ELEC 4511/5511

Lab 6

Customize an FIR filter with AXI-Stream Interface

In this lab, you will design an "Finite Impulse Response (FIR) filter" accelerator with AXI-Stream interface using Verilog, and verify your design on the PYNQ board.

1. Experiment goal:

- Familiar with the process of using Vivado IDE to customize hardware IP with AXI-Stream interface.
- Understand the role of Direct Memory Access (DMA) in the system
- Review using testbench for simulation in Vivado
- Understand how to accelerate software programs by designing hardware modules

2. Introduction:

In digital signal processing, an FIR is a filter whose impulse response is of finite period, as a result of it settles to zero in finite time. FIR filters are most popular kind of filters that can be continuous time, analog or digital and discrete time. For an FIR design, we have a couple of options such as the windowing method, the frequency sampling method, and more. In this lab, we will choose the easiest one, that is, "Direct-Form Structure".

The direct-form structure is directly obtained from the difference equation. Assume that the difference equation of the FIR filter is given by

$$y(n) = \sum_{k=0}^{M-1} b_k x(n-k)$$

Therefore, for an *N-tap* FIR filter with h(k) coefficient, the output is defined as:

$$y(n)=h(0)x(n) + h(1)x(n-1) + h(2)x(n-2) +...+ h(N-1)x(n-(N-1))$$

The Z-transform of the filter is:

$$H(z)=h(0)z^{0} + h(1)z^{-1} + h(2)z^{-2} +...+ h(N-1)\ z^{-(N-1)}$$

3. The coefficients of the FIR filter and the input data for test:

In this Lab, the noisy input data sequence x(n) is generated by the python script below (20,000 sample data are generated). The frequency of the dominant sinusoidal signal is 0.2MHz, and another two noise signals with frequencies of 46MHz and 12MHz are added to it. The goal of this lab is to design an FIR filter to remove the high-frequency noise as much as possible and retain useful low-frequency information.

The coefficients of the desired FIR filter are achieved by using an online filter design simulator (http://t-filter.engineerjs.com/). The passband is set from 0 to 5MHz, and the stopband is from 10MHz to 50MHz. Since the coefficients are floating-point decimals, they are not suitable for implementation on hardware. Therefore, we choose 16-bit integers to represent them. Thus, on average, each coefficient has been increased by 32767 times.

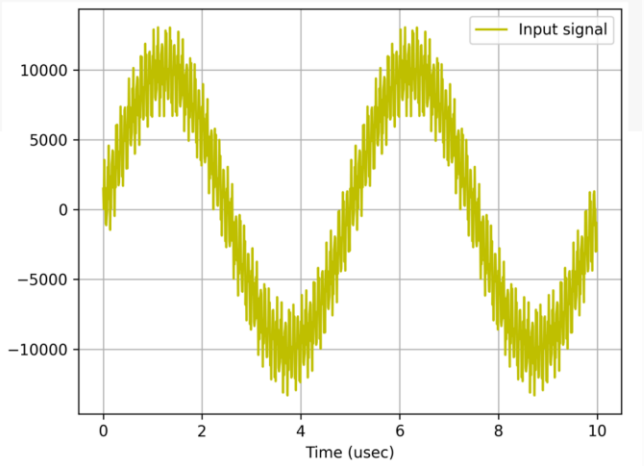
"Double"
-0.007775715121256279, -0.007938974136595618, -0.00953426578824613, -0.008779578259641298, -0.004381884421750875 , 0.004666131585205167, 0.018804473122893708, 0.03764144706001849, 0.05992101812383004, 0.08357444021744635, 0.10601855702701225, 0.12454015906119098, 0.13674393462068657, 0.14100385434561774, 0.13674393462068657, 0.12454015906119098, 0.10601855702701225, 0.08357444021744635, 0.05992101812383004, 0.03764144706001849, 0.018804473122893708, 0.004666131585205167, -0.004381884421750875, -0.008779578259641298, -0.00953426578824613, -0.007938974136595618, -0.007775715121256279

"16-bit interger"
-255,-260,-312,-288,-144,153,616,1233,1963,2739,3474,4081,4481,4620,4481,4081,3474,2739,1963,1233,616,153,-144,-288,-312,-260,-255

```python
import numpy as np

# Total time
T = 0.002
# Sampling frequency
fs = 100e6
# Number of samples
n = int(T * fs)
# Time vector in seconds
t = np.linspace(0, T, n, endpoint=False)
# Samples of the signal
samples = 10000*np.sin(0.2e6*2*np.pi*t) + 1500*np.cos(46e6*2*np.pi*t) + 2000*np.sin(12e6*2*np.pi*t)
# Convert samples to 32-bit integers
samples = samples.astype(np.int32)
print('Number of samples: ',len(samples))

# Plot signal to the notebook
plot_to_notebook(t,samples,1000)
```



4. Software Implementation - Python

First, you will design an FIR filter in Python. Please note that the filter coefficients are known (please use 16-bit integer). Therefore, you can use a multi-layer for loop to output y(n) according to the equation

$$y(n)=h(0)x(n) + h(1)x(n-1) + h(2)x(n-2) +...+ h(N-1)x(n-(N-1))$$

And use a list to store all the output.

You can write and verify this program on your computer first, and then copy the code to the PYNQ board.

