Audio Transmission and Reception over RF

Final MTP Report

submitted in the partial fulfillment of
requirement for the degree of

MASTER OF TECHNOLOGY

in

Electronic Systems

Submitted by

Anil Kumar Garg
(173074018)

Under the supervision of
Prof. Shalabh Gupta

Department of Electrical Engineering
Indian Institute of Technology, Bombay
Mumbai - 400076

June 2020

Acknowledgment

First of all, I would like to express my sincere gratitude to **Prof. Shalabh Gupta** for his valuable guidance and support throughout this project. I would like to thank him for his useful suggestions which helped me in smooth and successful completion of the project. I am very thankful to **Mr. Salil Tambe**, **Mr. Sandeep Goyal** for her constant support and motivation at each and every step of this work. I thank all the members of communication and VLSI lab and my colleagues for making my stay in IIT Bombay a lifetime experience with tons of memories. Finally, I would like to thank my family and friends for their constant moral support. This work is dedicated to them.

Abstract

Security applications where access is very difficult and risky in narrow space like ground tunnels, hostile environment and enemy captured area etc. For solving this issue, one electronics device with camera and speaker is mounted on trained dog, which is instructed from the base station. Camera captures the internal structure of target place and provide information to base station. After assessing video, base station gives audio commands to dog. Trained dog performs actions as per audio commands.

This type of device is very useful for places where human being access is restricted, or preventive security action is required.The success of entire mission is dependent on quality of audio and video signals, processing, and effective communication range. The available solution for audio trans receiver is walkie talkie. But available walkie talkies are very bulky and costly solution. They are also based on principal of analog audio RF transmission and reception which occupy large bandwidth.

For solving this issue, digital transmission of audio signal is suggested. Using the digital transmission, audio signal quality can be improved by applying companding, compression, forward error correction mechanism. Effective bandwidth is also reduced by compression (encoding/decoding) technique so signal to noise ratio is also improved.

The suggested audio transmission solution is tested on evaluation board (CC1310 launchpad) with PCB mounted antenna (868Mhz@Monopole) at 14dbm(25mW) transmitting power. Effective range is getting 750 meters on the main road in IIT Mumbai.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

This work implemented an audio transmission and reception system that is working on a higher sampling rate and lower channel symbol rate. The trade-off between the symbol rate and the sampling rate is the actual motivation to develop this prototype. Using this scheme we can transmit 200 bytes of information with sampling rate 8000 SPS using RF channel bit rate 50 kbps. In normal operation as per calculation, the RF channel is required a 200 kbps symbol bit rate. So actual motivation is to improve system efficiency up to approximate 400%.

Using the suggested method, wireless transmission of large amounts of audio data at low speed (symbol rate) from transmitter to receiver is possible. This audio transmission link is very useful in a security operation in enemy captured areas. Digital transmission of voice signal provides the flexibility of encoding the decoding the signal by different compression and companding techniques so that more data can be transfer with the same bandwidth with high quality. The proof of concept (POC) is done on the evaluation board (CC1310 launchpad) with PCB mounted antenna (868Mhz@Monopole) at 14dbm(25mW) transmitting power.

How the application will work?

- Access for gradually tapered spaces like ground tunnels and enemy occupied buildings is very difficult.
- For solving this problem, a camera and audio receiver is mounted on trained animal. Camera captures images and send back to control office.
- Control office gives audio command by transmitter and animal performs actions.

Figure 1.1: Application setup (Ref: communication lab doc)

1.2 Benefit Over Analog Communication

Audio transmission and reception over RF offers several advantages as listed below

- Better Signal to noise ratio.
- Easy to handle and install.
- Cost effective and light weight
- Less bandwidth is required to transmit same analog signal
- Power consumption is very less.

A extra training is required to use the equipment for animals. Another challenge is the high signal attenuation by different obstacles, which may reduces the receiver signal power strength, that, further reduces the effective communication range.

1.3 Dissertation Organization

Chapter-1 introduced briefly about Audio transmission and reception over RF and its merits/demerits. In this section we will discuss the motivation regarding this implementation. Chapter-2 describe System level architecture and expectation with each module. Here we will discuss regarding design strategy and application. Chapter 3 describe the details of individual modules and block diagram details.Here we will discuss schematic design and calculation details. Chapter 4 describes the design implementation from software and hardware point of view. In this section,we will also discuss the firmware flow diagram and modeler block diagrams. Chapter 5 describes firmware source code and programming methods. Chapter 6 describes the test results during testing of RF interface and simulations.

Chapter 2

Architecture Design

In this section, details of Audio transmission and reception over RF module have to be discussed. Here we will discuss on interface level design and architecture of scheme. We will also discuss about the technology used and calculations from design to implementation point of view. Design rationale are also described in following section. Few key points are also shared with software design strategy from implementation point of view.

2.1 Overview Of Technology Used

Following listed technologies are used during design of Audio transmission and reception over RF module

- Multicore RF controller CC1310 for sensing and RF application
- ADPCM encoder/decoder technique for data compression (16 BITS to 4 BITS)
- Customize frame format for synchronization between transmitter and receiver.
- Analog signal conditioning for microphone interface
- High sampling rate (8ksps) with DMA ping pong mechanism.
- Configurable TI Easylink RF library stack.
- MCP4921 SPI interface for digital to analog conversion.
- Speaker interface with class D audio amplifier and LC filter.

2.2 Product Architecture

Overall scheme of application is discussing in following section.

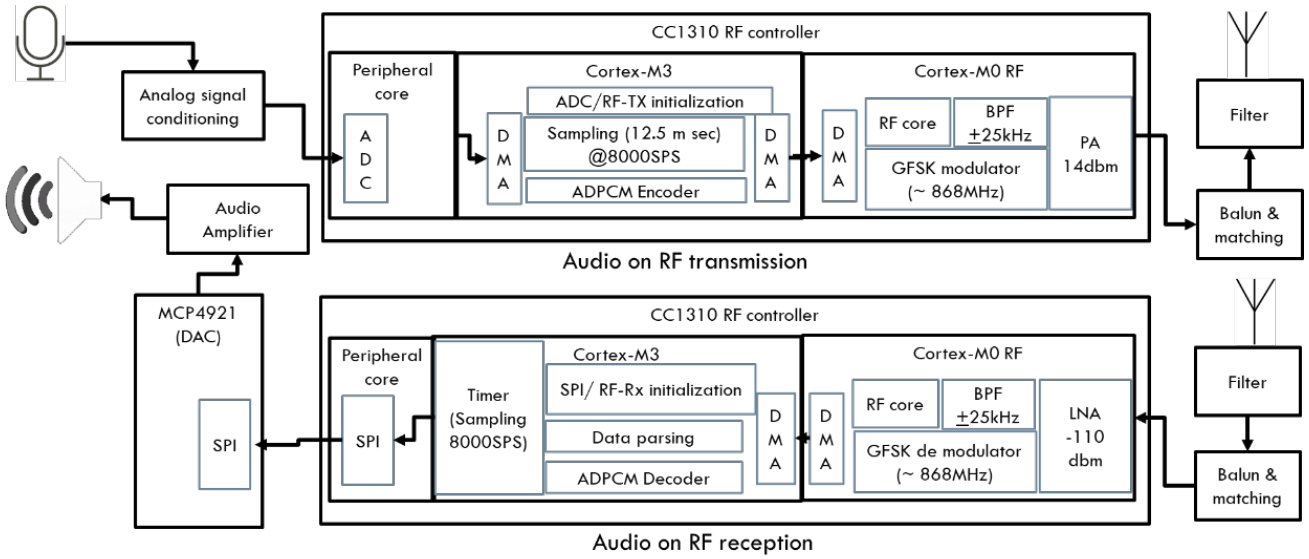

Figure 2.1: Product Architecture

Audio transmission and reception over RF module is divided in two modules

- RF Transmitter
- RF Receiver

2.3 RF Transmitter

Above module consists different sub-modules like microphone, analog signal conditioning, RF multi-core CC1310 micro-controller, balance to unbalance circuit, matching circuit and antenna. This module is used to convert the voice signal into the digital form and transmit compressed digital data via RF antenna to the receiver. The details module designing need to be discuss in the coming section.

2.4 RF Receiver

This module consists different sub modules like digital to analog converter, audio amplifier, filter, speaker, RF multi-core CC1310 micro-controller, unbalance to balance circuit, matching circuit

and antenna. This module is used to receive RF signal sent by transmitter, validate, decompressed the digital data and sent to the DAC. DAC converts the digital data into the analog form and passes to class D audio amplifier. After filtering the output of amplifier, it goes to speaker as final output. The details module designing need to be discuss in the coming section.

2.5 Hardware Block Diagram

Below module consists different sub modules like microphone, analog signal-conditioning, RF multi-core CC1310 micro controller, balance to unbalance circuit, matching circuit and antenna. This module is used to convert the voice signal into the digital form and transmit compressed digital data via RF antenna to the receiver. 5V DC power is used for signal conditioning and 3.3 V DC is used for other modules. UART port is used for boot loading and debugging purpose. XDS100 is used for controller programming purpose. LED indication is used for showing healthiness of communication.

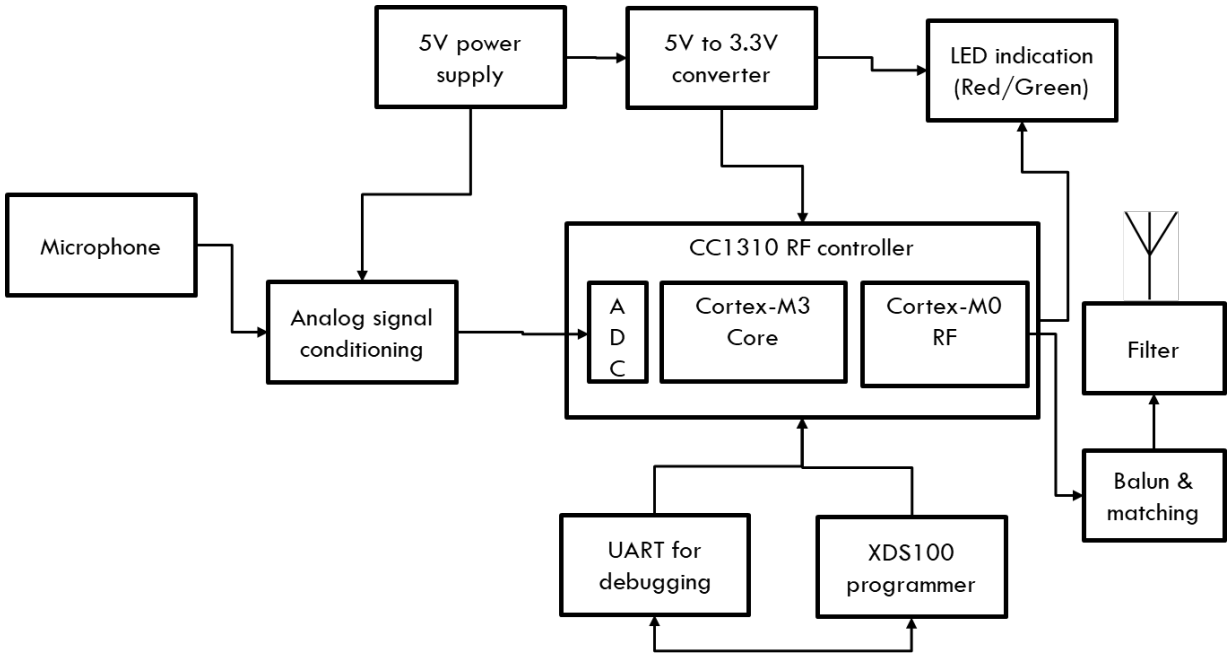

Figure 2.2: Hardware Block Diagram : Transmitter

This module consists different sub modules like digital to analog converter, audio amplifier, filter, speaker, RF multi core CC1310 micro controller, unbalance to balance circuit, matching circuit and antenna. This module is used to receive RF signal sent by transmitter, validate, decompressed

the digital data and sent to the DAC. DAC converts the digital data into the analog form and passes to class D audio amplifier. After filtering the output of amplifier, it goes to speaker as final output. 5V DC power is used for speaker, audio amplifier, MCP4921 DAC and 3.3 V DC is used for other modules. UART port is used for boot loading and debugging purpose. XDS100 is used for controller programming purpose. LED indication is used for showing healthiness of communication.

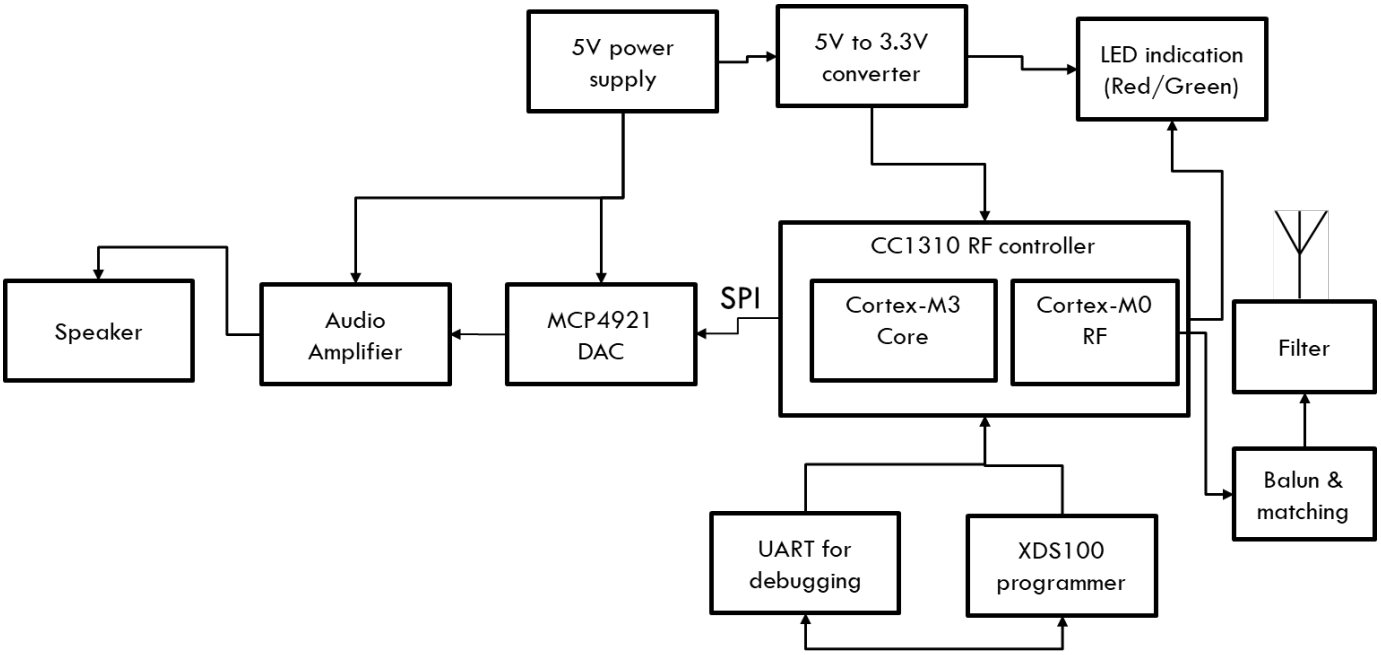

Figure 2.3: Hardware Block Diagram : Receiver

2.6 Transmitter LED Indication

It is used two different color LEDs for indicating the healthiness of communication on transmitter and receiver end. Healthiness is representing by toggling the LED with specific manner. At transmitter end, green LED toggle on each successful transmission of packets during command time. If transmission is fails or abort then RED LED blinks.

2.7 Receiver LED Indication

Healthiness is representing by toggling the LED with specific manner. At Receiver end, RED LED toggle on each successful reception of packets during command time. If reception is fails or abort then Green LED blinks.

2.8 RF Controller

CC1310 SimpleLink™ Ultra-Low-Power Sub-1 GHz Wireless MCU is used for the application. The specifications of controllers are:

- **Arm CORTEX M3 processor** with Harvard architecture (separate bus for data and instructions)
- Part code is CC1310F128RGZT
- Flash size is 128kB
- RAM size is: 20 kB
- 1.25 DMIPS per MHz.
- Total GPIOs are 30 numbers and total pins are 48
- SPI, I2C, UART, RTC interface available
- Current consumption in active mode M3 48 MHz running coremark:2.5 mA (51 μA/MHz)
- High performance interrupt handling for the critical applications.
- Optimized for single cycle flash memory use.
- Separate **Ultra-Low-Power Sensor Controller** which supports 16 bits architecture.
- Internal ADC with 12 bits /200ksps, 8 channel analog mux.
- Current consumption in active mode sensor controller at 24 MHz: 0.4 mA + 8.2 μA/MHz
- Dedicated **Radio Controller (Cortex®-M0)** handles low-level RF protocol commands that are stored in ROM or RAM.
- Receiver Sensitivity for long range mode, –110 dBm at 50 kbps
- Selectivity ($\pm$100 kHz): 56 dB
- Blocking Performance ($\pm$10 MHz): 90 dB
- Programmable Output Power up to +15 dBm
- Radio supports: proprietary, wireless Mbus, IEEE 802.15.4g
- Single-Ended or Differential RF Interface (balanced or unbalanced)

- Supports wide range of data rate: 625bps to 4 Mbps.
- Wide range of modulation formats:
  - Multi-label FSK and MSK.
  - On-Off Key (OOK)
  - Coding gain support

2.9 Communication Interface:

2.9.1 Communication Interface On Transmitter:

- Between sensor controller and main controller M3: Sensor cobtroller is doing ADC sampling and providing the data to main controller over DMA. For memory swapping after getting 100 samples of data, ping pong mechanism is used.
- Between main controller M3 and RF controller M0: After frame preparation, main controller provides the data to RF controller over DMA so that main controller can work simultaneously for other activities.

2.9.2 Communication Interface On Receiver:

- Between DAC MCP4921 and main controller M3: Main controller sends the digital data to DAC4921 on SPI interface. A timer is used for triggering the SPI sending task in a particular interval as per sampling rate.
- Between main controller M3 and RF controller M0: After data reception, main controller receives the data from RF controller over DMA so that RF controller can work simultaneously for other activities.

2.10 Design Rationale

2.10.1 Pros Of Architecture:

- Main controller M3 processor utilization will be reduced as RF communication handling is offloaded from the RF controller.
- Architecture distributes the processing load among three micro controllers.
- Number of interrupts to be processed will be distributed among three controllers which will make system simpler.

2.10.2 Cons Of Architecture:

- Communication response time latency of 25 milliseconds will be added while modifying parameters from transmitter to receiver.
- Handshaking will be required between transmitter and receiver.

2.11 Details Of Interfaces

2.11.1 SW-SW Interfaces:

- Transmitter: ADC call-back - ADPCM encoder
- Transmitter: ADPCM encoder – RF data preparation
- Receiver: RF data reception – ADPCM decoder
- Receiver: ADPCM decoder – DAC SPI data transmission

2.11.2 SW-HW Interfaces

- Transmitter: Signal conditioning – ADC
- Transmitter: Controller M0 – balun circuit
- Transmitter: JTAG programmer - Controller M3
- Receiver: Balun circuit - Controller M0
- Receiver: Controller M3 – DAC MCP4921
- Receiver: JTAG programmer - Controller M3
- Controller M3- LED indication

2.11.3 HW-HW Interfaces

- Transmitter: Signal conditioning – microphone
- Transmitter: Antenna impedance matching circuit – balun circuit
- Transmitter: Antenna impedance matching circuit – Antenna
- Receiver: Antenna impedance matching circuit – balun circuit

- Receiver: Antenna impedance matching circuit – Antenna
- Receiver: Audio amplifier – DAC MCP4921
- Receiver: Audio amplifier – low pass LC filter.

