

Introduction and Lab1

Aaron Zhao, Imperial College London

General Introduction

Introduction - Myself

My name is Aaron, and my research looks at the intersections between algorithm, hardware and security in Deep Learning (DL) Systems.

My email is a.zhao@imperial.ac.uk, and my office is 903.

Introduction - Myself

- Automatic Co-designing AI Systems with MASE**

MASE aims to provide a unified representation for software-defined ML heterogeneous system exploration.

- Beyond Structure Data**

Investigate on unstructured and multimodal data, such as graphs, hypergraphs and combinatorial complex.

- Efficient AI**

Different algorithmic aspects of efficient AI, including efficient training, efficient inference, efficient model search and efficient model deployment with state-of-the-art GenAI models (eg. language and diffusion models).

- System-level AI Safety**

This includes robustness, security, and red-teaming these models to understand new vulnerabilities.

Introduction - Teaching and RAs

The course is taught by me and Sarim Baig:

- I cover the FPGA parts
- Sarim covers the software parts

This course is also supported by the following GTAs and UTAs:

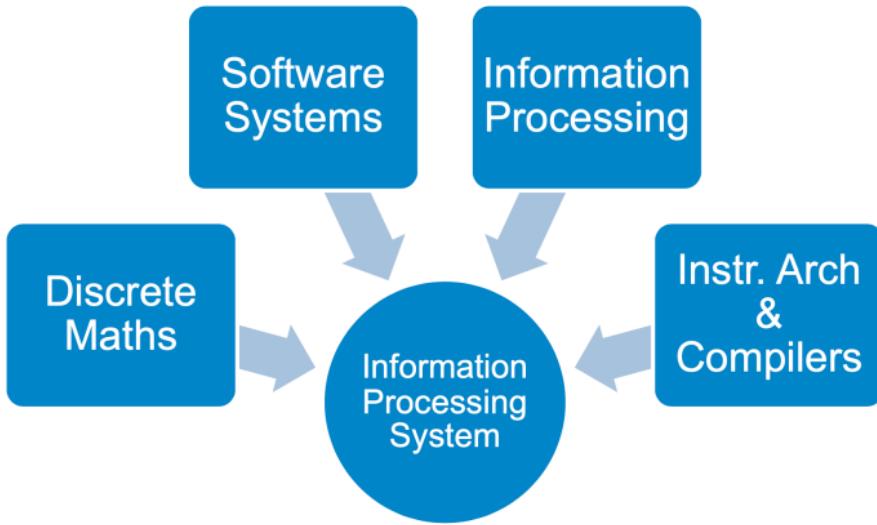
- Cheng Zhang
- Jeffrey Wong
- Alexander Charlton
- Rahul Ganeshsankar

Introduction - Where to find stuff?

Everything is online:

- Course webpage (https://aaron-zhao123.github.io/teaching/info_process/)
- The Labs (1-4) are in a Github Repository (<https://github.com/Aaron-Zhao123/ELEC50009>)
 - Do not push to this repository
 - You can fork it and push to your own repository
- If you do not understand what is push and fork, check this link: https://www.youtube.com/watch?v=nT8KGYVuriU&ab_channel=TheCodex

Objectives and delivery



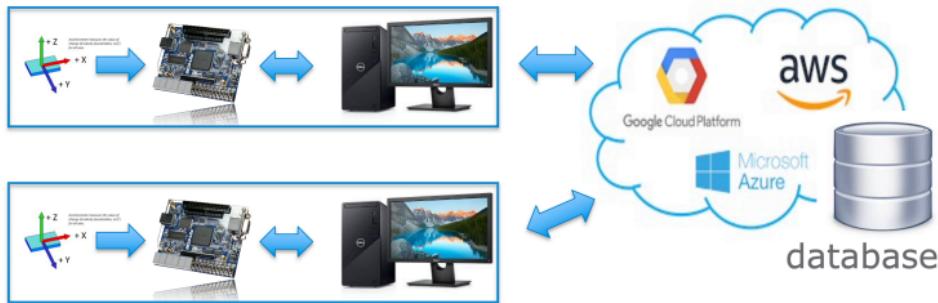
- Bring together theory and application from other modules
- Create an information processing system
- Project-based learning and integration of knowledge

Intended learning outcomes

- Design an information processing system that captures, analyses, manages, and outputs signals
- Implement an information processing system using a combination of software, hardware, networks, and databases
- Optimise a system to achieve given performance or quality targets

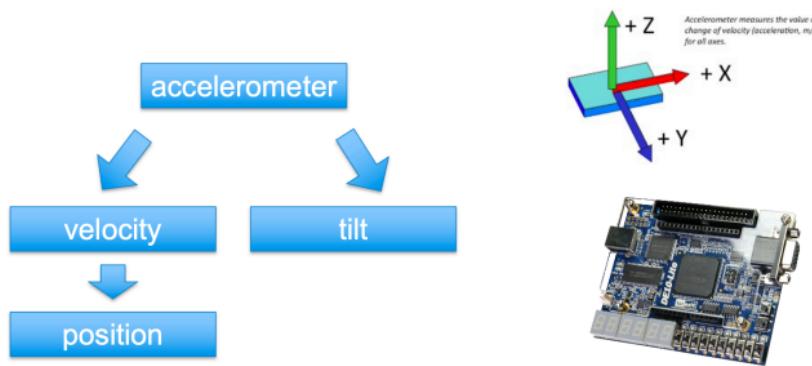
Let's be more specific: design an IoT system

