1.5em

CE 387 Final Project: FM Radio Design and Verification in System Verilog

Tyler Dempski and Jack Thoene

Friday, March 6th 2024

1 Introduction

In this assignment, we are tasked with constructing a streaming FM Stereo Radio modeled after the provided C software model. Our objective is to implement each of the function blocks in SystemVerilog, following the guidelines and utilizing the strategies detailed below. The project involves both software and hardware components, with a focus on digital signal processing and SystemVerilog design principles.

2 Submission Requirements

2.1 Implementation and Compilation

- Utilize the attached C software model for reference.
- Implement the function blocks in SystemVerilog.
- Compile the software and observe the FM Radio's functionality using raw binary data found in the ”test” directory.
- Adjustments may be necessary for different operating systems; recommendations include Cygwin for Windows and MS Visual Studio for debugging.

2.2 Design and Verification Strategies

- Implement C macros as inlined SystemVerilog functions.
- Maintain the same quantization bit width as in the software model (10-bits).
- Utilize a package for filter coefficients, configuring filter taps and coefficients via parameters.
- Replace high-level for-loops with streaming FIFOs.
- Unroll internal function loops, such as those in shift registers.
- Apply optimizations like loop unrolling or pipelining.
- Implement the division algorithm for the qarctan function as discussed in class.
- Optimize FIFO buffer sizes to achieve a target frequency of 100MHz.
- Validate the design using a UVM model and ensure the simulation runs to completion with the provided script.

2.3 Simulation and Verification

- Modify the C program for testing small subsets of data for rapid design and test cycles.
- Include Modelsim `sim.do` and `wave.do` files, UVM model, and input & output simulation data files.
- The simulation should execute start-to-finish with the `run_simulation` script, detailing total cycles and identifying any errors.

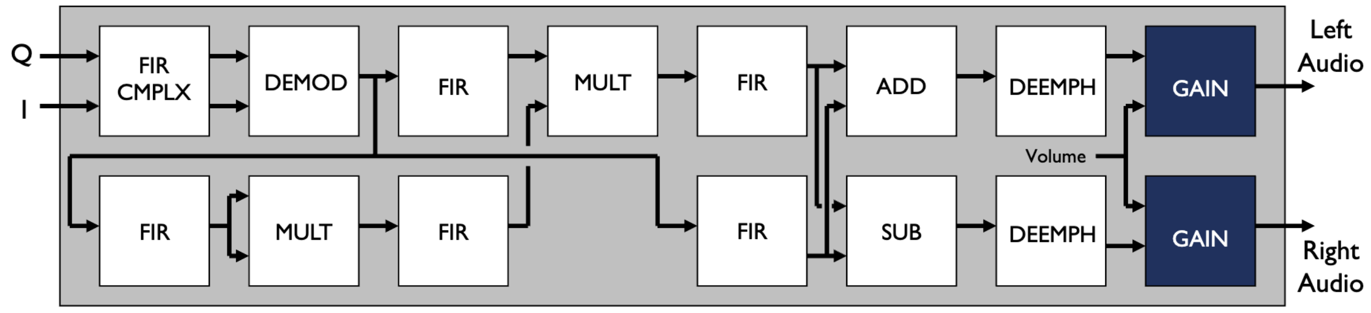
3 FM Radio Basics

Frequency Modulation (FM) radio uses the frequency of a carrier wave to encode audio signals, offering clearer sound compared to Amplitude Modulation (AM) by its resilience to static and noise. FM transmissions are allocated within the Very High Frequency (VHF) band of the electromagnetic spectrum, specifically from 88 to 108 MHz, for public broadcasting to minimize interference.