Chapter 3

Design Strategy and Calculation

In this section , we will study the different design consideration and calculation.

3.1 Structural Design Choice

From start of project two design choices were made that would influence the structure of whole application. Using these choices, we could make software robust and re-usable.

3.1.1 Modular Programming

The first choice was to use modular programming techniques. Modular programming divides an application into modules with each module handling a single function within the application. Each module consists of an interface and an implementation. The interface allows other modules to interact with the implementation while all the details of the implementation are hidden from the rest of the application.

3.1.2 Processor Utilization

Another basic design choice was to utilize task utilization in effective way. So, for this purpose, we have implemented ADC sequential sampling in effective way. Ping pong is a specific implementation on ADC DMA peripheral task. Two separates memory are used for buffering the quarter cycles of samples and swapping each time. So, there is no involvement of processor for capturing or buffering the ADC samples in each sampling time.

3.2 Design Calculations

3.2.1 Signal Conditioning On Transmitter

During the prototype development, testing is done electret microphone because of its good frequency response and small size. In electret microphone, due to variation in capacitance between two electrodes, the across voltage changes with fix charge. This voltage is passed through coupling capacitor C63 and gain resistor R76. A DC voltage shift is also provided added for converting bipolar output to uni-polar.

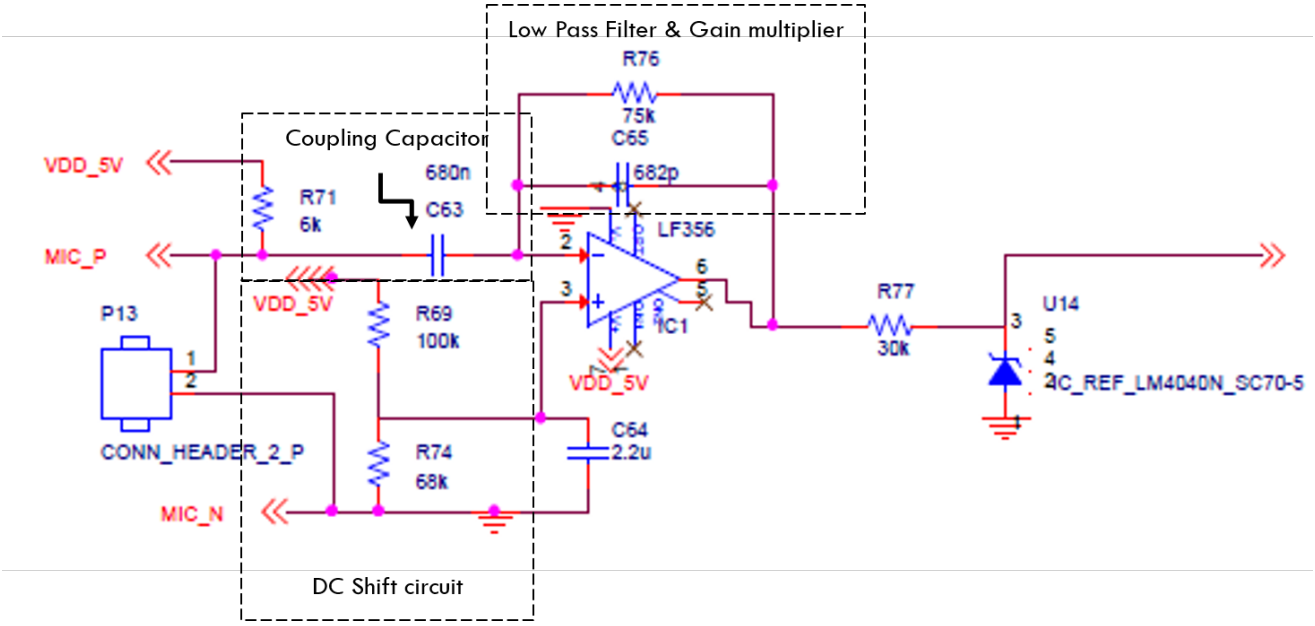

Figure 3.1: Schematics: Signal Conditioning On Transmitter

Overall interface is divided in below sections:

3.2.1.1 Mic Interface

In overall designing, we have considering below specifications of MIC as per table number 3.1

Sensitivity of microphone in mV

$$= 10^{\frac{dBV}{20}} = 10^{\frac{-35}{20}} = 17.78mV/Pascal$$

The output current per Pascal of air pressure

$$= \frac{SensitivityinmV}{Impedance} = \frac{17.78}{2.2} = 8.08\mu A/Pascal$$

Parameters	Value (Units)
Sensitivity	-35±4dBV
Standard Operating Voltage (V_{mic}	2 Vdc
Current Consumption (Max) (I_s)	0.5 mA
Impedance	2.2 kOhm
Signal to Noise Ratio (Min)	68dB

Table 3.1: Mic Specification For Design Calculation

microphone output current I_{mic} for 2Pa Air pressure = $8.08 \times 2 = 16.16\mu A$

3.2.1.2 Low Pass Filter And Gain Multiplier

Now if considering maximum swing across DC offset is $\pm$ $1.228V_{ac}$ then
R76 = $1.228 \div I_{mic}$ = **75 kOhm**
C65 is used to compensates the parasitic capacitance at the op amp inverting input which can be cause instability. Capacitor C65 also forms a pole with resistor R75 in the response of the pre-amplifier. The frequency of this pole must be such that it should not affect the microphone transfer function within the audible bandwidth. For this design, a response deviation of -0.1dB at 500Hz is acceptable.

The location of the pole can be calculated using the relative gain at 500Hz
$f_p = \frac{f}{\sqrt{(\frac{G_0}{G_t})^2-1}} = \frac{500}{\sqrt{(\frac{1}{0.989})^2-1}}$ = 3343 Hz
$C65 = \frac{1}{2\times\pi\times3343\times R76} = 634pF$ = 682 pF (nearest value)

3.2.1.3 Biasing and Coupling Parameters

V_{cc} = 5V
Biasing resister **R71** = $\frac{V_{cc}-V_{mic}}{I_s}$ = **6kOhm**
For high pass filter, assumes that cutoff frequency is 50Hz
$C63 = \frac{1}{2\times\pi\times50\times R71} = 530nF$ = 680 nF (nearest value)

3.2.1.4 DC Offset Parameter

If R69 = 100 kOhm then R74 will be 68k Ohm for getting DC offset of 2.02 V at positive terminal of op-amp.
C64 is used for compensation of thermal noise created by voltage divided circuit.
for C64 = 2.2 μF cutoff frequency $f_c = \frac{1}{2\times\pi\times2.2\times(R74||R69)}$= 1.7Hz

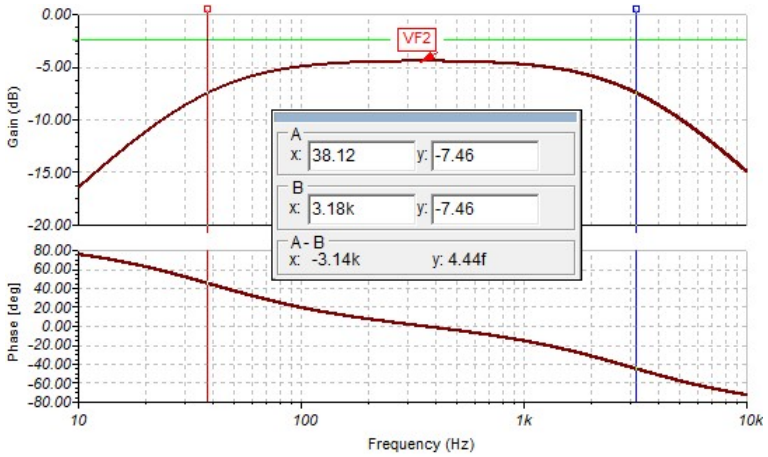

Figure 3.2: Mic Interface Signal Conditioning Circuit-Simulation In TINA-TI Tool

Simulation results are taken at output of coupling capacitor and outputs of op-amp which is forming a band pass filter in range of 40Hz to 3.3kHz. zener diode at output is connected to protect ADC pin for higher voltage.

3.2.2 MCP4921 DAC Interface On Receiver

DAC Specifications:

- 12-Bit resolution
- Rail-to-Rail output
- SPI interface with 1MHz clock
- Fast settling time of 4.5 μs
- Selectable unity or 2x gain output
- External VREF input: 3.3V
- 2.7V to 5.5V single-supply operation: 5V

For digital to analog conversion, MCP4921 is used. DAC includes rail-to-rail output amplifiers, reference buffers, input amplifier and reset circuit. For interfacing with micro-controller is done using SPI protocol. we are operating DAC on 5V and reference voltage is 3.3V.

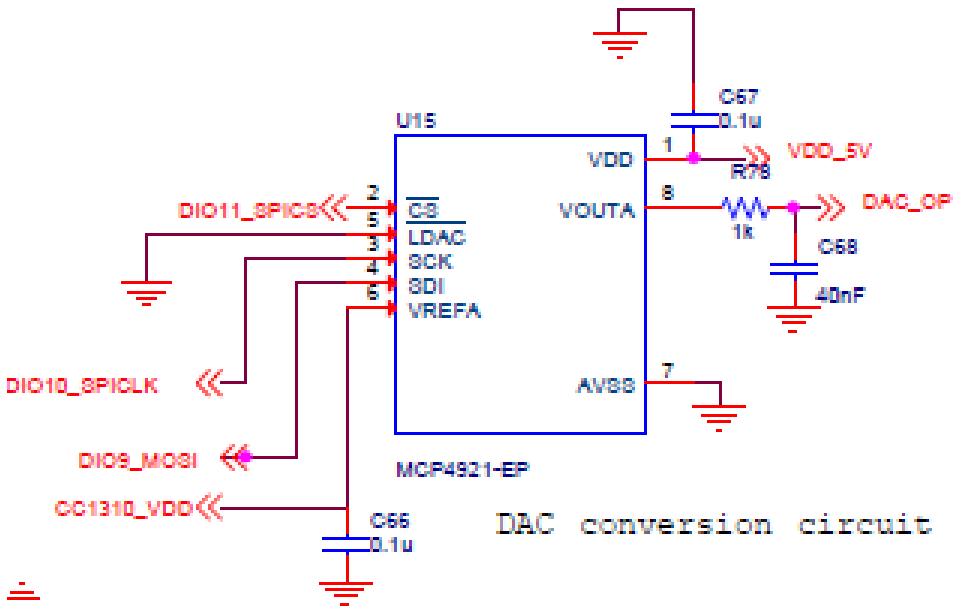

Figure 3.3: DAC MCP4921 Interface

DAC output volatge $V_{out} = \frac{V_{REF} \times G \times D_n}{2^n}$
here G= 1 , D_n = input digital value V_{REF} = 3.3V , n = 12 bits resolution

During the SPI communication, data bit rate is using 1Mbps and writing the 16 bits of data (12 bits sampled value + 4 bits header) in each sampling interval with sampling rate 8ksps.

3.2.3 Speaker Interface On Receiver

The output of DAC MC4921 passes though a class D audio amplifier and a LC filter. The designing of class D amplifier is explaining in below steps.
class D amplifier provides high efficiency, dissipates much less power then any of the other class A,B,C,and AB amplifier. Its output stage switches between positive and negative voltage to produce a train of voltage pulses. the waveform is not harmful for power dissipation, because of zero current on output transistor when not swishing.
Design calculation of single ended class D amplifier with LC filter is describing below:
AS class D amplifier, TPA2005D1 1.4-W MONO Filter-Free Class-D Audio Power Amplifier, is used.
For gain G = 1: R79 = R80 = 300k / G = 300k ohm
now if we assumed that we are designing a high pass filter of cutoff frequency 75 Hz, then

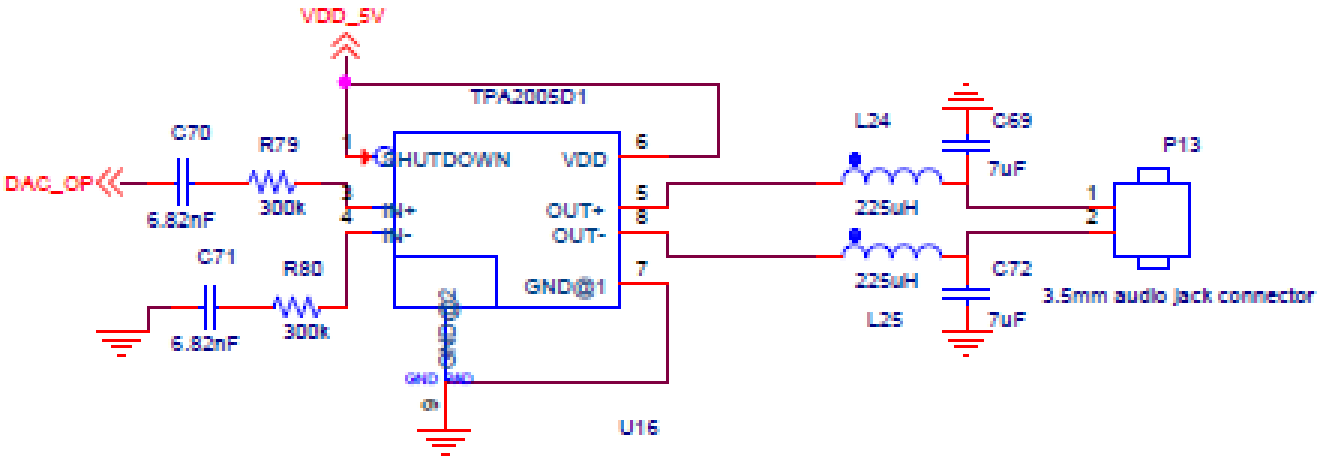

Figure 3.4: Speaker Interface With Audio Amplifier

$C70 = C71 = \frac{1}{2\times\pi\times75\times R79}$= 6.82 nF

Now we will discuss regarding designing calculation of LC filter. during designing we are assuming impedance of speaker (R_{BTL}) is 4 ohm and low pass filter cutoff frequency is 4000Hz so load impedance R_L = R_{BTL} , f_c = 4000 Hz

$L24 = L25 = \frac{R_L\times\sqrt{2}}{2\times\pi\times4000} = 225\mu H$

$$C72 = C69 = \frac{1}{2\times\pi\times4000\times R_L\times\sqrt{2}} = 7\mu F$$

3.2.4 RF Design Calculation

In this section we will see design constraints as well as RF frame structure with command time. In suggested designing we are targeting the audio frequency range 75Hz to 4000 Hz. so as per nyquist criteria minimum possible sampling rate is 8000 sps. At transmitter end, the cortex M3 processor samples the ADC input and store the 12 bits sampling value in DMA buffer. the predefined size of DMA buffer is 100. Means, after capturing 100 sampled values , DMA provides interrupt to M3 processor. M3 processor encodes the stored samples using ADPCM technique and send 50 bytes coded values to the Receiver via RF. Here predefined bit rate of transmiter modulated signal via RF is 50 kbps.

ADC sampling rate = f_s = 8 ksps, DMA buffer size = 100, ADC resolution = 12 bits

ADPCM compressed data = 12 bits to 4 bits information

DMA interrupt time = DMA buffer size ÷ ADC sampling rate = 12.5 msec.

Size of total transmiited frame = data size + overheads , data size = 52 byts + 14 bytes = 66 bytes

command time for sending 66 bytes of data over RF = (Size of total transmitted frame X 8) ÷ bit rate = (66 X 8) ÷ 50000 = 10.56 msec.
Notice that RF data transmit time is less then the sampling time. so remaining time we can use for ADPCM encoding algorithm execution.

Chapter 4

Design Implementation

In this section , we will study the Implementations and programming details of audio transmission and reception over RF.Here will see the detailing of individually modules and system integration.

4.1 Firmware Architecture

CC1310 RF controller is used for firmware programming for transmitter and receiver. CC1310 is the **Simple-Link** ultra low power sub-1 GHz wireless micro-controller .Multi-core architecture provides the advantage of multiple application processing simultaneously. Different interfaces with internal or external peripherals are detailing in following section. Figure 4.1 shows the software architecture block diagram in different layers.

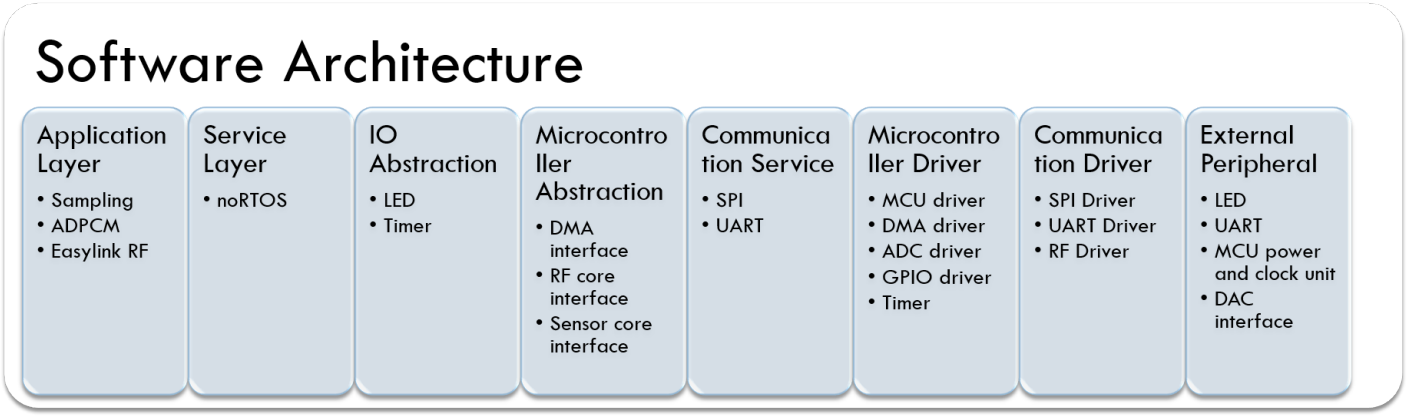

Figure 4.1: Product Software Architecture

4.1.1 Application Layer

Application layer is doing different functionality in transmitter and receiver.At transmitter end, Buffering the ADC input samples, compression technique implementation (ADPCM encoder) and RF data parsing and command handling via easy-link API are handling by application layer. Similarly on receiver end RF data parsing and command handling via easy-link API , decompression technique implementation (ADPCM decoder) and Sampling for DAC output are handled by application layer.

4.1.2 Service Layer

There is trade-off between memory utilization,task scheduling and processor utilization. In out application, main processor M3 is doing functionality of sampling buffering and ADPCM technique.In case of complex scheduling architecture, RTOS is a better option. But in case of only few activity , noRTOS is also a better choice . In out implementation, i have used noRTOS platform for avoiding multiple scheduling latency between tasks and overheads, also following linear implementation scheme. Same platform we have used for transmitter and receiver.

4.1.3 IO Abstraction

Input output activity on processor abstraction layer is also important functionality where we are using timer for sampling per-pose on receiver end as input and diagnosis LED's controlled mechanism as output in both transmitter and receiver.

4.1.4 Micro-controller Abstraction

As discussed earlier,CC1310 RF controller is used in design. CC1310 is a multi-core processor which have separate core for sensing,RF and main application.Here main MCU (M3) is running on frequency of 48 Mhz. Handshaking between different core is handled by direct memory access (DMA).

