



**Sultan Qaboos University**

College of Engineering

Department of Electrical and Computer Engineering

Mini-Project Report

Morse Code Interpreter Based on Acoustic Input Device

**Done by:**

Sulaiman Salim Al-Habsi

85258/08

Hamza Ali Al-Abri

85152/08

Ali Mohammed Redha Al-Lawati

89686/09

## Introduction

Morse code is a method of transmitting text information as a series of on-off tones, lights, or clicks that can be directly understood by a skilled listener or observer without special equipment. The International Morse Code encodes the ISO basic Latin alphabet, some extra Latin letters, the Arabic numerals and a small set of punctuation and procedural signals as standardized sequences of short and long signals called "dots" and "dashes" respectively, or "dits" and "dahs". Because many non-English natural languages use more than the 26 Roman letters, extensions to the Morse alphabet exist for those languages.

This project try to translate the Morse code automatically using an embedded systems.

## Aim

To design an embedded system device which can decode the Morse code and display it on a screen.

## Procedure

### 1- Hardware

First, a microphone (that receives the sound signal) was connected between VCC voltage and the ground. Since the received sound signal had a very small magnitude, an operational amplifier was connected to the microphone by a coupling capacitor (C1) in order to amplify the sound signal to an appropriate magnitude. Then, the output of the op-amp was connected to the ADC (Analog to Digital Converter) pin of the microcontroller by a coupling capacitor (C2). In addition to the capacitor, a pull-down resistor was connected to the ADC pin in order to prevent any noise signal. Furthermore, a pulse switch is connected to a pin of PORTB of the microcontroller and VCC voltage so that the user can control the activation of the circuit. Finally, an LCD display was connected to the microcontroller as shown in the Figure 1.

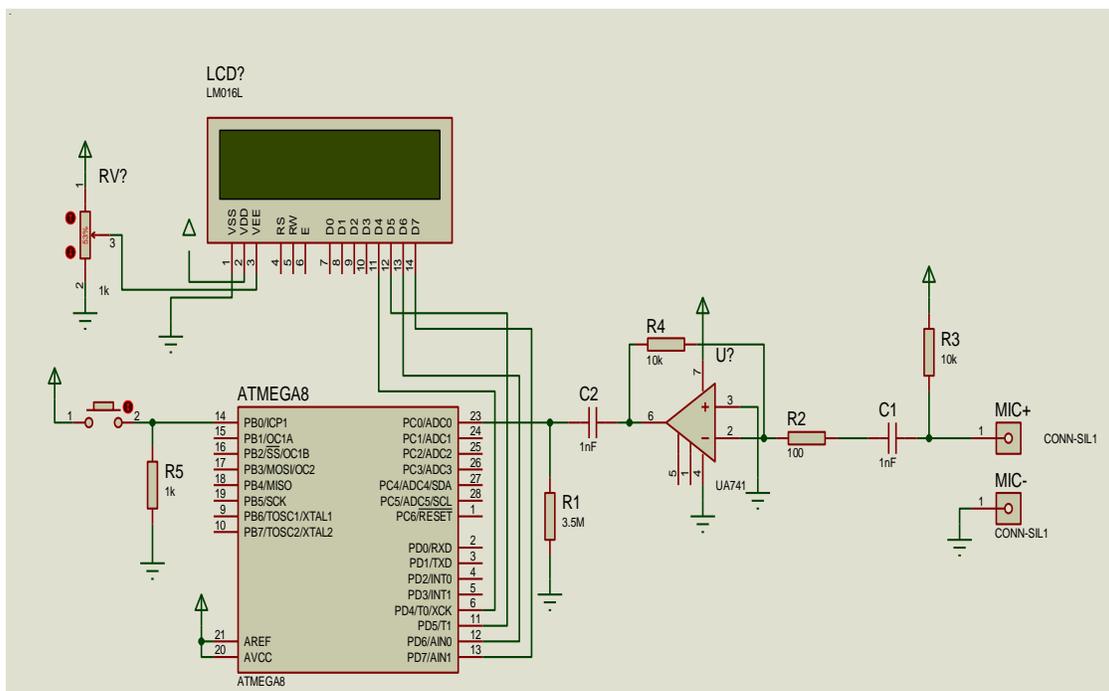


Figure 1 Project Circuit Simulaton

## 2- Software

The design of the software was done in a way so that it can be used in any application needs a Morse code. It consists of three main parts .a new Morse code representation was designed to represent Morse code in digital form. The format was such that , the first three LSB tell how many dot or dash in the character and remaining bits are to tell what are they, 0 for dots and 1 for dashes.

