CE-0114: Audio Modem for Packet Networks

NUSH

September 26, 2023

Abstract

Low-bandwidth data communication refers to independent communications systems that do not rely on major telecom providers or infrastructures such as radio towers and are often characterised by their low connection speed. In recent years, more and more places are covered by such infrastructures or telecom providers taht enable more people to have access to faster connections, with new technology such as mobile phones and 5G network becoming more popular, reducing the need for low-bandwidth data communication.

However, such concentration and a shift to rely on major telecom providers or infrastructures are only feasible when the infrastructures are working normally. Furthermore, many of the mountainous areas in the world are not covered by such technology. Hence, this makes such modes of communication vulnerable to infrastructure failures, be it through malice, natural disasters or worn out of machine, especially for less developed areas. Independent, easy-to-use backup options such as low-bandwidth data communication are important to ensure communication at critical times such as natural disaster. Such backup options may include the use of analogue radio systems like walkie talkies that are inexpensive and usable even at times of infrastructure failures.

In this paper, we propose a method of implementing a modulator-demodulator to transmit and receive data packets in the form of audio waves using Arduino as microcontroller. We also implmented a digital signal processing algorithms using the Arduino IDE to filter our the noise of the data and demodulate it. Lastly, we used oscilloscope to test our hard and software to show that it works properly.

Keywords: Data communication, Bandpass filters, Audio transmission, Natural disasters, microcontroller

Contents

1 Introduction

Low-bandwidth **data communication** is of immense importance with many applications in remote sensing(Haskovic et al., 2013), and military(Lee and Steele, 2014).

Today, there is widely available infrastructure (e.g. mobile phone and LoRaWAN networks) offered by major telecom providers, with China alone having more than 680 million people with internet access(West, 2015). Such concentration and outsourcing are economical when the infrastructure is working normally, but this dependence causes significant vulnerability towards infrastructure failures, be it through malice, acts of nature, or marginal infrastructure in less developed locations(Unicef et al., 2020).

Commodity analogue radio systems (e.g. walkie talkies) have a large installed base, are inexpensive, and can be pressed into service for data transmission at minimal cost. Applications of such includes during natural disasters where the normal data coomunication infrastructures are damaged or destroyed(Lien et al., 2009, 2010).

Such radios are designed for telephone-quality analogue audio transmissions with a frequency range of 300 to 3000 Hz. Data messages (datagrams) need to encoded ("modulated") as an audio waveform matching this requirement for transmission, and decoded ("demodulated") upon reception. This is the same principle as used for internet access via analogue telephone lines up to the early 2000s.

In this research, our team used Arduino as a **microcontroller** for processing, sending, and receiving the data to and from walkie talkies. Arduino is = a hardware and sofrware project which started in 2005 with the goal of simplifying electric-electronic devices with a microcontroller(Arduino, 2015; ard). It has many applications ranging from **audio transmission** (Duraisamy, 2021; Kuhite and Madankar, 2017; Bianchi and Queiroz, 2013) and other form of data transmissions such as visible light(Sandoval-Reyes and Hernandez-Balderas, 2017) and bluetooth communication(Lodhi et al., 2016; Asadullah and Ullah, 2017; Dai et al., 2016; Maity et al., 2017). We have selected Arduino as a suitable choice for microcontroller in such projects due to its audio to digital converter(ADC) and its ability to have its own software.

Our project's main goal is to develop a modulator-demodulator to transmit and receive data packets (datagrams) as audio waveforms. We will read the input from a computer's keyboard, which we will then process the string to binary and write it directly to Arduino using register and encoded using frequency shift keying(FSK). The Arduino will then send a corresponding voltage, which we will filter out the desired range using a **bandpass filter** to restrict the frequency to be within the transmission range of 300 to 3000 Hz. That signal would then be send to a walkie talkie for transmission. We used another walkie talkie connected to a second Arduino for receiving the signal. The received signal needs to be passed through an AC coupling to remove the DC component of the signal. Our team then offsets the signal with 2.5 volts such that the signal do not have negative voltage. Lastly, our team reads the input voltage and uses idea of Fourier series to detect the frequency of the signal to decode it back to the binary string after filtering out the noise.

2 Sending of the Signal

2.1 Reading of Input

Our team decided to read the input from the serial monitor of Arduino IDE. We have selected the pin A0 on the Arduino board as input. Due to the speed required for this project, we have decided to directly read from the register using the following code

```
flag=PIND&0x01;
```

Where flag is the bit that is about to be send.

The above code will set the bit that is to be send to be equal to the input from pin A0, which is the first pin of PIND, hence it is read by using

```
PIND&0x01;
```

2.2 Sending of Data

The sending of data is also done by writing directly into registers. In this project, our team have selected to use A1 as the output pin, which is the second pin of PORTC. Hence, putting this together with reading of input 2.1, the code will be as follow

```
ISR(TIMER2_COMPA_vect){
    // flag?PORTC^=2:(cnt&1?PORTC^=2:NULL);
    flag=PIND&0x01;
    // Serial.println(flag);
    if(flag){
        PORTC^=2;
    }else{
        if(cnt&1){
        PORTC^=2;
        // Serial.println(cnt);
        }
    }
    cnt^=1;
}
```

2.3 Use of Timer

Since we are using FST, our team would need to send high and low frequencies at certain intervals. We have chosen to send them at 2000 Hz and 1000 Hz respectively. This is done by using the timer2 object in Arduino, which is a 8-bit timer that is more accurate than using delay since it is using system timing. We choose to use interruptions(which is more regular) and the processing and sending of data would be in the interruption(ISR) function.

The code for setting up of that is below.

```
void setup() {
  // Serial.begin(115200);
  pinMode(A0,INPUT);
  pinMode(A1,OUTPUT);
  pinMode(8,OUTPUT);
  digitalWrite(8,1);
  cli();
  TCCR2A = (1 << WGM21);
  TCCR2B = (1 << CS21)|(1 << CS20);
  TIMSK2 = (1 << OCIE2A);
  OCR2A = 124; //define upper limit of counter before it resets.
  sei();
}
unsigned short cnt=0;
bool flag; // if it is high flag is true, else low frequency flag=false;
ISR(TIMER2_COMPA_vect){
    // flag?PORTC^=2:(cnt&1?PORTC^=2:NULL);
    flag=PIND&0x01;
    // Serial.println(flag);
    if(flag){
        PORTC^=2;
    }else{
        if(cnt&1){
        PORTC^=2;
        // Serial.println(cnt);
        }
    }
    cnt^=1;
}
```

Notice that 2000 Hz is twice the frequency of 1000 Hz. Therefore, to generate 1000 Hz, we can use a same timer as 2000 Hz, with the only difference being that we send the bit once every other time. This is done using the cnt variable and using the & operator to test its parity.