- Nodes for local (signal) processing of accelerometer data
- Communication to a server
- Integration with a database
- Adapt processing in nodes



Let's be more specific: accelerometer

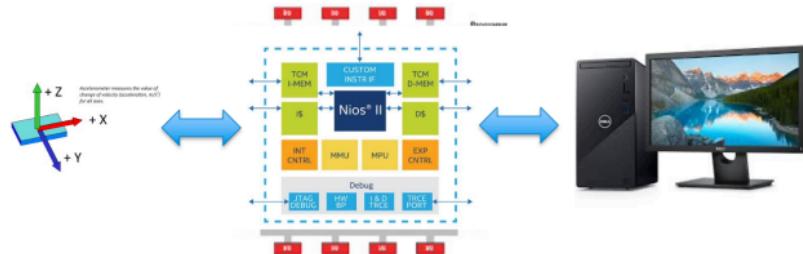
Accelerometer measures the value of change of velocity (acceleration)



Let's be more specific: the development board

DE10-lite

- FPGA (Intel)
- Instantiation of a soft processor (NIOSII)
- Processing capability (can perform computation) and communication capability (talk to a local PC)



Let's be more specific: AWS DB

- Instantiate and use a database on the cloud
- Communicate from the host PC to the database through network



Structure and dates

- Phase 1 - training (Week 2 - Week 5)
 - ▶ Lab based
 - ▶ To help you to build an understanding of the system
 - ▶ with GTAs for Q&A
- Mid-term Assessment (20%) (Week 6)
 - ▶ Lab orals
 - ▶ Completion and understanding of the labs
 - ▶ Assessed as a group
- Phase 2 - group project (Week 7 - Week 10)
 - ▶ Functional requirements
 - ▶ Non-functional requirements
 - ▶ GTA help
- Final Assessment (80%) (Week 11)

Phase 1 - Training

- Week 2 (Aaron)
 - Lab 1: Introduction to DE10-Lite. Install tools and learn how to program the device
 - Lab 2: Instantiate a NIOSII system, use the accelerometer
- Week 3 (Aaron)
 - Lab 3: Establish a UART-base communication between the board and the desktop PC
 - Lab 4: Design an IP module for performing a moving average in HW. Connect to NIOSII and process the accelerometer data
- Week 4 (Sarim)
 - Lab 5: Create a remote server in AWS and run a custom service
- Week 5 (Sarim)
 - Lab 6: Create a remote database and perform queries

Phase 2 - Team project

- In-person and remote working support
- General idea
 - Local node needs to talk to a server (on the cloud).
 - Server needs to talk back, the information needs to propagate back.
- Elaboration
 - Try to see how nodes can affect each other.
 - Detect events, and change the processing in the nodes through a centralised server.
- Log your events, or perform actions on the events.
- Detailed functional and non-functional requirements will be communicated

Logistics

- Lectures (Every Tuesday Weeks 2-4, then in ad-hoc basis as needed)
- Weekly support hours for the Team Projects.
 - Prepare your questions
- Groups and Communications
 - Groups of five or six, have to all come from either group A or B.
 - If you cannot find a group, we will make one for you
 - You should start making a group now, deadline for this was 13th January
 - We will provide a finalized group list by the end of this week.
- Course Material
 - Teams
 - Coursework wiki and Github Repo

An introduction to FPGAs

Why an FPGA is an interesting device to consider

FPGA sells more than 40 million units per year, and is used in many applications, and we have the following vendors:

- AMD (Xilinx): around 50% of the market
- Intel (Altera): around 30% of the market
- Microsemi
- Lattice
- and more

Why an FPGA is an interesting device to consider



Figure 1: Xilinx Virtex chip (left), Xilinx Pynq Dev Board (middle), Xilinx Alveo accelerator card (right)

Similar to all other chips, FPGAs can be used in many applications and exist in many forms, such as on a development board (eg. Pynq), or as an accelerator card (eg. Alveo).

The growing need of energy efficient computing

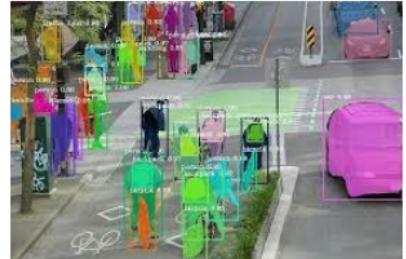


Figure 2: Robotics (left), UAVs (middle), Vision systems in cars (right)

There is now an increasing need for high-performance systems on low power systems (eg. robotics, UAVs, vision systems in cars).

The growing need of energy efficient computing



Figure 3: Bio-computing (left), large-scale simulation such as weather prediction (middle), AI computing (right)

There is now an increasing need for high-performance systems on large-scale workloads too. This is normally in large-scale data centres, where energy efficiency is a key concern.

