FPGA Generic Fixed Point FIR Filter Implementation

BORJA MIGUEL PEÑUELAS MORALES

Borja.Penuelas@gmail.com
(+34 696084742)

1. Implementation

This is how each one of the requirements were met:

1- Order

The Parallel FIR architecture and the testbench are designed to accept an arbitrary number of taps and thus any filter order. Since it is fully pipelined, increasing the order of the filter will not affect the maximum operating frequency, only the process latency.

2- Input samples width, coefficients width, output samples width

Analogously, it supports any combination of input, coefficient and output widths, internal widths are optimally adjusted.

3- Sampling frequency, clock frequency

Any given combination of sampling and clocking frequency requires n samples parallel processing. The filter is designed as a MIMO system that can be configured to process any number of samples in parallel.

4- Pipelined for minimum combinational delay

Every stage of the architecture is designed so that only one combinational logic operation is performed between registered stages. Since the filter was designed for a configurable number of input samples in parallel, the parts where combinational operations grow linearly are automatically divided in stages where the combinational logic delay remains constant, and the latency grows with log2(n parallel inputs).

5- Matlab fixed point filter implementation

A simple fi model of the filter was implemented in Matlab to generate reference output data. Fixed binary point positions where chosen so that the range was not affected across the calculations.

6- Self-checking VHDL testbench

The testbench reads the reference output data generated by the Matlab script and compares it against the outputs of the Device Under Test. If at any moment the DUT output did not match exactly the model data, the simulation would be halted and the reason of the failure would be reported. (If the simulation finishes "by stop-time", it means that the assertions did not stop the simulation and the VHDL verification has been completed successfully).

2. Other improvements

1- Software debugging

Additionally, for convenience while debugging, output samples from the VHDL testbench are written to a file and read by the Matlab script. Samples are compared here again against the ones generated by the Matlab model and also displayed in a PSD.

2- Automated simulation run

Instead of using conventional simulators such as Modelsim, since this is an all-VHDL project, I used a compiler-simulator (GHDL), which runs orders of magnitude faster.

To prepare and control the GHDL simulation, a Python 3 program was written. This in turn is controlled from Matlab so that the script is in full control of the process. The characteristics of the filter can be changed in Matlab and all the simulation and verification chain will adapt accordingly.

3- FIR symmetrical architecture

The provided coefficients for a linear phase response filter allowed a symmetrical FIR filter architecture, however, the VHDL implementation was designed to be able to implement both, symmetrical and normal architectures. It will detect automatically whether if the coefficients are symmetrical and generate the required logic in each case.

4- Documentation

Doxygen is configured to automatically generate HTML documentation from the source files and draw architecture schematics.

3. Included files

Brief explanation of each source code file created for this project. Every file has been written from scratch, no third party code or libraries were used.

VHDL code (/rtl)

The implementation is modularized for better extensibility and code reuse.

1- ParallelFIR.vhd
   Top-level module, encloses the modules that conform the Parallel filter architecture (mainly the samples SLR, individual FIR filters and interconnection matrix).

2- FIR.vhd
   Implementation of a generic fixed-point FIR filter with an arbitrary number of taps and symmetrical / non-symmetrical architecture.

3- AddersTree.vhd
   The addition of multiple elements is performed in parallel pipeline stages that keep combinational delay constant increasing latency by log2(n inputs).

4- ParallelFIR_tb.vhd
   Self-verifying test-bench. Feeds reference samples to the VHDL module and compares its output against the samples from the model. The filter architecture is generated from the desired configuration on each run.

Packages (/pkg)

pkg_CoefficientsFi.vhd, pkg_InputSamplesFi.vhd, pkg_OutputSamplesFi.vhd
These packages are generated automatically by the Matlab script and contain the coefficients configuration and the reference samples for the testbench.

pkg_Common.vhd
Generic functions used pre-synthesis such as log2 or the ternary operator.

pkg_ComponentDeclaration.vhd
Contains the declaration of all the components so as to keep them centralized.

Matlab code (/scripts)

The whole flow (model generation, VHDL simulation and verification) is controlled from Matlab.

1- testLowPassFilter.m

The provided code is completed to include the fixed-point model generation, HDL Simulation run, verification and result display.

2- filterFPGA.m

Fixed point model of the filter.

3- create_package.m

Generate packages containing the reference samples for the VHDL testbench to use them in verification.

4- runSimulationVHDL.m

Control the Python script that handles the GHDL simulation flow.

5- readSimResults.m

Read the VHDL testbench results output file.

GHDL Simulation (/sim)

For performance in an all-VHDL project, GHDL compiler-simulator is used along GTKWave.

1- run_ghdl_sim.py

Handles the GHDL compilation and simulation run process.

Usage is: run_ghdl_sim [simulation time]

Finds and compiles the dependencies, packages and source files and runs the simulation for the specified amount of time. If no simulation or verification errors are found, the simulation will finish by “stop-time”. In case of verification error, the simulation will be stopped and the error will be reported.

/waves

The waveform files in this folder will be automatically displayed and populated upon simulation.

Documentation (/docs)

Doxyfile

Project-specific Doxygen configuration file. Run Doxywizard pointing to this file to update the RTL documentation.

/html

To navigate the generated HTML documentation, open /html/index.html .

4. Possible further improvements

1- Use constant multipliers

If coefficients are not expected to change dynamically, multiplier usage can be reduced by implementing multiplications in a constant multiplier submodule using shift and add operations.

2- Use DSP resources

Should we want to increase performance by tying the implementation to a specific FPGA family, the arithmetic operations can be implemented in a specific submodule that maps to the DSP primitives.

3- Handle binary point at different positions

With some additional work, data with the binary point located at different positions can be handled, the resulting binary point position would be calculated and propagated through the stages.

Work log

The development of this Parallel FIR Filter and verification procedures has been tracked using Version Control. Full project and change history can be acessed.