



**AMRITA VISHWA VIDHYAPEETHAM**

**Amaravathi campus, AP**

Dept. of Computers and Communication Engineering

**MICRO CONTROLLERS AND INTERFACING**

COURSE NAME: **MICRO CONTROLLERS AND INTERFACING**

REVANTH ROY NAIDU

AV.EN.U4CCE23034

# INDEX

1. INTRODUCTION 2
2. OBJECTIVES 4
3. METHODOLOGY 4
4. COMPONENTS REQUIRED 4
5. CIRCUIT DIAGRAM 5
6. CODE 6-9
7. CODE EXPLANATION 10
8. APPLICATIONS 12
9. RESULT 13
10. CONCLUSION 14

### INTRODUCTION

MORSE CODE - is a method of encoding textual information using sequences of dots and dashes, or short and long signals, to represent each letter of the alphabet, numeral, and some punctuation characters. Developed in the early 1830s and 1840s by Samuel Morse and Alfred Vail, Morse code was initially used for long-distance communication via telegraph systems. The simplicity and efficiency of Morse code allowed it to become a widely adopted method for transmitting messages over wire-based telegraph systems, radio communication, and even visual signaling methods.

**Structure and Usage**

 Symbols: Morse code uses combinations of dots (·) and dashes (−) to represent letters, numbers, and punctuation. For example, the letter "A" is represented as "·−", and the number "1" is "·−−−−".

 Transmission: Originally used with telegraph machines, Morse code can be transmitted by sound (such as beeps or clicks), light (flashes), or visual signals (like flags or written symbols).

 Standardization: The International Morse Code, which is the most commonly used version today, was standardized in the 1860s and includes distinct codes for letters, numbers, and a limited set of punctuation marks.

## PROTEUS

Proteus serves as the backbone for designing and simulating the Morse code generator door With Proteus, engineers and developers can visually construct the circuit diagram, integrating components such as the LPC2148 microcontroller, keypad,LCD etc

Features:

**Intuitive Interface**: The schematic capture tool allows users to create and edit circuit designs easily with a user-friendly interface.

**Component Library:** Proteus includes a vast library of electronic components, such as resistors, capacitors, ICs, transistors, and more.

**Hierarchical Design:** Supports hierarchical schematics, enabling the creation of complex designs with multiple levels.

#### Schematic Capture:

Designers can visually represent the connections and interactions between components, providing a clear blueprint for the project.

#### Real-Time Simulation:

Proteus enables users to simulate the circuit in real-time, allowing for thorough testing and debugging before physical implementation.

**Microcontroller Simulation:** With Proteus, developers can simulate the behaviour of the Arduino microcontroller, ensuring that the code interacts seamlessly with the hardware components.

**3. Keil IDE:**

Keil uVision 4 is an integrated development environment (IDE) widely used for embedded systems development. Developed by Keil, a subsidiary of ARM Holdings, this IDE provides comprehensive tools for microcontroller programming, particularly for ARM-based microcontrollers. It is known for its robust feature set, ease of use, and support for a wide range of microcontroller families.

**Features**

**Project Management:** Keil uVision 4 allows developers to manage multiple projects simultaneously,

**:**providing a structured environment for organizing source files, header files, and libraries. Its project management capabilities include easy configuration of project settings, device selection, and peripheral simulation.

**Editor:** The IDE includes a powerful code editor with syntax highlighting, code completion, and other productivity features. These tools help developers write, navigate, and debug their code more efficiently.

**Compiler and Assembler:** Keil uVision 4 integrates with the ARM RealView Compilation Tools (RVCT), which include a C/C++ compiler and assembler. These tools generate optimized code for ARM microcontrollers, ensuring efficient use of resources and high performance.

**Debugger:** The integrated debugger in Keil uVision 4 supports various debugging methods, including simulation, hardware debugging with JTAG, and real-time trace. It provides comprehensive debugging features such as breakpoints, watch windows, call stack visibility, and memory inspection.

**Simulator:** Keil uVision 4 includes a built-in simulator that allows developers to test and debug their code without needing the actual hardware. This feature is particularly useful during the initial stages of development.

The seamless integration of Proteus and Keil IDE streamlines the development process of the Morse Code generator system. We can design and simulate the circuit in Proteus,

ensuring that all components interact as intended. Simultaneously, they can write and test the code in Keil IDE, verifying that the microcontroller effectively manages inputs and controls the mechanism.This integration allows for efficient design, simulation, and implementation, ultimately enhancing security and providing users with a seamless and user-friendly access control solution.

