

Tuning into MnM

Problem statement

To build an FM radio receiver that does not need to be manually tuned, but is able to hop on to the nearest channel that is broadcasting.

Team

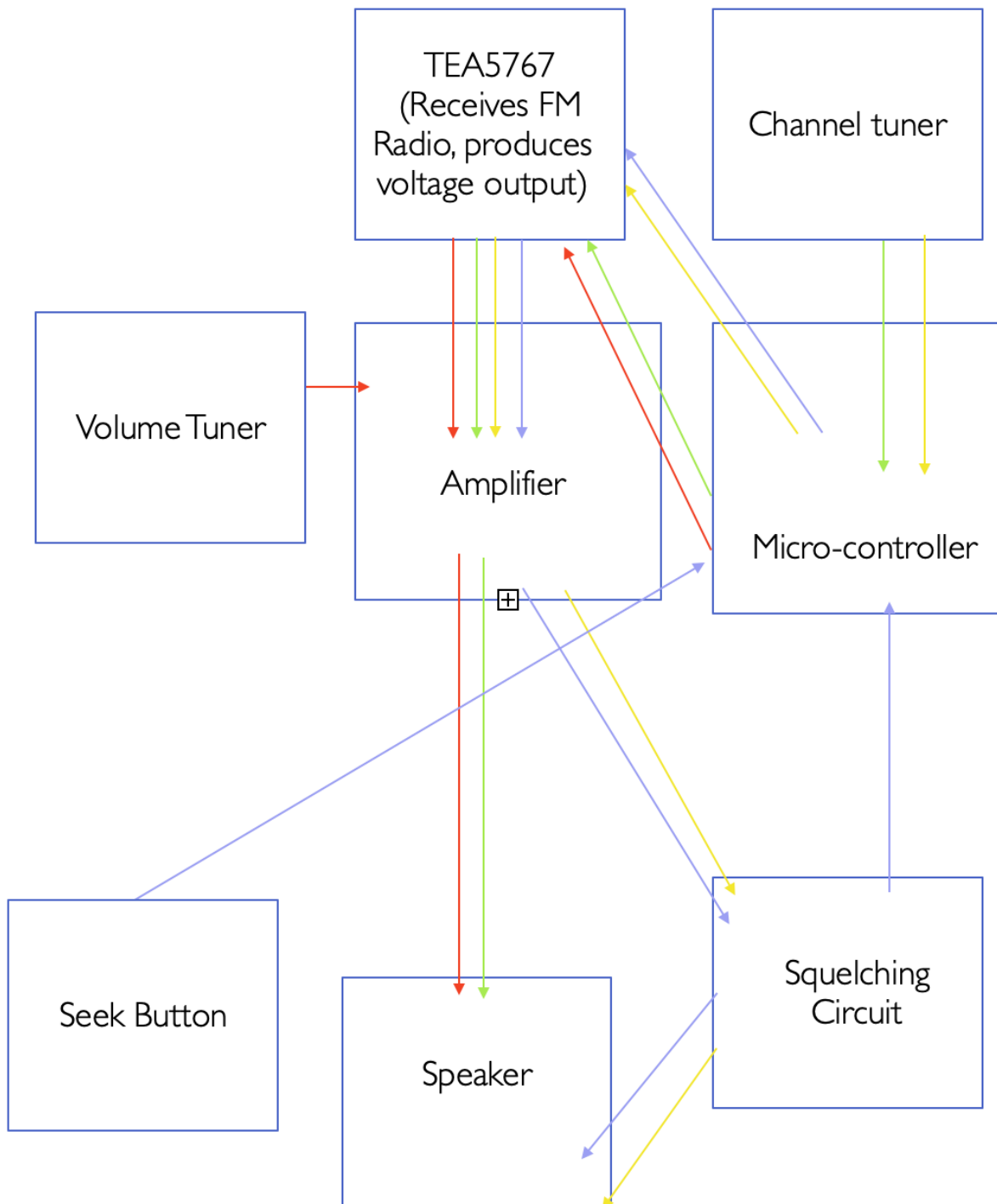
B.Malavika (190260014)
Moysha Gera (190260031)
Nabeel Ahmed (19B030016)

Description

We shall use the TEA5767 Radio Module. We shall filter and amplify the demodulated output to produce sound. We shall add features in the following order, which act as intermediate milestones:

- 1) **Amplification:** Building an amplifier with volume tuning: We shall take the demodulated output produced from the TEA5767 radio module and then build an amplifier circuit to be connected to a speaker which can play a fixed channel, but at different volumes of choice.
- 2) **Channel Tuning:** This is an easy addition to part 1 and doesn't really involve much work from the micro-controller
- 3) **Squelching:** We can modify part 2 to squelch, i.e selectively mute channels that are just noise, while tuning. Thus the tuning will become a more user-friendly task as in between the channels, one will not have to hear white noise. The main feature of this particular setup is identifying the noise from the actual channel and involves hardware and software work. This is probably going to be the hardest part of the project.
- 4) **Seeking:** Once we have separated the channels from the noise, we can modify the above setup to be able to skip to channels that are not noise.

Block diagram



Design details

The TEA5767 Radio module demodulates the radio waves received via the antenna and produces an audio output in the form of a voltage signal. Thus the sole purpose of the radio module is to receive the radio signal and separate the carrier wave from the audio signal.

