CMR UNIVERSITY
Private University Established in Karnataka State by Act No. 45 of 2013

School of Engineering Technology

Main Campus, Off Hennur-Bagalur Main Road, Chagalahatti, Bengaluru-562149

MINI PROJECT REPORT

"FM Transmitter using Raspberry Pi "

submitted to,

School of Engineering and Technology, CMR University

in partial fulfilment of the requirement for the award of the degree of

Bachelor of Technology,
in
COMPUTER SCIENCE AND ENGINEERING/ INFORMATION TECHNOLOGY

by

1)MOHAMMED SHOAIB (USN: 16UG08024)
2)PRANAV REDDY (USN: 16UG08033)
3)KISHORE.K (USN: 16UG08015)
4)SARANYA.T (USN: 16UG08047)

Under the guidance of,

Prof. GEETHA N

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING/INFORMATION TECHNOLOGY,

CMR UNIVERSITY

2018-19

CMR UNIVERSITY
Private University Established in Karnataka State by Act No. 45 of 2013

School of Engineering and Technology, CMR

Main Campus, Off Hennur-Bagalur Main Road, Chagalahatti, Bengaluru-562149

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING/INFORMATION TECHNOLOGY

CERTIFICATE

Certified that the mini project titled ***FM TRANSMITTER USING RASPBERRY PI*** *carried out by Mr./Ms. 1)**MOHAMMED SHOAIB, (USN: 16UG08024), 2)PRANAV REDDY (USN: 16UG08033)**, 3)**KISHORE K(USN: 16UG08015) AND SARANYA T(USN: 16UG08047)** in partial fulfilment for the award of Bachelor of Technology in **COMPUTER SCIENCE AND ENGINEERING** of CMR University, during the year 2018-19. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The mini project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.*

Name of the Guide ***Name of the Dean***

Signature of the Guide *Signature of the Dean*

Examiners

Name of the examiners **Signature with date**

1

2

DECLARATION

*We, **MOHAMMED SHOAIB, PRANAV REDDY, KISHORE K AND SARANYA T** students of School of Engineering and Technology, CMR university, hereby declare that the dissertation titled **"FM TRANSMITTER USING RASPBERRY PI"** embodies the report of my mini project carried out independently by us during fifth semester of **Bachelor of Technology in Computer Science and Engineering/Information Technology,** under the supervision of **Prof. GEETHA N,** Department of Computer Science and Engineering and this work has been submitted in partial fulfilment for the award of the **Bachelor of Technology** degree.*

We have not submitted the project for the award of any other degree of any other university or institution.

Date :

Place :

MOHAMMAED SHOAIB (16UG08024)

PRANAV REDDY(16UG08033)

KISHORE K(16UG08015)

SARANYA T(16UG08047)

ACKNOWLEDGEMENT

We express our foremost gratitude to ***Dr. Harish****, Dean for his constant support.*

We express our foremost gratitude to my guide ***Prof. Geetha N****, Dept of Computer Science & Engineering/Information Technology, CMR University, Bagalur for his/her inspiration, adroit guidance, constant supervision, direction and discussions in successful completion of this dissertation.*

Our sincere thanks to all teaching and non-Teaching Staff of ***Computer Science & Engineering/Information Technology Department*** *for all the facilities provided, without which, we could not have progressed with my work. Thanks to our parents who have been a great source of strength in the completion of this dissertation.*

MOHAMMED SHOAIB
(16UG08024)
PRANAV REDDY
(16UG08033)
KISHORE K
(160G08015)
SARANYA T
(16UG08047)

ABSTRACT

Our project involves using of a Raspberry Pi as an FM Transmitter. A Raspberry Pi is an ARM cortex based popular development board designed for electronics engineers and makers. It’s a single board computer working on low power. FM stations get very boring with the RJ blabbering irrelevant stuff or some bugging advertisements and that might have kept you guessing why you can’t have your own FM Broadcast station to air your voice and music over a short distance. Surprising enough with the help of Raspberry Pi it should hardly take less than half an hour to set up your own FM broadcasting station and get on air within a local area. With the help of a proper antenna you should be able to cover an area of 50m Radius which should be enough to broadcast within your school or locality.

