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

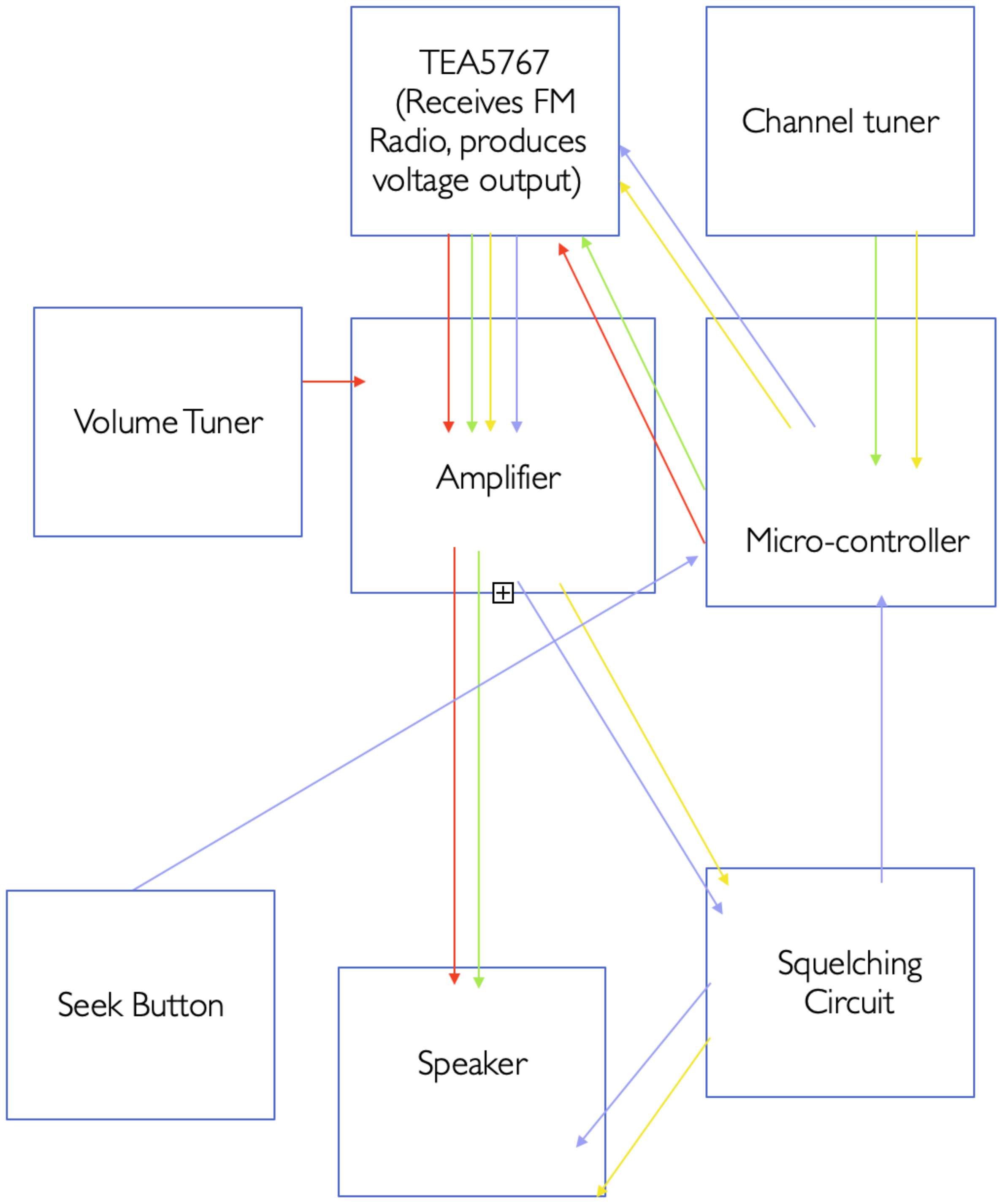
B.Malavika (190260014)
Moysha Gera (190260031)
Nabeel Ahmed (19B030016)

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.