4.1.5 Communication Service

To interfacing between CC1310 and MCP4921 DAC, SPI is used for data transmitting on receiver end. SPI is running on frequency 1 Mhz. DAC sampling rate is 8000sps. For RF testing and boot-loading purpose, UART interface is used on transmitter and receiver end.

4.1.6 Micro-controller Driver

Drivers are used for interfacing the application with physical layer. ADC driver is used for interfacing the ADC with micro-controller. Here M3 core need to initialize with ADC configuration settings to the sensor core so that sensor core is able to accomplish the sampling task minimum intervention of M3 core.

DMA driver is also as important as ADC driver. A ping pong mechanism is used during interfacing with DMA. Using ping pong mechanism, after capturing first 100 ADC samples dma destination memory swaps with other memory address to avoid data corruption. After frame preparation, M3 core provides the data to RF core also by DMA driver.

For diagnosis and communication health monitoring purpose two RED and GREEN LEDs are used. At transmitter end green led indicates the successive transmission of RF frame and red for aborting the communication. At receiver end red led indicates the successive reception of RF frame and green for aborting or timeout the communication.

Timer is also used for sampling the digital to analog converter with sampling rate 8ksps. In each 125 μ sec , timer is getting overflow and triggers the SPI buffer for sending data to DAC.

4.1.7 Communication Driver

Application of SPI and UART driver is already discussed above section. RF driver configuration is also very important part here. RF configuration files can also be generated using **smart RF studio 7**. Detailing of configuration will be later programming section.

4.1.8 External Peripheral

For diagnosis and communication health monitoring purpose two RED and GREEN LEDs are used. At transmitter end green led indicates the successive transmission of RF frame and red for aborting the communication. At receiver end red led indicates the successive reception of RF frame and green for aborting or timeout the communication.

MCP4921 digital to analog converter is used for converting digital data in to the analog form. 1x and 2x gain multiplication scheme is available .Here we have used gain of 1.DAC includes rail-to-rail output amplifiers, reference buffers, input amplifier and reset circuit. For interfacing with micro-controller is done using SPI protocol.

we are operating DAC on 5V and reference voltage is 3.3V.
DAC output volatge $V_{out} = \frac{V_{REF} \times G \times D_n}{2^n}$
here G= 1 , D_n = input digital value V_{REF} = 3.3V , n = 12 bits resolution

During the SPI communication, data bit rate is using 1Mbps and writing the 16 bits of data (12 bits sampled value + 4 bits header) in each sampling interval with sampling rate 8ksps.

UART is using for debugging and boot-loading purpose. ADC, DAC and amplifiers are operating with power supply on 5V and CC1310 processor is operating on 3.3 V. So for converting 5V to 3.3 v a DC to DC converter is used.

4.2 Block Diagram

Product block diagram of transmitter and receiver is shown in following section.

4.2.1 Audio Transmitter

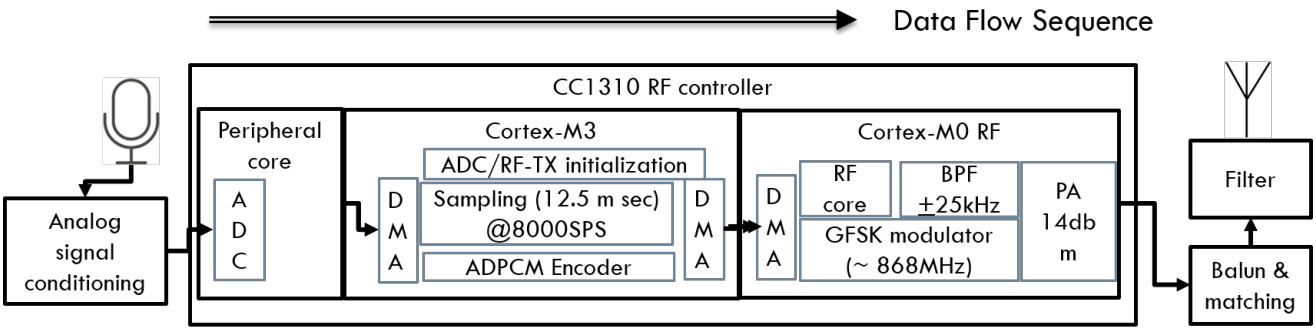

Figure 4.2: Audio RF Transmitter

Function block diagram of audio transmitter over RF is shown in figure 4.2. This module consists different sub-modules like microphone, analog signal conditioning, RF multi-core CC1310 micro-controller, balance to unbalance circuit, matching circuit and antenna.This module is used to convert the voice signal into the digital form and transmit compressed dig-ital data via RF antenna to the receiver.

Mic is converting acoustic voice signal into the electrical signal. This electrical signal behaves as a input signal for analog signal conditioning circuit.Electrical signal passes through coupling capacitor which provides the high pass filter of cutoff frequency 50Hz. further output passes through

Preamble	Syncword	Length	Address	Sequence number	ADPCM code	ADPCM init	CRC
4 bytes	4 bytes	1 byte	1 byte	2 bytes	50 bytes	2 bytes	2 bytes

Table 4.1: RF Frame Structure (Parameter Size And Query Length)

another low pass filter of cutoff frequency 3.3kHz with 2V DC offset.

CC1310 sensor core reads adc input signal with reconfigured sampling rateof 8000SPS . In each 125 μ sec time interval , sensor core triggers the ADC for conversion starts. After conversion ending ADC is copying the into the DMA buffer. size of DMA is also predfined as 100. so after capturing 100 samples, DMA provides interrupt to main MCU M3.

After getting callback interrupt, MCU M3 stores the bunch of 100 samples and passes though adaptive delta pulse code modulation encoder. encoder converts the 16 bits of data in 4 bits binary format and compressed the 200 bytes into the 50 bytes of data.

During conversion of delta modulation, initialization part is also important for matching and synchronization the input signal with decoded signal.Previous sample value and quantizer step size are two parameters in initialization part. We store the initialization value in two bytes of data. these two bytes are appended with decoded 50 bytes of data. So total data size becomes 52 bytes.

Table number 4.1 shows the RF query format.
Preamble: Preamble byte is a alternating sequence of 0 and 1 used for automatic gain control.

syncword: syncword field is require to synchronize the receiver with transmitter,determines the actual start of the message.
Length:In Simple packet format, the size of length is 8 bits long.Length is the part of header

Address:Address is the part of header of size 1 byte. It can be filter up to 2 address in simple packet format.

Sequence number:Sequence number is the part of payload which has size of 2 bytes. maximum possible payload size is 2 bytes in case of simple packet format.

ADPCM code:ADPCM code is part of payload. Compressed encoded code of 50 bytes is the part of this field. This is the application dependent field.

ADPCM init:This field consist of two byte of data. In first byte, scaled information of previous sampled value is stored. In next byte, quantization step size index information is stored.This is also part of payload.

CRC: It is the two byte polynomial checksum information for transmitting frame.

RF data sequencing will be discuss in Easylink API reference section.After packet preparation main MCU M3 triggers the RF core M0 for sending the data frame over RF.
CC1310 RF core have soft provision to configure frequency deviation (used $\pm$25kHz),Low noise amplifier with good sensitivity (110 dbm) and GFSK modulator for long range communication.

Output signal further transmit to antenna via balance to unbalance and impedance matching circuit.

4.2.2 Audio Receiver

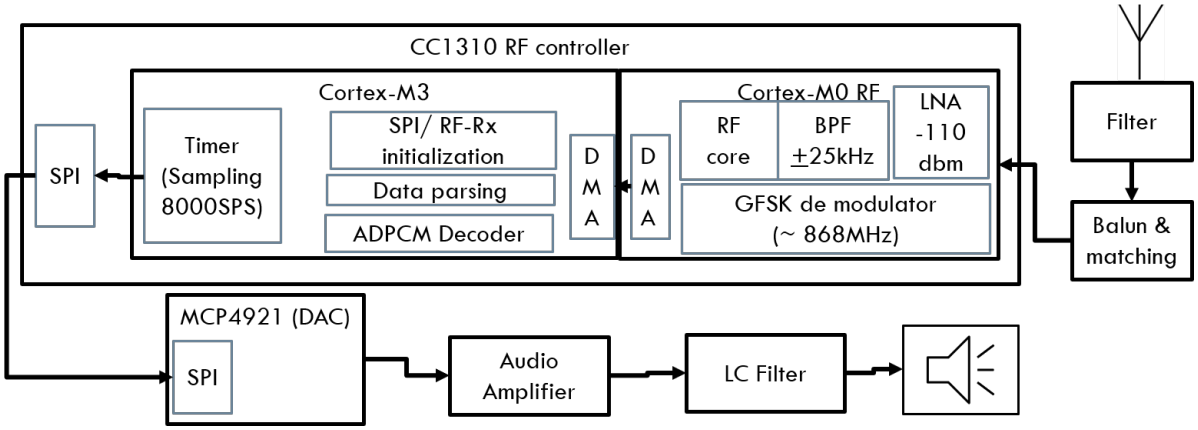

Figure 4.3: Audio RF Receiver

Function block diagram of audio Receiver over RF is shown in figure 4.3.This module consists different sub modules like digital to analog converter, audio amplifier, filter, speaker, RF multi-core CC1310 micro-controller, unbalance to balance circuit, matching circuit and antenna. This module is used to receive RF signal sent by transmitter, validate, decompressed the digital data and sent to the DAC. DAC converts the digital data into the analog form and passes to class D audio amplifier. After filtering the output of amplifier, it goes to speaker as final output. The details module designing need to be discuss in the coming section.

After getting RF callback interrupt, MCU M3 stores the bunch of 100 samples encoded values and passes though adaptive delta pulse code modulation decoder. decoder converts the 4 bits of data in 12 bits binary format and decompressed the 50 bytes into the 200 bytes of data.

During conversion of delta modulation, initialization part is also important for matching and synchronization the input signal with decoded signal.Previous sample value and quantizer step size are two parameters in initialization part. In receive data payload, We will parse these initialization values and pass to the adpcm decoder algorithm before decoding.

Now CC1310 sends these decoded sampled values to the MCP4921 DAC through SPI communication. Sampling rate of DAC is 8000SPS. In each 125 μ sec time interval , timer triggers the M3 core for conversion starts. M3 core processor copy the sample value into the SPI buffer and send it.

Table number 4.1 shows the RF query format.CC1310 RF core have soft provision to configure frequency deviation (used $\pm$25kHz),Low noise amplifier with good sensitivity (110 dbm) and GFSK modulator for long range communication.

For diagnosis purpose, RED and GREEN LED's are used for indicating the status of communication.If data is received over RF successfully then red LED blinks otherwise green.

4.3 Data Flow Diagram

4.3.1 Audio Transmitter

Audio RF data flow diagram is shown in figure 4.4. After power recycle, micro-controller initialize internal RAM and other peripherals before communication. Mic is converting acoustic voice signal into the electrical signal. This electrical signal behaves as a input signal for analog signal conditioning circuit.

CC1310 sensor core reads adc input signal with reconfigured sampling rate of 8000SPS . In each 125 μ sec time interval , sensor core triggers the ADC for conversion starts. After conversion ending ADC is copying the into the DMA buffer. size of DMA is also predefined as 100. so after capturing 100 samples, DMA provides interrupt to main MCU M3.

After getting callback interrupt, MCU M3 stores the bunch of 100 samples and passes though adaptive delta pulse code modulation encoder. encoder converts the 16 bits of data in 4 bits binary

format and compressed the 200 bytes into the 50 bytes of data.

During conversion of delta modulation, initialization part is also important for matching and synchronization the input signal with decoded signal.Previous sample value and quantizer step size are two parameters in initialization part. We store the initialization value in two bytes of data. these two bytes are appended with decoded 50 bytes of data. So total data size becomes 52 bytes.

After packet preparation,main MCU M3 triggers the RF core M0 for sending the data frame over RF. For diagnosis purpose, RED and GREEN LED's are used for indicating the status of communication.If data is transmitted over RF successfully then green LED blinks otherwise red.

4.3.2 Audio Receiver

Audio RF data flow diagram is shown in figure 4.5. After power recycle, micro-controller initialize internal RAM and other peripherals before communication.This module consists different sub modules like digital to analog converter, audio amplifier, filter, speaker, RF multi-core CC1310 micro-controller, unbalance to balance circuit, matching circuit and antenna.

For diagnosis purpose, RED and GREEN LED's are used for indicating the status of communication.If data is received over RF successfully then red LED blinks otherwise green.

After getting RF callback interrupt, MCU M3 stores the bunch of 100 samples encoded values via data parsing and passes though adaptive delta pulse code modulation decoder. decoder converts the 4 bits of data in 12 bits binary format and decompressed the 50 bytes into the 200 bytes of data.

During conversion of delta modulation, initialization part is also important for matching and synchronization the input signal with decoded signal.Previous sample value and quantizer step size are two parameters in initialization part. In receive data payload, We will parse these initialization values and pass to the adpcm decoder algorithm before decoding.

Now CC1310 sends these decoded sampled values to the MCP4921 DAC through SPI communication. Sampling rate of DAC is 8000SPS. In each 125 μ sec time interval , timer triggers the M3 core for conversion starts. M3 core processor copy the sample value into the SPI buffer and send it.

Table number 4.1 shows the RF query format.CC1310 RF core have soft provision to configure frequency deviation (used $\pm$25kHz),Low noise amplifier with good sensitivity (110 dbm) and GFSK modulator for long range communication.

