FPGA-BASED FM BROADCAST MONITORING, RECORDING & PLAYBACK SYSTEM

Group Members

Osama Tahir	18i-0832
Muhammad Zaid	18i-0789
Muhammad Ali	18i-0758

Project Supervisor

Dr. Rashad Ramzan

Department of Electrical Engineering

National University of Computer and Emerging Sciences, Islamabad

2022

Developer's Submission

"This report is being submitted to the Department of Electrical Engineering of the National University of Computer and Emerging Sciences in partial fulfillment of the requirements for the degree of BS in Electrical Engineering."

Developer's Declaration

"We take full responsibility for the project work conducted during the Final Year Project (FYP) titled **"FPGA-BASED FM BROADCAST MONITORING, RECORDING & PLAYBACK SYSTEM"**. We honestly declare that the project work presented in the FYP report is done entirely by us with no significant help from any other person; however, small help wherever taken is duly acknowledged. We have also written the complete FYP report by ourselves. Moreover, we have not presented this FYP (or substantially similar project work) previously to any other foreign or national degree-awarding institution. We understand that the management of Department of Electrical Engineering of the National University of Computer and Emerging Sciences has a zero-tolerance policy toward plagiarism. Therefore, we as the author of the this FYP report solemnly declare that no portion of our report has been plagiarized and any material used in the report from other sources is properly cited. Furthermore, the work presented in the report is our own work and we have positively cited the related work of the other projects by clearly differentiating our work from their relevant work.

We further understand that if we are found guilty of any form of plagiarism in our FYP report even after our graduation, the University reserves the right to withdraw our BS degree.

_____	_____	_____
Osama Tahir	Muhammad Zaid	Muhammad Ali
BS(EE) 18I-0832	BS(EE) 18I-0789	BS(EE) 18I-0758

_____

Certified by Supervisor

_____

Verified by Plagiarism Cell Officer Dated: _____

Abstract

Radio has been one of the simplest yet most effective modes of communication since 1920s. This project focuses on Frequency Modulation (FM) based radio systems as they are the most widely used systems, offering high degree of noise immunity. Conventionally, radio systems have been implemented using analog components but these systems lack multi-channel data processing capability. So, in this project we have implemented a digital, FPGA based FM receiver which can automatically detect, stream, record and playback FM channels of a particular area for security monitoring. This system is unique in its capability to record multiple channels simultaneously.

Acknowledgments

The final year project “FPGA- Based FM Broadcast Monitoring, Recording and Playback System" was successfully completed in the RFCS^2 Lab of National University of Computer and Emerging Sciences (FAST-NUCES), Islamabad Campus under the Pakistan Engineering Council (PEC) Annual Award for Final Year Design Projects (FYDPs) for the year 2022-2023. The project was supervised by Dr. Rashad Ramzan.

We would like to express our sincerest gratitude to our supervisor, Dr. Rashad Ramzan for providing us with the opportunity to work under his guidance. He provided us with invaluable guidance and unprecedented help. Without his support, the project would not have been possible. We would also like to express special thanks to Dr. Hassan Saif and Sir Mudassir Ali for their brotherly support and to our families who put up with us throughout.

Table of Contents

Table of Figures

1 Introduction

This section offers explanation about the problem that this project addresses, the impact of its continued existence and our attempt to help in this situation. It explains how radio communication has influenced people's lives in the not-so-distant past. Further, it explains what this project is about and our method of implementation

1.1 Problem Statement

Authorities require automatic detection of active FM channels and their simultaneous recording. While keeping the implementation simple, a high-speed, robust and power efficient digital system is proposed, which will be able to stream and record the FM channels through automatic detection for security monitoring.

1.2 History & Impact

The fact that radio waves do not require a medium to be transmitted is of great significance and in part the reason for undertaking this project. Radio waves can travel through space undetected. As much as it is beneficial, it can be dangerous. With time, the listenership of radio has rose significantly and today, it's already on its peak [21]. Something which is so easily available and accessible by the general public poses a threat of mischief and terrorism, even to the innocents [11].

During the war times in Afghanistan, radio was the most widely used mode of communication. It was used by non-state forces as a part of their war strategy. The regulatory authorities needed to monitor the content on FM radios. Since this terrain was mostly hilly, therefore authorities required the monitoring equipment in a large number to use it locally therefore, the solution had to be economical. The influence that Mullah Fazlullah had in Swat District of Pakistan is not unknown. How he become the leader of an extremist group by his fiery speeches. Initially a ski-lift operator and later the chief of Taliban; Mullah Fazlullah. It was in 2005 that he realized the potential of radio and its ability to reach people from far away. Mullah Fazlullah started using unlicensed radio broadcasting to spread his extremist view among the people of Swat. Shortly, he got famous due to his firebrand speeches and earned himself the title of "FM Mullah", locally and internationally. He found FM broadcasting very easy to launch and cost-effective as a 10-watt FM radio transmitter costed about US $200, back then. With negligible technical skills required to set up the FM station he could send his voice to each home across the village. His speeches and sermons gained significant popularity among the locals; people would wait for him to broadcast again soon. He had become a household name. He attracted hundreds of Taliban recruits from the districts of KP [23]. So much so, that on his arrest by the security forces, the villagers came to his rescue. All these details tell the extent of his influence. In such scenarios, monitoring of communication channels as simple as FM becomes necessary.

Moreover, in countries where free expression is suppressed and access to technology is expensive, radio continues to play an important role in information sharing. Even though new technologies are increasing, none have reached the simplicity and effectiveness of traditional radio [2]. Radio is still the most dominant long distance communication medium in Africa, reaching further than newspapers and television, both in terms of audience and geographical reach [22]. In such cases, organizations like the UN, WHO, NGOs, etc. are compelled to use radio to stay involved with the local communities and remain updated.

1.3 Motivation

In this age of wireless technologies where there is an overload of data, monitoring this data for people's security has become equally important. Authorities want to deter the threat of people exploiting unmonitored modes of communication and thus require to maintain a record of wireless communication. This project is aimed to resolve these concerns and enable the relevant quarters to stream, record & playback every transmission passing through the nearby air.

1.4 Literature Review

The idea of wireless telegraph first came to life in the 1890s and has evolved a great deal since then. FM transmission is one of the most effective yet simple mode of wireless communication. The Federal Communications Commission (FCC) allocates a band of 88-108 MHz for FM broadcasts with each channel having a bandwidth of 200 kHz [6]. Theoretically, noise power of a signal is proportional to its modulated bandwidth, so there were efforts to find a modulation scheme which would have a lower bandwidth. However, this reasoning didn't fit well in context to the experimental results of frequency modulation which had a higher bandwidth [26]. The Major benefit of FM systems is their high degree of immunity to noise. In fact, these systems trade bandwidth for higher noise immunity. This is the reason that FM systems are preferred for high-fidelity content broadcasting and other communication systems, where the transmitter power is limited. Another advantage of frequency-modulated signals is that their quality is not affected if amplified by a non-linear amplifier. This is due to their constant envelope [25]. Conventionally, these radios have been implemented using analog components but ever since the advent of high-speed Analog-to-Digital Converters (ADC) these systems have transitioned into much more robust digital devices. Moreover, the implementation of radio on a Field Programmable Gate Array (FPGA) commonly known as Software Defined Radio (SDR) provides the flexibility of reconfiguring the whole system for different radio communication schemes such as AM or PM without even altering a single physical component.

In order to understand the design of analog front-end the architecture of a superheterodyne receiver was studied and understood. After receiving, filtering, amplifying, and digitizing the whole FM band with the help of the RF front-end, the next part is to identify the active FM channels. For this purpose, different spectrum sensing techniques were explored. After reviewing several spectrum sensing methods such as matched filter, energy detection, and FFT, it was concluded that FFT in combination with a peak detector qualified as the best technique since it

gave a high signal-to-noise ratio (SNR) [7]. Further, DFT via FFT can be implemented using various algorithms such as Good-Thomas, Cooley-Tukey, Radix-2, and Rader's algorithm. After research, it was concluded that the radix-2 algorithm proves to be the most time and resource-efficient algorithm [3]. For filter design, several approaches were discovered. MATLAB HDL coder was found to be the most optimized option. Other approaches included filter design using Xilinx FIR compiler, and through DFT. The demodulator is made using the IQ demodulation technique [4]. Not only IQ demodulation technique is as universal as PLL based demodulator, on top of that its implementation on the FPGA was found to be easier than the latter. For the last stage of digital to analog conversion, a very hardware appropriate technique was explored; PWM with a reconstruction filter [27]. This is well suited for this project since the available FPGA development board already has a mono output port with reconstruction filter on it.

1.5 Project Overview

1.5.1 Description of Block Diagram

Given below in *Figure 1.1* is the block diagram of the entire project, explaining the entire project flow. The FM signals are first received through the analog RF front-end which conditions the FM signals to be read by the ADC. Then the ADC converts these signals into digital data. This digital data is processed by the FPGA, where it detects the active FM channels, extract each channel through filtering and then demodulates them, so that they can be played or stored in the memory. The purpose of each block and its importance is discussed in the subsequent sections.

1.5.1.1 Analog Front-end

The project begins with an FM-relevant RF front-end, which is used to catch FM signals from the air and condition them. Further, it down converts the frequencies to a lower frequency by mixing them with 86 MHz sine wave so that they can be properly sampled by the ADC, abiding well by the Nyquist criteria.

1.5.1.2 ADC

The ADC then converts these RF signals into digital signals. It is a high-speed ADC with a maximum sampling rate of 65 Mega Samples per Second (MSPS) and a resolution of 12 bits. This digital data is fed to the FPGA through 12 wire parallel interface.

1.5.1.3 DFT via FFT

To obtain the frequency spectrum, a 2048-point Discrete Fourier Transform (DFT) is performed on this data. To optimally perform the Fourier transform, Radix-2 Fast Fourier Transform algorithm is employed.

1.5.1.4 Peak Detector

The peak detector calculates the magnitude spectrum and finds the dominant spectral peaks. The Fourier Transform produces complex output and a comparison of signal strengths cannot be done as is, so first magnitude spectrum is computed. Then these signal strengths are compared with a threshold value to differentiate between FM channels and noise.

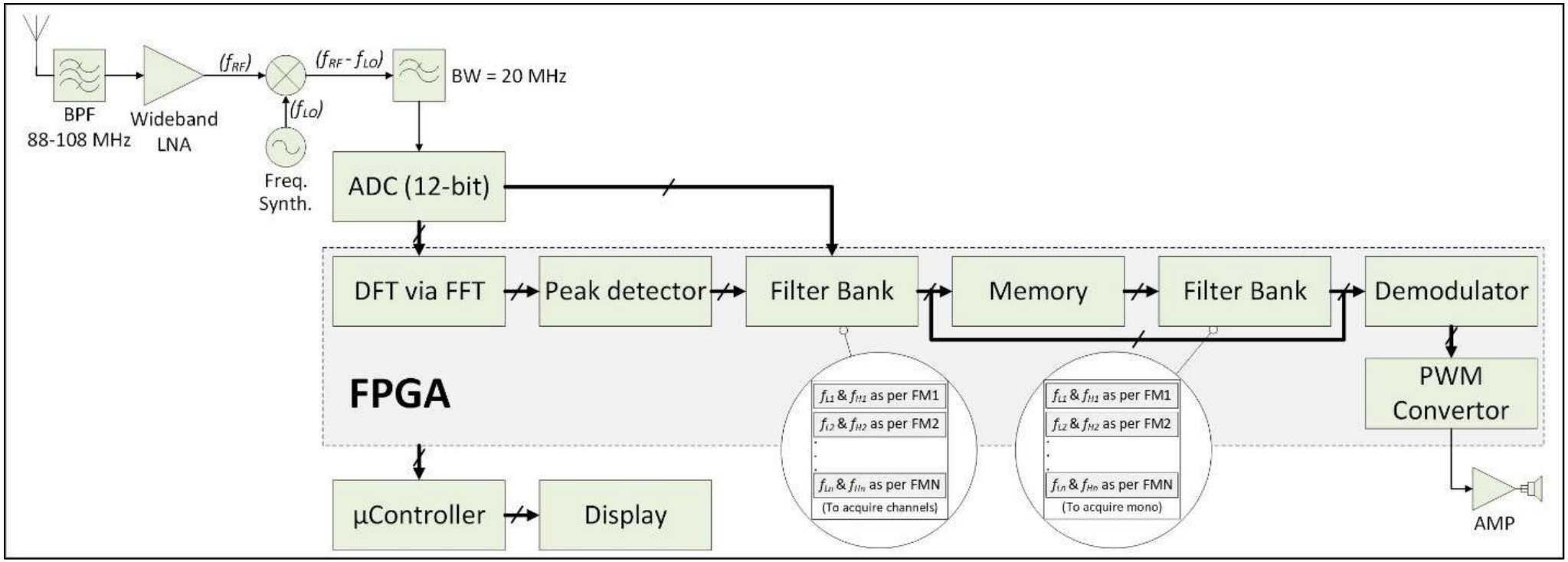


Figure 1.1: Project Block Diagram

1.5.1.5 1st Filter Bank

Based on the dominant frequencies detected by the peak detector, filter banks filter these frequencies. This first bank extracts an individual channel of 200 kHz bandwidth completely. The idea is that the entire channel information could be stored in memory as well if the user requires.

1.5.1.6 Memory

The extracted channels can be stored directly into the memory, entirely or after demodulation. In the latter case one may lose the extra information such as stereo data of the FM channel but content monitoring would still be applicable.

1.5.1.7 2nd Filter Bank

The second bank extracts the mono band of the FM channel which lies in the first 15 kHz of the channel. This band is sufficient for retrieving the audio from the channel.

1.5.1.8 Demodulator

After extracting the mono band of a channel, the selected channel is demodulated using the IQ demodulation scheme.

1.5.1.9 PWM Converter

The audio extracted from demodulation is converted to a PWM wave. This replaces the need for an extra peripheral i.e., Digital to Analog Converter (DAC). This PWM wave when passed through a reconstruction filter, results in a perfect analog signal and can be played through a speaker.

1.5.1.10 User Interface

For controlling the FPGA and maneuvering between different options easily, a simple user interface with an LCD screen is designed. At the heart of the interface is an Arduino Mega which communicates with the FPGA and display received data on the LCD Screen.

1.5.2 Description of Flowchart

Given below in *Figure 1.2* is the flowchart of the entire project. It explains each necessary step in order of operation.

The FPGA receives the digital data and performs a 2048-point Fourier transform to obtain the frequency spectrum. This spectrum information is stored in memory since it is later needed for peak detection for the comparison of signal frequency strengths. The maximum and minimum values of the spectrum are searched. These values are then used to compute the midrange which serves as our threshold. Each signal's strength is compared against the threshold and if the signal's strength is greater than this threshold, they are noted else discarded. The detected signal frequencies are then sorted based on their magnitude. The user can now opt for either of two options; streaming or recording. On selecting a channel, it is filtered, demodulated, and stored in memory or converted to PWM signal for it to be played through a speaker. Later on, If the user opts for playback, the screen displays the list of recorded channels and he can select from among them. The selected channel is then converted to PWM wave and can be played through the speaker.

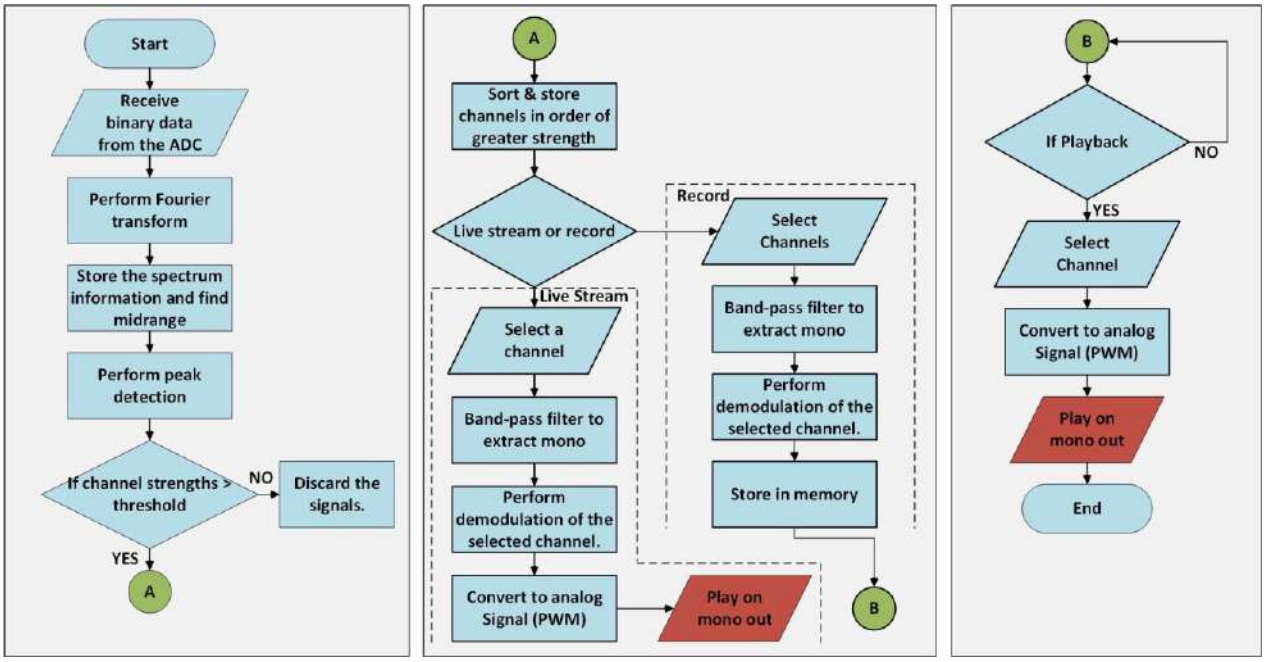


Figure 1.2: Project Flowchart

2 Implementation of the FM System

This section discusses in detail, how the entire system is implemented. It explains how the proposed approach is first verified with the help of system simulations and then the process of implementation is started, based on the outcomes of the simulation. It further explains the implementation of each deliverable separately.