The “ideal” computing device

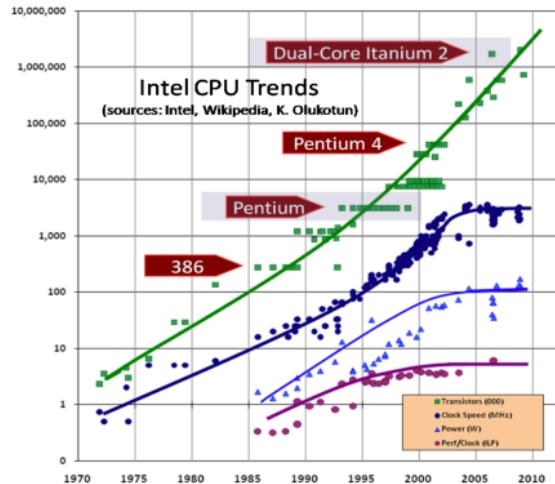
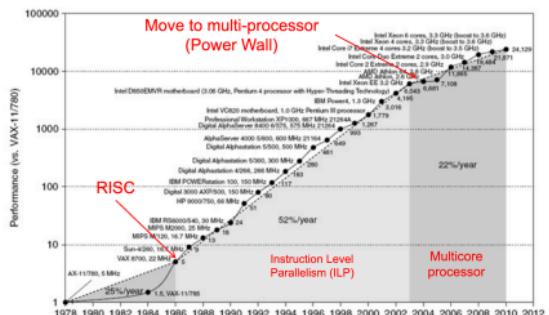
- Energy efficient
 - Better Ops/Watt/Second/Area
- Cost efficient
 - Better running cost, somehow it is the same as energy efficiency
 - Better manufacturing cost
 - Die area
 - Yield
 - Time to market
 - nm node

Approach 1

- Design or write more “efficient” algorithms
 - Better data structures
 - Better software engineering practices
 - Better compilers
 - Better hardware
- Use approximations
 - for instance, quantization or sub-sampling
- Consider the architecture of the system and optimize your program

Approach 2

Do nothing. Just wait for the next generation processor.



No more free lunch.

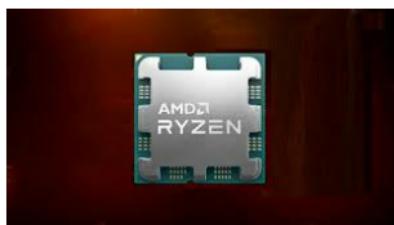
Approach 3

Use heterogeneous systems:

CPUs combined with other hardware (eg. parallel architectures, or ASICs) on the same SoC.

- Multi- or even Many-core CPUs (AMD Ryzen Threadripper PRO 7995WX, 96 Cores, 192 Threads 2.5GHz and max 5.1GHz)
- Graphic Processing Units (Nvidia H100, 640 Tensor Cores, 128 RT Cores, 80 Streaming Multiprocessors (SMs) and 18,432 CUDA cores)
- Field Programmable Gate Arrays (around 2M Logic Elements/LUTs, around 5K DSPs)

Approach 3 (ii)



The flexibility and performance trade-off



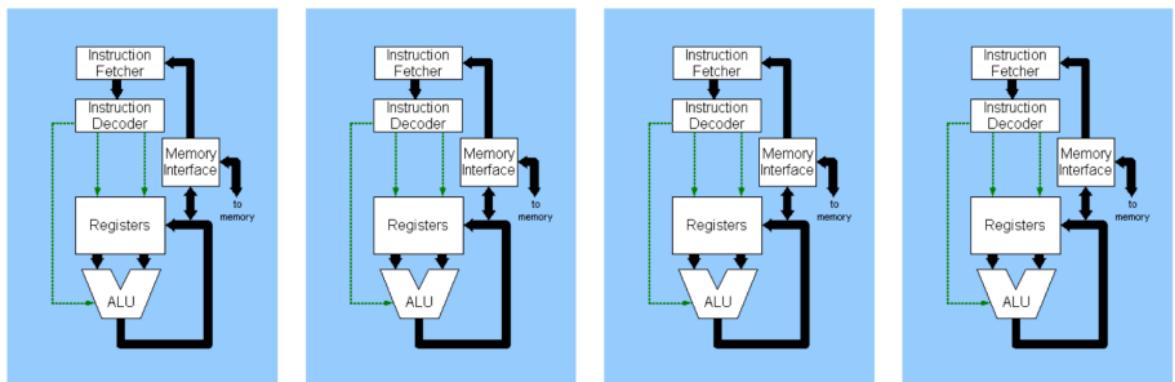
,

- The left-hand side may have better power efficiency on the specific tasks.
- The right-hand side is more flexible and requires shorter dev cycles and efforts.

Benefits come from customising the hardware to the application, and also by tuning your application for the hardware, this is also known as **software-hardware co-optimization**.

Device comparison: multi-core CPUs

- Each core is fairly powerful and runs at a very high frequency.
- Complex memory hierarchy, eg. L1, L2, L3 caches.
- Up to linear speed up (extremely optimistic!).