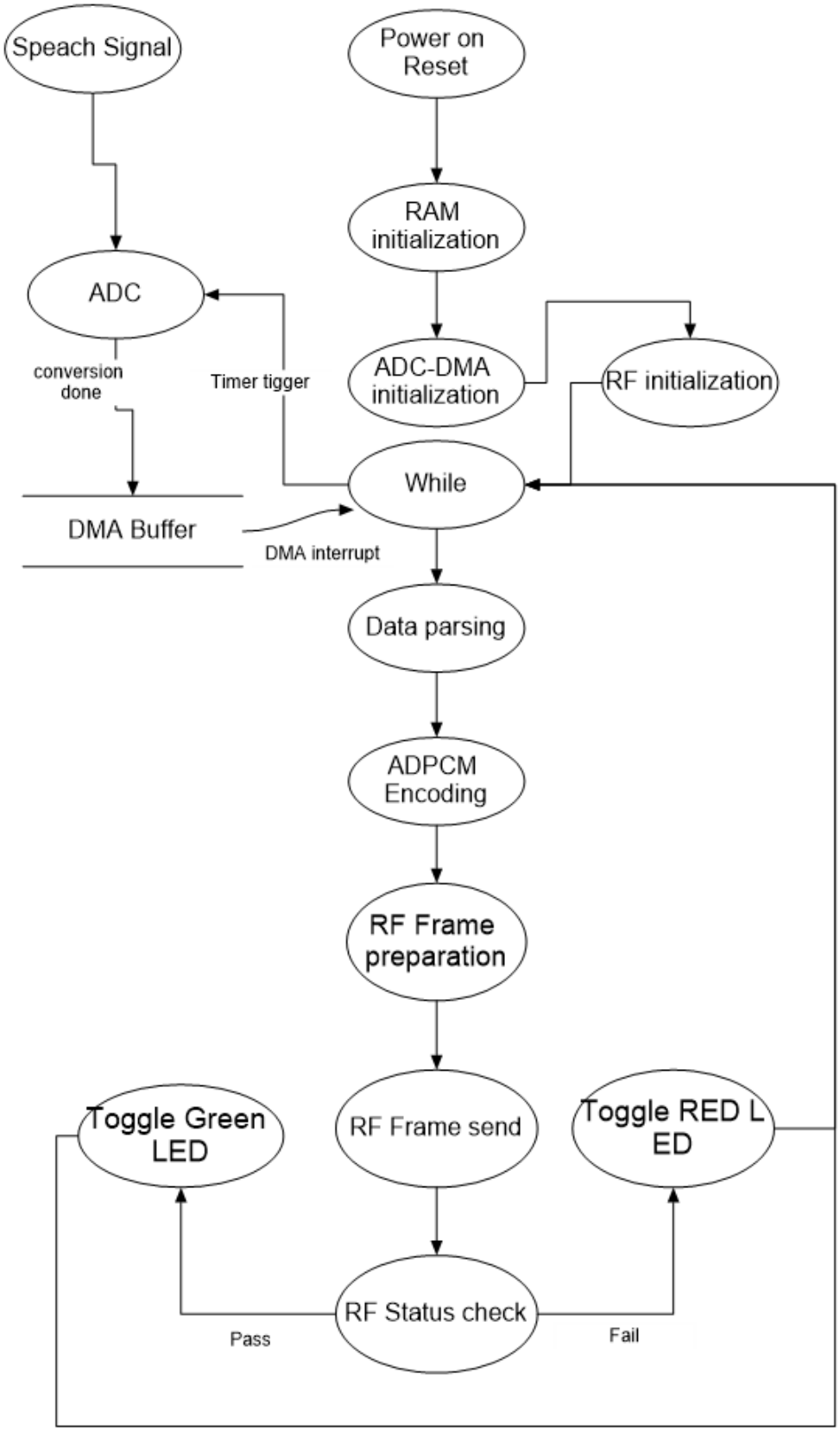

Figure 4.4: Audio RF Data Flow Diagram:Transmitter

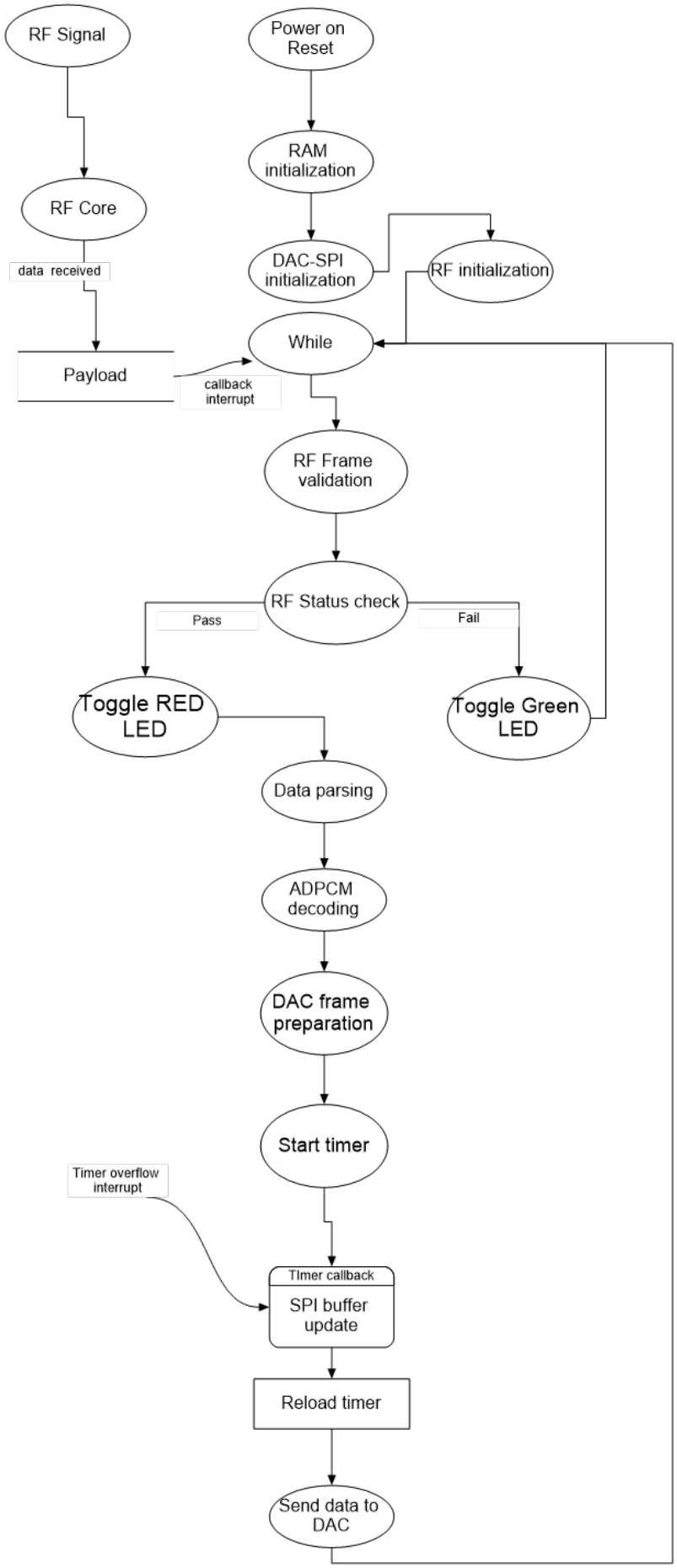

Figure 4.5: Audio RF Data Flow Diagram:Receiver

4.4 ADPCM Technique

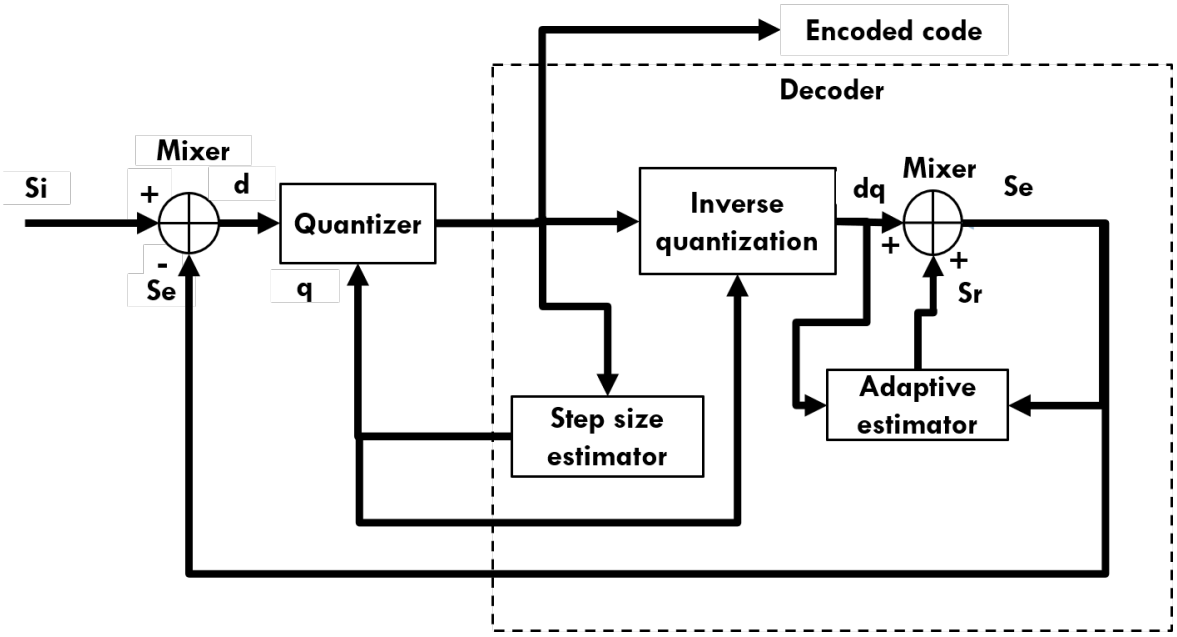

Figure 4.6: Adaptive Delta PCM Technique

ADPCM[9] is the technique which is based on the concept of delta modulation. In this technique input signal is passes from a mixer where difference is calculated between input signal and predicted signal. Predicted signal is nothing but is only quantized form of previous input signal. The difference is passes as encoded signal.

The beauty of this concept is adaptability of quantization steps[10]. So for this purpose, value of encoded code is decide the index[10] of quantization level. If the value is higher or in increasing order then quantization step size[10] will also be higher.

Inverse quantization[9] and adaptive predictor[9] are used for estimating the predicted samples from encoded code. The only difference is that here we need to add the predicted samples in inverse quantized value for getting the predicted value.Figure 4.7 and figure 4.8 is input MATLAB simulated input and output signal which is passed through algorithm.

In transmitter, after getting callback interrupt, MCU M3 stores the bunch of 100 samples and passes though adaptive delta pulse code modulation encoder[10]. encoder converts the 16 bits of data in 4 bits binary format and compressed the 200 bytes into the 50 bytes of data.

During conversion of delta modulation, initialization part is also important for matching and

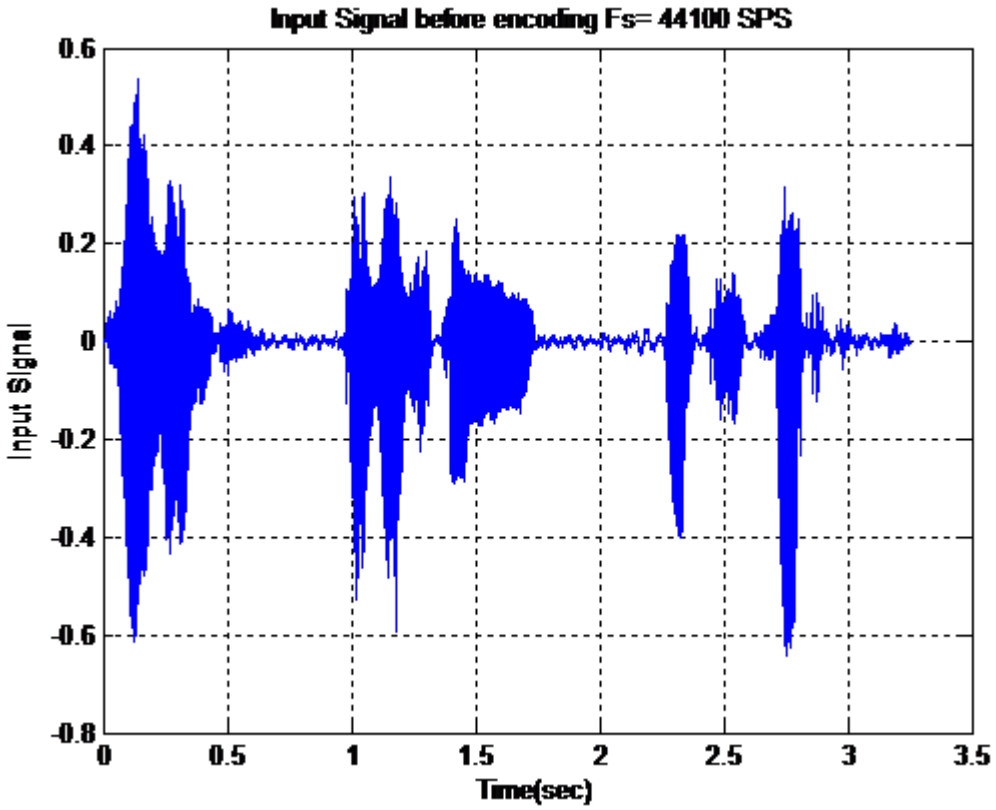

Figure 4.7: ADPCM:Input Signal Which is Passed Through Algorithm

synchronization the input signal with decoded signal.Previous sample value and quantizer step size are two parameters in initialization part. We store the initialization value in two bytes of data. these two bytes are appended with decoded 50 bytes of data. So total data size becomes 52 bytes.

At receiver end after getting RF callback interrupt, MCU M3 stores the bunch of 100 samples encoded values via data parsing and passes though adaptive delta pulse code modulation decoder[10] . In receive data payload,We will parse initialization values of adpcm and pass to the adpcm decoder algorithm before decoding. decoder converts the 4 bits of data in 12 bits and decompressed the 50 bytes into the 200 bytes of data.

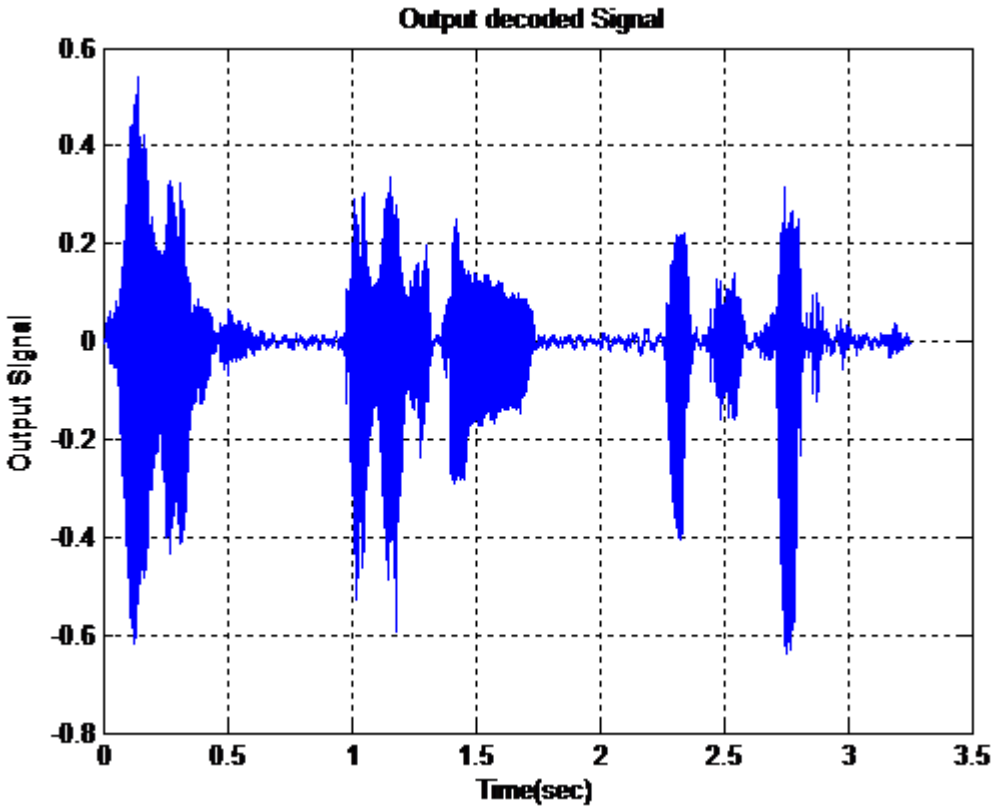

Figure 4.8: ADPCM:Output Signal Which Is Decoded By Algorithm

4.5 Current Consumption

In this section , we will calculate the power consumption using CC1310 micro-controller.

Table4.2 showes that CC1310 current consumption of individual peripherals. Total current is coming approximate 32.9 mA.

Parameters	Value (Units)
Main MCU (Active@48Mhz)	1.2mA + 25.5μA/Hz
Radio Rx	5.5mA
RF core	0.237mA
uDMA	0.130mA
Timer	0.113mA
I2C	0.012mA
SPI	0.093mA
UART	0.164mA
ADC	0.750mA
Radio Tx (VDDR=1.95V) , 14dbm o/p power GFSK	23.5mA

Table 4.2: CC1310 : Current Consumption

4.6 Component List And Costing

Here we will list the important components and their costing. Table4.3 shows the list of electronics component used for transmiiter end. Total cost is coming approximately 10.38$.

Table4.4 shows the list of electronics component used for receiver end. Total cost is coming approximately 12.04$.

Please note that mentioned price is referred by DigiKey for 500 to 2000 quantity. Price data has been recorded on 21-may-2020.

Quantity	Parts	Cost ($)
1	LS L296-P2Q2-1 RLED	0.11821
1	LS LPL296-J2L2-25 GLED	0.11821
1	BLM18HE152SN1 Inductor	0.06619
1	LF356 Opamp	0.297
1	CC1310F128RGZT	4.8564
1	TPS79601DR	2.3403
1	LM4040N	0.5292
1	32.768kHz Xtal	0.77
1	24MHz Xtal	0.77
27	capacitors	0.04131
11	Resistors	0.01474
6	Indicators	0.4662

Table 4.3: Transmitter : Component List

Quantity	Parts	Cost ($)
1	LS L296-P2Q2-1 RLED	0.11821
1	LS LPL296-J2L2-25 GLED	0.11821
1	BLM18HE152SN1 Inductor	0.06619
1	TPA2005D1	0.35
1	CC1310F128RGZT	4.8564
1	TPS79601DR	2.3403
1	MCP4921EP	1.49
1	32.768kHz Xtal	0.77
1	24MHz Xtal	0.77
1	3.5mm audio jack connector	0.4832
31	capacitors	0.0474
9	Resistors	0.0121
8	Indicators	0.6216

Table 4.4: Receiver : Component List

4.7 Easylink API Reference

Easylink API is the library function which is used for RF interface data transmission and reception. Before using the EasyLink API , following steps[11] need to be take care.

- To initialises and opens the RF driver and configuration the modulation scheme , initially we need to call the EasyLink_init().
- Rx and Tx are independent activities.

4.7.1 API For Transmitting Operation

- Initially check the current RF status , any pending async command,data length, busy or free. in case of error, it abort the current operation.
- Prepare transmitting packet and calculate command time.
- Configure packet in trigger mode and configure start time and end time.
- Call to EasyLink_transmitAsync() as non blocking and post the command.
- check EasyLink_Status_Busy_Error and cancelled with EasyLink_abort() if required.

4.7.2 API For Receiving Operation

- Initially check the current RF status , any pending async command, busy or free. in case of error, it abort the current operation.
- Rx is enabled by calling EasyLink_receiveAsync().
- Entering RX can be immediate or scheduled.
- Call to EasyLink_receiveAsync() as nonblocking and post the command.
- the EasyLink API does not queue messages so calling another API function while in EasyLink_receiveAsync will return EasyLink_Status_Busy_Error
- check EasyLink_Status_Busy_Error and cancelled with EasyLink_abort() if required.

Chapter 5

Firmware Coding

In this chapter we will see firmware coding and programming method.

5.1 Firmware Programming

There are two separate module are available. 1. Transmitter 2. Receiver
During programming for RF interface RF Simplelink proprietary TI library[1] is used.Details explanation of code is showing in next section.

5.1.1 Transmitter Programming

At power recycle MCU M3 initializes all peripherals. Following section is showing ADC initialization.

```
1   ///*ADC Initialization */
2    ADCBuf_init();
3    ADCBuf_Params_init(&adcBufParams);
4    adcBufParams.callbackFxn = adcBufCallback;
5    /* Set up an ADCBuf peripheral in ADCBuf_RECURRENCE_MODE_CONTINUOUS */
6    adcBufParams.recurrenceMode =  ADCBuf_RECURRENCE_MODE_CONTINUOUS;
7    adcBufParams.returnMode = ADCBuf_RETURN_MODE_CALLBACK;
8    adcBufParams.samplingFrequency = 8000;
9    adcBuf = ADCBuf_open(Board_ADCBUF0, &adcBufParams);
10
11   /* Configure the conversion struct */
12   continuousConversion.arg = NULL;
13   continuousConversion.adcChannel = Board_ADCBUF0CHANNEL0;
14   continuousConversion.sampleBuffer = sampleBufferOne;
15   continuousConversion.sampleBufferTwo = sampleBufferTwo;
16   continuousConversion.samplesRequestedCount = ADCBUFFERSIZE;
17   if (adcBuf == NULL){
```

```
        /* ADCBuf failed to open. */
        while(1);
    }

    /* Start converting. */
    if (ADCBuf_convert(adcBuf, &continuousConversion, 1) !=
        ADCBuf_STATUS_SUCCESS) {
        /* Did not start conversion process correctly. */
        while(1);
    }
```

Radio parameters are initialized as per sequence given below.

```
#ifdef RFEASYLINKTX_ASYNC
    /* Reset the sleep period counter */
    txSleepPeriodsElapsed = 0;
    /* Set the transmission flag to its default state */
    txDoneFlag = false;
#endif //RFEASYLINKTX_ASYNC

    // Initialize the EasyLink parameters to their default values
    EasyLink_Params easyLink_params;
    EasyLink_Params_init(&easyLink_params);

    /*
     * Initialize EasyLink with the settings found in easylink_config.h
     * Modify EASYLINK_PARAM_CONFIG in easylink_config.h to change the default
     * PHY
     */
    if (EasyLink_init(&easyLink_params) != EasyLink_Status_Success){
        while(1);
    }
```

In following section , variables which are using in adpcm algorithm,initialized. while(1) is a infinite loop which use for preparation of radio frame and sending it.

```
    init_adpcm(); /* ADPCM initialization*/
    while(1) {
        if(RF_Tx_Fn_f)
        {
            RF_Tx_Fn();
            RF_Tx_Fn_f = 0;
        }

    }// while
```

Below is a call back function which interrupt comes after filling ADC DMA buffer of 200 bytes of data.Using ping pong mechanism, we parse the data and call the adpcm encoder algorithm[10].

```
void adcBufCallback(ADCBuf_Handle handle, ADCBuf_Conversion *conversion,
    void *completedADCBuffer, uint32_t completedChannel)
{
    /* Adjust raw ADC values and convert them to compressed form */
    ADCBuf_adjustRawValues(handle, completedADCBuffer, ADCBUFFERSIZE,
    completedChannel);

    if(completedADCBuffer==&sampleBufferOne)
      {
        ADPCMEncoder(&sampleBufferOne[0],&mVoltBuffer[0],int16_prevSampleForRF
    , ADCBUFFERSIZE);
        RF_Tx_Fn_f = 1;
      }
      else if(completedADCBuffer==&sampleBufferTwo)
      {
          ADPCMEncoder(&sampleBufferTwo[0],&mVoltBuffer[0],
    int16_prevSampleForRF, ADCBUFFERSIZE);
        RF_Tx_Fn_f = 1;
       }
}
```

below function is use for radio frame preparation and commanding to send via RF core.

```
void RF_Tx_Fn(void)
{
    uint32_t absTime;
    static uint8_t txBurstSize = 0;
    EasyLink_TxPacket txPacket =  { {0}, 0, 0, {0} };

    /* Create packet with incrementing sequence number and random payload */
    txPacket.payload[0] = (uint8_t)(seqNumber >> 8);
    txPacket.payload[1] = (uint8_t)(seqNumber++);
    uint8_t i;
    for (i = 2; i < (RFEASYLINKTXPAYLOAD_LENGTH-2); i++)
    {
        txPacket.payload[i] = mVoltBuffer[i-2];
    }
    txPacket.payload[i] = int16_prevSampleForRF & 0x00ff;
    txPacket.payload[i+1] = (int16_prevSampleForRF>>8) & 0x00ff;

    txPacket.len = RFEASYLINKTXPAYLOAD_LENGTH;

    /*
     * Address filtering is enabled by default on the Rx device with the
```

```
     * an address of 0xAA. This device must set the dstAddr accordingly.
     */
    txPacket.dstAddr[0] = 0xaa;

    /* Add a Tx delay for > 500ms, so that the abort kicks in and brakes the burst */
    if(EasyLink_getAbsTime(&absTime) != EasyLink_Status_Success)
    {
        // Problem getting absolute time
    }
    if(txBurstSize++ >= RFEASYLINKTX_BURST_SIZE)
    {
        /* Set Tx absolute time to current time + 1s */

        txPacket.absTime = absTime ;//+ EasyLink_ms_To_RadioTime(3000);//1000 // commented because not needed delayed data transmission
        txBurstSize = 0;
    }
    /* Else set the next packet in burst to Tx in 100ms */
    else
    {
        /* Set Tx absolute time to current time + 100ms */
        txPacket.absTime = absTime ;//+ EasyLink_ms_To_RadioTime(100);
    }
    /*
     * Set the Transmit done flag to false, callback will set it to true
     * Also set the sleep counter to 0
     */
    txDoneFlag = false;
    txSleepPeriodsElapsed = 0;

    /* Transmit the packet */
    EasyLink_transmitAsync(&txPacket, txDoneCb);

}
```