2.1 Simulation of the Complete FM System

Before the actual system level implementation, entire project is initially simulated using MathWorks MATLAB to verify the practicality of the idea and design flow. The simulation assumes the FM signals to have been received through an RF front-end which also down-converts the FM spectrum to 2-22 MHz range. Only four FM channels are considered here as a test case, as shown in *Figure 2.1.*

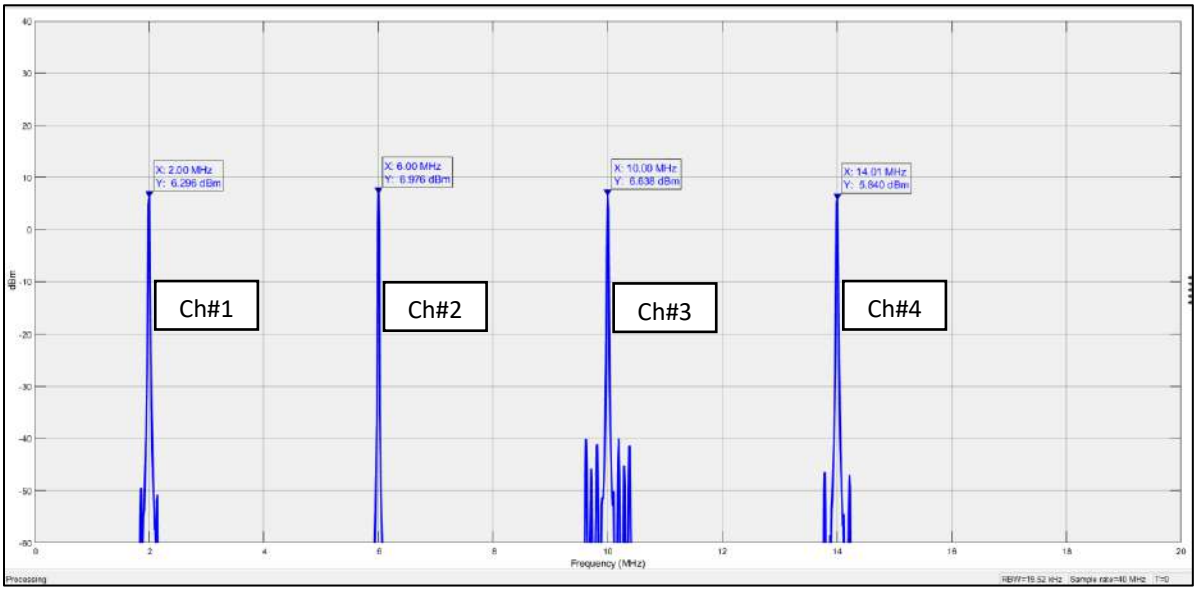


Figure 2.1: 4 Fm Channels

For an FM receiver, the first step is to find the carrier frequency of a channel. For that, FFT is performed and further the magnitude spectrum is computed. 2048-point FFT is computed on a dataset of 480,000 samples. The magnitude spectrum can be observed in *Figure 2.2.* It can be observed that 2048-point Fourier transform proves to be reasonably accurate since it results in very minute deviations from the actual carrier frequencies.

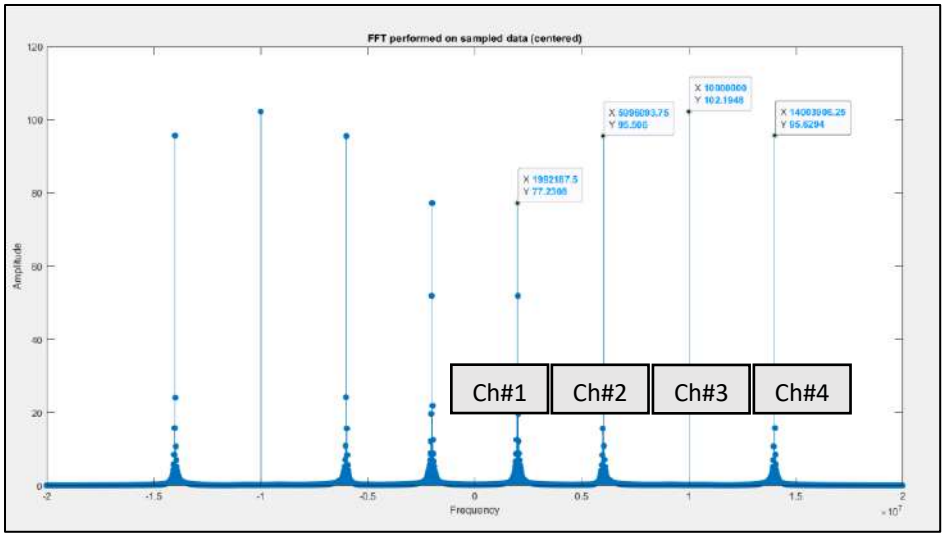


Figure 2.2: 2048-Point DFT

The midrange of this magnitude spectrum serves as our threshold later (for peak detection). After the midrange is obtained, spectral peaks with magnitudes greater than the threshold are searched. Every such peak and its corresponding frequency is noted and stored in an array. After the entire spectrum has been searched, the qualified frequencies are sorted based on their magnitudes. This helps in removing the characteristic sidelobe peaks of an FM modulated signal which may otherwise appear as a separate channel. Since now the frequencies at which these channels exist are known, an individual channel can be selected for demodulation. The selected channel is filtered through a bandpass filter of bandwidth 200 kHz with its center frequency being equal to the carrier frequency of the channel. As an example; 4th channel is selected (at 14 MHz). Post filtering spectrum can be observed in *Figure 2.3* below.

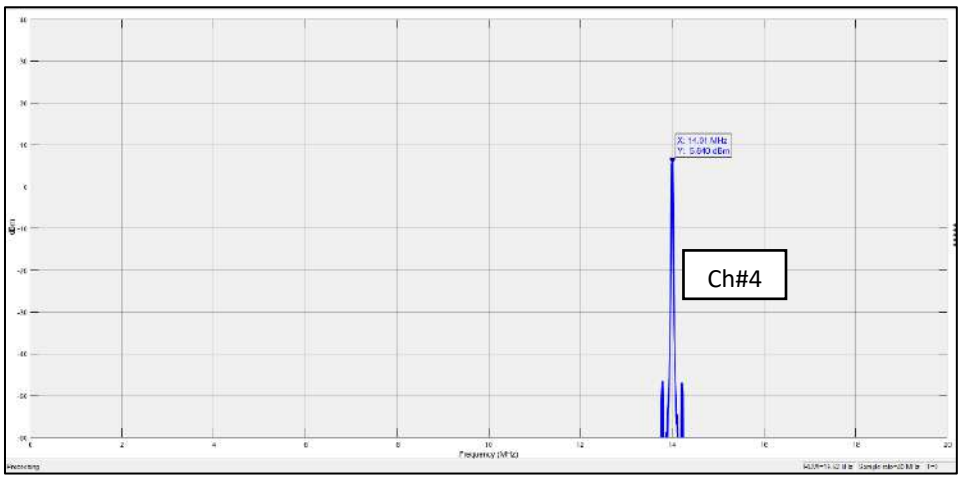


Figure 2.3: Channel 4 Filtered

Now, this signal is demodulated, and low pass filtered. A comparison of the original audio and demodulated channel can be seen in *Figure 2.4*.

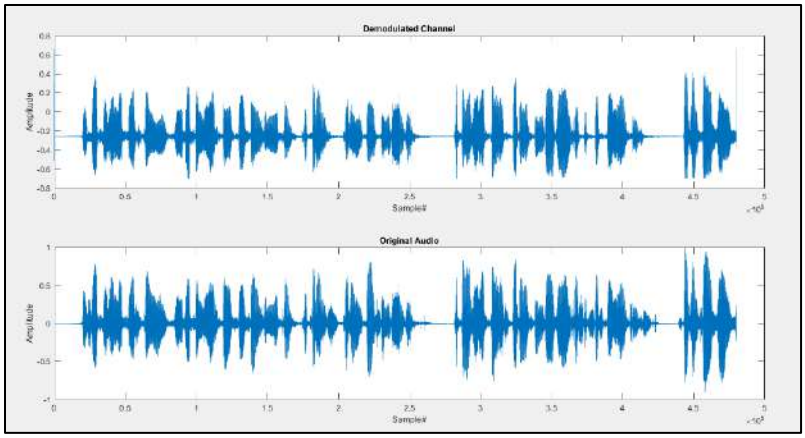


Figure 2.4: Demodulated Signal vs Original Audio

2.1.1 Conclusion of the Simulation

Based on the simulation results, it is clear that the proposed design is practically implementable. However, the influence of transmission noise, sampling noise, and resource constraints are dealt with during the implementation itself.

2.2 User Interface of the FM System

The user interface features a single control knob, an LCD screen and an Arduino to establish communication between the user and the FPGA. Options displayed on the LCD screen can me maneuvered using the control knob and selected by pushing it inwards. Every move made by the user on the user interface is confirmed to him/her by auditory feedback i.e., with the help of a buzzer (beeping patterns).

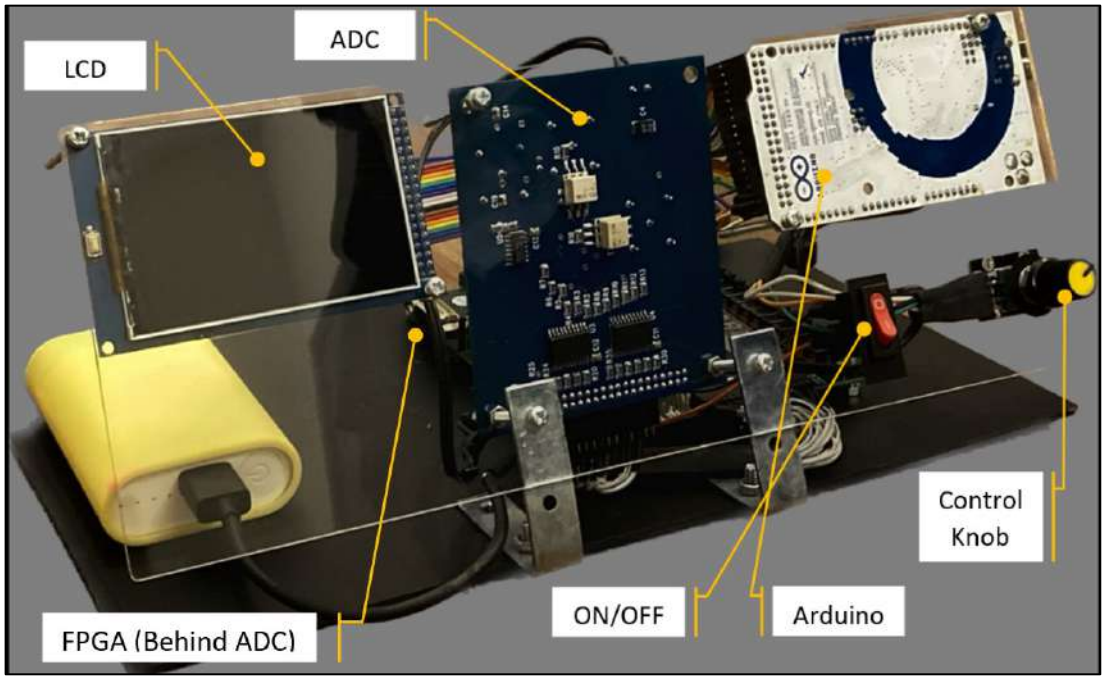


Figure 2.5: Project Hardware

2.2.1 Components of the User Interface

- Arduino Mega 2560
- 3.2-inch TFT Display (CTE32HR)
- Rotary Encoder (KY-040)
- Digilent Nexys A7 FPGA Development Board
- Buzzer

2.2.2 Communication Protocol Employed Between Arduino & FPGA

Serial Peripheral Interface (SPI) communication protocol is used to connect FPGA and the user interface. SPI is configured to run at 2Mbps with Arduino being the master, and FPGA the slave. A total of 6 wired connections exist between the two devices; MISO, MOSI, SS, SCK, VCC and GND. Due to difference in logic levels of either devices, a logic level converter is also used in between.

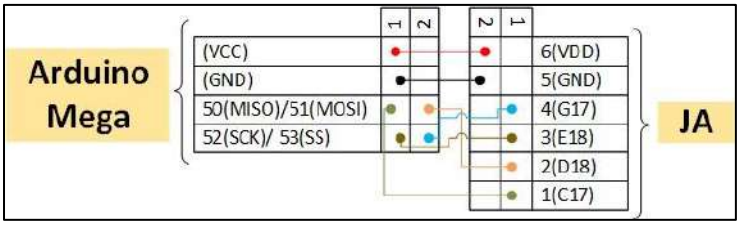


Figure 2.6: Arduino-FPGA SPI Connections

2.2.3 Working of the User Interface

Each command is assigned an 8-bit unique code. Every time the user selects an option, a unique code for that option is sent to the FPGA and it behaves correspondingly. For example; when the user selects the sense option, '1' is sent over to the FPGA and in response FPGA initiates spectrum sensing and on successful completion sends back '0' and then data transfer occurs between the Arduino and FPGA, where the information about detected channels is shared. The screen graphics are also an independent feature of Arduino, implemented through extensive code as given in Appendix B under *B2.1 Arduino*.

The screen graphics that appear on the screen are identical to what are shown in *Figure 2.7*.

Figure 2.7: GUI Example

2.3 Analog Front-end

Analog front-end is necessary to catch FM signals from the air. First, the FM band is extracted through a band-pass filter, then LNA amplifies these signals, and then these signals are down-

converted with f_{lo} = 86 MHz for the ADC. For the design of this RF front-end PathWave ADS is used. The block diagram given in ***Error! Reference source not found.*** shows the structure of the d esigned RF front-end.

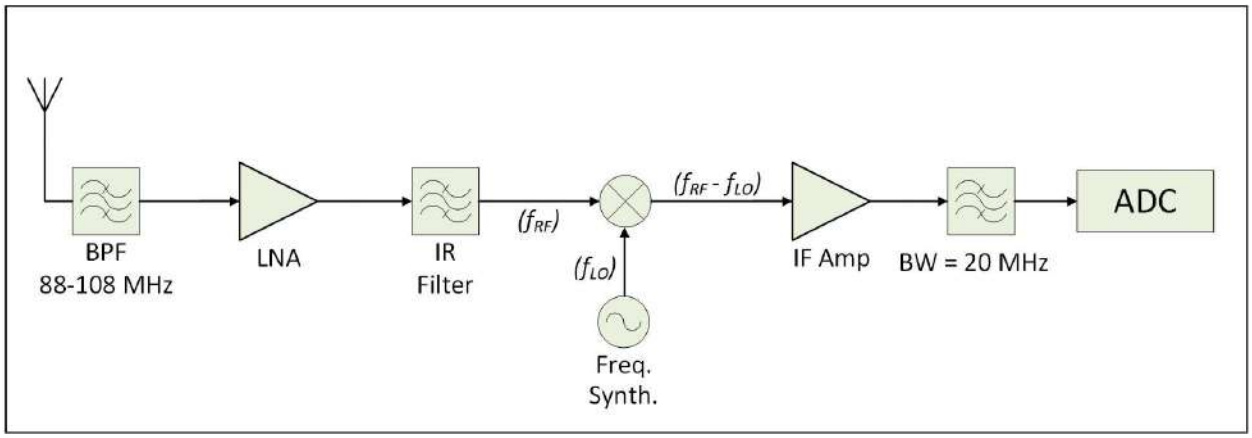


Figure 2.8: RF Front-end

2.3.1 ADS Design Process for RF Front-end

PathWave Advanced Design System (ADS) is a software tool used for designing and simulating electronic systems, including FM systems. Here is a brief overview of the process of designing an FM system using PathWave ADS:

- **Define the System Requirements:** The first step in designing an FM system is to define the system requirements. This includes determining the desired operating frequency range, the required sensitivity and selectivity, and any other performance criteria.
- **Create a Schematic:** The schematic is a graphical representation of the FM system, showing the various components and their interconnections. In PathWave ADS, the schematic can be created using the schematic editor.
- **Simulate the System:** Once the schematic is complete, the next step is to simulate the performance of the FM system. This can be done using PathWave ADS' simulation tools, which allow you to analyze the behavior of the system under different conditions.
- **Optimize the Design:** Based on the simulation results, the design can be modified and optimized to meet the desired performance criteria. This may involve adjusting component values or adding or removing components.

Overall, the process of designing an RF front end using PathWave ADS involves defining the system requirements, creating a schematic, simulating the system, optimizing the design, and fabricating and testing the system.

2.3.2 Design and Simulation of Each Stage of the Receiver

For the design of all filters, substrate specifications are given below in ***Error! Reference source n ot found.***.

Substrate:	FR-4
Thickness:	1.5 mm
Permittivity:	4.4
Loss Tangent:	0.019

Table 1: Substrate Specifications

2.3.2.1 RF Band-pass Filter (88 – 108 MHz)

After an antenna, this is the first component. It extracts the FM band which lies in the frequency range of 88 – 108 MHz, all other RF signals outside this band are attenuated as can be seen in *Figure 2.11*Figure 2.10.

- Schematic of RF Band-pass Filter

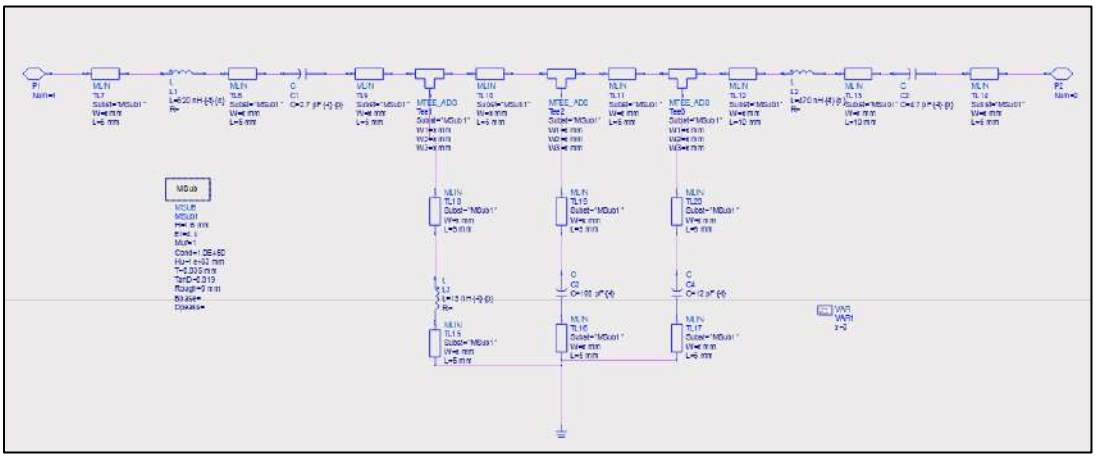


Figure 2.9: Schematic of RF Band-pass Filter

- Layout of RF Band-pass Filter

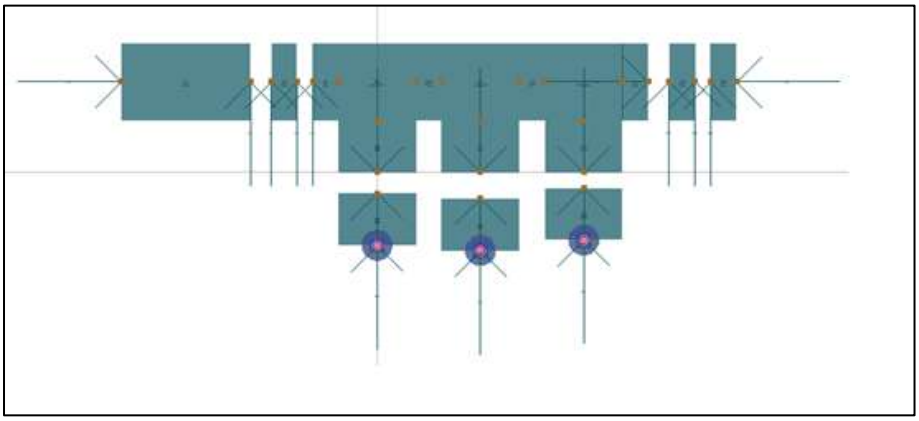


Figure 2.10: Layout for RF Band-pass Filter

- Simulation Results of RF Band-pass Filter

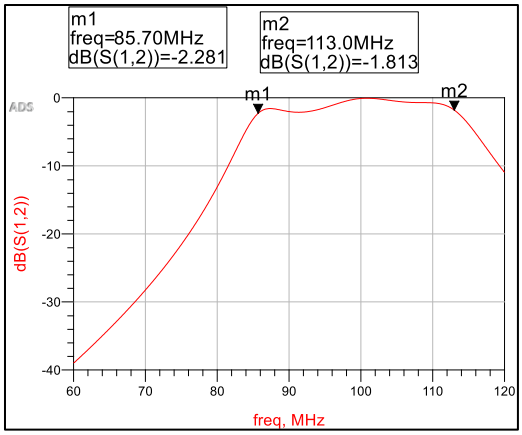


Figure 2.11: Simulation Results of RF Band-pass Filter

2.3.2.2 IR Band-pass Filter (64 – 88 MHz)

This filter serves the purpose of image rejection and filters out the unwanted signals before mixing, so that the desired band is not corrupted before reaching to the Intermediate Frequency (IF) amplifier.