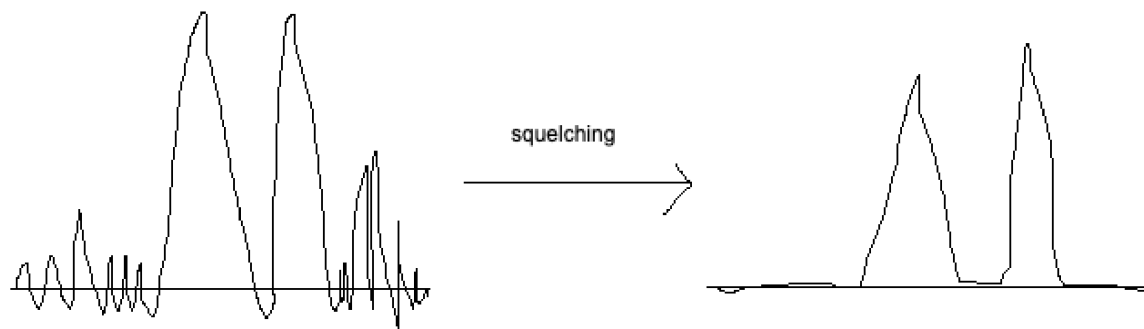
This signal goes through a volume tuner, which is basically an Op-amp based amplifier circuit and we can vary the gain by changing the potentiometer values. There are a few additional functionalities that we have added that is described in the following paragraph:

There is a channel tuner, which consists of a simple potentiometer that acts as input into the Arduino and the code changes the frequency of the station based on the voltage of the input. Apart from that, we have a seek button that allows us to hop to the nearest channel which has a “strong enough” audio signal. The precise meaning of strong enough is described later.

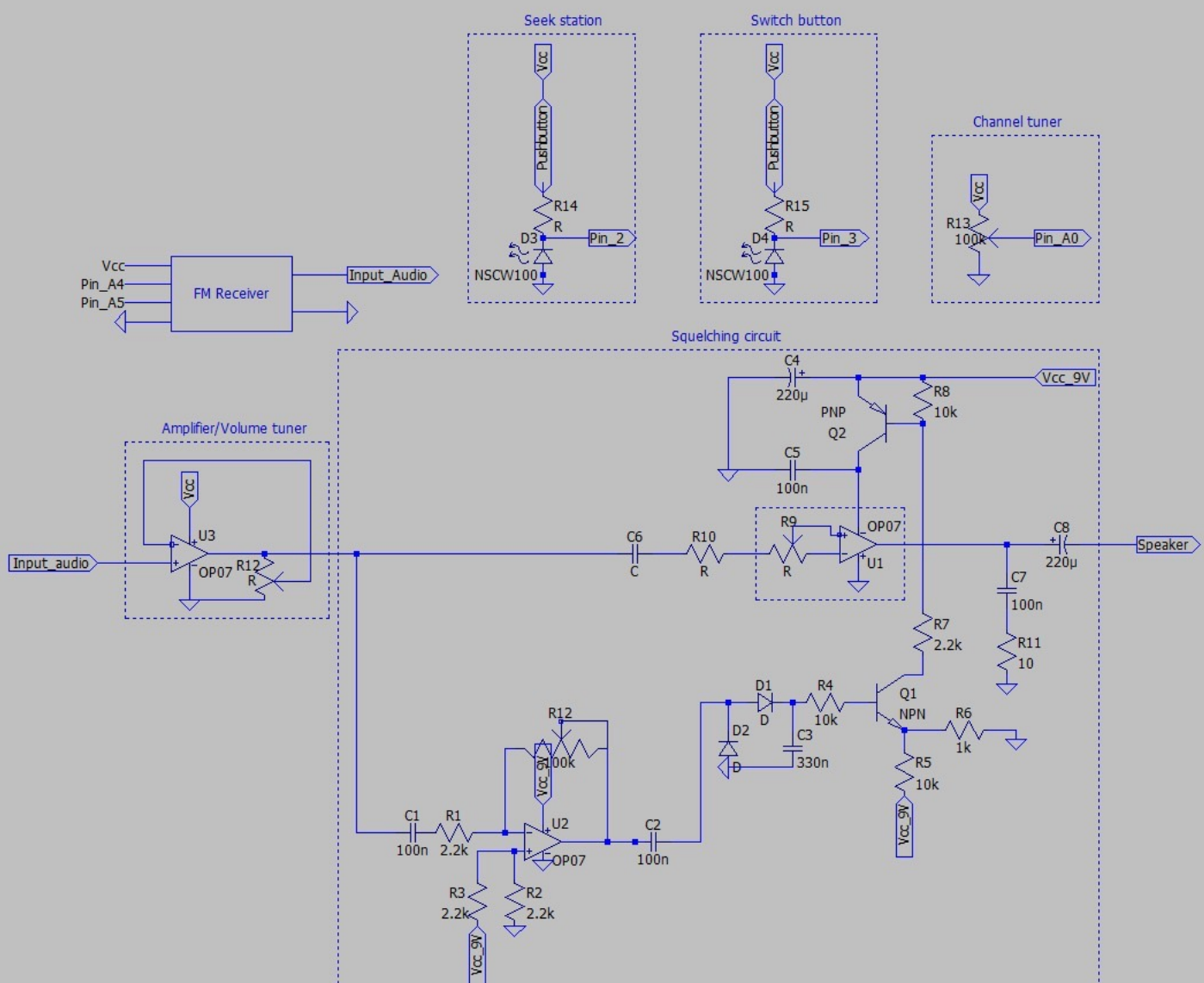
Now the audio signal that is outputted by the TEA5767 FM Receiver module is put through a squelching circuit. This circuit attenuates the background noise that is present between the channel frequencies when tuning the radio. The circuit consists of an LM386 low-voltage audio amplifier and an op-amp. The former delivers drive for the external speaker and the gain is controlled by changing the value of R12, and the 741 op-amp regulates the squelch's edge level. The audio input is connected to the input of both the audio amplifier and the op-amp. The output of the amplifier circuit is changed to DC by D1 and D2. This results in the forward biasing of the Q1 bjt. Once Q1 is active, it pulls R7 to almost ground potential and triggers Q2. This powers the audio amplifier. A constant average input signal will permit U1 to remain on. However, once the audio falls just enough to abruptly stops, the process will reverse and U1 switches off.

