# Educational Objective

In this lab, you will learn how to…

1. Use fixed point **DSP** resources of the FPGA.
2. Write VHDL code for a **digital filter**.
3. Use **ModelSim** to verify HDL code.
4. Design and code a test-bench, and perform functional **verification** of your code.
5. Add basic **timing constraints** to your Quartus project.
6. Use the **dual port ram** buffer to write a new waveform to the audio waveRAM created in lab 4.
7. Perform a hardware **validation** of your design.

# Technical Objective

There are six main technical objectives in this assignment

1. Experiment with changing the waveform of the audio signal.
2. Measure the CODEC sample rate using digital oscilloscope.
3. Implement a low pass digital filter in VHDL.
4. Verify your filter design using a test-bench in VHDL.
5. Add basic timing constraints in your Quartus project.
6. Validate your filter design in hardware, by listening to and comparing the sound of a sequence of tones before and after filtering.

# Deliverables

* VHDL Code for Digital Filter
* VHDL Test Bench
* All appropriate sign-offs.
* Demo of working code with Digital Filter

# Part 1 – Create Quartus Project With Timing Constraints

In this section you will create a Quartus project for lab 5, which is essentially a clone of the project for lab 4. However, before modifying this, you will add some basic timing constraints. Please read the document ug\_tq\_tutorial, which will explain timing constraints, and how to implement them using TimeQuest, from Quartus. You will need to tell the tools that your clock rate is 50MHz. We will talk more about timing closure and pipelining later in the semester, after the mid-term.

Verify that your code still functions as it did for the lab 4 project.

# Part 2 – Measure the sample rate of the CODEC.

In order to measure the sample rate, you will route the AUD\_DACLRCK signal to an FPGA pin, connected to one of the 40 pin GPIO headers on the DE1. Using the digital oscilloscope you will measure the frequency of AUD\_DACLRCK.

In the reference material for this lab, you will find the data sheet for the audio codec, as well as documentation for the DE1 board. You will need the DE1 documentation to locate a pin on the one of the 40 pin GPIO headers. You should examine the CODEC data sheet to determine what the DACLRCK signal does.

You are basically routing an input pin (from the CODEC) directly to an output pin (connected to one of the 40 pin headers). In order to do so, you should use an intermediate signal, for example:

aud\_daclrck\_sig <= AUD\_DACLRCK;

GPIO\_1(0) <= aud\_daclrck\_sig;

# Part 3 – Experiment With Different Audio Waveshapes

For this part of the lab, you will write a different wave shape to the waveRAM, and listen to the different sound created by the new wave-shape. You can generate the waveform data through any means you choose. You will then need to write c-code to send 256 audio samples to the waveRAM, which you created in the previous lab. This can be done algorithmically, for example generating a ramp (triangular) wave function would be fairly easy. Or, for an arbitrary waveform, you will need 256 lines of c-code, to download the entire thing. You can also generate random numbers to be sent to the waveRAM, and this should produce roughly white noise at your speakers.

Each sample in the waveRAM is 32 bits wide, 16 bits for each of the two channels. The 16 bit channel data is in signed two’s complement format​.

# Part 4 – Write VHDL Implementation of Digital Filter

The low pass filter was designed using Matlab. The filter design is documented in this section, including filter coefficients, block diagram, and the required port map for the digital filter VHDL entity. A large part of your task is to figure out how to implement this filter using fixed point arithmetic in the FPGA. In order to do this, you may need guidance, which will be provided in the lecture portion of the course. Some of the specifications for how this filter should be implemented are…

* The digital filter will be implemented as three second order stages. See Figure 1 through Figure 3.
* Each delay (i.e. z-1), can of course be implemented by a clocked register in VHDL.
* You should also put a clocked register between each filter stage. This introduces a clock cycle of delay between each stage, but of course with clock rate of 50MHz this will have no discernable effect.
* We will use signed 36 bit, fixed point math, with 17 bits of fraction.
* The multiplications will be implemented by creating a multiplier IP, (similar to how we created the RAM for lab 4), and instantiating 5 of these for each filter stage.