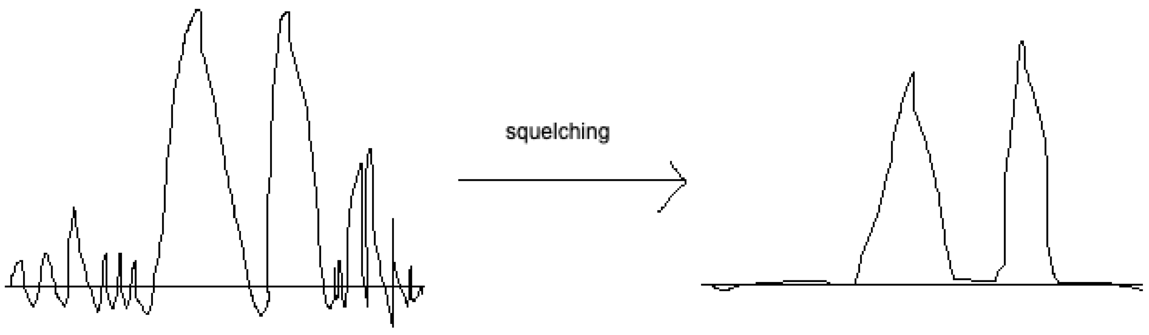
This signal goes through a volume tuner, which is basically an Op-amp based amplifier circuit and we can vary the gain by changing the potentiometer values. There are a few additional functionalities that we have added that is described in the following paragraph:

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.

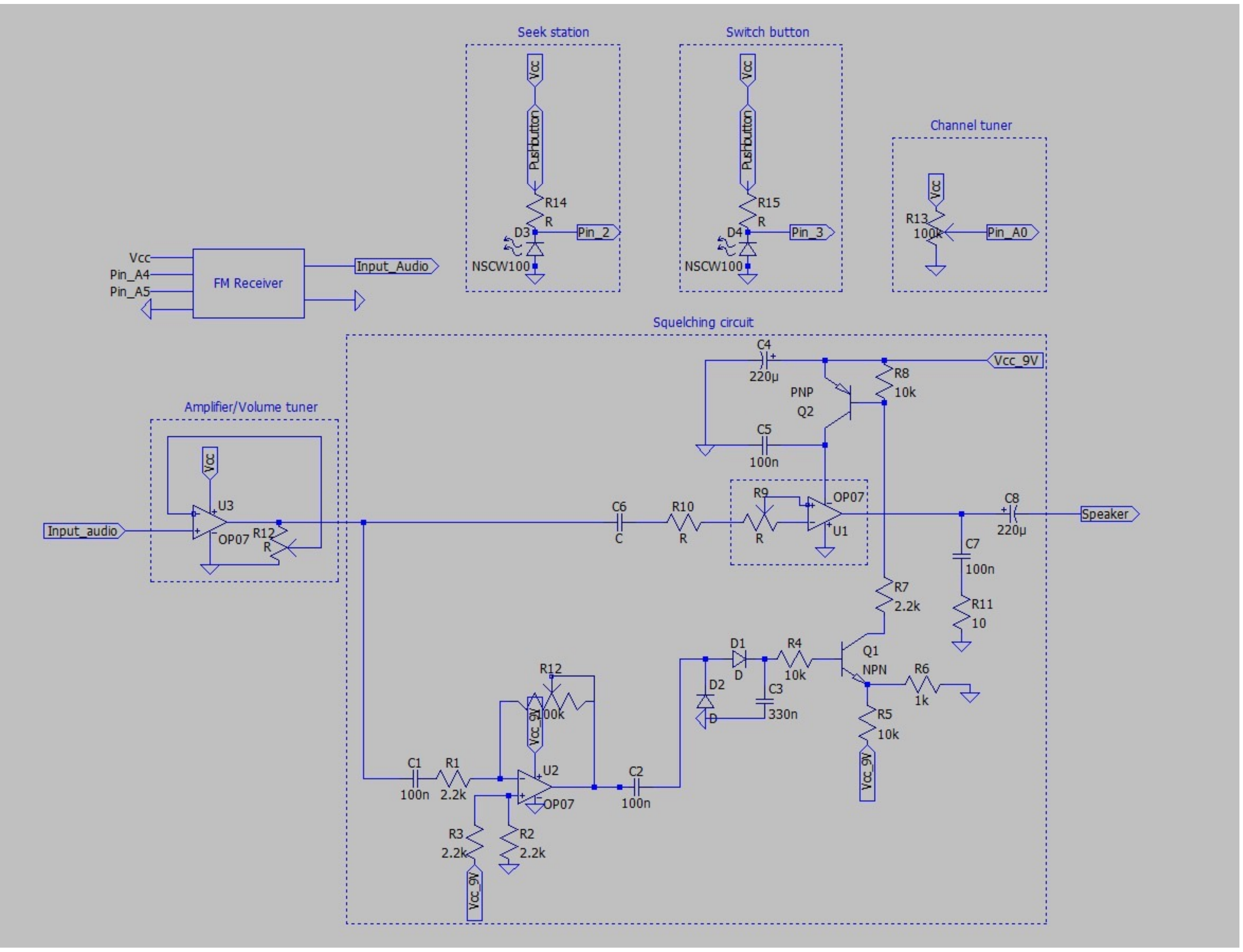
Now the audio signal that is outputted by the TEA5767 FM Receiver module is put through a squelching circuit. This circuit attenuates the background noise that is present between the channel frequencies when tuning the radio. The circuit consists of an LM386 low-voltage audio amplifier and an op-amp. 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.

