ISLAMIC UNIVERSITY OF TECHNOLOGY (IUT)

ORGANISATION OF ISLAMIC COOPERATION (OIC)

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

Project Title: Bluetooth Interfacing with Microcontrollers

Author(s)/Student(s) Names: Zakia Zaman ID: 200021302
Fariha Mahjabin Islam ID: 200021304
Ragib Yeaser ID: 200021324
Tanzim Noor Tanmoy ID: 200021328
Md. Emon ID: 200021332
Nasteho Mohamoud Dahir ID: 200021360

Course Code : EEE 4706
Course Name :Microcontroller Based System Design Lab

Name of Supervisor: Md. Arif Hossain

Submission Date : 28.04.2025

1. Objective :

The objective of this project is to integrate Bluetooth communication with the 8051 microcontroller to wirelessly control and monitor various hardware components. The system allows users to interact remotely with LEDs, relays, a stepper motor, and a buzzer, as well as to send and decrypt encrypted messages.

The main goals of the project are:

- **LED Control and Morse Code Transmission**: Controlling all 8 LEDs through Bluetooth commands and transmitting Morse code using one LED. The Morse signals (dots and dashes) are displayed live on an LCD, along with the decoded short message (such as a name).

- **Relay-Controlled Lights**: Operating two relays connected to external lights using Bluetooth, demonstrating wireless switching of electrical loads.

- **Encrypted Communication**: Sending an encrypted message via Bluetooth and displaying the decrypted version on the LCD, implementing basic encryption and decryption techniques.

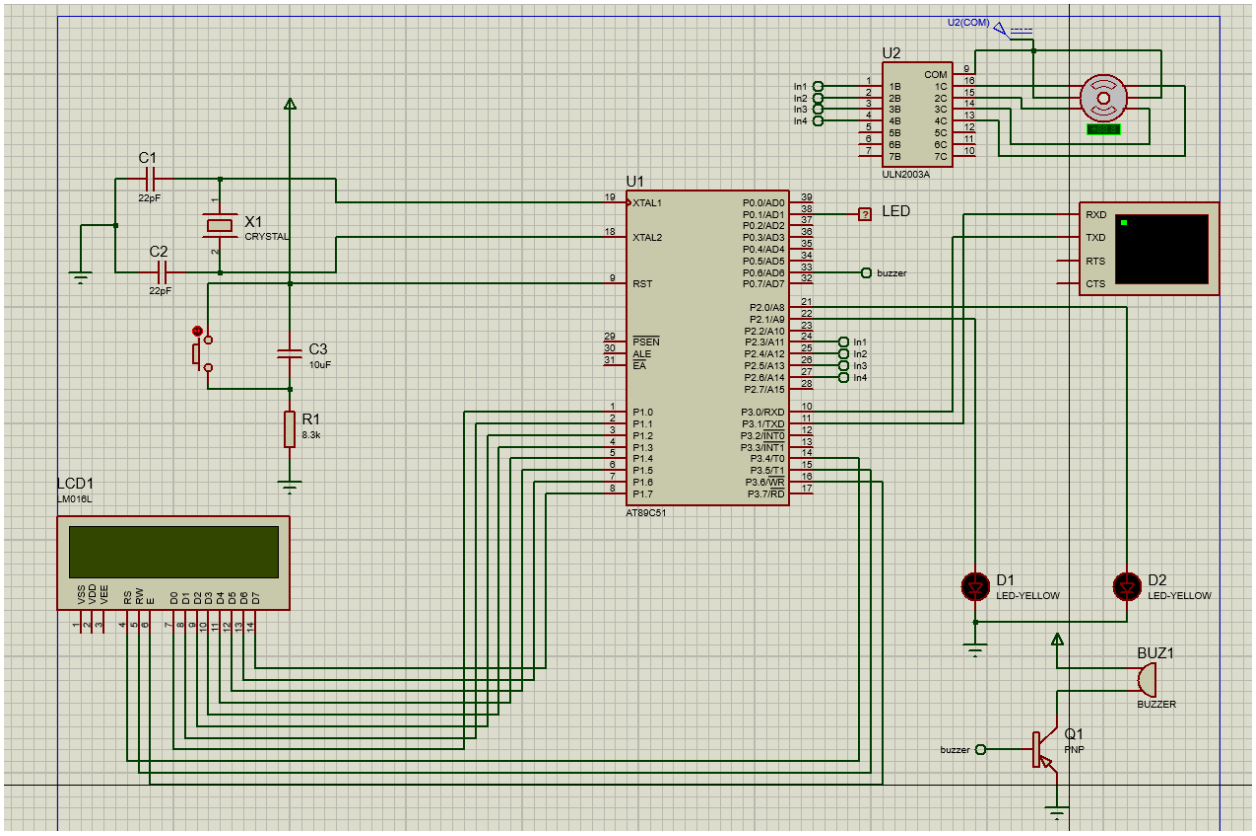
- **Stepper Motor Speed Control**: Adjusting the speed of a stepper motor remotely through Bluetooth commands using the 8051 microcontroller.

- **Alarm System for Morse Code**: Triggering a buzzer alarm to notify the arrival of incoming Morse code messages, ensuring real-time alerts.

2. Required Components

Name	Price (BDT)
8051 Development Board	7000
Bluetooth Module (HC-05)	350
2-Channel 12V Relay Module	245
Stepper Motor Driver Module: ULN2003A	230
DC 5V Stepper Motor	
Total	**7825**

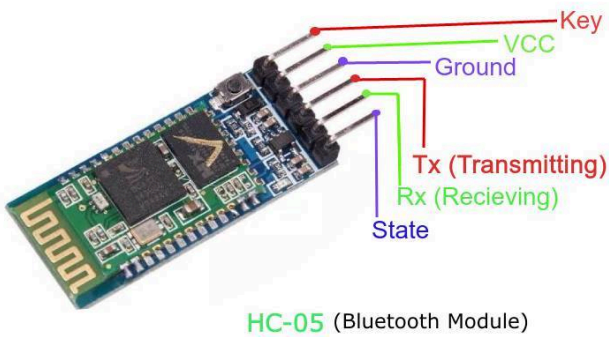
3. Circuit Diagram :



Components' Description:

- **Bluetooth Module (HC-05):**

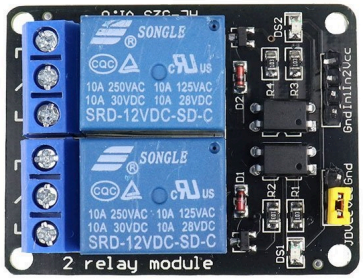
The HC-05 is a Bluetooth 2.0+EDR module operating at 2.402–2.480 GHz with FHSS for interference reduction. It supports UART communication and both master/slave modes, enabling wireless control between devices.