Every microprocessor will have a synchronous digital system associated with it which is used to reduce the electromagnetic interference. This EMI suppression is done by a signal called Spread-spectrum clock signal or SSCS for short. The frequency of this signal can vary from 1MHz to 250MHz which luckily for us falls within the FM band. So by writing a code to perform frequency modulation using the spread-spectrum clock signal we can tweak the Pi to work as a FM transmitter. The modulated signal will be given out through the GPIO pin 4 of the Raspberry Pi. We can simply attach a normal wire of 20 cm maximum to this pin to act as an antenna. So hence we are able to broadcast our own songs and create a personal FM station.

TABLE OF CONTENTS

CHAPTER 1

PREAMBLE

1.1 Introduction

Be it a boring afternoon, a monotonous job or a lonely long drive FM radio stations have always kept us entertained. While on the contradictory it should also be agreed that sometimes these FM stations get very boring with the RJ blabbering irrelevant stuff or some bugging advertisements and that might have kept you guessing why you can’t have your own FM Broadcast station to air your voice and music over a short distance!

With the help of Raspberry Pi it should hardly take less than half an hour to set up your own FM broadcasting station and get on air within a local area. With the help of a proper antenna you should be able to cover an area of 50m Radius which should be enough to broadcast within your school or locality. We can hence set up our own FM station and use it within our college and implement the idea of a College Radio.

1.2 Literature Survey

Joseph Mitola, R [1]: the term software defined radio introduced Joseph Mitola , who published the first paper on the topic in 1992 though the concept was proposed in 1991.

Heaps J, C [2]: the first military application that used SDR was speak easy program that was initiated by the US military. The program aim to use programmable processing to emulate multiple existing radios and easily incorporate new coding and modulation standards.

***DeFelice, Bill [*3]**: Carrier current broadcasting goes back to the 1960's as the original budget-friendly method that allowed colleges and universities to have their very own license-free, student-run campus radio station. Carrier current transmission equipment and studio accessories are available from Radio Systems, Inc. of Logan Township, New Jersey, and were formerly offered by the now-defunct Low Power Broadcasting, Inc. of Pennsylvania.

R Gandhiraj , Ranjini Ram, KP Soman [4]:Analog and digital modulation for software defined radio volume 30,2012,1155-1162.

1.3 Problem statement

FM stations get very boring with the RJ saying irrelevant stuff or some bugging advertisements which are avoidable. So why not create our own FM radio station?

1.4 Objective

To set up our own personal FM radio station using a Raspberry Pi .

1.5 Methodology

Every microprocessor will have a synchronous digital system associated with it which is used to reduce the electromagnetic interference. This EMI suppression is done by a signal called Spread-spectrum clock signal or SSCS for short. The frequency of this signal can vary from 1MHz to 250MHz which luckily for us falls within the FM band. So by writing a code to perform frequency modulation using the spread-spectrum clock signal we can tweak the Pi to work as a FM transmitter. The modulated signal will be given out through the GPIO pin 4 of the Raspberry Pi. We can simply attach a normal wire of 20 cm maximum to this pin to act as an antenna.

CHAPTER 2

GENERAL ASPECTS AND TECHNOLOGY

Raspberry Pi is an ARM cortex based popular development board designed for electronics engineers and makers. It’s a single board computer working on low power. With the processing speed and memory, Raspberry Pi can be used for performing different functions at a time, like a normal PC, and hence it is called Mini Computer in your palm. With its powerful features and easy to use functionalities, raspberry pi has become one of the most popular open source hardware platform to build a long range of projects - from simple electronics projects for students and beginners to high end industrial applications.

The raspberry pi comes in two models, they are model A and model B. The main difference between model A and model B is USB port. Model A board will consume less power and that does not include an Ethernet port. But, the model B board includes an Ethernet port and designed in china. The raspberry pi comes with a set of open source technologies, i.e. communication and multimedia web technologies.In the year 2014, the foundation of the raspberry pi board launched the computer module, that packages a model B raspberry pi board into module for use as a part of embedded systems, to encourage their use.

The raspberry pi boards are used in many applications like Media streamer, Arcade machine, Tablet computer, Home automation, Carputer, Internet radio, Controlling robots, Cosmic Computer, Hunting for meteorites, Coffee and also in raspberry pi based projects.