The calculation is as follow:

$$\frac{16 * 10^6}{64} = 250000 \tag{1}$$

$$\frac{250000}{124 + 1} = 2000 \tag{2}$$

Which is the desired frequency (the +1 is due to programmers counting from 0 instead of 1)

2.4 Bandpass Filter

After sending the bits into Arduino, it will generate a 5 volts max frequency/signal. However, that is a square wave containing a wide range of frequencies, many of them which lies outside of the desired range of 300 to 3000 Hz used by walkie talkies. Hence, we neea a bandpass filter which compose of a low-pass filter and a high-pass filter, connected in that order.

Low-pass Filter

A low-pass filter is a filter which allows currents with frequency less than a certain value to pass through. In this project, our team has adopted the RC(resistor-capacitor) low-pass filter, which composes of a resistor and capacitor, as shown in the diagram below(Omegatron, a).

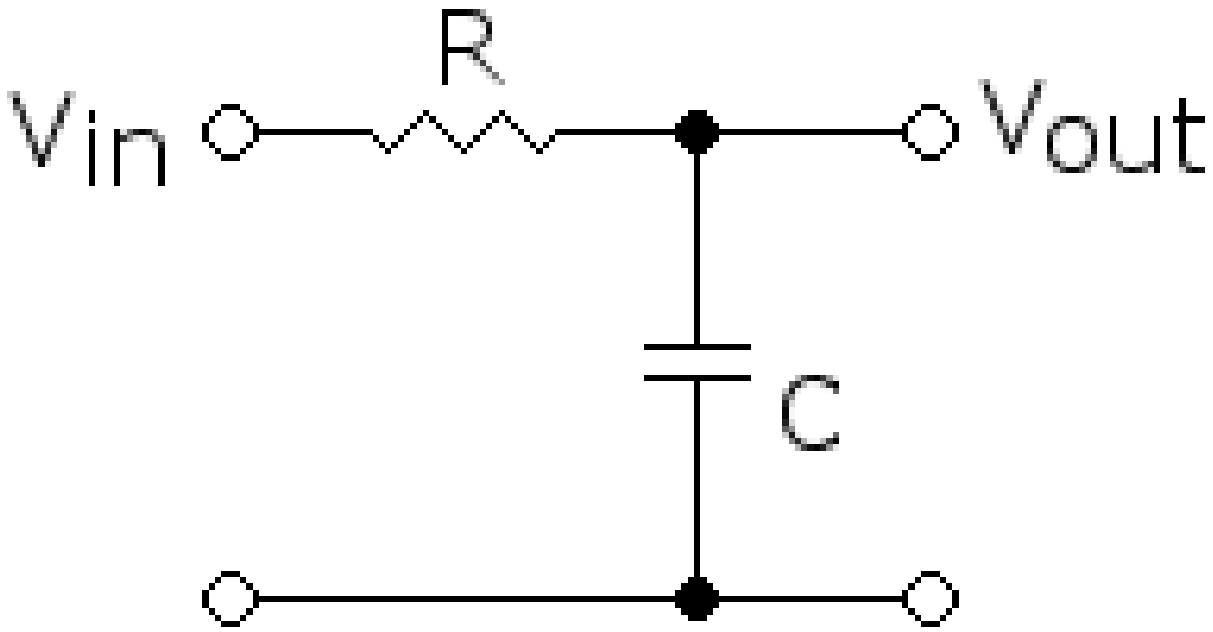

Figure 1: An example of low-pass filter

To calculate the required capacitance and resistance, we would need to find the cutoff frequency of the filter, which can be derived as following(Storr, 2022): For two resistor in series, by the voltage divider rule

$$V_{out} = V_{in}\frac{R_2}{R_1 + R_2} \tag{3}$$

We also know that the circuit has an impedence of

$$Z = \sqrt{R^2 + X_c^2} \tag{4}$$

Where $X_c = \frac{1}{2\pi f}$, with f denoting the frequency of the AC current.

Hence, substituting the above into equation 3, we get

$$V_{out} = V_{in} * \frac{X_c}{Z} \tag{5}$$

This allows us to find the resultant voltage for any frequency.

Notice how the above decreases as frequemcy increases. Hence, very high frequencies will get greatly attenuated

as there is very little voltage of them at the output. Hence, a common cutoff frequency used is

$$f_c = \frac{1}{2\pi RC} \tag{6}$$

Where R is the resistance, C is the capacitance.

Since we want the low-pass filter to filter out the AC currents with frequency above 3000 Hz, we have selected a 680 Ohm resistor and a 100 nana Farads capacitor, with a cutoff frequency of $2340 \leq 3000$ Hz, which filters out the undesired high frequency currents.

High-pass filter

A high-pass filter has a similar cutoff frequency as the low-pass filter of $f_c = \frac{1}{2\pi RC}$. However, the position of the resistor and capacitor are swapped, causing it to allow AC currents with frequency above cutoff frequency f_c to pass.

Below is an image of high-pass filter(Omegatron, b)

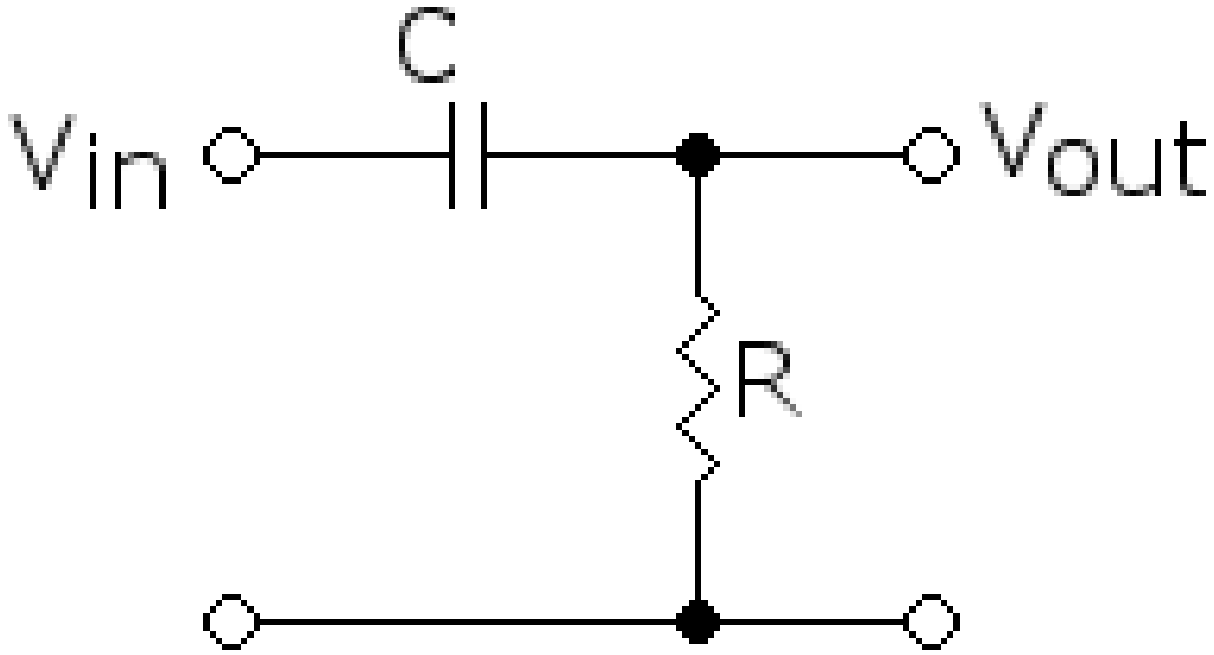

Figure 2: An example of high-pass filter

We have selected a resistor of 10k Ohms and capacitor of 47 nano Farads, which allows AC current with frequency above $339 \leq 300$ Hz to pass, which filters out the undesired low frequency currents.

