Proposal: LMS and RLS Adaptive Filtering on Pynq

Zach Neveu

February 12, 2020

Abstract

This project will implement least mean squares (LMS) and recursive least squares (RLS) adaptive filters to perform the task of adaptive noise cancellation for a real-time audio input on a Pynq development board. The goal of this work is to determine performance differences between using the programmable logic (PL) and processing system (PS) for adaptive filtering, and to find a combination of resources and number representation which produces an efficient implementation for large size filters. To evaluate performance, end-to-end latency and throughput criteria are determined for each implementation.

Advice from Informal Proposal

> Good project. What will your sources of input data be? Can you use any of the interfaces on the PYNQ board to stream data in?
>
> For LMS and RLS you need to be careful about data representation. Floating point is expensive to implement in FPGA hardware.

Introduction

Speech communication via telephone, VOIP, and other technologies has become one of the primary ways that people interact. The presence of external noise can have a greatly detrimental effect on the intelligibility and quality of transmitted speech. Unfortunately, it is generally not possible to remove this noise through purely acoustic means, therefore algorithms are needed to reduce the amount of noise present. If the noise is known and stationary (i.e. Gaussian white noise), it can be removed using a simple Wiener filter [1]. If the noise is unknown or not stationary, a different approach is needed. One common strategy used is to place two headphones on a headset, one microphone on the mouthpiece that picks up both the voice and noise, and one microphone far from the mouth which captures primarily noise. In this scenario, the noise picked up in isolation is generally a linear transformation of the noise mixed with the speech at the other microphone, and by learning this linear transformation, the noise can be removed.

Design

Inputs and Outputs

The implemented designs will make use of the audio inputs and outputs of the Pynq development board. The line input provides two channels of input. The left channel will represent a mix of signal and noise, while the right channel will carry only noise. For testing and development, synthetic data will be passed from a host computer to the line input via an aux cable. In practice, two microphones could be connected to this line input for a real-world demonstration. The headphone output includes two channels, so the single channel output of the algorithms will be duplicated across both channels.

Algorithms

LMS and RLS are two adaptive filtering algorithms which can utilize a two microphone technique to remove noise [2]. Both algorithms use a finite impulse response (FIR) filter to estimate the linear system between the two inputs. LMS minimizes a mean squared error (MSE) cost function using gradient descent, and under steady-state conditions converges to the results of a Wiener filter. RLS is based on minimizing an exponentially-weighted squared-error cost function with the idea that in a non-stationary environment more recent errors should be weighted more heavily. Because of this optimization, RLS can converge faster than LMS, but incurs a higher computational cost. One key difference between the two techniques for the context of this work is that using RLS, each estimate relies on previous estimates in a recursive manner. When using fixed-point number systems, quantization errors can accumulate in the recursive formulation causing potential instability [3].

Plan

I plan to divide the work on this project into 4 key stages as shown in 1. The first stage will be to get versions of both algorithms running on the PS in Python. I have done this stage before for these two algorithms, so the key here will be to only use the libraries present on the Pynq board. Stage 2 will be to get working versions of both algorithms on the PL. This stage could be divided into first performing filtering on the PL similar to the FIR lab, then performing weight updates on the PL as well. Stage 3 will be to optimize designs for the PL, including changing data formats, utilizing pipelining and parallelization. This stage can take as much time as is available as there are many different combinations and architectures to try. Finally, stage 4 will involve analyzing the trade-offs between designs, aggregating data, and reporting results.

Table 1: Work Timeline

Stage	Work to Do	Due Date
1	LMS & RLS in Python	Feb. 19
2	Basic PL designs of LMS & RLS	Feb. 26
3	Optimized designs & profiling	Mar. 4
4	Analysis and report	Mar. 11

Evaluation

Both LMS and RLS algorithms produce a scalar error for each sample. By analyzing these errors, the time needed for convergence, as well as the error of the converged algorithm can be determined. Input signals will be synthetic, so identical inputs can be replicated for both hardware and software designs. A correct hardware implementation will have convergence properties and errors very similar to the reference software implementation. In order to benchmark different designs, the criteria used will be input-to-output latency and system throughput. Latency can be measured by Vivado for hardware designs, and can be timed for both software and hardware implementations. Throughput can also be estimated via timing with large input arrays, or can be calculated analytically for hardware designs. A successful design will maximize the amount of throughput and minimize latency while still fitting within the hardware resources of the Pynq development board.

References

[1] N. Wiener, *Extrapolation, Interpolation, and Smoothing of Stationary Time Series.* The MIT Press, 1964.

[2] P. S. R. Diniz, *Adaptive Filtering Algorithms and Practical Implementation*, 2020, oCLC: 1129184597.

[3] T. Adali and S. H. Ardalan, “On steady-state performance of the fixed-point RLS algo-

rithm," *Computers and Electrical Engineering*, vol. 25, no. 1, pp. 1–16, 1999.