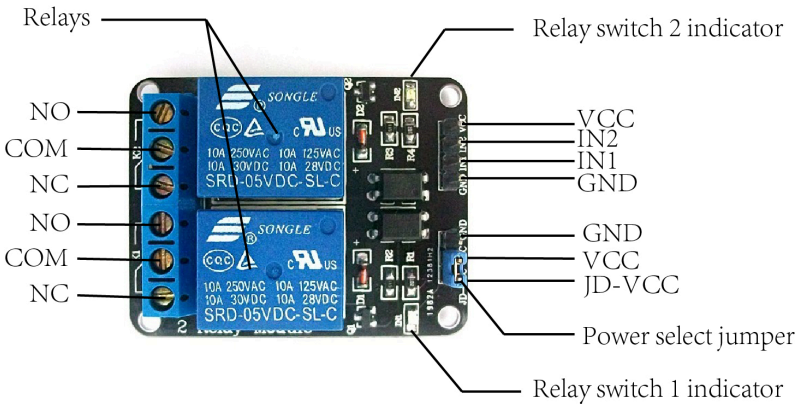


HC-05 (Bluetooth Module)

- **2-Channel 12V Relay Module:**

A dual-channel relay board operating at 12V DC, capable of switching 10A/250V AC or 10A/30V DC loads. Features optocoupler isolation, freewheeling diode protection, and LED status indicators. Compatible with 3.3V-5V control signals, it includes screw terminals (C/NC/NO) and mounting holes for easy integration in automation projects. Ideal for controlling lights, motors, and appliances.





- **Stepper Motor Driver (ULN2003A):**

The ULN2003A is a Darlington transistor array driver capable of sinking 500mA per channel. It interfaces with unipolar stepper motors (e.g., 28BYJ-48) using 4/5-wire control from a microcontroller.

Proteus Circuit Pinout of the Project:

P0.0	N/A
P0.1	LED
P0.2	N/A
P0.3	N/A
P0.4	N/A
P0.5	N/A
P0.6	BUZZER
P0.7	N/A
P1.0	LCD
P1.1	
P1.2	
P1.3	
P1.4	
P1.5	
P1.6	
P1.7	
P2.0	RELAY-1
P2.1	RELAY-2
P2.2	N/A
P2.3	Motor Driver
P2.4	
P2.5	
P2.6	

P2.7	N/A
P3.0	TXD
P3.1	RXD
P3.2	N/A
P3.3	N/A
P3.4	RS
P3.5	RW
P3.6	ENABLE
P3.7	N/A

4. Various Modes:

Mode No.	Mode Name	Description
1	Morse Code Transmission	Sends a Morse code message via LED with real-time display of dots, dashes, and decoded text on the LCD
2	Relay and Motor Control	Input = 1,2: Controls two lights using relays
		Input =4,5: Operates a stepper motor in slow and fast mode through Bluetooth commands.
3	Caesar Cipher Decryption	Receives an encrypted message via Bluetooth and displays the decrypted text on the LCD.

5. Features :

- **Main Features-**

I. Sending Morse Code using Bluetooth:

In this project we have implemented a simple morse code decoder where the user will send “.” and “-” using UART serial communication and the code builds a pattern based on sequence of dots and dashes and then decodes the pattern into a letter when a space and carriage return is received. The Morse mode is selected through sending 1 through the serial communication.

Upon sending 1 the morse mode would be activated and the assembler would jump to the morse subroutine for initializing the morse decoding pattern.

```
MORSE:
        MOV A, #01H
        LCALL COMNWRT
        MOV DPTR, #MSG8
        ACALL LCD_MSG
        ACALL DELAY1S
        LCALL REC_FUNC

        CLR C
                SUBB A, #30h

                MOV DPTR, #bitpattern
                MOVC A, @A+DPTR

                MOV 50h, A

        MOV A, #01H
        LCALL COMNWRT
        LCALL DELAY
        LCALL DELAY
        MOV DPTR, #MSG4
        ACALL LCD_MSG
        ACALL DELAY
        ACALL DELAY
        ACALL DELAY1S
        MOV A, #01H
        ACALL COMNWRT

        MOV pattern, #0
        MOV length,  #0
                MOV R1, #70H
        ; Serial setup for 9600-8-N-1 (assuming 11.0592
MHz)
          MOV TMOD,#20H ; timer 1 mode 2 is selected
                MOV TH1,#0FDH ; baud rate
                MOV SCON,#50H ; serial mode 1 10 bit
total isn, 8db, 1STOPb
                CLR TI ; making TI reg zero
                SETB TR1 ; starting timer 1
        SETB ES
        SETB EA

        ; stay here forever
HERE:     SJMP   HERE
```

In this subroutine we send a msg saying “Morse mode selected” and show the message in the LCD screen. Next the LED is selected which would light up upon receiving a dot or dash pattern. In this subroutine we have set up the initialization for serial interrupt routine by initializing TMOD and TH1. Whenever the serial interrupt is invoked the microcontroller jumps to the serial interrupt vector table and performs the next procedure.

Now for the decoding, whenever the microcontroller receives a dot or dash the serial interrupt (ISR_SERIAL) is triggered. After receiving a “dot” the LED is blinked for 500ms then the Carry Flag is cleared as dot represents 0. The pattern variable carries the information of whether a dot has been received or a dash has been received. When a dot is received the carry flag is cleared and the patterned rotated to left to put the 0 in the least significant bit. Pattern remembers the sequence and keeps to order intact so that the letter can be decoded from the **decodetable** lookup table. For every dot or dash the length variable is incremented. The length variable records how many symbols were received or in other words what is the length for the received morse code so that the letter can be interpreted from the lookup table. Every time a dash or dot is being sent the **LED at pin 1.6** lights up and a **buzzer** gives off an alarm indicating the incoming morse message. Same procedure is performed upon receiving a “dash” as well. The following code keeps track of pattern and length and stores these information to later perform the decoding :

```
ISR_SERIAL:

        CLR   TI
        CLR   RI      ; clear TX flag (if you send in ISR)
        MOV   A, SBUF
        ;ACALL PUT_CHAR    ; get Rx'd ASCII char
        MOV   R3, A    ; save for later

  ;—— handle "."
————————————————————————————
        CJNE   A, #'.', CHECK_DASH
        MOV P0, 50H
                ACALL DELAY500MS
                MOV P0, #0
        ACALL PUT_CHAR
        LCALL BUZZ
        CLR   C
        MOV   A, pattern
        RLC   A
        MOV   pattern, A
        INC   length
        LJMP   ISR_EXIT


  ;—— handle "–"
————————————————————————————
```

```
CHECK_DASH:   CJNE   A, #'-', CHECK_TERM
                MOV P0, 50h
                ACALL DELAY1S
                MOV P0, #0
                ACALL PUT_CHAR
                LCALL BUZZ
        SETB   C
        MOV   A, pattern
        RLC   A
        MOV   pattern, A
        INC   length
        LJMP   ISR_EXIT

  ;—— handle terminators
————————————————————————
   ; space = 20h, CR = 0Dh, LF = 0Ah
CHECK_TERM:   CJNE   A, #' ', NOT_CR
        ACALL PUT_CHAR

        ACALL   DECODE_LETTER
        ;MOV    A, #' '
        ;ACALL   PUT_CHAR
        LJMP   ISR_EXIT
NOT_CR:      CJNE   A, #0Dh, CHECK_END
        ACALL   DECODE_LETTER
        LJMP   ISR_EXIT
```

