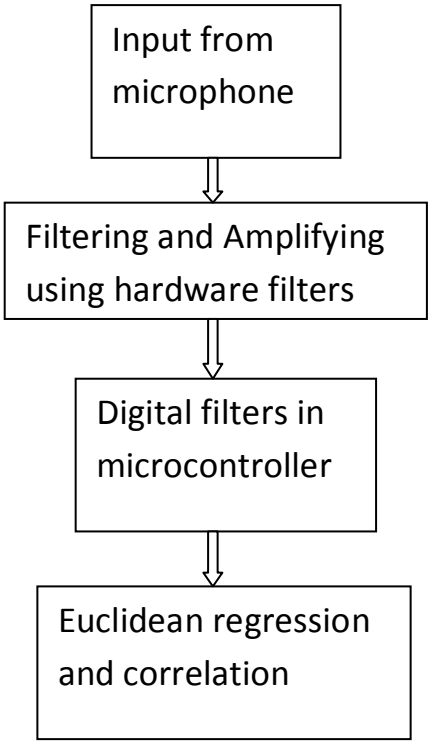
VOICE RECOGNITION SECURITY SYSTEM

Raunak Sakhardande, 08026001

Arun Ginjala, 08026012

We planned to make a voice recognition security system using an Atmega168 16-MHz microcontroller on an Arduino platform. The aim of the project was to unlock upon a specific voice password. We intended to compute the entire speech processing on an atmega168 microcontroller using real time processing. This involves simultaneously accepting the input from the user and using hardware and software filters to analyze the data. The comparison was then to be established by using Euclidean distance and correlation.



BLOCK DIAGRAM OF THE PROJECT

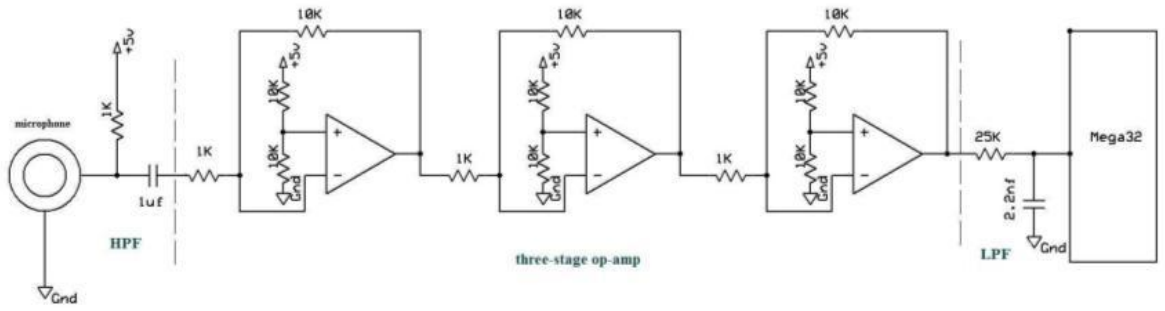
SUMMARY

The structure involves a microphone circuit followed by amplifiers, after which the samples are fed into the ADC and then passed through digital band pass filters. We sampled with respect to the Nyquist rate theorem i.e. 4 KHz. This was done as the human spectrum lies below 2 KHz. We obtained filter coefficients for digital filters using MATLAB 6.5. We reduced the samples from 4000 to 80 by using stepwise accumulation and finally compared using Euclidean regression.

DESIGN

1. Microphone circuit

Since the output from the microphone is in mV we use three stages of operational amplifiers. Two hardware filters-one low pass filter (around 150 Hz) and a high pass filter (around 2500 Hz) is also implemented.



2. Digital filters

The coefficients of the digital band pass filters have been calculated using MATLAB. First, the human speech spectrum has been analyzed and appropriate filters have been proposed. 5 band pass filters in the frequency range 50Hz – 1500Hz are used. The coefficients for these filters have been calculated in MATLAB using the frequency range of the filters as parameters. We used chebychev filters in MATLAB. The following frequency ranges of band pass filters are used.