Next section is the radio command diagnosis part where we check the previous command status and if is sent successfully then related LED is blinked.

```
void txDoneCb(EasyLink_Status status)
{
    if (status == EasyLink_Status_Success)
    {
        /* Toggle LED1 to indicate TX */
        PIN_setOutputValue(pinHandle, Board_PIN_LED1,!PIN_getOutputValue(Board_PIN_LED1));
```

```
    }
    else if(status == EasyLink_Status_Aborted)
    {
        /* Toggle LED2 to indicate command aborted */
        PIN_setOutputValue(pinHandle, Board_PIN_LED2,!PIN_getOutputValue(
    Board_PIN_LED2));
    }
    else
    {
        /* Toggle LED1 and LED2 to indicate error */
        PIN_setOutputValue(pinHandle, Board_PIN_LED1,!PIN_getOutputValue(
    Board_PIN_LED1));
        PIN_setOutputValue(pinHandle, Board_PIN_LED2,!PIN_getOutputValue(
    Board_PIN_LED2));
    }
    txDoneFlag = true;
    txSleepPeriodsElapsed = 0;
}
```

ADPCMEncoder[10] uses for compressing 200 bytes of sampled data in to the 50 bytes coded value.

```
void ADPCMEncoder(uint16_t * p_ui16_inputsample,uint8_t * p_ui8_outsample,
    int16_t prevsample,uint8_t buffer_size)
{
    int8_t  index = 0,code,int8_evensample,int8_lock = 0 ;
    uint8_t  ui8_count = 0,uint8_codeTemp = 0 ,ui8_count_codeindex = 0;
    int16_t  predsample,diff,step,tempstep,diffq,sample;


    int8_evensample = 0; // use for combined upper and lower code nibble in
    one byte
    for(ui8_count = 0; ui8_count < buffer_size ; ui8_count ++) // loop till
    100 samples
    {
        // lock initial index as well as sample value for synchronization with
     transmitter during decoding
        if(int8_lock == 0)
        {
            int8_lock = 1;
            int16_prevSampleForRF = adpcm_state.previndex ;
            int16_prevSampleForRF = (int16_prevSampleForRF <<8)+ (uint8_t)(
    adpcm_state.prevsample >>4) ;
        }

        predsample = adpcm_state.prevsample; // buffer previous sampled value
```

```
20        index = adpcm_state.previndex;       // index for step size
21        step = StepSizeTable[index];         // quantization step size
22
23        sample = p_ui16_inputsample[ui8_count];  // buffer current sampled
    value
24        diff =  sample- predsample; // calculate difference between  current
    and previous sampled value
25
26        /* quantization of difference value as per size defined in lookup
    table */
27        if(diff >= 0)
28             code = 0;
29        else
30        {
31             code = 8;
32             diff = - diff;
33        }
34        tempstep = step;
35        if(diff >= tempstep)
36        {
37             code |= 4;
38             diff -= tempstep;
39        }
40        tempstep >>= 1;
41        if(diff >= tempstep)
42        {
43             code |= 2;
44             diff -= tempstep;
45        }
46        tempstep >>= 1;
47        if(diff >= tempstep)
48             code |= 1;
49
50        /* de-quantization of code as per size defined in lookup table */
51        diffq = step >> 3;
52        if(code & 4)
53             diffq += step;
54        if(code & 2)
55             diffq += step >> 1;
56        if(code & 1)
57             diffq += step >> 2;
58
59        /* find the original value from decoded code */
60        if(code & 8)
61             predsample -= diffq;
```

```
62          else
63               predsample += diffq;
64          /* define the maximum and minimum cap */
65          if(predsample > 32767)
66               predsample = 32767;
67          else if(predsample < -32768)
68               predsample = -32768;
69
70          /* define the maximum and minimum cap */
71          index += IndexTable[code];
72
73          /* define the maximum and minimum cap of index */
74          if(index < 0)
75               index = 0;
76          if(index > 88)
77               index = 88;
78
79
80          if(predsample & 0x8000 ) // minimum value in case of negative
81          {
82            predsample = 0;
83          }
84          else if(predsample>4095) // maximum value in case of sample diff value
    is higher then 12 bits
85          {
86            predsample = 4095;
87          }
88          else
89          {
90             // nothing
91          }
92          /* buffer current difference sampled value and index  */
93          adpcm_state.prevsample = predsample;
94          adpcm_state.previndex = index;
95          int8_evensample++;
96
97          // use for combined upper and lower code nibble in one byte
98         if(int8_evensample >= 2)
99         {
100             uint8_codeTemp = uint8_codeTemp + ((code & 0x0f)<<4);
101             int8_evensample = 0;
102             p_ui8_outsample[ui8_count_codeindex] = uint8_codeTemp;
103             ui8_count_codeindex++;
104          }
105          else
```

```
106         {
107             uint8_codeTemp = (code & 0x0f);
108 
109         }
110 
111     }
112 }
```

5.1.2 Receiver Programming

Below is the initialization part of GPIOs which will be use for LEDs interface.

```
1     /* Open LED pins */
2     pinHandle = PIN_open(&pinState, pinTable);
3     if (pinHandle == NULL)
4     {
5         while(1);
6     }
7 
8     /* Clear LED pins */
9     PIN_setOutputValue(pinHandle, Board_PIN_LED1, 0);
10    PIN_setOutputValue(pinHandle, Board_PIN_LED2, 0);
```

Below is the initialization part of timer which will be use for SPI DAC sampling and timeout purpose.

```
1     /* Reset the timeout flag */
2     rxTimeoutFlag = false;
3     /* Set the reception flag to its default state */
4     rxDoneFlag = false;
5 
6     /* Open the GPTimer driver */
7     GPTimerCC26XX_Params params;
8     GPTimerCC26XX_Params_init(&params);
9     params.width          = GPT_CONFIG_32BIT;
10    params.mode           = GPT_MODE_ONESHOT;
11    params.direction      = GPTimerCC26XX_DIRECTION_UP;
12    params.debugStallMode = GPTimerCC26XX_DEBUG_STALL_OFF;
13    hTimer = GPTimerCC26XX_open(Board_GPTIMER0A, &params);
14    if(hTimer == NULL)
15    {
16        while(1);
17    }
18 
19    /* Set Timeout value to 125$\mu$s */
20    rxTimeoutVal = 5850; // 125usec
```

```
    GPTimerCC26XX_setLoadValue(hTimer, rxTimeoutVal);


    /* Register the GPTimer interrupt */
    GPTimerCC26XX_registerInterrupt(hTimer, rxTimeoutCb, GPT_INT_TIMEOUT);
```

Below section includes the radio ,SPI and adpcm initialization and start to timer.

```
    // Initialize the EasyLink parameters to their default values
    EasyLink_Params easyLink_params;
    EasyLink_Params_init(&easyLink_params);

    /*
     * Initialize EasyLink with the settings found in easylink_config.h
     * Modify EASYLINK_PARAM_CONFIG in easylink_config.h to change the default
     * PHY
     */
    if (EasyLink_init(&easyLink_params) != EasyLink_Status_Success){
        while(1);
    }

    /*
     * If you wish to use a frequency other than the default, use
     * the following API:
     * EasyLink_setFrequency(868000000);
     */

    /* SPI function initialization*/
    SPI_FN();

    /* Timer start*/
    GPTimerCC26XX_start(hTimer);

    /* ADPCM function initialization*/
    init_adpcm();
```

below is the inifite loop where we wait for receiving the radio command.

```
    while(1) {

        // Set the Receive done flag to false, callback will
        // set it to true
        rxDoneFlag = false;

        // Wait to receive a packet
        EasyLink_receiveAsync(rxDoneCb, 0);

```

```
        /*
         * Start the Receiver timeout timer (300ms) before
         * EasyLink_receiveAsync enables the power policy
         */
    //    // reset counter for Timeout of 150 msec
        Timer_counter = 0;

        while(rxDoneFlag == false){

            /* Break if timeout flag is set */
            if(rxTimeoutFlag == true){
                /* Reset the timeout flag */
                rxTimeoutFlag = false;
                /* RX timed out, abort */
                if(EasyLink_abort() == EasyLink_Status_Success)
                {
                    /* Wait for the abort */
                    while(rxDoneFlag == false){};
                }
                break;
            }
        }
    }
```

Now we will diagnosis RF command and blink the particular LEDs as per status.After receiving the payload, we parse the data and perform adpcm decoding algorithm[10] for original samples conversion.

```
void rxDoneCb(EasyLink_RxPacket * rxPacket, EasyLink_Status status)
{
    if (status == EasyLink_Status_Success)
    {
        /* Toggle LED2 to indicate RX */
        PIN_setOutputValue(pinHandle, Board_PIN_LED2,!PIN_getOutputValue(
    Board_PIN_LED2));

        /* data parsing of RF received payload */
         /* parsing of adpcm initialization values */
        int16_prevSampleForRF = rxPacket->payload[53]; // Previous sample
        int16_prevSampleForRF = (int16_prevSampleForRF<<8) + rxPacket->payload
    [52]; // append with step index

        ADPCMDecoder(&rxPacket->payload[2],&TxBufferToDAC[0],
    int16_prevSampleForRF, 100 ); // calling adpcm decoder


```

```
16          TxBufferToDACSampleCounter =0; // index for DAC sampling
17          SendDataToDACTrigger = 1;      // trigger the SPI for sending data to
    DAC
18      }
19      else if(status == EasyLink_Status_Aborted)
20      {
21          /* Toggle LED1 to indicate command aborted */
22          PIN_setOutputValue(pinHandle, Board_PIN_LED1,!PIN_getOutputValue(
    Board_PIN_LED1));
23      }
24      else
25      {
26          /* Toggle LED1 and LED2 to indicate error */
27          PIN_setOutputValue(pinHandle, Board_PIN_LED1,!PIN_getOutputValue(
    Board_PIN_LED1));
28          PIN_setOutputValue(pinHandle, Board_PIN_LED2,!PIN_getOutputValue(
    Board_PIN_LED2));
29      }
30
31      rxDoneFlag = true;
32  }
```

Below function is a call for timer timeout. In this function we buffer the sample for SPI DAC interface.

```
1
2   /* GP Timer Callback Function */
3   void rxTimeoutCb(GPTimerCC26XX_Handle handle,
4                    GPTimerCC26XX_IntMask interruptMask)
5   {
6       /*
7          * Timer is automatically stopped in one-shot mode and needs to be reset
        by
8          * loading the interval load value
9          */
10        GPTimerCC26XX_setLoadValue(hTimer, rxTimeoutVal);
11        GPTimerCC26XX_start(hTimer);
12
13      Timer_counter ++;
14
15      if(Timer_counter >= 1200) // Timeout of 150 msec
16      {
17          Timer_counter = 0;
18          /* Set the Timeout Flag */
19          rxTimeoutFlag = true;
20      }
```

```
    if(SendDataToDACTrigger ==1) // send data sample to DAC
    {
        SPI_SEND_TO_DMA((TxBufferToDAC[TxBufferToDACSampleCounter++]) | 0x7000
    );
        if(TxBufferToDACSampleCounter >= 100)
        {
        TxBufferToDACSampleCounter = 0;
          SendDataToDACTrigger = 0;
        }
    }

}
```

Below is the SPI initialization function for DAC interfacing.

```
void SPI_FN(void)
{
    /* INITIALIZATION OF SPI PARAMS */
    SPI_init();

     SPI_Params_init(&spiParams);
     spiParams.frameFormat = SPI_POL0_PHA0;                    // AFE4300 uses
    POL0_PHA1 as mode: idle state of SCLK is LOW, Data is changed on rising
    edge and sampled on falling edge
      spiParams.mode = SPI_MASTER;                             // CC1310 is
    transmitting as master
      spiParams.transferMode = SPI_MODE_BLOCKING;              // blocking mode:
    we only use one thread, so callback function would only cause unnecessary
    overhead
      spiParams.bitRate = 1000000;                             // supported bit
    rate by AFE4300: <= 4MHz
      spiParams.dataSize = 16;
      spiHandle = SPI_open(Board_SPI_MASTER, &spiParams);

}
```

Here MCU copy the sample in SPI buffer

```
void SPI_SEND_TO_DMA( uint16_t  data )
{
    uint16_t rxBuf[2];
    uint16_t txBuf[2];

    /* sample copy in spi buffer*/
    txBuf[0] = data;

    bool            transferOK = false;
```

```
    SPI_Status      transactionStatus;
    transaction.count = 1;
    transaction.txBuf = (void *) txBuf;
    transaction.rxBuf = (void *) rxBuf;

    transferOK = SPI_transfer(spiHandle, &transaction);
    transactionStatus = transaction.status;
}
```

Below is the ADPCM decode function[10] where we convert 50 bytes of code in to the 200 bytes of sampled data.

```
/* ADPCM decoder algorithm */
void ADPCMDecoder(uint8_t * p_ui8_inputsample,uint16_t * p_ui16_outsample,
    uint16_t prevSample, uint8_t buffer_size)
{
    int8_t  index = 0,code,int8_evensample =0;
    uint8_t  ui8_count = 0,ui8_count_codeindex = 0,uint8_codeTemp =0;
    int16_t  predsample,step,diffq ;

    // scaled frame sample value from 8 bits to 12 bits
    adpcm_state.prevsample = ((int16_t)(prevSample & 0x00FF))<<4;

    // frame index value for initial step size
    adpcm_state.previndex = (prevSample >>8) & 0xff;

    // looping up to 100 samples
    for(ui8_count = 0; ui8_count < buffer_size ; ui8_count ++)
    {
        /* previous sample and step size calculation */
        predsample = adpcm_state.prevsample;
        index = adpcm_state.previndex;
        step = StepSizeTable[index];

        /* current sample code extraction from lower and upper nibble*/
        int8_evensample++;
       if(int8_evensample >= 2)
       {
           int8_evensample = 0;
           code = (uint8_codeTemp>>4) & 0x0f;
           ui8_count_codeindex++;
        }
        else
        {
            uint8_codeTemp = p_ui8_inputsample[ui8_count_codeindex];
            code = (uint8_codeTemp & 0x0f);
```

```
34
35          }
36
37         /*  De-quantization of calculated step size*/
38          diffq = step >> 3;
39          if(code & 4)
40               diffq += step;
41          if(code & 2)
42               diffq += step >> 1;
43          if(code & 1)
44               diffq += step >> 2;
45
46          /*  calculate original sample from quantization sample code and
    calculated step size*/
47          if(code & 8)
48               predsample -= diffq;
49          else
50               predsample += diffq;
51
52          /* Precaution in case of higher sample limit*/
53          if(predsample > 32767)
54               predsample = 32767;
55          else if(predsample < -32768)
56               predsample = -32768;
57
58          /* calculate new index*/
59          index += IndexTable[code];
60
61          /* Precaution in case of higher index limit*/
62          if(index < 0)
63               index = 0;
64          if(index > 88)
65               index = 88;
66
67
68          /*  buffer decoded sample in array*/
69          if(predsample & 0x8000 )
70          {
71              p_ui16_outsample[ui8_count] = 0;// Limit for negative sample
72              predsample = 0;
73              index = 0;
74          }
75          else if(predsample>4025)
76          {
77              p_ui16_outsample[ui8_count] = 4026;// Limit for higher sample
```

```
78              predsample = 4026;
79              index = 66;
80          }
81          else
82          {
83              p_ui16_outsample[ui8_count] = predsample;// Buffer the sample
    value
84
85          }
86
87          /* pass calculated prev sample and index for further new sample
    calculation */
88          adpcm_state.prevsample = predsample;
89          adpcm_state.previndex = index;
90      }
91  }
```

5.2 Method Of Programming

In this section we will see the method of programming of CC1310 micro-controller. In this section we see that how start RF studio configures the radio operation source code.

5.2.1 RF Core Programming

5.2.1.1 Environment

- TI Code Composer Studio 9.0.1 IDE
- XDS100 debugger and programmer
- SmartRF Studio 7
- Two numbers CC1310 lunch-pad
- Two numbers Type A usb to Type B usb male cable

Radio configuration steps using SmartRF Studio 7 are given below:

- Initially, we need to configure the radio command as per our need. Here i have taken example of default settings as per given in figure 5.1.open the SmartRF Studio 7 , select CC1310 device, and configure the default settings.

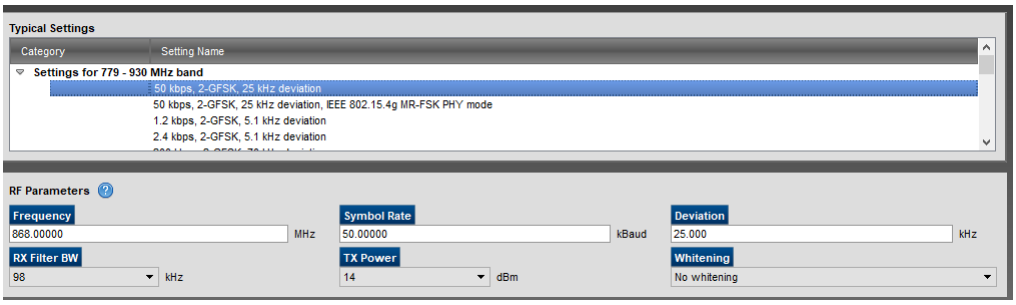

Figure 5.1: Default Configuration Settings : SmartRF Studio

- Now go in code export window and select command as per given in figure 5.2. Check the generated code.