Now in the decoding part the **DECODE_LETTER** subroutine is called to decode the pattern that has been built so far. In the subroutine if length=0 then the process returns immediately to the previous position and keeps checking. When length is not zero a length offset is taken based on the length from the length offset lookup table. This length offset tells where to start in the decode table depending on how many symbols the code has received. Pattern value is added to the offset to get the final index. From the decode table the letter is fetched at that index and the letter is stored and displayed. In the end of the subroutine the pattern and length is cleared for the next round. The following code is the subroutine for decoding the morse code:

```
DECODE_LETTER:
        MOV   A, length
        JZ    DL_DONE

        ; fetch offset = 1<<length
        MOV   DPTR, #lengthOffset
        MOVC   A, @A+DPTR   ; A = lengthOffset[length]
        MOV   R0, A

        ; compute final index
        MOV   A, pattern
        ADD   A, R0       ; A = idx
```

```
        ; lookup decodeTable[idx]
        MOV   DPTR, #decodeTable
        MOVC   A, @A+DPTR

        ;ACALL   PUT_CHAR
        ACALL SAVE_CHAR

        MOV   pattern, #0
        MOV   length, #0
DL_DONE:
        RET
```

II. Encryption-Decryption(Caesar Cypher):

In this project ,an encryption - decryption mechanism based on the Caesar cipher technique has been implemented using the 8051 Assembly Language. In the Caesar cipher method, each letter in the text is shifted by a fixed number of positions in the alphabet.

In this implementation, the encryption process involves shifting each letter **one position to the left**. As a result, during decryption, the encrypted message is also shifted **one position to the right** to retrieve the original text.

For instance, if the encrypted character 'B' is received, it will be decrypted back to 'A'.

Code Explanation:

```
PROCESS_CIPHER:
                CJNE  A, #'A', PC_NOT_A
        MOV   A, #'Z'
        ACALL PUT_CHAR
        ACALL SAVE_CHAR
        SJMP  ISR_EXIT
PC_NOT_A:
  ; — if below 'A' or above 'Z', treat as terminator
  CLR   C
  SUBB  A, #'A'
  JC    C_CHECK_TERM    ; A < 'A'
  CLR   C
  SUBB  A, #'Z'
  JNC   C_CHECK_TERM    ; A > 'Z'

  ; — now it's B…Z, do the -1 shift
  MOV   A, R3
  CLR   C
  SUBB  A, #1
  ACALL PUT_CHAR
  ACALL SAVE_CHAR
  SJMP  ISR_EXIT
C_CHECK_TERM:
  MOV   A, R3
  CJNE  A, #' ', C_NOT_CR
    ACALL PUT_CHAR
    ACALL LINE
    SJMP ISR_EXIT
C_NOT_CR:
  CJNE  A, #0Dh, ISR_EXIT
  ACALL LINE
  SJMP ISR_EXIT
```

When the user sends the input ‘3’ via Bluetooth during the **mode selection** stage, the **‘Cipher’** mode is activated. Then as we give any letter as input, the **Process_Cypher** interrupt is invoked where the input letters(encrypted) are decrypted using the Caesar-cipher technique. Each character transferred through the UART is handled separately.

At first, the code checks if the received character is ‘A’. If it is, the character is wrapped to 'Z'. Then, the character is displayed by calling the PUT_CHAR subroutine and saved by calling the SAVE_CHAR subroutine.

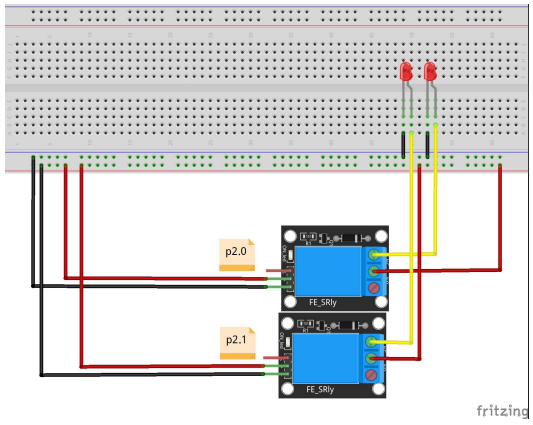
In the **PC_NOT_A** section, the program checks whether the received character falls within the range 'B' to 'Z'. The carry flag is cleared before performing subtraction operations to compare the character with 'A' and 'Z'. If the character is found to be outside this range, it is treated as a terminator and handled separately.

For characters within the valid range, decryption is performed by shifting the character one position to the left. This is achieved by moving the original character stored in register R3 to the accumulator, clearing the carry flag, and subtracting 1 from its ASCII value. The resulting decrypted character is then displayed using the PUT_CHAR subroutine and stored by calling the SAVE_CHAR subroutine.

In the **C_CHECK_TERM** section, the program handles special characters. If the received character is a space (' '), it is displayed normally by calling the PUT_CHAR subroutine, and a new line is started using the LINE subroutine. If the character is a carriage return (0Dh), a new line is inserted to maintain proper formatting of the output. Any other characters that do not match these conditions are ignored, and the program exits the decryption routine.

III. Relay Control:

Relay actual circuit diagram:



Code:

```
NOT_ALL:    CLR P2.0
            CLR P2.1
            SJMP CHECK_1ST
CHECK_1ST:  CJNE A,#'1',CHECK_2ND
            SETB P2.0
            CLR P2.1
            LJMP ISR_EXIT
CHECK_2ND:  CJNE A,#'2', PROCESS_MOTOR
            SETB P2.1
```

```
CLR P2.0
LJMP ISR_EXIT        ; Otherwise, turn OFF LED
```

The relay module allows a low-power device like an MCU, Raspberry Pi, or ESP32 to control higher voltage or current devices (like lamps, motors, etc.).

Here the power is connected to the com port of the relay and LED with the NC (Normally Closed) part of the relay. When 0v is provided to the signal port of the relay the com port connects with the NO (Normally open) pin. So the power flow is cut. When 5v is provided from the P0.0/P0.1 of the MCU the NC is connected with the com port. Then the LED turns on.
The Second relay works the same.

