CE387 – Final Lab Report

Anthony Bartolomei and Josh Linero

# FM Radio Background and Theory

Frequency Modulation (FM) has played a groundbreaking role not only in audio broadcasting but also in fields such as telecommunications, radar systems, and medical imaging. The fundamental principle of FM—encoding information in frequency variations rather than amplitude—has made it an essential technology in modern communication. From enabling clearer radio transmissions to its application in two-way radios, satellite communications, and even MRI scanning in medical diagnostics, FM has profoundly influenced the way information is transmitted and received. The development of FM technology has revolutionized audio broadcasting, minimizing the noise and interference that plagued earlier amplitude-based systems, and laid the groundwork for many other communication technologies that shape our world today.

The origins of frequency modulation can be traced back to the early 20th century, with American electrical engineer Edwin Howard Armstrong making significant contributions to its development. In the 1920s and 1930s, Armstrong sought to improve radio transmission by reducing the static and interference issues inherent in Amplitude Modulation (AM). His transformative work and research led to the successful demonstration of FM radio, where he showcased its superior sound quality. Despite initial resistance from established AM broadcasters and regulatory hurdles, FM radio gained traction, and had widespread adoption in the 1940s.

The principles behind FM include how it operates within the Very High Frequency (VHF) range of the electromagnetic spectrum, specifically between 88 to 108 MHz for commercial radio broadcasting. In FM transmission, an audio signal modulates a high-frequency carrier wave by altering its frequency in direct proportion to the signal’s amplitude. This method allows FM to maintain a constant amplitude, reducing susceptibility to static and signal degradation.

The amount of frequency deviation in FM signals is controlled to ensure clarity and consistency. This deviation is directly linked to the amplitude of the audio signal; a higher amplitude corresponds to a greater frequency shift. This approach enables the transmission of stereo audio and additional subcarrier signals for supplementary data services, such as Radio Data System (RDS) functionality.

FM broadcasting relies on engineered transmission and reception processes. At a broadcasting station, the audio wave signal is encoded onto a carrier frequency. This is then amplified and transmitted via antennas. At the receiver (think FM radio) it captures the incoming signal through its antenna and processes it through several key stages. First, tuning and filtering take place, the receiver selects the desired station using a tuner finding the starting point. This is aided by a phase-locked loop (PLL) circuit for precise frequency control. A bandpass filter then isolates the intended signal while eliminating adjacent channel noise. Next demodulation takes place. The core function of an FM receiver is to extract the original audio signal from the modulated carrier wave. This is achieved using demodulators such as discriminators or PLL-based circuits, which convert frequency variations into corresponding voltage fluctuations, representing the audio waveform. Lastly, hearing the sound audio processing and output is needed. After demod, the signal undergoes low-pass filtering to remove high-frequency noise. The processed signal is then amplified and sent to speakers, converting electrical impulses into sound waves for human perception.

Beyond traditional radio broadcasting, FM has been instrumental in advancing numerous technological domains. Its noise-resistant properties make it ideal for two-way communication systems, including aviation, maritime, and military radio networks. In radar systems, FM is used for precise distance measurement and object detection. Moreover, Frequency Modulated Continuous Wave (FMCW) radar technology plays a vital role in automotive applications, such as adaptive cruise control and collision avoidance systems.

The invention and evolution of Frequency Modulation have greatly shaped the world, enhancing the quality and reliability of communication systems. FM's impact extends far beyond radio broadcasting, permeating fields such as defense, healthcare, and transportation. Its development by Edwin Armstrong marked a pivotal moment in technological history, proving that innovation in signal processing could revolutionize how we transmit and receive information. Today, FM remains a cornerstone of wireless communication, continuing to adapt and integrate into emerging technologies that define the modern age.

# System Architecture

Our system architecture is based on the top-level design presented in class and elaborated on by the C++ source code. Our design includes the use of FIFOs to separate the main components and ensure proper streaming architecture. See below for a top-level architecture comparison.

A diagram of a multi-tasking process

AI-generated content may be incorrect.

In-Class Top-Level Architecture

A computer screen shot of a computer program

AI-generated content may be incorrect.

Our Top-Level Architecture

Our top-level design contains a read\_IQ, FIR, FIR\_complex, IIR, demod, add, sub, mult, and gain modules. To ensure greater control over critical path length, we implemented all of these modules with finite state machines (see page 10 for all FSMs). A short summary of each module’s architecture is provided below.

