Department of I & CT

MIT, Manipal

Morse Code Encoder & Decoder

ICT 3143 Embedded Systems lab mini project

Vth Sem B.Tech (CCE)

Ashwin Mittal(Reg. No- 220953128)
Anushri Viraj Sakhardande (Reg No- 220953138)
Madhav Arora (Reg No- 220953150)

Under the guidance of

Dr. Santosh Kamath
Associate Professor
Department of I&CT
Manipal Institute of Technology
Manipal, Karnataka, India

Dr. Raviraja Holla M
Assistant Professor - Senior
Department of I&CT
Manipal Institute of Technology
Manipal, Karnataka, India

November,2024

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
A Constituent Unit of MAHE, Manipal

ABSTRACT

Abstract for the Morse Code Decoder and Encoder Project

In today's rapidly advancing technological landscape, efficient, reliable communication methods remain crucial, especially in contexts where traditional communication may fail or be inaccessible. Despite its long history, Morse code is still relevant in various emergency and military settings due to its simplicity and effectiveness. The Morse Code Decoder and Encoder project focuses on developing a versatile system using the LPC1768 microcontroller, enabling the encoding and decoding of Morse code and making it accessible for real-time communication needs. This project aims to demonstrate how embedded systems can be utilised to support essential communication protocols and enhance user interaction with Morse code in a modernised way.

The project design incorporates the LPC1768 microcontroller, chosen for its robust processing capabilities and ease of interfacing. A matrix keyboard serves as the primary input device, allowing users to enter characters for encoding into Morse code, while audiovisual feedback on LCD and buzzer assists in verifying input accuracy. Additionally, the system is designed to decode Morse code input back into readable characters displayed on a seven-segment display. Critical methodologies include embedded systems integration, encoding, decoding, and input-output interfacing, providing a comprehensive learning opportunity for handling hardware-software interfacing and real-time data processing. Tools such as Keil µVision4 for programming and debugging the microcontroller were used to ensure optimal performance.

The system successfully encodes user inputs into Morse code and displays accurate decoded characters, demonstrating its reliability in real-time communication applications. Results indicate that the project integrates input devices and display modules with the microcontroller to create a seamless user experience. This project's significance lies in its application potential for low-resource or emergency communication, where Morse code may be a critical backup communication method. The conclusions drawn from this project emphasise the feasibility and relevance of embedded systems in reviving traditional communication protocols for contemporary needs.

Contents:

List of Figures:

Introduction:

Morse code, a method of encoding text characters as sequences of dots and dashes, has been historically used in telecommunications. This project seeks to modernise the use of Morse code by implementing a system capable of encoding and decoding it in real time. Using an LPC1768 microcontroller efficiently handles user input and display output. The system offers a hands-on demonstration of basic microcontroller principles and peripheral interfacing.

Brief Description About the Project:

The project focuses on implementing a Morse code encoding and decoding system using the LPC1768 microcontroller. There will be a choice between encoding and decoding, which will be inputted through the matrix keyboard.

1. Encoding:

    a. Input: Input from a matrix keyboard, where the user can input the alphabets spaced by an escape character.

    b. Output: LED will light up to denote the dots and dashes. The buzzer will beep alongside the LED at a conventional ratio of 1:3.

2. Decoding:

    a. Input: Input from matrix keyboard where one of the keys represents a dot and another a dash, separated by escape characters.

    b. Output: Displays the decoded work on the LCD.

Methodology

a) Components Required:

- NXP LPC1768 Microcontroller: Processor for handling input and display.
- Matrix Keyboard: Used for user input. Specific keys will be designated for characters (in encoding mode) and dots/dashes (in decoding mode).
- Buzzer: Used to create audio output in the form of long and short beeps
- LED: This will visually represent the Morse code during encoding, with short flashes for dots and long flashes for dashes.
- LCD: Displays decoded and encoded characters and messages to direct the user.