- **Additional Features-**

IV. Speed Control of Stepper Motor :

A 28BYJ-48 stepper motor has been controlled using the 8051 microcontroller , with the help of ULN2003 Darlington motor driver IC as an interface between them.

28BYJ-48 Stepper Motor:
The 28BYJ-48 is a unipolar 5V stepper containing four electromagnetic coils internally.These coils are arranged in such a way that they can be energized individually or in combinations to rotate the motor shaft in small, precise steps.
The motor has an internal **gear reduction ratio of 1/64** and an internal step angle of **5.625° per step.** Hence, it takes 64 steps per revolution internally, resulting in **4096 steps** per output shaft revolution. Since the coils require more current than the 8051 board can safely provide, a ULN2003 driver board is used.

ULN2003A driver:
The ULN2003 allows the low-current control signals from the microcontroller to safely switch the higher current needed by the motor coils, using high gain Darlington transistor pairs.

Working Principle:

- Four control pins (IN1-IN4) are connected to pins (P2.3- P2.6) of the microcontroller board respectively.
- A half-step driving sequence is implemented for smoother rotation and better control.

The control sequence has been shown below:

Step	P2.6	P2.5	P2.4	P2.3	INPUT HEX VALUE	COILS ENERGIZED
1	0	0	0	1	08H	IN1
2	0	0	1	1	18H	IN1+IN2
3	0	0	1	0	10H	IN2
4	0	1	1	0	30H	IN2+IN3
5	0	1	0	0	20H	IN3
6	1	1	0	0	60H	IN3+IN4
7	1	0	0	0	40H	IN4
8	1	0	0	1	48H	IN4+IN1

Here we have specified two modes of operation for the control of stepper motor using Bluetooth. The first mode is initiated by “4” and when given the motor runs at slow speed while the second is initiated by “5” and when given the motor runs at fast speed. When “4” is sent through the bluetooth the initiated delay is large and the motor runs at slow speed. When “5” is sent through bluetooth the initiation delay is smaller and the motor runs at a faster speed. Here we have controlled the motor in full step mode with eight step mode. In full step mode the motor moves one full step for an electrical pulse sent to the coils. In an eight step sequence the motor will have 8 phases and the control is precise in this way. We have used a ULN2003A motor driver for controlling the motor. For each input of “4” or ‘5” one revolution of the motor will be completed.

Code :

```
PROCESS_MOTOR: ;Input 4 for slow spin, 5 for fast spin
        CJNE A, #'4', FAST  ; Check if '4' (slow)
SLOW:           MOV R4,#20
        SJMP ROTATE

FAST:           CJNE A, #'5', NEXT   ; Check if '5' (fast)
                MOV R4,#10
        SJMP ROTATE

ROTATE:
    ; Step 1
    MOV A, #08H
    ACALL OUTPORT
    LCALL DELAY1

    ; Step 2
    MOV A, #018H
    ACALL OUTPORT
    LCALL DELAY1

    ; Step 3
    MOV A, #010H
    ACALL OUTPORT
    LCALL DELAY1

    ; Step 4
    MOV A, #030H
    ACALL OUTPORT
    LCALL DELAY1
```

```
    ; Step 5
    MOV A, #020H
    ACALL OUTPORT
    LCALL DELAY1

    ; Step 6
    MOV A, #060H
    ACALL OUTPORT
    LCALL DELAY1

    ; Step 7
    MOV A, #040H
    ACALL OUTPORT
    LCALL DELAY1

    ; Step 8
    MOV A, #048H
    ACALL OUTPORT
    LCALL DELAY1

; Send value to Port 2
OUTPORT:
    MOV P2, A
    SJMP ISR_EXIT
```

V. Buzzer:

The built-in buzzer of the microcontroller board was utilized for the system. The low (negative) terminal of the buzzer was connected to pin P1.6 of the microcontroller, while the ground terminal was connected to the system ground. The buzzer is activated with a beep each time a dot or dash of the Morse code is received, effectively serving as an alarm system to indicate the transmission of each Morse code symbol.

```
BUZZ:
        SETB P0.6
        ACALL DELAY500MS
        CLR P0.6
        ACALL DELAY500MS
        RET
```

6. Working Principle :

In this project , there are **three modes** of operation.

1. Morse Code mode: Interpret “.” and “-” inputs and decode letters from the code.
2. Relay and Motor Control : Activate relay based on input “1” or “2” and run the stepper motor in slow or fast mode based on “4” or “5”.
3. Cipher Mode: Decrypt letters from encrypted messages.

We have used the Bluetooth HC-05 module for UART serial communication. Through the bluetooth module the user will be sending commands and selecting modes for performing different tasks. In case of different modes the Interrupt Service Routine (ISR) handles the incoming serial characters. However, initially while selecting modes the microcontroller uses polling to continuously monitor the serial pin for input.

Upon reset the microcontroller starts at 000H location and jumps to Start subroutine. In this subroutine the LCD module is initialized for showing output and some messages are shown indicating the program has started.