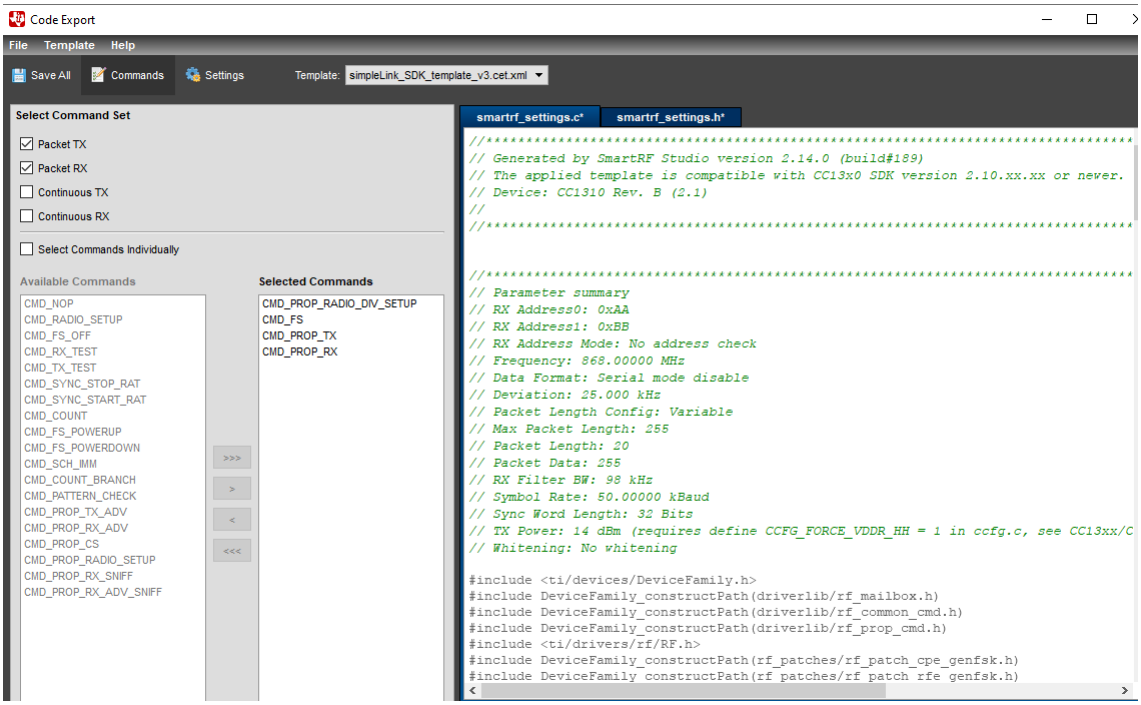

Figure 5.2: Command Selection : SmartRF Studio

- Now copy these generated file into the CCS code as per figure 5.3. Code import procedure in CCS is coming in next section.

5.2.2 Main Processor Programming

Following procedure is required to program the main controller :

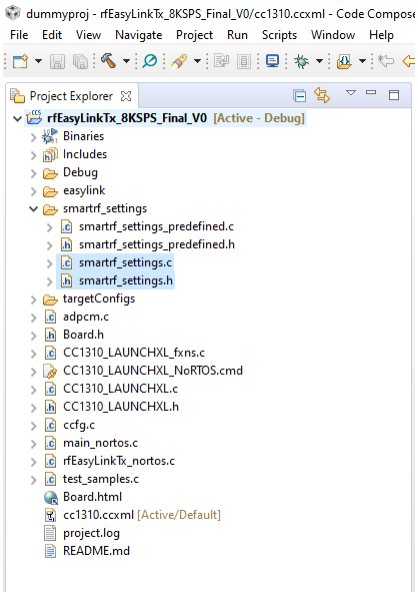

Figure 5.3: CCS File Replacement From SmartRF Studio File

- Import the working project into CCS.In example, i have created a dummy project as per given screen figure 5.4.

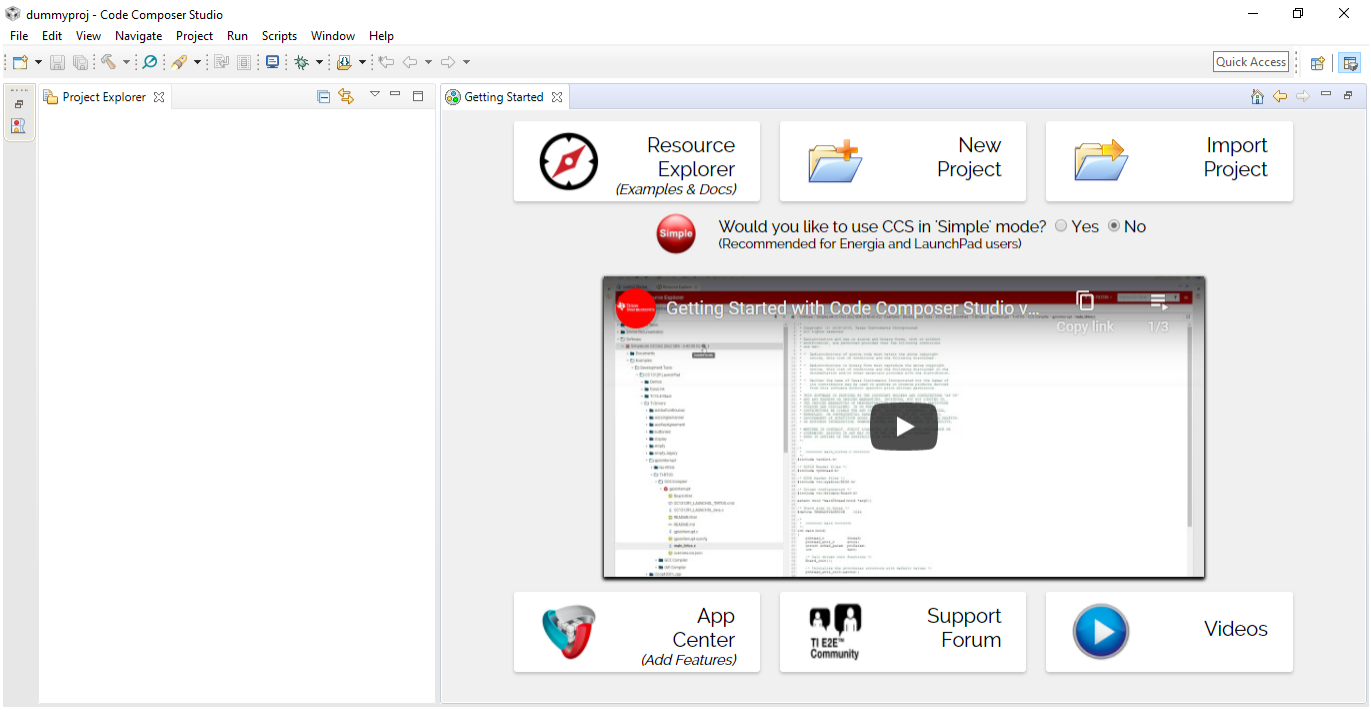

Figure 5.4: Screen After Open The CCS

- Now go in project import window and select the particuler path of source code as per figure

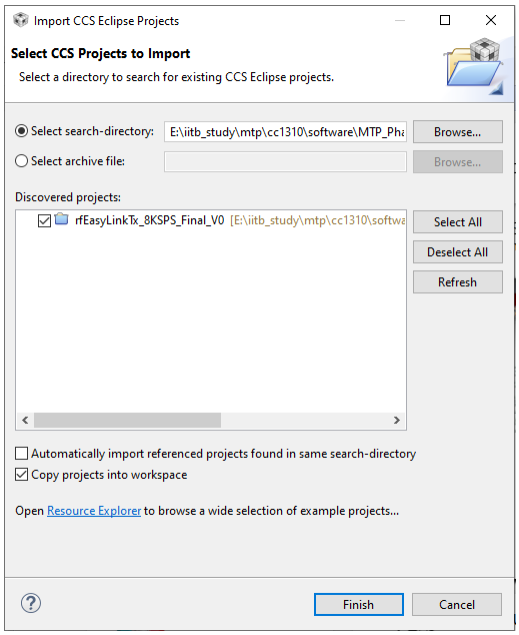

Figure 5.5: Transmitter Code Imported In CCS

5.6. and copy the entire code in present directory as per figure 5.5.

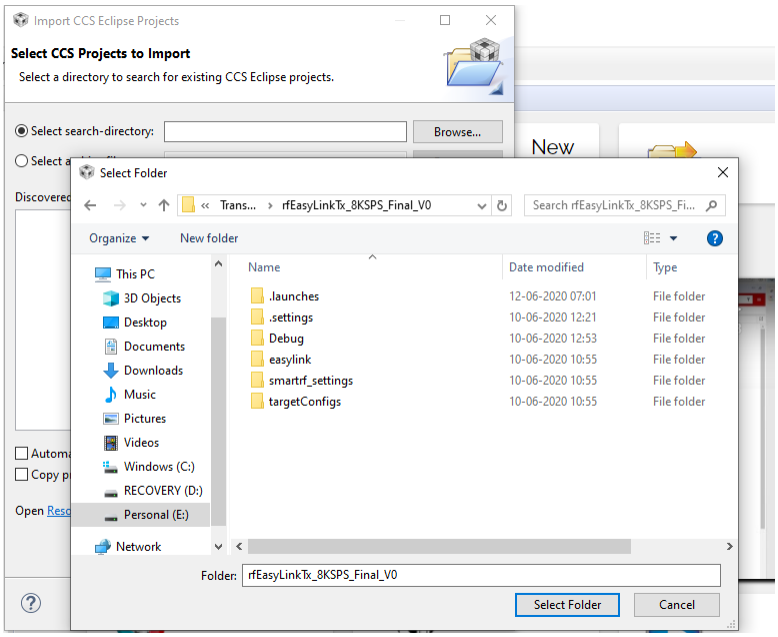

Figure 5.6: Directory Selection During Code Import In CCS

- Figure 5.7 is showing window after successfully import of code.

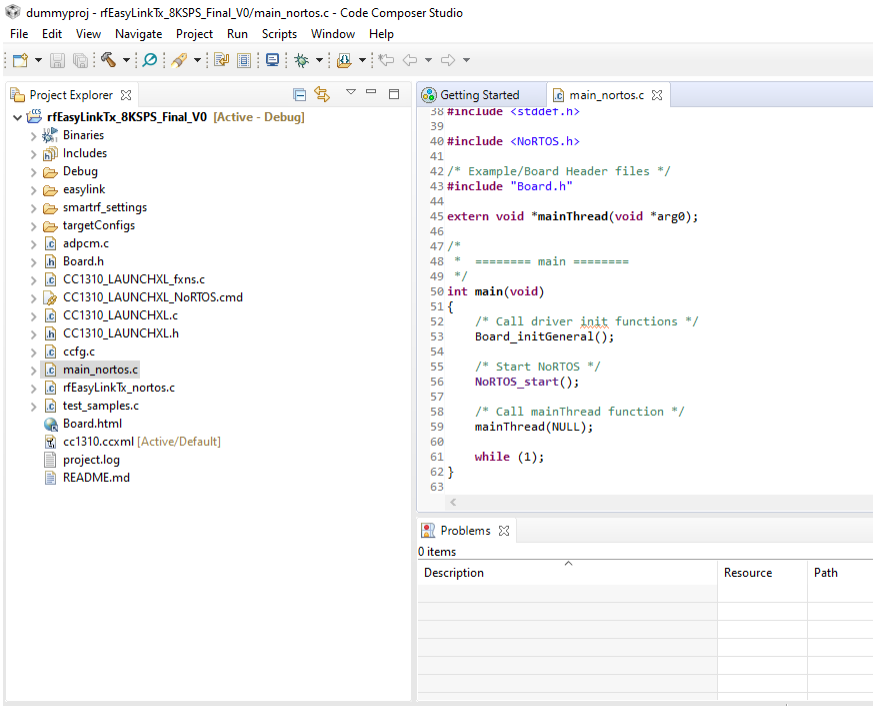

Figure 5.7: Window After Import Code In CCS

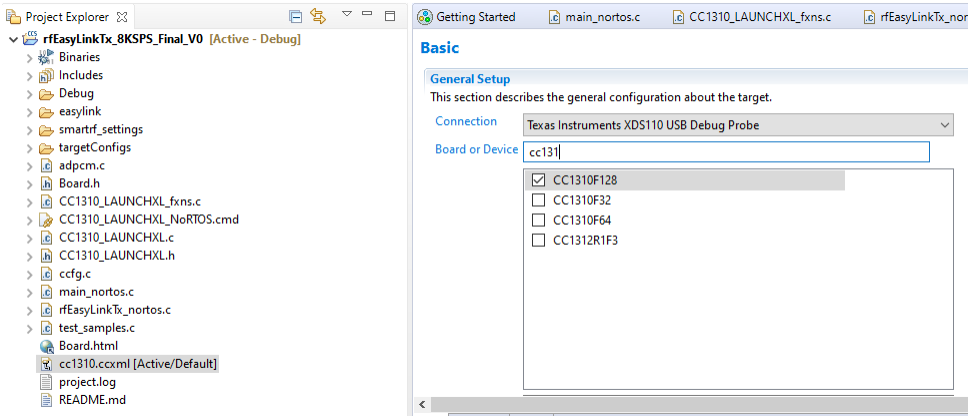

Figure 5.8: Target Section In .ccxml File

- Please check target selection setting by clicking on cc1310.ccxml file as per figure 5.8.

- Now give the bulid command to compile the code. After compilation total error should be zero as per figure 5.9.

- Now for programming , give the debugg command as per figure 5.10.

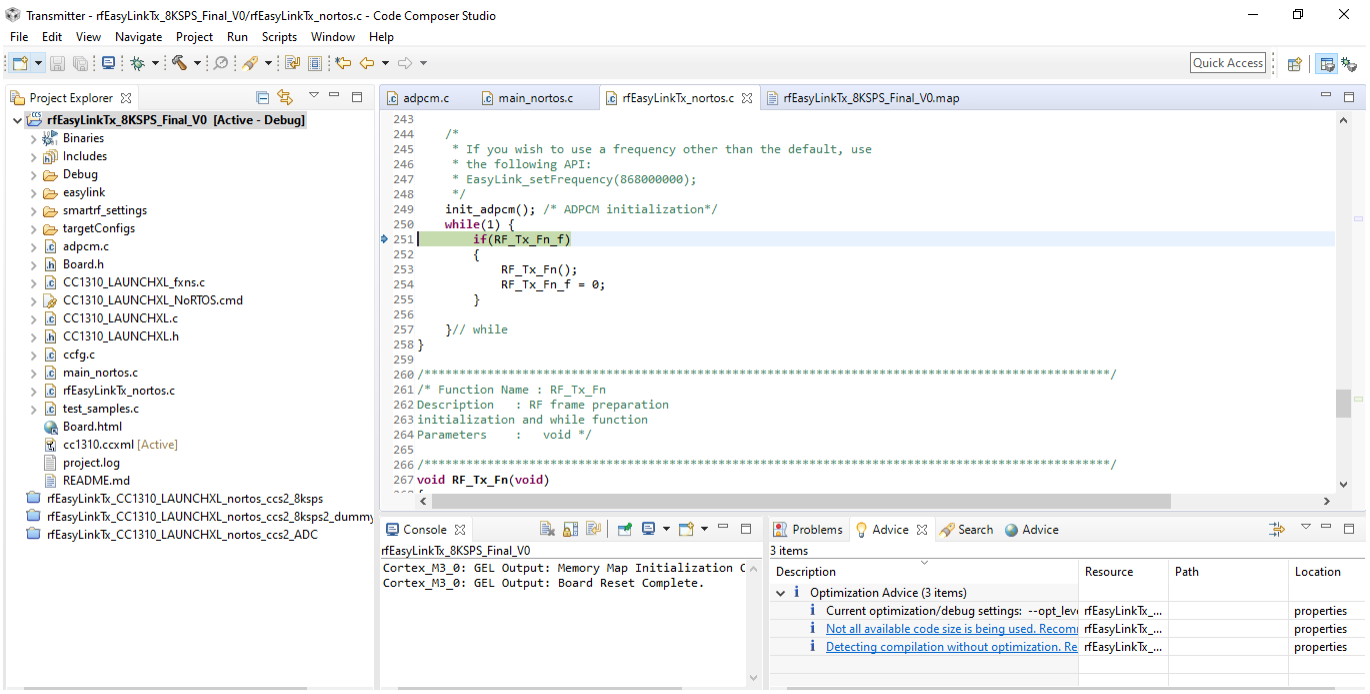

Figure 5.9: Window After Compilation Of Code

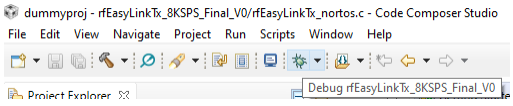

Figure 5.10: Code Programming In CCS

- After coming in debugg mode, start to run the code by clicking on green icon and check the respected variable. In case of voice injection, as input of signal conditioning on actual hardware,screen in figure 5.11 will be appear.

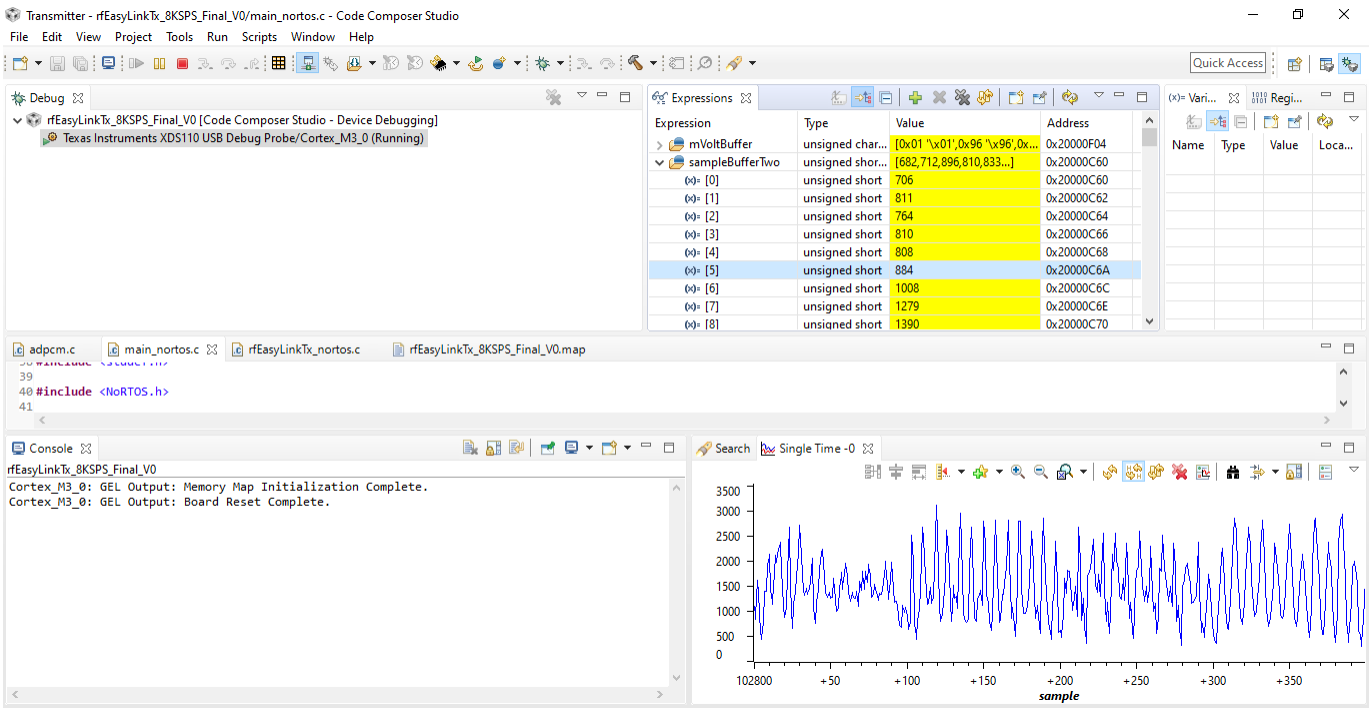

Figure 5.11: Debugger Window in CCS

5.3 Memory Utilization

```
MEMORY CONFIGURATION OF rfEasyLinkTx_8KSPS_Final_V0

 name     origin     length     used       unused     attr  fill

 FLASH    00000000   00020000   0000ae4a   000151b6   R     X
 SRAM     20000000   00005000   000019b0   00003650   RW    X
```

In transmitter map file we have seen that total flash utilization is 44k among available 128k memory. So flash utilization is 34%. total SRAM utilization is 6.4k among available 20k memory. So SRAM utilization is 32%.