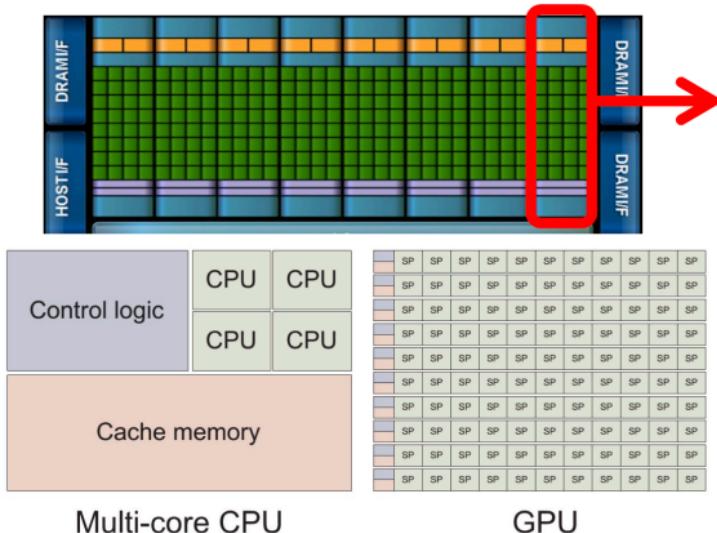


Device comparison: GPUs

- Many light-weight thread processors (SMs, in Nvidia world) => Hide memory latency.
- All thread processors execute the same sequential code.
- SIMD architecture
 - massive data parallelism.
 - optimized memory access

Existing GPUs take advantage of the SIMD architecture, and are optimized for massive data parallelism. They also take over the top manufacturing process (eg. 3nm at TSMC). They are currently dominating the AI computing market.

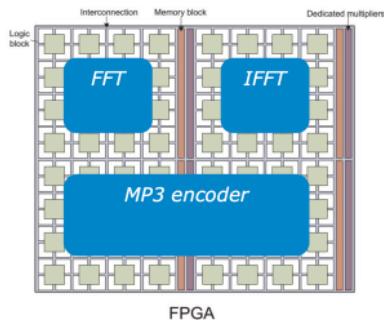
Device comparison: GPUs (ii)



**1 multiprocessor
=
32 thread
processors**

Device comparison: FPGAs

- (Re-)programmable digital hardware – can implement any digital circuit.
- Can exploit low-level pipeline parallelism Logic blocks evaluate simple Boolean functions.
- Interconnection resources connect blocks to implement complex systems.
- One way to understand is this is computation in “space” rather than “time”.



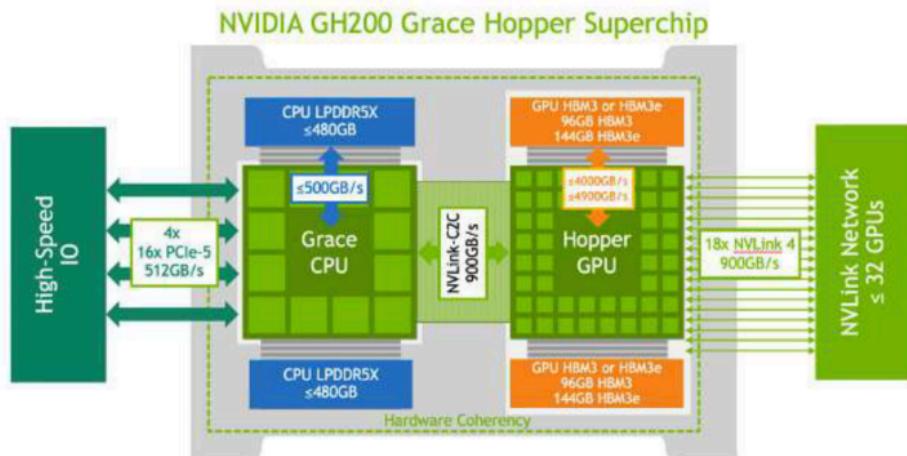
Computation in space

,
What price do you think we are paying for this reconfigurability?