CHAPTER 3

SYSTEM SPECIFICATION AND DESIGN

We use a Rasbian Jessie installed Raspberry Pi 3. The raspberry pi board comprises a program memory (RAM), processor and graphics chip, CPU, GPU, Ethernet port, GPIO pins, Xbee socket, UART, power source connector. And various interfaces for other external devices. It also requires mass storage, for that we use an SD flash memory card. So that raspberry pi board will boot from this SD card similarly as a PC boots up into windows from its hard disk..

Essential hardware specifications of raspberry pi board mainly include

- SD card containing Linux OS
- USB keyboard
- Monitor
- Power supply
- HDMI video cable
- USB mouse
- Powered USB hub case
- Internet connection
- Model   B: USB WiFi adaptor is used and internet connection is LAN cable and
- A wire(20 cm) which serves as a temporary antenna

CHAPTER 4

IMPLEMENTATION

Converting Pi into FM transmitter

Step 1: Create a *New Folder* (directory) inside which we will place all our required program files.

a) Main program:

```
#include
<iostream>
             #include "wave_reader.h"
             #include "transmitter.h"
             #include <cstdlib>
             #include <csignal>

             using namespace std;

             bool stop = false;
             Transmitter* transmitter = NULL;

             AudioFormat* getFormat(string filename) {
                 stop = false;
                 WaveReader* reader = new WaveReader(filename, stop);
                 AudioFormat* format = reader->getFormat();
                 delete reader;
                 return format;
             }

             void sigIntHandler(int sigNum)
             {
                 if (transmitter != NULL) {
                     cout << "Stopping..." << endl;
                     transmitter->stop();
                     stop = true;
                 }
             }

             int main(int argc, char** argv)
             {
                 double frequency = 100.0;
                 bool loop = false;
                 string filename;
```

```
    bool showUsage = true;
    for (int i = 1; i < argc; i++) {
        if (string("-f") == argv[i]) {
            if (i < argc - 1) {
                frequency = ::atof(argv[i + 1]);
                i++;
            }
        } else if (string("-r") == argv[i]) {
            loop = true;
        } else {
            if (i == argc - 1) {
                showUsage = false;
                filename = argv[i];
            }
        }
    }
    if (showUsage) {
        cout << "Usage: " << argv[0] << " [-f frequency] [-r] FILE" <<
endl;
        return 0;
    }

    signal(SIGINT, sigIntHandler);

    try {
        transmitter = Transmitter::getInstance();

        if (filename != "-") {
            AudioFormat* format = getFormat(filename);
            cout << "Playing: " << filename << ", "
                 << format->sampleRate << " Hz, "
                 << format->bitsPerSample << " bits, "
                 << ((format->channels > 0x01) ? "stereo" : "mono") <<
endl;
            delete format;
        } else {
            cout << "Playing: STDIN" << endl;
        }

        transmitter->play(filename, frequency, loop);
    } catch (exception &error) {
        cout << "Error: " << error.what() << endl;
        return 1;
    }
```

```
                return 0;
            }
```