Putting them Together

We have connected the above two components in series, with the low-pass filter placed before the high-pass filter. Below is an image of our circuit

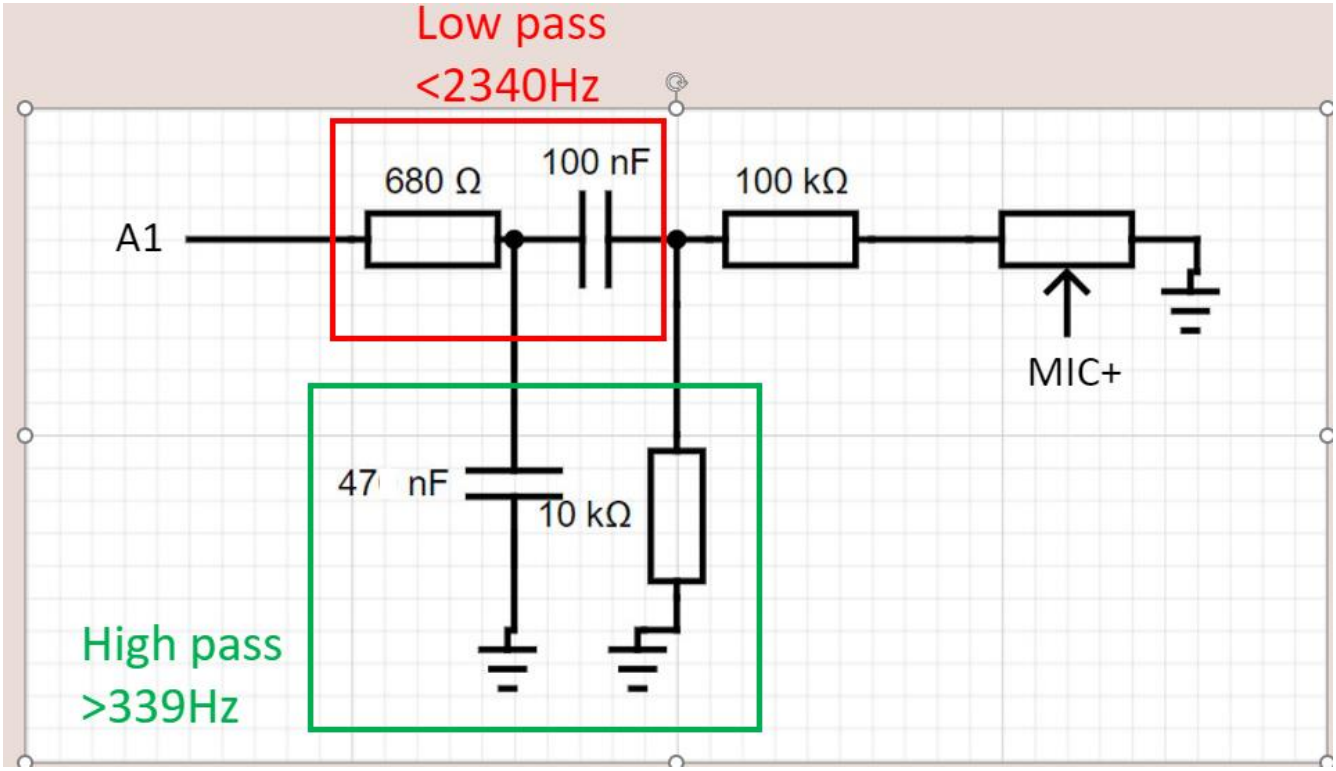

Figure 3: Our bandpass filter

3 Transmitter

A transmitter is a device used to magnify and/or send the audio/radio signal. In our project, we have chosen to use a transistor to send the input signal as shown in the circuit below.

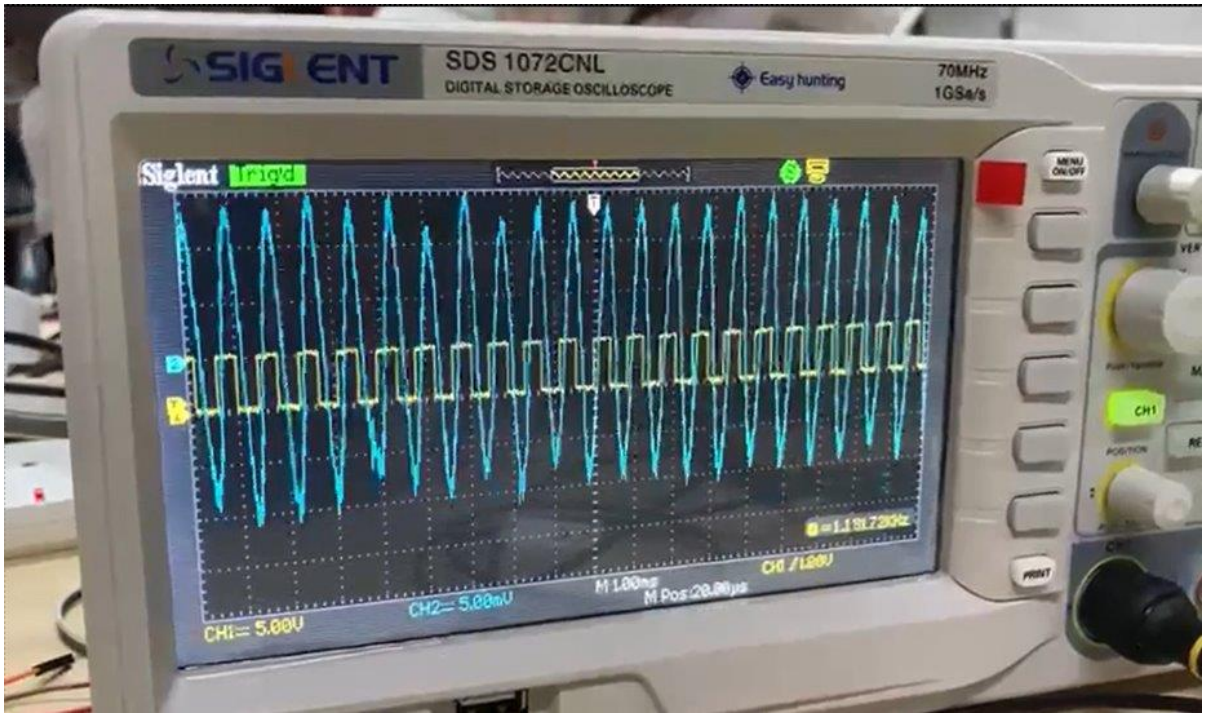

Figure 5: The yellow curve is the current from Arduino, the blue curve is after all the steps