This circuit ‘squelches’ or ‘mutes’ the noise, and selectively amplifies the channels with a considerably strong audio signal. This audio signal is identified on the basis of frequency as well as amplitude. A simple factor for differentiating between noise and a channel is that channels have larger signal amplitudes. But this is not enough. The squelching circuit contains filters that block the noise based on frequencies too. Since noise is inherently random and the audio has a fixed frequency range, suitably tuning the parameters in the squelch circuit can allow us to differentiate between noise and a channel.

We can then feed back the squelched output into an analog pin of the arduino to implement a seeking feature. The seeking function hops on to the nearest non-zero channel.



Circuit diagram and Image



Preview File Edit View Go Tools Window Help

Circuit Diagram.jpeg

Search

Seek station

Switch button

Channel tuner

FM Receiver

Input_Audio

Pin_A4

Pin_A5

NSCW100

Pin_2

Pin_3

Pin_A0

Amplifier/Volume tuner

U3

OP07

R12

R13

R14

R15

R16

R17

R18

R19

R20

R21

R22

R23

R24

R25

R26

R27

R28

R29

R30

R31

R32

R33

R34

R35

R36

R37

R38

R39

R40

R41

R42

R43

R44

R45

R46

R47

R48

R49

R50

R51

R52

R53

R54

R55

R56

R57

R58

R59

R60

R61

R62

R63

R64

R65

R66

R67

R68

R69

R70

R71

R72

R73

R74

R75

R76

R77

R78

R79

R80

R81

R82

R83

R84

R85

R86

R87

R88

R89

R90

R91

R92

R93

R94

R95

R96

R97

R98

R99

R100

R101

R102

R103

R104

R105

R106

R107

R108

R109

R110

R111

R112

R113

R114

R115

R116

R117

R118

R119

R120

R121

R122

R123

R124

R125

R126

R127

R128

R129

R130

R131

R132

R133

R134

R135

R136

R137

R138

R139

R140

R141

R142

R143

R144

R145

R146

R147

R148

R149

R150

R151

R152

R153

R154

R155

R156

R157

R158

R159

R160

R161

R162

R163

R164

R165

R166

R167

R168

R169

R170

R171

R172

R173

R174

R175

R176

R177

R178

R179

R180

R181

R182

R183

R184

R185

R186

R187

R188

R189

R190

R191

R192

R193

R194

R195

R196

R197

R198

R199

R200

R201

R202

R203

R204

R205

R206

R207

R208

R209

R210

R211

R212

R213

R214

R215

R216

R217

R218

R219

R220

R221

R222

R223

R224

R225

R226

R227

R228

R229

R230

R231

R232

R233

R234

R235

R236

R237

R238

R239

R240

R241

R242

R243

R244

R245

R246

R247

R248

R249

R250

R251

R252

R253

R254

R255

R256

R257

R258

R259

R260

R261

R262

R263

R264

R265

R266

R267

R268

R269

R270

R271

R272

R273

R274

R275

R276

R277

R278

R279

R280

R281

R282

R283

R284

R285

R286

R287

R288

R289

R290

R291

R292

R293

R294

R295

R296

R297

R298

R299

R300

R301

R302

R303

R304

R305

R306

R307

R308

R309

R310

R311

R312

R313

R314

R315

R316

R317

R318

R319

R320

R321

R322

R323

R324

R325

R326

R327

R328

R329

R330

R331

R332

R333

R334

R335

R336

R337

R338

R339

R340

R341

R342

R343

R344

R345

R346

R347

R348

R349

R350

R351

R352

R353

R354

R355

R356

R357

R358

R359

R360

R361

R362

R363

R364

R365

R366

R367

R368

R369

R370

R371

R372

R373

R374

R375

R376

R377

R378

R379

R380

R381

R382

R383

R384

R385

R386

R387

R388

R389

R390

R391

R392

R393

R394

R395

R396

R397

R398

R399

R400

R401

R402

R403

R404

R405

R406

R407

R408

R409

R410

R411

R412

R413

R414

R415

R416

R417

R418

R419

R420

R421

R422

R423

R424

R425

R426

R427

R428

R429

R430

R431

R432

R433

R434

R435

R436

R437

R438

R439

R440

R441

R442

R443

R444

R445

R446

R447

R448

R449

R450

R451

R452

R453

R454

R455

R456

R457

R458

R459

R460

R461

R462

R463

R464