- Schematic of IR Band-pass Filter

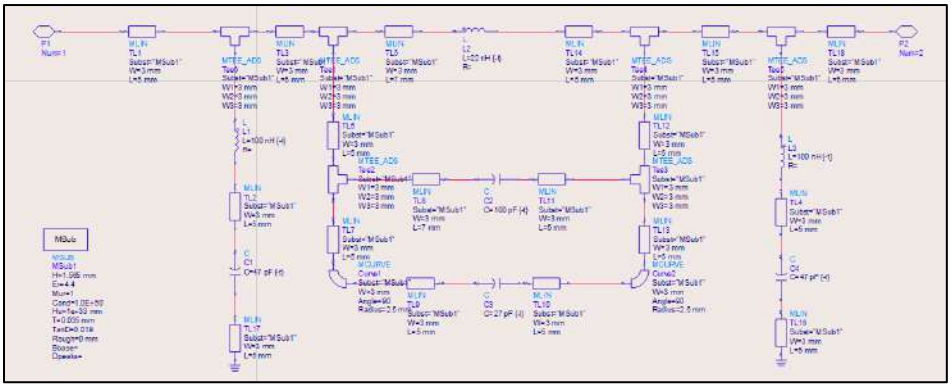


Figure 2.12: Schematic for IR Band-pass Filter

- Layout of IR Band-pass Filter

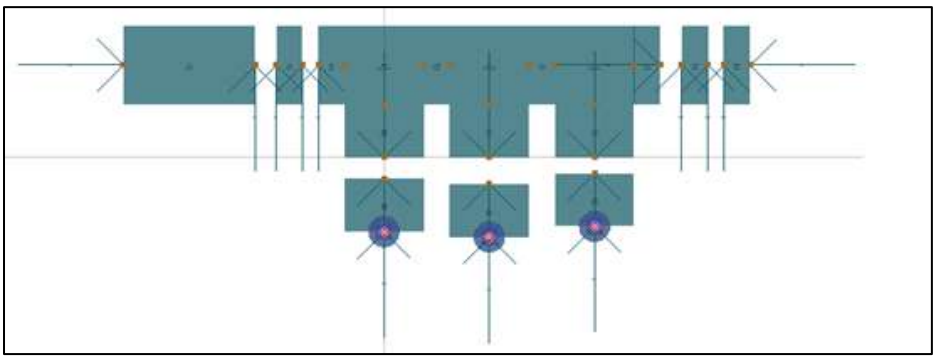


Figure 2.13: Layout for IR Band-pass Filter

- Simulation Results of IR Band-pass Filter

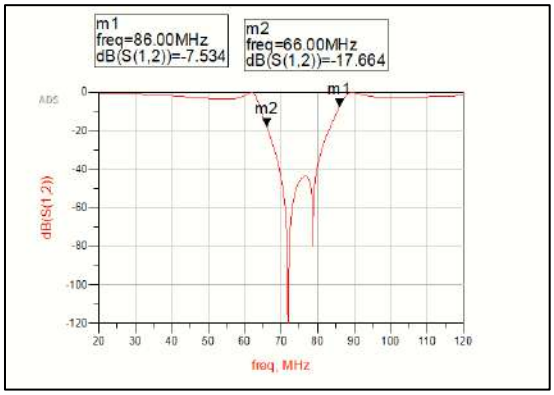


Figure 2.14: Simulation Results of IR Band-pass Filter

2.3.2.3 IF Band-pass Filter (2 – 22 MHz)

This is the final stage of filters. After mixing, two images translated at $f_{RF} + f_{Lo}$ and $f_{RF} - f_{Lo}$ are obtained. This filter extracts the latter frequency range as it is to be sampled by the ADC following the Nyquist criteria. It can be seen in *Figure 2.17* that only 2-22 MHz frequency range is allowed to pass through, to avoid aliasing.

- Schematic of IF Band-pass Filter

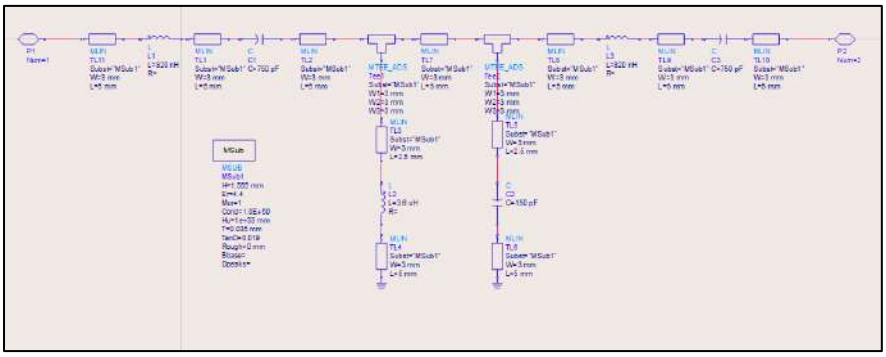


Figure 2.15: Schematic of IF Band-pass Filter

- Layout of IF Band-pass Filter

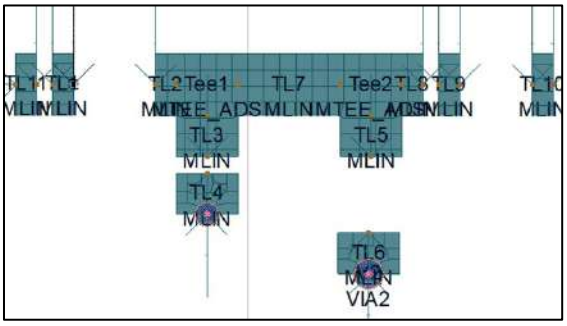


Figure 2.16: Layout of IF Band-pass Filter

- Simulation Results of IF Band-pass Filter

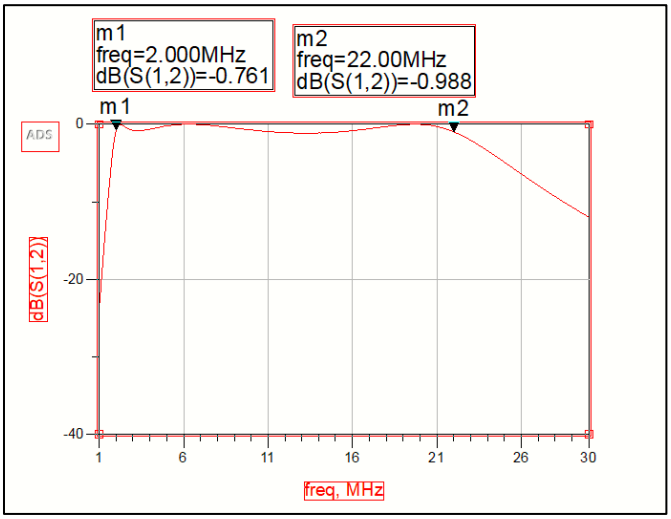


Figure 2.17: Simulation Results of IF Band-pass Filter

2.3.2.4 Mixer Design

Mixer serves the purpose of down-conversion of the FM spectrum. Port 2, receives the oscillator input i.e., 86 MHz sine wave and port 1 receives the FM signal. The mixer's simulation is shown in *Figure 2.19.* These results are computed using harmonic balance. The RF power is set to -60 dBm (96 MHz), The LO power is set to 0dBm (86 MHz) which gave power of -51 dBm at IF frequency of 10 MHz.

- Schematic of the Mixer

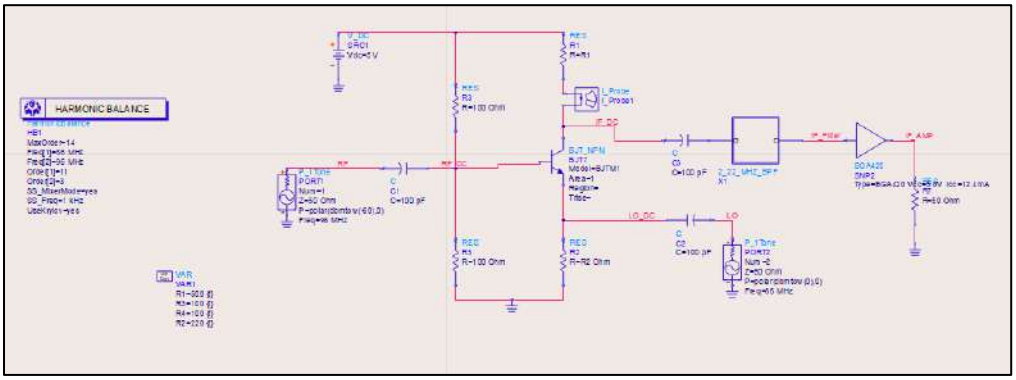


Figure 2.18: Schematic of Mixer

- Simulation Results of the Mixer

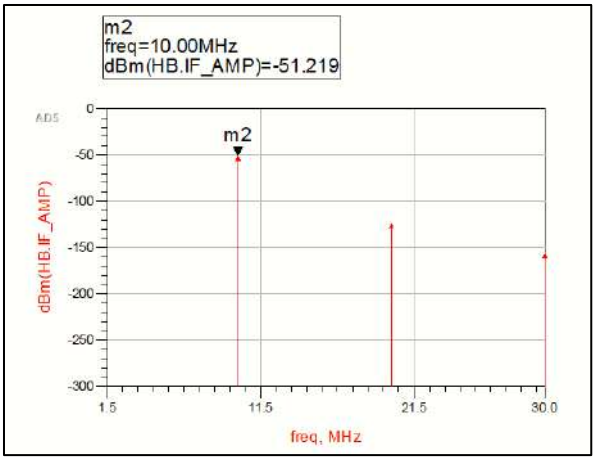


Figure 2.19: Simulation Results of Mixer

2.3.3 Design and Simulation of Complete RF Front-end

Here, the design of the complete RF front-end is presented. The simulation results can be seen in *Figure 2.22.* During the simulation, a single FM signal is assumed at 96 MHz as a test case. One can observe that this signal is down-converted to a 10 MHz FM signal by mixing it with 86 MHz sine wave. Now this signal can be easily and properly sampled by the ADC.

- Schematic of the Complete RF Front-end

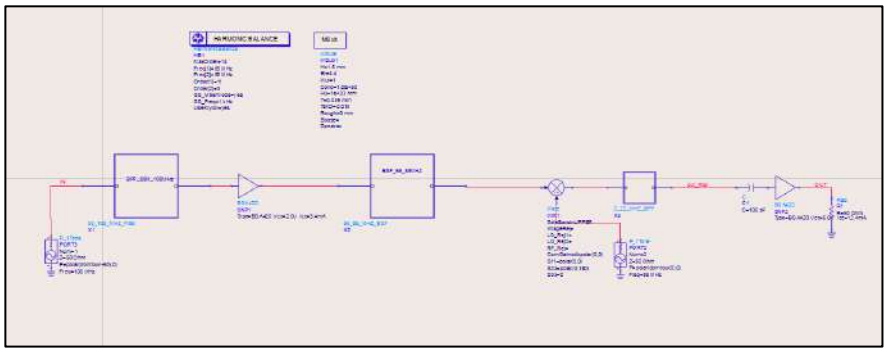


Figure 2.20: Schematic of Complete Design

- Layout of the Complete RF Front-end (120*40 mm)

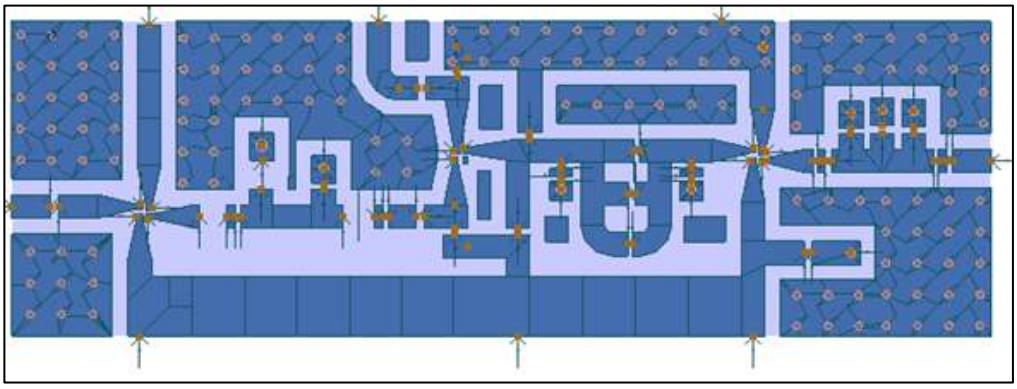


Figure 2.21: Layout of the Complete Design

- Simulation Results of the Complete RF Front-end

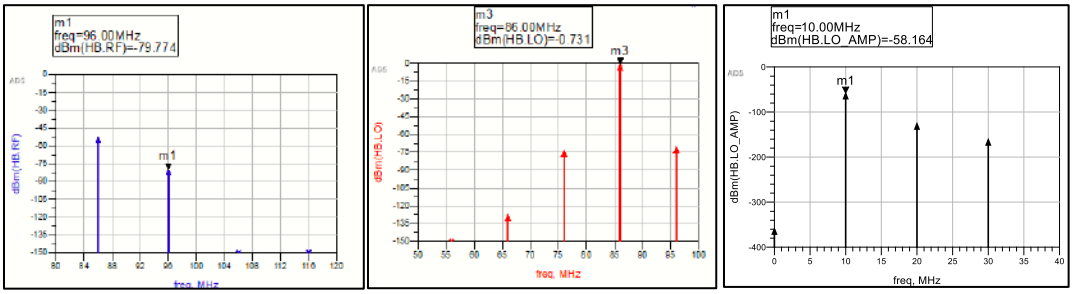


Figure 2.22: (Left to Right) Input RF Signal, Input LO (Oscillator), IF Output

2.3.4 RF Front-end Hardware

After populating the RF front-end PCB, its final form is shown in *Figure 2.23.*

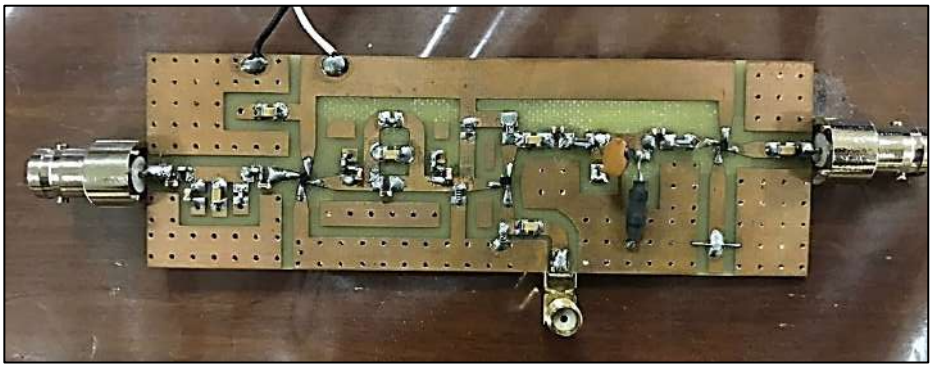


Figure 2.23: RF Front-end

2.3.5 Alternative Approach to RF Front-end

Due to the unavailability of a component; an amplifier IC, we couldn't complete the Analog front-end and it remained nonfunctional. However, an alternate approach is adopted to generate our own FM transmission that mimics the FM transmission received using an antenna. For this purpose, Keysight N9310A RF signal generator is used.

Figure 2.24: Keysight N9310a

2.4 ADC Interfacing

2.4.1 Introduction to ADC Interfacing

ADC converts the analog FM signal into digital data. The ADC used in this project is AD6640 by Analog Devices. It has a maximum sampling rate of 65 MSPS and 12-bit resolution. ADC is connected to the FPGA through 12 data wires and 6 ground wires. Further details about driving the ADC are given below.

2.4.2 Connections (ADC - FPGA)

The ADC is connected to the FPGA using JD and JC PMOD ports. Multiple grounds are connected between the both devices for improved data transmission. The wiring diagram is shown in *Figure 2.25.*

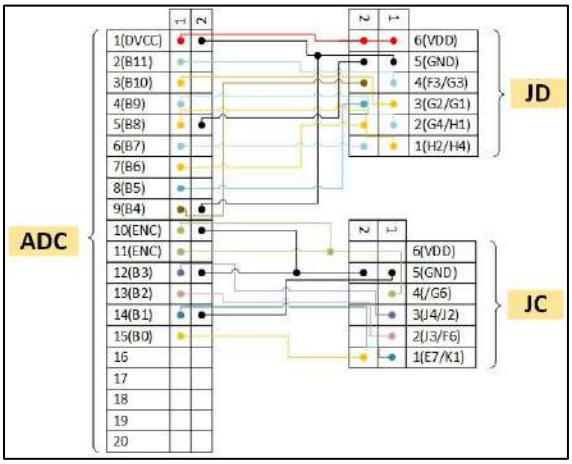


Figure 2.25: ADC-FPGA Connections

2.4.3 ADC Trigger Signal "ENC"

The ADC requires a trigger signal to initiate a sample, it's called "encode (ENC)". On every rising edge of this signal, the ADC triggers conversion of the sample. ENC signal must obey TTL logic levels and both LOW and HIGH pulse widths should not fall short of 6.5 ns. The typical output delay is 10.5 ns. Keeping these specifications in mind the ADC and FPGA were synchronized to avoid metastability of data. This ENC signal is supplied to the ADC by the FPGA. *Figure 2.26* shows the timing diagram of the ADC.