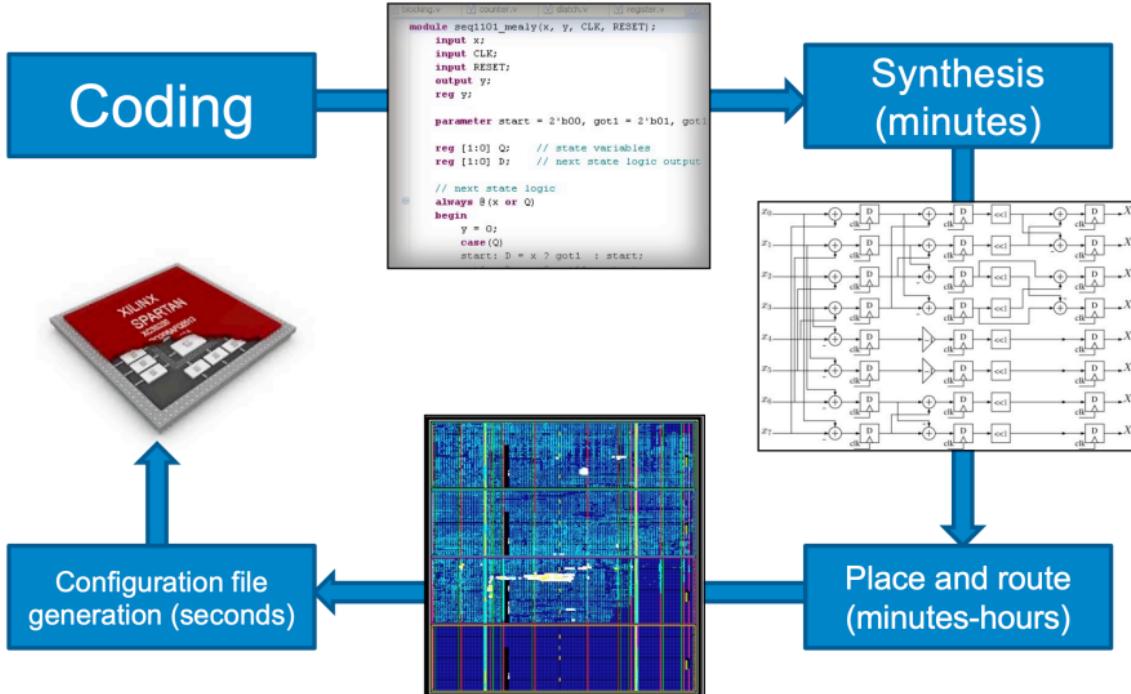
Heterogeneous computing

Normally in today's market, we see a combination of CPUs, GPUs, and FPGAs in the same system. This is known as a heterogeneous system. And this integration can actually happen at various granularities:

- Intel Xeon Gold 6138P with Arria 10 FPGA: server-scale CPU with FPGA
- NVIDIA GB200 architecture: Grace CPU with Hopper GPU



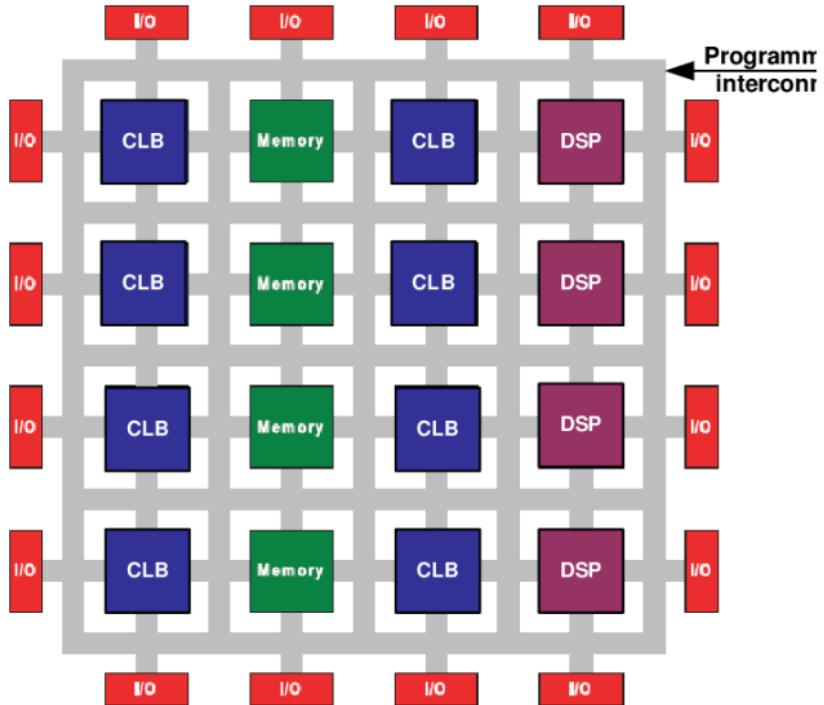
FPGA design flow



Field Programmable Gate Arrays (FPGAs)

- Xilinx (now part of AMD) is the first to introduce SRAM based FPGA using Lookup Tables (LUTs)
- Components
 - Configurable Logic Block (CLB)
 - Input/Output Blocks (IOBs)
 - Programmable Interconnects
 - DSPs
 - Block RAMs

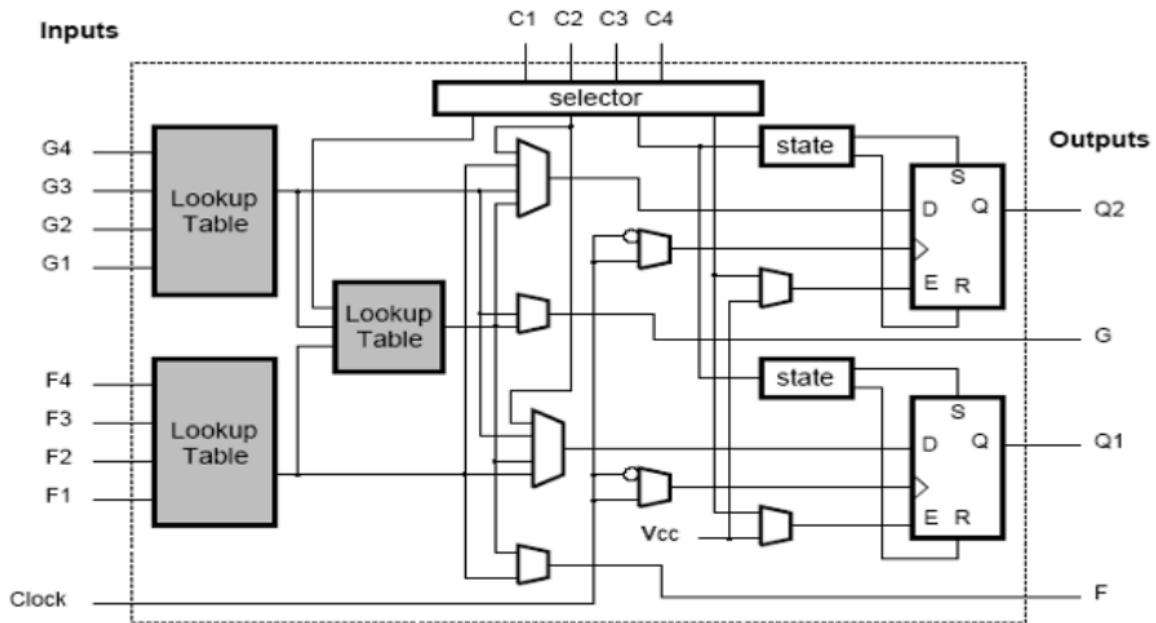
Field Programmable Gate Arrays (FPGAs) (ii)



