**EECS 395 Final Report**

By: Jae Woo Ok

**Introduction**

In this project, an FM radio was created using digital logic in VHDL using the techniques and methods learned in the class. This report will go into depth about the background knowledge of an FM radio and DSP (or digital signal processing) and then will describe the general data path and the different components that went into creating this design.

**Basic Background Knowledge**

**FM Radio**

An FM radio uses the principle of frequency modulation to broadcast signals. The original signal is added to a carrier wave before it is broadcast, which is less prone to signal deterioration by noise than the older amplitude modulation method. The original signal can be derived by removing the carrier frequency from the received signal. Useful information is encoded in the original signal, such as mono audio, stereo audio, and a stereo pilot tone.

**Fixed Point Numbers**

Fixed point numbers have three components: a sign bit, integer bits, and fractional bits. As expected, the sign bit determines if the number is positive or negative, the integer bits determine the value of the number part, and the fractional bits determine the value of the fraction.

**Quantization**

Quantization is a mapping of a set of numbers to a different set. Quantization can be used to map fixed point real numbers to integer numbers. One of the benefits of quantizing fixed-point numbers is to allow for easier arithmetic operations. Integer operations are typically simpler and quicker to compute than real number operations. Fixed point numbers can be quantized by left shifting by the number of fractional bits and dequantized by shifting right by the number of fractional bits. The total bit width of the number and the fractional bit width determine the overall accuracy of quantized fixed-point operations.

**Sampling**

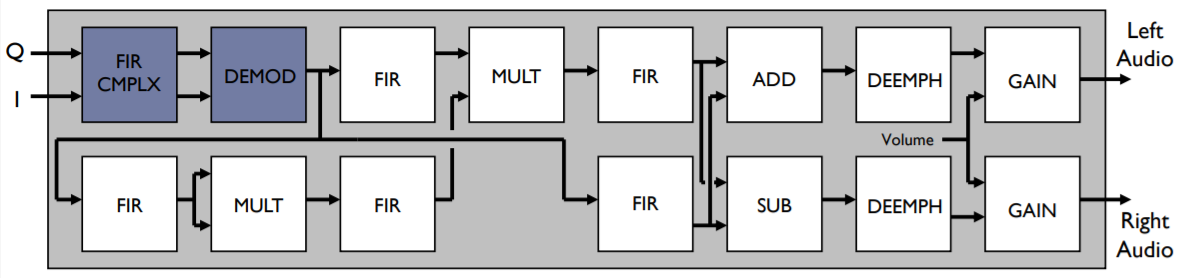
Continuous signals can be digitized by sampling, which is the process of reading the value of the signal at given time intervals. Sampling creates a discrete signal that represents a continuous signal. Some data is lost during sampling, so if the digital signal is converted back to a continuous signal, the quality may be decreased. The sampling rate determines how much data is captured and in turn the quality of the resulting signal.

**Filtering**

Filters are often used to isolate and extract information, or remove unwanted information, from a signal. Common filters include low pass, high pass, and band pass. Low pass filters filter out data at a frequency higher than some cutoff frequency and keep the low values. High pass filters are the opposite, keeping high frequency data and removing low frequency data. Band pass filters are a combination of

low and high pass filters, used to select a range of frequencies. Typically, a filter would be implemented with a combination of capacitors, resistors, and inductors, but it is possible to implement them digitally.

**Components and Datapath**



The data path and components that was used in the design followed the diagram above. The design uses FIFO architectures to stream the inputs in, outputs out, and to buffer the data. The components can be grouped into four broad sections: input, pilot determination, left branch, and right branch.

The input section includes an IQ-reader, FIR-complex, and a demodulator. The IQ reader takes input samples and splits them into an I and Q component. The complex FIR treats each I/Q pair as a complex number then filters them. The demodulator is used to find the phase change between I/Q samples to determine the original frequency, which in turn determines pitch of the sound. The output of the demodulator is piped to the other three sections of the design for further processing.

Next, in order to determine the pilot tone, the demodulated signal is passed through a 32-tap FIR band pass filter to isolate the 19 kHz tone. If the tone exists, it indicates that stereo data is available in the 23 kHz to 53 kHz range. A squared component is also used to create a reference point at 38 kHz, splitting the stereo range into two components. The act of squaring also creates a 0 Hz tone that is removed with a second FIR as a high pass filter.

The left branch is used to generate the left component of stereo audio. A 32-tap FIR band pass filter is used to isolate the 23 kHz to 53 kHz stereo audio range. The band is multiplied by the squared pilot tone to shift the frequencies down by 38 kHz. The result is passed through a decimating low pass FIR to isolate the stereo component from the rest of the data and to reduce the overall sampling rate by a factor of the decimating constant. The stereo data is added to the mono sound data generated by the right branch at an adder-subtractor component that joins the two branches temporarily. The addition isolates the left audio, which is deemphasized by passing it through an IIR filter. The filter result is passed through a final gain component, which multiplies the signal by a constant to change the volume level. The right branch is basically the same as the left branch, except the stereo sound from the left branch is subtracted from the mono sound to isolate the right audio.