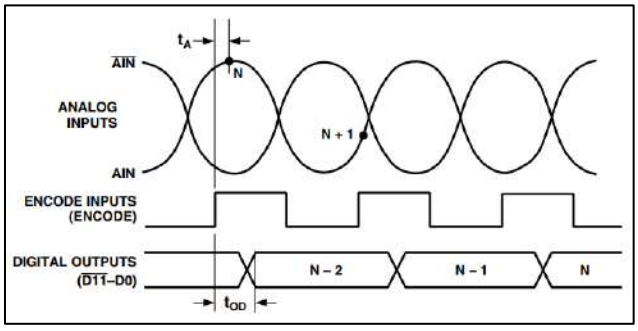


Figure 2.26: ADC Timing (Retrieved From: AD6640 Reference Manual)

Given below in *Figure 2.27* are the waveform of the ENC signal as received by the ADC and the corresponding reconstructed data. On the left is shown a 10 MHz encode signal and the corresponding reconstructed analog signal using it. On the right is shown a 60 MHz encode signal and the reconstructed analog signal using it.

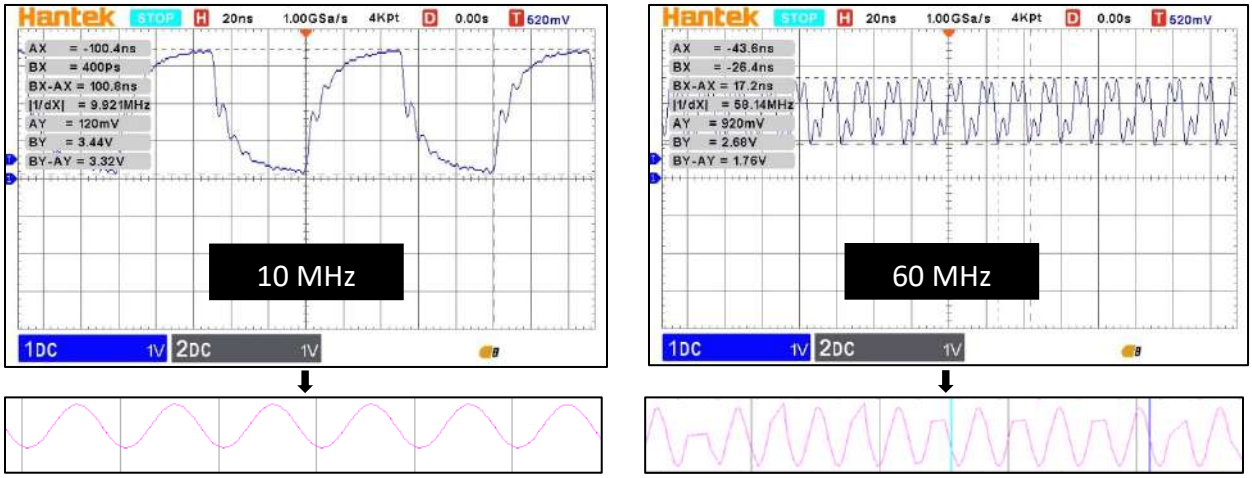


Figure 2.27: ENC Signal and Corresponding ADC Output

It can be observed that the 60 MHz signal fails to follow the TTL logic levels and the minimum pulse width specifications. This is due to attenuation in the probe used to transmit the ENC signal. So, temporarily 10 MHz ENC signal is being used to drive the ADC.

2.5 Spectrum Sensing

2.5.1 Introduction to Spectrum Sensing

How does one find out that at which frequencies are the FM channels to be found? For that purpose, spectrum sensing is performed. FFT algorithm is employed to compute 2048-point Discrete Fourier Transform. Further, the magnitude spectrum is calculated and its maximum and minimum values are found. These two values are used in computing the midrange of the magnitude spectrum, and then this midrange is used as threshold to classify frequencies as useful FM signals or noise. Since, FM signals have characteristic sidelobes which may mistakenly be identified as separate channels, the qualified frequencies are sorted in order of greater magnitude to get rid of these sidelobes.

2.5.2 Implementation of Spectrum Sensing

For computation of Discrete Fourier Transform, LogiCore FFT IP is used. This IP is configured to perform 2048-point FFT. For better performance; fixed point format and burst I/O format is used.

With these settings the latency of FFT output is 1555 µs. So, approximately after 1600 µs, a valid spectrum is completely obtained. After that, the magnitude spectrum is computed using the formula given in equation *(2.1)*.

$$magnitude = \sqrt{(real^2 + img^2)}$$

(2.1)

In order to implement square root on an FPGA, we used CORDIC algorithm. This is facilitated by the LogiCore CORDIC IP. The magnitude spectrum is stored in the memory and midrange is computed using the formula given in equation *(2.2)*.

$$midrange = (\max(spect) + \min(spect))/2$$

(2.2)

After the midrange is obtained, search for spectral peaks with magnitudes above the threshold is started. Every such peak and its corresponding frequency is noted and stored. After the entire magnitude spectrum has been searched, the qualified frequencies are sorted based on their magnitudes. This helps in removing the characteristic sidelobe peaks of an FM modulated signal which may otherwise appear as a separate channel. In the current design, only 8 of the strongest channels will qualify for demodulation. The Verilog code and other modules used for spectrum sensing are given under *Appendix B.*

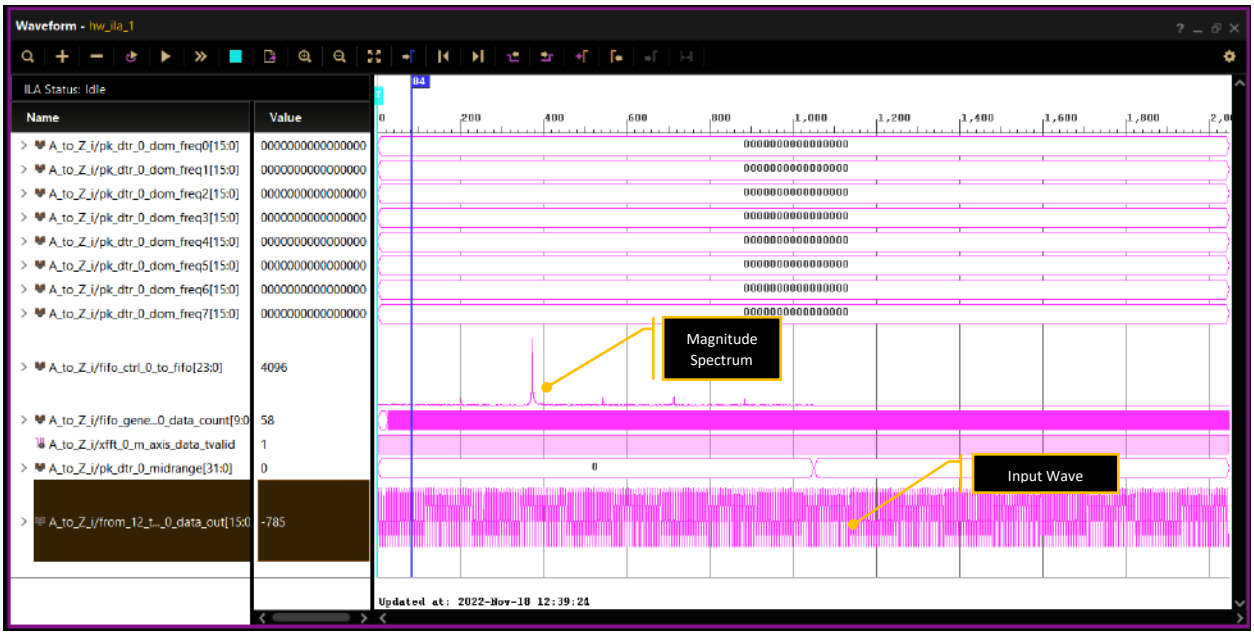


Figure 2.28: Real-Time ILA Capture of Magnitude Spectrum

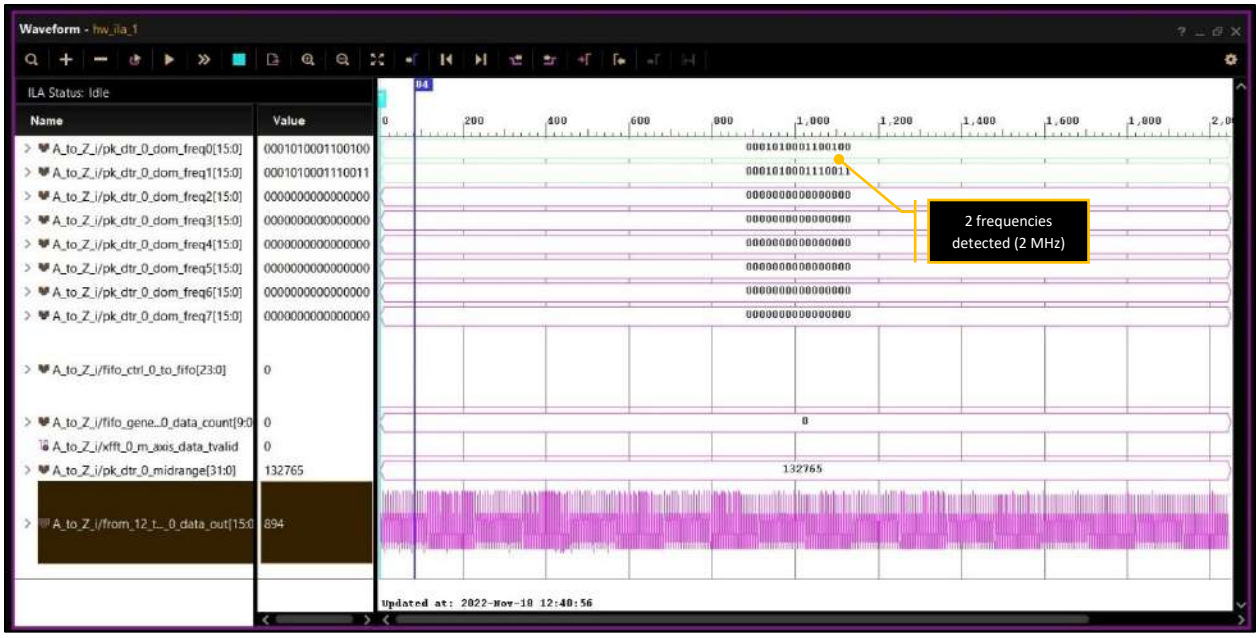


Figure 2.29: Detected Frequencies in Fix16_11 Format

2.6 Filtering

2.6.1 Introduction to Digital Filtering

Digital filtering is a fundamental operation in DSP (Digital Signal Processing). It involves processing a digital signal to modify or enhance certain features or characteristics of the signal. Digital filters are used in various applications, such as audio enhancement, image processing, communications, and control systems.

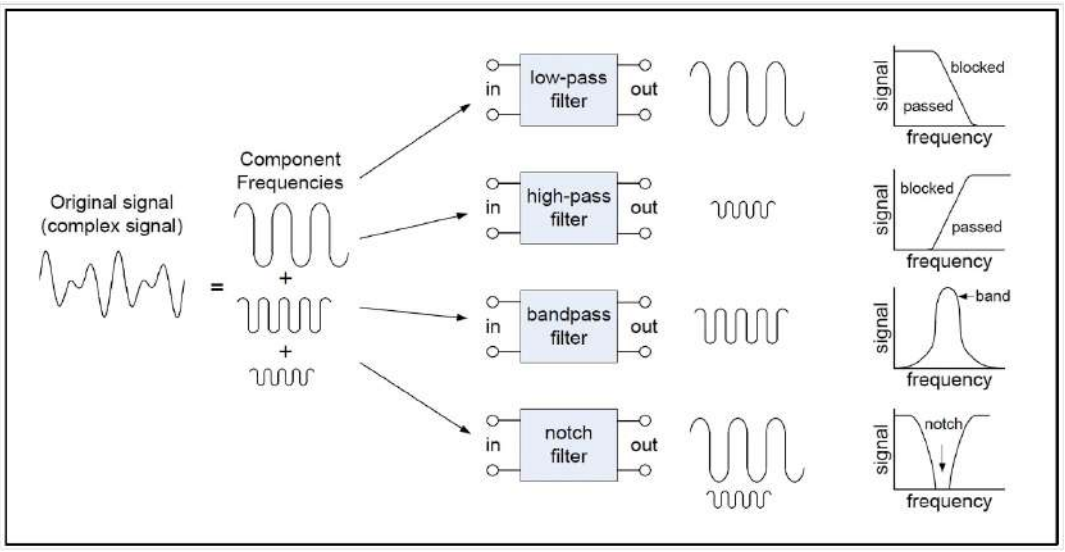


Figure 2.30: Basic Depiction of Major Types of Filters

Digital filters can be segregated into two main types based on their impulse response; Finite Impulse Response (FIR) filters and Infinite Impulse Response (IIR) filters. FIR filters have a fixed,

finite number of taps, the coefficients determining the filter's response to an input signal. These taps are fixed and do not change over time. IIR filters, on the other hand, have an infinite number of taps, and the tap weights are updated at each sample time based on the input signal. One important property of digital filters is their frequency response, which describes how they respond to different frequencies in the input signal. Filters can be designed to have a variety of frequency responses, such as low-pass, high-pass, band-pass, or band-stop.

Digital filters are essential in DSP because they allow us to modify certain digital signal features selectively. For example, one can remove noise from an audio signal or enhance the edges of an image. Digital filters can also extract meaningful information from a signal, such as extracting the frequency components of an audio signal using a Fourier transform.

2.6.2 Adopted Filtering Scheme

2.6.2.1 FIR Filters

The technique used for designing the proposed filter banks is FIR filtering. FIR filters are a type of digital filter that have a fixed, finite number of taps, which's coefficients determine the filter's response to an input signal. These taps are fixed and do not change over time. The impulse response of a filter is the output of the filter when the input is a unit impulse. An FIR filter has a finite impulse response because it only depends on a finite number of past samples of the input signal. This means that the output of an FIR filter only depends on the current and past input samples, and not on the previous output samples. They can be designed to have a linear phase response, which means that the phase response is a linear function of the frequency. This is useful in applications where phase distortion is not acceptable, such as in audio processing.

2.6.2.2 Why Use FIR Filters

There are several advantages of FIR filters compared to IIR filters:

Linear Phase Response: As mentioned earlier, FIR filters can be designed to have a linear phase response, which is not possible with IIR filters. This makes FIR filters suitable for applications where phase distortion is not acceptable, such as in audio processing.

Stability: FIR filters are generally more stable than IIR filters. They do not have feedback, which means that they do not amplify noise or oscillate. This makes them suitable for applications where stability is important, such as in control systems.

Robustness: FIR filters are generally more robust than IIR filters. They are not sensitive to coefficient quantization errors and are not prone to round-off errors. This makes them suitable for applications where the filter coefficients must be implemented with low precision.

FIR filters come with their own disadvantages,

Length: FIR filters generally have a longer impulse response and a larger number of taps compared to IIR filters. This makes them more computationally intensive and requires more memory to implement.

Delay: FIR filters generally have a longer delay compared to IIR filters. This means that the output of an FIR filter lags behind the input by a longer time.

Non-causal: Some FIR filters are non-causal, which means that the output depends on future input samples. This is not practical in most applications, so causal FIR filters must be used, which results in a longer delay.

Conclusively, the fact that FIR filtering provided the most stable and undeterred output, they were opted. The usage of IIR filtering, upon testing generated outputs that were highly affected by noise and other interferences.

2.6.3 Filter Banks for FM system

2.6.3.1 Architecture

- Two Filter Banks are used. One to extract out channels and the other one to extract mono audio
- All filters work on down converted frequencies; $F_D = F_{center} - F_{LO}$. Where F_{LO} in this case is 86MHz.
- Sampling frequencies for filters is 48MHz according to Nyquist standard.

Channel Filter Bank Characteristics:

- Parallelly operating Bandpass filters as they are fed with same input
- These filters have a bandwidth of 200kHz according to the bandwidth of an FM channel
- Orders in the range of < 1000
- Each channel filter has a cascaded single-input-single-output mono filter

Mono Filter Bank Characteristics:

- The mono filter bank comprises of lowpass filters
- These filters have bandwidth of 22kHz as mono audio is present on 15kHz bandwidth
- Mono filters have orders in the range of < 100

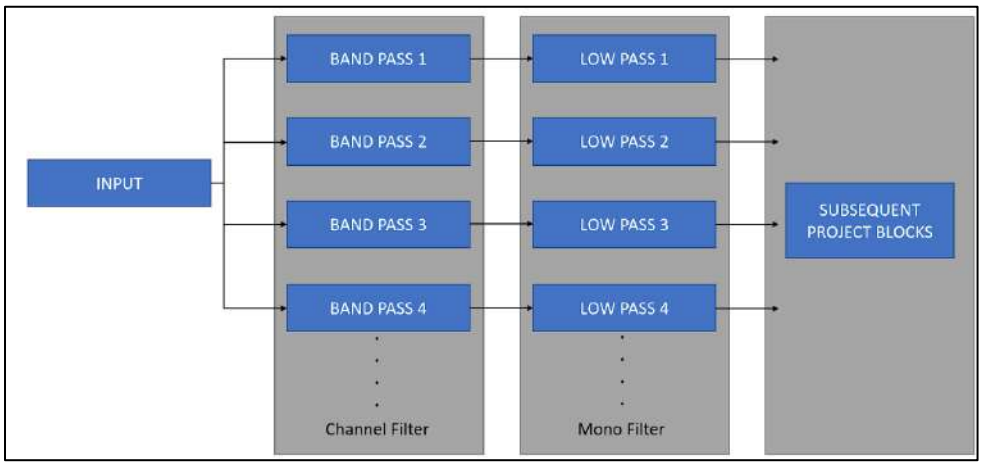


Figure 2.31: Filter Bank Architecture

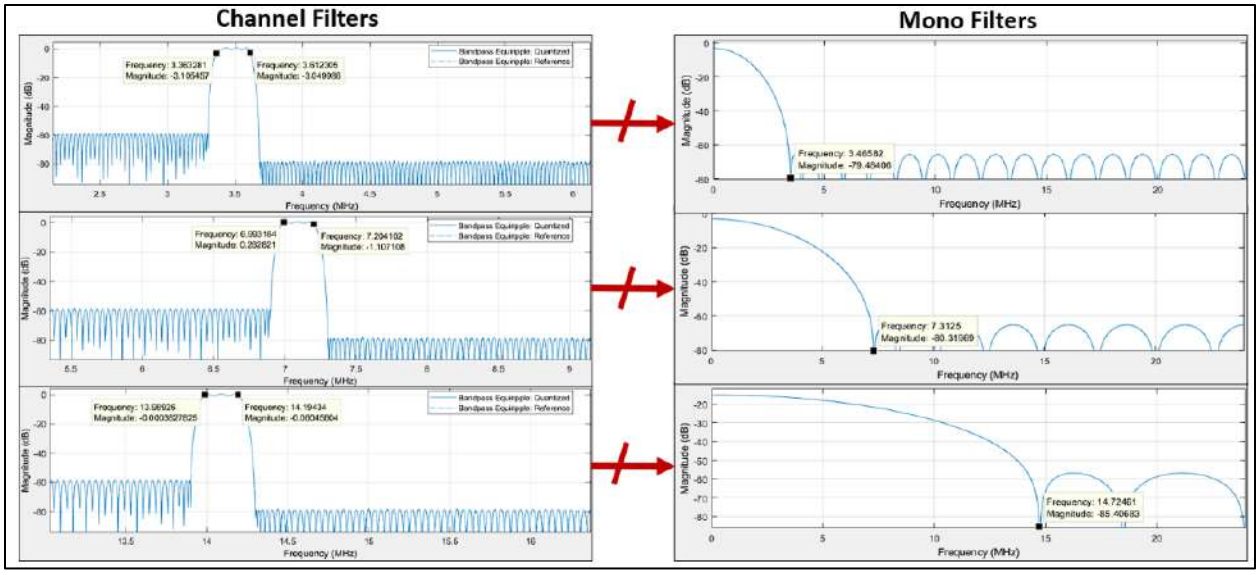


Figure 2.32: MATLAB Results for Filtering.