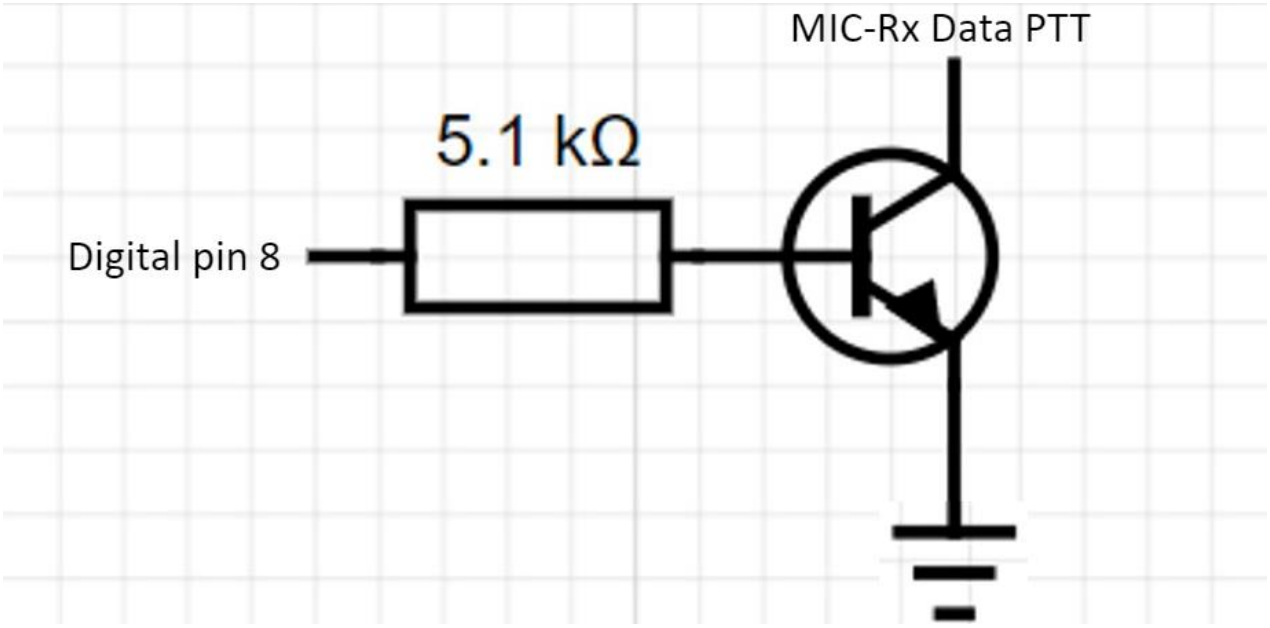

Figure 4: Our transmitter circuit

To test our sending part of the circuit, we have connected the above to an oscilloscope which would allow us to see the result of the bandpass filter and the sending. As we can see from above, our bandpass filter is effective as it filtered out too high and too low frequencies of the Arduino current(which is a square wave). We can tell that by noticing that there is less sharp turns as compared to the square wave that comes out directly from the Arduino(yellow curve).

4 Receiving of the Signal

The current and voltage directly received from the walkie talkie is not usable in the Arduino and would require further processing for the following reaons

- There may be a DC component due to background noise which is unwanted
- The AC current from walkie talkie may be less than 0 volts(due to it being AC), which the Arduino board is unable to take care of

Hence, to solve the above two problems, our team used the following solutions

- Use AC coupling to filter out the DC components
- Use an DC offset to make all the voltages positive

The AC coupling is done by simply putting an capacitor which would only allow AC currents to pass through.

The DC offset is done by connecting a constant 5 volts supply from the Arduino to the filtered current, as shown in the diagram below

