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

General FPGA system architecture:

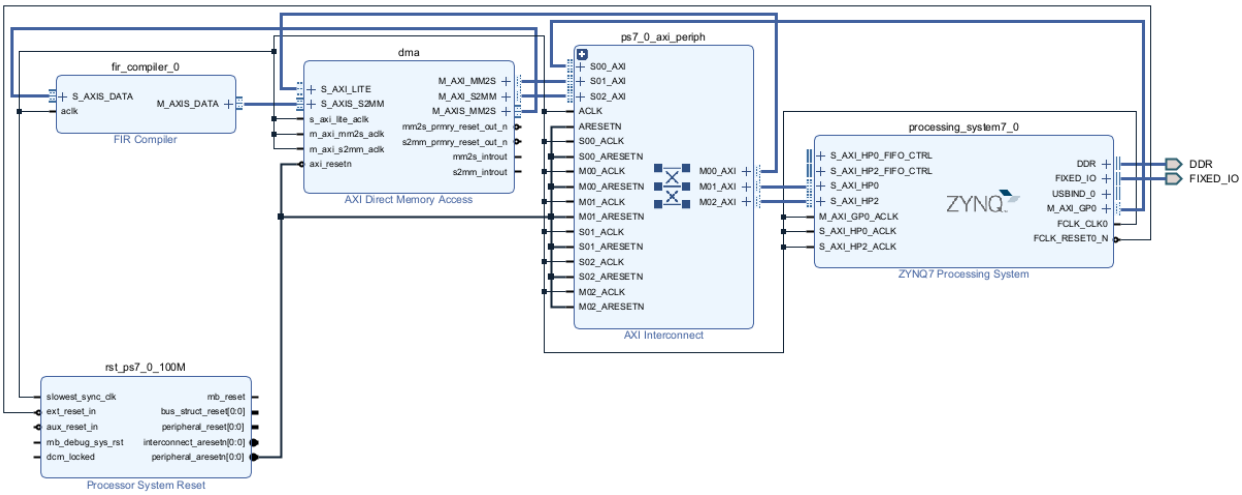

Figure 1: FPGA block design in Vivado

The Audio Characterization System implements a low pass filter (LPF) in the FPGA fabric. On the Processing System (ARM Cortex processor) side, a numpy array represents audio data and it is allocated in the DRAM. When Python signals the start of a transaction, the data passes through the FPGA logic, is filtered, and is placed back into the DRAM to be read.

The system architecture on the FPGA is composed of the following modules:

- Processing System: This module represents the processor of the Zynq SoC.
- Direct memory access (DMA): This module is signaled by Python to start a transaction and controls the flow of information between the DRAM and the rest of the FPGA fabric.
- Finite impulse response (FIR): This module implements a low pass filter with a cutoff frequency of the sample rate divided by 4 used during the decimation and upsampling of data.
- AXI_peripheral module: This module is a bridge between AXI interfaces of the DMA and processing system.

All the modules are connected through AXI4 interfaces. AXI, which means Advanced eXtensible Interface, is an interface protocol defined by ARM as par of the AMBA (Advanced Microcontroller Bus Architecture) standard.

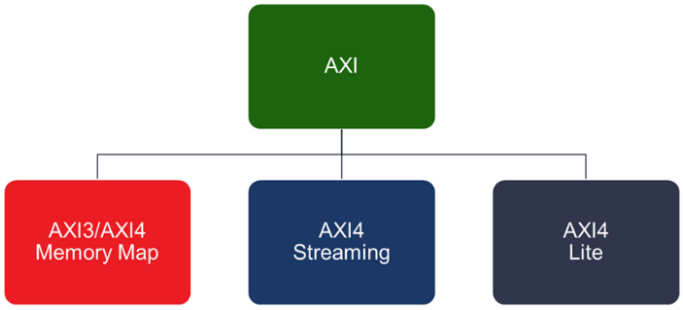

Figure 2: AXI interface types. Obtained from
https://support.xilinx.com/s/article/1053914?language=en_US

AXI4 Memory mapped interfaces were used for the interconnection of the DMA and the processing system. These interfaces require of memory addressing to obtain data from the DRAM. High performance ports were used to avoid any kind of bottleneck on the Processing System. The connection between the DMA and the FIR module were AXi4 Stream interfaces. These interfaces are characterized by high throughput of data and do not require memory addressing, because all the data is moved sequentially through the bus. When the DMA starts a transaction, it moves data from the DRAM into an internal FIFO inside the DMA. When the FIFO is full, the DMA sends the data sequentially through AXI4 Stream data interfaces to the FIR module to be processed. The FIR module uses a special bit to signal the DMA that the processing is finished, and the DMA raises a flag that is read by Python on the Processing System. This flag signals that the DMA transaction is complete, and the processed data can be read from the DRAM.

Configuration:

Processing System module configuration:

On the Processing System module, its important to enable at least two S_AXI_HPX interfaces to move data between the DMA and the DRAM. For this project, S_AXI_HP0 and S_AXI_HP2 were enabled.

NOTE: The data width of these interfaces should be set to 64 bits. This value is hardcoded on the default PYNQ image and thus misconfiguration of the interface can lead to unexpected results.