### 1- Sampling and buffering

To analyze the signal an ADC unit was used to sample the signal along with timer/counter1. The samples are compared with the threshold voltage. The threshold voltage is the maximum amplitude of the background noise. If the samples' amplitude greater than the threshold the sample is considered as a top of the peak of the signal and so a variable that count the number of peaks is incremented by one and the timer\counter is reset to zero. The 16 bit timer\counter1 was configured to run on clear on compare mode <sup>(1)</sup>, so that when the counter reaches the compare it generates an interrupt. The interrupt service routine will save number of peaks calculated from the main routine in a buffer, set number of peaks to zero and increment the index of the buffer. Each time an interrupt occur another variable is incremented and when the main routine detects peaks it will save that variable in another buffer. The first buffer will save the number of peaks and the second number will tell the spacing between characters. Both buffers will be used in the next stage for encoding. Appendix A section b. 1 shows the code implementation of the algorithm described above.

### 2- Encode buffer to 3,5 representation

To encode the two buffers produced from step 1 to 3,5 representation described above the software need to know where is the end of each character . and it is done by looking for lager number in the gabs buffer. When the index of that number is found that indicate that the index of last sample of a character in peaks buffer is that number -1. The number of dot and dashes is determined by subtracting that index from peaks buffer form another number initially zero but then it is updated to the index of the large number found in gabs buffer. Then it will look for the next large number in the buffer till the end of the buffers. On the other hand the value of dots and dashes is identified by number of peaks stored on the peaks buffer. If it is greater than 80 peak it will dash and high will be written and else it is a dot and so low will be written. Appendix A section b. 2 shows the code implementation of this mechanism.

### 3- Decode 3,5 representation to ASCII

The third part is a dictionary the translate 3, 5 representation byte on ASCII byte . The content of the dictionary is sorted by the first 3 LSB of the 3, 5 representation for fast search. This part will receive the entire encoded message stored in an array and look up each byte for its ASCII equivalent. First it compares the 3 LSB and then the other 5 bits and return a character. The character will be displayed on the screen. When the first line of the LCD is full it will move to the next line and continue the message. Appendix A section b. 3 show the source code of two function written to achieve this part.

## Result



Figure 2

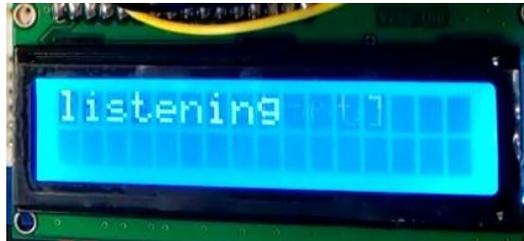


Figure 3



Figure 4

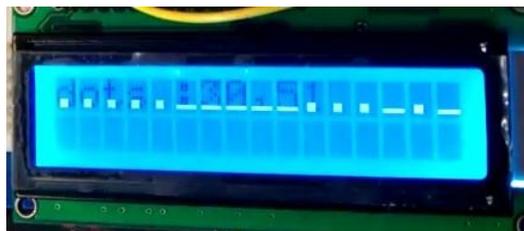


Figure 5



Figure 6

## Discussion of result

Figure 2 to 6 shows the result, Figure 2 system responding to the user interaction through the button and it listen to the sound signal through a microphone. Figure 3 shows starting listening to the signal. Figure 4 shows the signal is been processed by the system and it shows the values stored in each buffer. Not this information is not necessary and it decreases the user's experience but it is shown for testing purpose only. Figure 5 show the Morse code in dot(.) and dash(-) for few time. Figure 6 shows the expected result, the signal is been translated into text and it is displayed on the LCD.

## Conclusion

It can be concluded from the results that the Morse Code Interpreter is a useful device. It can be used as a method to communicate between far distant places where direct voice communication is unavailable. Also, it can help the user to create and transmit secret text messages due to some security reasons and so on.

Many improvements can be added to enhance the Morse Code Interpreter device. For example, An LED indicator can be used in a way such that the LED is ON when there is sound signal and OFF during cutoff. Also, the source code can be adjusted so that the device can be more dynamic and accepts various frequencies of the sound signal. Moreover, the distant range (where the device "hears" the sound signal) can be increased.

### Design constraints:

**Economic:** The project had low-cost components that were available in the university labs.

**Environmental:** The project involved environmentally—safe electronic components.

**Sustainability:** The project can be extended in many ways. For example, the frequency range of the sound signal can be increased and get accepted by the device. Also, the distant of the sound signal from the device can be increased.

**Manufacturability:** The circuit can be easily manufactured and soldered in a printed circuit board.

**Ethical:** The project was entirely done by the group and there was no help from any other person or previous work.

**Health and Safety:** No health and safety concerns of this project.

**Social:** The project may help increasing communication but there are no negative social concerns of this project.

**Political:** There is nothing related to the political issues.

## Appendix A

### Section a. main.c

```
int main()
{
    init_codec();
    LCDinit();

    DDRB &= ~0x01;
    while(1)
    {
        LCDGotoXY(0,0);
        LCDstring("Press [Start]",13);
        uint8_t ok_button = (PINB &0x01);
        if(ok_button !=0)
        {
            LCDGotoXY(0,0);
            LCDstring("listening      ",15);
            my_delay(50);
            listen();
            display();
            uint8_t x = encode();

            LCDclr();
            decode(x);
            init_codec();

            clear_buffers();
            my_delay(10000);
            LCDclr();
        }
    }
    return 0;
}
```