### METHODOLOGY

#### System Design and Planning

 **Define Requirements**: Outline the project scope, including hardware components (LPC2148 microcontroller, keypad, LCD, LED/buzzer) and software requirements.

 **Create Schematics**: Design the circuit diagram for connecting the keypad, LCD, and LED/buzzer to the LPC2148.

#### Hardware Setup

 **LPC2148 Microcontroller**: Set up the LPC2148 development board and ensure it is functioning correctly.

 **Connect Keypad**: Interface the keypad with the LPC2148 using GPIO pins. Assign specific pins for row and column connections.

 **Connect LCD**: Interface the LCD with the LPC2148. If using an I2C-enabled LCD, connect it via the I2C bus.

 **Connect LED/Buzzer**: Connect an LED or buzzer to the LPC2148 for Morse code output signals.

#### Software Development

 **Environment Setup**: Install Keil uVision 4 IDE and configure it for LPC2148 development.

 **Project Initialization**: Create a new project in Keil and configure it for the LPC2148 microcontroller. Add necessary startup files and initialize the microcontroller settings.

#### Keypad Interface Programming

 Keypad Initialization: Write code to initialize the keypad GPIO pins.

 Keypad Reading: Implement a function to read input from the keypad and debounce the keys.

#### LCD Interface Programming

 LCD Initialization: Write code to initialize the LCD. If using I2C, include I2C initialization.

 Display Functions: Develop functions to send characters and strings to the LCD for display.

#### I2C Communication Setup

 I2C Initialization: Write code to initialize the I2C peripheral on the LPC2148.

 I2C Communication: Implement functions for I2C communication to interact with the LCD or other I2C peripherals.

#### Morse Code Conversion and Signal Generation

 Morse Code Algorithm: Develop a function to convert input text characters to their Morse code equivalents using a lookup table.

 Signal Output: Implement a function to generate Morse code signals using the LED or buzzer. Ensure correct timing for dots (short signal), dashes (long signal), and spaces.

#### Integration and Testing

 Combine Modules: Integrate the keypad reading, Morse code conversion, and signal output functions into the main program.

 Debugging: Use Keil's debugging tools to test the system. Set breakpoints and monitor variables to ensure correct operation.

 Testing: Test the entire system with different inputs to verify correct Morse code generation and display.