b) Transmitting code:

```
#include
"transmitter.h"
                  #include "wave_reader.h"
                  #include <sstream>
                  #include <cmath>
                  #include <string.h>
                  #include <stdio.h>
                  #include <unistd.h>
                  #include <fcntl.h>
                  #include <sys/mman.h>

                  using std::ostringstream;

                  #define GPIO_BASE 0x00200000
                  #define CLK0_BASE 0x00101070
                  #define CLK0DIV_BASE 0x00101074
                  #define TCNT_BASE 0x00003004

                  #define ACCESS(base, offset) *(volatile unsigned*)((int)base + offset)
                  #define ACCESS64(base, offset) *(volatile unsigned long
                  long*)((int)base + offset)

                  bool Transmitter::transmitting = false;
                  bool Transmitter::restart = false;
                  unsigned Transmitter::clockDivisor = 0;
                  unsigned Transmitter::frameOffset = 0;
                  vector<float>* Transmitter::buffer = NULL;
                  void* Transmitter::peripherals = NULL;

                  Transmitter::Transmitter()
                  {
                      bool isBcm2835 = true;

                      FILE* pipe = popen("uname -m", "r");
                      if (pipe) {
                          char buffer[64];
                          string machine = "";
                          while (!feof(pipe)) {
                              if (fgets(buffer, 64, pipe)) {
                                  machine += buffer;
```

```
            }
        }
        pclose(pipe);

        machine = machine.substr(0, machine.length() - 1);
        if (machine != "armv6l") {
            isBcm2835 = false;
        }
    }

    int memFd;
    if ((memFd = open("/dev/mem", O_RDWR | O_SYNC)) < 0) {
        throw ErrorReporter("Cannot open /dev/mem (permission 
denied)");
    }

    peripherals = mmap(NULL, 0x002FFFFF, PROT_READ | PROT_WRITE, 
MAP_SHARED, memFd, isBcm2835 ? 0x20000000 : 0x3F000000);
    close(memFd);
    if (peripherals == MAP_FAILED) {
        throw ErrorReporter("Cannot obtain access to peripherals (mmap 
error)");
    }
}

Transmitter::~Transmitter()
{
    munmap(peripherals, 0x002FFFFF);
}

Transmitter* Transmitter::getInstance()
{
    static Transmitter instance;
    return &instance;
}

void Transmitter::play(string filename, double frequency, bool loop)
{
    if (transmitting) {
        throw ErrorReporter("Cannot play, transmitter already in 
use");
    }

    transmitting = true;
    forceStop = false;
    WaveReader* reader = new WaveReader(filename != "-" ? filename :
string(), forceStop);
    AudioFormat* format = reader->getFormat();

    clockDivisor = (unsigned)((500 << 12) / frequency + 0.5);
    unsigned bufferFrames = (unsigned)((unsigned long long)format-
>sampleRate * BUFFER_TIME / 1000000);

    frameOffset = 0;
    restart = false;

    try {
        vector<float>* frames = reader->getFrames(bufferFrames,
forceStop);
        if (frames == NULL) {
            delete format;
            delete reader;
            return;
        }
        eof = frames->size() < bufferFrames;
        buffer = frames;

        pthread_t thread;
        void* params = (void*)&format->sampleRate;

        int returnCode = pthread_create(&thread, NULL,
&Transmitter::transmit, params);
        if (returnCode) {
            delete reader;
            delete format;
            delete frames;
            ostringstream oss;
            oss << "Cannot create new thread (code: " << returnCode <<
")";
            throw ErrorReporter(oss.str());
        }

        usleep(BUFFER_TIME / 2);

        while (!forceStop) {
            while (!eof && !forceStop) {
                if (buffer == NULL) {
                    if (!reader->setFrameOffset(frameOffset +
bufferFrames)) {
                        break;
                    }
                    frames = reader->getFrames(bufferFrames,
forceStop);
                    if (frames == NULL) {
                        forceStop = true;
                        break;
                    }
                    eof = frames->size() < bufferFrames;
                    buffer = frames;
                }
                usleep(BUFFER_TIME / 2);
            }
            if (loop && !forceStop) {
                frameOffset = 0;
                restart = true;
                if (!reader->setFrameOffset(0)) {
                    break;
                }
                frames = reader->getFrames(bufferFrames, forceStop);
                if (frames == NULL) {
                    break;
                }
                eof = frames->size() < bufferFrames;
                buffer = frames;
                usleep(BUFFER_TIME / 2);
            } else {
                forceStop = true;
            }
        }
        transmitting = false;

        pthread_join(thread, NULL);
   } catch (ErrorReporter &error) {
        delete reader;
        delete format;
        throw error;
    }

    delete reader;
    delete format;
 }

void* Transmitter::transmit(void* params)
{
    unsigned long long current, start, playbackStart;
    unsigned offset, length, temp;
    vector<float>* frames = NULL;
    float* data;
    float value;
#ifndef NO_PREEMP
    float prevValue = 0.0;
#endif

    unsigned sampleRate = *(unsigned*)(params);

#ifndef NO_PREEMP
    float preemp = 0.75 - 250000.0 / (float)(sampleRate * 75);
#endif

    ACCESS(peripherals, GPIO_BASE) = (ACCESS(peripherals, GPIO_BASE) &
0xFFFF8FFF) | (0x01 << 14);
    ACCESS(peripherals, CLK0_BASE) = (0x5A << 24) | (0x01 << 9) |
(0x01 << 4) | 0x06;

    playbackStart = ACCESS64(peripherals, TCNT_BASE);
    current = playbackStart;

    while (transmitting) {
        start = current;
        while ((buffer == NULL) && transmitting) {
            usleep(1);
            current = ACCESS64(peripherals, TCNT_BASE);
        }
        if (!transmitting) {
            break;
        }
        if (restart) {
            playbackStart = current;
            start = current;
            restart = false;
        }
        frames = buffer;
        frameOffset = (current - playbackStart) * (sampleRate) /
1000000;
        buffer = NULL;

        length = frames->size();
        data = &(*frames)[0];

        offset = 0;

        while (true) {
            temp = offset;
```

```
                if (offset >= length) {
                    offset -= length;
                    break;
                }

                value = data[offset];

#ifndef NO_PREEMP
                value = value + (value - prevValue) * preemp;
                value = (value < -1.0) ? -1.0 : ((value > 1.0) ? 1.0 :
value);
#endif

                ACCESS(peripherals, CLK0DIV_BASE) = (0x5A << 24) |
((clockDivisor) - (int)(round(value * 16.0)));
                while (temp >= offset) {
                    asm("nop");
                    current = ACCESS64(peripherals, TCNT_BASE);
                    offset = (current - start) * (sampleRate) / 1000000;
                }
#ifndef NO_PREEMP
                prevValue = value;
#endif
        }

        delete frames;
    }

    ACCESS(peripherals, CLK0_BASE) = (0x5A << 24);

    return NULL;
}

void Transmitter::stop()
{
    forceStop = true;
}
```