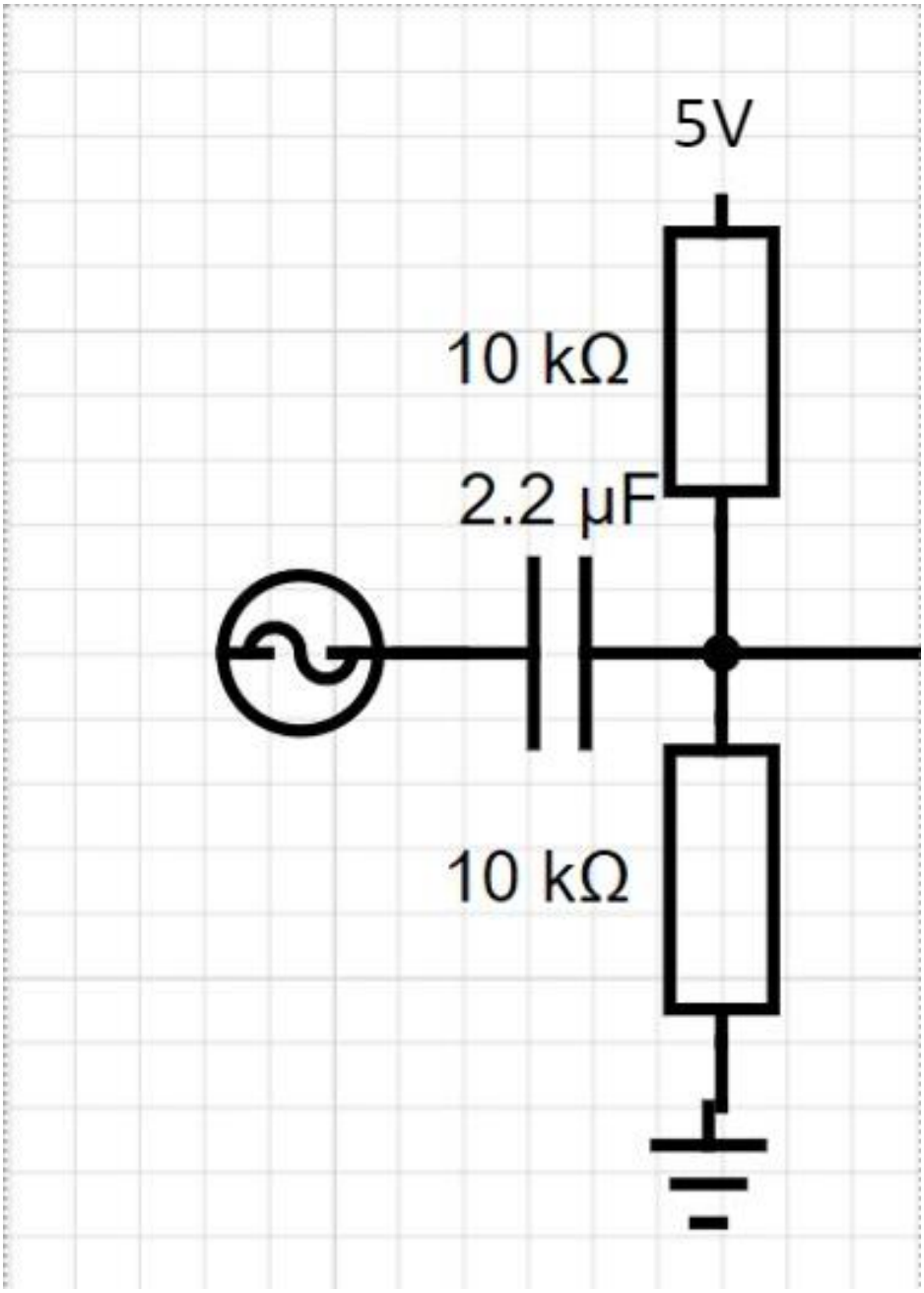

Figure 6: Diagram showing our circuit for the receiving end

In the above circuit, the left side with the AC symbol is the AC current from the walkie talkie(possibly containing DC components), and the right side is where the processed current will flow to.

The above circuit has an offset of $\frac{5}{2} = 2.5V$. Since the top is 5 volts and the ground is 0 volts, and thw two resistor have the same resistance. By the voltage divider rule, the potential at the middle junction(and hence the offset) would be

$$V = \frac{10000}{10000 + 10000} * 5 = 2.5 \tag{7}$$

5 Combining Sending and Receiving

Before we proceed on the last step of decoding the data, we would need to make sure that our received signal is at least reasonably clean for us to be able to effectively remove noise and accurately decode the data. In the following picture, the yellow curve is the data output from the Arduino, and the blue curve is the data received from the walkie talkie after the offset circuit.

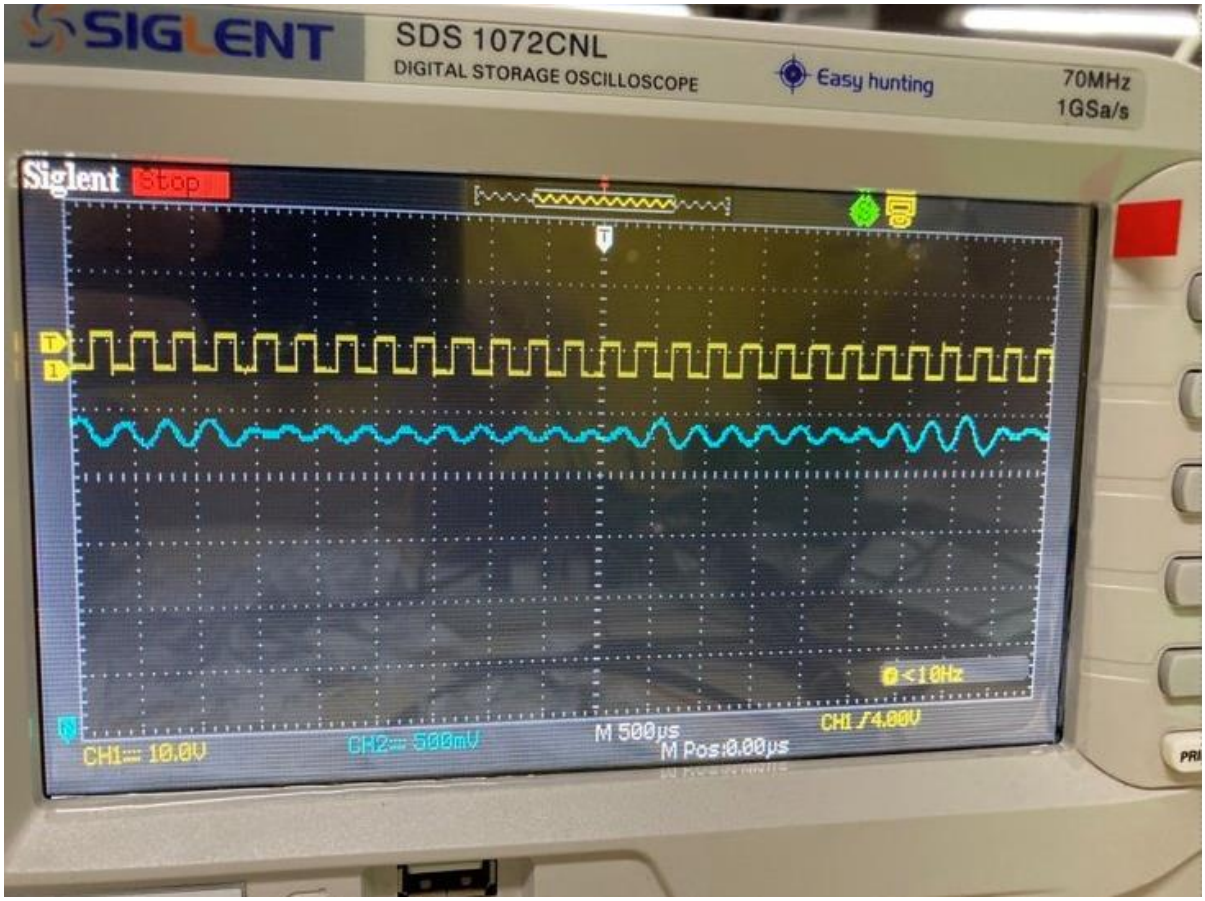

Figure 7: Showing our results of sending and receiving

As we can see from the above, the received signal is reasonably without noise and smooth, which allow us to proceed to the final steps of finding the frequency and reducing noise

6 Finding the Frequency

At the very beginning of this paper, our team has mentioned that we will be using frequency shift keying to encoding the data. That means a higher frequency of 2000 Hz will correspond to 1 bit, while a lower frequency of 1000 Hz would correspond to a 0 bit.

The good thing of frequency shift keying is the fact that frequency would not change significantly via the course of transmission.

However, in order to find the frequency of the data, we first need to sample the data at certain intervals. The reason we need to sample is due to the fact that the input data is continuous, but our processing takes time, making the data we are able to read discrete. Hence, to obtain a reasonable representation of the continous data, we need to sample above a certain rate, which is given by the Nyquist frequency to be two times the frequency we are sending, which would be $2 * 2000 = 4000$ Hz. However, our team decided to use 8000 Hz to increase the accuracy of our sampling.