## COMPONENTS REQUIRED

LPC2148

Keypad

LCD : LM018L, LM016L

I2C extender LCD : PCF8574A LED-RED

Transistor : BC547 Resistors : 10K ohm Ground Pin

Power Supply : 5.5v, 5v and 3.3v

## CIRCUIT DIAGRAM

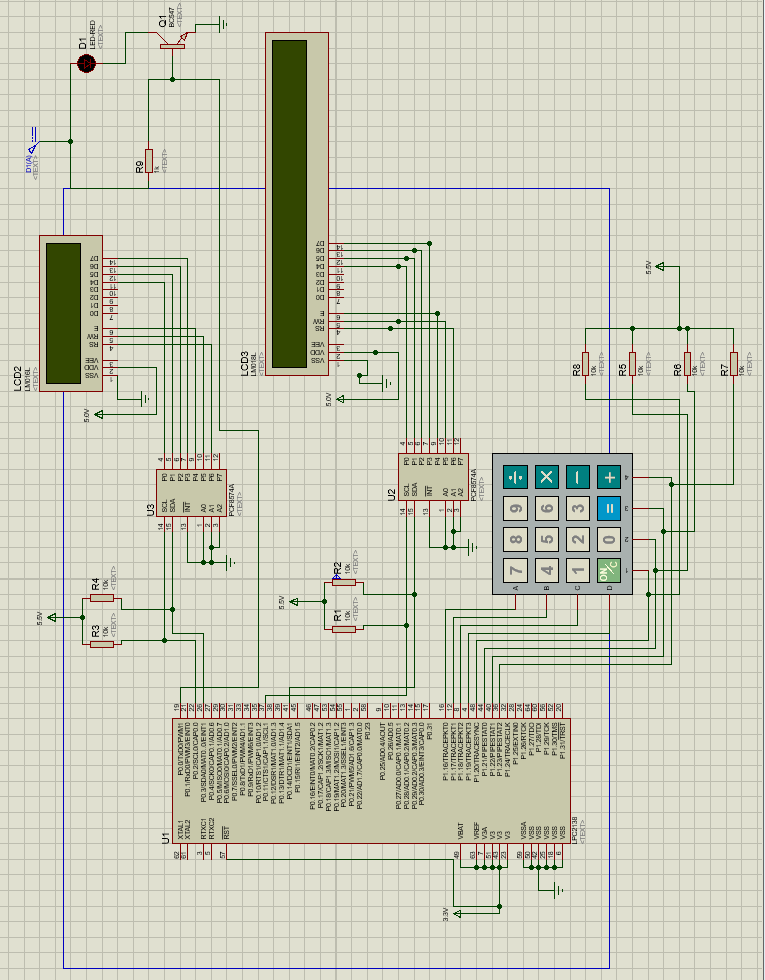


Figure 1

**CODE**

#include <lpc213x.h> #include <string.h>

#define SLAVE\_ADDR 0x70 // constants #define MAX 12

#define AA 2

#define SI 3

#define STO 4

#define STA 5

#define I2EN 6

unsigned char write[] = {0xf7, 0xfd, 0xfb, 0xfe};

// array to write in row of keypad

unsigned char comb[] = {0x77, 0xb7, 0xd7, 0xe7, 0x7b, 0xbb, 0xdb, 0xeb, 0x7d, 0xbd, 0xdd, 0xed, 0x7e, 0xbe, 0xde, 0xee}; // combination of row and column

char num[] = "0000032106540987";

// number corresponding to combination

unsigned char seg\_values[] = {0x3F, 0x3F, 0x3F, 0x3F, 0x3F, 0x4F, 0x5B, 0x06, 0x3F, 0x7D, 0x6D, 0x66, 0x3F, 0x6F, 0x7F, 0x07}; // 7 segment data corresponding to number

void wait(int count)

{

while (count--)

;

}

const char \*digitMorseCodeTable[10] = {

"-----", ".----", "..---", "...--", "....-", " ",

"-....", "--...", "---..", "----."}; // morse code for 0 to 9

void I2C\_Init(void) // initialization for i2c

{

VPBDIV = 0x02; // sets FOSC = 60MHHZ

PINSEL0 = 0x30C00050; // set po.2,p0.3 to sda scl

I2C1SCLH = 150; // 50%duty,I2CFreq->100KHz,PCLK=30MHz I2C1SCLL = 150;

I2C1CONSET = (1 << I2EN); // Enable I2C module

I2C0SCLH = 150; // 50%duty,I2CFreq->100KHz,PCLK=30MHz I2C0SCLL = 150;

I2C0CONSET = (1 << I2EN);

}

int I2C1\_Start() // start i2c communication

{

I2C1CONCLR = 1 << STO; I2C1CONCLR = 1 << SI; I2C1CONSET = 1 << STA; I2C0CONCLR = 1 << STO; I2C0CONCLR = 1 << SI; I2C0CONSET = 1 << STA;

return 0;

}

void delay\_ms(int count)

{

int j = 0, i = 0;

for (j = 0; j < count; j++)

{

for (i = 0; i < 35; i++)

; // At 60Mhz, the below loop introduces delay of 10 us

}

}

void senddata1(char data)

{

while (!(I2C1CONSET & 0x08)) // wait till SI becomes 1

;

I2C1DAT = data;

I2C1CONCLR = 1 << SI; // clearing SI bit delay\_ms(200);

}

void senddata0(char data)

{

while (!(I2C0CONSET & 0x08))

;

I2C0DAT = data; I2C0CONCLR = 1 << SI;

delay\_ms(200);

}

void sendchar1(char data)

{

senddata1(0x50 | (data >> 4)); // sending 1st 4 bits with en =1 delay\_ms(50);

senddata1(0x40 | (data >> 4)); // sending 1st 4 bits with en =0 delay\_ms(50);

senddata1(0x50 | (data & 0x0f)); // sending next 4 bits with en =1 delay\_ms(50);

senddata1(0x40 | (data & 0x0f)); // sending next 4 bits with en =0 delay\_ms(50);

delay\_ms(50);

}

void sendchar0(char data)

{

senddata0(0x50 | (data >> 4)); delay\_ms(50);

senddata0(0x40 | (data >> 4)); delay\_ms(50);

senddata0(0x50 | (data & 0x0f)); delay\_ms(50);

senddata0(0x40 | (data & 0x0f)); delay\_ms(50);

delay\_ms(50);

}

void LCD\_init()

{

int i = 0;

char code = SLAVE\_ADDR; I2C\_Init(); I2C1\_Start(); wait(4000);

while (!(I2C1CONSET & 0x08)) // wait till SI becomes 1

{

};

IO1SET = (1 << 21); I2C1CONCLR = 1 << STO; I2C1CONCLR = 1 << STA; I2C1CONSET = 1 << AA;

I2C1DAT = code; I2C1CONCLR = 1 << SI;

while (!(I2C0CONSET & 0x08))