b)Block Diagram

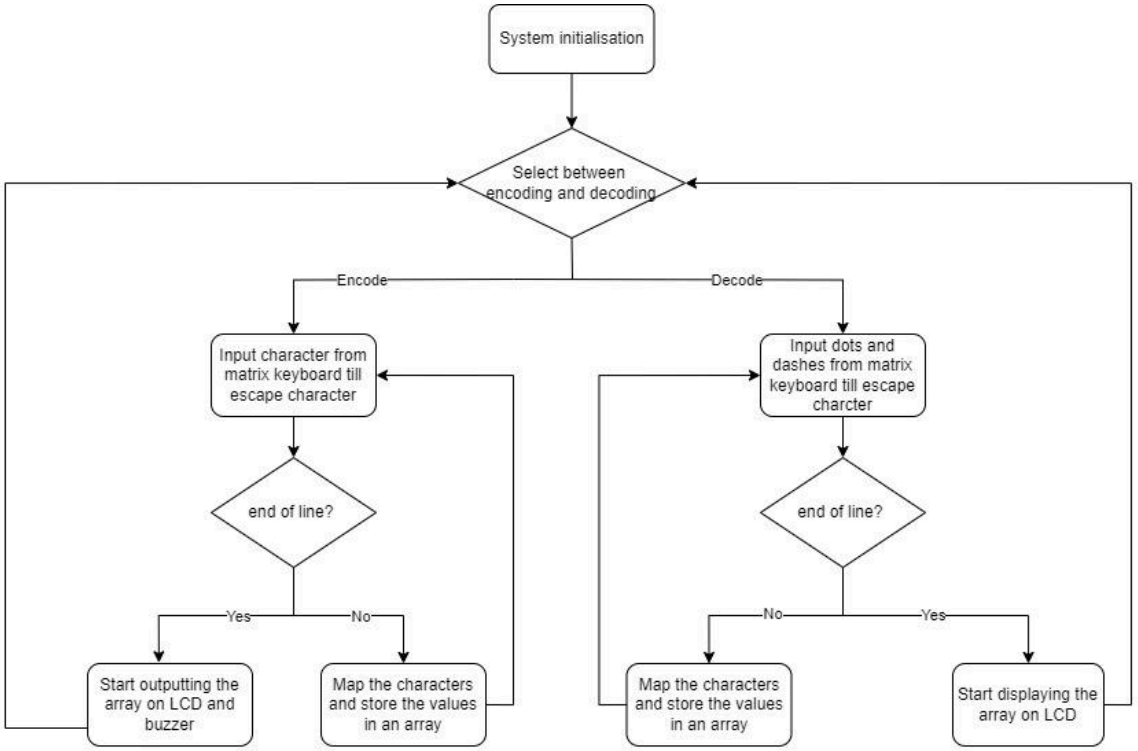

Fig1. Connection diagram of the system

c) Connections:

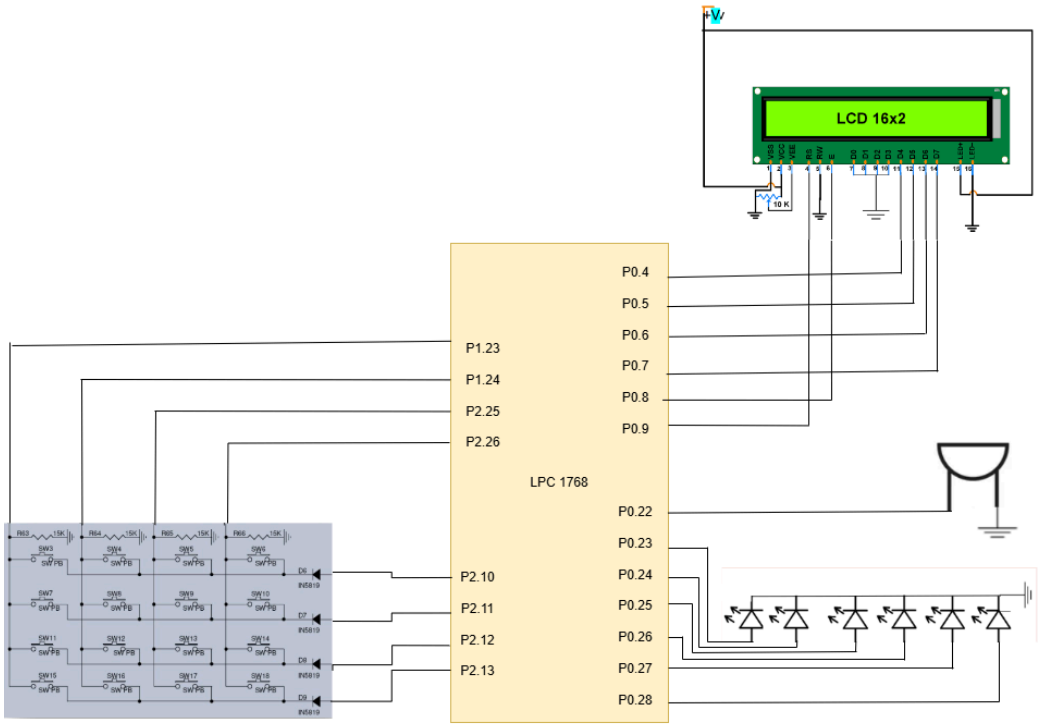

Fig2. Connection diagram of the system

d) Method:

This project designs a Morse code encoder and decoder using the LPC1768 microcontroller, with a matrix keyboard for user input. The system provides encoding and decoding modes, allowing users to interact with Morse code through visual, auditory, and text-based feedback.

1. **Encoding Mode**:
    - **Input**: Users enter alphabetic characters through the matrix keyboard, separated by escape characters, for a clear distinction.
    - **Processing**: The microcontroller converts each character into Morse code, representing dots and dashes with corresponding LED light patterns and buzzer signals.
    - **Output**: The LED and buzzer follow the Morse timing ratio of 1:3, with dots and dashes indicated by short and long signals.

2. **Decoding Mode**:
    - **Input**: Users manually input Morse code via keys for dots and dashes, using escape characters to separate sequences.
    - **Processing**: Each Morse sequence is stored and processed by the microcontroller.
    - **Output**: The decoded text is displayed on an LCD, instantly verifying the entered Morse code.

The system's LED, buzzer and LCD feedback offers an engaging approach to Morse code, making it suitable for hands-on learning with the LPC1768 microcontroller.

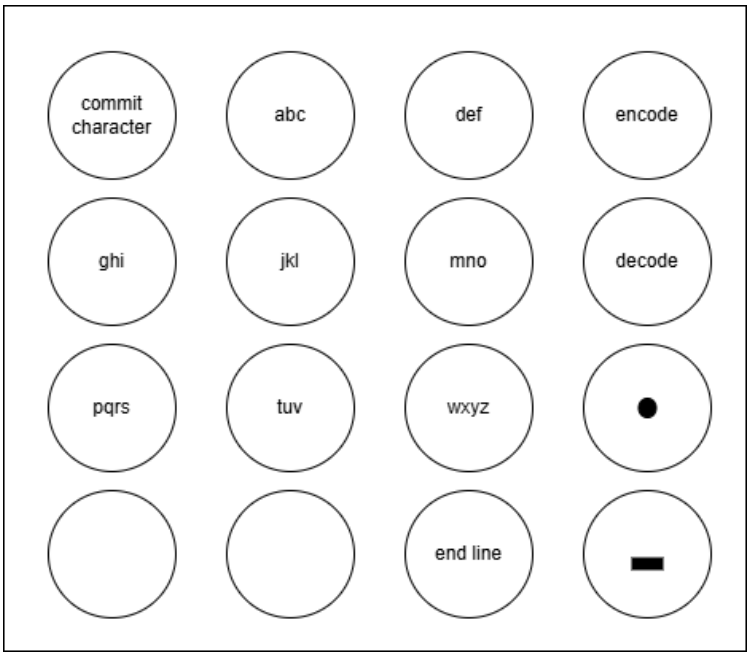

Fig3. Diagram of the matrix keyboard assigned characters

Results and Discussion:

A choice is given to the user to select encode or decode operation. The choice is taken through the matrix keyboard, proceeding to the chosen function.