CLBs

- Each Configurable Logic Block (CLB) has 2 main Look-up Tables (LUTs) and 2 registers.
- The two LUTs implement two independent logic functions F and G.
- Shown here is the CLB for Xilinx XC4000 devices.

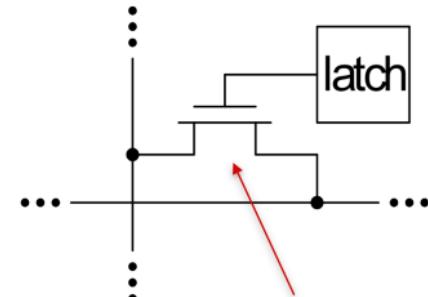
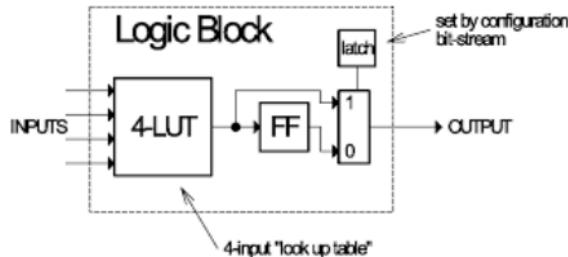
CLBs (ii)



LUTs

- LookUp Table (LUT) is implemented using latches
 - 4-LUT (i.e. 4-input LUT) implements any truth table with 4 inputs, this constructs combinatorial logic functions
 - Requires 24 storage elements, each implemented with a latch (similar to a flip-flop, but half the size roughly, 1-bit memory)
 - Multiplexer select one latch to output
 - “Configuration bit stream” is loaded under user control

LUTs (ii)

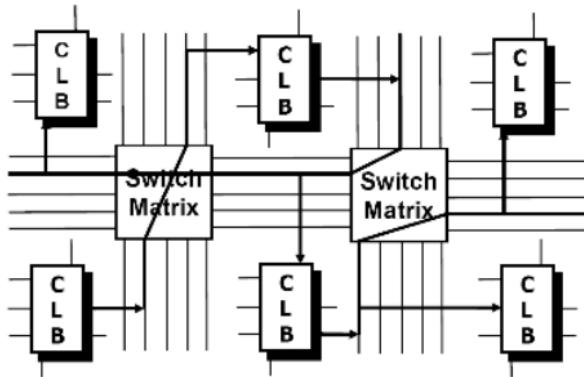


- + reconfigurable
- volatile
- relatively large.

Typical interview questions: how many 4-LUTs are needed to implement an 8-bit adder?

Programmable interconnect

- Switch-box provides programmable interconnect
 - Local interconnects are fast and short
 - Horizontal and vertical interconnects are of various lengths

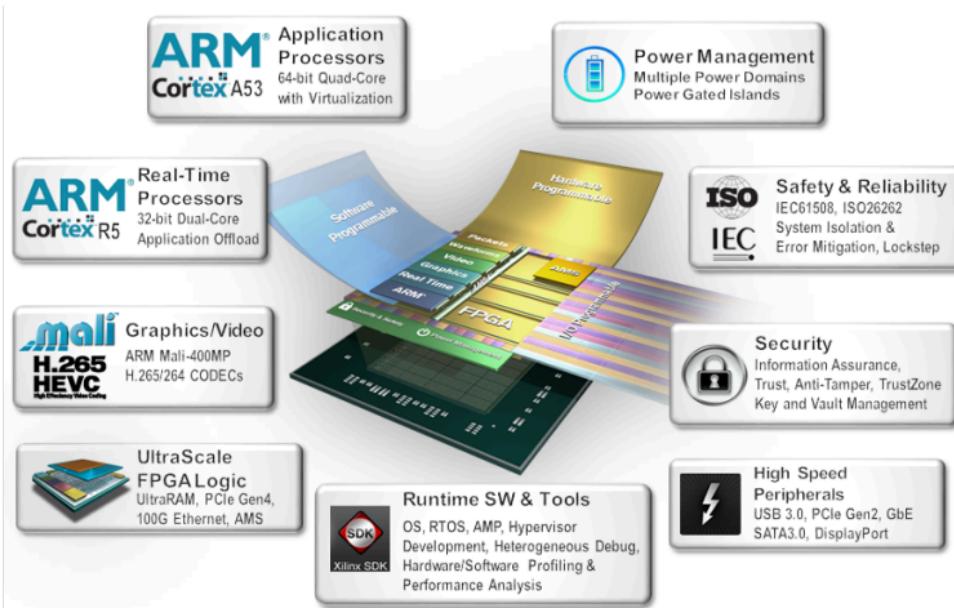


End of the Dinosaurs Age



Modern FPGA devices -- heterogeneous

Pre-built ASIC components are already integrated



Current available FPGAs from Intel



Intel® Agilex™ FPGAs
Intel® Agilex™ FPGA family, built on 10nm technology, enables customized acceleration and connectivity for a wide range of compute and memory intensive applications while providing an improvement in performance and reduction in power.