## Read\_IQ

The read\_IQ module has a sequential FSM that read’s each component of the real and imaginary inputs of the FM receiver.

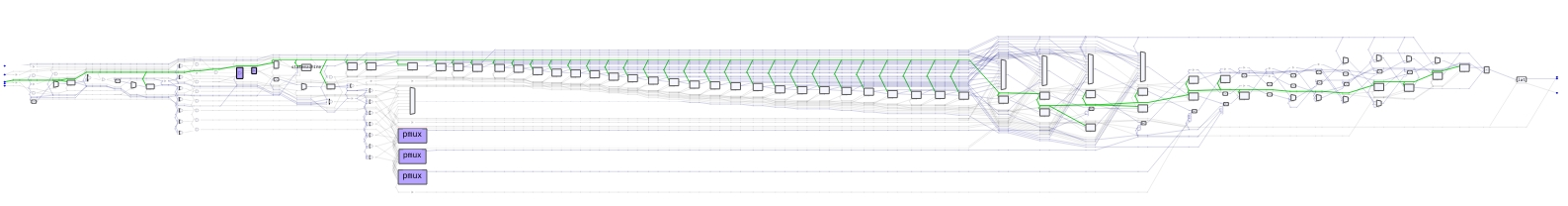
A diagram of a network

AI-generated content may be incorrect. A diagram of a diagram

AI-generated content may be incorrect.

## FIR

The FIR module contains a large shift register (shown below) to store the previous inputs. Our initial design created a long critical path due to the multiple rounds of multiplication necessary for the compute stage. We modified the design to loop unroll such that the multiplication is distributed. This vastly increased the hardware usage but massively improved our clock speed.



## FIR\_Complex

Our FIR\_complex module follows a similar design to our loop unrolled FIR. As shown in the diagrams, the main multiplication calculations are unrolled and parallelized.

A white and green lines and dots

AI-generated content may be incorrect.

## IIR

The IIR module is again implemented similar to the FIR, but with an added shift register to hold previous outputs. The IIR module is not loop unrolled, but the calculation is divided into multiple stages for reasons discussed in the design section.

A close-up of a diagram

AI-generated content may be incorrect.

## Demod

The demod module is fairly unique, as it implements a dot product and arctan calculation, including division. As such, we created a qarctan module and a division module which are instantiated int eh demod module. These modules are discussed in the design section.

A diagram of a map

AI-generated content may be incorrect.

A diagram of a diagram

AI-generated content may be incorrect.

## Add, Sub, Mult, and Gain

Each of these modules implements a trivial addition, subtraction or multiplication of two inputs respectively. They all operate with a similar architecture. Mult is pictured below.

A diagram of a computer

AI-generated content may be incorrect.

# Design Process

The design process for this project was fairly straightforward in the typical hardware design process. We began by creating empty modules for each of the components we would need. From there, we designed interfaces for each one. While working on the implementations of each module, we also created basic test benches to verify functionality before adding them to the top level.

Once each module had been created and tested, we designed our top-level file, connecting each instance with FIFOs. We then began work on the UVM testing suite for the full design. Once completed, we simulated our design with sample data extracted from the original .dat file. After ensuring bit-true correctness, we synthesized our design and analyzed its architecture.

While this process was straightforward and fairly standard, we encountered many issues along the way. We specifically had trouble with our IIR module, as it was very difficult to manually verify our algorithm. Much of the data going through the FM Radio system was in our fixed-point format, which made by-hand calculations nearly impossible.

We demodulate also proved to be fairly difficult to implement. The algorithm the C++ source code followed was fairly straightforward, but we had to port over a qarctan generator from a previous assignment as well as a divider module from last quarter. The divider module was originally written in VHDL, so we had to re-write it in order to use it.

# Optimizations

Once we had verified our design and synthesized it, we found quite a few performance bottlenecks. Our original designs had little to no optimizations built in, as we initially focused on correctness. This led to a maximum clock speed of only about 57MHz. Synthesis results from this initial attempt showed the critical path details as follows:

A screenshot of a computer program

AI-generated content may be incorrect.

These performance issues stemmed from our complex, 32 tap FIR and FIR\_complex instances. To help fix this, we opted to loop-unroll our designs. The original designs are left in our source code commented out to show the evolution of the design.

