Real-Time FM Demodulation

John Andrews
Byron Chan

December 14, 2023

Abstract

In this paper, we present our results from building a real-time FM demodulator. We used Vitis HLS to design and optimize an FM demodulator IP which would be run on a Xilinx PYNQ-Z2, an SoC board that integrates an ARM processor with FPGA programmable logic. We also used Vivado to integrate our demodulator IP core with the PYNQ-Z2 and generate a bitstream. This bitstream would be run on the PYNQ-Z2 board using a Jupyter Notebook. To test the performance of our IP, we measured the time it took for our IP to process a 1-second, 2.4 MHz audio recording, as well as how long a software implementation running on the PYNQ-Z2's ARM core would take to process the same recording. Our IP achieved a speedup factor of 5.2 over the software implementation. It also was fast enough for real-time demodulation, unlike the software algorithm. After successfully demodulating recorded audio in real-time, we used an RTL2832U radio receiver with an antenna to sample local FM radio signals and then demodulated them with our IP in real-time. The demodulator's performance on the live signal was about the same as its performance on the recorded signal, in both the software and hardware implementations.

1 Introduction

Demodulation is a subfield of the radio frequency (RF) domain that focuses on the process of extracting information from a signal-carrying wave [Dem23]. It is done on the radio receiver so that a device can capture and process radio waves into usable information. Wireless communication is essential in transferring data and has many well-known applications today, such as WiFi, Bluetooth, and radio broadcasting. Modern applications of WiFi that focus on 5G have allowed computers to upload and download data at very high speeds. At these high speeds, it becomes more challenging to design better hardware and software that processes large amounts of data in real-time and understands how to use the decoded information.

To implement demodulation, traditional systems primarily used hardware to process data. As technology advanced, the ease of reconfiguring software caused a shift toward using software defined radios (SDRs) and digital signal processors (DSPs) to handle data. The processing speed of a typical desktop computer is relatively fast and is capable of demodulating FM signals in real-time through software. However, in an embedded system, it becomes more difficult to demodulate signals in real-time on embedded processors because they are slower and have less resources. In this paper, we aim to demodulate FM signals in real-time on an embedded system, in particular, the PYNQ-Z2. We plan to utilize its programmable logic to handle the data so that we can achieve real-time FM demodulation.

1.1 Purpose

The primary goal of this project was to implement and run a real-time FM demodulator on the PYNQ-Z2 to process both live and recorded radio signals.

1.2 Problem Solved

The demodulation algorithm uses two linear filters, two downsamples, and a discriminant. We implemented individual HLS functions to perform a linear filter, a downsample, and a discriminant before combining them to form a demodulator.

1.3 Major Contributions

The main contribution this project brings is real-time FM demodulation enabled by the PYNQ-Z2's programmable logic. We also performed optimizations on the demodulation algorithm to increase throughput.

2 Background

No background knowledge outside of the CSE 237C course curriculum is required. This paper assumes readers have an understanding of Vitis HLS and have completed projects up to the Discrete Fourier Transform chapter on pp4fpgas. Topics such as AXIS, FIR, throughput and resource usage should be well-understood.

3 Implementation

3.1 Architecture

There are three primary components of the FM demodulator: downsample, linear filter, and discriminant. In this section, we will apply all three methods from [Kas21] to design our demodulator.

3.1.1 Downsample

The parameters of the downsample function are an input signal and a downsampling factor. If the downsampling factor is N, then the downsample function will return every N^{th} sample of the input. The length of the output signal is equal to the quotient of the length of the input signal and the downsampling factor.

3.1.2 Linear Filter

Given input coefficients $a_0, ..., a_N$ and $b_0, ..., b_M$, input signal $x_0, ..., x_k$, and output signal $y_0, ..., y_{k-1}$, the linear filter is computed as:

$$y_k = \frac{1}{a_0}(\sum_{i=0}^{M} b_i x_{n-i} - \sum_{j=1}^{N} a_j y_{n-j}) \tag{1}$$

We use $x_t = 0$ and $y_t = 0$ if t is negative. While we started by implementing a general-purpose linear filter, we later noticed that all the linear filter operations used in demodulation, $a_0 = 1$ and $a_t = 0$ for all $t \neq 0$. Therefore, this linear filter becomes a finite impulse response:

$$y_k = \sum_{i=0}^{M} b_i x_{n-i} \tag{2}$$

The length of the linear filter's output is equal to that of its input.