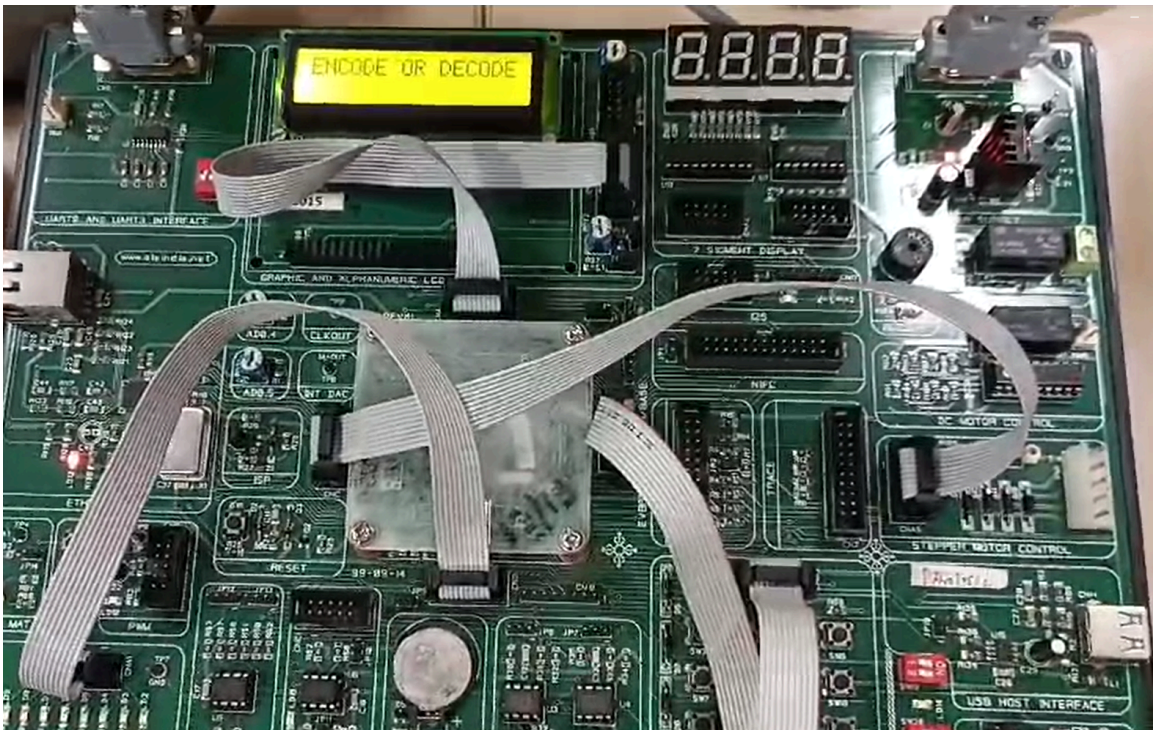

Fig4.The user is prompted to choose encoding or decoding functionality

a) Encoding: The input of characters is taken from the keyboard matrix, and the output of the Morse code encoded string is displayed on the LCD. The beeps of the dots and dashes are produced on the buzzer.

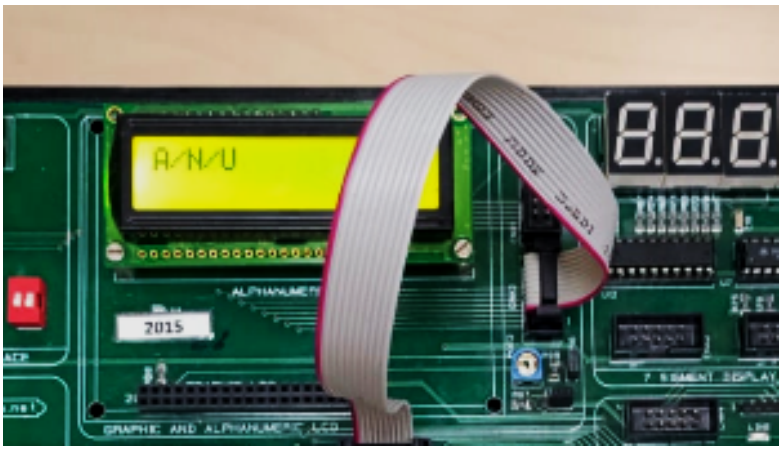

Fig4. Input by user taken through matrix keyboard

Fig5. Resulting in output in Morse

b) Decoding: The characters are taken one after the other on the matrix keyboard and appended to the output array, decoding each character and displaying it on the LCD.