2.6.4 Design Methodology of Filter Banks

For designing the proposed filters and creating their HDL counterparts, MATLAB's filter designer tool is used. Here are the steps one can follow.

1) Open MATLAB and navigate to the "Filter Designer" tool by going to the "Apps" tab in the toolstrip and selecting "Filter Designer".
2) In the Filter Designer tool, click on the "New Filter" button to create a new filter design.
3) In the "Filter Design Method" section, select "Finite Impulse Response (FIR)" as the type of filter you want to design
4) In the "Filter Specifications" section, specify the desired filter characteristics, such as the filter order, passband frequency, and stopband frequency
5) Click on the "Design Filter" button to generate the FIR filter coefficients
6) To generate the HDL code for the filter, click on the "Generate HDL" button in the toolstrip. This will open the "HDL Code Generation" dialog box.
7) In the "HDL Code Generation" dialog box, specify the desired parameters for the HDL code generation, such as the target hardware and the desired optimization level.
8) Click on the "Generate" button to generate the HDL code for the filter. The generated code will be displayed in the editor window.

2.6.5 Integration to Form Banks

The designed bandpass filters were integrated in form of banks using AMD Xilinx VIVADO. The integration is done by using Verilog code that fed the same input to all filters. The top module instantiated all filters and provided them with system clock, and input. The output from these filters is fed to corresponding low pass filter to extract out mono audio.

The entire VIVADO project contained HDL files of all filters and a constraint file for testing on FPGA. 16 pins were defined to get 16-bit output from the ADC. The output is observed using Integrated Logic Analyzer.

2.6.6 FPGA Implementation of Filter Banks

2.6.6.1 Issues in Implementing Filter Banks

Lowpass filter bank is successfully implemented but the issue lies with bandpass filtering. The issue is caused by the roll-off of bandpass filters which is in the range of 0.5MHz to 1MHz. This roll-off is necessary for proper channel segregation. Consequently, the order of filters exceeds 1500 and our FPGA is not resourceful enough to handle even one such filter bandpass filter let alone a whole filter bank.

Upon investigating, it was found that this FPGA has a very small number of DSP splices. The number of DSP splices to be precise is 284. The second issue is with the limited number of CARRY-4 cells. These cells are responsible for producing inputs and outputs at high frequencies. The evidence from VIVADO simulator is attached as follows.

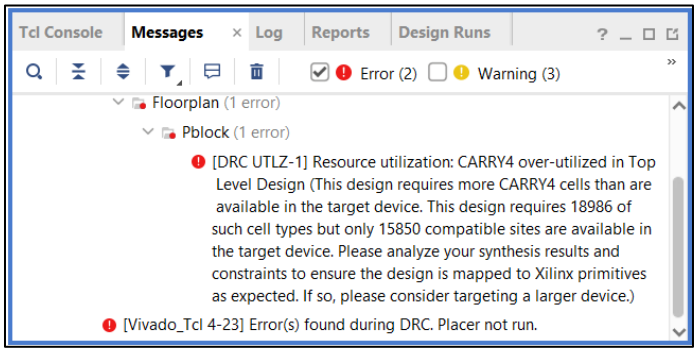


Figure 2.33: VIVADO Simulator Error States "Use a Larger Device".

Following are the utilization reports of this single filter project.

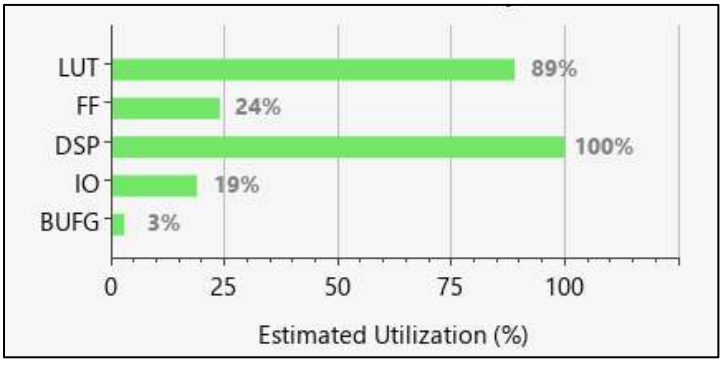


Figure 2.34: VIVADO Utilization Chart of a Single Most Optimized Filter.

The following table shows how the single bandpass filter exhausted the total number of DSP splices. The FPGA could not burn the code as its resources were over utilized.

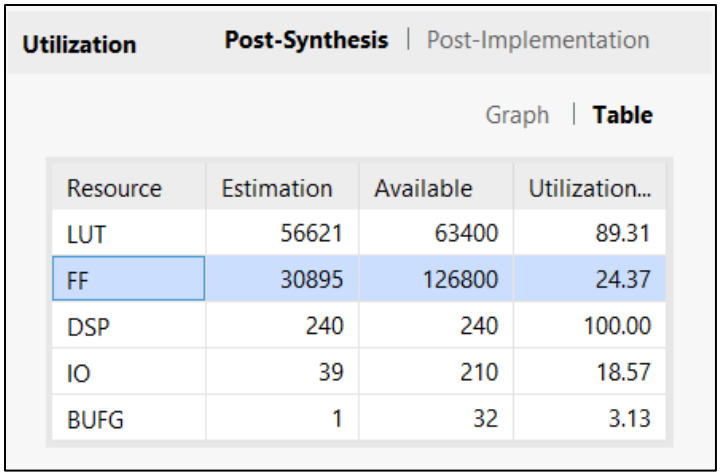


Figure 2.35: VIVADO Utilization Table of a Single Most Optimized Filter.

2.6.6.2 Optimization Techniques for Filters

Since, the FPGA was unable to burn the RTL of our filter banks, an attempt to optimize resource usage was made. For that different approaches were tried, and the following table shows the approached and their results.

APPROACHES	RESULTS
IIR filtering	Highly unstable, poor reconstruction, doesn't minimize resources at all. The order reduces but computational complexity increases due to other parameters.
Optimizing for HDL	MATLAB gives options to optimize codes by reducing the number of repetitive jobs. Only reduced the usage of LUT tables by 11%.
Adding pipelining registers	This option reduced number of adders by adding pipelining registers for reducing the number of executable code-chunks. No apparent effect on resources.
Designing filter using VIVADO's FIR Compiler	Instead of using MATLAB for designing filters, VIVADO's own IP to generate FIR bandpass filter. VIVADO's FIR compiler makes use of a coefficient file to design filters. Coefficient file was generated via MATLAB and was fed into the IP, but it resulted in the most unoptimized and basic version of the filter.

Above mentioned approaches were tried but there was no significant achievement. So, the filters had to be reduced in number along with their sampling frequency.

2.6.7 Alternative Approach to Resource Expensive Filters

At high sampling frequencies the filter order increases and thus becomes resource expensive. By decreasing the system frequency to just 10 MHz, the resource utilization was relaxed to a great

extent. Given below is the frequency response of the filter designed for a sampling frequency of 10 MHz.

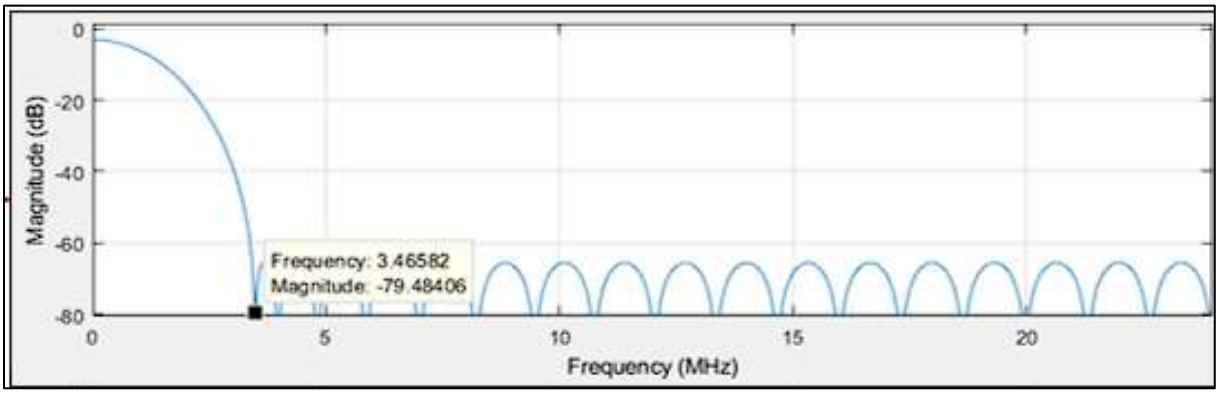


Figure 2.36: 10 MSPS Filter to Filter 1 MHz Signal

2.7 Demodulation

2.7.1 Introduction to Demodulation

For a FM signal, the data is encoded in frequency variations. The process of retrieving this data is called demodulation. Several techniques are available for demodulation, for example PLL based or quadrature demodulation. In this design, quadrature demodulation is implemented due to its versatility. Quadrature demodulation is commonly used in cases where the transmitter and receiver are not synchronized. During IQ demodulation, when two reference signals separated by 90° of phase are multiplied with the input signal, the increasing amplitude of one multiplier compensates for the decreasing amplitude of the other multiplier. For an IQ demodulator, the worst-case phase difference is 45°. A 45° phase difference does not result in a significant reduction in the amplitude of the demodulated signal and thus provides a reasonable output.

Figure 2.37 shows the block diagram of implemented design. The function of each block is explained in the subsequent sections.

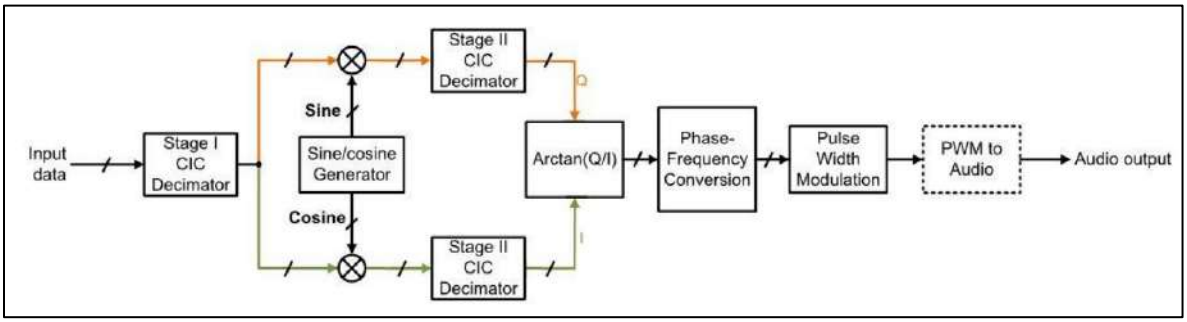


Figure 2.37: IQ Demodulator. Retrieved From [4].

2.7.2 Working of the Demodulator

2.7.2.1 Decimators

Decimation is defined as the process of down-sampling in combination with low-pass filtering. In this project Cascaded Integrator-Comb (CIC) decimators are used. CIC filters, also known as Hogenauer filters, are typically employed in applications which have a large excess sample rate compared to the signal bandwidth. CIC filters are implemented using adders, subtracters, and delay elements only. Due to their lightweight structures, CIC filters are preferred for hardware-efficient implementations of filtering. In our implementation, stage-1 decimator can be skipped since our sample rate is not much higher than the required Nyquist criterion. To perform Decimation LogiCore CIC Compiler IP is used. The stage-2 decimator is used to extract the down converted I & Q components of the input data.

2.7.2.2 Sine/Cosine Generator

The sine/cosine generator block is the digital equivalent of a local oscillator. To generate sine/cosine wave, LogiCore DDS IP is used. This IP exploits the quarter wave symmetry of sine/ cosine wave to implement a full cycle, thus optimizing the LUT usage. The purpose of this block is to down convert the data and to obtain the quadrature components.

2.7.2.3 Arctan

This block is used to find the phase between the I & Q components of the data. Using CORDIC algorithm, arctan is computed to find the phase information from the I and Q components. The phase information is then used to obtain the frequency variation in the signal. LogiCore CORDIC IP is used to compute the arctan.

2.7.2.4 Phase-Frequency Conversion

To find the frequency variation in the signal, difference relationship between frequency and phase is exploited as explained by equations *(2.3)* and *(2.4).*

$$\omega = \frac{\mathrm{d}\phi}{\mathrm{d}t} \approx \frac{\Delta\phi}{\Delta t} \tag{2.3}$$

$$\omega = \phi_{i+1} - \phi_i \tag{2.4}$$

As, for a FM signal the information is stored in the frequency variations, the change in frequency can be obtained by comparing two consecutive phase samples as explained by equation *(2.4)*.

2.8 PWM to Audio Conversion

2.8.1 Introduction to PWM to Audio Conversion

Pulse Width Modulation (PWM) is a method of reducing the average power delivered by an electrical signal, by effectively chopping it up into discrete parts. This technique is used to replace

the need of an extra peripheral i.e., DAC. PWM with a reconstruction (LPF) filter can be used to produce reasonably good analog waves especially below 100 kHz. Since the requirement is just to produce an audio signal (max 20 kHz), it works just fine.

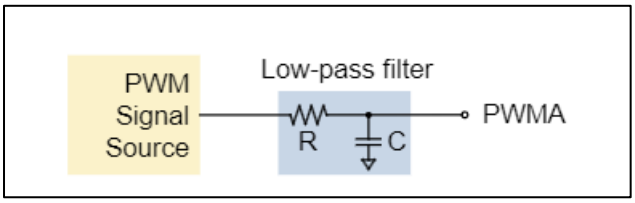


Figure 2.38: PWM to Analog Conversion. Retrieved From [27]

2.8.2 Working of PWM to Audio Conversion

The PWM Block is actually a first-order sigma-delta modulator which can be implemented with a hardware accumulator. Every time the accumulator overflows, output a '1'. Otherwise output a '0'. That's very easily done in an FPGA. Given Below is the Verilog code for 1st order sigma delta modulator.

```
module PWM(clk, PWM_in, PWM_out);
input clk;
input [7:0] PWM_in;
output PWM_out;

reg [8:0] PWM_accumulator;
always @(posedge clk) PWM_accumulator <= PWM_accumulator[7:0] + PWM_in;

assign PWM_out = PWM_accumulator[8];
endmodule
```

The higher the input value, the faster the accumulator overflows "PWM_accumulator[8]", and the more frequent are the output "1"s. The PWM output is connected to the input pin of the reconstruction filter as shown in *Figure 2.38.* The output of this reconstruction filter is the corresponding analog signal as shown in *Figure 2.39*.

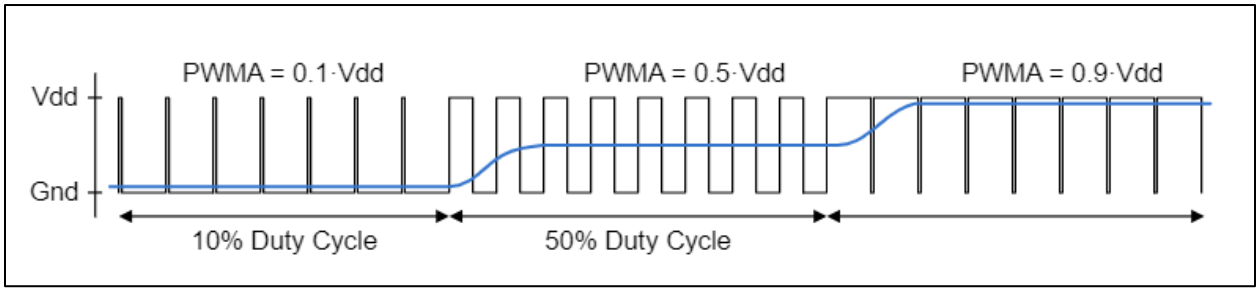


Figure 2.39: Analog Equivalent Signal of PWM After Filtering. Retrieved From [27]

3 FYP Deliverables & Timeline

3.1 Deliverables

3.1.1 FYP-1

- Complete Design and Implementation of the FFT block.
- Windowing operation and averaging of Fourier transform for accurate channel detection.
- Integration of peak detector for detecting the exact number of channels.
- FYP-1 Report Submission.

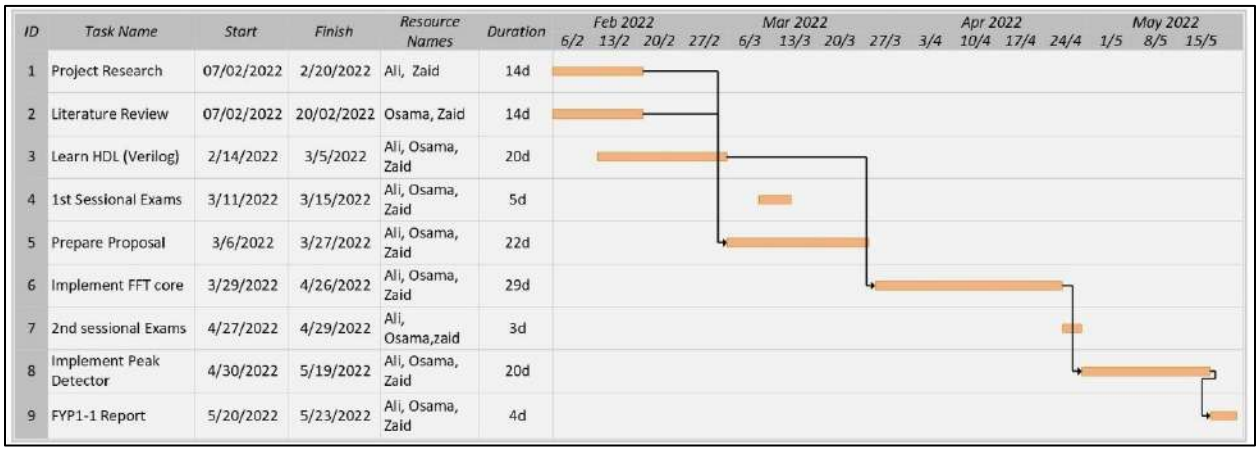
3.1.2 FYP-2:

- Complete Design and Implementation of Filter banks and integration with proposed system.
- Complete Implementation of Demodulator and integration with proposed system.
- Integration of Memory.
- Displaying active channels on a screen and playback on speaker.
- FYP Report submission.