{

};

IO0SET = (1 << 21); I2C0CONCLR = 1 << STO; I2C0CONCLR = 1 << STA; I2C0CONSET = 1 << AA;

I2C0DAT = code; I2C0CONCLR = 1 << SI;

while (!(I2C1CONSET & 0x08))

{

};

if (I2C1STAT == 0x18) // status code which indicates ready to receive

{

IO1SET = (1 << 23); I2C1CONSET = 1 << AA; I2C1DAT = 0x00; I2C1CONCLR = 1 << SI;

while (!(I2C1CONSET & 0x08))

;

for (i = 0; i < 2000; i++) wait(800);

if (I2C1STAT == 0x28) // when ack received

{

senddata1(0x10); // function set senddata1(0x00);

senddata1(0x12); // display off cursor on senddata1(0x02);

senddata1(0x12); // display off cursor on senddata1(0x02);

senddata1(0x18); // entry mode set senddata1(0x08);

senddata1(0x10); // display on senddata1(0x00);

senddata1(0x1e); // display shift left senddata1(0x0e);

senddata1(0x10); // function set senddata1(0x00); senddata1(0x16); // entry mode senddata1(0x06);

senddata1(0x10); // clear display senddata1(0x00);

senddata1(0x11); // display on blink on senddata1(0x01);

senddata1(0x18); // display shift right senddata1(0x08);

senddata1(0x10); // function set senddata1(0x00);

}

}

while (!(I2C0CONSET & 0x08))

;

if (I2C0STAT == 0x18)

{

IO0SET = (1 << 23); I2C0CONSET = 1 << AA; I2C0DAT = 0x00; I2C0CONCLR = 1 << SI;

while (!(I2C0CONSET & 0x08))

;

for (i = 0; i < 2000; i++) wait(800);

if (I2C0STAT == 0x28)

{

senddata0(0x10); senddata0(0x00); senddata0(0x12); senddata0(0x02); senddata0(0x12); senddata0(0x02); senddata0(0x18); senddata0(0x08); senddata0(0x10); senddata0(0x00); senddata0(0x1e); senddata0(0x0e); senddata0(0x10); senddata0(0x00); senddata0(0x16); senddata0(0x06); senddata0(0x10); senddata0(0x00); senddata0(0x11); senddata0(0x01); senddata0(0x18); senddata0(0x08); senddata0(0x10); senddata0(0x00);

}

}

}

void init()

{

VPBDIV = 0x02; PINSEL1 = 0x0; PINSEL2 = 0x0;

IODIR0 = 0xFF;

IODIR1 = 0x000F0000; // 16,17,18,19 as output (row)

}

char GetKey()

{

int row = 0;

unsigned char w, w\_final; int ind;

int i; int temp; while (1)

{

IO1CLR = 0xffffffff; w = write[row];

IO1SET |= (w << 16); // writing to the row (16,17,18,19). so shifting 16 bits

delay\_ms(1000);

temp = IO1PIN; // reading the bits

w\_final = ((temp >> 16) & 0xFF); // right shift 16 times and ANDing with FF to remove unnecessary data

if (w\_final != w) break;

row++;

if (row >= 4)

row = 0;

}

for (i = 0; i < 16; i++)

{

if (comb[i] == w\_final) ind = i;

}

return num[ind];

}

const char \*getMorseCode(char digit)

{

switch (digit)

{

case '0':

return " ";

case '1':

return " ";

case '2':

return ". ";

case '3':

return "...--";

case '4':

return "....-"; case '5':

return ". ";

case '6':

return "- ";

case '7':

return "-- ";

case '8':

return "---.."; case '9':

return "----."; default:

return "Invalid input"; // In case the input is not a digit

}

}

void delay\_ms\_1(int j)

{

int x, i;

for (i = 0; i < j; i++)

{

for (x = 0; x < 6000; x++)

;

}

}

void led(char input)

{

int led\_pin = 0; // connection of LED to p0.0 int dot = 100; // ON time for dot

int dash = 250; // ON time for dash switch (input)

{

case '0': // -----

IOSET0 |= 1 << led\_pin; delay\_ms\_1(dash); IOCLR0 |= 1 << led\_pin; delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(500);

break;

case '1': // .----

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(500);

break;

case '2': // ..---

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(500);

break;

case '3': // ...--

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(500);

break;

case '4': // ....-

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(500);

break;

case '5': // .....

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(500);

break;

case '6': // -....

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(500);

break;

case '6': // -....