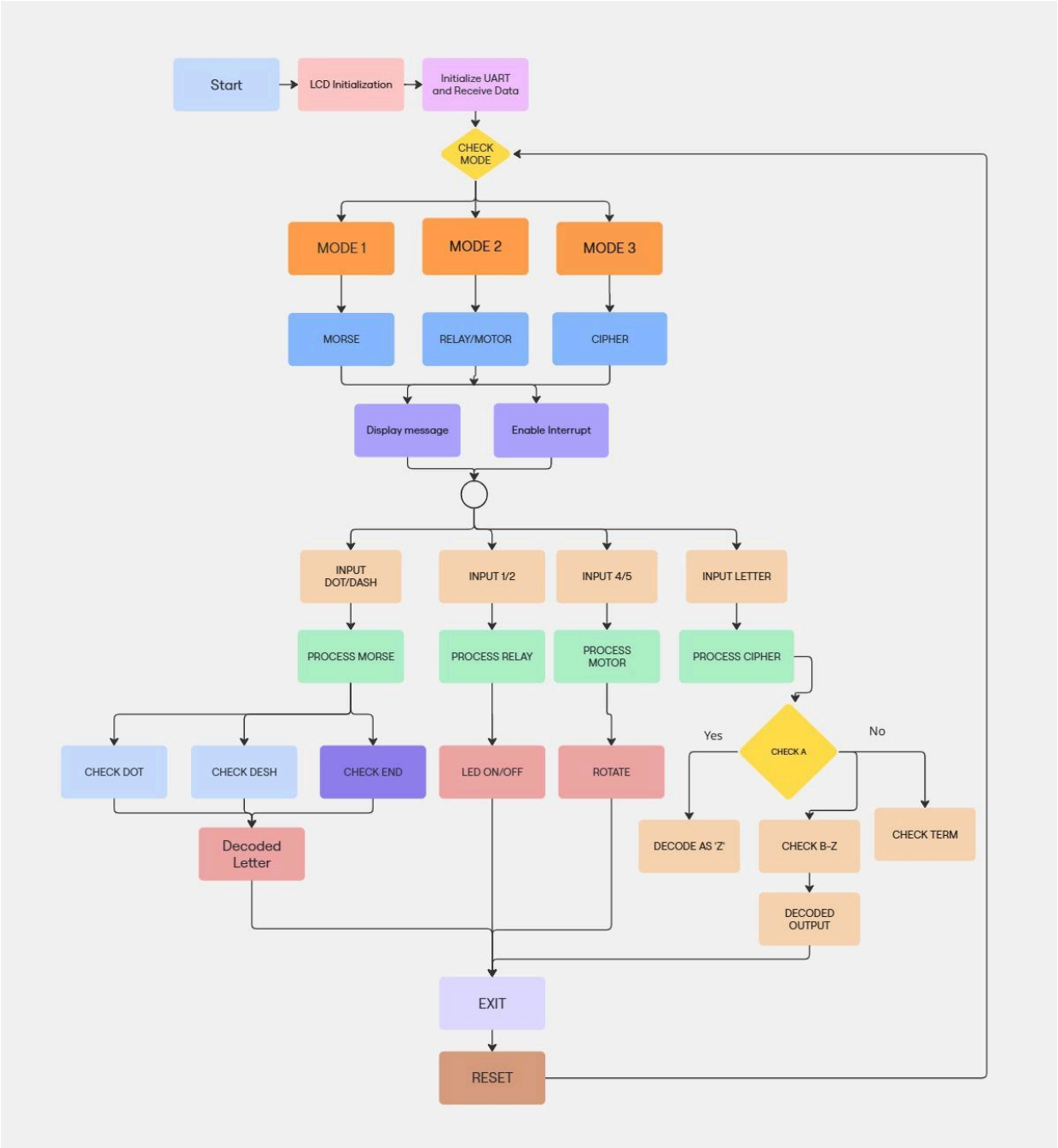
```
ORG    0000h
        LJMP   START
START:

MOV P0, #0H

RS          EQU P2.0
RW          EQU P2.1
E           EQU P2.2

  ;MOV  SP, #90H
  MOV  PSW, #00H

LCD_IN:
  MOV  A, #38H   ;init. LCD 2 lines, 5x7 matrix
  ACALL COMNWRT   ;call command subroutine
  ACALL  DELAY   ;give LCD some time
  MOV   A, #0FH   ;dispplay on, cursor on
  ACALL COMNWRT   ;call command subroutine
  ACALL  DELAY   ;give LCD some time
  MOV  A, #01    ;clear LCD
  ACALL COMNWRT   ;call command subroutine
  ACALL  DELAY   ;give LCD some time
  MOV  A, #06H   ;shift cursor right
  ACALL COMNWRT   ;call command subroutine
  ACALL  DELAY   ;give LCD some time
  MOV  A, #80H   ;cursor at line 1 postion 4
  ACALL COMNWRT   ;call command subroutine
  ACALL  DELAY   ;give LCD some time
        ;MOV    SP, #70h
  MOV DPTR, #MSG1
  LCALL LCD_MSG

  LCALL DELAY

  MOV A, #0C0H
  LCALL COMNWRT
  MOV DPTR, #MSG2
  LCALL LCD_MSG
  LCALL DELAY
  LCALL DELAY
  LCALL DELAY1S
  MOV A, #01H
  ACALL COMNWRT
  MOV DPTR, #MSG3
  LCALL LCD_MSG

   LCALL REC_FUNC
```

After the initialization of LCD the microcontroller jumps to REC_FUNC subroutine where the serial communication is initialized. Here the microcontroller is initialized for 9600 baud rate serial communication and keeps polling the RI register until the user gives any input mode. Whenever the user gives any mode as input the microcontroller comes out of polling and initializes the serial interrupt. After initialization it jumps to corresponding mode according to “1” or “2” or “3” for initialization of the corresponding mode.

```
        CJNE A, #'1', NOT_MODE_1
        SJMP MORSE
        NOT_MODE_1:
        CJNE A, #'2', NOT_MODE_2
        SJMP RELAY_MOTOR
        NOT_MODE_2:
        CJNE A, #'3', NOT_MODE_3
        SJMP CYPHER
        NOT_MODE_3:
        SJMP LCD_IN

REC_FUNC:
        MOV TMOD,#20H ; timer 1 mode 2 is
selected
        MOV TH1,#0FDH ; baud rate
        MOV SCON,#50H ; serial mode 1 10 bit
total isn, 8db, 1STOPb
        CLR TI ; making TI reg zero
        SETB TR1 ; starting timer 1

        ACALL DELAY

        CLR RI ; register involved in receiving
data from bluetooth and ensuring it
        REP: JNB RI, REP
        ACALL DELAY
        ACALL DELAY
        CLR RI

        MOV A,SBUF
        RET
```

In each mode code is written for initialization of the mode and upon entering each mode the microcontroller keeps performing its tasks. Upon receiving a serial interrupt it jumps to Interrupt Service Routine and performs tasks according to different modes along with displaying in LCD.

FLOWCHART:



7. Complete Code :

```
; DATA area
;-------------------------------------------------------
 ;DATA    30h, 31h
pattern       EQU     30h      ; 8-bit shift-accumulator
length        EQU     31h      ; 1–5 symbols received
LCD_DATA      EQU     P1
;-------------------------------------------------------
; VECTOR table
;-------------------------------------------------------
          ORG     0000h
          LJMP    START
          ORG     0023h    ; Serial interrupt vector
ISR_SERIAL:

          CLR     TI
          CLR     RI       ; clear TX flag (if you send in ISR)
          MOV     A, SBUF
          ;ACALL PUT_CHAR     ; get Rx'd ASCII char
          MOV     R3, A    ; save for later

   ;—— handle "."
————————————————————————————
          CJNE    A, #'.', CHECK_DASH
          MOV P0, 50H
            ACALL DELAY500MS
             MOV P0, #0
          ACALL PUT_CHAR
          LCALL BUZZ
          CLR     C
          MOV     A, pattern
          RLC     A
          MOV     pattern, A
          INC     length
          LJMP    ISR_EXIT


   ;—— handle "–"
————————————————————————————
CHECK_DASH:     CJNE    A, #'-', CHECK_TERM
                     MOV P0, 50h
                     ACALL DELAY1S
                     MOV P0, #0
                     ACALL PUT_CHAR
                     LCALL BUZZ
                SETB    C
          MOV     A, pattern
          RLC     A
          MOV     pattern, A
          INC     length
          LJMP    ISR_EXIT

   ;—— handle terminators
————————————————————————————
   ; space = 20h, CR = 0Dh, LF = 0Ah
CHECK_TERM:     CJNE    A, #' ', NOT_CR
          ACALL PUT_CHAR

          ACALL   DECODE_LETTER
          ;MOV     A, #' '
          ;ACALL   PUT_CHAR
          LJMP    ISR_EXIT
NOT_CR:       CJNE    A, #0Dh, CHECK_END
          ACALL   DECODE_LETTER
          LJMP    ISR_EXIT
CHECK_END:     CJNE A, #'0', PROCESS_RELAY
;PROCESS_CYPHER
                    ACALL LINE
                    LJMP ISR_EXIT


PROCESS_RELAY:  ;Input 3-both LEDS, 1,2- one LED

         CJNE A,#'3',NOT_ALL
           SETB P2.0
           SETB P2.1
           LJMP ISR_EXIT

NOT_ALL:            CLR P2.0
                    CLR P2.1
                    SJMP CHECK_1ST
CHECK_1ST:          CJNE A,#'1',CHECK_2ND
                    SETB P2.0
                    CLR P2.1
                    LJMP ISR_EXIT
CHECK_2ND:          CJNE A,#'2', PROCESS_MOTOR
                    SETB P2.1
                    CLR P2.0
                    LJMP ISR_EXIT          ; Otherwise, turn
OFF LED

PROCESS_MOTOR: ;Input 4 for slow spin, 5 for fast spin
          CJNE A, #'4', FAST  ; Check if '4' (slow)
SLOW:               MOV R4,#20
          SJMP ROTATE

FAST:               CJNE A, #'5', NEXT   ; Check if '5' (fast)
                    MOV R4,#10
                  SJMP ROTATE

ROTATE:
     ; Step 1
     MOV A, #08H
     ACALL OUTPORT
     LCALL DELAY1

     ; Step 2
```

```
    MOV A, #018H
    ACALL OUTPORT
    LCALL DELAY1

    ; Step 3
    MOV A, #010H
    ACALL OUTPORT
    LCALL DELAY1

    ; Step 4
    MOV A, #030H
    ACALL OUTPORT
    LCALL DELAY1

    ; Step 5
    MOV A, #020H
    ACALL OUTPORT
    LCALL DELAY1

    ; Step 6
    MOV A, #060H
    ACALL OUTPORT
    LCALL DELAY1

    ; Step 7
    MOV A, #040H
    ACALL OUTPORT
    LCALL DELAY1

    ; Step 8
    MOV A, #048H
    ACALL OUTPORT
    LCALL DELAY1

; Send value to Port 2
OUTPORT:
    MOV P2, A
    SJMP ISR_EXIT

NEXT:  ;CLR RI
    SJMP PROCESS_CIPHER

PROCESS_CIPHER:

        CJNE  A, #'A', PC_NOT_A
        MOV   A, #'Z'
        ACALL PUT_CHAR
        ACALL SAVE_CHAR
        SJMP  ISR_EXIT

PC_NOT_A:
  ; — if below 'A' or above 'Z', treat as terminator
  CLR   C
  SUBB  A, #'A'
  JC    C_CHECK_TERM    ; A < 'A'
  CLR   C
  SUBB  A, #'Z'
  JNC   C_CHECK_TERM    ; A > 'Z'
  ; — now it's B…Z, do the -1 shift
  MOV   A, R3
  CLR   C
  SUBB  A, #1
  ACALL PUT_CHAR
  ACALL SAVE_CHAR
  SJMP  ISR_EXIT

C_CHECK_TERM:
  MOV   A, R3
  CJNE  A, #' ', C_NOT_CR
  ACALL PUT_CHAR
  ACALL LINE
  SJMP ISR_EXIT
C_NOT_CR:
  CJNE  A, #0Dh, ISR_EXIT
  ACALL LINE
  SJMP ISR_EXIT


ISR_EXIT:
        RETI

;-------------------------------------------------------
; DECODE_LETTER:
;   if length=0 ? RET
;   else idx = lengthOffset[length] + pattern
;       A = decodeTable[idx]
;       PUT_CHAR(A)
;       clear pattern & length
;-------------------------------------------------------
DECODE_LETTER:
        MOV    A, length
        JZ     DL_DONE

        ; fetch offset = 1<<length
        MOV    DPTR, #lengthOffset
        MOVC   A, @A+DPTR   ; A = lengthOffset[length]
        MOV    R0, A

        ; compute final index
        MOV    A, pattern
        ADD    A, R0        ; A = idx

        ; lookup decodeTable[idx]
        MOV    DPTR, #decodeTable
        MOVC   A, @A+DPTR

        ;ACALL   PUT_CHAR
        ACALL SAVE_CHAR

        MOV    pattern, #0
        MOV    length, #0
DL_DONE:
        RET

;-------------------------------------------------------
```

```
; lengthOffset: index = length (0…5) ? gives 2^length
;-------------------------------------------------------------



;-------------------------------------------------------------
; UART-putchar subroutine: A?SBUF, wait TI
;-------------------------------------------------------------
PUT_CHAR:
        ACALL DATAWRT
        RET
SAVE_CHAR:
                ;ACALL PUT_CHAR
                MOV @R1, A
                INC R1
                MOV @R1, #0
                RET
LINE:
                MOV R1, #70H
                MOV A, #0C0H
                ACALL COMNWRT
                UP:
                MOV A, @R1
                JZ DOWN
                ACALL DATAWRT
                INC R1
                SJMP UP
                DOWN:
                RET

;-------------------------------------------------------------
; START: basic init, enable serial interrupt
;-------------------------------------------------------------
START:

MOV P0, #0H

RS      EQU P2.0
RW      EQU P2.1
E       EQU P2.2

  ;MOV  SP, #90H
  MOV  PSW, #00H

LCD_IN:
  MOV  A, #38H   ;init. LCD 2 lines, 5x7 matrix
  ACALL COMNWRT   ;call command subroutine
  ACALL  DELAY   ;give LCD some time
  MOV   A, #0FH   ;display on, cursor on
  ACALL COMNWRT   ;call command subroutine
  ACALL  DELAY   ;give LCD some time
  MOV  A, #01   ;clear LCD
  ACALL COMNWRT   ;call command subroutine
  ACALL  DELAY   ;give LCD some time
  MOV  A, #06H   ;shift cursor right
  ACALL COMNWRT   ;call command subroutine
  ACALL  DELAY   ;give LCD some time
  MOV  A, #80H   ;cursor at line 1 postion 4
  ACALL COMNWRT   ;call command subroutine
  ACALL  DELAY   ;give LCD some time
        ;MOV     SP, #70h
MOV DPTR, #MSG1
LCALL LCD_MSG

LCALL DELAY

MOV A, #0C0H
LCALL COMNWRT
MOV DPTR, #MSG2
LCALL LCD_MSG
LCALL DELAY
LCALL DELAY
LCALL DELAY1S
MOV A, #01H
ACALL COMNWRT
MOV DPTR, #MSG3
LCALL LCD_MSG

        LCALL REC_FUNC


                CJNE A, #'1', NOT_MODE_1
                SJMP MORSE
                NOT_MODE_1:
                CJNE A, #'2', NOT_MODE_2
                SJMP RELAY_MOTOR
                NOT_MODE_2:
                CJNE A, #'3', NOT_MODE_3
                SJMP CYPHER
                NOT_MODE_3:
                SJMP LCD_IN

MORSE:

        MOV A, #01H
        LCALL COMNWRT
        MOV DPTR, #MSG8
        ACALL LCD_MSG
        ACALL DELAY1S
        LCALL REC_FUNC

        CLR C
                SUBB A, #30h

                MOV DPTR, #bitpattern
                MOVC A, @A+DPTR

                MOV 50h, A



        MOV A, #01H
        LCALL COMNWRT
        LCALL DELAY
```

```
        LCALL DELAY
        MOV DPTR, #MSG4
        ACALL LCD_MSG
        ACALL DELAY
        ACALL DELAY
        ACALL DELAY1S
        MOV A, #01H
        ACALL COMNWRT

        MOV pattern, #0
        MOV length,  #0
                MOV R1, #70H
        ; Serial setup for 9600-8-N-1 (assuming 11.0592
MHz)
         MOV TMOD,#20H ; timer 1 mode 2 is selected
                MOV TH1,#0FDH ; baud rate
                MOV SCON,#50H ; serial mode 1 10 bit
total isn, 8db, 1STOPb
                CLR TI ; making TI reg zero
                SETB TR1 ; starting timer 1
        SETB ES
        SETB EA

        ; stay here forever
HERE:       SJMP    HERE

RELAY_MOTOR:
          MOV A, #01H
          LCALL COMNWRT
          LCALL DELAY
          MOV DPTR, #MSG5
          ACALL LCD_MSG
          ACALL DELAY
        ACALL DELAY
        ACALL DELAY1S
        MOV A, #01H
        ACALL COMNWRT

        ;Process Initialization in ISR:
        MOV P2, #00H

        ;Re-trigger
        MOV TMOD,#20H ; timer 1 mode 2 is selected
                MOV TH1,#0FDH ; baud rate
                MOV SCON,#50H ; serial mode 1 10 bit
total isn, 8db, 1STOPb
                CLR TI ; making TI reg zero
                SETB TR1 ; starting timer 1
        SETB    ES
        SETB    EA

                SJMP $

CYPHER:
                MOV A, #01H
                LCALL COMNWRT
                LCALL DELAY
                MOV DPTR, #MSG6
                ACALL LCD_MSG
                ACALL DELAY
        ACALL DELAY
        ACALL DELAY1S
        MOV A, #01H
        ACALL COMNWRT

        ;Process Initialization in ISR:



        ;Re-trigger
        MOV TMOD,#20H ; timer 1 mode 2 is selected
                MOV TH1,#0FDH ; baud rate
                MOV SCON,#50H ; serial mode 1 10 bit
total isn, 8db, 1STOPb
                CLR TI ; making TI reg zero
                SETB TR1 ; starting timer 1
        SETB    ES
        SETB    EA


                SJMP $

LCD_MSG:
    CLR     A           ; Clear A just in case
NEXT_CHAR:
    MOVC    A, @A+DPTR    ; Load character from code
memory using DPTR
    JZ      END_PRINT     ; If A == 0, end of message
    LCALL   DATAWRT       ; Display the character on LCD
    INC     DPTR
    CLR A         ; Move to next character
    SJMP    NEXT_CHAR     ; Repeat
END_PRINT:
    RET



COMNWRT:
  LCALL READY   ;send command to LCD
  MOV  P1, A    ;copy reg A to port 1
  CLR  RS    ;RS=0 for command
  CLR  RW    ;R/W=0 for write
  SETB  E    ;E-1 for high pulse
  ACALL DELAY   ;give LCD some time
  CLR  E    ;E=0 for H-to-L pulse
  RET


DATAWRT:
  LCALL READY   ;write data to LCD
  MOV  P1, A    ;copy reg A to port1
  SETB  RS    ;RS=1 for data
  CLR  RW    ;R/W=0 for write
  SETB  E    ;E=1 for high pulse
  LCALL DELAY   ;give LCD some time
  CLR  E    ;E=0 for H-to-L pulse
```

```
  RET

READY:
  SETB  P1.7
  CLR  RS
  SETB  RW
WAIT:
  CLR  E
  LCALL DELAY
  SETB  E
  JB  P1.7, WAIT
  RET

DELAY:  MOV  R3, #50   ;50 or higher for fast CPUs
HERE2:  MOV  R4, #255   ;R4=255
HERE3:   DJNZ  R4, HERE3   ;stay untill R4 becomes 0
      DJNZ   R3, HERE2
      RET

BUZZ:
            SETB P0.6
            ACALL DELAY500MS
            CLR P0.6
            ACALL DELAY500MS
            RET

REC_FUNC:
          MOV TMOD,#20H ; timer 1 mode 2 is selected
                      MOV TH1,#0FDH ; baud rate
                      MOV SCON,#50H ; serial mode 1 10 bit
total isn, 8db, 1STOPb
                      CLR TI ; making TI reg zero
                      SETB TR1 ; starting timer 1

                      ACALL DELAY

                      CLR RI ; register involved in receiving
data from bluetooth and ensuring it
                      REP: JNB RI, REP
                      ACALL DELAY
                      ACALL DELAY
                      CLR RI

                      MOV A,SBUF
                      RET

DELAY1:
   MOV B,R4
   MOV R3, B
D1: MOV R2, #255
D2: MOV R1, #255
D3: DJNZ R1, D3
   DJNZ R2, D2
   DJNZ R3, D1
   RET


DELAY1S:
   MOV R3, #20      ; outer loop (20x)
DELAY_LOOP1:
   MOV R2, #250     ; middle loop (250x)
DELAY_LOOP2:
   MOV R5, #200     ; inner loop (200x)
DELAY_LOOP3:
   DJNZ R5, DELAY_LOOP3
   DJNZ R2, DELAY_LOOP2
   DJNZ R3, DELAY_LOOP1
   RET
; DELAY500MS: Creates approximately 500ms delay at 12 MHz
DELAY500MS:
   MOV R3, #10      ; Outer loop - 10 times
LOOP1:
   MOV R2, #250     ; Middle loop - 250 times
LOOP2:
   MOV R5, #200     ; Inner loop - 200 times
LOOP3:
   DJNZ R5, LOOP3    ; Inner loop
   DJNZ R2, LOOP2    ; Middle loop
   DJNZ R3, LOOP1    ; Outer loop
   RET


ORG    0300h
lengthOffset:  DB     0, 2, 4, 8, 16, 32

;-------------------------------------------------------------
; decodeTable: 64 entries, idx = (1<<len)+pattern, for len=1…5,
pattern<2^len
;-------------------------------------------------------------
decodeTable:
   ; 0–1 unused
   DB  0, 0
   ; len=1 (idx=2–3): E, T
   DB  'E','T'
   ; len=2 (4–7): I, A, N, M
   DB  'I','A','N','M'
   ; len=3 (8–15): S, U, R, W, D, K, G, O
   DB  'S','U','R','W','D','K','G','O'
   ; len=4 (16–31): H, V, F, –, L, –, P, J, B, X, C, Y, Z, Q, –, –
   DB  'H','V','F',0,'L',0,'P','J'
   DB  'B','X','C','Y','Z','Q',0,0
   ; len=5 (32–63): digits & prosigns (fill as needed)
   DB  '5','4',0,'3',0,'2',0,'1'
   DB  '6',0,0,'7',0,'8','9','0'
   ; rest = 0
   DB  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
   DB  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

  ORG 400H      ; You can start at any ROM location that
doesn't conflict
MSG1:   DB 'BT INTERFACING', 0
MSG2:   DB 'Group C2-G4', 0
MSG3:   DB 'Select a mode', 0
MSG4:   DB 'Morse mode selected', 0
MSG5:   DB 'Relay & Motor selected', 0
MSG6:   DB 'Cypher selected', 0
```

```
MSG8:   DB 'Select LED', 0

ORG 500H
```

```
bitpattern:
DB 1H, 2H, 4H, 8H, 10H, 20H, 40H, 80H

END
```