DMA module configuration:

For the DMA module configuration, “Scatter Gather Engine” and “Micro DMA” are NOT supported by current PYNQ configuration, thus must not be enabled. As a design choice, the DMA was made to handle the biggest possible amount of data with a single

transaction. This is controlled by the “Width of Buffer Length Register” and it was set to 26 bits. This means that the maximum amount of data that the DMA can hold at a given time is 2^26 = 67108864 bits = 6.71 MBits.

The “Memory Data Width” was set to 64 bits for a correct connection between AXI memory mapped interfaces and the High-Performance interfaces in the Processing System that are set to 64 bits as well. The actual audio data in the DRAM is set as 32-bit integers, and thus the “Stream Data Width” was set to match that value. Specifying any amount lower than 32 bits could produce loss of information and incorrect results for the FIR module.

FIR module:

The FIR module implements a low pass filter with an ideal cutoff frequency of 11,025 Hz (microphone sample rate divided by 4). The coefficient vectors were calculated using python. 35 coefficients were used for the filter. Here is the frequency response of the filter:

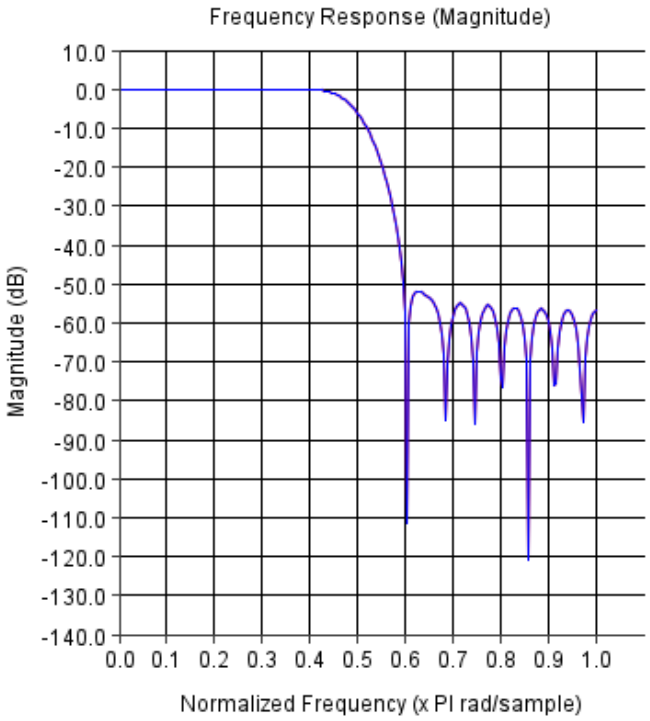

Figure 3: Frequency response of LPF implemented in the FPGA

DSP slices were used to implement the filter and provide a rapid processing of the information. The module is also AXI4 Stream compliant, and the TLAST and TREADY control signals had to be implemented for the DMA to know when the processing of information was complete.

AXI peripheral module and reset system:

These two modules are handled automatically by Vivado.

Q&A:

How to open the project in Vivado?

- Open Vivado and on the Tcl Console:
  - Git clone the repository.
  - Change directory where audio_characterization_system/Vivado/ is located.
  - Run in the console: source generateSystemProject.tcl.

The Tcl script was provided to quickly recreate the project on your local machine.

How to modify the project in Vivado?

Whenever a new change is made to the FPGA system architecture and it needs to be tested, it is necessary to generate the bitstream.

1. While on Vivado, and with the Block Design window opened, right click “design_1_i” (under Design Sources and design_1_wrapper) and click on “Create HDL Wrapper”. This will create a wrapper for the project.
2. Click on Generate Bitstream on the left column. This will start compiling the code, synthesizing, and implementing stages to create the bitstream. This process can take a few minutes.

After modifying the FPGA system, how do you upload the changes into the Zynq SoC?

It can be tricky to find where Vivado places all the files required to configure the FPGA. The PYNQ system requires the following files:

- *.bit file.
- *.hwh file.
- *.tcl file.

If you run the tcl script provided to recreate the project, the project default name is “single_DMA”. Thus,

- The *.bit file can be found in single_DMA/ single_DMA.runs/impl_1/. Its default name is “design_1_wrapper.bit”.
- The *.hwh file can be found in single_DMA/ single_DMA.gen/sources_1/bd/design_1/hw_handoff/. Its default name is “design_1.hwh”.
- The *.tcl file must be exported from Vivado. While the Vivado project is opened and the block design is shown, go to File->Export->Export Block Design. This will generate the desired tcl file.

NOTE: All three files must have the same name. For the code provided, they were named low_pass_filter.bit, low_pass_filter.hwh, and low_pass_filter.tcl. These names must match with the path that was provided in Python for the overlay.

Now that these three files were generated, they need to be placed on the PYNQ SD Card. To access the SD Card from the File Manager, type on the left upper bar "\\pynq". This will access the PYNQ file system. Go to "/home/xilinx/pynq/overlays/" and create a folder to store the files. For the code provided, this directory is called "LPF". Inside that directory, place the three files. Now the PYNQ system can access the bitstream required to configure the FPGA.

Appendix:

- Vivado might be able to successfully generate a bitstream from the project, but when uploading the Overlay in Python it might throw an exception. In most cases, this resulted from an error in the Vivado design. There was a bug in which uploading the Overlay in Python threw an exception but the code in Vivado was correct. Resetting the PYNQ system solved the problem.
- In Vivado 2022.2, running a simulation with new IPs or custom Verilog code does not work by default. The default library must be added to each custom module in its properties for the RTL simulation to start.