- **Frequency modulation:** Varying the carrier frequency in direct proportion to the amplitude of the input audio signal allows FM to transmit sound with high fidelity and reduced susceptibility to noise.
- **FM Demodulation:** Extracting the original audio from the modulated carrier wave described above.
- **DSP basics; ADCs, and DACs:** Digital Signal Processing (DSP), including Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs), are fundamental: they convert analog audio signals to digital for processing and then reconvert to analog for output.
- **Fourier Series basics:** The decomposition of any periodic signal into sine and cosine functions facilitates the design of efficient signal processing algorithms and filters for FM radio. This takes a complex signal in the time domain and outputs separate signals in the frequency domain.
- **DSP Filtering:** Digital filters are essential for enhancing signal quality by removing noise and interference, ensuring clear audio output. These can be low pass, high pass, or band pass, for the purposes of this report.
- **FIR Filters and IIR Filters:** One might Employ a Finite Impulse Response (FIR) filters for their stability and phase response, and Infinite Impulse Response (IIR) filters for their analog filter emulation efficiency.
- **Stereo Pilot Tone:** A 19 kHz signal embedded in FM stereo broadcasts indicates the presence of stereo information, enabling accurate two-channel audio decoding.
- **Left and Right Interleaving/FM Receivers:** Technique for efficiently separating and processing left and right audio channels enhance stereo broadcasting quality.
- **Channel Filter:** Isolating the selected FM station's signal from all others received for focused processing.
- **L-R Channel Filter and Channel Reconstruction:** The modulation and reconstruction of the stereo audio using the sum and difference of left and right channels maintain the integrity of stereo sound.
- **De-emphasis:** Applied to counterbalance pre-emphasis on higher frequencies at transmission, improving signal-to-noise ratio and sound quality.
- **Volume:** Allows listeners to adjust the output level of the audio to their preference.

4 System Architecture

Below is a functional block diagram depicting the datapath of the modulated audio data from input to output.



Overview of block diagram from class

4.1 Design Process and Approach

The design process for our project began with an in-depth analysis of the provided C++ models, focusing on understanding the inputs, outputs, and functionality of each module. This stage laid the groundwork for the entire project by identifying the key components and their interactions within the system.

After gaining a clear understanding of the models, we moved on to visualizing the system's workflow through the above provided flow chart. This step helped clarify the process flow and identify critical points within the system, making it easier to approach the design with a unified vision, especially as it pertained to number of inputs and outputs. In this stage, we drew out the top level module to understand these inputs and outputs and general dataflow, and settled on the below modules:

- Fifo In
- Input IQ
- Read IQ
- FIR CMPLX
- Fifo Demod
- Demod
- FIR
- Fifo Mult
- Mult
- FIR
- Fifo Add
- Add
- Deemph
- Gain
- FIR
- Mult
- FIR
- FIR
- Fifo Sub
- Sub
- Deemph
- Gain
- Output
- Fifo Left Output
- Fifo Right Output
- UVM
- Top Level

The Top-Level module was of particular importance as it integrated the numerous identified modules with seven FIFOs for data flow management.

Following the task enumeration, the team focused on building and testing the individual modules. This involved creating unit-test testbench modules for each component, using dummy data to ensure they produced the expected outputs. This step was crucial for identifying and fixing issues early in the development process, which in turn made it easier to integrate into the larger system.

The final stage of the design process involved the creation of a Universal Verification Methodology (UVM) testbench for the system's final verification.

4.2 Optimization Techniques

The majority of our architecture is designed with configurability in mind with regard to optimization for both speed and area.

To improve performance and resource usage of the naive solution, we use several structures of such as FSMs and Sequential Shift registers to improve the synthesis tool's use of resources. Following these design patterns also improves the reusability and readability of our code.