With our new loop unrolled version, we are able to adjust the performance-hardware usage tradeoff through a module parameter called UNROLL. With a smaller board, you are able to implement our design so long as you do not unroll very much. With the larger board we are synthesizing for, we are able to unroll our FIR and FIR\_complex instances to 4 levels. This drastically improved our clock speed up to 80MHz.

From there, we found that the next bottleneck came from the IIR module. This bottleneck was handled differently, as the IIR filter has only 2 taps. Instead of unrolling these taps, we opted to increase the complexity of our finite state machine, adding extra cycles to the compute time. We found that with the original design, our IIR filter took about 3000 cycles to process 1000 inputs, and with our new design, it took 7000 cycles. While this seems like a massive time loss, we made this change, as we noticed that our IIR was left inactive for more cycles than the new implementation needed, thus leaving our total FM Radio cycle count exactly the same, but with a faster clock of about 87.5MHz.

The next bottleneck came from the divider module in the qarctan calculator of the demodulator. We found that our divider module was operating in O(N) time, so we modified our design to reimplement the O(log(N)) division algorithm.

The final optimization we found came in the gain module. This module is extremely simple, as it multiplies the input by a constant value. The calculation is a bit more complex due to bit-shifting to account for quantization and a base volume multiplier. We found two possible optimizations for our design. We could either split the gain calculation into two states, much like we did for IIR, or we could turn the volume input into a parameter.

Because the volume input is set to a constant 1 in the C++ source code, the synthesizer could automatically get rid of the multiply in the gain module. This brought our clock speed to a whopping 101.4MHz. This optimization however is unrealistic, as in a real-life scenario, the user would not want the volume to be dependent on the hardware implementation.

Due to this real-life scenario, we decided to keep our design as is, as we believe it is truer to the heart of the assignment.

All of these optimizations are in our final designs, with the initial, un-optimized versions commented out.

# Simulation and Performance

We simulated our design with UVM in questasim taking in the first inputs from the .dat file provided with the assignment to output 1000 outputs. Zero errors were reported during the simulation, which means our design fully compiles and outputs all 1000 outputs from the original inputs properly. While the full waveform is far too large to include in this report, some snippets, including the UVM output, are included below:

A white screen with blue text

AI-generated content may be incorrect.

A grid with green lines

AI-generated content may be incorrect. A grid with green and white lines

AI-generated content may be incorrect.

Looking at our output, we are able to analyze the performance of the system.

With our clock speed of 87.5MHZ (a period of 11.43 ns), we analyze 1000 outputs (with 4 bytes per output) in 196398 cycles.

* 11.43 ns/cycle \* 196398 cycles = 2.24ms
* 4000 bytes / 2.24ms = 1.79MBps

After adding execution time tracking code to the original source, we found the output of the same amount of data to be 14ms:



* 4000 bytes / 14ms = 0.29MBps
* 1.79/0.29 = 6.17x Speedup!

To look at the individual functions in the FM Radio system, we analyzed the simulation output for our module test benches. The results are in the following table:

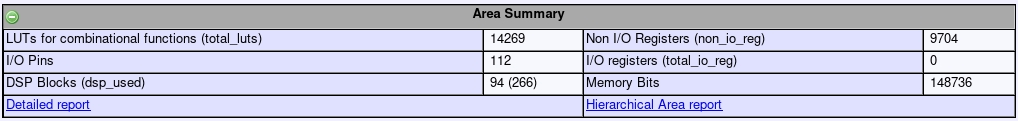
### Performance Analysis of Modules

|  |  |  |  |
| --- | --- | --- | --- |
| Module Name | Cycle Count (for 4000 bytes) | Run Time (ms) | Output Rate (MBps) |
| FM\_Radio\_Top | 196398 | 2.24 | 1.79 |
| Read\_IQ | 5004 | 0.0572 | 69,900 |
| Demod | 24825 | 0.284 | 14,100 |
| FIR\_Complex | 8003 | 0.0915 | 43,700 |
| FIR | 1504 | 0.0172 | 233,000 |
| IIR | 7004 | 0.0801 | 49,900 |
| Add | 2003 | 0.0229 | 175,000 |
| Sub | 2003 | 0.0229 | 175,000 |
| Mult | 5008 | 0.0572 | 69,900 |
| Gain | 2004 | 0.0229 | 175,000 |

With this data, it is clear that the demodulate module is our bottleneck. This makes sense as it has the largest FSM due to it having to calculate division and qarctan, leading to nested FSM. We can confirm this by observing the demod\_out\_full signal in our waveform. This signal represents the full state of the FIF that our demodulate instance fills with its output. In our simulation, this FIFO never gets full, meaning this is the point at which data is bottlenecked.