R465

R466

R467

R468

R469

R470

R471

R472

R473

R474

R475

R476

R477

R478

R479

R480

R481

R482

R483

R484

R485

R486

R487

R488

R489

R490

R491

R492

R493

R494

R495

R496

R497

R498

R499

R500

R501

R502

R503

R504

R505

R506

R507

R508

R509

R510

R511

R512

R513

R514

R515

R516

R517

R518

R519

R520

R521

R522

R523

R524

R525

R526

R527

R528

R529

R530

R531

R532

R533

R534

R535

R536

R537

R538

R539

R540

R541

R542

R543

R544

R545

R546

R547

R548

R549

R550

R551

R552

R553

R554

R555

R556

R557

R558

R559

R560

R561

R562

R563

R564

R565

R566

R567

R568

R569

R570

R571

R572

R573

R574

R575

R576

R577

R578

R579

R580

R581

R582

R583

R584

R585

R586

R587

R588

R589

R590

R591

R592

R593

R594

R595

R596

R597

R598

R599

R600

R601

R602

R603

R604

R605

R606

R607

R608

R609

R610

R611

R612

R613

R614

R615

R616

R617

R618

R619

R620

R621

R622

R623

R624

R625

R626

R627

R628

R629

R630

R631

R632

R633

R634

R635

R636

R637

R638

R639

R640

R641

R642

R643

R644

R645

R646

R647

R648

R649

R650

R651

R652

R653

R654

R655

R656

R657

R658

R659

R660

R661

R662

R663

R664

R665

R666

R667

R668

R669

R670

R671

R672

R673

R674

R675

R676

R677

R678

R679

R680

R681

R682

R683

R684

R685

R686

R687

R688

R689

R690

R691

R692

R693

R694

R695

R696

R697

R698

R699

R700

R701

R702

R703

R704

R705

R706

R707

R708

R709

R710

R711

R712

R713

R714

R715

R716

R717

R718

R719

R720

R721

R722

R723

R724

R725

R726

R727

R728

R729

R730

R731

R732

R733

R734

R735

R736

R737

R738

R739

R740

R741

R742

R743

R744

R745

R746

R747

R748

R749

R750

R751

R752

R753

R754

R755

R756

R757

R758

R759

R760

R761

R762

R763

R764

R765

R766

R767

R768

R769

R770

R771

R772

R773

R774

R775

R776

R777

R778

R779

R780

R781

R782

R783

R784

R785

R786

R787

R788

R789

R790

R791

R792

R793

R794

R795

R796

R797

R798

R799

R800

R801

R802

R803

R804

R805

R806

R807

R808

R809

R810

R811

R812

R813

R814

R815

R816

R817

R818

R819

R820

R821

R822

R823

R824

R825

R826

R827

R828

R829

R830

R831

R832

R833

R834

R835

R836

R837

R838

R839

R840

R841

R842

R843

R844

R845

R846

R847

R848

R849

R850

R851

R852

R853

R854

R855

R856

R857

R858

R859

R860

R861

R862

R863

R864

R865

R866

R867

R868

R869

R870

R871

R872

R873

R874

R875

R876

R877

R878

R879

R880

R881

R882

R883

R884

R885

R886

R887

R888

R889

R890

R891

R892

R893

R894

R895

R896

R897

R898

R899

R900

R901

R902

R903

R904

R905

R906

R907

R908

R909

R910

R911

R912

R913

R914

R915

R916

R917

R918

R919

R920

R921

R922

R923

R924

R925

R926

R927

R928

R929

R930

R931

R932

R933

R934

R935

R936

R937

R938

R939

R940

R941

R942

R943

R944

R945

R946

R947

R948

R949

R950

R951

R952

R953

R954

R955

R956

R957

R958

R959

R960

R961

R962

R963

R964

R965

R966

R967

R968

R969

R970

R971

R972

R973

R974

R975

R976

R977

R978

R979

R980

R981

R982

R983

R984

R985

R986

R987

R988

R989

R990

R991

R992

R993

R994

R995

R996

R997

R998

R999

R1000

R1001

R1002

R1003

R1004

R1005

R1006

R1007

R1008

R1009

R1010

R1011

R1012

R1013

R1014

R1015

R1016

R1017

R1018

R1019

R1020

R1021

R1022

R1023

R1024

R1025

R1026

R1027

R1028

R1029

R1030

R1031

R1032

R1033

R1034

R1035

R1036

R1037

R1038

R1039

R1040

R1041

R1042

R1043

R1044

R1045

R1046

R1047

R1048

R1049

R1050

R1051

R1052

R1053

R1054

R1055

R1056

R1057

R1058

R1059

R1060

R1061

R1062

R1063

R1064

R1065

R1066