Fig6. Input is taken in Morse through matrix keyboard character by character

Fig7. Obtained output in ASCII characters

References

1. https://www.arduino.cc/education/morse-code-project/

2. https://www.instructables.com/Morse-Code-EncoderDecoder-using-LinkIt-One/

C code with comments:

```
#include <LPC17xx.h>
#include <stdio.h>
#include <string.h>
#define RS_CTRL 0x00000100                  // P0.8
#define EN_CTRL 0x00000200                  // P0.9
#define DT_CTRL 0x000000F0                  // P0.4 TO P0.7
#define BUZZER_LED (0xFF << 22)             // P0.22-P0.28 for buzzer and LED
#define PRESCALE (25000 - 1)

unsigned long int temp1 = 0, temp2 = 0, i, j, temp;
unsigned char flag1 = 0, flag2 = 0;
unsigned int flag;
unsigned char msg[] = "DECODE OR ENCODE";
unsigned char err_msg[] = {"Not a thing"};
unsigned long int init_command[] = {
        0x30,  // 8 bit
        0x30,  // 8 bit
        0x30,  // 8 bit
        0x20,  // 4 bit
        0x28,  // use 2 lines
        0x0c,  // display ON
        0x06,  // increment cursor
        0x01,  // clear display
        0x80}; // set first line first position

signed int row, col;

unsigned char morse[26][5] = {
        ".-",// A
        "-...", // B
        "-.-.", // C
        "-..",// D
        ".",// E
        "..-.", // F
```

```
        "--.",// G
        "....", // H
        "..",// I
        ".---", // J
        "-.-",// K
        ".-..", // L
        "--",// M
        "-.",// N
        "---",// O
        ".--.", // P
        "--.-", // Q
        ".-.",// R
        "...",// S
        "-",// T
        "..-",// U
        "...-", // V
        ".--",// W
        "-..-", // X
        "-.--", // Y
        "--.."// Z
};

unsigned int alpha[26] = {'A', 'B',' C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
'X', 'Y', 'Z'};

unsigned int MatrixMap[3][3] = {
        {0, 1, 2},
        {3, 4, 5},
        {6, 7, 8}};

unsigned char KeyMap[9][5] = {
        {0, 0, 0, 0},          // Key 1
        {'A', 'B', 'C', 0},    // Key 2: A, B, C
        {'D', 'E', 'F', 0},    // Key 3: D, E, F
        {'G', 'H', 'I', 0},    // Key 4: G, H, I
        {'J', 'K', 'L', 0},    // Key 5: J, K, L
```

```
        {'M', 'N', 'O', 0},    // Key 6: M, N, O
        {'P', 'Q', 'R', 'S'},  // Key 7: P, Q, R, S
        {'T', 'U', 'V', 0},    // Key 8: T, U, V
        {'W', 'X', 'Y', 'Z'}   // Key 9: W, X, Y, Z
};
//declare all functions
void encode(void);
void decode(void);
void buzzer_dot(void);
void buzzer_dash(void);
void play_morse(unsigned char[]);
void display_lcd(unsigned char[]);
void lcd_write(void);
void port_write(void);
void input_keyboard(void);
void scan(void);

void delay_lcd(unsigned int r1);
void delayinUS(unsigned int microseconds);
void delayMS(unsigned int milliseconds);

int main(void)
{
        SystemInit();
        SystemCoreClockUpdate();

        // configuration for matrix keyboard
        LPC_PINCON->PINSEL3 = 0;            // P1.23 to P1.26 MADE GPIO
        LPC_PINCON->PINSEL4 = 0;            // P2.10 t P2.13 made GPIO
        LPC_GPIO2->FIODIR = 0x00003C00;     // made output P2.10 to P2.13 (rows)
        LPC_GPIO1->FIODIR = 0;              // made input P1.23 to P1.26 (cols)
        delayMS(100000);
        // configuration for LCD
        LPC_GPIO0->FIODIR |= DT_CTRL | RS_CTRL | EN_CTRL;
        LPC_GPIO0->FIODIR |= BUZZER_LED;
```

```
	// select encode or decode(maybe display a select option in LCD)
	while (1)
	{
		unsigned char msg[] = {"ENCODE OR DECODE"};
		display_lcd(msg);
		input_keyboard();

		// encode to Morse
		if (row == 0 && col == 3)
		{
			unsigned char msg[] = {"Enter letter"};
			display_lcd(msg);
			delay_lcd(10000000);
			encode();
		}
		// decode from Morse
		else if (row == 1 && col == 3)
		{
			unsigned char msg[] = {"Enter code"};
			display_lcd(msg);
			delay_lcd(10000000);
			decode();
		}
		// incorrect key press
		else
		{
			unsigned char msg[] = {"ERROR"};
			display_lcd(msg);
			delay_lcd(10000000);
		}
	}
}

void encode(void)
{
	unsigned char msg[] = {"Encoding.."};
```

```
int keyPress1;                  // keeps track of the number of times a button is pressed
int keyPress2 = -1; // keeps track of the key that is pressed
int count = 0;                  // number of times that the key is pressed
char currChar;                  // show the character that is currently under selection
