Hw/Sw-Codesin

Create CPU+IP Systems

José T. de Sousa

Técnico

January 14, 2026

What is This Course About?

- Designing complete computing systems: **hardware + software**
- Understanding **where** to run functionality: CPU or custom hardware
- Joint optimization of:
    - Performance
    - Power and energy
    - Area and cost
    - Flexibility
- Learning how modern SoCs are built
- Implementing real systems on a **Zynq-based FPGA board**

Why Hardware/Software Codesign?

- Modern applications are increasingly **compute-intensive**
    - Signal processing
    - Computer vision
    - Machine learning
    - Cryptography
- Pure software solutions are often:
    - Too slow
    - Too power-hungry
- Pure hardware solutions are often:
    - Inflexible
    - Expensive to modify
- Codesign allows us to combine the **best of both worlds**

Course Learning Objectives

- Understand the fundamental principles of **hardware/software codesign**
- Learn how to analyze applications and identify performance bottlenecks
- Decide which parts of a system should run in:
  - Software on a CPU
  - Custom hardware accelerators
- Design, implement, and integrate accelerators using:
  - Vitis HLS
  - Verilog RTL
- Deploy and evaluate complete systems on a Zynq-based FPGA board

Tools and Experimental Platform

- This course is **hands-on** and tool-oriented
- You will work with industry-standard tools:
  - **Vivado** – RTL design, synthesis, implementation
  - **Vitis HLS** – C/C++ to hardware synthesis
  - **Verilog** – low-level hardware design
- Target platform:
  - Zynq SoC (ARM CPU + FPGA fabric)
  - Zybo development board
- Software will run in a **bare-metal** environment

What Is a Computing System?

- A computing system is more than just a processor
- It is composed of multiple interacting components:
  - Processing elements (CPUs, GPUs, accelerators)
  - Memory hierarchy (registers, caches, RAM, external memory)
  - Interconnects and buses
  - Input/Output devices
  - Software stack
- Performance depends on how well these components work **together**
- Codesign focuses on optimizing the **whole system**

Outline

- Introduction
- Project setup
- Instantiate a RISC-V CPU in IOb-SoC
- Instantiate an IP core in your SoC
- Write the software to drive the IP core
- Simulate IOb-SoC
- Run IOb-SoC on an FPGA board
- Conclusion

Figure: IOb-SoC block diagram

Introduction

- Building processor-based systems from scratch is challenging
- The IOb-SoC template eases this task by providing a base Verilog SoC equipped with
  - a RISC-V CPU
  - a memory system including boot ROM, RAM, 2-level cache system and an AXI4 interface to external memory (DDR)
  - a UART communications module
  - an example firmware program
- Users can add IP cores and software to build their SoCs
- This tutorial exemplifies the addition of a Timer IP core and the use of its software driver

Project setup

- Use a Linux real or virtual machine (see the README file to download a VM)
- Install `nix-shell` to deal with dependencies, especially open-source simulators such as `iverilog` or `verilator`
- Commercial EDA tools must be installed locally or on some remote server (Vivado, Quartus, Cadence, etc)
- FPGA boards must be attached to your Linux machine or to some remote server
- Set up **ssh** access key to GitHub (`github.com`) (using https will ask for your password many times)
- Follow the instructions in the IOb-SoC repository's README file to clone the repository and install the tools