R1067

R1068

R1069

R1070

R1071

R1072

R1073

R1074

R1075

R1076

R1077

R1078

R1079

R1080

R1081

R1082

R1083

R1084

R1085

R1086

R1087

R1088

R1089

R1090

R1091

R1092

R1093

R1094

R1095

R1096

R1097

R1098

R1099

R1100

R1101

R1102

R1103

R1104

R1105

R1106

R1107

R1108

R1109

R1110

R1111

R1112

R1113

R1114

R1115

R1116

R1117

R1118

R1119

R1120

R1121

R1122

R1123

R1124

R1125

R1126

R1127

R1128

R1129

R1130

R1131

R1132

R1133

R1134

R1135

R1136

R1137

R1138

R1139

R1140

R1141

R1142

R1143

R1144

R1145

R1146

R1147

R1148

R1149

R1150

R1151

R1152

R1153

R1154

R1155

R1156

R1157

R1158

R1159

R1160

R1161

R1162

R1163

R1164

R1165

R1166

R1167

R1168

R1169

R1170

R1171

R1172

R1173

R1174

R1175

R1176

R1177

R1178

R1179

R1180

R1181

R1182

R1183

R1184

R1185

R1186

R1187

R1188

R1189

R1190

R1191

R1192

R1193

R1194

R1195

R1196

R1197

R1198

R1199

R1200

R1201

R1202

R1203

R1204

R1205

R1206

R1207

R1208

R1209

R1210

R1211

R1212

R1213

R1214

R1215

R1216

R1217

R1218

R1219

R1220

R1221

R1222

R1223

R1224

R1225

R1226

R1227

R1228

R1229

R1230

R1231

R1232

R1233

R1234

R1235

R1236

R1237

R1238

R1239

R1240

R1241

R1242

R1243

R1244

R1245

R1246

R1247

R1248

R1249

R1250

R1251

R1252

R1253

R1254

R1255

R1256

R1257

R1258

R1259

R1260

R1261

R1262

R1263

R1264

R1265

R1266

R1267

R1268

R1269

R1270

R1271

R1272

R1273

R1274

R1275

R1276

R1277

R1278

R1279

R1280

R1281

R1282

R1283

R1284

R1285

R1286

R1287

R1288

R1289

R1290

R1291

R1292

R1293

R1294

R1295

R1296

R1297

R1298

R1299

R1300

R1301

R1302

R1303

R1304

R1305

R1306

R1307

R1308

R1309

R1310

R1311

R1312

R1313

R1314

R1315

R1316

R1317

R1318

R1319

R1320

R1321

R1322

R1323

R1324

R1325

R1326

R1327

R1328

R1329

R1330

R1331

R1332

R1333

R1334

R1335

R1336

R1337

R1338

R1339

R1340

R1341

R1342

R1343

R1344

R1345

R1346

R1347

R1348

R1349

R1350

R1351

R1352

R1353

R1354

R1355

R1356

R1357

R1358

R1359

R1360

R1361

R1362

R1363

R1364

R1365

R1366

R1367

R1368

R1369

R1370

R1371

R1372

R1373

R1374

R1375

R1376

R1377

R1378

R1379

R1380

R1381

R1382

R1383

R1384

R1385

R1386

R1387

R1388

R1389

R1390

R1391

R1392

R1393

R1394

R1395

R1396

R1397

R1398

R1399

R1400

R1401

R1402

R1403

R1404

R1405

R1406

R1407

R1408

R1409

R1410

R1411

R1412

R1413

R1414

R1415

R1416

R1417

R1418

R1419

R1420

R1421

R1422

R1423

R1424

R1425

R1426

R1427

R1428

R1429

R1430

R1431

R1432

R1433

R1434

R1435

R1436

R1437

R1438

R1439

R1440

R1441

R1442

R1443

R1444

R1445

R1446

R1447

R1448

R1449

R1450

R1451

R1452

R1453

R1454

R1455

R1456

R1457

R1458

R1459

R1460

R1461

R1462

R1463

R1464

R1465

R1466

R1467

R1468

R1469

R1470

R1471

R1472

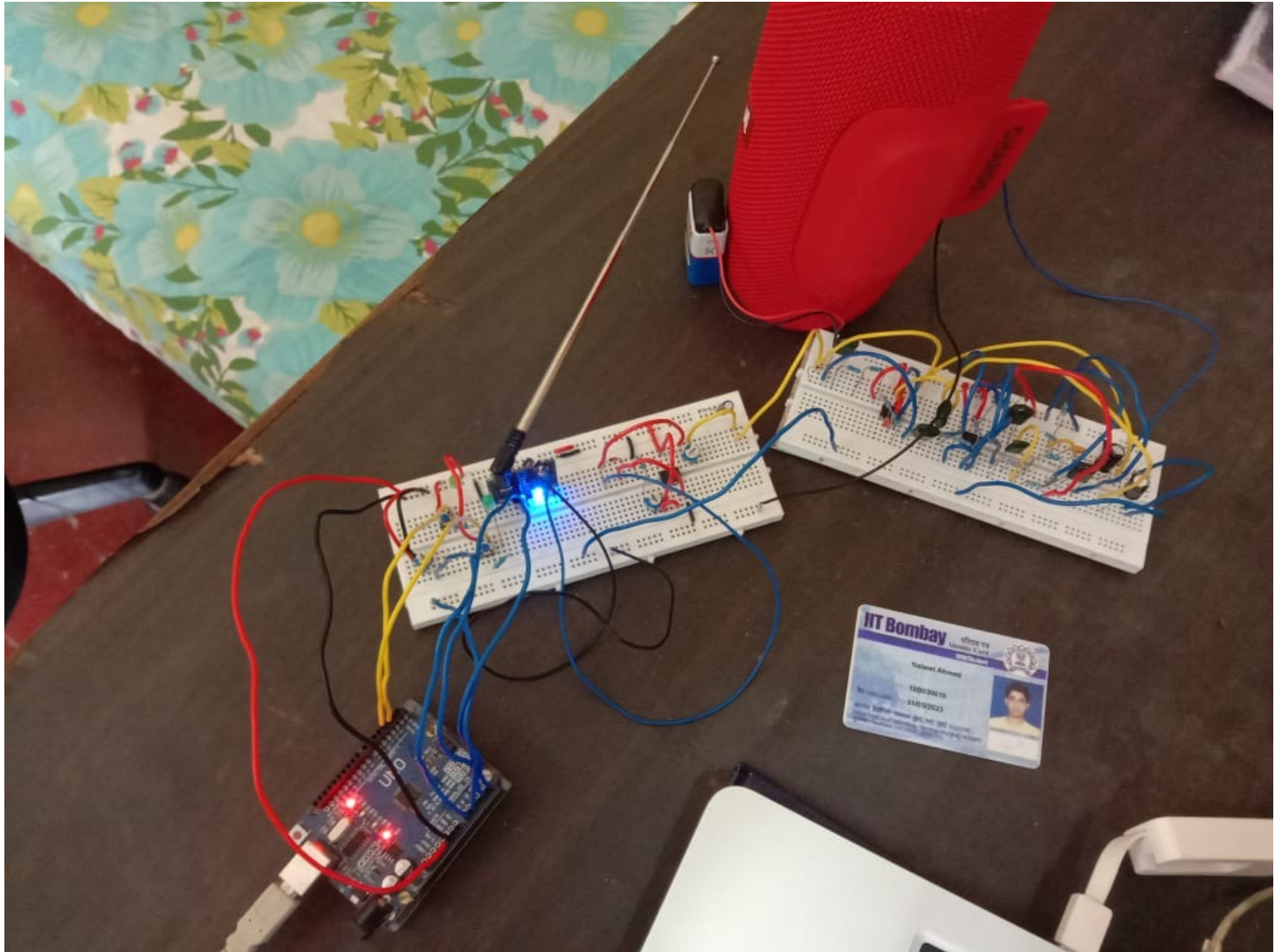
R1473

R1474

R1475

R1476

R147



Link to video demo:

<https://drive.google.com/drive/folders/1KtcuRmL9yDo7dIcxFs-BYRj1llgHIcuW?usp=sharing>

Bill of materials

- 1) TEA5767 Radio Module
- 2) AUX Cable
- 3) Speaker
- 4) Switches
- 5) LEDs
- 6) Potentiometers (10^4 k Ohm)
- 7) Resistors(1k, 2.2k, 10k,
- 8) Capacitors(100nF, 330nF, 1uF, 220uF)
- 9) Diodes
- 10) NPN Transistors (bc107b)
- 11) PNP Transistors (bc117b)
- 12) Connecting wires
- 13) 0-9V Battery
- 14) UA741CP Operational Amplifiers

Status and conclusion

We have achieved all the goals that were defined in the previous report. Our bonus goal was to build a seek for SoS circuit also but since that involved buying an FM radio transmitter and didn't involve any new insight to our project, we did not implement that part.

