Author: Edgar Alejandro Recancoj
Date: 06/14/2023

General Processing system architecture:

The Audio Characterization System uses Python on top of Linux to run various functionalities in the embedded system. Audio recording, certain parts of audio processing, and audio output are done from Python.

The main workflow is as follows:

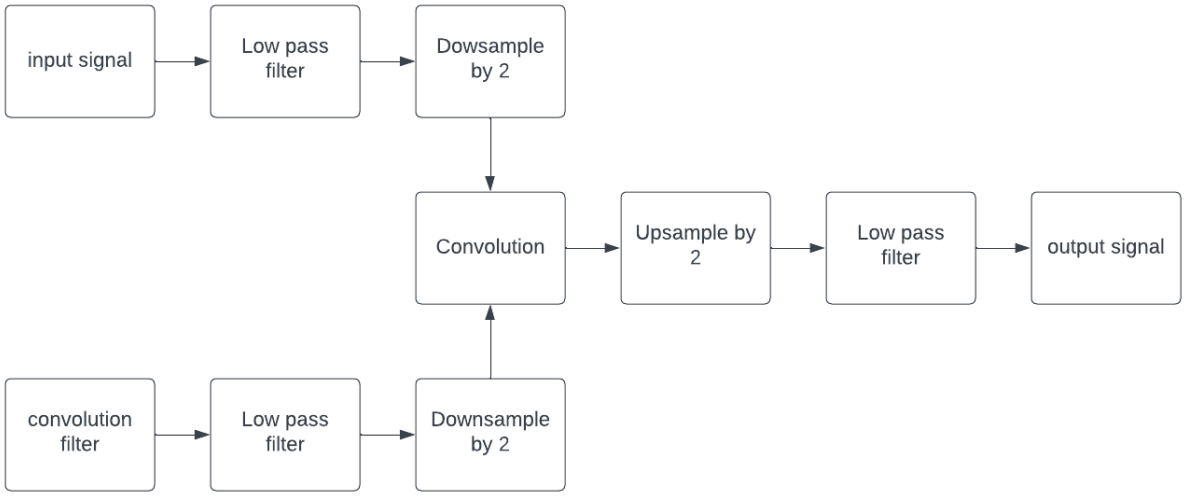

Figure 1: Main workflow of system

The input signal is a wav file that represents the audio that we want to process. The convolution filter is the impulse response of the environment to recreate. To show the functionality of the interconnection between the Processing System and the FPGA fabric, the low pass filter functionality is done completely in the FPGA. Using filtering before down sampling and after up sampling is required to avoid aliasing effects.

Configuration:

Audio input files:

The system allows to use pre recorded audio files or record new ones. Importing the base overlay with “base = BaseOverlay(base.bit)” is necessary to add the recording functionality of the ADAU1761 Codec. When configuring the audio driver, its important to specify the sample rate of the microphone (which is set to 44,100 Hz) and select the input of the microphone as “line in”.

NOTE: All wav files that are recorded locally on the embedded system are 24-bit wav files. To open them correctly, soundfile is used as a dependency NOT included in the default PYNQ Python environment. To install soundfile, run “sudo pip3 install soundfile” when the PYNQ board is connected to the internet. In the audio_characterization_main.ipynb script, choosing option=”file” allows to use pre recorded audio files, while choosing option=”line” allows to record new files to be used in the system. For this version of the project, all audio data is managed in Mono.

Audio processing:

To leverage the functionality of the FPGA, its necessary to import the Overlay. The path of the overlay should match with the bitstream file that was created from Vivado (refer to *FPGA_Documentation* about this issue). Some audio processing is done on Python, such as up sampling, down sampling, and normalization of audio. Low Pass Filtering is done on the FPGA using dedicated DSP slices.