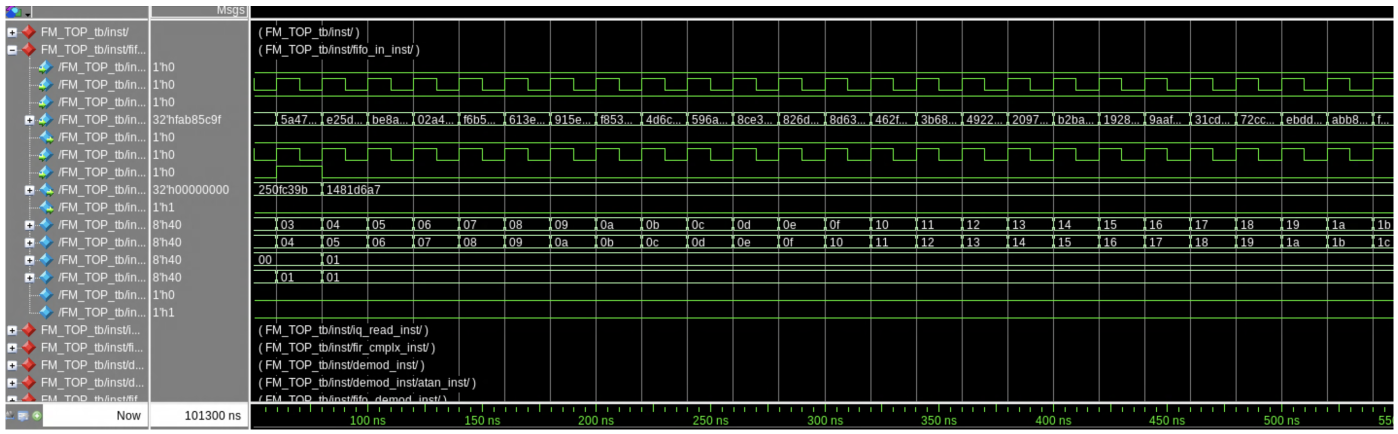
All modules are parameterized with values that can be adjusted to improve performance beyond the naive solution. One of these parameters, present in the FIR, IIR, and FIRCOMPLEX is the option for unrolling multiplication. These modules perform several multiplies over their operating time, which causes the cycles per value to be very high. By adjusting the multiplications per cycle parameter, the output can instead be calculated in batches in parallel. This parameterization has its tradeoffs however, as too much unrolling causes the module to over extend the resources of the on board DSP's, which requires a serialization of multiplies. Serialization of critical path steps has a linear effect on cycle time, and therefore must be minimized.

There are several ways in which performance could be further optimized given more development time that would decrease the total cycle time. The majority of this performance gain would come from increasing the amount of states in a module to allow for higher granularity in operations during a cycle, which would reduce critical path serialization and decrease cycle time. This comes with tradeoffs however, as decreasing clock period at the cost of cycle count can increase total latency, therefore inserting registers along a critical path must be done iteratively to prevent sharp increases in cycles per value.

5 Simulation

Simulation was conducted on individual models as well as the top level module. For the purposes of reducing total development time, individual modules were tested using ModelSim and compared against the C++ code provided. Modules were extracted from the FM Radio C++ code and run with identical inputs to the testbench to ensure bit true accuracy.

The top level module was simulated with both ModelSim for initial connection verification and QuestaSim for UVM Verification. Due to the scope of the design UVM was preferable due to its ability to quickly evaluate correctness compared to manually evaluating waveforms.


Msgs
FM_TOP_tb/inst/
FM_TOP_tb/inst/fif...
/FM_TOP_tb/in... 1'h0
/FM_TOP_tb/in... 1'h0
/FM_TOP_tb/in... 1'h0
/FM_TOP_tb/in... 32'hfab85c9f
/FM_TOP_tb/in... 1'h0
/FM_TOP_tb/in... 1'h0
/FM_TOP_tb/in... 1'h0
/FM_TOP_tb/in... 32'h00000000
/FM_TOP_tb/in... 1'h1
/FM_TOP_tb/in... 8'h40
/FM_TOP_tb/in... 8'h40
/FM_TOP_tb/in... 8'h40
/FM_TOP_tb/in... 8'h40
/FM_TOP_tb/in... 1'h0
/FM_TOP_tb/in... 1'h1
FM_TOP_tb/inst/i...
FM_TOP_tb/inst/fi...
FM_TOP_tb/inst/d...
FM_TOP_tb/inst/d...
(FM_TOP_tb/inst/)
(FM_TOP_tb/inst/fifo_in_inst/)
5a47... e25d... be8a... 02a4... f6b5... 613e... 915e... f853... 4d6c... 596a... 8ce3... 826d... 8d63... 462f... 3b68... 4922... 2097... b2ba... 1928... 9aaf... 31cd... 72cc... ebdd... abb8...
250fc39b 1481d6a7
03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b
04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c
00 01
01 01
(FM_TOP_tb/inst/iq_read_inst/)
(FM_TOP_tb/inst/fir_cmplx_inst/)
(FM_TOP_tb/inst/demod_inst/)
(FM_TOP_tb/inst/demod_inst/atan_inst/)
Now 101300 ns
100 ns 150 ns 200 ns 250 ns 300 ns 350 ns 400 ns 450 ns 500 ns

Top Level Waveform

5.1 Performance

Our model's performance was limited by several patterns in the architecture that emerged as key sites for optimization.

