

Project 5: Equalizer

1. Introduction

The purpose of equalization is to remove channels effects assumed to be of the form of many delayed and attenuated replicas of the original signal (e.g. multipath). These effects are removed by filtering the received signal using an FIR filter with taps calculated to minimize these channel effects. To calculate these taps we need to receive a “known signal” from the channel that will allow us to estimate the correct filter coefficients. This known signal is called the training sequence and is transmitted periodically to update the equalizer taps. In general this is a two-step process: first determining the channel distortions and second finding the best taps to negate these effects. In the direct version of the equalizer we do this in a single step.

The received signal will have the form:

$$y[n] = \sum_{i=0}^L x[i]h[n-i] + v[n]$$

Where $y[n]$ is the received signal, $x[n]$ is the transmitted signal, $h[n]$ is the impulse response of the channel, L is the order of the channel response and $v[n]$ is additive noise.

The goal of our equalizer is to find coefficients $\{f_{n_d}[l]\}_{l=0}^{L_f}$ where n_d is the delay of the equalizer (increasing delay can improve the estimate) and L_f is the order of the filter, such that the effects of $h[n]$ are cancelled to the maximum degree possible.

The transmitted signal has the structure $x[n] = t[n], n = 0, 1, \dots, N$ where $t[n]$ is the known training sequence and $N+1$ is the length of $t[n]$. The rest of the signal $x[n], n = N+1, N+2, \dots$ is the data being transmitted and is therefore unknown. We can write an equation in terms of the desired values of f_{n_d} as follows:

$$t[n] = \sum_{l=0}^{L_f} f_{n_d}[l]y[n+n_d-l]$$

In words: the left side of the equation is the training sequence we transmitted and the right side is the output of filtering the received signal with the equalizer coefficients. We now write this equation in matrix form, for $t[n], n = 0, 1, \dots, N$

$$\begin{bmatrix} t[0] \\ \vdots \\ t[N-1] \end{bmatrix} = \begin{bmatrix} y[n_d] & \dots & y[n_d-L_f] \\ \vdots & \ddots & \vdots \\ y[n_d+N-1] & \dots & y[n_d+N-L_f] \end{bmatrix} \begin{bmatrix} f_{n_d}[0] \\ \vdots \\ f_{n_d}[L_f] \end{bmatrix}$$

Or with matrix notation: $t = Yf$

This would be our ideal output, but we can't guarantee that our filter is capable of reversing the channel effects. The best we can do is solve these equations in the linear least-squares sense to get the closest possible linear answer:

$$\underset{f}{\operatorname{argmin}} \|Yf - t\|^2$$

It is a well-known result in linear algebra that the solution to this linear least-squares problem is:

$$f_{LS} = (Y^T Y)^{-1} Y^T t$$

A computationally efficient way to solve this equation is using the LU decomposition. This allows us to rewrite a matrix in the form $M = LU$ where L is a lower-triangular matrix and U is an upper-triangular matrix.

We form the matrix $A = Y^T Y$ and rewrite our equation:

$$Af = Y^T t$$

Now we perform the LU decomposition:

$$LUf = Y^T t$$

We introduce a new vector b and solve the following equation for b (note that solving this equation is simple because L is triangular so we can solve by back-substitution):

$$Lb = Y^T t$$

We can now use b to rewrite our original equation as:

$$LUf = Lb$$

$$Uf = b$$

We then solve for f (note that once again we are solving a linear equation with a triangular matrix, resulting in an easy solution using back-substitution):

$$Uf = b$$

We now have the best estimate for f to filter our incoming signal. The taps for our FIR filter are set to the values $f[n], n = 0, 1, \dots, L_f$ and the rest of the received signal is equalized by passing it through this filter.

2. Materials

You are given a zip file with two folders named `ChannelEqualizer` and `LU_Doolittle`. `LU_Doolittle` is useful for developing the code for the `Doolittle` function and the `solveDoolittle` function. These are the primary functions that you must create and are used in the Channel Equalizer. You simply need to cut and paste these functions into the `LU_Doolittle.c` file in the `ChannelEqualizer` folder.

The `LU_Doolittle` folder is organized as follows:

- `LU_Doolittle.cpp` → The place where you write your synthesizable code.
- `LU_Doolittle.h` → header file with various definitions that may be useful for developing your code.
- `LU_Doolittle_test.cpp` → test bench

- `script.tcl` and `directive.tcl` → These allow you to easily create a project. To do this, open the Vivado HLS Command Prompt tool (Start-->Xilinx Design Tools-->Vivado HLS->Vivado HLS Command Prompt) and go to the directory where source files and script files reside by using `cd` command. Then type `vivado_hls script.tcl`. This will create a HLS project automatically, so you do not have to add source files, set the top function, select the target device, etc. This will create a folder called `hls` in that directory with the project. It will also synthesize the project.