c) Wave reader code:

```
#include
"wave_reader.h"
#include "error_reporter.h"
#include <sstream>
#include <string.h>
```

```
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>

using std::ostringstream;
using std::exception;

WaveReader::WaveReader(string filename, bool &forceStop) :
    filename(filename), currentOffset(0)
{
    char* headerData = (char*)((void*)&header);
    vector<char>* data;
    unsigned bytesToRead, headerOffset;
    ostringstream oss;

    if (!filename.empty()) {
        fileDescriptor = open(filename.c_str(), O_RDONLY);
    } else {
        fcntl(STDIN_FILENO, F_SETFL, fcntl(STDIN_FILENO, F_GETFL, 0) |
O_NONBLOCK);
        fileDescriptor = STDIN_FILENO;
    }

    if (fileDescriptor == -1) {
        oss << "Cannot open " << getFilename() << ", file does not
exist";
        throw ErrorReporter(oss.str());
    }

    try {
        bytesToRead = sizeof(PCMWaveHeader::chunkID) +
sizeof(PCMWaveHeader::chunkSize) +
            sizeof(PCMWaveHeader::format);
        data = readData(bytesToRead, true, forceStop);
        if (data == NULL) {
            throw ErrorReporter("Cannot obtain header, program
interrupted");
        }
        memcpy(headerData, &(*data)[0], bytesToRead);
        headerOffset = bytesToRead;
        delete data;

        if ((string(header.chunkID, 4) != string("RIFF")) ||
(string(header.format, 4) != string("WAVE"))) {
            oss << "Error while opening " << getFilename() << ", WAVE
file expected";
```

```
            throw ErrorReporter(oss.str());
        }

        bytesToRead = sizeof(PCMWaveHeader::subchunk1ID) +
sizeof(PCMWaveHeader::subchunk1Size);
        data = readData(bytesToRead, true, forceStop);
        if (data == NULL) {
            throw ErrorReporter("Cannot obtain header, program
interrupted");
        }
        memcpy(&headerData[headerOffset], &(*data)[0], bytesToRead);
        headerOffset += bytesToRead;
        delete data;

        unsigned subchunk1MinSize = sizeof(PCMWaveHeader::audioFormat)
+ sizeof(PCMWaveHeader::channels) +
            sizeof(PCMWaveHeader::sampleRate) +
sizeof(PCMWaveHeader::byteRate) + sizeof(PCMWaveHeader::blockAlign) +
            sizeof(PCMWaveHeader::bitsPerSample);
        if ((string(header.subchunk1ID, 4) != string("fmt ")) ||
(header.subchunk1Size < subchunk1MinSize)) {
            oss << "Error while opening " << getFilename() << ", data
corrupted";
            throw ErrorReporter(oss.str());
        }

        data = readData(header.subchunk1Size, true, forceStop);
        if (data == NULL) {
            throw ErrorReporter("Cannot obtain header, program
interrupted");
        }
        memcpy(&headerData[headerOffset], &(*data)[0],
subchunk1MinSize);
        headerOffset += subchunk1MinSize;
        delete data;

        if ((header.audioFormat != WAVE_FORMAT_PCM) ||
            (header.byteRate != (header.bitsPerSample >> 3) *
header.channels * header.sampleRate) ||
            (header.blockAlign != (header.bitsPerSample >> 3) *
header.channels) ||
            (((header.bitsPerSample >> 3) != 1) &&
((header.bitsPerSample >> 3) != 2))) {
            oss << "Error while opening " << getFilename() << ",
unsupported WAVE format";
            throw ErrorReporter(oss.str());
        }

        bytesToRead = sizeof(PCMWaveHeader::subchunk2ID) +
sizeof(PCMWaveHeader::subchunk2Size);
        data = readData(bytesToRead, true, forceStop);
        if (data == NULL) {
            throw ErrorReporter("Cannot obtain header, program
interrupted");
        }
        memcpy(&headerData[headerOffset], &(*data)[0], bytesToRead);
        headerOffset += bytesToRead;
        delete data;

        if (string(header.subchunk2ID, 4) != string("data")) {
            oss << "Error while opening " << getFilename() << ", data
corrupted";
            throw ErrorReporter(oss.str());
        }
    } catch (ErrorReporter &error) {