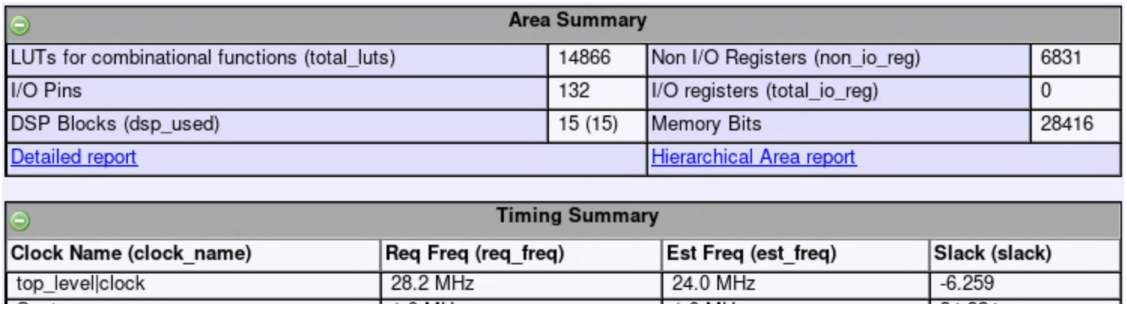
The critical path from the top module begins at the input, travels through the Demodulator, into the pilot tone sequence of two FIRs and a Multiplier, then into another sequence of a multiplier and FIR, and finally through the Add, Deemphasis, and gain phases.

This path acts as a bottleneck as the two other data paths stemming from the output of the Demodulator must wait for its arrival to be processed. For this reason it is more effective to use DSP resources unrolling pilot tone FIRs than any other, as it will ease the queue times before the demodulator outputs are processed in parallel.

The input processing is the second most costly pipeline, as the FIRCOMPLEX and demodulator take many cycles to complete their operation. Balancing optimization between these two regions requires careful examination of how much each level of unrolling bottleneck issues against how much additional resource usage (and potential DSP serialization) it incurs.

6 Synthesis

The top level design was synthesized using Synplify Premier. This required that the design adhered to the priniciples of synthesizable System Verilog



Area Summary

LUTs for combinational functions (total_luts)	14866	Non I/O Registers (non_io_reg)	6831
I/O Pins	132	I/O registers (total_io_reg)	0
DSP Blocks (dsp_used)	15 (15)	Memory Bits	28416
Detailed report		Hierarchical Area report	

Timing Summary

Clock Name (clock_name)	Req Freq (req_freq)	Est Freq (est_freq)	Slack (slack)
top_level|clock	28.2 MHz	24.0 MHz	-6.259

Area and Resource Allocation

The total resource usage and cycle time were not ideal, and we would have liked to improve upon the efficiency of our design, however due to the time restrictions of this endeavor, we prioritized the correctness of the design over optimization.

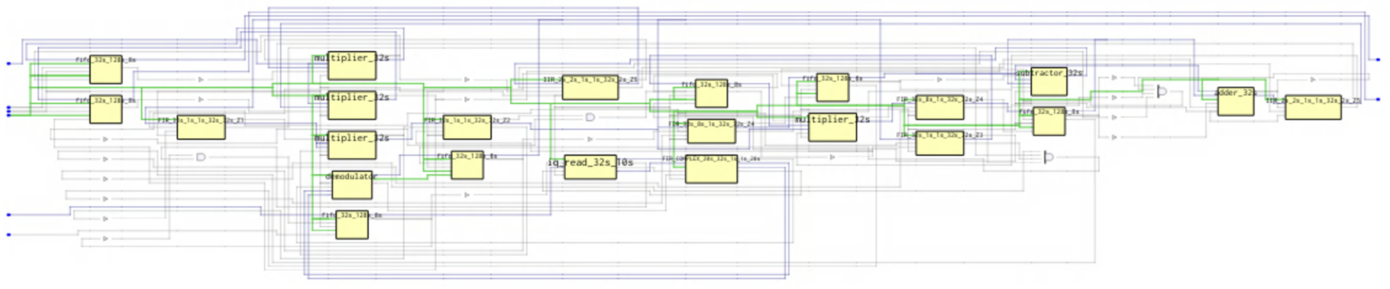
Addressing the requirement of performance in Mb/s, below is the calculation converting a 1 bit output every 32 cycles operating at 24 MHz:

$$\text{Data Rate (Mbps)} = \frac{\text{Cycles per Second}}{\text{Cycles for One Bit} \times 10^6} = \frac{24 \times 10^6}{32 \times 10^6} = 0.75\,\text{Mbps}$$

This latency reflects the bottleneck experienced at the arctan module.