For our radio, there were many challenges involved in properly connecting up the components, especially in trying to figure out how to connect the aux to the breadboard properly, and in building and configuring our squelching circuit. We still have some noise in our radio that we haven't been able to remove completely. So, if the connections are loose, the radio makes a lot of noise.

Another issue that we faced was the reception of radio across different cities. Only a few channels play across the country 24x7.

The lesson we learnt is that calibration takes a lot of time and a one-size-fits-all machine is difficult to manufacture unless one has absolutely precise, perfectly manufactured, error-free resistors and capacitors with non-standard values. Each one of our team members has slightly different component values and tunings depending on our immediate reception capacities.

Source codes

```
#include <Arduino.h>
#include <Wire.h>

#ifndef TEA5767N_h
#define TEA5767N_h

//Note that not all of these constants are being used. They
//are here for
//convenience, though.
#define TEA5767_I2C_ADDRESS          0x60

#define FIRST_DATA                    0
#define SECOND_DATA                   1
#define THIRD_DATA                    2
#define FOURTH_DATA                   3
#define FIFTH_DATA                    4

#define LOW_STOP_LEVEL                1
#define MID_STOP_LEVEL                2
#define HIGH_STOP_LEVEL               3

#define HIGH_SIDE_INJECTION           1
#define LOW_SIDE_INJECTION            0

#define STEREO_ON                     0
#define STEREO_OFF                    1

#define MUTE_RIGHT_ON                 1
#define MUTE_RIGHT_OFF                0
#define MUTE_LEFT_ON                  1
#define MUTE_LEFT_OFF                 0

#define SWP1_HIGH                     1
#define SWP1_LOW                      0
#define SWP2_HIGH                     1
#define SWP2_LOW                      0

#define STBY_ON                       1
#define STBY_OFF                      0

#define JAPANESE_FM_BAND              1
#define US_EUROPE_FM_BAND             0

#define SOFT_MUTE_ON                  1
```



```
#define SOFT_MUTE_OFF 0

#define HIGH_CUT_CONTROL_ON 1
#define HIGH_CUT_CONTROL_OFF 0

#define STEREO_NOISE_CANCELLING_ON 1
#define STEREO_NOISE_CANCELLING_OFF 0

#define SEARCH_INDICATOR_ON 1
#define SEARCH_INDICATOR_OFF 0

class TEA5767N {
private:
    float frequency;
    byte hiInjection;
    byte frequencyH;
    byte frequencyL;
    byte transmission_data[5];
    byte reception_data[5];
    boolean muted;

    void setFrequency(float);
    void transmitFrequency(float);
    void transmitData();
    void initializeTransmissionData();
    void readStatus();
    float getFrequencyInMHz(unsigned int);
    void calculateOptimalHiLoInjection(float);
    void setHighSideLOInjection();
    void setLowSideLOInjection();
    void setSoundOn();
    void setSoundOff();
    void loadFrequency();
    byte isReady();
    byte isBandLimitReached();

public:
    TEA5767N();
    void selectFrequency(float);
    void selectFrequencyMuting(float);
    void mute();
    void turnTheSoundBackOn();
    void muteLeft();
    void turnTheLeftSoundBackOn();
    void muteRight();
    void turnTheRightSoundBackOn();
    float readFrequencyInMHz();
    void setSearchUp();
    void setSearchDown();
```

```
void setSearchLowStopLevel();
void setSearchMidStopLevel();
void setSearchHighStopLevel();
void setStereoReception();
void setMonoReception();
void setSoftMuteOn();
void setSoftMuteOff();

void setStandByOn();
void setStandByOff();
void setHighCutControlOn();
void setHighCutControlOff();
void setStereoNoiseCancellingOn();
void setStereoNoiseCancellingOff();

byte searchNext();
byte searchNextMuting();
byte startsSearchFrom(float frequency);
byte startsSearchFromBeginning();
byte startsSearchFromEnd();
byte startsSearchMutingFromBeginning();
byte startsSearchMutingFromEnd();
byte getSignalLevel();
byte isStereo();
byte isSearchUp();
byte isSearchDown();
boolean isMuted();
boolean isStandBy();

};

#endif

TEA5767N::TEA5767N() {
    Wire.begin();
    initializeTransmissionData();
    muted = false;
}

void TEA5767N::initializeTransmissionData() {
    transmission_data[FIRST_DATA] = 0;           //MUTE: 0 -
    not muted                                     //SEARCH MODE:
                                                0 - not in search mode

    transmission_data[SECOND_DATA] = 0;           //No frequency
    defined yet

    transmission_data[THIRD_DATA] = 0xB0;         //10110000
```

```
//SUD: 1 -
search up
//SSL[1:0]: 01
- low; level ADC output = 5
//HLSI: 1 -
high side LO injection
//MS: 0 -
stereo ON
//MR: 0 -
right audio channel is not muted
//ML: 0 - left
audio channel is not muted
//SWP1: 0 -
port 1 is LOW

    transmission_data[FOURTH_DATA] = 0x10; //00010000
//SWP2: 0 -
port 2 is LOW
//STBY: 0 -
not in Standby mode