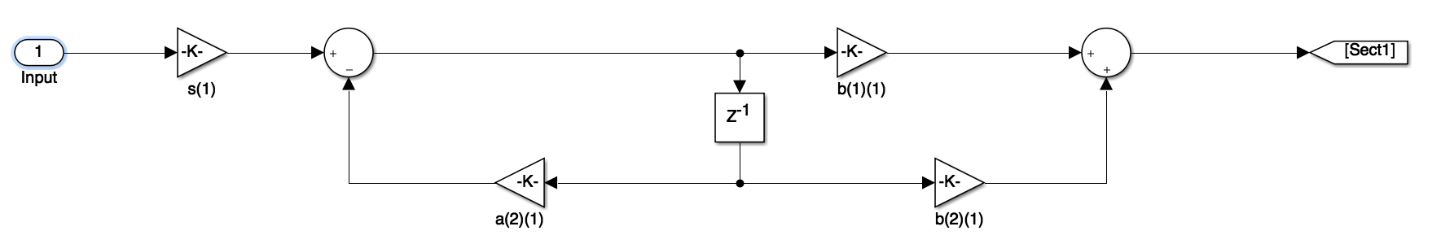


Figure : First Stage

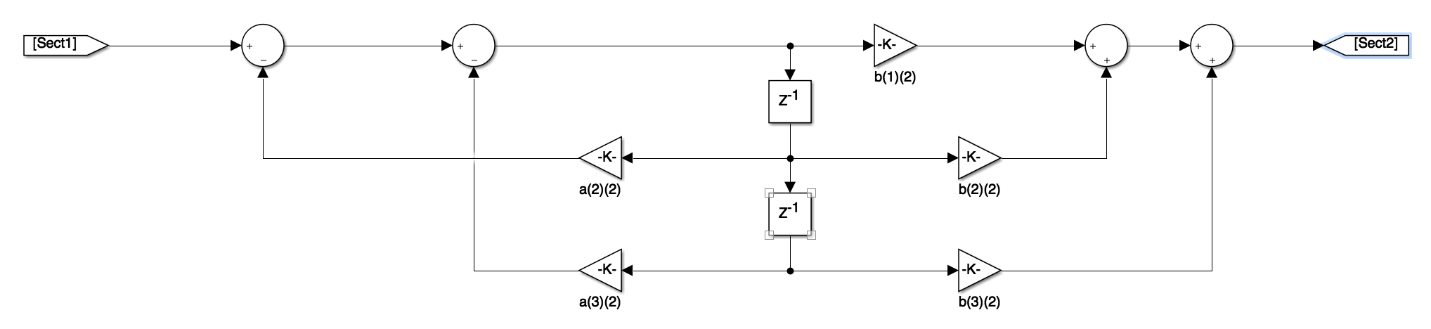


Figure : Second Stage

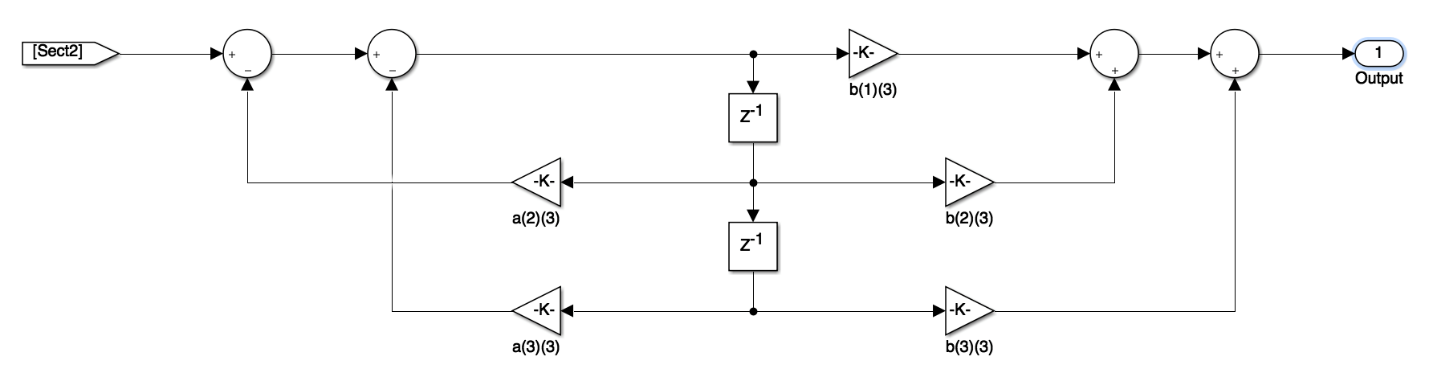


Figure : Third Stage

|  |  |  |
| --- | --- | --- |
| First Stage | b(1)(1) | 0.0033507 |
|  | b(2)(1) | 0.0033507 |
|  | b(3)(1) | 0 |
|  | a(2)(1) | -0.91 |
|  | a(3)(1) | 0 |
| Second Stage | b(1)(2) | 0.0026446 |
|  | b(2)(2) | 0.0052893 |
|  | b(3)(2) | 0.0026446 |
|  | a(2)(2) | -1.9349 |
|  | a(3)(2) | 0.94353 |
| Third Stage | b(1)(3) | 0.25008 |
|  | b(2)(3) | 0.50015 |
|  | b(3)(3) | 0.25008 |
|  | a(2)(3) | -1.8504 |
|  | a(3)(3) | 0.85861 |

Figure : Filter Coefficients

entity audio\_filter is

port (

i\_clk\_50 : in std\_logic;

i\_reset : in std\_logic;

i\_audioSample : in std\_logic\_vector(31 downto 0);

i\_dataReq : in std\_logic;

o\_audioSampleFiltered : out std\_logic\_vector(31 downto 0)

);

end entity;

Figure : VHDL Entity Declaration

# Part 5 – Verify Your Digital Filter VHDL

In this part of the lab, you will write a test bench to verify your design from part 3. You will execute this test bench using ModelSim. I have included two example test-benches, along with some information on doing file I/O, in a VHDL test bench, (see the lab 5 folder on MyCourses). These are only examples demonstrating the techniques, they are NOT intended to be something easily modified to create your test-bench.

In order verify your design, you will read a waveform from a file, process it through your filter, and output the filtered waveform to a file. The file containing the input waveform is included in the lab 5 folder on MyCourses, and is called one\_cycle\_200\_8k.

In order to verify functionality, you will plot your output waveform, and verify that it shows the appropriate response based on the filter transfer function.

# Part 6 – Implement the Digital Filter in Hardware

In this part of the lab, you will add the digital filter to your design, and validate its performance on the actual hardware. In order to do this validation, you will use switch **SW0** to select either the original tone, or the filtered tone to be sent to the speakers or headphones. As you step through the frequencies, using **KEY1** you should be able to hear that the higher frequency tones are attenuated relative to the lower frequency tones.