6.1 Top-Level Module

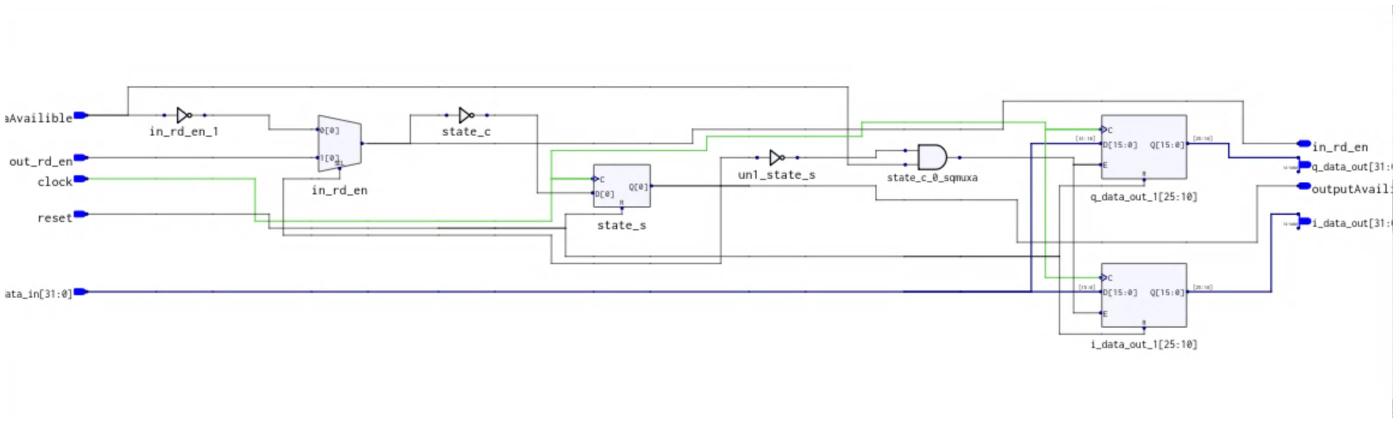
The RTL view for the top level module very closely resembles the block diagram provided for the FM radio concept.



Top Level Synthesis

6.2 Read IQ

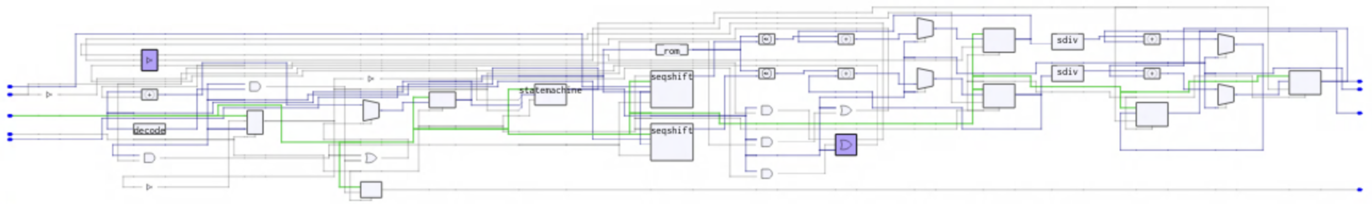
The data path begins by reading the IQ value from the FFT into a usable form for processing.