Below is the memory utimization on receiver end.

```
MEMORY CONFIGURATION OF rfEasyLinkRx_8KSPS_Final_V0

 name     origin     length     used       unused     attr  fill

 FLASH    00000000   00020000   0000b02e   00014fd2   R     X
 SRAM     20000000   00005000   00001a50   000035b0   RW    X
```

In receiver map file we have seen that total flash utilization is 44k among available 128k memory. So flash utilization is 34%. total SRAM utilization is 6.5k among available 20k memory. So SRAM utilization is 32.89%.

Chapter 6

Data Analysis

In this chapter we will see the final test results during testing.

6.1 Prototype Testing

A prototype is developed during proof of concept. Also i have did multiple testing on actual voice signal as well as hard-coded signal.

6.1.1 Transmitter

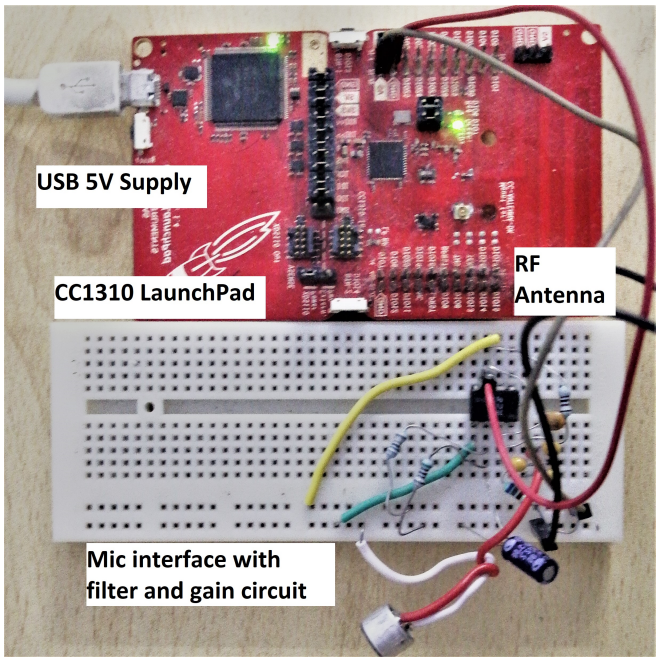

Figure 6.1: Transmitter Prototype

On general purpose board, one circuit has been designed for signal conditioning and mic interface.The output of this circuit will be input of ADC pin (IO23) as sown in figure 6.1. 5V DC supply has been given from CC1310 lunch-pad. Signal are transmitting over RF via antenna with frequency 868 Mhz.

6.1.2 Receiver

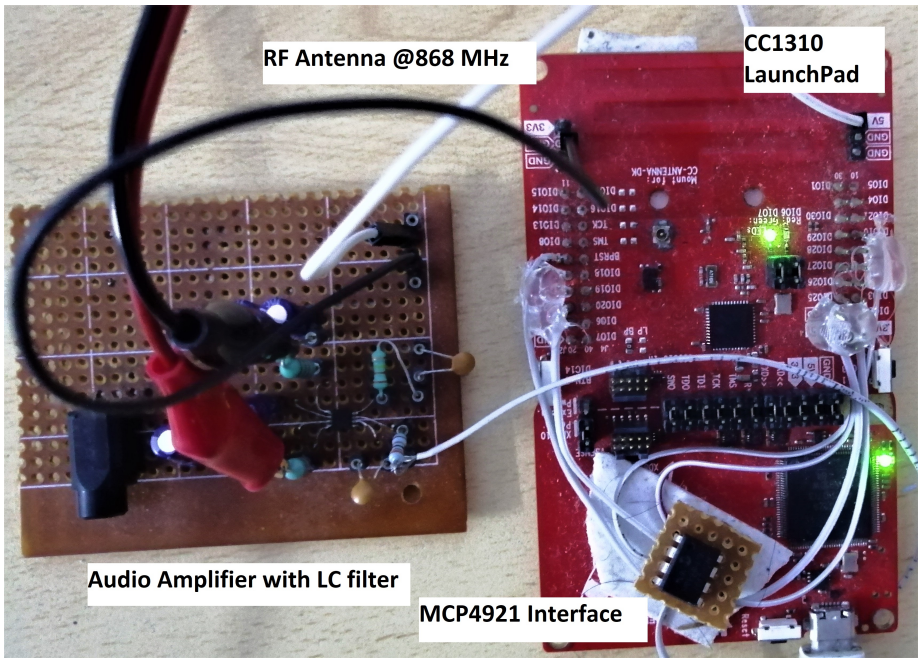

Figure 6.2: Receiver Prototype

On general purpose board, two circuits has been designed First for MCP4921 interface over SPI with CC1310 controller.The output of this circuit will be input of audio amplifier and LC filter circuit. as sown in figure 6.2. 5V DC supply has been given from CC1310 lunch-pad. Signal are receiving over RF via antenna with frequency 868 Mhz. 35mm jack is also available to connect the earphone on secondary board.

6.2 Testing In Transmitter

6.2.1 Signal Conditioning on Transmitter

On general purpose board, one circuit has been designed for signal conditioning and mic interface.The output of this circuit will be input of ADC pin (IO23) as sown in figure 6.1. 5V DC supply has been given from CC1310 lunch-pad. The matlab simulation has been shown in figure 6.3, where we can see that its making a

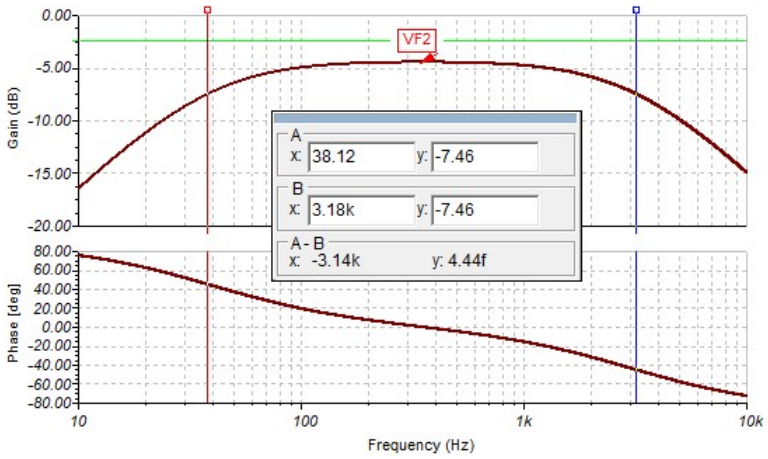

Figure 6.3: Matlab Simulation Results Of Analog Signal Conditioning: BPF 38Hz To 3.18 kHz

6.2.2 Voltage Input vs ADC Output

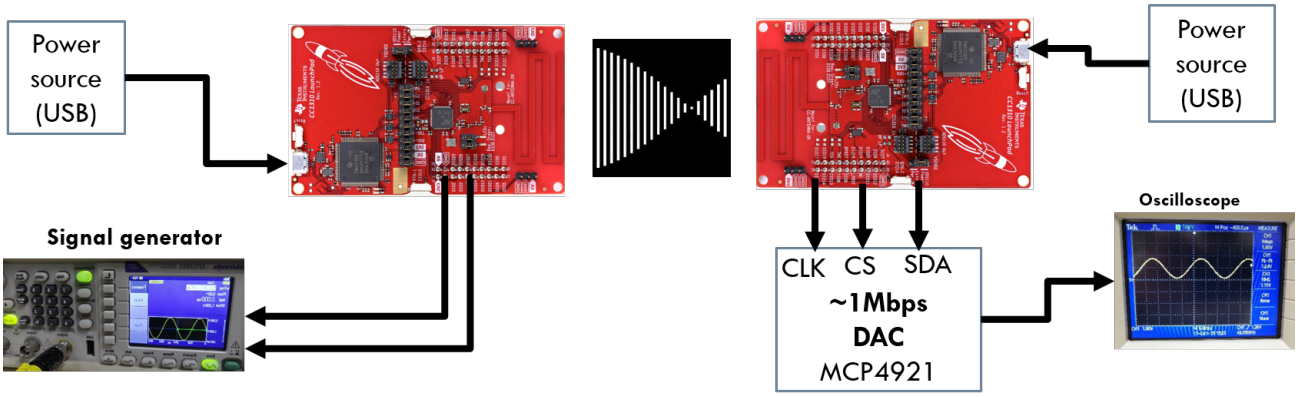

Figure 6.4: Test Setup : Transmitter And Receiver

Figure 6.4 shows the test setup for modular testing of transmitter and receiver. Input signals are injected by function generator and output of receiver is tested on oscilloscope as per figure 6.4.

Figure 6.5 is showing ADC output in term of counts with respect of input DC voltage.Counts are measured on CCS IDE. Up to 3.3V ADC input, counts are linear after that it is saturating because of maximum controller voltage 3.3V DC.

6.2.3 Transmitter ADC Output

Figure 6.6 is showing ADC output for sinusoidal input voltage in term of voltage.

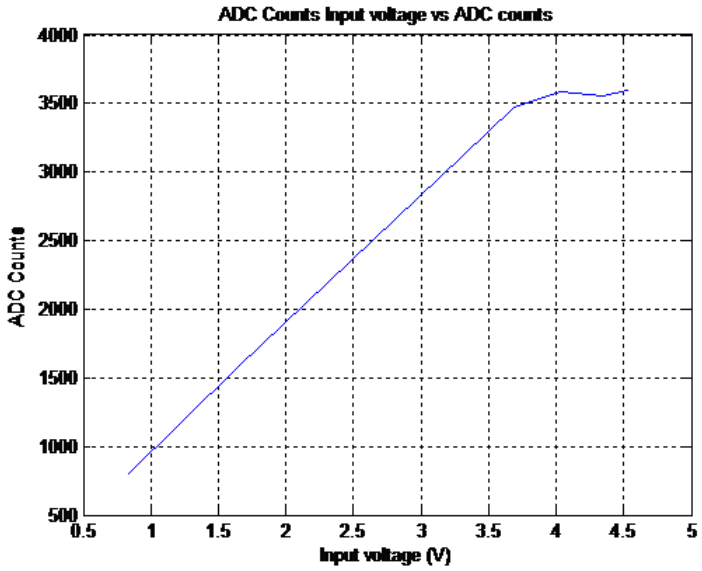

Figure 6.5: Input DC voltage vs Output ADC Counts

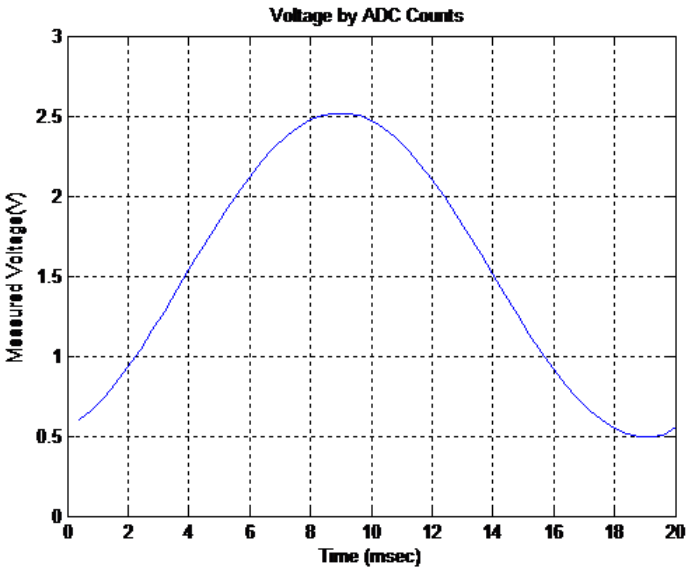

Figure 6.6: ADC Output For Sinusoidal Input Injection

6.3 Testing In Receiver

6.3.1 DAC Output Signal

In this section we will see DAC output on different frequency input waveform. Please note that, all waveforms are captured during Phase-1 work. so here sampling rate is 4ksps instead of 8ksps. It is also noted that, ADPCM technique was also not implemented in phase-1 so the signal from

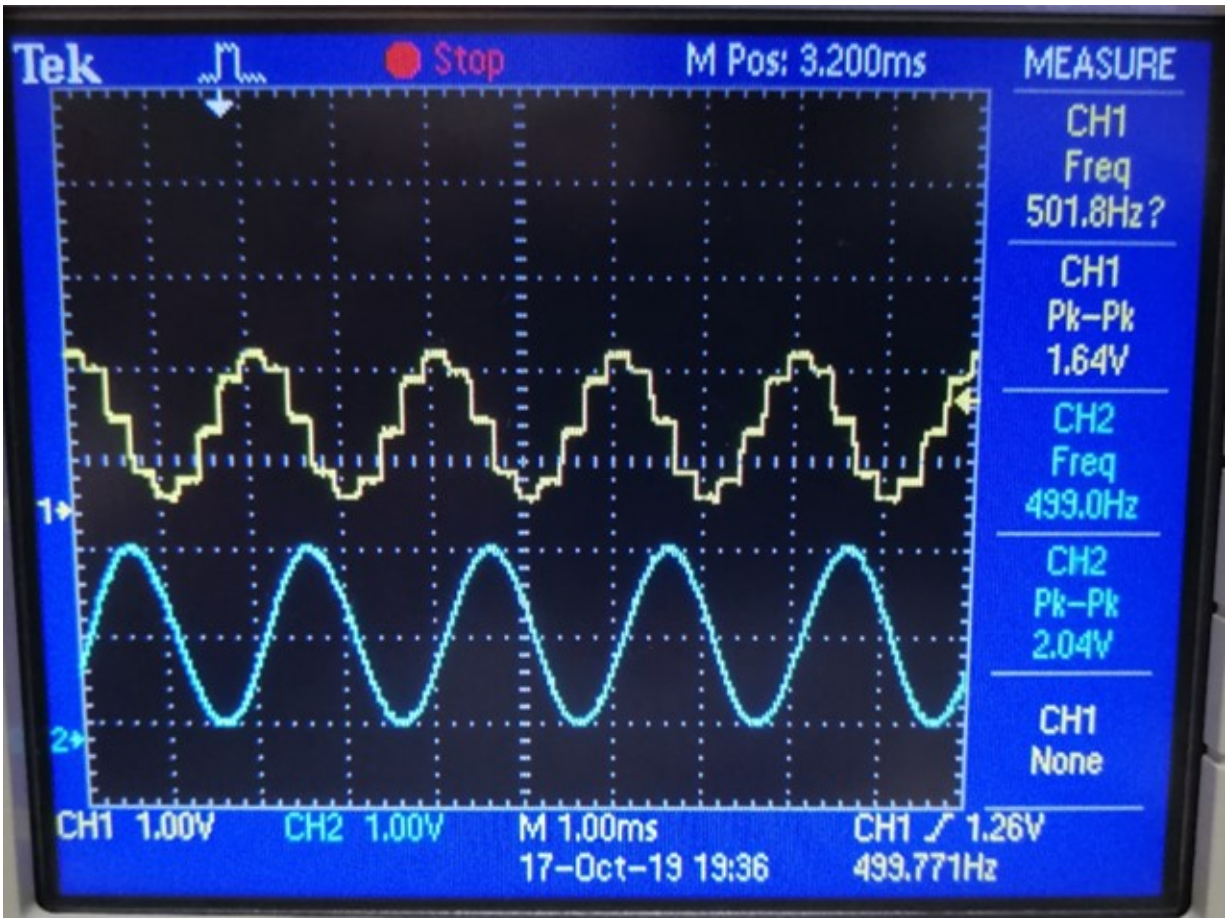

Figure 6.7: Transmitter Inp vs Receiver DAC Out Input Freq:500Hz , Sampling Rate: 4ksps

transmitter of receiver was already scaled down from 12 bits to 8 bits for 50kbps symbole rate.

- Figure6.7 is showing input- output waveform at input of $2V_{PP}$ and 499.77 Hz frequency.
- Figure6.8 is showing input- output waveform at input of $2V_{PP}$ and 50 Hz frequency.
- Figure6.9 is showing input- output waveform at input of $2V_{PP}$ and 1000 Hz frequency.

6.3.2 Time Delay Transmitter To Receiver

Figure6.10 is showing the time delay between data captured by transmitter to signal delivered by receiver.Again please note that, all waveforms are captured during Phase-1 work. so here sampling rate is 4ksps instead of 8ksps. It is also noted that, ADPCM technique was also not implemented in phase-1 so the signal from transmitter of receiver was already scaled down from 12 bits to 8 bits for 50kbps. So total time latency is 46 msec.
Bifurcation of 46 is 25msec (sampling time) + 18 msec RF time) + 3 msec (others).

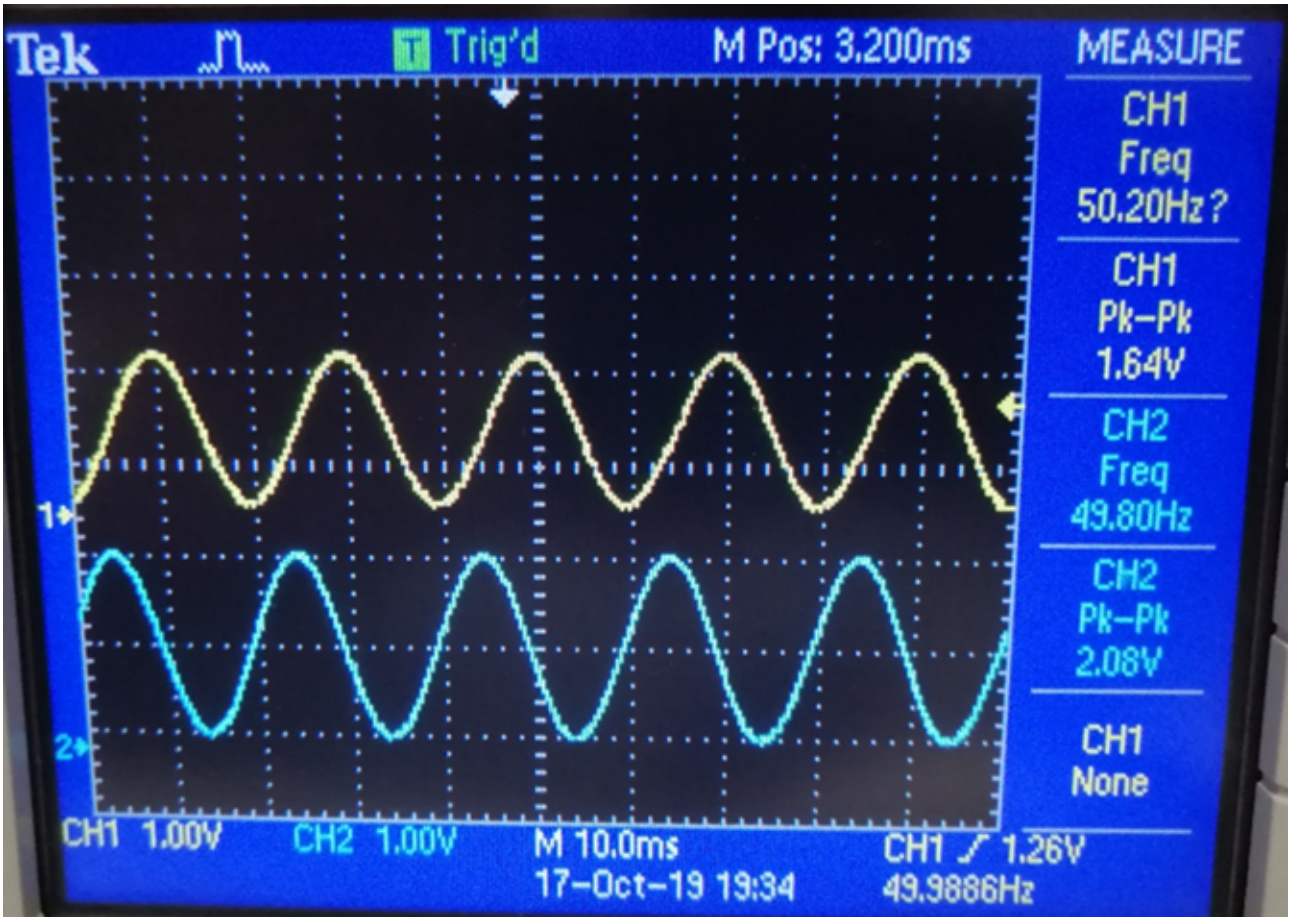

Figure 6.8: Transmitter Inp vs Receiver DAC Out Input Freq:50Hz , Sampling Rate: 4ksps

After delta modulation technique final time should be (need to be test) 12.5 msec (8ksps sampling rate) + 10 msec (RF) + 2 .5msec (adpcm encoder) + 2.5msec (adpcm decoder) + 2.5msec (others) **=30 msec**.