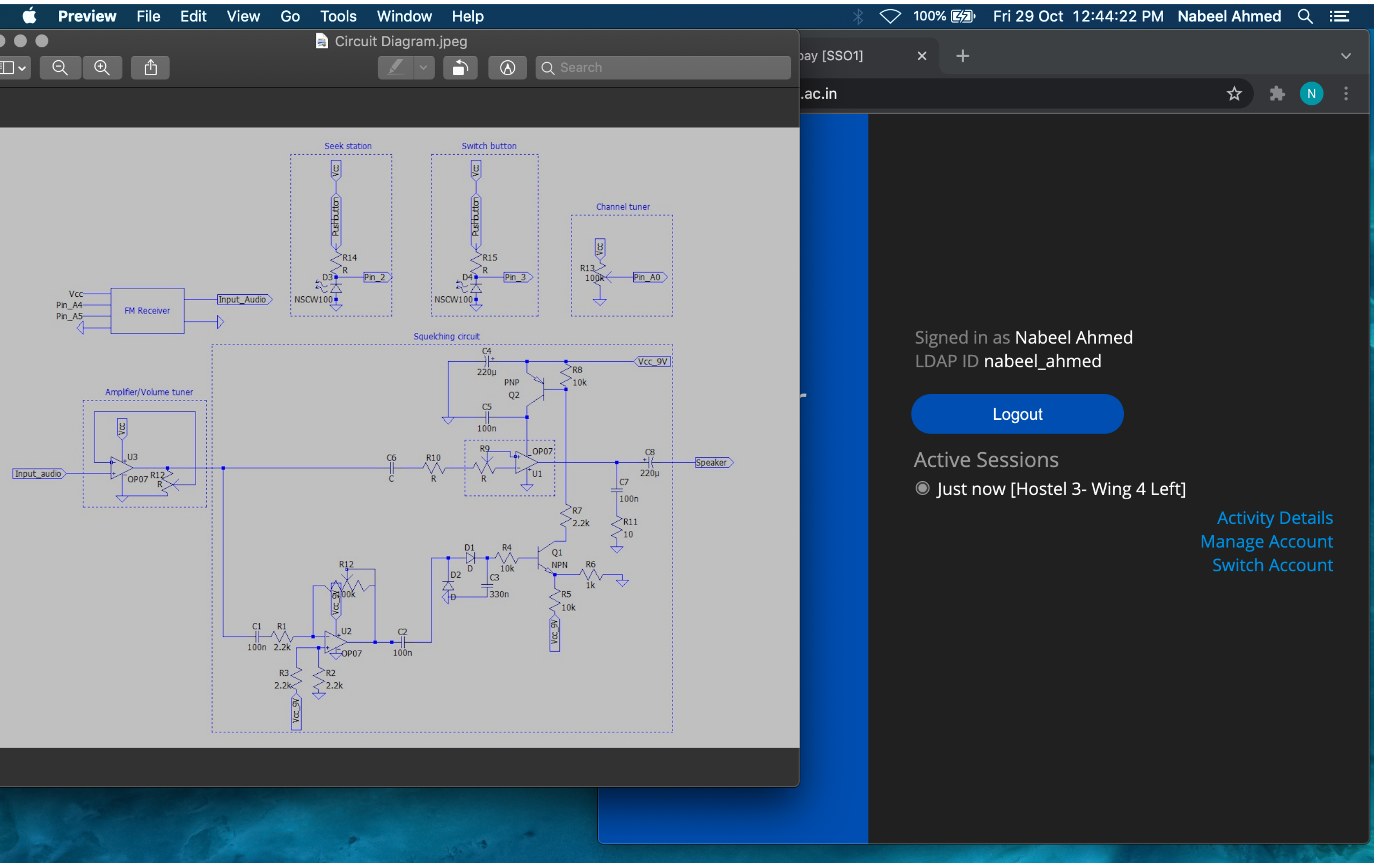
This circuit 'squelches' or 'mutes' the noise, and selectively amplifies the channels with a considerably strong audio signal. This audio signal is identified on the basis of frequency as well as amplitude. A simple factor for differentiating between noise and a channel is that channels have larger signal 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