Processing system and FPGA interconnection:

For this we use the PYNQ abstraction layer to communicate with the FPGA. The workflow is as follows:

1. A DMA python object is created. This object is used to represent the DMA and start and stop memory transactions from the DRAM into the FPGA fabric.
2. Memory is allocated to DRAM. This memory allocations are filled with the audio data to filter.
3. A DMA transaction is initialized with “dma.sendchannel.transfer(input_buffer)”. This command signals the DMA to transfer the memory into the DMA FIFO. From this stage onwards refer to *FPGA_Documentation* to see what happens to the data through the FPGA DSP pipeline.
4. When the data is filtered, the DMA raises a flag that is read on “dma.recvchannel.transfer(output_buffer)”. This means that the processing is done, and the result was placed in the output buffer located in DRAM.

NOTE: The DMA needs AXI4 Stream compliant interfaces to function correctly. The DMA in the PYNQ board requires the optional bit TLAST to know when the transaction if finalized. Failure of implementing this feature will result in the DMA never finalizing a transaction and erroneous behaviour of the system.

Audio output:

Soundfile is used to change bit depth between wav files and “wavfile” library from SciPy to write the result as a wav file into the SD card. The result can be played later or directly through Jupyter Notebook.

Q&A:

How to run the example script (audio_characterization_main.ipynb)?

- Have Jupyter notebook installed on your PC.
- Use a PYNQ image of V3 or above.
- Install the package soundfile into the Pynq system by running sudo pip3 install soundfile on the bash shell.
- Move the contents of PYNQ_overlay_files into \pynq\xilinx\pynq\overlays directory on the embedded system. The notebook uses the following path for the bitstream file: /home/xilinx/pynq/overlays/LPF/low_pass_filter.bit. The rest of the files from PYNQ_overlay_files are on the same directory.

How to verify that the Low Pass Filter works on the FPGA?

Refer to the “LPF_FPGA_Verification.ipynb” file that showcases its correct functionality. A sinusoid signal was created that had high frequency noise added to it.

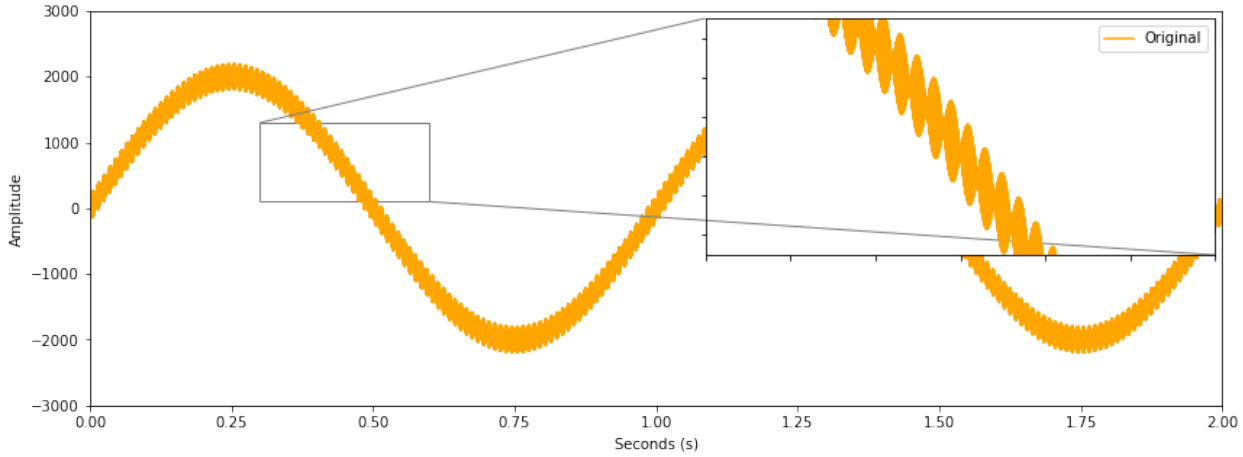

Figure 2: Input signal with high frequency noise.

A low pass filter was implemented on software using Python. This method used 35 filter coefficients and convolution using numpy to filter the signal.