We achieved 8000 Hz sampling frequency by using timer2 in a similar way as Sending of the Data 2.3. The only difference would be changing the value of OCR2A from 124 to 62, as shown in the code below

```
void setup() {
    // Serial.begin(115200);
    pinMode(A0,INPUT);
    pinMode(A2,OUTPUT);
    pinMode(A3,OUTPUT);
    TCCR2A = (1 << WGM21);
    TCCR2B = (1 << CS21)|(1 << CS20);
    TIMSK2 = (1 << OCIE2A);
    OCR2A = 61; //define upper limit of counter before it resets.
    ADMUX=0x40;
    ADCSRA=B11000111;
}
```

The main difficulty(and the key component) for this project is on finding the frequency from the sampled data.

Our team originally tried to do this by finding the time difference T between adjacent maximum, and the frequency would be given as $f = \frac{1}{T}$. However, we soon found that to be infeasible for the following reasons

- Finding the maximum would require storing all the values, which may cause Memory Limit Exceeded(MLE) on the Arduino
- Finding adjacent maximums would at best require only another loop through the stored number to find the maximum, which would half the processing speed

- There may not even be a complete cycle for us to find the frequency
- The wave may be an addition of many waves of different frequencies(due to noise)

To overcome that problem, our team used the idea of Fourier series. This comes from the fact that each of these waves can be taken to be addition of sine waves with different amplitude and frequency, i.e.,

$$f = A_1 \sin(\omega_1 x) + A_2 \sin(\omega_2 x) + ... + A_n \sin(\omega_n x) \tag{8}$$

Where n might tends to infinity.

Hence, to find the frequency, we just need to find the maximum component in the above. This is done by noting the following facts.

Theorem 6.1. *For positive integer* $n \neq m$

$$\int_0^{2\pi} sin(nx)sin(mx)dx = 0 \tag{9}$$

Proof. By product to sum formula

$$\int_0^{2\pi} \sin(nx)\sin(mx)dx = \tag{10}$$

$$\int_0^{2\pi} \frac{1}{2}(\cos((n-m)x) - \cos((n+m)x))dx = \tag{11}$$

$$\frac{1}{2}(\int_0^{2\pi} \cos((n-m)x)dx - \int_0^{2\pi} \cos((n+m)x)dx) = \tag{12}$$

$$\frac{1}{2}((\frac{1}{n-m}(\cos(2\pi(n-m)) - \cos 0)) + (\frac{1}{n+m}(\cos(2\pi(n+m)) - \cos 0))) = 0 \tag{13}$$

□

Theorem 6.2. *But if* $n = m$

$$\int_0^{2\pi} sin(nx)^2 dx = \pi \tag{14}$$

Proof.

$$\int_0^{2\pi} \sin(nx)^2 dx = \int_0^{2\pi} (\frac{1-\cos(2nx)}{2})dx = \tag{15}$$

$$(\int_0^{2\pi} \frac{1}{2}) - \frac{1}{2}\int_0^{2\pi} \cos(2nx)dx = \tag{16}$$

$$\frac{1}{2}(2\pi - 0) - \frac{1}{2}(\cos(4n\pi) - \cos(0)) = \pi \tag{17}$$

□

Hence, given a vector of input data $[a_1, a_2, a_3, ..., a_n]$, we simply needs to dot product it with $[\sin(\frac{\pi}{1000}), \sin(\frac{2\pi}{1000}), \sin(\frac{3\pi}{1000}), ..., \sin(\frac{n\pi}{1000})]$, which would get us

$$a_1 \sin(\frac{\pi}{1000}) + a_2 \sin(\frac{2\pi}{1000}) + ... + a_n \sin(\frac{n\pi}{1000}) \tag{18}$$

Since our $n = 8000$ which is a reasonably high sampling frequency

$$a_1 \sin(\frac{\pi}{1000}) + a_2 \sin(\frac{2\pi}{1000}) + ... + a_n \sin(\frac{n\pi}{1000}) \approx \tag{19}$$

$$\int_0^{2\pi} a_x \sin(mx)dx = \int_0^{2\pi} A_1 \sin(\omega_1 x) \sin(mx)dx + \int_0^{2\pi} A_2 \sin(\omega_2 x) \sin(mx)dx + ...+ \tag{20}$$

$$\int_0^{2\pi} A_n \sin\omega_n x \sin(mx)dx = 0 + 0 + ... + \pi + 0 + ...0 = \pi \tag{21}$$

Notice how the above equals to 0 except for the component with same frequency, which is the 1000 Hz component. Hence, this would allow us to extract the component of 1000 Hz.

Hence, given the input data $[a_1, a_2, a_3, ..., a_n]$, we can dot product it with sine and cosine waves at 1000 and 2000 Hz, and the larger component would be the desired frequency.

7 Reducing Noise

Due to possible errors in recognizing, we have used the majority vote method, which keeps track of the past 8 data points. Together with this newly recognized data point, they form a total of 9 data points which are either 1000 Hz or 2000 Hz.

In the majority voting method, the current data point would be the mode of the 9 data points, which means that if there are more than or equal to 5 2000 Hz points, then the current point would be recognized as 2000 hz. Otherwise, it will be recognized as 1000 Hz. The implementation for that is given below

```
ISR(TIMER2_COMPA_vect){
  PORTC^=0x08;
  r++;
  r&=(7);
  buff[r]=ADC;
  ADCSRA|=0x40;
  //in order is sin 1000,sin 2000,cos 1000,cos 2000
  vals[0]=buff[0]+buff[1]-buff[4]-buff[5];
  vals[2]=buff[2]+buff[3]-buff[6]-buff[7];
  vals[1]=buff[0]-buff[2]+buff[4]-buff[6];
```

```
  vals[3]=buff[1]-buff[3]+buff[5]-buff[7];
  long freq_1000=vals[0]*vals[0]+vals[2]*vals[2];//is this correct?
  long freq_2000=vals[1]*vals[1]+vals[3]*vals[3];
  short cnt=0;
  bool this_bit;
  if(freq_1000<=freq_2000){
    cnt>=5?this_bit=1:this_bit=0;
    // else this bit is 0
    // since this bit is originally a 0 bit, cnt doesn't change
    // this_bit?PORTC|=0x04:PORTC&=~0x04;
    if(!this_bit){
      PORTC|=0x04;
    }else{
      PORTC&=~0x04;
    }
    cnt+=(this_bit);// then update the count of number of 1s(if this bit is 1 then cnt++)
    cnt-=(((history&(1<<7)))>>7);// remove the highest bit
    //the left and right shifts extract the highest bit and decrement cnt accordingly
    history=(history<<1);
  }else{
    cnt++;
    cnt>=5?this_bit=1:this_bit=0;
    // this_bit?PORTC&=~0x04:PORTC|=0x04;
    if(!this_bit){
      PORTC&=~0x04;
    }else{
      PORTC|=0x04;
    }
    cnt-=(!this_bit);
    cnt-=(((history&(1<<7)))>>7);
    history=(history<<1)|1;
  }
}
```

8 Conclusion

In this project, our team used Arduino as microcontroller to send and receive data to and from walkie talkies. We used frequency shift keying of 1000 and 2000 Hz to encode the data when sending. A bandpass filter is used to select the desired range of frequency to send to the walkie talkie. AC coupling is also utilised to remove the DC component in receiving the signal from the walkie talkie. A majority vote filter is then used to filter out the noise and interpret the data by finding the frequency of the signal received.