Intel® Stratix® Series
The Intel® Stratix® FPGA and SoC family enables you to deliver high-performance, state-of-the-art products to market faster with lower risk and higher productivity.



Intel® Arria® Series
The Intel® Arria® device family delivers Intel® performance and power efficiency in the midrange.

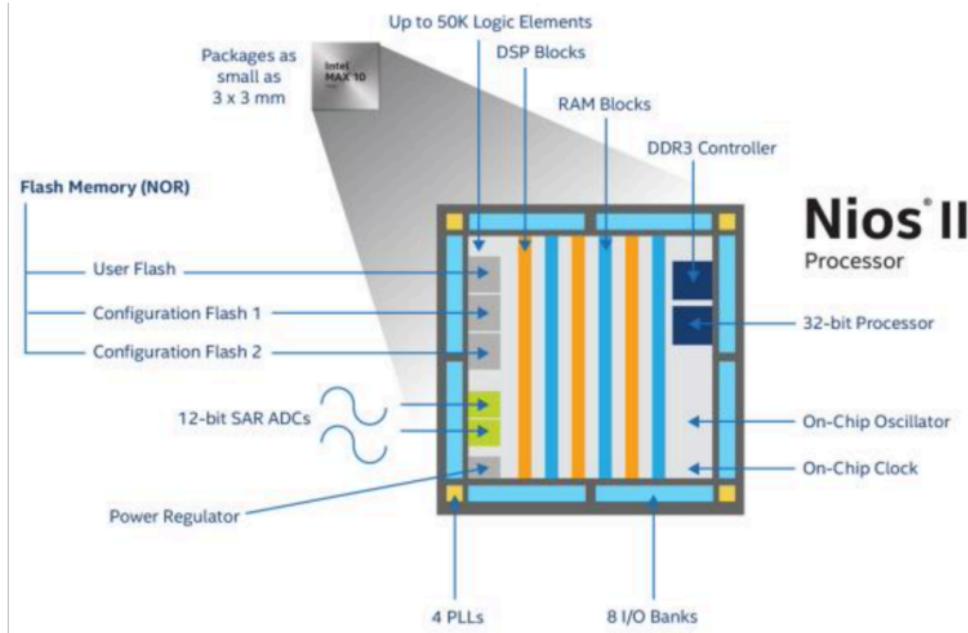


Intel® Cyclone® Series
The Intel® Cyclone® FPGA series is built to meet your low-power, cost-sensitive design needs, enabling you to get to market faster.



Intel® MAX® Series
The Intel® MAX™ 10 FPGA revolutionizes non-volatile integration by delivering advanced processing capabilities in a low-cost, single chip small form.

Current available FPGAs from Intel



Questions?

Any questions?

An introduction to Lab1

What is in this lab?

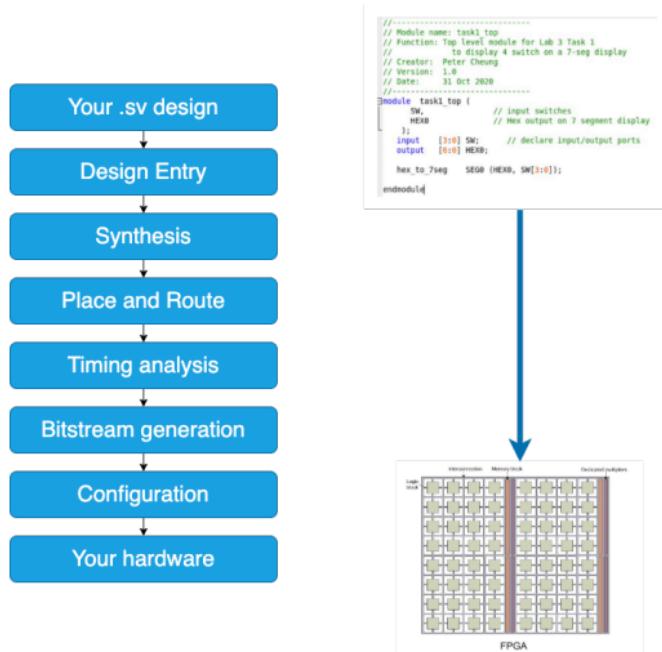
- Setting up Quartus Prime Lite to run (Maybe actually harder than you think!).
- Create a directory structure for this and subsequent labs.
- Create a new project in Quartus and complete a basic 7-segment LED display decoder design using Quartus and Verilog.
- Program the MAX10 FPGA chip on the DE10-Lite board with your design.
- Understand the FPGA compilation process.
- Create another project for hex-to-BCD decoding.
- Explore and test your design.