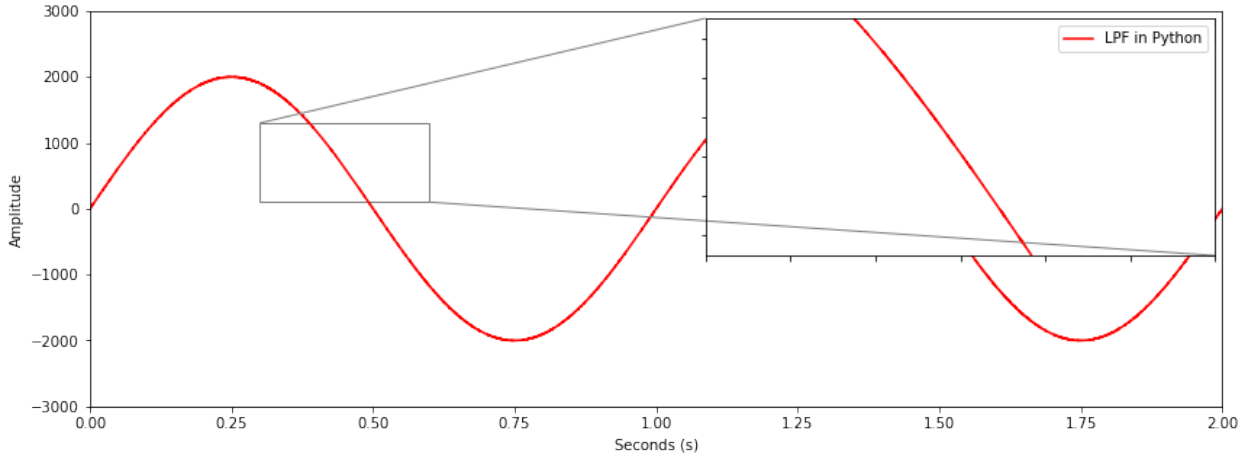

Figure 3: Low Pass Filter using Python

The same test signal was passed through the low pass filter in the FPGA. The result showcases it eliminates the high frequency data similarly as the software version did.

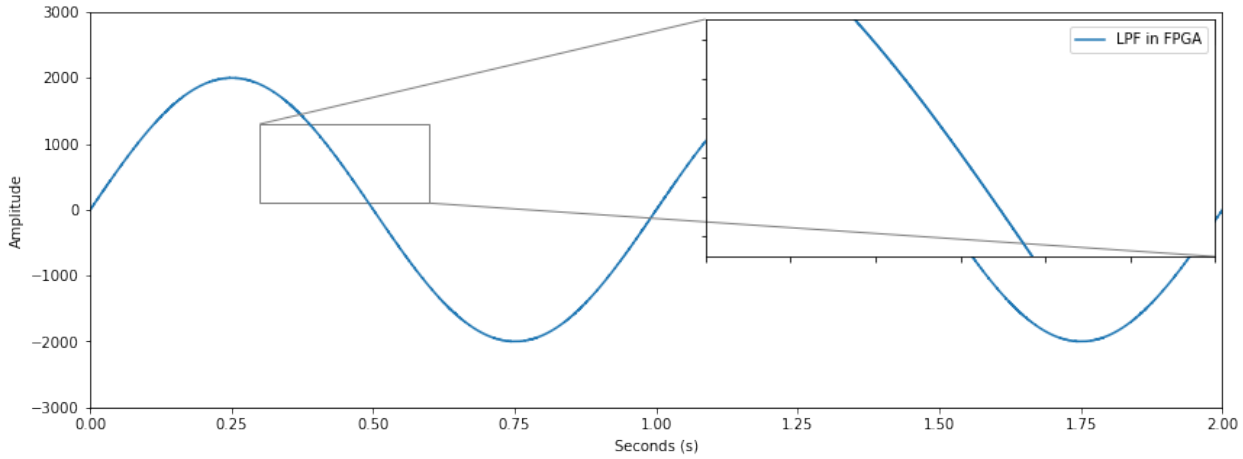

Figure 4: Low Pass Filter using FPGA

Appendix:

- After connecting to the internet for the first time through a direct connection to the router, later re connection to the PC with an ethernet cable does not work with the current PYNQ image. The PYNQ board is not accessible through ethernet connected to the PC, but it can be accessed again when placed directly connected to a router. Solution to this bug is unknown.
- The initial loading of the bitstream into the FPGA takes some seconds. Consequent loads are done much faster.