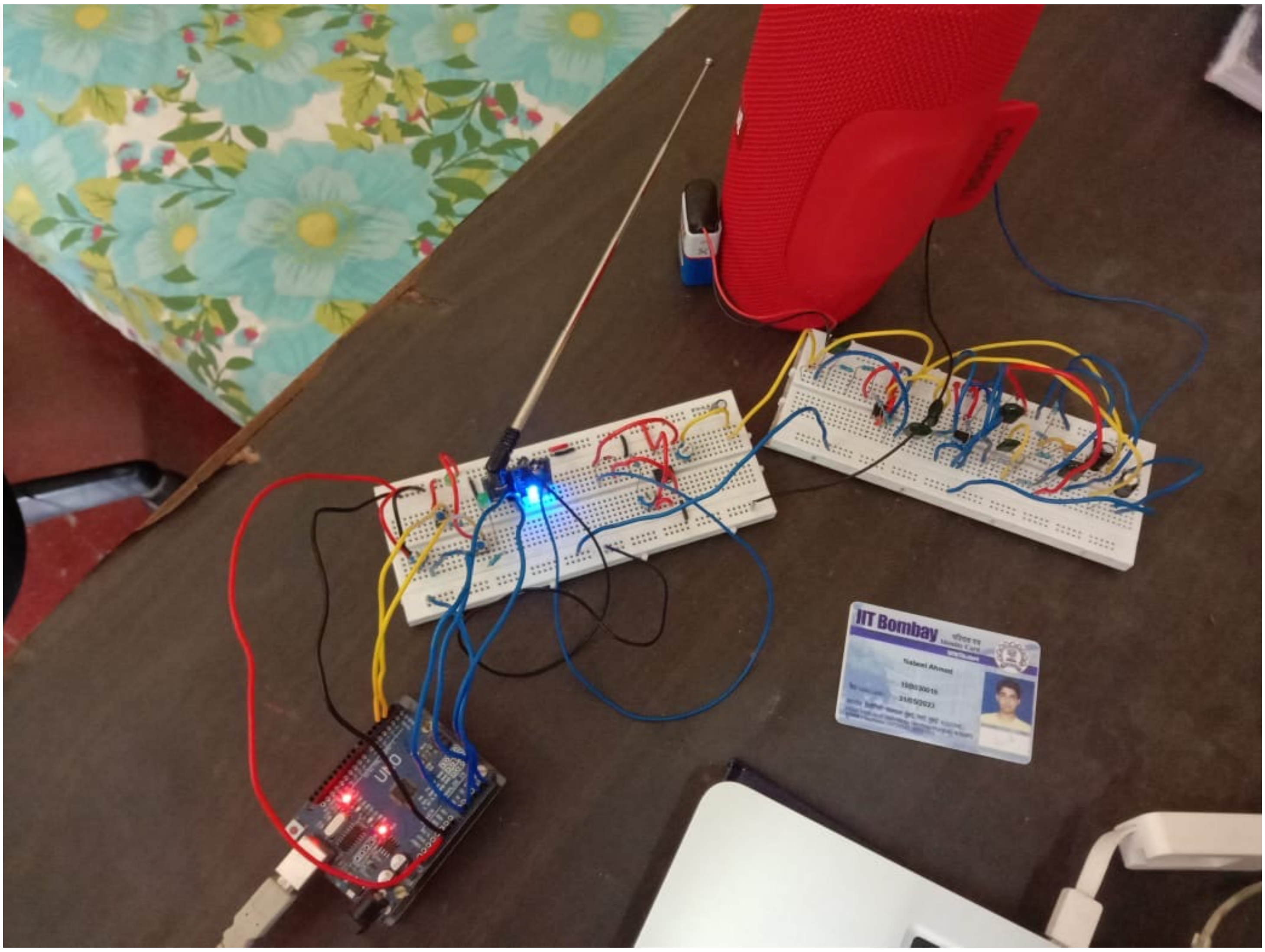
Signed in as Nabeel Ahmed
LDAP ID nabeel_ahmed

Logout

Active Sessions

Just now [Hostel 3- Wing 4 Left]

Activity Details
Manage Account
Switch Account



Link to video demo:

https://drive.google.com/drive/folders/1KtcuRmL9yDo7dIcxFs-BYRj1IlgHIcuW?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 thr 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]     =
(transmission_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;
}
```