3.2 Timelines

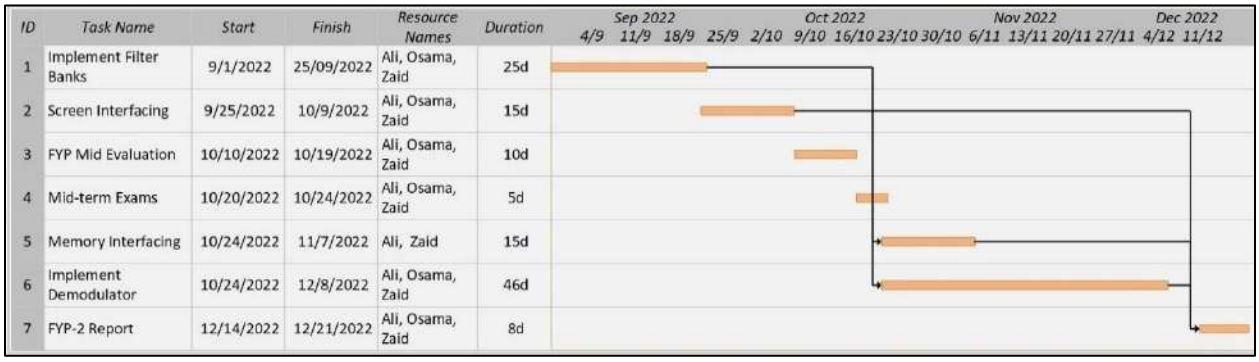
3.2.1 FYP-1 Timeline

ID	Task Name	Start	Finish	Resource Names	Duration
1	Project Research	07/02/2022	2/20/2022	Ali, Zaid	14d
2	Literature Review	07/02/2022	20/02/2022	Osama, Zaid	14d
3	Learn HDL (Verilog)	2/14/2022	3/5/2022	Ali, Osama, Zaid	20d
4	1st Sessional Exams	3/11/2022	3/15/2022	Ali, Osama, Zaid	5d
5	Prepare Proposal	3/6/2022	3/27/2022	Ali, Osama, Zaid	22d
6	Implement FFT core	3/29/2022	4/26/2022	Ali, Osama, Zaid	29d
7	2nd sessional Exams	4/27/2022	4/29/2022	Ali, Osama,zaid	3d
8	Implement Peak Detector	4/30/2022	5/19/2022	Ali, Osama, Zaid	20d
9	FYP1-1 Report	5/20/2022	5/23/2022	Ali, Osama, Zaid	4d



3.2.2 FYP-2 Timeline

ID	Task Name	Start	Finish	Resource Names	Duration
1	Implement Filter Banks	9/1/2022	25/09/2022	Ali, Osama, Zaid	25d
2	Screen Interfacing	9/25/2022	10/9/2022	Ali, Osama, Zaid	15d
3	FYP Mid Evaluation	10/10/2022	10/19/2022	Ali, Osama, Zaid	10d
4	Mid-term Exams	10/20/2022	10/24/2022	Ali, Osama, Zaid	5d
5	Memory Interfacing	10/24/2022	11/7/2022	Ali, Zaid	15d
6	Implement Demodulator	10/24/2022	12/8/2022	Ali, Osama, Zaid	46d
7	FYP-2 Report	12/14/2022	12/21/2022	Ali, Osama, Zaid	8d



4 Conclusion

4.1 About the Project

In this project we have implemented a digital, FPGA based FM receiver which can automatically detect, demodulate and record FM channels in a particular area for security monitoring. The motivation behind this project is to make possible the monitoring (recording) of multiple channels, simultaneously. At the time of writing this report, the implemented design is 70% functional with demodulator being out of action. However, the system is able to detect FM channels and filter them. Moreover, due to FPGA's resource constraints, the system is set to operate at 10 MHz instead of the proposed 60 MHz system frequency. The FPGA's resource utilization for the complete project is shown in *Figure 4.1.*

Resource	Utilization	Available	Utilization %
LUT	8049	63400	12.70
LUTRAM	640	19000	3.37
FF	7748	126800	6.11
BRAM	17.50	135	12.96
DSP	17	240	7.08
IO	29	210	13.81
BUFG	5	32	15.63
MMCM	1	6	16.67

Figure 4.1: Complete Project's Resource Utilization

4.2 What we Learned

This project has been instrumental in learning about FPGA based design. During this project we had the opportunity to practice HDL design, FPGA implementation of DSP systems, RF PCB design, interfacing Xilinx's products, multi-modular design and MATLAB prototyping. We think that this valuable experience will help us throughout in our academic and professional careers.

Appendix A: Glossary

ADC	Analog to Digital Converter
AM	Amplitude Modulation
BPF	Band Pass Filter
BUFG	Global Buffer
CIC	Cascaded Integrator Comb
CORDIC	Coordinate Rotation Digital Computer
DAC	Digital to Analog Converter
DDS	Digital Direct Synthesis
FIR	Finite Impulse Response
FM	Frequency Modulation
FPGA	Field Programmable Gate Array
GSPS	Giga Samples per Second
HDL	Hardware Descriptive Language
IF	Intermediate Frequency
IIR	Infinite Impulse Response
ILA	Integrated Logic Analyzer
IP	Intellectual Property
IQ	In-phase & Quadrature
IR	Image Rejection
LCD	Liquid Crystal Display
LNA	Low Noise Amplifier
LPF	Low Pass Filter
LUT	Look-up Table
MISO	Mater In Slave Out
MMCM	Mixed Mode Clock Manager
MOSI	Mater Out Slave In
MSPS	Mega Samples per Second
PWM	Pulse Width Modulation
SPI	Serial Peripheral Interface

Appendix B: Codes

B1: MATLAB Simulation

```
clc; clear all; close all;
%%                                  Importing audios----(Transmitter)
Fs = 48000; %The sampling freq of audio.
samples = [1, 10*Fs]; %Just 10 seconds of audios are imported by defining samples.
[CH1,Fs] = audioread('audio1.wav', samples);
[CH2,Fs] = audioread('audio2.wav', samples);
[CH3,Fs] = audioread('audio3.wav', samples);
[CH4,Fs] = audioread('audio4.wav', samples); %For low power channel
disp('All audios imported correctly.');
%sound(CH1, Fs);%Uncomment to test the audio.
%%                                  Band limiting & scaling the audios----(Transmitter)
Fs = 48000; %The sampling freq of audio.
CH1_BL= lowpass(CH1,15e3,Fs); CH1_BLS = CH1_BL/max(CH1_BL);
CH2_BL= lowpass(CH2,15e3,Fs); CH2_BLS = CH2_BL/max(CH2_BL);
CH3_BL= lowpass(CH3,15e3,Fs); CH3_BLS = CH3_BL/max(CH3_BL);
CH4_BL= lowpass(CH4,15e3,Fs); CH4_BLS = CH4_BL/max(CH4_BL);
n = 1:480000;figure(); subplot(2,1,1);plot(n,CH1);title("Actual Audio");subplot(2,1,2);plot(n,CH1_BLS);title("Band-limited Aduio");
%sound(CH1_BLS, Fs);%Uncomment to test the audio.
disp('All audios band limited between 0-15KHz & amplitude is scaled between -1 to 1.');
%%                                  FM of channels----(Transmitter)
Fs = 40e6; %Fs>2Fcmax; however Fs is kept slightly higher.
fdev = 15000; %B = fdev*max(m(t))/BW; for B=1 & BW=15kHz. Where B is the modulation index.
Fc1=2e6;    Fc2=6e6;    Fc3=10e6;    Fc4 = 14e6;
u1 = 0.1*fmmod(CH1_BLS,Fc1,Fs,fdev); u1_bl = bandpass(u1,[Fc1-100e3 Fc1+100e3], Fs);
u2 = 0.1*fmmod(CH2_BLS,Fc2,Fs,fdev); u2_bl = bandpass(u2,[Fc2-100e3 Fc2+100e3], Fs);
u3 = 0.1*fmmod(CH3_BLS,Fc3,Fs,fdev); u3_bl = bandpass(u3,[Fc3-100e3 Fc3+100e3], Fs);
u4 = 0.1*fmmod(CH4_BLS,Fc4,Fs,fdev); u4_bl = bandpass(u4,[Fc4-100e3 Fc4+100e3], Fs);% this channel is low powered signal.
disp('Channels individually modulated.')
%freqz(d1)
%%                                  Multiplexing of channels----(Transmitter)
Fs = 40e6; %Fs>2Fcmax; however Fs is kept slightly higher.
U_20MHz = u1+u2+u3+u4; %multiplexing of channels.
disp('Channels multiplexed.')
    sa2 = dsp.SpectrumAnalyzer('SampleRate',Fs, ...
      'PlotAsTwoSidedSpectrum',false,...
      'YLimits',[-60 40]);
    sa2(U_20MHz)

%%                                  Analog to digital conversion(12-bit)----(Data processor)
% Max amplitude can be 0.499750v as restricted by the FPGA's ADC, so normalize
% the modulated signal with twice(actually 2.002)
% the max amplitude.
wait0 = U_20MHz/(2.002*max(U_20MHz));
wait1 = ((single(wait0))*(1e6))/244; %since the data is represented in terms of LSBs and the 1st LSB is equivalent to 0.244 mV. Also, 10e6 is multiplied to state the data in micro volts
bits = 14; %Here, specify the number of bits that you want in binary e.g 12.
q = quantizer('fixed', [bits 0]);
wait2 = num2bin(q,wait1);
data_bin = wait2; %data_bin contains the complete binary data.
fprintf('converted to %d-bit binary data.\n',bits) %d signifies decimal.
%%                                  Writing to .txt file----(Data processor)
% NOTE: The FFT IP only supports at max 2^16 = 65536 points.
data_samples = 65536; %Specify the number of samples(no# of columns) that you want to write to the file 'c'.
for_file = data_bin(1:data_samples,:);
cd 'D:\1X2\season 8\FYP\FYP Simulation';%change the current directory
writematrix(for_file,'c.txt','Delimiter',' ')
disp('Written to txt file.')
%open('c.txt')
%%                                  Performing FFT----(Data processor)
fs = 40e6;
no_of_samples = 2048; %2^10, Specify the number of samples that you want to use to compute FFT.
t_limit = no_of_samples/fs;
t = 0:1/fs:t_limit-1/fs; %time vector dictated by t_limit
n = no_of_samples; %number of samples
y = fft(U_20MHz,n);
y0 = fftshift(y); %look in shift 1-D Signal example of fftshift
f0 = (-n/2:n/2-1)*(fs/n); %0-centered frequency range
y1 = (y); %Actual FFT.
f1 = (0:n-1)*(fs/n); %Actual frequency range
jff = (0:n-1);
amplitude0 = abs(y0);
amplitude1 = (y1);
amplitude2 = abs(y1);
  figure();
  stem(f0,amplitude0, 'filled'); xlabel('Frequency'); ylabel('Amplitude')
  title('FFT performed on sampled data (centered)')
disp('FFT done.')
%%                                  Performing Peak detection----(Data processor)
a = max(amplitude0);
b = min(amplitude0);
greater = zeros();
```

```
thr = (a+b)/2;
i = 1;
for x = 1:length(amplitude0)
  if amplitude0(x)>thr
     greater(i) = x;
     i = i + 1;
  end
end
temp = zeros();
dominant  = zeros();
i = 1;
for x = 1:length(greater)
temp(i) = f0(greater(x));
i = i+1;
end
dominant = temp( temp>=0 );
dominant = sort(dominant, 'descend');
disp('Peak detection done.')
%%                                          Displaying detected channels
temp2 = round(dominant,2,'significant')
for x = 1:length(temp2)
channels(x) = x;
end
channel_no = transpose(channels);
for x = 1:length(temp2)
freq(x) = temp2(x);
end
frequency = transpose(freq);
T = table(channel_no, frequency)
%%                                          Asking for channel to demodulate
prompt = 'Enter channel number to demodulate: ';
input1 = input(prompt);
%%                                          Demodulating
Fs = 40e6;%Fs>2Fcmax; however Fs is kept slightly higher.
% R0 = transpose(wait1);
R0 = U_20MHz;
BW = 15e3;
beta = fdev/(15e3);
carsons = 2*(beta+1)*(BW);
f1 = dominant(input1)-(carsons/2)
f2 = dominant(input1)+(carsons/2)
% bpFilt = designfilt('bandpassfir','FilterOrder',800, ...
%        'CutoffFrequency1',f1,'CutoffFrequency2',f2, ...
%        'SampleRate',Fs);
R1 = bandpass(R0,[f1 f2], Fs);
sa2 = dsp.SpectrumAnalyzer('SampleRate',Fs, ...
         'PlotAsTwoSidedSpectrum',false,...
         'YLimits',[-60 40]);
      sa2(R1)
R2 = fmdemod(R1,dominant(input1),Fs,fdev);
disp('Demodulation done.')
%%                                          Low pass filtering 0-15KHz
Fs = 40e6;
R3 = lowpass(R2, 20e3, Fs);
sound(R3, 48e3);
plot(R3);
%%                                          converting from Digital to analog signal
for a = 1:480000
  if(abs(R3(a))>0.7)
     R3(a)=0;
  end
end
Fs = 48000;
figure;
subplot(2,1,1)
plot(R3);
xlabel("Sample#"); ylabel("Amplitude")
title("Demodulated Channel")
subplot(2,1,2)
plot(CH4_BLS);
title("Original Audio")
sound(R3, Fs)
xlabel("Sample#"); ylabel("Amplitude")
```