Filter 1: 50-350 Hz

Filter 2: 350-500 Hz

Filter 3: 500-750 Hz

Filter 4: 750-1000 Hz

Filter 5: 1000-1500 Hz

We used fourth-order band pass filters which were done by cascading two second order filters.

The fingerprints of the speech are obtained by accumulation of the squares of outputs of each filter. This was done for 250 samples at a time.

We were planning to achieve real time processing by filtering and accumulating both while the speech was being accepted by the device. However, to achieve this, the digital filters are to be coded in assembly language. We found it very difficult to implement existing codes on the Arduino platform and due to our own inadequacy we could not code filters in assembly language.

Due to lack of time we coded in C language for the entire code however, this came at a price. Although we had achieved efficient processing due to the memory required it was not possible to attain real time processing. We hence decided to store the co-efficients in microcontroller and engage in filtering it subsequently. However as the memory of microprocessor is 8Kb we could store at most one word at a time.

3. Fingerprint Analysis

The fingerprint of each filter is obtained by the accumulation of the outputs of the samples passed through the band pass filters. However, the fingerprints are different for different words as they have different speech spectrum. Also, when the same person speaks the same word different times, its fingerprints can be slightly different. Hence, we need

to calculate the difference between the fingerprints of different words and compare them to test whether the microcontroller can recognize it. The analysis of the voice is done using correlation and regression techniques to find the voiceprints of the words.

The first step is to compute the Euclidean distance which is the accumulation of the square of the difference of a word.

$$Euclidean\ distance = \sum_{i} (dic[i] - word(i))^2$$

Sometimes, the fingerprints of the same word may have same shape but different amplitude. Then, the Euclidean distance might judge these as two different words and hence correlation is used to tell the similarities of the same word's fingerprint.

Functions written in the code:

1. Sampling Function

For collecting the samples we used timer0 adjusting the prescalars suitably and setting a frequency of 4000Hz. We kept a check on the OCF0A bit of the TIFR0 register.

Setting up the desired frequency was achieved with the help of the following code

```
ADMUX = B00100000;
  ADCSRA = B11000101; //prescalar is 32
   TIMSK0 = B00000010;  //turn on timer 0 cmp match ISR
  OCR0A = 63;
  TCCR0A = B00000000;
  TCCR0B = B00001011;
```

2. Filter functions

Each sample from the ADC is passed through a filter function. There are 5 such filters. Each filter is of the form

```
float filter( float x )
{
        static float d01 = 0.0;
        static float d11 = 0.0;
        static float d02 = 0.0;
        static float d12 = 0.0;
        float y1, y2, t0, t1;

        /* first 2nd-order filter stage */
        t0 = x - a2_1*d01 - a3_1*d11;
        y1 = b1_1*t0 + b2_1*d01 + b3_1*d11;
        d11 = d01;
        d01 = t0;
        /* second 2nd-order filter stage */
        t1 = y1 - a2_2*d02 - a3_2*d12;
        y2 = b1_2*t1 + b2_2*d02 + b3_2*d12;
        d12 = d02;
        d02 = t1;
        return g*y2;
}
```

The filter code is based on the Tor's speech algorithm.

The coefficients of these filters (b1_1, b2_1, b3_1, a2_1, a3_1, b1_2, b2_2, b3_2, a2_2, a3_2 in the above case) are obtained using chebychev filter functions in MATLAB.

3. Analyze function

The analyze function takes the samples of the word as input and gives the fingerprints of the word as output. 250 samples are accumulated as a fingerprint by passing through each of the 5 filters, thereby giving 16 fingerprints per filter (if the total number of samples were 4000)

$y = y + (\text{filter}(i))^2$

4. Euclidean function

The Euclidean function calculates the accumulation of the square of the difference between two words.

5. Comparator function

Used to compare the fingerprints obtained from test word with the stored fingerprints of words.

Work Division

Since most of the work was involved in software part of the design everything was done jointly.

Conclusion

We have written codes for different parts and tested them. Though there were no compilation errors, the code cannot do real-time processing with the given 16 MHz processor. We are trying to write the filters in assembly code so that real-time processing can be done.