XILINX FPGA design software

(ISE 14 + QuestaSim)

Laboratory project 2

The XILINX ISE design flow for FPGAs

for non-digital engineers AND ONLY to build simple and unconstrained designs

- **Write the Verilog code**
- **Execute the process "Generate programming file"**
- **Configure the FPGA *et voilá* !**

- What is a "simple … design" ?
  - Something small, clocked synchronous, like a counter, a small finite-state machine, a multiplexer, etc
- What is a "… unconstrained design" ?
  - Any design for which the timing and area (size) requirements are very relaxed when compared to the limits imposed by the technology, for example a 8-bit counter running at 1 MHz and occupying only 0.2% of the FPGA resources

This is not for us !

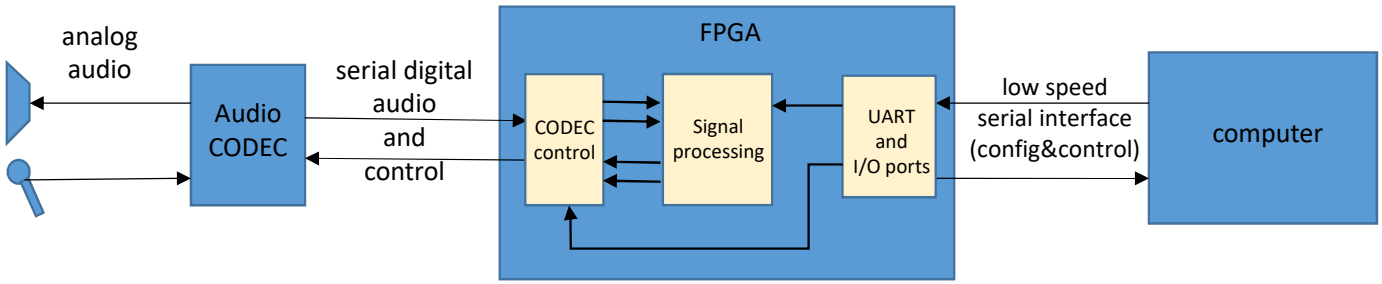
We want to learn how to design complex digital circuits pushing the limits of the technology

Design flow (simplified)

- **Design the digital circuit at the RTL level**: abstract block diagram, define the system hierarchy, set main design constraints and goals
- **Build the RTL digital models** of each sub-circuit (or *modules,* in Verilog HDL)
- **Verify each individual module** to check their functionality (functional simulation)
- **Assemble the whole circuit** by building the toplevel module
- **Build a toplevel testbench and verify** the toplevel circuit
- **Synthesize the whole circuit**: set timing constraints, iterate with different synthesis options to meet the design goals; if needed redesign the RTL code
    - Look at the synthesis warnings: look for latch inference, unconnected outputs, unplanned logic trimming, pin-net width mismatch,…
- **Verify the circuit after synthesis** (post-translate simulation), look for possible errors due to wrong interconnections, bus widths, …
- **Place and route the design** (physical synthesis): set timing constraints, I/O placement and electrical constraints, optimization effort
- **Check the static timing reports**: which parts of the design are not meeting the timing constraints ?
    - Static timing analysis (STA) calculates the propagation delays of all flop-to-flop paths
- **Verify the circuit after P&R** (post-route simulation). This includes the most accurate timing models of the logic blocks and interconnections. Timing information is in file “xxx.sdf” (standard delay format)
- **“*Fabricate*” the circuit: generate the bitstream file** to configure the FPGA

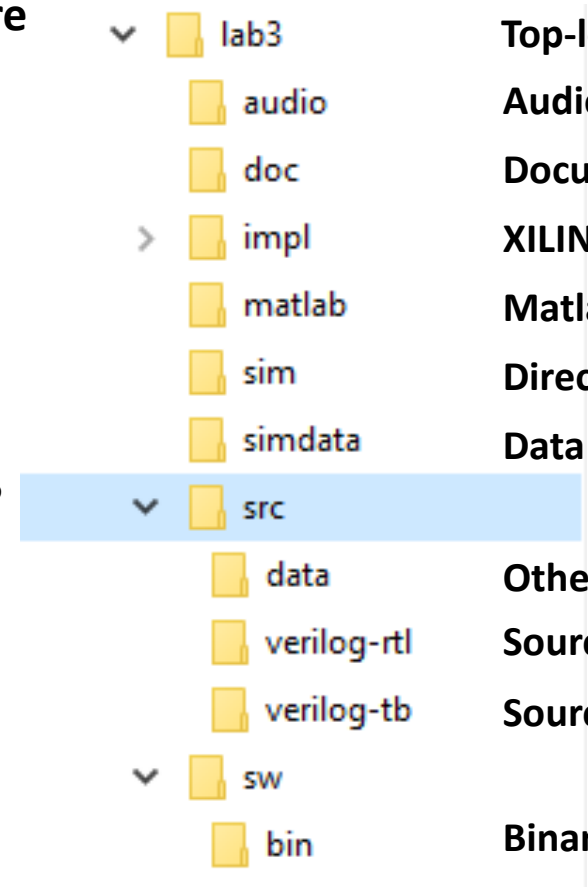
Reference project – digital stereo audio

- Receives two streams of high quality digital audio
  - From an external audio CODEC connected to an audio source
  - 18 bits/sample, two's complement, 48 kHz sampling frequency
- Performs a simple signal processing operation
  - Calculates the sum and the difference of the left and right channels
- Outputs two streams of digital audio in the same format
  - To the external CODEC, connected to headphones or loudspeakers



Design kit (PSD2021-lab3.zip)

- **Directory structure**



Top-level project directory

Audio samples (should be within “simdata”)

Documentation

XILINX implementation directory

Matlab/Octave code for creating simulation data

Directory for running standalone simulations

Data files for simulation

Other sources (design constraints)

Source files for implementation

Source files for verification

Binaries for controlling the system from a Windows PC

KEEP ALL FILES IN

THE RIGHT PLACES

The project

- Build a digital signal processing circuit to perform the following tasks:
  - Programmable anti-aliasing low pass filter (architecture to be defined later)
  - Downsample the input audio stream to Fs/N (N integer)
  - Requantize the audio samples to K bits (K less than 18)
  - Interpolate the signal to increase the sampling frequency to Fs/M, (M/N integer)
    - Linear or quadratic interpolation (circuit architecture to be defined later)
  - Include multiplexers to include or not each block in the DSP chain
  - Parameters N, M, K and mux selection are different for the left and right channels
- The circuit must be synchronous with a single clock (12.288 MHz)
- Multiplications and divisions should be done with sequential circuits
  - RTL code available for free but you may use your own sequential divider