IQ Read Architecture

6.3 FIR Complex

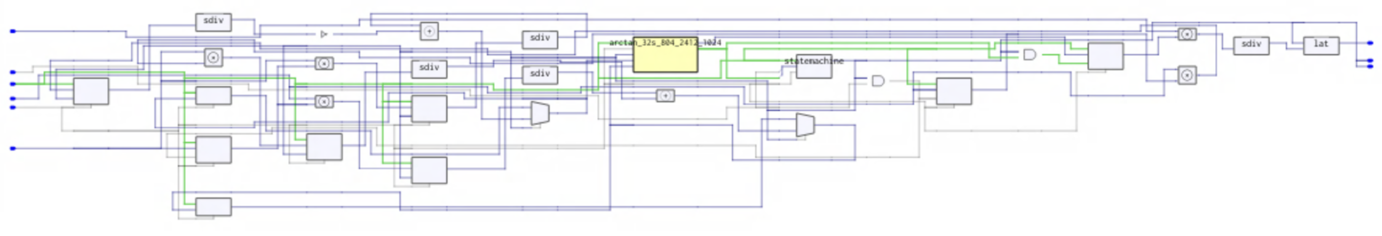
The FIR complex operates similarly to a single FIR, however dot products are calculated by interleaving the values of two tap arrays and data queues. Queues are successfully synthesized as sequential shift registers. This module is parameterized to allow for loop unrolling to improve performance by calculating multiple dot product addends in parallel.



FIR Complex Architecture

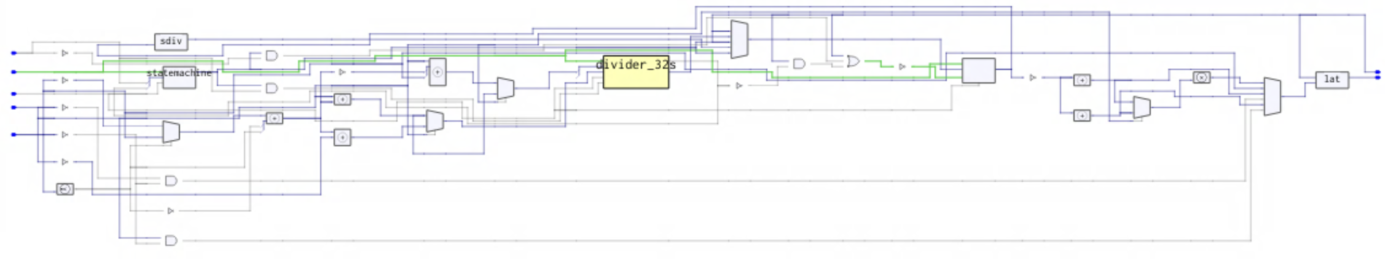
6.4 Demodulator

In order to process the signal further, the complex values read in must be converted into scalar values with the demodulator. This module combines the I and Q outputs of the FIR Complex, and calculates the arctangent of the combined values. This angle result is then amplified by a constant gain, and passed on for processing



Demodulator Architecture

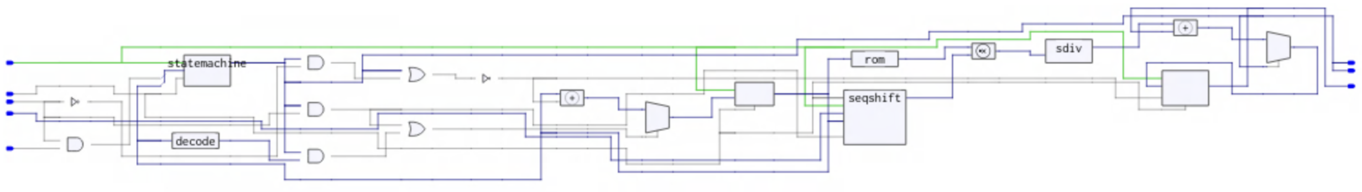
The arctangent uses division and trigonometric identities to evaluate the angle that would represent the result of an arctangent taken on the two inputs, in radians.