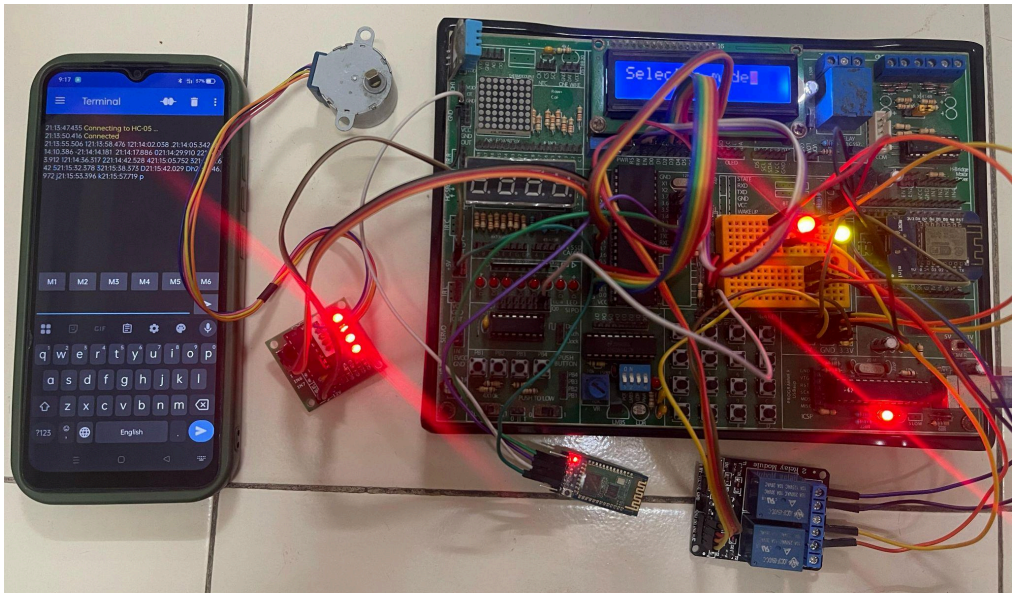
8. Hardware Implementation :