3.1.3 Discriminant

Given a complex-valued input signal $x_0, ..., x_k$, the discriminant could be computed as follows:

1. Let $r_0, ..., r_k$ and $s_0, ..., s_k$ be the real and imaginary parts of $x_0, ..., x_k$, respectively.
2. Let $a_0 = 1$ and $(b_0, b_1) = (1, -1)$ be coefficients to be used in linear filter operations.
3. Using linear filters with these coefficients, compute $r'_0, ..., r'_k$ and $s'_0, ..., s'_k$, the derivatives of $r_0, ..., r_k$ and $s_0, ..., s_k$, respectively. This can be calculated as $r'_i = r_i - r_{i-1}$ and $s'_i = s_i - s_{i-1}$.
4. Each entry of the discriminant $d_0, ..., d_k$ is computed as:

$$d_i = \frac{r_i s'_i - r'_i s_i}{r_i^2 + s_i^2} \tag{3}$$

3.1.4 Demodulator

The demodulator consists of the following steps:

1. Determine finite impulse response coefficients. We computed the coefficients with in Python with the scipy.signal.firwin function and stored them as arrays. Two sets of coefficients are to be computed, as the two linear filters in the demodulator have different input sizes and require different coefficients.
2. Perform a linear filter using the first set of FIR coefficients on the input signal.
3. Downsample the filtered signal by a factor of 10.
4. Compute the discriminant of the downsampled signal.
5. Perform a linear filter using the second set of FIR coefficients on the discriminant.
6. To get the output signal, downsample the filtered discriminant by a factor of 5.

3.2 Constraints

3.2.1 Chunking

We initially attempted to design a demodulator function that can process 2.4 million samples at once, but we found that the input would be too big for the PYNQ-Z2 to handle. Therefore, we decided to create a demodulator that processes one smaller chunk of data at a time. To implement a chunked demodulator, we needed to save certain data to be transferred from one iteration of the function to the next. We used static arrays to save this data in between function calls.

3.2.2 AXI DMA and HLS Streams

We initially tried to use memory mapped I/O to send data to the IP core. However, we found that memory mapped I/O used too much resources to feasibly send data. We used two AXI DMAs to stream data to and from the IP core. AXI DMAs only support signed and unsigned integers, and because all our calculations required floating point data types, we used union data types to interact with the AXI interface.

3.3 System Integration

The FM demodulation IP block contains two inputs: the real and imaginary parts of the input audio signal. It has a single output, the demodulated signal (which consists only of real numbers). We used two AXI DMAs to stream data to and from the IP block. Our Vivado block diagram was configured as seen in Figure 1.

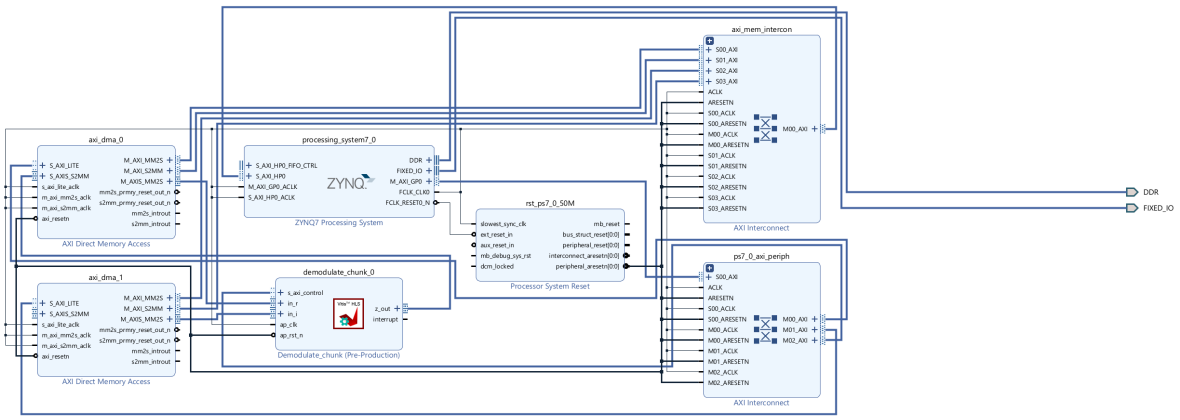

Figure 1: Vivado block diagram.