Arctangent Architecture

6.5 FIR

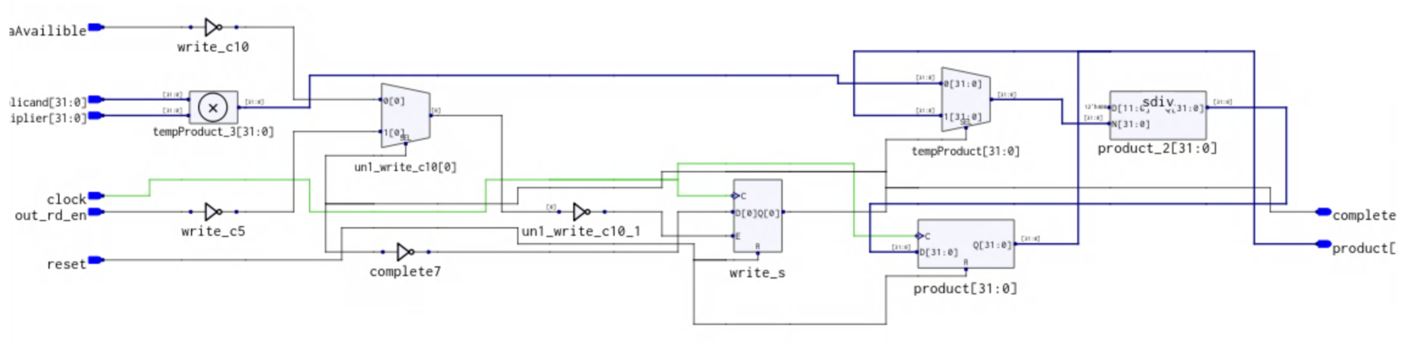
The FIR is the most utilized module in the FM Radio. It allows for the device to apply low pass, high pass, and band pass filters to the signals it is provided. It does so by caching input signals in an internal sequential shift register, which applies constant ”tap” coefficients to, and outputs the sum. This module is parameterized for any number of taps, level of unrolling, and tap values, so that it can be easily configured and reused throughout the design.



FIR Architecture

6.6 Functional Units

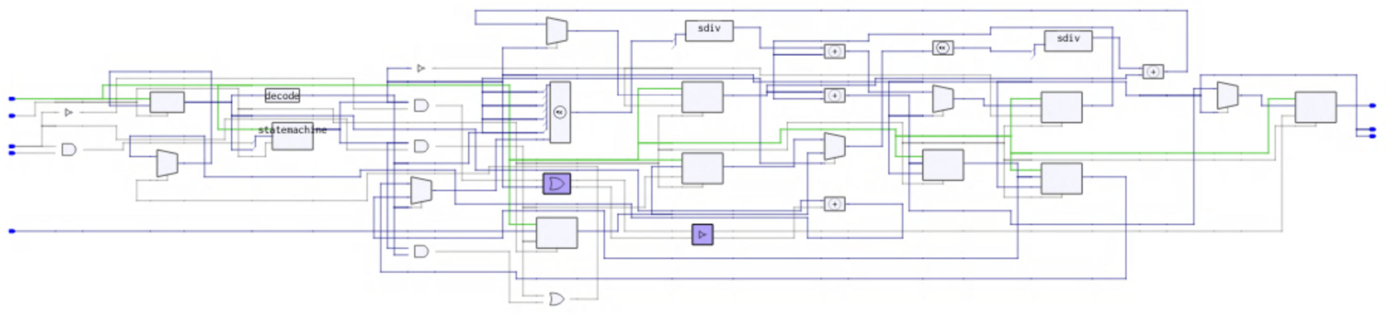
The adder, subtractor, and multiplier operate in similar ways, with the only difference being the arithmetic operation performed. They are implemented as simple state machines with a read and a write phase, rather than combinational circuits. This helps with reducing critical path length, as well as allowing the module to easily hold its previous value when required to stall due to a bottleneck further down in the datapath.



FIR Complex Architecture

6.7 IIR

The final processing module in the sequence is the IIR, which operates similarly to the FIR, with the addition of a cache and set of tap coefficients for previous outputs. This module deemphasizes the signal, where it is multiplied by volume and output.



FIR Complex Architecture

7 Conclusion

In conclusion, this project was successful. We were able to translate a functional C software model into a RTL system that functions in an appropriate and functional manner.

The main blocker for achieving full performance and minimized resource space was predominantly the amount of time available to develop. Given an extended period of time, we have no doubts in the optimized functionality of this base system.

Our efforts in optimizing the design for performance and resource usage, while ensuring bit-true accuracy and functionality, underscore the importance of a thoughtful and iterative design approach. Despite facing challenges in synthesis and resource allocation, our project prioritizes correctness and functionality. capture