//BL: 0 - US/
Europe FM band
//XTAL: 1 -
32.768 kHz
//SMUTE: 0 -
soft mute is OFF
//HCC: 0 -
high cut control is OFF
//SNC: 0 -
stereo noise cancelling is OFF
//SI: 0 - pin
SWPORT1 is software programmable port 1

    transmission_data[FIFTH_DATA] = 0x00; //PLLREF: 0 -
the 6.5 MHz reference frequency for the PLL is disabled
//DTC: 0 - the
de-emphasis time constant is 50 ms
}

void TEA5767N::calculateOptimalHiLoInjection(float freq) {
    byte signalHigh;
    byte signalLow;

    setHighSideLOInjection();
    transmitFrequency((float) (freq + 0.45));

    signalHigh = getSignalLevel();

    setLowSideLOInjection();
```

```
    transmitFrequency((float) (freq - 0.45));

    signalLow = getSignalLevel();

    hiInjection = (signalHigh < signalLow) ? 1 : 0;
}

void TEA5767N::setFrequency(float _frequency) {
    frequency = _frequency;
    unsigned int frequencyW;

    if (hiInjection) {
        setHighSideLOInjection();
        frequencyW = 4 * ((frequency * 1000000) + 225000) / 32768;
    } else {
        setLowSideLOInjection();
        frequencyW = 4 * ((frequency * 1000000) - 225000) / 32768;
    }

    transmission_data[FIRST_DATA] =
((transmission_data[FIRST_DATA] & 0xC0) | ((frequencyW >> 8) &
0x3F));
    transmission_data[SECOND_DATA] = frequencyW & 0xFF;
}

void TEA5767N::transmitData() {
    Wire.beginTransmission(TEA5767_I2C_ADDRESS);
    for (int i=0 ; i<5 ; i++) {
        Wire.write(transmission_data[i]);
    }
    Wire.endTransmission();
    delay(100);
}

void TEA5767N::mute() {
    muted = true;
    setSoundOff();
    transmitData();
}

void TEA5767N::setSoundOff() {
    transmission_data[FIRST_DATA] |= 0b10000000;
}

void TEA5767N::turnTheSoundBackOn() {
    muted = false;
    setSoundOn();
    transmitData();
}
```

```
void TEA5767N::setSoundOn() {
    transmission_data[FIRST_DATA] &= 0b01111111;
}

boolean TEA5767N::isMuted() {
    return muted;
}

void TEA5767N::transmitFrequency(float frequency) {
    setFrequency(frequency);
    transmitData();
}

void TEA5767N::selectFrequency(float frequency) {
    calculateOptimalHiLoInjection(frequency);
    transmitFrequency(frequency);
}

void TEA5767N::selectFrequencyMuting(float frequency) {
    mute();
    calculateOptimalHiLoInjection(frequency);
    transmitFrequency(frequency);
    turnTheSoundBackOn();
}

void TEA5767N::readStatus() {
    Wire.requestFrom (TEA5767_I2C_ADDRESS, 5);

    if (Wire.available ()) {
        for (int i = 0; i < 5; i++) {
            reception_data[i] = Wire.read();
        }
    }
    delay(100);
}

float TEA5767N::readFrequencyInMHz() {
    loadFrequency();

    unsigned int frequencyW = (((reception_data[FIRST_DATA] &
0x3F) * 256) + reception_data[SECOND_DATA]);
    return getFrequencyInMHz(frequencyW);
}

void TEA5767N::loadFrequency() {
    readStatus();
}
```

```
//Stores the read frequency that can be the result of a
search and it's not yet in transmission data
```

```
//and is necessary to subsequent calls to search.
```

```
        transmission_data[FIRST_DATA]      =
(ttransmission_data[FIRST_DATA] & 0xC0) |
(reception_data[FIRST_DATA] & 0x3F);
        transmission_data[SECOND_DATA]      =
reception_data[SECOND_DATA];
    }
```

```
float TEA5767N::getFrequencyInMHz(unsigned int frequencyW) {
    if (hiInjection) {
        return (((frequencyW / 4.0) * 32768.0) - 225000.0) /
1000000.0;
    } else {
        return (((frequencyW / 4.0) * 32768.0) + 225000.0) /
1000000.0;
    }
}
```

```
void TEA5767N::setSearchUp() {
    transmission_data[THIRD_DATA] |= 0b10000000;
}
```

```
void TEA5767N::setSearchDown() {
    transmission_data[THIRD_DATA] &= 0b01111111;
}
```

```
void TEA5767N::setSearchLowStopLevel() {
    transmission_data[THIRD_DATA] &= 0b10011111;
    transmission_data[THIRD_DATA] |= (LOW_STOP_LEVEL << 5);
}
```

```
void TEA5767N::setSearchMidStopLevel() {
    transmission_data[THIRD_DATA] &= 0b10011111;
    transmission_data[THIRD_DATA] |= (MID_STOP_LEVEL << 5);
}
```

```
void TEA5767N::setSearchHighStopLevel() {
    transmission_data[THIRD_DATA] &= 0b10011111;
    transmission_data[THIRD_DATA] |= (HIGH_STOP_LEVEL << 5);
}
```

```
void TEA5767N::setHighSideLOInjection() {
    transmission_data[THIRD_DATA] |= 0b00010000;
}
```

```
void TEA5767N::setLowSideLOInjection() {
    transmission_data[THIRD_DATA] &= 0b11101111;
```

```
}

byte TEA5767N::searchNextMuting() {
    byte bandLimitReached;

    mute();
    bandLimitReached = searchNext();
    turnTheSoundBackOn();

    return bandLimitReached;
}

byte TEA5767N::searchNext() {
    byte bandLimitReached;

    if (isSearchUp()) {
        selectFrequency(readFrequencyInMHz() + 0.1);
    } else {
        selectFrequency(readFrequencyInMHz() - 0.1);
    }

    //Turns the search on
    transmission_data[FIRST_DATA] |= 0b01000000;
    transmitData();

    while(!isReady()) { }
    //Read Band Limit flag
    bandLimitReached = isBandLimitReached();