        if (fileDescriptor != STDIN_FILENO) {
            close(fileDescriptor);
        }
        throw error;
    }

    if (fileDescriptor != STDIN_FILENO) {
        dataOffset = lseek(fileDescriptor, 0, SEEK_CUR);
    }
}

WaveReader::~WaveReader()
{
    if (fileDescriptor != STDIN_FILENO) {
        close(fileDescriptor);
    }
}

vector<char>* WaveReader::readData(unsigned bytesToRead, bool
headerBytes, bool &forceStop)
{
    unsigned bytesRead = 0;
    vector<char>* data = new vector<char>();
    data->resize(bytesToRead);

    while ((bytesRead < bytesToRead) && !forceStop) {
        int bytes = read(fileDescriptor, &(*data)[bytesRead],
bytesToRead - bytesRead);
        if (((bytes == -1) && ((fileDescriptor != STDIN_FILENO) ||
(errno != EAGAIN))) ||
            (((unsigned)bytes < bytesToRead) && headerBytes &&
(fileDescriptor != STDIN_FILENO))) {
            delete data;

            if (fileDescriptor != STDIN_FILENO) {
                close(fileDescriptor);
            }

            ostringstream oss;
            oss << "Error while reading " << getFilename() << ", file
is corrupted";
            throw ErrorReporter(oss.str());
        }
        if (bytes > 0) {
            bytesRead += bytes;
        }
        if (bytesRead < bytesToRead) {
            if (fileDescriptor != STDIN_FILENO) {
                data->resize(bytes);
                break;
            }
            usleep(1);
        }
    }

    if (!headerBytes) {
        currentOffset += bytesRead;
    }

    if (forceStop) {
        delete data;
        data = NULL;
    }

    return data;
}

vector<float>* WaveReader::getFrames(unsigned frameCount, bool
&forceStop) {
    unsigned bytesToRead, bytesLeft, bytesPerFrame, offset;
    vector<float>* frames = new vector<float>();
    vector<char>* data;
    bytesPerFrame = (header.bitsPerSample >> 3) * header.channels;
    bytesToRead = frameCount * bytesPerFrame;
    bytesLeft = header.subchunk2Size - currentOffset;
    if (bytesToRead > bytesLeft) {
        bytesToRead = bytesLeft - bytesLeft % bytesPerFrame;
        frameCount = bytesToRead / bytesPerFrame;
    }

    try {
        data = readData(bytesToRead, false, forceStop);
    } catch (ErrorReporter &error) {
        delete frames;
        throw error;
    }
    if (data == NULL) {
        delete frames;
        return NULL;
    }
    if (data->size() < bytesToRead) {
        frameCount = data->size() / bytesPerFrame;
    }

    for (unsigned i = 0; i < frameCount; i++) {
        offset = bytesPerFrame * i;
        if (header.channels != 1) {
            if (header.bitsPerSample != 8) {
                frames->push_back(((int)(signed char)(*data)[offset +
1] + (int)(signed char)(*data)[offset + 3]) / (float)0x100);
            } else {
                frames->push_back(((int)(*data)[offset] +
(int)(*data)[offset + 1]) / (float)0x100 - 1.0f);
            }
        } else {
            if (header.bitsPerSample != 8) {
                frames->push_back((signed char)(*data)[offset + 1] /
(float)0x80);
            } else {
                frames->push_back((*data)[offset] / (float)0x80 -
1.0f);
            }
        }
    }

    delete data;
    return frames;
```

```
}

bool WaveReader::setFrameOffset(unsigned frameOffset) {
    if (fileDescriptor != STDIN_FILENO) {
        currentOffset = frameOffset * (header.bitsPerSample >> 3) *
header.channels;
        if (lseek(fileDescriptor, dataOffset + currentOffset,
SEEK_SET) == -1) {
            return false;
        }
    }
    return true;
}

string WaveReader::getFilename()
{
    return fileDescriptor != STDIN_FILENO ? filename : "STDIN";
}

AudioFormat* WaveReader::getFormat()
{
    AudioFormat* format = new AudioFormat;
    format->channels = header.channels;
    format->sampleRate = header.sampleRate;
    format->bitsPerSample = header.bitsPerSample;
    return format;
```