Future work could include using more efficient data encoding such as Huffman encoding to further compress the data send. A better noise filtering/parity check bits could be used to reduce the chance of error in transmission of signal and remove the effect of background noise.

9 Appendix

Code for Sending

```
#pragma comment(linker,"/stack:200000000")
#pragma GCC optimize("Ofast")
#pragma GCC optimize("inline")
#pragma GCC optimize("unroll-loops")
#define int long
#define string String
#define MAXN 205
#define SEND_MAX 10010
#define dl 100
#define stdout A1
#define high 199
#define low 500
#include <avr/io.h>
#include <avr/interrupt.h>
const int prescale = 8;
const int ocr2aval = (16000000/(16000000/(3000*255)))/3000-1;
// const int ocr2aval = 254;
const float iinterval = prescale * (ocr2aval+1) / (F_CPU / 1.0e6);
const float period = 2.0 * iinterval;
const float freq   = 1.0e6 / period;
void setup() {
  // Serial.begin(115200);
  pinMode(A0,INPUT);
  pinMode(A1,OUTPUT);
  pinMode(8,OUTPUT);
  digitalWrite(8,1);
  cli();
  TCCR2A = (1 << WGM21);
  TCCR2B = (1 << CS21)|(1 << CS20);
  TIMSK2 = (1 << OCIE2A);
  OCR2A = 82; //define upper limit of counter before it resets.
  sei();
}
short arr[MAXN][10];
```

```
int id=0;
inline string to_string(int x){
  string str="";
  while(x){
    str+=(x%10);
    x/=10;
  }
  return str;
}
inline int to_int(char x){
  return x-'0';
}
inline void to_bin(int x){
  id++;
  int bin=0;
  int r=8;
  int sum=0;
  while(x){
    bin=(bin<<3)+(bin<<1)+x%2;
    arr[id][r]=x%2;
    r--;
    sum+=(x%2);
    x>>=1;
  }
  arr[id][8]=sum%2;
}
inline int which(int x){
  if(x==1){
    return high;
  }
  return low;
}
inline void send_num(){
  for(int i=7;i>=0;i--){
    analogWrite(stdout,which(id&(1<<i)));//not using
  }
  // delay(dl);
}
```

```
unsigned short cnt=0;
bool flag; // if it is high flag is true, else low frequency flag=false;
inline void sendh(){
  cli();
  TCCR2A = (1 << WGM21);
  TCCR2B = (1 << CS21)|(1 << CS20);
  TIMSK2 = (1 << OCIE2A);
  OCR2A = 124; //define upper limit of counter before it resets.
  sei();
}
inline void send_chars(){
  send_num();
  send_num();
  send_num();
  register int aa;
  for(int i=1;i<=id;i++){
    for(int j=1;j<=8;j++){
      flag=arr[i][j];
      sendh();
    }
  }
}
void loop() {
  // register char ch;
  // while(Serial.available()){
  //  delay(3);
  //  if(Serial.available()>0){
  //    // ch=Serial.read();
  //    // to_bin(ch);
  //  }
  //  send_chars();
  //  send_sums();
  // }
  // tone(A1,3000,100);
  // flag=true;
  sendh();
}
volatile unsigned char value = 0;
```

```
ISR(TIMER2_COMPA_vect){
  // flag?PORTC^=2:(cnt&1?PORTC^=2:NULL);
  flag=PIND&0x01;
  // Serial.println(flag);
  if(flag){
    PORTC^=2;
  }else{
    if(cnt&1){
      PORTC^=2;
      // Serial.println(cnt);
    }
  }
  cnt^=1;
}
```

