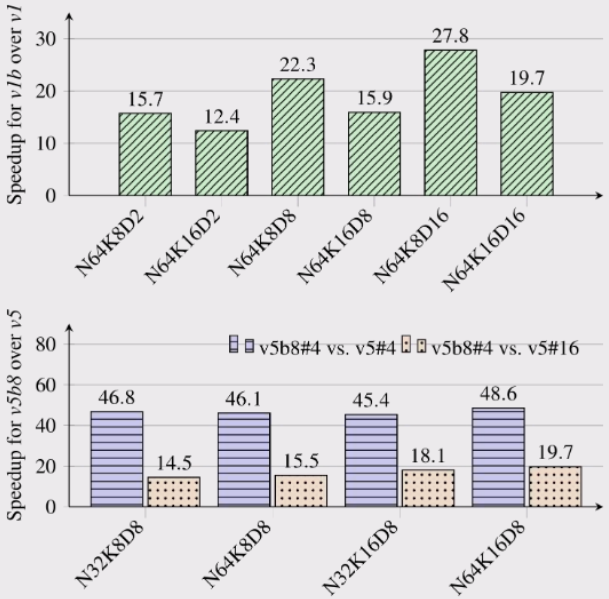
(...continued)

```
  int ptctr = TMPPTS;                                        B
  for(int h = 0; h < npoints; h++) {
    if(ptctr == TMPPTS) {                                    E3
      ptctr = 0;
      for(int j = 0; j < TMPPTS/2; j++) {
        int idx = ((offset + h)/2) + j;
        uint16 tmpread = data[idx];
        tmppts[(j*2)+0] = tmpread.lo;
        tmppts[(j*2)+1] = tmpread.hi;
      }
    }

    (...) // for every centroid                             C
      // adapt D segment in kmeansv2/v3                     D
      // to resort to "tmppts" and
      // "tmpcntr" to compute distances
      (...) // compare dist with mindist

    ptctr++;
  }

  int i = 0, idx = (offset/16);                              E4
  for(i = 0, j = 0; i < npoints; i += 16, j++)
    labels[idx + j] = *(uint16 *) &(tmplabels[i]);

  for(i = 0, j = 0; i < npoints; i += 16, j++)
    min_dist[idx + j] = *(uint16 *) &(tmpdist[i]);
}
```

Version with vectorization, local partitioned memories, and burst accesses

Example: OpenCL (2/3)

- 725x Over the baseline
  - By combining loop pipelining, burst memory accesses, and vectorization
  - But the code **needed a lot of work**
  - And features outside the OpenCL standard are needed… (e.g., partitioning attributes)
- But more cost effective!
  - CPU: $450 on release (2014)
  - FPGA: $2700… but 1.5x faster and 4.8x less power



Speedup for v1b and v5b8 vs. the respective versions without burst optimization

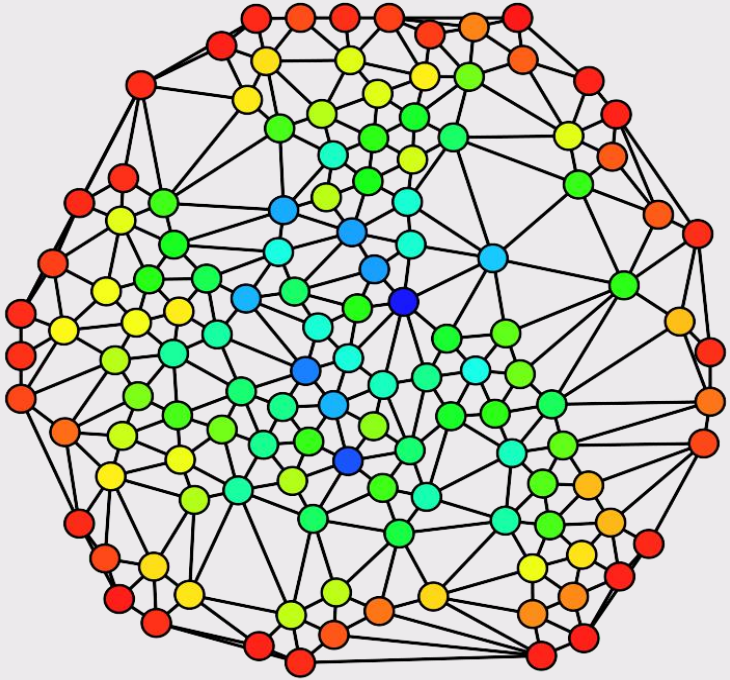
05. Witness Testemonies

Pedro Silva

FEUP, MIEIC

FPGAs as Accelerator for Graph Analysis Algorithms

- Impressions from a user perspective
  - *“Lots of boilerplate”*
  - *“Leaky abstractions”*
  - *“Thinking outside the Sofware Engineering box“*
  - *“Slow compilation times”*
  - *“Breaking through the C abstraction (when it doesn’t fall apart on its own, see above)”*



Source: WIkipedia Commons

- Accelerating Graph Centrality Algorithms on FPGAs via HLS
  - Relatively unexplored on FPGAs
  - Can they be easily expressed through HLS abstractions?
  - How central is each node to the graph?
  - Uses algorithms like *Shortest Path*
    - Up to x100 slower than GPU, using out-of-the box Xilinx libs.