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin);

delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(500);

break; case '9':

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dash);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOSET0 |= (1 << led\_pin); delay\_ms\_1(dot);

IOCLR0 |= (1 << led\_pin); delay\_ms\_1(500);

break;

}

}

int main()

{

char x; int i = 0;

int count = 20;

const char \*morseCode; LCD\_init();

init();

while (count > 0)

{

x = GetKey(); // keyboard input

sendchar0(x); // sending to LCD 1 for display morseCode = getMorseCode(x); // morsecode of character input for (i = 0; morseCode[i] != '\0'; i++)

{

sendchar1(morseCode[i]); // sending to second LCD for display

}

sendchar1(' ');

led(x); // switching led according to Morse code delay\_ms(35000);

count--;

}

senddata1(0x10); senddata1(0x0C);

senddata0(0x10); // display off cursor off senddata0(0x0C);

return 0;

}

**Aviation Communication:**

# APPLICATIONS

 Morse code is still used in aviation for navigational aids. Many navigation beacons transmit their identifiers in Morse code, allowing pilots to verify the beacon's identity and ensure correct navigation.

#### Maritime Communication:

 Historically critical for maritime communication, Morse code remains important in maritime operations. It is used for distress signals (e.g., SOS: ···−−−···) and identifying navigational aids like lighthouses and buoys.

#### Amateur Radio (Ham Radio):

 Enthusiasts in the amateur radio community often use Morse code for communication. It's valued for its simplicity and effectiveness in low-signal conditions. Ham radio operators often need to pass a Morse code proficiency test to obtain certain licenses.

#### Emergency Signaling:

 Morse code can be used in emergency situations where other communication methods fail. It's possible to send Morse code signals using improvised means such as light flashes, sound (taps or beeps), or visual signals (hand signals).

#### Assistive Technology:

 Morse code is used as an assistive technology for people with disabilities. Specialized devices and software allow individuals with limited mobility or speech to communicate using Morse code inputs, which are then converted to text or speech.

#### Broadcasting and Identification:

 Some radio and television stations use Morse code to transmit their call signs periodically. This helps in identifying the station, especially in regions where audio signals may be unclear or in regulatory compliance.

# RESULT

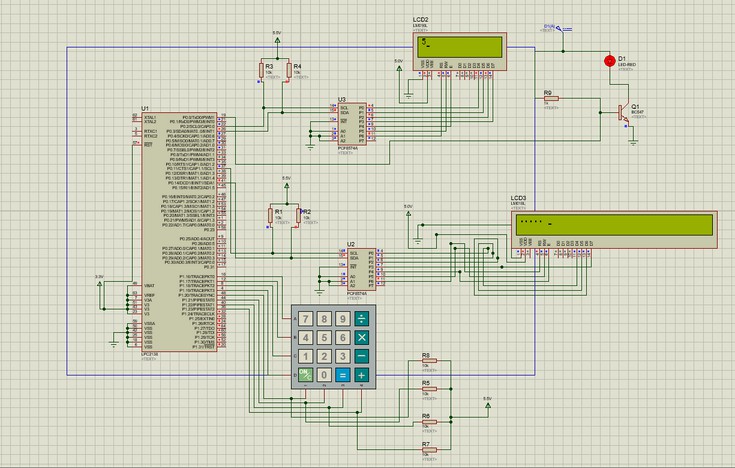


Figure 2: Working of Morse Code Generator (On entering the correct single input)

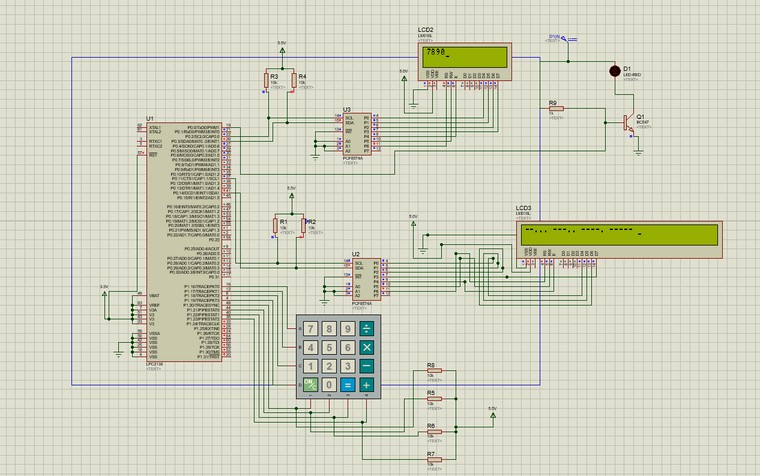


Figure 3: Working of the Morse Code Generator (On entering multiple inputs