d) Error –reporting code:

```
#include
"error_reporter.h"

ErrorReporter::ErrorReporter(string message) :
    errorMessage(message)
{
}

ErrorReporter::~ErrorReporter() throw()
{
}

const char* ErrorReporter::what() const throw()
{
        return errorMessage.c_str();
```

Step 2: Download the g++ and gcc compilers

Step 3: Link the program into the directory that is created.

Step 4: Compilng the program.

Step 5: The final step is launch the program. While launching the program we have to mention the frequency at which we want to broadcast and the name of the audio file which we want to play.

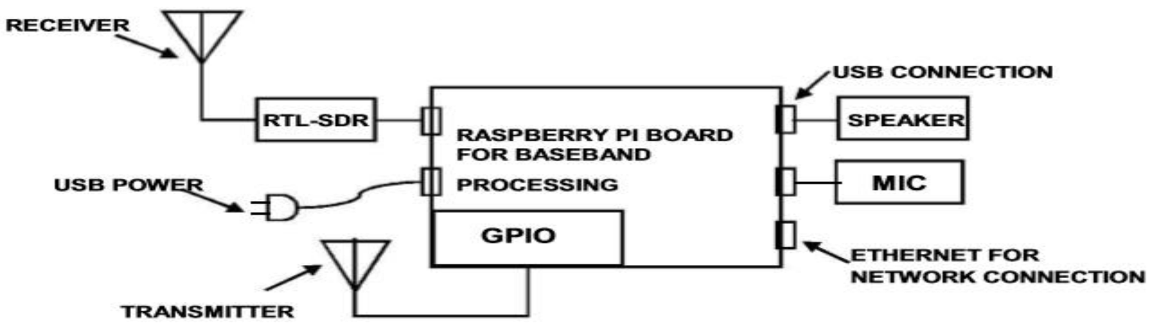
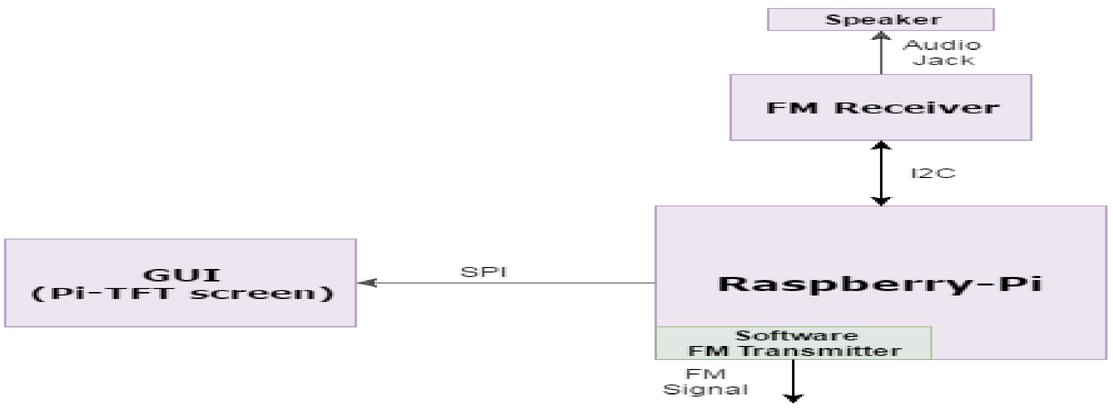


Fig. 1. Overall block diagram



Testing your Raspberry Pi FM Transmitter

Once you have launched the program and you get the playing message as shown , we can attach an antenna to the GPIO pin 4 of the Pi, we have used a normal hook up wire .