**Optimization**

**FIFO Streaming**

The design had a streaming architecture through all the FIFO’s. Each data sample is piped through each of the components, one after the other, as opposed to calculating the result of each component on the entire data set, which allows the design to perform using fewer cycles overall.

**Parallelism**

Because HDL is a hardware description language, this allows for parallelism in the design process, which decreases the overall cycle count, and increases the speed tremendously. In this design, the output of the demodulator component is piped to three different branches at the same time. The demodulator output in each branch can be calculated concurrently in the hardware implementation and only serially in the software implementation.

**Loop Unrolling**

In the FIR and IIR components, compared to the software implementation, the loops in these components were converted into shift registers, since the filters operate by saving a buffer of previous data and shifting new data in and old data out. By using a shift register instead of creating a state machine to simulate a loop, many cycles were saved. However, loop unrolling does create a larger number of multiplier elements.

**Decimation**

Decimation is the process of ignoring some samples to allow for reduced processing, while maintaining a decent quality output signal. In the design, decimation was used as a method for synchronizing signals.

**Problems and Issues**

**Bottleneck Performance**

The overall path from demodulator through pilot to the left branch is significantly longer than the path directly from the demodulator to the right path. Without any form of optimization this leaves the right branch waiting a while for the left branch to catch up before data can be sent through to the output. The demodulator contains a divider, which does not run for a set number of cycles and forces latter components of the design to wait.

**Debugging**

It was much faster to debug the entire process by testing each of the components by itself first, before running it through the entire data path testbench.

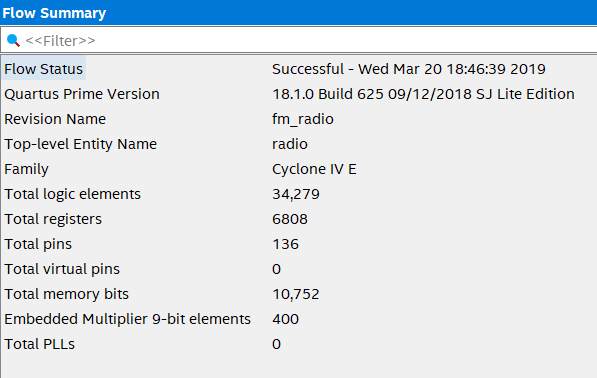
**Simulation, Performance, and Results**

**Performance**

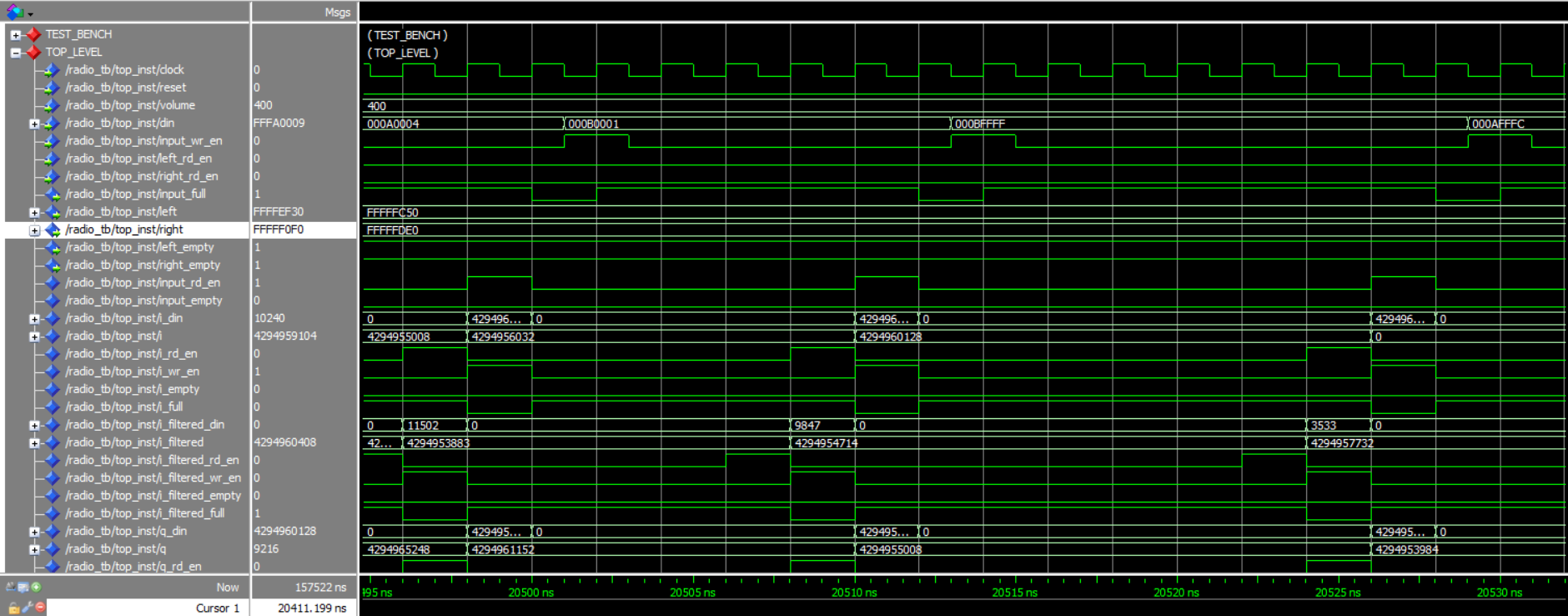
The simulated design can process 10000 inputs, creating 1250 output samples due to decimation. The simulation took 78757 cycles, which is about 8 cycles per input sample and 63 cycles per output sample. The FPGA processor is 50 MHz, so the design is capable of producing 793650 samples per second which more than exceeds the necessary 44 kHz typically used in digital audio processing. There is around 46 errors within the results, but that is only a margin of 3.6% error which is very small and within range for results in DSP.

