## Integrated Systems Architectures

# Lab 1

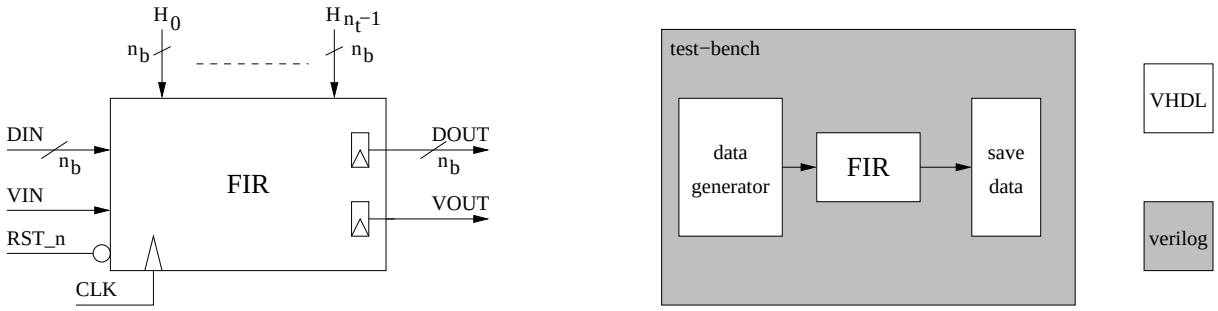


Figure 1

Design in VHDL the architecture of an FIR filter according to the following requirements:

1. order the surnames of the members of the group;
2. let $x$, $y$ and $z$ be the number of characters in the first three surnames, then obtain the number of taps for the filter ($n_t$), the number of bits to represent the input/output data $n_b$ and the number of pipeline levels $n_p$ as follows

$$n_t = (x \bmod 5) + 4 \quad (1)$$
$$n_b = (y \bmod 9) + 8 \quad (2)$$
$$n_p = (z \bmod 3) + 2. \quad (3)$$

Example (Bianchi, Rossi, Verdi, ...) leads to $x = 7$, $y = 5$ e $z = 5$ and so $n_t = 6$, $n_b = 13$ and $n_p = 4$. In case there are only two people (e.g. Bianchi, Verdi), wrap back to the first surname to obtain (Bianchi, Verdi, Bianchi) and apply the previous algorithm.

The filter interface is show in Fig. 1 (a): the samples ($DIN$) enter one each clock cycle with a validation signal ($VIN$) into a shift-register. When $VIN$='1' a new sample is loaded and the other ones are shifted into the shift-register. The shift-register contains $n_t$ samples. The samples in the shift-register ($x_i$) are multiplied by the filter taps ($H_j$) to obtain:

$$y_i = \sum_{j=0}^{n_t-1} x_{i-j} \cdot H_j \quad (4)$$

The implementation of (4) must take into account the $n_p$ parameter. The output $DOUT$ contains the result of the filtering ($y$) and $VOUT$ is a validation signal: $VOUT$='1' when $DOUT$ is ready. **Note:** the output of the FIR must be produced by registers, these registers **are not** part of the pipeline. Samples and taps are represented as 2-complement normalized-fixed-point values: the weight of the most significant bit is $-2^0$, the weight of the least significant bit is $2^{-n_b+1}$. Finally, prepare a hierarchical test-bench, where the signal generation is implemented in VHDL, whereas the top of the hierarchy is a verilog file (see Fig. 1 (b)).

**Note:** we suggest to avoid the use of the *generic* statement in the top entity of the filter. If you want to implement a parametric design we suggest to use a package.

**Step 1** Simulate and verify with Modelsim the system you described. Then, run a 100 random samples simulation and save the output samples in a text file. *Suggestion* structure your design environment as follows. Create a working directory (e.g. `lab1`). In `lab1` create:

- `src` as the directory containing the VHDL file/files of the filter;
- `tb` as the directory containing the VHDL and the verilog of the test-bench;
- `sim` as the directory containing the project files created by Modelsim and your simulation scripts (if any).

**Step 2** Synthesize the FIR with Synopsys Design Compiler. Set $f_{clk} = 100$ MHz, find the area of your design. Verify the netlist via simulation and obtain the switching activity to estimate the power consumption of your design. *Suggestion* structure your design environment as follows. Create in `lab1`:

- `syn` as the directory containing the synthesis scripts and the files produced by the logic synthesizer;
- `netlist` as the directory containing the output of the logic synthesizer.

*Optional:* find the maximum clock frequency your design achieves.

**Step 3** Place and route the FIR with Cadence SOC Encounter. Set $f_{clk} = 100$ MHz, find the area of your design. Verify the netlist via simulation and obtain the switching activity to estimate the power consumption of your design. *Suggestion* structure your design environment as follows. Create in `lab1`:

- `soce` as the directory containing the place and route files and results