Broadcasting live voice using Pi

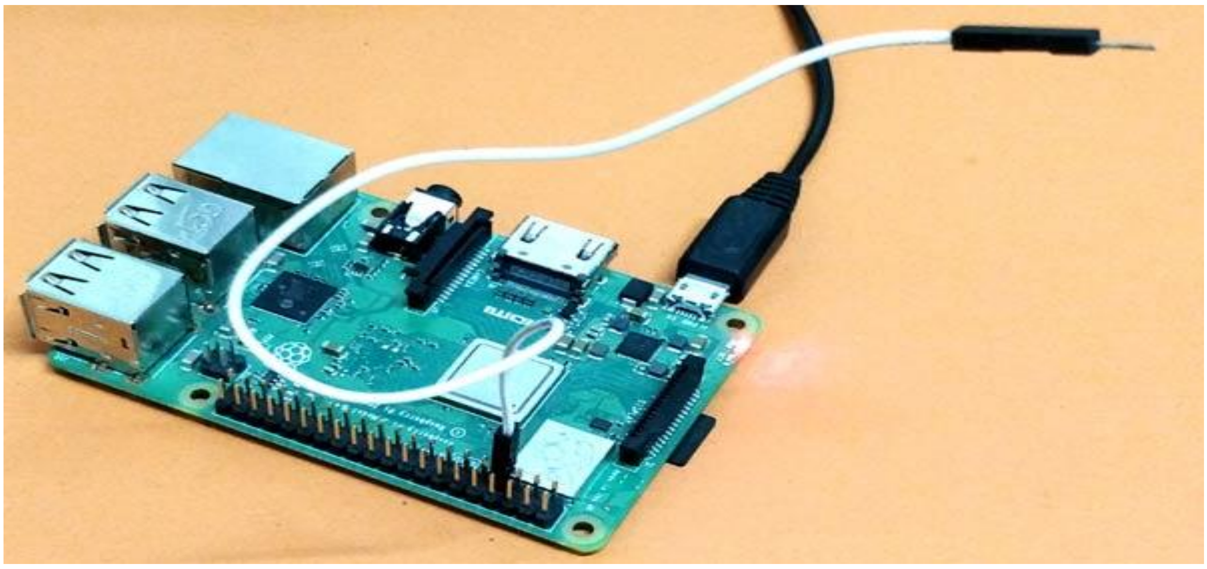
While it is fun to play pre-recorded music clips, it would be more appealing if we can broadcast live voice using you Pi. Simply connect a microphone to the USB port of Pi and use the arecord function for voice inputs

CHAPTER 5

RESULT and DISCUSSIONS:

Now we can easily tune into our own personal FM radio by specifying the particular frequency.

We can also implement this in our college and use it as our college radio for various activities.



CONCLUSION

We may have many advantages of setting up our own FM station but you do not want to be using this as a long term, stationary FM transmitter. If any of your signals interfere with air traffic control, emergency services (police, ambulance), etc they will come looking for your signal and ultimately you. Looking at a spectrum analyzer with a frequency of 88.7MHz there are harmonics all across the VHF band. Centering on 121.5MHZ, your aviation band emergency frequency, there are harmonics and intermod products all through aviation VHF band (118-136MHz,) and VHF navaids (108-117MHz). If you use this you will splash onto these frequencies. If nobody complains, nobody will come looking for the signal. If this does interfere with aircraft, I guarantee you someone, (FAA in US, Industry Canada in CA) will come by with a DF and find you and you can be charged. Use with caution.

REFERENCES

https://www.instructables.com/id/Raspberry-Pi-Radio-Transmitter/
https://circuitdigest.com/microcontroller-projects/raspberry-pi-fm-transmitter
https://core.ac.uk/download/pdf/81941726.pdf