unsigned char inputBuffer[16];
int bufferIndex = 0;
memset(inputBuffer, 0, sizeof(inputBuffer)); // reset the input buffer each time
display_lcd(msg);
while (1)
{
        input_keyboard(); // get the key pressed
        delayMS(50);
        // commit to memory
        if (row == 0 && col == 0)
        {
                buzzer_dot(); // play dot to confirm the character is committed to the buffer
                delayMS(3000);
                currChar = KeyMap[keyPress2][count];   // get the character that is currently under
selection
                inputBuffer[bufferIndex++] = currChar; // add the character to the input buffer
                display_lcd(inputBuffer);                       // display the input buffer
                keyPress2 = -1;
        }
        // play the dots and dashes
        if (row == 3 && col == 3)
        {
                inputBuffer[bufferIndex++] = '\0';
                buzzer_dash();
                delayMS(3000);
                play_morse(inputBuffer);
                break;
        }
        if (row == 3 && col == 2) // return to main menu
        {
                break;
        }
```

```
                else
                {
                        keyPress1 = MatrixMap[row][col]; // get the key pressed
                        if (keyPress2 == -1) // if no key is pressed previously, log the key pressed
                        {
                                count = 0;
                                keyPress2 = keyPress1;
                        }
                        else if (keyPress1 == keyPress2) // if the same key is pressed increment the count
                        {
                                count++;
                        }
                        else
                        { // if a different key is pressed reset the count and log the key pressed
                                keyPress2 = -1;
                        }
                        currChar = KeyMap[keyPress2][count]; // get the character that is currently selected
                        inputBuffer[bufferIndex] = currChar; // add the character to the inputBuffer
                        display_lcd(inputBuffer);
                }
        }
}

void play_morse(unsigned char input[])
{
        int i = 0, j = 0;
        int index;// Index of the character in the Morse code table
        unsigned char morseBuffer[128]; // Buffer to store Morse code for the input string
        while (input[i] != '\0')
        {
                index = input[i] - 'A'; // Get the index of the character in the Morse code table
                strcpy((char *)&morseBuffer[j], (char *)morse[index]); // Copy Morse code for the character
                j += strlen((char *)morse[index]); // Update the buffer index
                morseBuffer[j++] = '/';  // Add '/' as separator
                i++;
        }
```

```
        morseBuffer[j - 1] = '\0'; // Null-terminate the buffer and remove the last '/'

        display_lcd(morseBuffer);

        i = 0;
        while (i < j) // Loop through the Morse code buffer
        {
                if (morseBuffer[i] == '.')
                {
                        buzzer_dot();
                        delayMS(2000);
                }
                else if (morseBuffer[i] == '-')
                {
                        buzzer_dash();
                        delayMS(2000);
                }
                else if (morseBuffer[i] == '/')
                {
                        delayMS(6000);
                }
                i++;
        }
}

// Play a dot
void buzzer_dot(void)
{
        LPC_GPIO0->FIOSET = BUZZER_LED;
        delayMS(2000);
        LPC_GPIO0->FIOCLR = BUZZER_LED;
}

// Play a dash
void buzzer_dash(void)
{
```

```
	LPC_GPIO0->FIOSET = BUZZER_LED;
	delayMS(6000);
	LPC_GPIO0->FIOCLR = BUZZER_LED;
}

void decode(void)
{
	unsigned char msg[] = {"Decoding.."};
	// unsigned char msg1[5];
	unsigned char inputBuffer[16];// Buffer to store the input Morse code
	unsigned char outputBuffer[16]; // Buffer to store the output string
	unsigned int ctr = 0;// Counter for the input buffer
	unsigned int ctr2 = 0;// Counter for the output buffer
	unsigned int z;

	display_lcd(msg);
	// matrix keyboard input for characters expect an escape sequence after each character
	while (1)
	{
		input_keyboard();// get the key pressed
		if (row == 3 && col == 3) // for the dot