# Synthesis Results

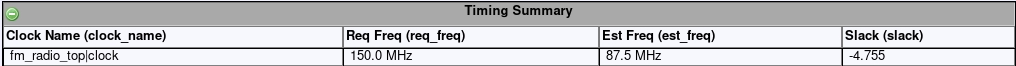
We synthesized our design in two different ways. We first synthesized without optimizations to the gain module, then with optimizations. The resource utilization is about the same for both, but the clock speed is greatly different. The board utilization stats are below:



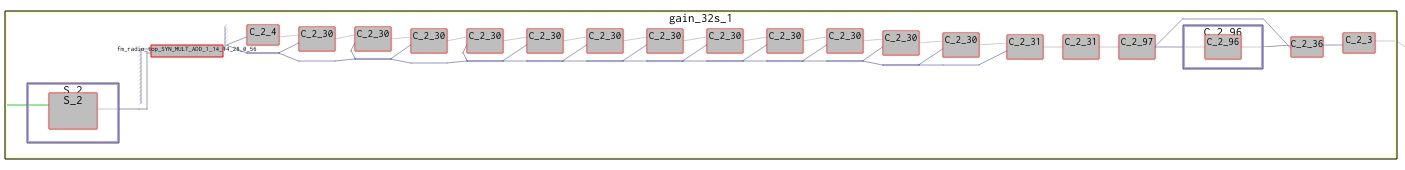
Our LUT usage comes from the dozens of constants present in the FM radio system. Most of these are dedicated to the tap coefficients for the FIR and IIR modules. The memory bits are likely all dedicated to our FIFOs. FIFOs along paths that are longer have smaller capacity, and those on shorter paths have larger capacity. This helps balance out the imbalance of cycles needed to push data through each path in the system. We decided to strike a balance between minimizing resources and improving the stream throughput. The FIFO size does have room to decrease significantly (from a base size of 128), however the board that we are targeting has a large amount of storage, so we opted for a lower cycle count.

We use a large number of DSPs in our design, but this number is influenced by our loop-unrolling. Once again, the resource usage can change based on the unroll factor, but we optimized for clock speed given the resources available on board.

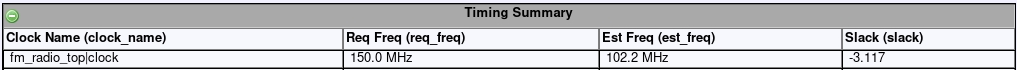
As we discussed in the design and optimizations sections, we have decreased our critical path length and upped our clock speed over various iterations in our design. Without gain optimization, we have the following clock speed:



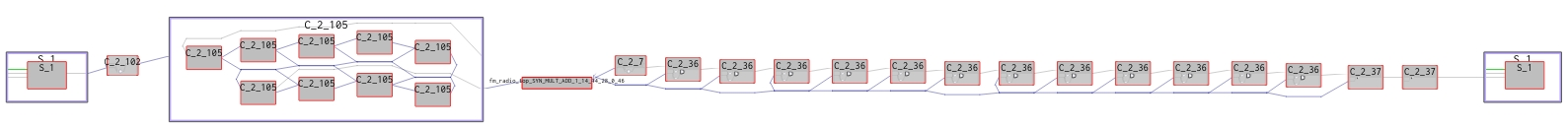
The critical path is through gain as shown here:



With optimizations to gain, we can achieve the following:



This turns the critical path into one through demodulate:



With further optimization, we believe that we could achieve an even higher clock rate.

Additional FSMs are attached below.

A diagram of a diagram

AI-generated content may be incorrect.

Read\_IQ FSM

A diagram of a diagram

AI-generated content may be incorrect.

Qarctan

A diagram of a computer process

AI-generated content may be incorrect.

Divider

A diagram of a computer process

AI-generated content may be incorrect.

FIR Complex (and most other modules)