B2: User Interface

B2.1 Arduino

```
#include<SPI.h>//Library for SPI
#include <UTFT.h>//Library for 3.2 TFT LCD
#include "pitches.h"
//#include <avr/pgmspace.h>
//Rotary Encoder Inputs
#define clk 2
```

```
#define DT 3
#define SW 4
#define buzzer 12
// Defining frequency of each music note
#define NOTE_C4 262
#define NOTE_D4 294
#define NOTE_E4 330
#define NOTE_F4 349
#define NOTE_G4 392
#define NOTE_A4 440
#define NOTE_B4 494
#define NOTE_C5 523
#define NOTE_D5 587
#define NOTE_E5 659
#define NOTE_F5 698
#define NOTE_G5 784
#define NOTE_A5 880
#define NOTE_B5 988
// Change to 0.5 for a slower version of the song, 1.25 for a faster version
const float songSpeed = 1.0;
unsigned long time1=0;
unsigned long time2=0;
unsigned long time_diff=0;
int counter = 0;
int modulo = 0;
int aState;
int aLastState;
int x;
// Music notes of the song, 0 is a rest/pulse
int notes[] = {
  NOTE_E4, NOTE_G4, NOTE_A4, NOTE_A4, 0,
  NOTE_A4, NOTE_B4, NOTE_C5, NOTE_C5, 0,
  NOTE_C5, NOTE_D5, NOTE_B4, NOTE_B4, 0,
  NOTE_A4, NOTE_G4, NOTE_A4, 0,
};
// Durations (in ms) of each music note of the song
// Quarter Note is 250 ms when songSpeed = 1.0
int durations[] = {
  125, 125, 250, 125, 125,
  125, 125, 250, 125, 125,
  125, 125, 250, 125, 125,
  125, 125, 375, 125,
};
byte Mastersend,Mastereceive;
bool value = LOW;
bool flag1_sense = LOW;
bool flag2_stream = LOW;
bool flag3_record = LOW;
bool flag4_playback = LOW;
bool flag5_settings = LOW;
bool flag_streaming = LOW;

UTFT myGLCD(CTE32HR, 38, 39, 7, 41); //480x320 pixels
extern unsigned int bird01[];//NUCES Logo
extern uint8_t BigFont[];
extern uint8_t Ubuntu[];
extern uint8_t arial_bold[];//16x16 pixels
char title1[] = "Digital Radio";
char menu1[5][15] = {"Sense", "Stream", "Record", "Playback", "Settings"};
char menu1_a[9][150] = {"CH1: ", "CH2: ", "CH3: " , "CH4: ", "CH5: ", "CH6: ", "CH7: " , "CH8: ",  "Back"};
float freqs[8] = {0,0,0,0,0,0,0,0};
byte temp1 [18] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

void setup()
{
  //Rotary encoder & buzzer connections.
  pinMode (clk, INPUT);
  pinMode (DT, INPUT);
  pinMode (SW, INPUT_PULLUP);
  pinMode (buzzer, OUTPUT);
  //
  Serial.begin(115200);               //Starts Serial Communication at Baud Rate 115200
  SPI.begin();                        //Begins the SPI commnuication
  SPI.setClockDivider(SPI_CLOCK_DIV8);   //Sets clock for SPI communication at 8 (16/8=2Mhz)
  digitalWrite(SS,HIGH);              // Setting SlaveSelect as HIGH (So master doesnt connnect with slave)
  aLastState = digitalRead(clk);

  myGLCD.InitLCD(LANDSCAPE);
  myGLCD.setFont(Ubuntu);
  myGLCD.fillScr(255, 255, 255);
  //Welcome with NUCES logo & bigger font.
  myGLCD.drawBitmap (180, 100, 120, 120, bird01);
  myGLCD.setBackColor(255, 255, 255);
  myGLCD.setColor(0, 0, 0);
  myGLCD.print(title1, 90, 20);
  //Buzzer alert on powering ON.
  digitalWrite(buzzer, HIGH);
  delay(1000);
```

```
  digitalWrite(buzzer, LOW);
  delay(2000);
  //Main menu in smaller font with 1st option highlighted.
  myGLCD.setFont(arial_bold);
  main_menu();
}
void loop()
{
                                                    // Rotary encoder rotation sensing.
  {
  value = digitalRead(SW);
  aState = digitalRead(clk); // Reads the "current" state of the clk
  // If the previous and the current state of the clk are different, that means a Pulse has occured
  if (aState != aLastState) {
    // If the DT state is different to the clk state, that means the encoder is rotating clockwise
    if (digitalRead(DT) != aState) {
      counter ++;
    } else if (digitalRead(DT) == aState) {
      counter --;
    }
    modulo = counter % 10;
    Serial.print("Position: ");
    Serial.println(modulo);
    Serial.println(flag2_stream);
                                        //****For single sided menu****//
    if (flag2_stream == 0 && flag3_record == 0 && flag4_playback == 0)
    {
      if (modulo == 0 || modulo == 5 || modulo == -5 )
      { high_lighter_l(0);}
      if (modulo == 1 || modulo == 6)
      { high_lighter_l(1);}
      if (modulo == 2 || modulo == 7)
      { high_lighter_l(2);}
      if (modulo == 3 || modulo == 8)
      { high_lighter_l(3);}
      if (modulo == 4 || modulo == 9)
      { high_lighter_l(4);}
      if (modulo == -4 || modulo == -9)
      { high_lighter_l(1);}
      if (modulo == -3 || modulo == -8)
      { high_lighter_l(2);}
      if (modulo == -2 || modulo == -7)
      { high_lighter_l(3);}
      if (modulo == -1 || modulo == -6)
      { high_lighter_l(4);}
    }
                                      //****For double sided menu****//
    if (flag2_stream == 1 || flag3_record == 1|| flag4_playback == 1)
    {
      {
      if (modulo == 0)
      { high_lighter_l(0);}
      if (modulo == 1 || modulo == -9)
      { high_lighter_l(1);}
      if (modulo == 2 || modulo == -8)
      { high_lighter_l(2);}
      if (modulo == 3 || modulo == -7)
      { high_lighter_l(3);}
      if (modulo == 4 || modulo == -6)
      { high_lighter_l(4);}
      if (modulo == 5 || modulo == -5)
      { high_lighter_r(0);}
      if (modulo == 6 || modulo == -4)
      { high_lighter_r(1);}
      if (modulo == 7 || modulo == -3)
      { high_lighter_r(2);}
      if (modulo == 8 || modulo == -2)
      { high_lighter_r(3);}
      if (modulo == 9 || modulo == -1)
      { high_lighter_r(4);}
    }
    }
    digitalWrite(buzzer, HIGH);
    delay(30);
    digitalWrite(buzzer, LOW);
  }
  aLastState = aState; // Updates the previous state of the clk with the current state
  }

                                                    // Rotary encoder button press sensing.
  if (!value) {  //Rtry enc button has been pressed when 0
    digitalWrite(buzzer, HIGH);
    delay(60);
    digitalWrite(buzzer, LOW);
    myGLCD.fillScr(255, 255, 255);

    if (!flag2_stream && !flag3_record && !flag4_playback)
    {
```

```
////////////////////////////////////////////////////////        option1        ////////////////////////////////
if ((modulo == 0 || modulo == 5 || modulo == -5) )
{
  flag1_sense = HIGH;//In secondary menu of "Sense".
  myGLCD.setBackColor(255, 255, 255);
  myGLCD.setColor(0, 0, 0);
  myGLCD.fillScr(255, 255, 255);
  myGLCD.print("Fetching new data...", 90, 152);
  //add code to send command to FPGA to send dom_freqs over SPI
  time_diff = 0;
  time1 = millis();
     while(time_diff<3000)//will run until 3 seconds have elapsed
     {
     time2 = millis();
     spi_master(1, 0, 0, 0);
     time_diff = time2-time1;
     }

  int j=0, k=0;
  for(j=0; j<18; j++)
  {
   temp1[j] = 0;
  }
  for(j=0; j<8; j++)
  {
   freqs[j] = 0;
  }
  j=0;
  while(j<18)
     {
     spi_master(0, 1, 0, 0);
     temp1[j] = Mastereceive;
     j = j+1;
     }

     j = 2;
     while(j<17)
     {
      freqs[k] = float ((((temp1[j+1]<<8) + temp1[j])-0.032)/1024);//concatenation of two bytes into one word.
      j = j+2;
      k = k+1;
     }

  myGLCD.fillScr(255, 255, 255);
  myGLCD.print("Done!", 200, 152);
  spi_master(0, 0, 0, 0);
  delay(100);
      digitalWrite(buzzer, HIGH); delay(100); digitalWrite(buzzer, LOW); delay(100);
      digitalWrite(buzzer, HIGH); delay(100); digitalWrite(buzzer, LOW); delay(100);
      digitalWrite(buzzer, HIGH); delay(300); digitalWrite(buzzer, LOW); delay(100);
  main_menu();
  flag1_sense = LOW;// This will make it exit "sense" secondary menu.
}

////////////////////////////////////////////////////////        option2        ////////////////////////////////
if ((modulo == 1 || modulo == 6 || modulo == -4 || modulo == -9))
{

  flag2_stream = HIGH;
  double_menu();

  myGLCD.setColor(0, 0, 0);
  myGLCD.print(menu1_a[0], 15, 36);
  myGLCD.print(menu1_a[1], 15, 98);
  myGLCD.print(menu1_a[2], 15, 160);
  myGLCD.print(menu1_a[3], 15, 222);
  myGLCD.print(menu1_a[8], 15, 284);
  myGLCD.print(menu1_a[4], 255, 36);
  myGLCD.print(menu1_a[5], 255, 98);
  myGLCD.print(menu1_a[6], 255, 160);
  myGLCD.print(menu1_a[7], 255, 222);
  myGLCD.print("Next", 255, 284);

  myGLCD.printNumF(float(freqs[0]), 3, 80, 36);
  myGLCD.printNumF(float(freqs[1]), 3, 80, 98);
  myGLCD.printNumF(float(freqs[2]), 3, 80, 160);
  myGLCD.printNumF(float(freqs[3]), 3, 80, 222);
  myGLCD.printNumF(float(freqs[4]), 3, 320, 222);
  myGLCD.printNumF(float(freqs[5]), 3, 320, 36);
  myGLCD.printNumF(float(freqs[6]), 3, 320, 98);
  myGLCD.printNumF(float(freqs[7]), 3, 320, 160);

}
////////////////////////////////////////////////////////        option3        ////////////////////////////////
if ((modulo == 2 || modulo == 7 || modulo == -3 || modulo == -8))
{
  flag3_record = HIGH;
  double_menu();
```

```
  streaming_menu();

 }
 ////////////////////////////////////////////////////        option4        ////////////////////////////////
 if ((modulo == 3 || modulo == 8 || modulo == -2 || modulo == -7))
 {
  flag4_playback = HIGH;
  myGLCD.setBackColor(255, 255, 255);
  myGLCD.setColor(0, 0, 0);
  myGLCD.fillScr(255, 255, 255);
  myGLCD.print("Music!", 200, 152);
  myGLCD.setBackColor(200, 225, 225);
  myGLCD.setColor(200, 225, 225);
  myGLCD.fillRect(10, 258, 230, 310); //4
  myGLCD.setColor(0, 0, 0);
  myGLCD.print("Back", 15, 284);

  myGLCD.setColor(0, 0, 0);
  myGLCD.drawRect(10, 257, 230, 310);
  myGLCD.drawRect(11, 256, 229, 309);
  // full_music();
  // delay(1000);
  // shapeofyou();
  spi_master(0, 0, 0, 1);
  value = 1;
  while (value)
  {  //Rtry enc button has been pressed when 0
    value = digitalRead(SW);
    delay(40);
  }
  // full_music();
  // delay(1000);
  // shapeofyou();
  digitalWrite(buzzer, HIGH);
  delay(60);
  digitalWrite(buzzer, LOW);
  spi_master(0, 0, 0, 0);
  main_menu();
  flag4_playback = LOW;
 }
 ////////////////////////////////////////////////////        option5        ////////////////////////////////
 if ((modulo == 4 || modulo == 9 || modulo == -1 || modulo == -6))
 {
  flag5_settings = LOW; ////////////////////////////////jugaaaaar//////////////////////
  flag2_stream = LOW;   flag3_record = LOW;   flag4_playback = LOW;
  main_menu();
 }
 }
 else
 {
  if(flag2_stream)
 {
  if (modulo == 4 || modulo == 9){
  flag2_stream = LOW;
  main_menu();
  }
  else
  {
  myGLCD.setBackColor(255, 255, 255);
  myGLCD.setColor(0, 0, 0);
  myGLCD.fillScr(255, 255, 255);
  myGLCD.print("Playing channel# ", 80, 152);
  myGLCD.printNumI(modulo+1, 352, 152);

  myGLCD.setBackColor(200, 225, 225);
  myGLCD.setColor(200, 225, 225);
  myGLCD.fillRect(10, 258, 230, 310); //4
  myGLCD.setColor(0, 0, 0);
  myGLCD.print("Back", 15, 284);

  myGLCD.setColor(0, 0, 0);
  myGLCD.drawRect(10, 257, 230, 310);
  myGLCD.drawRect(11, 256, 229, 309);
  spi_master(0, 0, 1, 0);
  value = 1;
  while (value)
  {  //Rtry enc button has been pressed when 0
    value = digitalRead(SW);
    delay(40);
  }

  digitalWrite(buzzer, HIGH);
  delay(60);
  digitalWrite(buzzer, LOW);
  spi_master(0, 0, 0, 0);
  double_menu();
  streaming_menu();
  }
```

```
   }
   if(flag3_record)
   {
    if (modulo == 4 || modulo == 9){
    flag3_record = LOW;
    main_menu();
    }
    else
    {
    myGLCD.setBackColor(255, 255, 255);
    myGLCD.setColor(0, 0, 0);
    myGLCD.fillScr(255, 255, 255);
    myGLCD.print("Recording channel# ", 80, 152);
    myGLCD.printNumI(modulo+1, 368, 152);
    delay(2000);
    double_menu();
    streaming_menu();
    }
   }
   }
   counter = 0;//For the selection to return back to 1st entry.
   modulo = 0;
 }

}
//**********************************************************************************************************************************************************
******************
//**********************************************************************************************************************************************************
******************
void main_menu()
 {
 myGLCD.setBackColor(200, 225, 225);
 myGLCD.setColor(200, 225, 225);
 myGLCD.fillScr(255, 255, 255);

 myGLCD.fillRect(10, 10, 230, 62); //0
 myGLCD.fillRect(10, 72, 230, 124); //1
 myGLCD.fillRect(10, 134, 230, 186); //2
 myGLCD.fillRect(10, 196, 230, 248); //3
 myGLCD.fillRect(10, 258, 230, 310); //4

 myGLCD.setColor(0, 0, 0);
 myGLCD.print(menu1[0], 15, 36);
 myGLCD.print(menu1[1], 15, 98);
 myGLCD.print(menu1[2], 15, 160);
 myGLCD.print(menu1[3], 15, 222);
 myGLCD.print(menu1[4], 15, 284);

 myGLCD.setColor(0, 0, 0);
 myGLCD.drawRect(10, 10, 230, 62);
 myGLCD.drawRect(11, 11, 229, 61);
 }

void single_menu()
 {
 myGLCD.fillScr(255, 255, 255);
 myGLCD.setBackColor(200, 225, 225);
 myGLCD.setColor(200, 225, 225);
 myGLCD.fillRect(10, 10, 230, 62); //0
 myGLCD.fillRect(10, 72, 230, 124); //1
 myGLCD.fillRect(10, 134, 230, 186); //2
 myGLCD.fillRect(10, 196, 230, 248); //3
 myGLCD.fillRect(10, 258, 230, 310); //4
 myGLCD.setColor(0, 0, 0);
 myGLCD.drawRect(10, 10, 230, 62);
 myGLCD.drawRect(11, 11, 229, 61);
 }
void double_menu()
 {
 myGLCD.fillScr(255, 255, 255);
 myGLCD.setBackColor(200, 225, 225);
 myGLCD.setColor(200, 225, 225);
 myGLCD.fillRect(10, 10, 230, 62); //0
 myGLCD.fillRect(10, 72, 230, 124); //1
 myGLCD.fillRect(10, 134, 230, 186); //2
 myGLCD.fillRect(10, 196, 230, 248); //3
 myGLCD.fillRect(10, 258, 230, 310); //4
 myGLCD.drawRect(10, 10, 230, 62);
 myGLCD.drawRect(11, 11, 229, 61);
 myGLCD.fillRect(250, 10, 470, 62); //5
 myGLCD.fillRect(250, 72, 470, 124); //6
 myGLCD.fillRect(250, 134, 470, 186); //7
 myGLCD.fillRect(250, 196, 470, 248); //8
 myGLCD.fillRect(250, 258, 470, 310); //9
 myGLCD.setColor(0, 0, 0);
 myGLCD.drawRect(10, 10, 230, 62);
 myGLCD.drawRect(11, 11, 229, 61);
```

```
  }
  void streaming_menu()
  {
  myGLCD.setColor(0, 0, 0);
  myGLCD.print(menu1_a[0], 15, 36);
  myGLCD.print(menu1_a[1], 15, 98);
  myGLCD.print(menu1_a[2], 15, 160);
  myGLCD.print(menu1_a[3], 15, 222);
  myGLCD.print(menu1_a[8], 15, 284);
  myGLCD.print(menu1_a[4], 255, 36);
  myGLCD.print(menu1_a[5], 255, 98);
  myGLCD.print(menu1_a[6], 255, 160);
  myGLCD.print(menu1_a[7], 255, 222);
  myGLCD.print("Next", 255, 284);

  myGLCD.printNumF(float(freqs[0]), 3, 80, 36);
  myGLCD.printNumF(float(freqs[1]), 3, 80, 98);
  myGLCD.printNumF(float(freqs[2]), 3, 80, 160);
  myGLCD.printNumF(float(freqs[3]), 3, 80, 222);
  myGLCD.printNumF(float(freqs[4]), 3, 320, 222);
  myGLCD.printNumF(float(freqs[5]), 3, 320, 36);
  myGLCD.printNumF(float(freqs[6]), 3, 320, 98);
  myGLCD.printNumF(float(freqs[7]), 3, 320, 160);

  // myGLCD.setColor(0, 0, 0);
  // myGLCD.drawRect(10, 10, 230, 62);
  // myGLCD.drawRect(11, 11, 229, 61);
  }

 void high_lighter_I(int entry)
 {
  myGLCD.setColor(0, 0, 0);
  myGLCD.drawRect(10, (10 + (62 * entry)), 230, (62 + (62 * entry)));
  myGLCD.drawRect(11, (11 + (62 * entry)), 229, (61 + (62 * entry)));

  myGLCD.setColor(245, 245, 255);
  if (entry == 0)
  {
    myGLCD.setColor(245, 245, 255);
    myGLCD.drawRect(10, 72, 230, 124);
    myGLCD.drawRect(10, 258, 230, 310);

    myGLCD.setColor(255, 255, 255);
    myGLCD.drawRect(11, 73, 229, 123);
    myGLCD.drawRect(11, 259, 229, 309);

    if (flag2_stream == 1 || flag3_record == 1 || flag4_playback == 1)
    {
      myGLCD.drawRect(251, 259, 469, 309);
      myGLCD.setColor(245, 245, 255);
      myGLCD.drawRect(250, 258, 470, 310);
    }
  }
  if (entry == 1)
  {

    myGLCD.drawRect(10, 10, 230, 62);
    myGLCD.drawRect(10, 134, 230, 186);
    myGLCD.setColor(255, 255, 255);
    myGLCD.drawRect(11, 11, 229, 61);
    myGLCD.drawRect(11, 135, 229, 185);

  }
  if (entry == 2)
  {
    myGLCD.setColor(245, 245, 255);
    myGLCD.drawRect(10, 72, 230, 124);
    myGLCD.drawRect(10, 196, 230, 248);
    myGLCD.setColor(255, 255, 255);
    myGLCD.drawRect(11, 73, 229, 123);
    myGLCD.drawRect(11, 197, 229, 247);

  }
  if (entry == 3)
  {
    myGLCD.setColor(245, 245, 255);
    myGLCD.drawRect(10, 134, 230, 186);
    myGLCD.drawRect(10, 258, 230, 310);
    myGLCD.setColor(255, 255, 255);
    myGLCD.drawRect(11, 135, 229, 185);
    myGLCD.drawRect(11, 259, 229, 309);

  }
  if (entry == 4)
  {
    myGLCD.setColor(245, 245, 255);
    myGLCD.drawRect(10, 196, 230, 248);
    myGLCD.drawRect(10, 10, 230, 62);
```

```
    myGLCD.setColor(255, 255, 255);
    myGLCD.drawRect(11, 197, 229, 247);
    myGLCD.drawRect(11, 11, 229, 61);

    if (flag2_stream == 1 || flag3_record == 1 || flag4_playback == 1)
    {
      myGLCD.setColor(245, 245, 255);
      myGLCD.drawRect(250, 10, 470, 62);
      myGLCD.setColor(255, 255, 255);
      myGLCD.drawRect(251, 11, 469, 61);
    }

  }

}
void high_lighter_r(int entry)
{
  myGLCD.setColor(0, 0, 0);
  myGLCD.drawRect(250, (10 + (62 * entry)), 470, (62 + (62 * entry)));
  myGLCD.drawRect(251, (11 + (62 * entry)), 469, (61 + (62 * entry)));

  if (entry == 0)
  {
    myGLCD.setColor(245, 245, 255);
    myGLCD.drawRect(250, 72, 470, 124);
    myGLCD.drawRect(250, 258, 470, 310);
    myGLCD.drawRect(10, 258, 230, 310);
    myGLCD.setColor(255, 255, 255);
    myGLCD.drawRect(251, 73, 469, 123);
    myGLCD.drawRect(251, 259, 469, 309);
    myGLCD.drawRect(11, 259, 229, 309);
  }
  if (entry == 1)
  {
    myGLCD.setColor(245, 245, 255);
    myGLCD.drawRect(250, 10, 470, 62);
    myGLCD.drawRect(250, 134, 470, 186);
    myGLCD.setColor(255, 255, 255);
    myGLCD.drawRect(251, 11, 469, 61);
    myGLCD.drawRect(251, 135, 469, 185);

  }
  if (entry == 2)
  {
    myGLCD.setColor(245, 245, 255);
    myGLCD.drawRect(250, 72, 470, 124);
    myGLCD.drawRect(250, 196, 470, 248);
    myGLCD.setColor(255, 255, 255);
    myGLCD.drawRect(251, 73, 469, 123);
    myGLCD.drawRect(251, 197, 469, 247);
  }
  if (entry == 3)
  {
    myGLCD.setColor(245, 245, 255);
    myGLCD.drawRect(250, 134, 470, 186);
    myGLCD.drawRect(250, 258, 470, 310);
    myGLCD.setColor(255, 255, 255);
    myGLCD.drawRect(251, 135, 469, 185);
    myGLCD.drawRect(251, 259, 469, 309);
  }
  if (entry == 4)
  {
    myGLCD.setColor(245, 245, 255);
    myGLCD.drawRect(250, 196, 470, 248);
    myGLCD.drawRect(250, 10, 470, 62);
    myGLCD.drawRect(10, 10, 230, 62);
    myGLCD.setColor(255, 255, 255);
    myGLCD.drawRect(251, 11, 469, 61);
    myGLCD.drawRect(251, 197, 469, 247);
    myGLCD.drawRect(11, 11, 229, 61);
  }

}
void spi_master(int take_fft, int send_freqs, int stream, int playback)
{
  if(take_fft == HIGH)            //Logic for Setting x value (To be sent to slave) depending upon input from pin 2
  {
    x = 1;
  digitalWrite(SS, LOW);              //Starts communication with Slave connected to master
  Mastersend = x;
  Mastereceive=SPI.transfer(Mastersend); //Send the mastersend value to slave also receives value from slave
  Serial.println("Data received: ");
  Serial.print(Mastereceive);
  Serial.println();
  delay(100);
  }
  else if(send_freqs==HIGH)
  {
```

```
  x = 2;
 digitalWrite(SS, LOW);              //Starts communication with Slave connected to master
 Mastersend = x;
 Mastereceive=SPI.transfer(Mastersend); //Send the mastersend value to slave also receives value from slave
 Serial.println("Data received: ");
 Serial.print(Mastereceive);
 Serial.println();
 delay(100);
 }
 else if (stream==HIGH)
 {
 x = 3;
 digitalWrite(SS, LOW);              //Starts communication with Slave connected to master
 Mastersend = x;
 Mastereceive=SPI.transfer(Mastersend); //Send the mastersend value to slave also receives value from slave
 Serial.println("Data received: ");
 Serial.print(Mastereceive);
 Serial.println();
 delay(100);
 }
 else if(playback==HIGH)
 {
  x = 4;
 digitalWrite(SS, LOW);              //Starts communication with Slave connected to master
 Mastersend = x;
 Mastereceive=SPI.transfer(Mastersend); //Send the mastersend value to slave also receives value from slave
 Serial.println("Data received: ");
 Serial.print(Mastereceive);
 Serial.println();
 delay(100);
 }
 else
 {
  x = 0;
 digitalWrite(SS, LOW);              //Starts communication with Slave connected to master
 Mastersend = x;
 Mastereceive=SPI.transfer(Mastersend); //Send the mastersend value to slave also receives value from slave
 Serial.println("Data received: ");
 Serial.print(Mastereceive);
 Serial.println();
 delay(100);
 digitalWrite(SS, HIGH);             //Stops communication with Slave connected to master
  }
}
```