Code for Receiving

```
// #pragma comment(linker,"/stack:200000000")
// #pragma GCC optimize("Ofast")
// #pragma GCC optimize("inline")
// #pragma GCC optimize("unroll-loops")
#define string String
#define MAXN 205
#define dl 50
#define stdout A2
#define stdin A0
#define high 700
#define low 100
short r=0;
int buff[10];
long vals[4];
uint8_t history=0;
//in order is sin 1000,sin 2000,cos 1000,cos 2000
void setup() {
  // Serial.begin(115200);
  pinMode(A0,INPUT);
  pinMode(A2,OUTPUT);
```

```
  pinMode(A3,OUTPUT);
  // Serial.println("hi3");
  // cli();
  TCCR2A = (1 << WGM21);
  TCCR2B = (1 << CS21)|(1 << CS20);
  TIMSK2 = (1 << OCIE2A);
  OCR2A = 61; //define upper limit of counter before it resets.
  ADMUX=0x40;
  ADCSRA=B11000111;
  // ADCSRA |= B00000100; // set ADIE bit in ADCSRA register
  // ADCSRA |= B01000000; // set ADSC bit in ADCSRA register
  // Serial.println("HI");
  // sei();
}
// int arr[MAXN][10];
// int buff[MAXN<<5];
// int num;
void loop() {
  // Serial.println("ahh");
  // cli();
  // Serial.println("hi2");
  // TCCR2A = (1 << WGM21);
  // TCCR2B = (1 << CS21)|(1 << CS20);
  // TIMSK2 = (1 << OCIE2A);
  // OCR2A = 20; //define upper limit of counter before it resets.
  // sei();
  // digitalWrite(8,1);
  // PORTB^=0x01;
}
ISR(TIMER2_COMPA_vect){
  PORTC^=0x08;
  // Serial.println("hi");// for testing
  // PORTC^=0x04;
  r++;
  // Serial.println("h2");
  r&=(7);
  // Serial.println("h3");
  buff[r]=ADC;
```

```
ADCSRA|=0x40;
// if(buff[r]>512){//this line works have signal in blue
//  PORTC|=0x04;
// }else{
//  PORTC&=~0x04;
// }
// ADCSRA|=0x40;
// Serial.println("h4");
// for(int i=0;i<=7;i++){
//  // Serial.println("h5");
//  Serial.println(buff[i]);
// }
// vals[0]=buff[(2)&7]+(buff[(3)&7]<<1)+buff[(4)&7]-buff[(6)&7]-(buff[(7)]<<1)-buff[0];
// vals[1]=(buff[(2)&7]<<1)-(buff[(4)&7]<<1)+(buff[(6)]<<1)-(buff[0]<<1);
// vals[2]=(buff[(1)&7]<<1)+(buff[(2)&7])-buff[(4)]-(buff[(5)]<<1)-(buff[(6)])+buff[0];
// vals[3]=-(buff[(2)&7]<<1)+(buff[(4)&7]<<1)-(buff[(6)]<<1)+(buff[0]<<1);
//in order is sin 1000,sin 2000,cos 1000,cos 2000
vals[0]=buff[0]+buff[1]-buff[4]-buff[5];
vals[2]=buff[2]+buff[3]-buff[6]-buff[7];
vals[1]=buff[0]-buff[2]+buff[4]-buff[6];
vals[3]=buff[1]-buff[3]+buff[5]-buff[7];
long freq_1000=vals[0]*vals[0]+vals[2]*vals[2];//is this correct?
long freq_2000=vals[1]*vals[1]+vals[3]*vals[3];
short cnt=0;
bool this_bit;
if(freq_1000<=freq_2000){
  cnt>=5?this_bit=1:this_bit=0;// if number of 1s(including the current bit) is more than 5
      then this bit is 1
  //else this bit is 0
  //since this bit is originally a 0 bit, cnt doesn't change
  // this_bit?PORTC|=0x04:PORTC&=~0x04;
  if(!this_bit){
    PORTC|=0x04;
  }else{
    PORTC&=~0x04;
  }
  cnt+=(this_bit);//then update the count of number of 1s(if this bit is 1 then cnt++)
```

```
  cnt-=(((history&(1<<7)))>>7);//remove the highest bit(since it is out of the window of
      past 8 values)
  //the left and right shifts extract the highest bit and decrement cnt accordingly
  history=(history<<1);//ok now it is unfiltered bit(since this bit is 0)
  // PORTC|=0x04;
  // not every sure why this doesn't work
  // it should enter either if or else
  //then it is 1000 Hz
  // Serial.println("1000 Hz"); //for testing
  // prev[1]=prev[2];
  // prev[2]=prev[3];
  // prev[3]=0;
  // if(!prev[1]||!prev[2]){
  //   // PORTC^=0x04;
  //   PORTC&=0x04;
  //   // sends low signal
  // }else{
  //   PORTC&=~0x04;
  //   prev[3]=1;
  // }
}else{
  cnt++;
  cnt>=5?this_bit=1:this_bit=0;
  // this_bit?PORTC&=~0x04:PORTC|=0x04;
  if(!this_bit){
    PORTC&=~0x04;
  }else{
    PORTC|=0x04;
  }
  cnt-=(!this_bit);
  cnt-=(((history&(1<<7)))>>7);
  history=(history<<1)|1;//this is unfiltered bit(current bit is 1)
  // PORTC&=~0x04;
  // prev[1]=prev[2];
  // prev[2]=prev[3];
  // prev[3]=1;
  // if(prev[1]||prev[2]){
  //   PORTC&=~0x04;
```

```
    // }else{
    //  PORTC&0x04;
    //  prev[3]=0;
    // }
    //it is 2000Hz
    // Serial.println("2000 Hz");//for testing
  }
}
```

References

Emir Y Haskovic, Sterling Walsh III, Glenn Cloud, Rick Winkelman, Yingqing Jia, Sergey Vishnyakov, and Feng Jin. Application of inexpensive, low-cost, low-bandwidth silhouette profiling ugs systems to current remote sensing operations. In *Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR IV*, volume 8742, pages 189–194. SPIE, 2013.

Ricky J Lee and Sarah L Steele. Military use of satellite communications, remote sensing, and global positioning systems in the war on terror. *J. Air L. & Com.*, 79:69, 2014.

Darrell M West. Digital divide: Improving internet access in the developing world through affordable services and diverse content. *Center for Technology Innovation at Brookings*, pages 1–30, 2015.

Unicef et al. How many children and young people have internet access at home?: estimating digital connectivity during the covid-19 pandemic. Technical report, UNICEF, 2020.

Yao-Nan Lien, Li-Cheng Chi, and Yuh-Sheng Shaw. A walkie-talkie-like emergency communication system for catastrophic natural disasters. In *2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, pages 309–314. IEEE, 2009.

Yao-Nan Lien, Li-Cheng Chi, and Chih-Chieh Huang. A multi-hop walkie-talkie-like emergency communication system for catastrophic natural disasters. In *2010 39th International Conference on Parallel Processing Workshops*, pages 527–532. IEEE, 2010.

Store Arduino Arduino. Arduino. *Arduino LLC*, 372, 2015.

URL `https://www.arduino.cc/`.

Balaganesh Duraisamy. Multipoint data transmission using li-fi with lgs formation. In *Intelligent Computing and Innovation on Data Science*, pages 559–567. Springer, 2021.

Deepshika L Kuhite and Mangala S Madankar. Wireless audio transmission system for real-time applications—a review. In *2017 International Conference on Inventive Systems and Control (ICISC)*, pages 1–5. IEEE, 2017.

André Jucovsky Bianchi and Marcelo Queiroz. Real time digital audio processing using arduino. In *Proceedings of the Sound and Music Computing Conference, Stockholm, Sweden*, pages 538–545, 2013.

Sergio Sandoval-Reyes and Arturo Hernandez-Balderas. Transmission of digital audio with visible light. *Research in Computing Science*, 138:61–68, 2017.

Deepak Kumar Lodhi, Prakshi Vats, Addala Varun, Prashant Solanki, Ritakshi Gupta, Manoj Kumar Pandey, and Rajat Butola. Smart electronic wheelchair using arduino and bluetooth module. *International Journal of Computer Science and Mobile Computing*, 5(5):433–438, 2016.

Muhammad Asadullah and Khalil Ullah. Smart home automation system using bluetooth technology. In *2017 International Conference on Innovations in Electrical Engineering and Computational Technologies (ICIEECT)*, pages 1–6. IEEE, 2017.

Bin Dai, Rung-Ching Chen, and Wei-Bin Yang. Using arduino to develop a bluetooth electronic scale for water intake. In *2016 International Symposium on Computer, Consumer and Control (IS3C)*, pages 751–754. IEEE, 2016.

Ayan Maity, Avijit Paul, Priyanka Goswami, and Ankan Bhattacharya. Android application based bluetooth controlled robotic car. *Int. J. Intell. Inf. Syst*, 6(5):62, 2017.

Omegatron. Low-pass filter, a. URL https://zh.wikipedia.org/zh-cn/%E4%BD%8E%E9%80%9A%E6%BB%A4%E6%B3%A2%E5%99%A8#/media/File:Low_pass_filter.png.

Wayne Storr. Low pass filter - passive rc filter tutorial, Aug 2022. URL https://www.electronics-tutorials.ws/filter/filter_2.html.

Omegatron. High-pass filter, b. URL https://zh.wikipedia.org/zh-cn/%E9%AB%98%E9%80%9A%E6%BB%A4%E6%B3%A2%E5%99%A8#/media/File:High_pass_filter.png.