The `ChannelEqualizer` folder is organized as follows:

- `equalizer.cpp` → This contains all of the code for calculating the taps for the equalizer and feeding those to an FIR filter. It includes several functions written for you that construct various matrices needed to calculate the taps. It calls the functions `Doolittle` and `solveDoolittle` that you will write.
- `equalizer.h` → header file with various definitions that may be useful for developing your code.
- `LU_Doolittle.cpp` → The place where you write synthesizable code for the functions `Doolittle` and `solveDoolittle`.
- `LU_Doolittle.h` → header file with various definitions that may be useful for developing your code.
- `fir.cpp` → The place where you write synthesizable code for the function `fir`. You can use your best architecture from Project 1. You should take into account the size of this filter when deciding the best architecture.
- `fir.h` → header file with various definitions that may be useful for developing your code.
- `testbench.cpp` → test bench that exercises the entire equalizer.
- `script.tcl` and `directive.tcl` → These allow you to easily create a project. To do this, open the Vivado HLS Command Prompt tool (Start-->Xilinx Design Tools-->Vivado HLS->Vivado HLS Command Prompt) and go to the directory where source files and script files reside by using `cd` command. Then type `vivado_hls script.tcl`. This will create a HLS project automatically, so you do not have to add source files, set the top function, select the target device, etc. This will create a folder called `hls` in that directory with the project. It will also synthesize the project.

`project5-equalizer.pdf` → project instructions: this file

3. Project Goal

The goal of this project is to create an equalizer core. You are required to write an LU decomposition function, and another function that uses the result of the LU decomposition to solve for the equalizer filter taps. These taps are then used by an FIR filter which performs the channel equalization. You are also responsible for developing the FIR filter code though you have already done that in Project 1, so you can use one of your architectures from that here. Do note that the number of taps is vastly different, and you should consider the best architecture for the equalizer. This may involve changing your `fir` code. You are responsible for performing design space exploration that trades off between area, performance and accuracy.

The first part of this project is to write a functional code. The second part is to create an optimal architecture with respect to for area, performance and accuracy.

You will create a report describing the different tradeoffs that you made, and how you (code restructuring, pragmas utilized, etc.). For each architecture you should provide its results including the resource utilization (BRAMs, DSP48, LUT, FF), accuracy (as reported by the testbench), and performance in terms of throughput (number of FFT operations/second), latency, clock cycles, clock frequency (which is fixed to 10 ns). Since it is expected that you will do a significant amount of design space exploration around data types, you do not need to include an architecture for every different of data type. However, it should be obvious from your report exactly how you changed the architecture to get the results that you are reporting. You should include an architecture for the most important optimizations and code restructuring.

4. Optimization Hints and Guidelines

- You must always use a clock period of 10 ns.
- A detailed description of all of the functions for this project was omitted on purpose. One of the major goals of this project is to understand how best utilize and integrate with code that is given to you. That being said, feel free to ask questions about the code on the discussion board.
- You will be able to judge the “correctness” of your code based upon the error reported by the testbench. The code should focus on the error of the equalizer. While there is a testbench for LU decomposition related code, do not report these results are not important unless there is something compelling.
- You should report the total results from the equalizer.cpp. If you have something noteworthy to say about the results from the subfunctions, you can talk about them, but I am most interested in the overall results.
- You should experiment with the data types. I expect to see figures in your report that detail how different data types affect the accuracy (and area/throughput). These should be presented in a cogent manner; don’t just plot a bunch of data, but plot important data that makes sense. Hint: if it is hard to explain, then you should rethink about what you are presenting, i.e., organize the data in a different way.
- As it is currently written you will need to change the data types in multiple .h files. You are welcome to change the #include in the .cpp files, e.g., if you wish to only change the data types in one place.
- The key to this project is a good report describing your design space exploration in terms of area, throughput, and precision.
- Your report should describe your code at a high level. Since each of you will likely write different code anyone that reads your report should be able to fully understand what each line of your code does. However, this does not mean that you need to explain each line of code in excruciating detail. It is possible to be succinct and thorough at the same time.
- Comment your code.

5. Submission Procedure

You must submit your code (and only your code, not project files). Your code should have everything in it so that we can synthesize it directly. This means that you should use pragmas in your code, and not use the GUI to insert optimization directives. Each folder should have the necessary files to synthesize and test the code.

The folder should be zipped with a similar file structure as below:

EQUALIZER_lab_YOUR_LAST_NAME.zip

Contents:

- “report.pdf” – A document describing all of your optimizations.
- Folder “architecture1”: the set of files required to synthesize your first architecture
- Folder “architecture2”: the set of files required to synthesize your second architecture
- ...
- Folder “architectureN”: the set of files required to synthesize your Nth architecture

Email this to Janarbek Matai <jmatai@cs.ucsd.edu> with the title “CSE 237C Project 5”.