    //Loads the new selected frequency
    loadFrequency();

    //Turns de search off
    transmission_data[FIRST_DATA] &= 0b10111111;
    transmitData();

    return bandLimitReached;
}

byte TEA5767N::startsSearchMutingFromBeginning() {
    byte bandLimitReached;

    mute();
    bandLimitReached = startsSearchFromBeginning();
    turnTheSoundBackOn();

    return bandLimitReached;
}

byte TEA5767N::startsSearchMutingFromEnd() {
```



```
    byte bandLimitReached;

    mute();
    bandLimitReached = startsSearchFromEnd();
    turnTheSoundBackOn();

    return bandLimitReached;
}

byte TEA5767N::startsSearchFromBeginning() {
    setSearchUp();
    return startsSearchFrom(87.0);
}

byte TEA5767N::startsSearchFromEnd() {
    setSearchDown();
    return startsSearchFrom(108.0);
}

byte TEA5767N::startsSearchFrom(float frequency) {
    selectFrequency(frequency);
    return searchNext();
}

byte TEA5767N::getSignalLevel() {
    //Necessary before read status
    transmitData();
    //Read updated status
    readStatus();
    return reception_data[FOURTH_DATA] >> 4;
}

byte TEA5767N::isStereo() {
    readStatus();
    return reception_data[THIRD_DATA] >> 7;
}

byte TEA5767N::isReady() {
    readStatus();
    return reception_data[FIRST_DATA] >> 7;
}

byte TEA5767N::isBandLimitReached() {
    readStatus();
    return (reception_data[FIRST_DATA] >> 6) & 1;
}

byte TEA5767N::isSearchUp() {
    return (transmission_data[THIRD_DATA] & 0b10000000) != 0;
```

```
}

byte TEA5767N::isSearchDown() {
    return (transmission_data[THIRD_DATA] & 0b10000000) == 0;
}

boolean TEA5767N::isStandBy() {
    readStatus();
    return (transmission_data[FOURTH_DATA] & 0b01000000) != 0;
}

void TEA5767N::setStereoReception() {
    transmission_data[THIRD_DATA] &= 0b11110111;
    transmitData();
}

void TEA5767N::setMonoReception() {
    transmission_data[THIRD_DATA] |= 0b00001000;
    transmitData();
}

void TEA5767N::setSoftMuteOn() {
    transmission_data[FOURTH_DATA] |= 0b00001000;
    transmitData();
}

void TEA5767N::setSoftMuteOff() {
    transmission_data[FOURTH_DATA] &= 0b11110111;
    transmitData();
}

void TEA5767N::muteRight() {
    transmission_data[THIRD_DATA] |= 0b00000100;
    transmitData();
}

void TEA5767N::turnTheRightSoundBackOn() {
    transmission_data[THIRD_DATA] &= 0b11111011;
    transmitData();
}

void TEA5767N::muteLeft() {
    transmission_data[THIRD_DATA] |= 0b00000010;
    transmitData();
}

void TEA5767N::turnTheLeftSoundBackOn() {
    transmission_data[THIRD_DATA] &= 0b11111101;
    transmitData();
}
```

```
}

void TEA5767N::setStandByOn() {
    transmission_data[FOURTH_DATA] |= 0b01000000;
    transmitData();
}

void TEA5767N::setStandByOff() {
    transmission_data[FOURTH_DATA] &= 0b10111111;
    transmitData();
}

void TEA5767N::setHighCutControlOn() {
    transmission_data[FOURTH_DATA] |= 0b00000100;
    transmitData();
}

void TEA5767N::setHighCutControlOff() {
    transmission_data[FOURTH_DATA] &= 0b11111011;
    transmitData();
}

void TEA5767N::setStereoNoiseCancellingOn() {
    transmission_data[FOURTH_DATA] |= 0b00000010;
    transmitData();
}

void TEA5767N::setStereoNoiseCancellingOff() {
    transmission_data[FOURTH_DATA] &= 0b11111101;
    transmitData();
}

TEA5767N radio = TEA5767N();

int analogPin = 0;
int val = 0;
float frequencyInt = 0;
float previousFrequency = 0;
int signalStrength = 0;
int squelchPin = 1;
float seekedfrequency = 0;
int mode = 0;

void setup()
{
    radio.setMonoReception();
    radio.setStereoNoiseCancellingOn();
    pinMode(analogPin, INPUT);
    pinMode(squelchPin, INPUT);
}
```

```
    pinMode(2, INPUT);
    pinMode(3, INPUT);
    attachInterrupt(digitalPinToInterrupt(2), seekchannel,
HIGH);
    attachInterrupt(digitalPinToInterrupt(3), sswitch, HIGH);
    Serial.begin(9600);
}

void loop() {

if(mode)
{
    for(int i = 0;i<100;i++)
    {
        val = val + analogRead(analogPin);
        delay(1);
    }

    val = val/100;

    float frequency = 93.0 + ( ((99.0-93.0)/1024) * val );

    Serial.println(frequency);

    if(frequency - previousFrequency >= 0.1f ||
previousFrequency - frequency >= 0.1f)
    {
        radio.selectFrequency(frequency);
        previousFrequency = frequency;
    }

    delay(1000);
    val = 0;
}
else
{
    Serial.println(seekedfrequency);
    radio.selectfrequency(seekedfrequency);
    delay(1000);
}

}

void seekchannel()
{
    Serial.println("Seek");
    float seeker = seekedfrequency;
```

```
int found = 0;

while(!found)
{
    if(analogRead(squelchPin) > 824) //VALUE OF 824 HAS BEEN
OBTAINED BY MANUAL CALIBRATION
    {
        seekedfrequency = seeker;
        found = 1;
    }
    else
    {
        if(seeker >= 99.0)
            seeker = 93.0;
        else
        {
            seeker = seeker + 0.01;
        }
    }
}

void sswitch()
{
    Serial.println("Switch");
    mode = !mode;
}
```