Tiago Santos

FEUP, MIEIC

Automatic Insertion of High-Level Synthesis Directives

- Impressions from a developer perspective
  - *“Instability between versions”*
  - *”Scattered documentation”*
  - *”Compile times”*
  - *“Which directives to chose?”*
  - *“How to configure directives?”*
  - *Can the process be **automated**?*

```
#define N 2000
void computeGrad(float grad[N],
float feature[N], int scale) {
    for (int i = 0; i < N; i++)
        grad[i] = scale * feature[i];
}
```

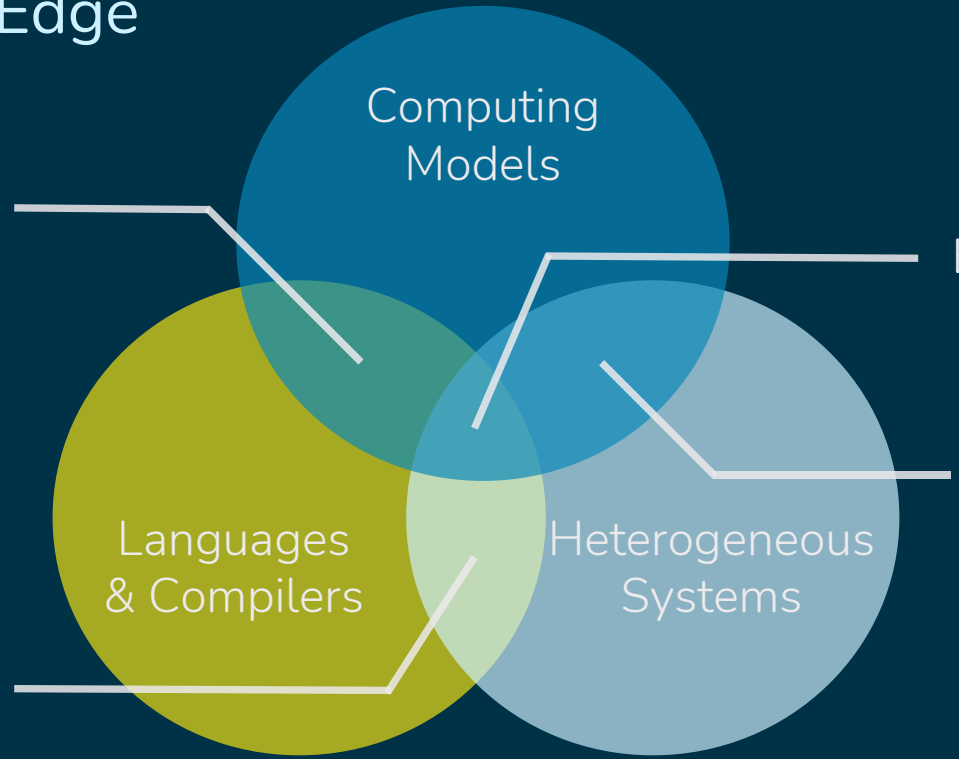
Add Secret Sauce ↓ CLAVA

```
#define N 2000
void computeGrad(float grad[N],
float feature[N], int scale) {

#pragma HLS array_partition
  variable=grad cyclic factor=32
#pragma HLS array_partition
  variable=feature cyclic factor=32

    for (int i = 0; i < N; i++)
#pragma HLS unroll factor=32
#pragma HLS pipeline
        grad[i] = scale * feature[i];
}
```

Master's Thesis: https://hdl.handle.net/10216/128984

- Do acceleration candidates need improvement for better HLS?
  - Iterations assumed sequential
  - Parallelism needs to be exposed
  - **Automatic annotation** with HLS directives
    - Using Source-to-Source tool Clava
  - 3x to 58x latency improvements

Living on the Edge



DSLs & Libraries

APIs

Co-Design:
Better Performance
Lower Power

Examples:
FPGA Streaming
FPGA SIMD
GPU SIMT

Thank you!

Stay tuned for REC'2021!

Nuno Paulino
INESC TEC
nuno.m.paulino@inesctec.pt