4 Results

4.1 Deliverables

4.1.1 Demodulation

In this project, we aimed to perform real-time demodulation on the PYNQ board. We timed and tested different demodulation strategies, and measured resource usage and throughput on the hardware implementations.

We tested each design by inputting the same 1 second, 2.4 MHz sample of audio, and timing how long demodulation took. The results can be found in Table 1. We found it interesting that the software approach on the PYNQ-Z2's ARM core demodulated sampled data faster than the baseline demodulation IP block. The best design contains the optimizations found on Figure 2. Our fastest design demodulated the samples in 0.85 seconds, which meant that it was capable of demodulating audio in real-time.

PYNQ Demodulation Method	Demodulation Time (seconds)
Software	4.56
Baseline	5.20
Best	3.16
Best, multithreaded	0.85

Table 1: PYNQ Processing Speed.

We designed three main demodulation IP blocks:

1. Baseline: Initial working demodulator. It used a general-purpose linear filter that was designed by directly implementing the definition of linear filter.

2. FIR: The generalized linear filter was replaced with the FIR.

3. Downsampling: Output entries of the FIR which would be discarded during downsampling were no longer unnecessarily computed.

The two optimizations we implemented in the FM demodulator significantly increased the throughput. Figure 2 demonstrates the increase in throughput from performing these optimizations. Both of our optimizations focused on reducing the number of calculations performed so that throughput would improve. None of our optimizations focused on reducing memory usage, and the differences in resource usage between the three implementations were minimal, as shown in Figure 3.

4.1.2 Baseline

The baseline architecture uses the implementation found in Section 3.1.4. It uses downsampling, linear filtering, and discriminant to build the demodulator.

4.1.3 FIR

While we initially implemented the a general linear filter as described in Section 3.1.2, we realized the all the linear filter operations in the demodulator were just finite impulse responses. Using Equation 2 enabled us to eliminate many unnecessary steps.

4.1.4 Downsampling

Upon our realization that we could implement the linear filters as FIRs, we also noticed that we would only be keeping every N^{th} value after downsampling by a factor of N. Unlike in a general linear filter, in the FIR case, future outputs do not depend on previous outputs. Therefore, we only needed to compute every N^{th} value of the output of the FIR. This reduced the number of FIR computations by a factor of N. In the demodulation algorithm, the output of the first FIR was downsampled by a factor of 10. Consequently, the throughput increased by nearly a factor of 10 once this optimization was introduced, as shown in Figure 2.

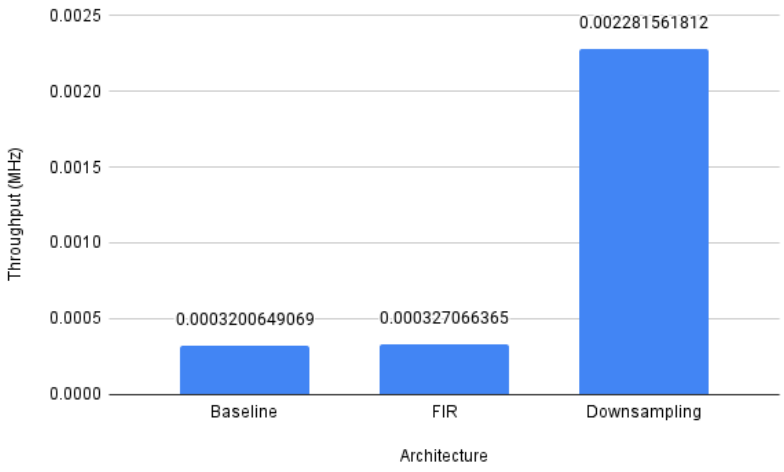

Figure 2: Performance Throughput.

4.1.5 PYNQ Demo

This short video demonstrates our functional FM demodulator working on both recorded samples of data and live data received from the RTL2832U. It shows the time taken for software, single-threaded hardware, and multithreaded hardware implementations.