		{
			inputBuffer[ctr++] = '.'; // add the dot to the inputBuffer
			inputBuffer[ctr] = '\0';
			display_lcd(inputBuffer);
		}
		else if (row == 2 && col == 2) // for the dash
		{
			inputBuffer[ctr++] = '-'; // add the dash to the inputBuffer
			inputBuffer[ctr] = '\0';
			display_lcd(inputBuffer);
		}
		else if (row == 0 && col == 0) // for end of character
		{
			unsigned char msg[] = {"Converting"};
			display_lcd(msg);
```

```
                        inputBuffer[ctr] = '\0';
                        for (z = 0; z < 26; z++)
                        {
                                if ((strcmp((const char *)morse[z], (const char *)inputBuffer) == 0))
                                        // Compare the input Morse code with the Morse code table
                                {

                                        outputBuffer[ctr2++] = alpha[z]; // Add character to output buffer
                                        outputBuffer[ctr2] = '\0';
                                        display_lcd(outputBuffer);
                                        delayMS(1000);
                                        inputBuffer[0] = '\0'; // Reset the input buffer
                                        ctr = 0;   // Reset the counter
                                        break;
                                }
                        }
                        if (z == 27)
                        {
                                display_lcd(err_msg); // Display error message
                        }
                }
                else if (row == 3 && col == 2) // return to main menu
                {
                        break;
                }
        }
}
// Returns which key pressed
void input_keyboard(void)
{
        int Break_flag = 0;
        row = 0;
        col = 0;
        while (1)
        {
                // poll rows
```

```
		for (row = 0; row < 4; row++)
		{
			//Send high to each row
			if (row == 0)
				temp = 1 << 10;
			else if (row == 1)
				temp = 1 << 11;
			else if (row == 2)
				temp = 1 << 12;
			else if (row == 3)
				temp = 1 << 13;
			// temp=1<<(10+row);
			LPC_GPIO2->FIOPIN = temp; // WRITING TO PORT 2
			flag = 0;
			delayMS(10000);
			scan();
			if (flag == 1)
			{
				Break_flag = 1; // break outer loop as well
				delayMS(2000);
				break;
			}
		}
		if (Break_flag == 1)
			break;
	}
}

void scan(void)
{
	unsigned long temp3;

	temp3 = LPC_GPIO1->FIOPIN;
	temp3 &= 0x07800000;
	if (temp3 != 0x00000000)
	{
```

```
                flag = 1;
                if (temp3 == 1 << 23)
                        col = 0;
                else if (temp3 == 1 << 24)
                        col = 1;
                else if (temp3 == 1 << 25)
                        col = 2;
                else if (temp3 == 1 << 26)
                        col = 3;
        }
}

void display_lcd(unsigned char msg[])
{
        flag1 = 0; // COMMAND MODE
        for (i = 0; i < 9; i++)
        {
                temp1 = init_command[i];
                lcd_write();
        }
        flag1 = 1; // DATA MODE
        i = 0;
        while (msg[i] != '\0')
        {
                temp1 = msg[i];
                lcd_write();
                i += 1;
        }
}

void lcd_write(void)
{

        flag2 = (flag1 == 1) ? 0 : ((temp1 == 0x30) || (temp1 == 0x20)) ? 1: 0;
        temp2 = temp1 & 0xf0;
        port_write();
```

```
	if (flag2 == 0)
	{
		temp2 = temp1 & 0x0f;
		temp2 = temp2 << 4;
		port_write();
	}
}

void port_write(void)
{
	LPC_GPIO0->FIOPIN = temp2;
	if (flag1 == 0)
		LPC_GPIO0->FIOCLR = RS_CTRL;
	else
		LPC_GPIO0->FIOSET = RS_CTRL;
	LPC_GPIO0->FIOSET = EN_CTRL;
	delay_lcd(25);
	LPC_GPIO0->FIOCLR = EN_CTRL;
	delay_lcd(300000);
}

// lcd delay
void delay_lcd(unsigned int r1)
{
	unsigned int r;
	for (r = 0; r < r1; r++);
	return;
}

void delayinUS(unsigned int microseconds)
{
	LPC_TIM0->TCR = 0x02;
	LPC_TIM0->PR = 0;// Set prescaler to the value of 0
	LPC_TIM0->MR0 = microseconds - 1; // Set match register for 10us
	LPC_TIM0->MCR = 0x01; // Interrupt on match
	LPC_TIM0->TCR = 0x01;  // Enable timer
```

```
	while ((LPC_TIM0->IR & 0x01) == 0); // Wait for interrupt flag
	LPC_TIM0->TCR = 0x00; // Stop the timer
	LPC_TIM0->IR = 0x01;  // Clear the interrupt flag
}
void delayMS(unsigned int milliseconds) // Using Timer0
{
	delayinUS(1000 * milliseconds);
}
```