### Section b. morse\_codec.c

1.

```
ISR(TIMER1_COMPA_vect)//interrupt service routine invoked by the timer
/counter 1 output compare
{
if(!saved && start)//save the peaks number only if the listening started and
if it was not save before
{
d_buffer[dots_index] = peaks_count;//save the number of peaks in the desired
index
peaks_count=0;//make the value of number of peaks zero
gabs_index++;//increment the index of the gabs
saved=1;//set save to true to prevent saving again and to tell listen
function that the number of beaks was saved
}
gabs_count++;//calculate the number of interrupts

if(gabs_count >50)//if number of interrupt exceeded 50 that's the end of
message
{

listening =0;//stop listening
}
}
```

```
void listen(void)
{
uint8_t th =25 ,x,flag=1;//set the thresh hold voltage to 0.49 volt and falg
is used to tell that the peak already counted
while(listening)//while listening is not terminated by the interrupt service
routine
{
x = ADCread(0);// read voltage form channel 0
if(x > th && flag)//if the voltage is more than the thresh hold and the peak
is not already counted
{
sei();//enable global interrupt
if(saved)//if the ISR saved number of samples save number of interrupt in the
gabs buffer
{
dots_index++;//increment dots index
g_buffer[gabs_index]=gabs_count;//save the number of gabs
gabs_count=0;
saved=0;//set save = 0 to tell the ISR to save next time
}
reset_timer();//set timer / counter 1 =0
start=1;//now you can execute ISR
peaks_count++;//increment number of peaks
flag=0;//do not count any sample greater the thresh hold
}
else if(x <= th)// if voltage less the thresh hold
{
flag=1;// you can count the peak again
}
}
cli();//disable global interrupt if you are done listening d
}
```

2.

```
uint8_t encode(void)
{
uint8_t counter=0;
uint8_t encoder, gabs=0, dash=0;

for(int i=0; i<100; i++)//first iteration will have no effect on the value
{
dash=0;
if((g_buffer[i]>=5) || (g_buffer[i]==0))
{
gabs =(i-counter); // assign how many "dashes" or "dots" in one letter

for(int j=counter; j<i; j++)
{

if(d_buffer[j]>=80)
{
dash|=1; // assign (HIGH) for "dash"
dash=(dash<<1);
}

else
{
dash&=0xFE; // assign (LOW) for "dot"
dash=(dash<<1);
}

encoder=gabs+(dash<<2); // main register
message[msg_index]=encoder;
msg_index++;
counter=i;
}
if(g_buffer[i]==0 && i >1)return msg_index-1;
}
}
```

3.

```
void decode(uint8_t x)
{
for(int i=0;i<x;i++)
{
if(i==16)LCDGotoXY(0,1);
LCDsendChar(translate(message[i+1]));
}
}
```

```
char translate(uint8_t x){
uint8_t tmp,let;
tmp=x&0x07;
// check # of components

if(tmp==5){ // numbers
tmp=0x1f;
let=(x>>3)&tmp;
```

```

        if(let==0x0f)
            return '1';
        else if(let==0x07)
            return '2';
        else if(let==0x03)
            return '3';
        else if(let==0x01)
            return '4';
        else if(let==0x00)
            return '5';
        else if(let==0x10)
            return '6';
        else if(let==0x18)
            return '7';
        else if(let==0x1C)
            return '8';
        else if(let==0x1E)
            return '9';
        else if(let==0x1F)
            return '0';}

else if(tmp==4){
    tmp=0x0F;
    let=(x>>3)&tmp;
    if(let==0x0D)
        return 'Q';
    else if(let==0x0C)
        return 'Z';
    else if(let==0x0B)
        return 'Y';
    else if(let==0x0A)
        return 'C';
    else if(let==0x09)
        return 'X';
    else if(let==0x08)
        return 'B';
    else if(let==0x07)
        return 'J';
    else if(let==0x06)
        return 'P';
    else if(let==0x04)
        return 'L';
    else if(let==0x03)
        return 'U';
    else if(let==0x02)
        return 'F';
    else if(let==0x01)
        return 'V';
    else if(let==0x05)
        return ' ';
    else
        return 'H';}

else if(tmp==3){
    tmp=0x07;
    let=(x>>3)&tmp;

```

```
        if(let==0x07)
            return 'O';
        else if(let==0x06)
            return 'G';
        else if(let==0x05)
            return 'K';
        else if(let==0x04)
            return 'D';
        else if(let==0x03)
            return 'W';
        else if(let==0x02)
            return 'R';
        else if(let==0x01)
            return 'U';
        else
            return 'S';}

    else if(tmp==2){
        tmp=0x03;
        let=(x>>3)&tmp;
        if(let==0x01)
            return 'A';
        else if(let==0x02)
            return 'N';
        else if(let==0x03)
            return 'M';
        else
            return 'I';}

    else if(tmp==1){
        tmp=0x01;
        let=(x>>3)&tmp;
        if(let==0x01)
            return 'T';
        else
            return 'E';}

    return '?';

}
```