6.3.3 Radio Signal Test

Figure6.11 is showing the radio command testing from SmartRFStudio TI tool. here we have configured the toll for 868 MHz frequency, 50kbps symbol rate and 25kHz deviation. Transmitting power is 14 dbm. Average received signal strength indicator (RSSI) is achieving -84 dbm without loss of packet. Also showing the command configured during communication.

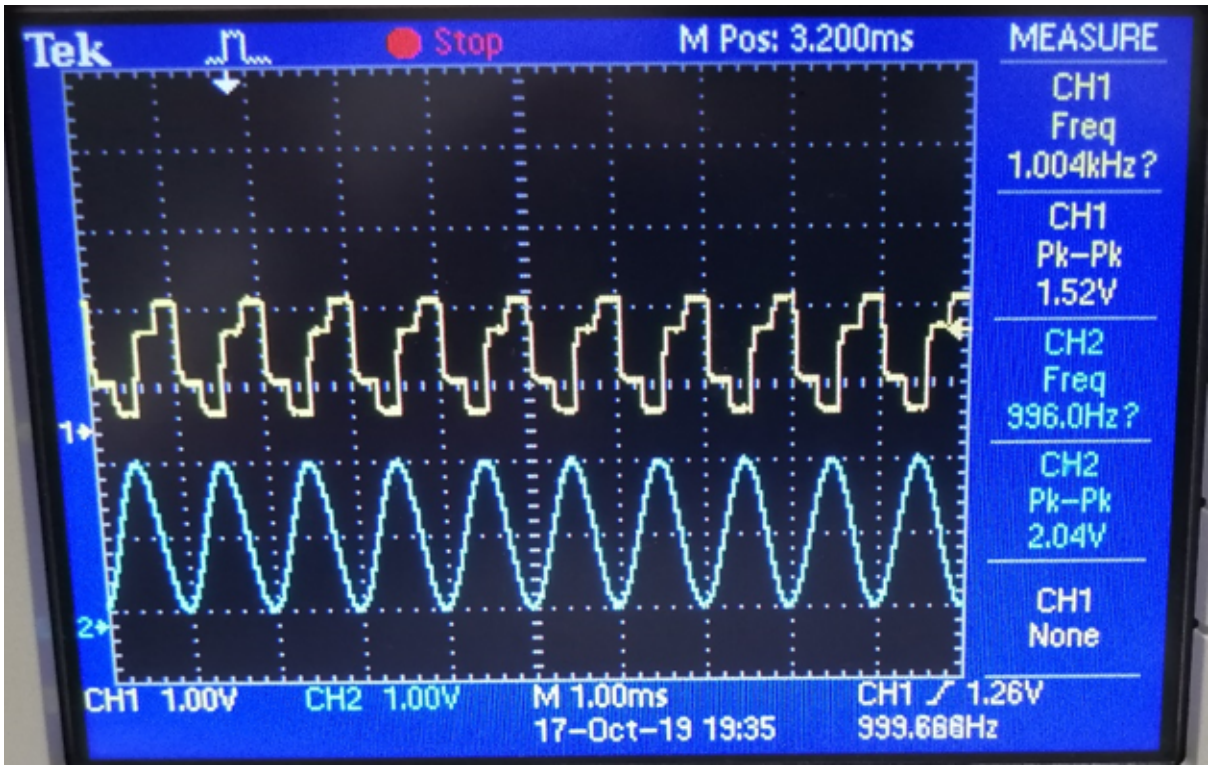

Figure 6.9: Transmitter Inp vs Receiver DAC Out Input freq:1kHz , Sampling Rate: 4ksps

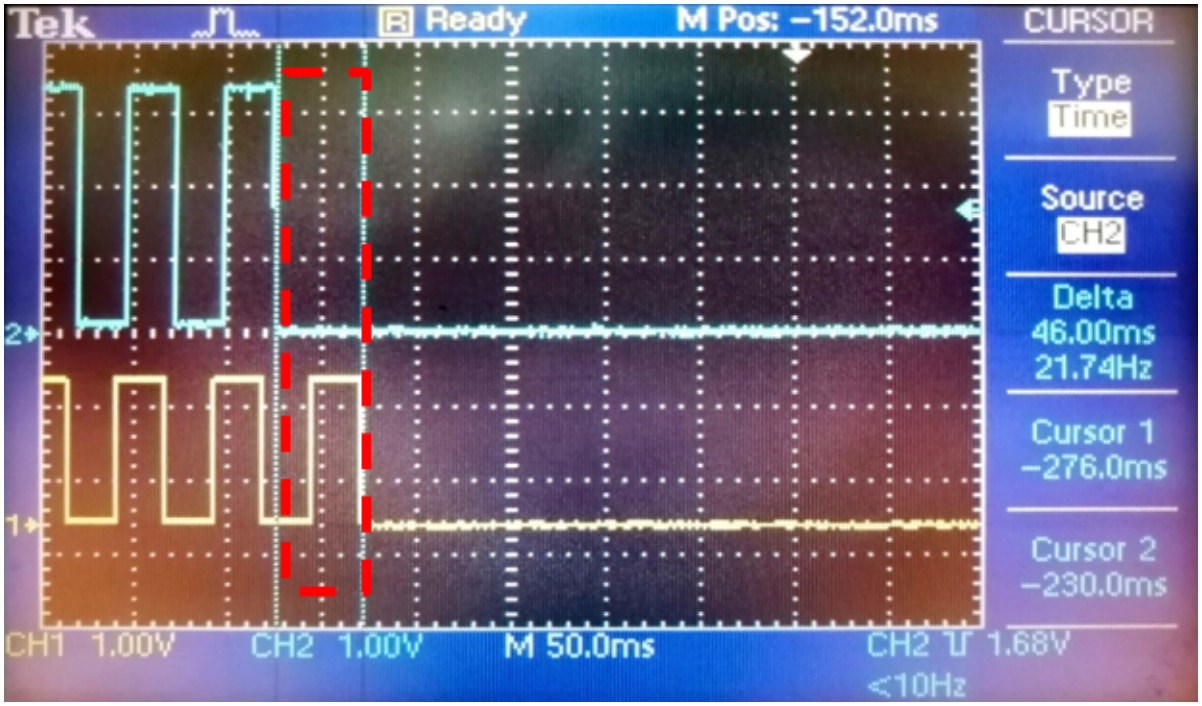

Figure 6.10: Signal Latency Between Transmission And Reception : F_s:4kHz Without Compression

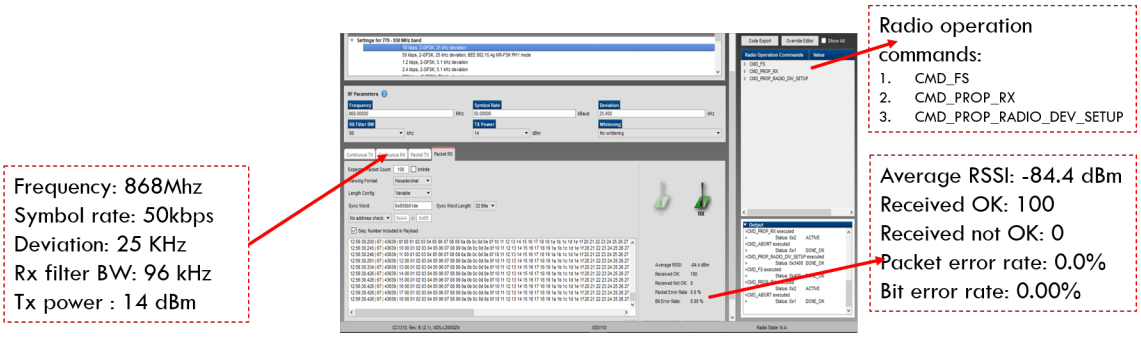

Figure 6.11: Packet Receive On SmartRFStudio

6.4 ADPCM Encoder-Decoder Testing

Figure6.12 is showing adpcm input - output results , simulated in matlab. Here 475 Hz signal is transmitted from tone generator tool which is given to the signal conditioning. Samples are captured by ADC and plotted in single line graph. Now transmitter encode these samples and transmitted to receiver. Output of adpcm decoded signal is also plotted in figure.

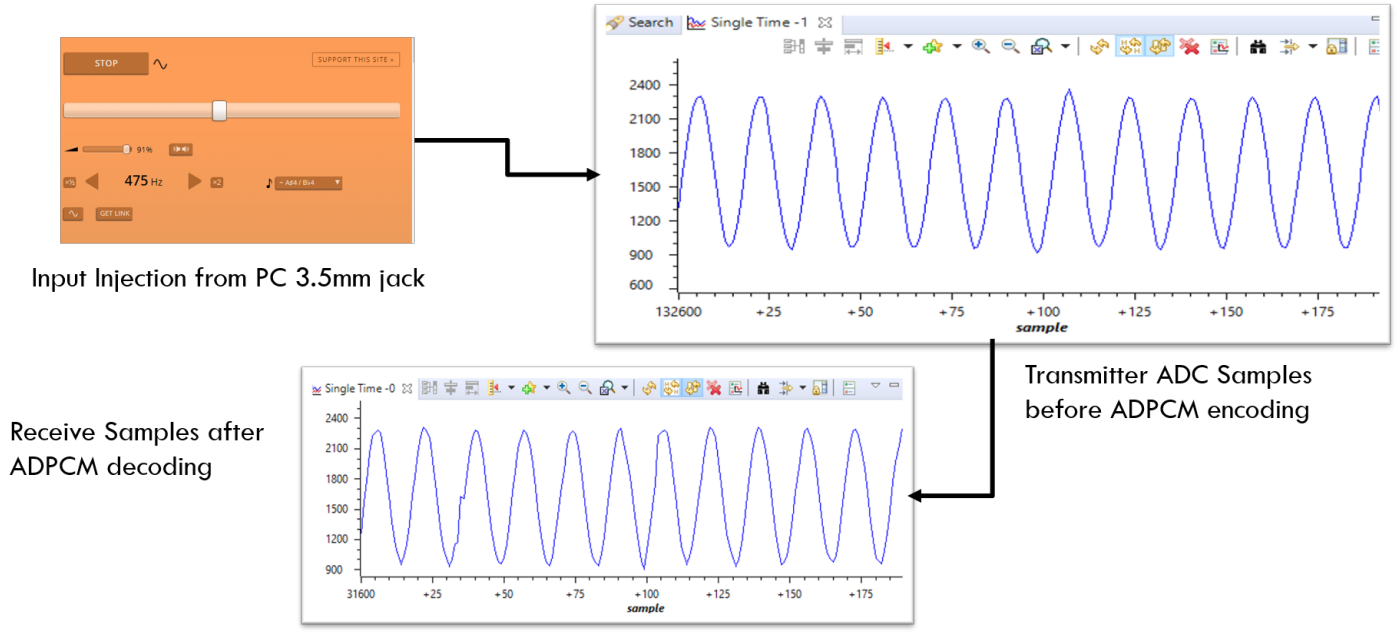

Figure 6.12: ADPCM Encoder Input vs Decoder Output

6.5 Range Testing

Range test has been done in iit mumbai from convocation hall to H7 canteen.Again please note that, test is done during Phase-1 work. so here sampling rate is 4ksps instead of 8ksps. It is also

noted that, ADPCM technique was also not implemented in phase-1 so the signal from transmitter of receiver was already scaled down from 12 bits to 8 bits for 50kbps.
As per figure6.13 ,the range was coming 772 meters without loss of packets LOS. As per friss equation, ideal range was calculated 2.2 KM. As per figure6.14 same setup is tested in Gymkhana ground (Maximum length : 300 meters).

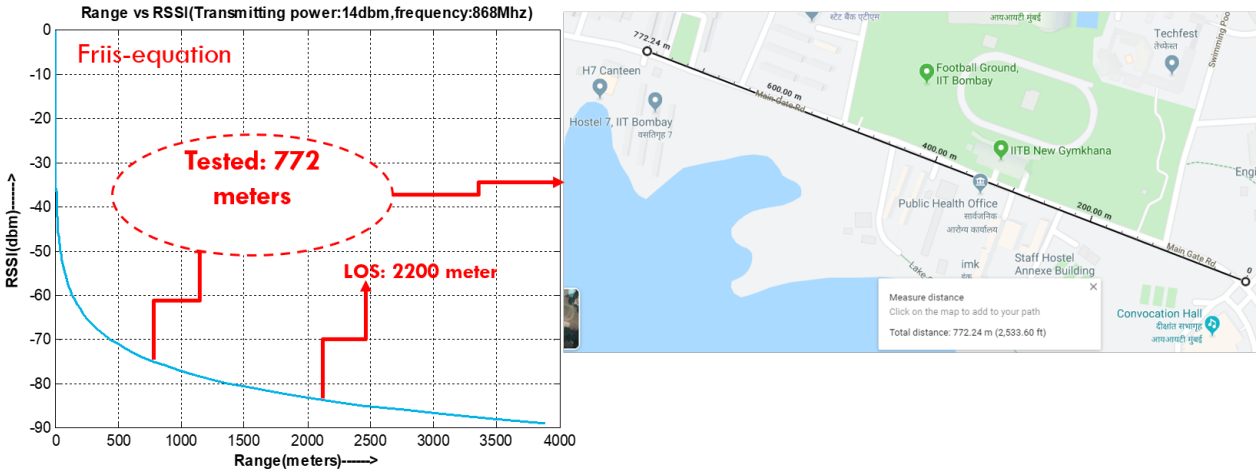

Figure 6.13: RF Range Test : LOS 772 Meter

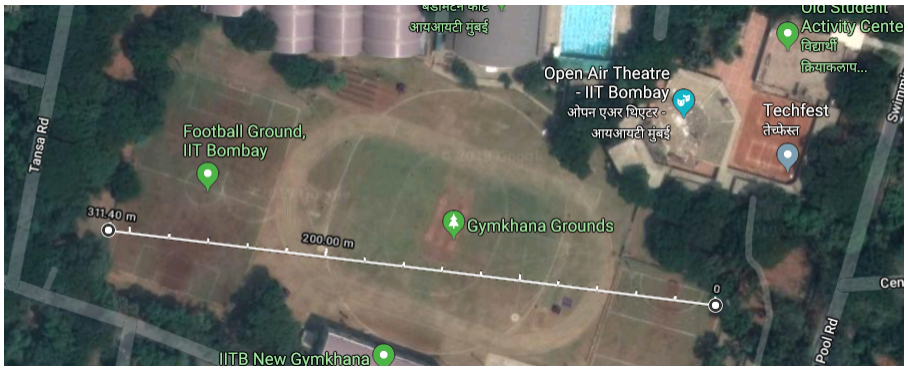

Figure 6.14: RF Range Test In IITB GymKhana : 300 Meter(Maximum Ground Length)

Chapter 7

Conclusion And Future Work

7.1 Conclusion

This work implemented a audio transmission and reception system which is working on higher sampling rate and lower channel symbol rate. Trade off between symbol rate and sampling rate is actual motivation to develop this prototype. Using this scheme we can transmit 200 bytes of information with sampling rat 8000 sps using RF channel bit rate 50 kbps. IN normal operation as per calculation, it is required 200 kbps bit rate. So we can say that i have improved system efficiency up to approximate 400%. We also designed different input output filter for noise suppression and signal improvement. Actual testing with voice signal is also done with adpcm delta modulation.RF communication is tested with 868Mhz, monopole PCB mounted antenna, 14 dbm transmitting power, bit rate: 50kbps in IIT Mumbai premises with range of 750 meters. All designing calculation, programming method and source code has been explained during documentation.

7.2 Future Work

Future work includes coming up with new topology which include companding circuit for controlling the amplitude level of input voice signal. Further we can also use a SPI based audio codec circuit at receiver output instead of MCP4921 and other interface. Forward error correction can also improve the range by correction of loss information but it also increase the RF channel bandwidth.

Bibliography

[1] “RF operation state chart”[Online][Accessed : Sep 20,2019]
http://dev.ti.com/tirex/content/simplelink_cc13x0_sdk_1_30_00_06/docs/proprietary-rf/html/rf-core/execution-and-status-codes.html

[2] “SDL diagram of the Evaluating phase” [Online], [Accessed : Sep 20,2019]
http://dev.ti.com/tirex/content/simplelink_cc13x0_sdk_1_30_00_06/docs/proprietary-rf/html/rf-core/conditional-execution-and-chaining.html.

[3] “The callback events” [Online][Accessed : Sep 20,2019]
http://dev.ti.com/tirex/content/simplelink_cc13x0_sdk_1_30_00_06/docs/proprietary-rf/html/rf-core/conditional-execution-and-chaining.html.

[4] “callback events for CMD_PROP_RX” [Online], [Accessed : Sep 20,2019]
http://dev.ti.com/tirex/content/simplelink_cc13x0_sdk_1_30_00_06/docs/proprietary-rf/html/rf-core/callback-events.html.

[5] “Packet format” [Online], [Accessed : Sep 20,2019]
http://dev.ti.com/tirex/content/simplelink_cc13x0_sdk_1_30_00_06/docs/proprietary-rf/html/rf-proprietary/packet-format.html.

[6] “Oscilloscope Widget” [Online], [Accessed : Oct 16,2019]
https://www.eclipse.org/nebula/widgets/oscilloscope/oscilloscope.php.

[7] “MCP4921/4922 12-Bit DAC with SPI™ Interface”, Datasheet by Microchip Technology Inc. DS21897B-page 1 , 2007 Microchip Technology Inc.

[8] “CC1310 SimpleLink™ Ultra-Low-Power Sub-1 GHz Wireless MCU SWRS181D –SEPTEMBER 2015–REVISED JULY 2018”, Datasheet Texas Instrument

[9] J. R. BODDIE, J. D. JOHNSTON, C. A. McGONEGAL, J. W. UPTON, D. A. BERKLEY, R. E. CROCHIERE, and J. L. FLANAGAN , “Adaptive Differential Pulse-Code-Modulation “ Copyright © 1981 American Telephone and Telegraph Company THE BELL SYSTEM TECHNICAL JOURNAL Vol. 60, No. 7. September 1981 Printed in U.S.A.

[10] “Adaptive Differential Pulse Code Modulation Using PIC® Microcontrollers AN643”, By Rodger Richey 2007 Microchip Technology Inc.. DS00643C-page 1

[11] “EasyLink API Reference” [Online], [Accessed : June 15,2010] https://processors.wiki.ti.com/index.php/SimpleLink-EasyLink