\*\* The design is run by using the command **“do execute.do”** after the appropriate files are included into the ModelSim \*\*

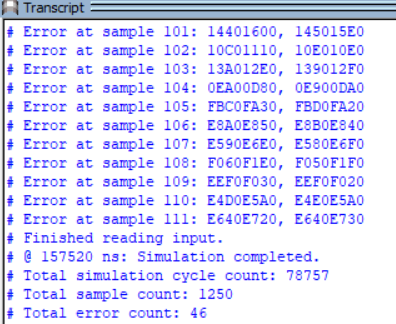
**Synthesis Results**



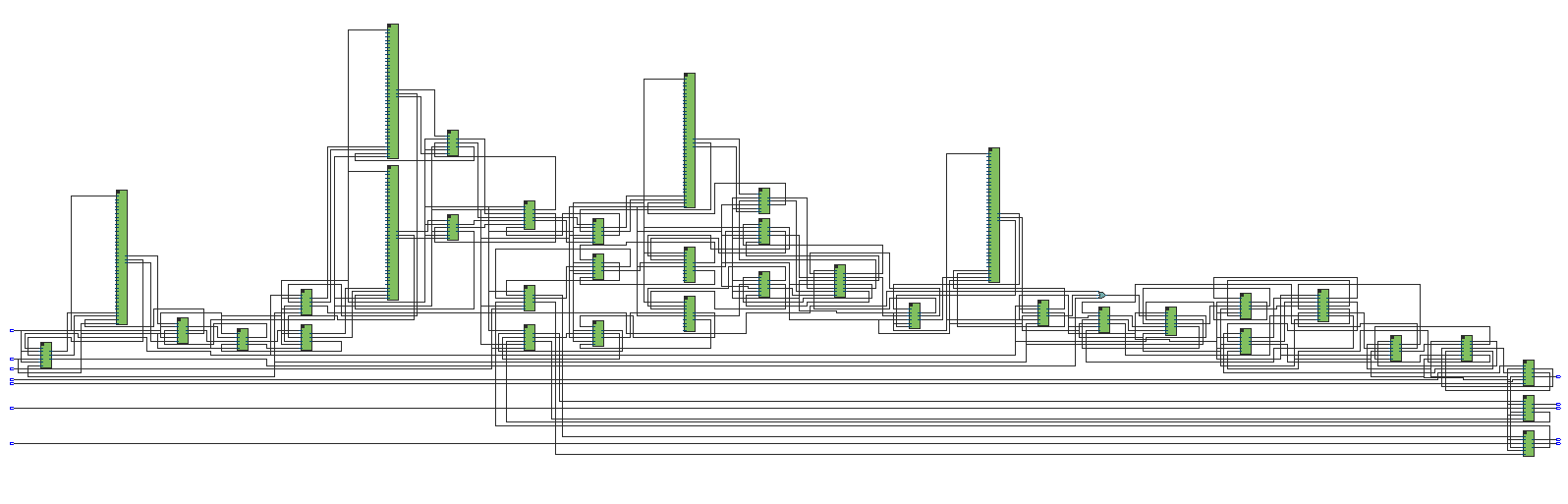
**Testbench Waveform**



**Cycle Count Results**



**RTL Diagram**



**\*\* A more detailed version of the diagram can be viewed through Quartus II \*\***