5 Challenges and Future Work

5.1 Challenges

Many challenges we faced were outside of Vitis HLS. Some of the challenges we faced included:

- Understanding how to interface AXI DMA was initially a challenge. Previous labs guided us through memory mapped I/O and DMA. Referencing the labs and experimenting with them on Jupyter notebook gave us a better understanding on how to use these protocols.

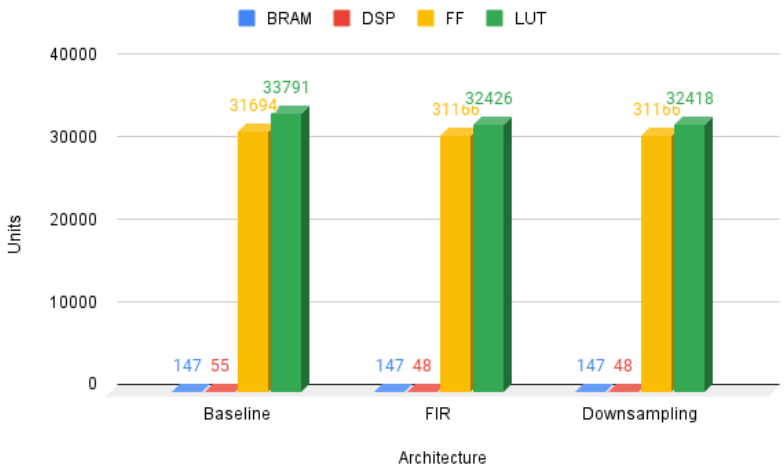

Figure 3: Resource Usage.

- The PYNQ board we initially used had a malfunctioned USB port. We spent many hours trying to figure out why the RTL2832 would not interface with the PYNQ board. We thought it was the software side, but we figured out the USB port was not working by testing it with a multimeter and further confirming it with a USB power meter.
- Connecting the PYNQ board to WiFi to install necessary libraries was not as straightforward as we hoped. We had to modify the network settings, and afterwards, reference pyrtlsdr's GitHub issues [rtl23] to understand import errors and find out how to download missing libraries that pyrtlsdr did not properly install.

5.2 Future Work

The next steps could potentially include resource and performance optimizations, live radio, and demodulating other frequencies.

5.2.1 Resource and performance optimizations

We have found several optimizations on the demodulator, notably FIR and downsampling. However, we are aware that there are more optimizations that we have not implemented, such as optimizing resource usage by reducing array sizes for downsampling. Other optimizations could also include combining linear filtering and downsampling, or performing optimizations based on memory access patterns.

5.2.2 Live radio

Our current Jupyter notebook implementation samples one second of live radio data and then outputs the demodulated audio. Extending this functionality by handling continuous data streaming would allow us to listen to live radio. Implementing audio output through the speaker on the PYNQ-Z2 allow us to listen to the radio directly from the board instead of having to connect the board to a computer.

5.2.3 Other frequencies

Experimenting and demodulating other radio frequency ranges such as AM radio or HAM radio is another potential avenue to explore. This would allow us to better understand how radio communication works. It would also allow us to expand the current capabilities of our current demodulator so that we can listen to other radio bands.

6 Conclusion

We successfully implemented a FM demodulation IP core that is capable of real-time demodulation on the PYNQ-Z2. By designing a more optimized demodulator IP core that considered downsampling, finite impulse response, and the discriminant, we were able to demodulate 1 second of audio sampled at 2.4 MHz in 0.85 seconds. In comparison, the ARM core implementation demodulated the same sample in 4.56 seconds. Additionally, we used an RTL2832U software-defined radio to receive live RF input samples, allowing us to demodulate the data using our IP block so that we can directly listen to the radio from our PYNQ-Z2 board.

References

[Dem23] Demodulation. Demodulation — Wikipedia, the free encyclopedia, 2023. [Online; accessed 11-December-2023].

[Kas21] Ryan Kastner. Project: FM Demodulator. https://github.com/KastnerRG/Read_the_docs/blob/master/docs/project7.rst, 2021.

[rtl23] pyrtlsdr. https://github.com/pyrtlsdr/pyrtlsdr/issues/7, 2023.