B2.2 FPGA SPI Slave

```
`timescale 1ns / 1ps
module SPI_slave1(clk, SCK, MOSI, pk_dtc_flag, data_8bit, MISO, SSEL, LED, byte_received);
input clk;
input SCK, SSEL, MOSI;
input pk_dtc_flag;
input [7:0]data_8bit;
output wire MISO;
output [7:0]LED;
output reg byte_received;  // high when a byte has been received
////////////////////////////////////////////////////////////////////////////////
// sync SCK to the FPGA clock using a 3-bits shift register
reg [2:0] SCKr;  always @(posedge clk) SCKr <= {SCKr[1:0], SCK};
wire SCK_risingedge = (SCKr[2:1]==2'b01);  // now we can detect SCK rising edges
wire SCK_fallingedge = (SCKr[2:1]==2'b10);  // and falling edges
// same thing for SSEL
wire SSEL_active, SSEL_startmessage, SSEL_endmessage;
reg [2:0] SSELr;  always @(posedge clk) SSELr <= {SSELr[1:0], SSEL};
assign SSEL_active = ~SSELr[1];  // SSEL is active low
assign SSEL_startmessage = (SSELr[2:1]==2'b10);  // message starts at falling edge
assign SSEL_endmessage = (SSELr[2:1]==2'b01);  // message stops at rising edge
// and for MOSI
reg [1:0] MOSIr;  always @(posedge clk) MOSIr <= {MOSIr[0], MOSI};
wire MOSI_data = MOSIr[1];
////////////////////////////////////////////////////////////////////////////////
// we handle SPI in 8-bits format, so we need a 3 bits counter to count the bits as they come in
reg [2:0] bitcnt;
reg [7:0] byte_data_received;
//////////////////////////////////////////Receiver/////////////////////////////
always @(posedge clk)
begin
 if(~SSEL_active)
   bitcnt <= 3'b000;
 else
 if(SCK_risingedge)
 begin
   bitcnt <= bitcnt + 3'b001;
   // implement a shift-left register (since we receive the data MSB first)
   byte_data_received <= {byte_data_received[6:0], MOSI_data};
 end
end
always @(posedge clk) byte_received <= SSEL_active && SCK_risingedge && (bitcnt==3'b111);
```

```
// we use the data received to control 8 LEDs
reg [7:0]LED;
always @(posedge clk) if(byte_received) LED <= byte_data_received;

////////////////////////////////Transmitter////////////////////////////////
reg [7:0] byte_data_sent=8'd0;
reg [7:0] cnt = 8'b11001000;//send 200 as the first message for successful connection.
//always @(posedge clk) if(SSEL_startmessage) cnt<=cnt+8'h1;  // count the messages
always @(posedge clk)
if(SSEL_active)
begin
//  if(SSEL_startmessage)
//    byte_data_sent <= cnt;  // first byte sent in a message is of successful connection.
//  else
  if(SCK_fallingedge)
  begin
    if(bitcnt==3'b000)//It is being administered by above as both (send/receive) follow the same clocks.
    begin
    if(pk_dtc_flag)
            begin
            byte_data_sent <= data_8bit;
            end
            else
            byte_data_sent <= cnt;//just send 0 after testing of 200.
    end
    else
      byte_data_sent <= {byte_data_sent[6:0], 1'b0};
  end
end

assign MISO = byte_data_sent[7];  // send MSB first

//The data in "byte_data_sent" will only be updated once it has been completely offloaded.
// we assume that there is only one slave on the SPI bus
// so we don't bother with a tri-state buffer for MISO
// otherwise we would need to tri-state MISO when SSEL is inactive
endmodule
```

B3: Peak Detector

```
`timescale 1ns / 1ns
module pk_dtr(

output reg [31:0] max, min,//maximum & minimum of the magnitudes
output reg [31:0] midrange,//midrange of the "magnitudes".

output reg [15:0]dom_freq0 = 16'd0,
output reg [15:0]dom_freq1 = 16'd0,
output reg [15:0]dom_freq2 = 16'd0,
output reg [15:0]dom_freq3 = 16'd0,
output reg [15:0]dom_freq4 = 16'd0,
output reg [15:0]dom_freq5 = 16'd0,
output reg [15:0]dom_freq6 = 16'd0,
output reg [15:0]dom_freq7 = 16'd0,
output reg fifo_read_en  = 1'd0,
output reg sorting_valid = 1'd0,

input wire [9:0] data_count,
input wire empty,
input wire [23:0] taqat,//It stores the magnitude of FFT spect. Can be plotted too!
input wire [23:0] fifo_in,
input wire fifo_full,
input wire m_axis_data_tvalid_in,//FFT is ready to produce valid data.
input clock,//The system clock
input del//system reset
);
////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////Peak Detector/////////////////////////////////////////
reg [15:0]dom_freqs[19:0]; //dominant frequencies
reg [23:0]freqs_pwr[19:0];
reg [11:0]counter1 = 12'd0;
reg [9:0]counter2 = 10'd0;// this is fix10_0(Q10.0) number. Range: 0-1023
reg [7:0]i,j,k;
reg check0 = 1'd0;
reg [15:0]temp1 = 16'd0;
reg [15:0]temp2 = 16'd0;
reg [23:0] temp3 = 24'd0;
reg [23:0] temp4 = 24'd0;
////////////////////////////////Reset all variables to zero/////////////////////////
always@(posedge clock)
begin
if (del)
begin
counter1  <= 12'd0;
counter2 <= 10'd0;
dom_freqs[0] = 16'd0;   dom_freqs[1] = 16'd0;   dom_freqs[2] = 16'd0;    dom_freqs[3] = 16'd0;
dom_freqs[4] = 16'd0;   dom_freqs[5] = 16'd0;   dom_freqs[6] = 16'd0;    dom_freqs[7] = 16'd0;
```

```
dom_freqs[8] = 16'd0;   dom_freqs[9] = 16'd0;   dom_freqs[10] = 16'd0;    dom_freqs[11] = 16'd0;
dom_freqs[12] = 16'd0;  dom_freqs[13] = 16'd0;  dom_freqs[14] = 16'd0;    dom_freqs[15] = 16'd0;
dom_freqs[16] = 16'd0;  dom_freqs[17] = 16'd0;  dom_freqs[18] = 16'd0;    dom_freqs[19] = 16'd0;


freqs_pwr[0] = 24'd0;  freqs_pwr[5] = 24'd0;  freqs_pwr[10] = 24'd0;  freqs_pwr[15] = 24'd0;
freqs_pwr[1] = 24'd0;  freqs_pwr[6] = 24'd0;  freqs_pwr[11] = 24'd0;  freqs_pwr[16] = 24'd0;
freqs_pwr[2] = 24'd0;  freqs_pwr[7] = 24'd0;  freqs_pwr[12] = 24'd0;  freqs_pwr[17] = 24'd0;
freqs_pwr[3] = 24'd0;  freqs_pwr[8] = 24'd0;  freqs_pwr[13] = 24'd0;  freqs_pwr[18] = 24'd0;
freqs_pwr[4] = 24'd0;  freqs_pwr[9] = 24'd0;  freqs_pwr[14] = 24'd0;  freqs_pwr[19] = 24'd0;



dom_freq0 <= 16'd0;     dom_freq1 <= 16'd0;     dom_freq2 <= 16'd0;     dom_freq3 <= 16'd0;
dom_freq4 <= 16'd0;     dom_freq5 <= 16'd0;     dom_freq6 <= 16'd0;     dom_freq7 <= 16'd0;

i <= 8'd0; j <= 8'd0; k <= 8'd0;
check0 <= 1'b0;
max <= 32'd0; min <= 32'd0; midrange <= 32'd0;
sorting_valid <= 1'd0;
temp1 = 16'd0;
temp2 = 16'd0;
end
//////////////////////block for  finding minimum, maximum & midrange of freq spect//////////////////////////
if (~check0 && counter1<1024 && m_axis_data_tvalid_in)//Valid signal from cordic with latency.
      begin
        if(taqat>=max)
        begin
        max <= taqat;
        end
        if(taqat<=min)
        begin
        min <= taqat;
        end
counter1 <= counter1 + 1;
check0 <= 1'b0;
end
if(counter1 >1023)
      begin
      check0 <= 1'b1;
      counter1 <= 1'b0;
      midrange <= (max+min)>>2;//unsigned division by 2, but efficient.
      end
//////////////////////block for finding dominant frequencies through fifo//////////////////////////
if (check0 && ~empty)
      begin
      fifo_read_en <= 1'b1;
      if(fifo_in>=midrange && i<20)
        begin
        dom_freqs[i] <= ((counter2)*11'd5);//variable sampling freq.
        freqs_pwr[i] <= fifo_in;
        i <= i + 1;//where 5 is the fix11_0 equivalent of sample freq/trnsfrm lngth. Actual (fix11_10(Q1.10)).
        //For 10 MHz it is "5" in fix11_10 & for 60 MHz it is "15" in fix11_9.
        end
      counter2 <= counter2 + 1;
      end
////////////////////////////////////////////////////////////////////////////////////////
if(check0 && empty && ~sorting_valid)//bubble sort
begin
fifo_read_en <= 1'b0;//Stop reading data from FIFO.
if (k<19)
   begin
   if (freqs_pwr[j]<=freqs_pwr[j+1])
      begin
      temp1 = dom_freqs[j];
      temp2 = dom_freqs[j+1];
      temp3 = freqs_pwr[j];
      temp4 = freqs_pwr[j+1];

      dom_freqs[j+1] = temp1;
      dom_freqs[j] = temp2;
      freqs_pwr[j+1] = temp3;
      freqs_pwr[j] = temp4;
      end
   if(j<18)
   j <= j+1;
   else
   begin
   j <= 8'd0;
   k <= k+1;
   end
   end
else
sorting_valid <= 1'd1;
end
////////////////////////////////////////////////////////////
if (sorting_valid)
begin
```

```
dom_freq0 <= dom_freqs[0];
dom_freq1 <= dom_freqs[1];
dom_freq2 <= dom_freqs[2];
dom_freq3 <= dom_freqs[3];
dom_freq4 <= dom_freqs[4];
dom_freq5 <= dom_freqs[5];
dom_freq6 <= dom_freqs[6];
dom_freq7 <= dom_freqs[7];
end
end
endmodule
```

B4: Entire Project

Scan the QR code below to access the complete and unified project files.

Bibliography

[1] Tech Wholesale, "History of the Radio: From Inception to Modern Day," Tech Wholesale, [Online]. Available: https://www.techwholesale.com/history-of-the-radio.html. [Accessed 1 06 2022].

[2] CJFE, "THE IMPORTANCE OF RADIO IN THE 21ST CENTURY," CANADIAN JOURNALISTS FOR FREE EXPRESSION, [Online]. Available: https://www.cjfe.org/the_importance_of_radio_in_the_21st_century#:~:text=Advances%20in%20technology%20have%20given,on%20cell%2Dphones%20or%20online.. [Accessed 1 06 2022].

[3] IJCSNS, "Implementing FFT Algorithms on FPGA," *IJCSNS International Journal of Computer Science and Network Security,* vol. 11, p. 10, 2011.

[4] K. Nair, *All Digital FM Demodulator,* Blacksburg, 2019.

[5] "Architectural design of a programmable cell for the implementation of a filter bank on FPGA," *Microelectronics Reliability,* p. 13, 2003.

[6] Federal Communications Commission, "Why Do FM Frequencies End in an Odd Decimal?," Federal Communications Commission, 1 september 2021. [Online]. Available: https://www.fcc.gov/media/radio/fm-frequencies-end-odd-decimal#:~:text=The%20FM%20broadcast%20in%20the,kHz%20(0.2%20MHz)%20wide.. [Accessed 1 06 2022].

[7] M. Subhedar, "SPECTRUM SENSING TECHNIQUES IN COGNITIVE RADIO NETWORKS," *International Journal of Next-Generation Networks,* vol. 3, p. 15, 2011.

[8] ScienceDirect, "Frequency Resolution," ScienceDirect, 2002. [Online]. Available: https://www.sciencedirect.com/topics/engineering/frequency-resolution. [Accessed 1 06 2022].

[9] Xilinx, "Vitis High-Level Synthesis User Guide," 2022. [Online]. Available: https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/Auto-Restart-Mode. [Accessed 1 06 2022].

[10] HBM, "Smart Peak Detection," HBM, [Online]. Available: https://www.hbm.com/en/4741/smart-peak-detection-for-fiber-bragg-sensors/. [Accessed 1 06 2022].

[11] A. Shah, "THE ROLE OF FM RADIO IN THE TALIBAN INSURGENCY IN SWAT," *PAKISTAN JOURNAL OF SOCIETY, EDUCATION AND LANGUAGE,* vol. 6, p. 9, 2022.

[12] "Quadrature Frequency and Phase Demodulation," All About Circuits, [Online]. Available: https://www.allaboutcircuits.com/textbook/radio-frequency-analysis-design/radio-frequency-demodulation/quadrature-frequency-and-phase-demodulation/. [Accessed 22 December 2022].

[13] "SPI 2 - A simple implementation," fpga4fun, [Online]. Available: https://www.fpga4fun.com/SPI2.html. [Accessed 22 December 2022].

[14] "CIC Compiler v4.0 Product Guide (PG140)," Feburary 2021. [Online]. Available: https://docs.xilinx.com/v/u/en-US/pg140-cic-compiler. [Accessed 22 December 2022].

[15] "DDS Compiler v6.0 Product Guide (PG141)," 21 January 2021. [Online]. Available: https://docs.xilinx.com/v/u/en-US/pg141-dds-compiler. [Accessed 22 December 2022].

[16] "PG109 Fast Fourier Transform LogiCORE IP Product Guide," 4 5 2022. [Online]. Available: https://docs.xilinx.com/r/en-US/pg109-xfft. [Accessed 22 December 2022].

[17] "FIFO Generator v13.1 Product Guide," 5 april 2017. [Online]. Available: https://docs.xilinx.com/v/u/13.1-English/pg057-fifo-generator. [Accessed 22 12 2022].

[18] "Keysight N9310A," [Online]. Available: https://www.keysight.com/us/en/assets/7018-02994/data-sheets/5990-8116.pdf. [Accessed 22 12 2022].

[19] "Nexys A7 Reference Manual," Digilent, [Online]. Available: https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual. [Accessed 22 12 2022].

[20] Y. W. a. Y. Lian, "A COMPUTATIONALLY EFFICIENT NON-UNIFORM DIGITAL," Singapore, 2004.

[21] "Online audio listenership," Pew Research Center, 29 June 2021. [Online]. Available: https://www.pewresearch.org/journalism/chart/sotnm-radio-online-radio-listenership/. [Accessed 24 December 2022].

[22] "Radio: One of the most powerful communication tools of the 21st Century," Myriad Global, [Online]. Available: https://myriadglobalmedia.com/radio-one-powerful-communication-tools-21st-

century/#:~:text=Despite%20the%20rise%20in%20social,especially%20in%20less%20developed%20countries.. [Accessed 24 December 2022].

[23] A. Basit, "A Journey from 'FM Mullah' to Head of the Pakistan Taliban," JSTOR, 2014.

[24] "Is radio relevant in the 21st century?," the hindu, 2018. [Online]. Available: https://www.thehindu.com/opinion/op-ed/is-radio-relevant-in-the-21st-century/article22423373.ece. [Accessed 24 December 2022].

[25] M. S. John G Proakis, FUNDAMENTALS of Communication Systems, Second ed., Pearson.

[26] b. Lathi, Modern Digital And Analog Communication Systems, New York: Oxford University Press, 2010.

[27] "Pulse Width Modulation," Real Digital, [Online]. Available: https://www.realdigital.org/doc/333049590c67cb553fc7f9880b2f79c3#pulse-density-modulation. [Accessed 24 December 2022].

Description of Cited Literature:

[1] This article discusses how radio evolved into a high-fidelity mode of communication and its impact on the world since WW1.

[2] This article discusses the dependence on radio networks in countries where free expression is suppressed, access to technology is expensive or illiteracy rates are high, and how international organizations use radio to tap into the networks of local communities.

[13] This article discusses the instrumental role that FM radio played during the insurgency attacks on swat by Mullah Fazlullah and how he used it to his advantage.

[3] This article compares the efficiency and implementation of different Discrete Fourier Transform (DFT) algorithms and concludes that Radix-2 is the most time-efficient algorithm of all.

[4] This article proposes an all-digital quadrature I-Q-based demodulator for FM demodulation. Then, the complete architecture of I-Q based demodulator is comprehensively explained for implementation on an FPGA.

[5] This article discusses the implementation of Filter banks on FPGAs. It proposes a three-stage Filter bank based on Modulation, Low pass Filtering, and Demodulation.

[6] This article presents the detailed specifications of the FM broadcast band and the bandwidth of each channel.

[7] This article compares different spectrum sensing techniques such as energy detection, matched filter, and FFT. It concludes that each technique may prove to be efficient in its specific scenario. Given a requirement of low signal-to-noise ratio (SNR), Fast Fourier Transform (FFT) proved to be the best.

[10] This article explains the effect of window length, sampling frequency, and other factors that determine the resolution of frequency spectrum of FFT.

[12] This article explains in depth the working of quadrature demodulation scheme specifically for FM signals.

[27] This article explains the working of PWM and a reconstruction filter as DAC, and how it is well suited for audio signals.