5. Hardware Implementation – Verilog

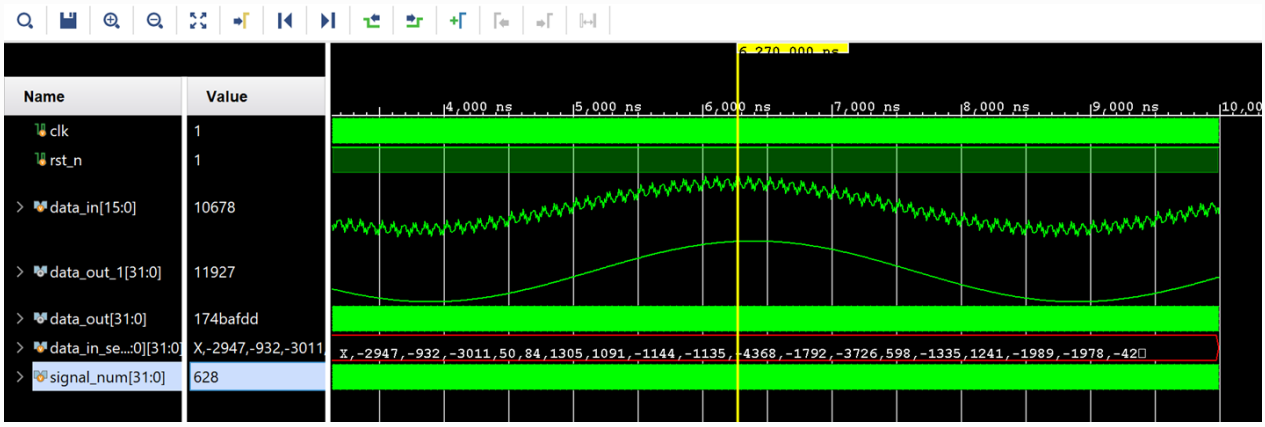
Please use Verilog to write an FIR_filter module. Please use the template shown in the figure below to define the ports. The input data "data_in" is 16 bits, and the output data "data_out" is 32 bits. At each rising edge of the clock signal "clk", there is a data sent to the module as input. When "rst_n" is 0, the output signal "data_out" is cleared.

Please do not add/delete or change the order of these four ports. Otherwise, you will not be able to use the simulation files and AXI stream wrapper provided to you.

As for the relevant tips on designing this module, please refer to the lecture.

```
module FIR_filter(
    input clk,
    input rst_n,
    input signed [15:0] data_in,
    output reg signed [31:0] data_out
);
```

After completing the design, please use the simulation program file "sim_FIR_filter.v" and the input data file "hex_signal.txt" provided to you to verify your module. The following is the result of a successful simulation for your reference



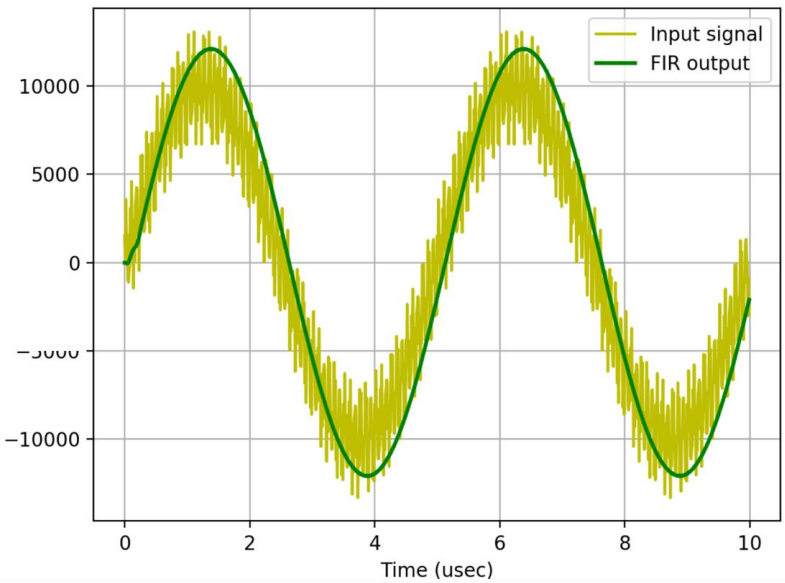
6. Hardware/Software Verification – PYNQ board

Please create a new project and a new AXI-stream IP template. As shown in the lecture, replace the generated template file with the AXI-stream wrapper provided to you, and incorporate your filter design into it.

Then follow the steps shown in the lecture to generate the system bit file and block design file, and upload them to the PYNQ board. Then **modify** and run the jupyter notebook template ("*Lab6 Customize a FIR filter with AXI-stream Interface (student).ipynb*") provided to you to verify your design. You should see similar results as shown in the image below. Please refer to the lecture for specific instructions.

```
Software FIR execution time:
39.314146518707275

Hardware FIR execution time:
0.003692626953125
```



Canvas submission: Please submit your report in ".**pdf**" file format and compress your Vivado project and your Jupyter notebook file into a "**.zip**" file.

What needs to be included in your report:

1. Copy your Python code in the report
2. Copy your Verilog code in the report
3. Screenshot of your simulation result
4. Screenshot of your verification result in Jupyter Notebook
5. Please write a short analysis of each problem, mainly explaining how you think about the design of the function, what troubles you encountered during the design process, and how you finally solved them, etc.

What needs to be included in your "**.zip**" file:

The folders of your vivado project and IP_repo. And the Jupyter notebook file