Setting up your environment

- Quartus Prime (preferred setup is in the VirtualBox or lab machine).
- Explore the programmer: this uses the USB-blaster to program the FPGA from your host machine.
- Explore the Ping Assignment tool.
- Netlist Viewer and Timing Analyzer.
- Be careful with your directory structure!

FPGA Compilation

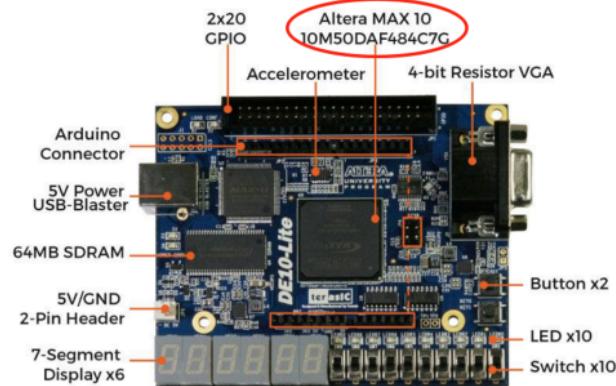
You will have to go through a number of steps to map your design to the actual FPGA.



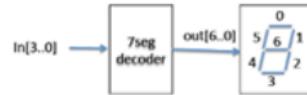
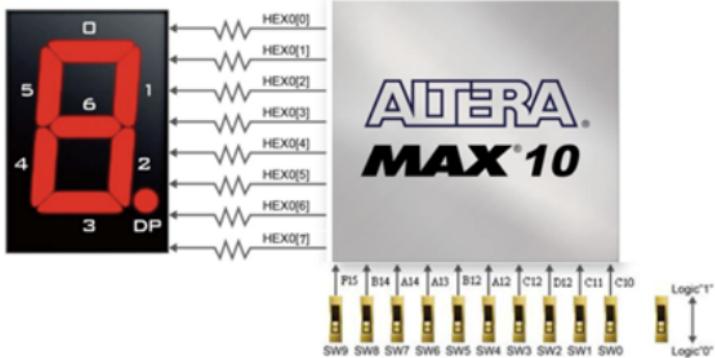
Your Design

In lab 1, you will design a 7-segment LED display decoder. This is a simple design that will help you to understand the FPGA design flow.

We will use four switches to input a 4-bit binary number, and the 7-segment display will show the corresponding decimal number.



Your Design (ii)

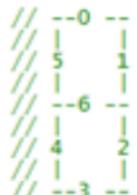


in[3..0]	out[6..0]	Digit	in[3..0]	out[6..0]	Digit
0000	1000000	0	1000	0000000	0
0001	1111001	1	1001	0010000	9
0010	0100100	2	1010	0001000	A
0011	0110000	3	1011	0000011	b
0100	0011001	4	1100	1000110	C
0101	0010010	5	1101	0100001	d
0110	0000010	6	1110	0000110	E
0111	1111000	7	1111	0001110	F

Your Design (iii)

```
module hex_to_7seg (out,in);
    output [6:0] out; // low-active outputs
    input [3:0] in; // 4-bit binary inputs
    reg [6:0] out; // make out a variable

    always @(*) begin
        case (in)
            4'h0: out = 7'b1000000;
            4'h1: out = 7'b1111001;
            4'h2: out = 7'b0100100;
            4'h3: out = 7'b0110000;
            4'h4: out = 7'b0011001;
            4'h5: out = 7'b0010010;
            4'h6: out = 7'b0000010;
            4'h7: out = 7'b1111000;
            4'h8: out = 7'b0000000;
            4'h9: out = 7'b0011000;
            4'ha: out = 7'b0001000;
            4'hb: out = 7'b0000011;
            4'hc: out = 7'b1000110;
            4'hd: out = 7'b0100001;
            4'he: out = 7'b00000110;
            4'hf: out = 7'b0001110;
        endcase
    endmodule
```



Verilog 101

- Verilog is a hardware description language (HDL) used to model electronic systems.
- The first line above defines the module name and the ports, we take an input of 4 bits and output 7 bits.
- The always block is a sequential block that is triggered by nothing, that means, this is a combinational logic.
- The case statement is a conditional statement that will output the corresponding 7-segment display.

Synthesis and Compilation

- Synthesis is the process of converting the Verilog code into a netlist.
- Compilation also includes the process of mapping the netlist to the FPGA, we call this the place and route process.
- For these steps, you will use the Quartus Prime software, and you normally would need to provide the FPGA device target for the tool, the constraints, and the pin assignment.

Ping assignment

- The ping assignment is a file that tells the tool how to map the pins of your design to the outside world.

Signal Name	Pin Location
HEXO[6]	PIN_C17
HEXO[5]	PIN_D17
HEXO[4]	PIN_E16
HEXO[3]	PIN_C16
HEXO[2]	PIN_C15
HEXO[1]	PIN_E15
HEXO[0]	PIN_C14
SW[3]	PIN_C12
SW[2]	PIN_D12
SW[1]	PIN_C11
SW[0]	PIN_C10

Timing Analysis

- The tool provides a timing analysis report that tells you how fast your design can run, in terms of clock rate (also known as FMax).
- This is a critical criteria for your design, as it tells you how fast your design can run, and normally also impacts your latency.
- Timing models in place
- Tools check all possible paths