Pinout Table for Hardware :

P0.0	LCD
P0.1	
P0.2	
P0.3	
P0.4	
P0.5	
P0.6	
P0.7	
P1.0	N/A
P1.1	LED
P1.2	N/A
P1.3	N/A
P1.4	N/A
P1.5	N/A
P1.6	Buzzer
P1.7	N/A
P2.0	RELAY-1
P2.1	RLEAY-2
P2.2	N/A
P2.3	

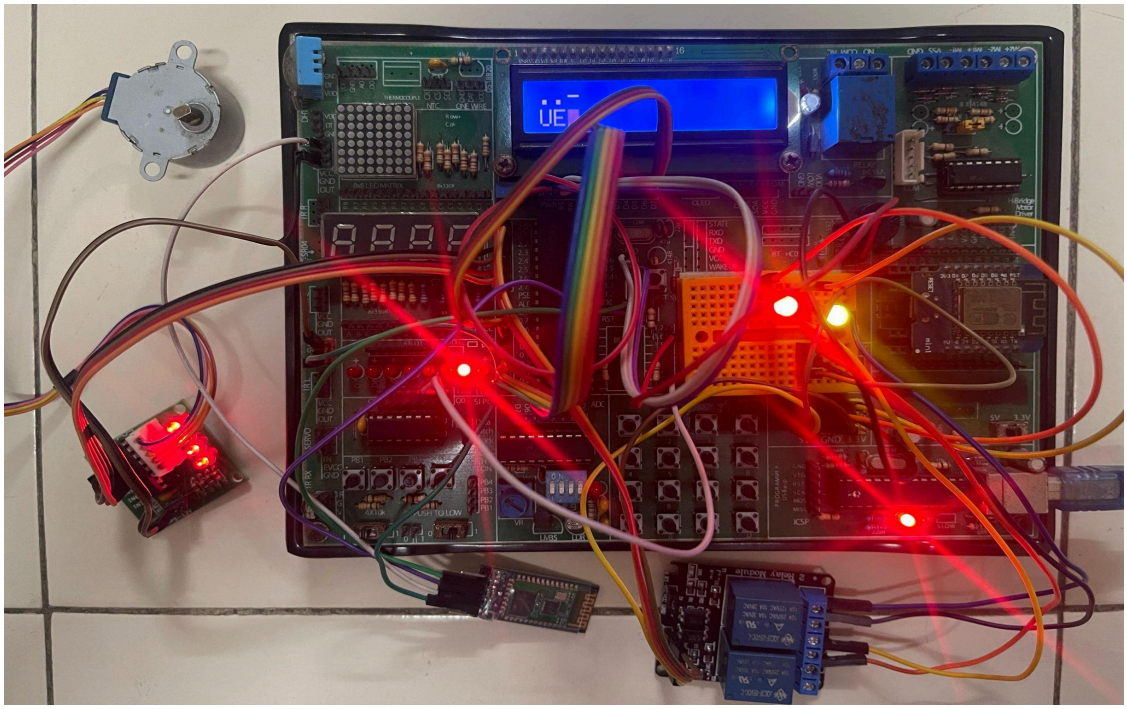
P2.4	MOTOR DRIVER
P2.5	
P2.6	
P2.7	N/A
P3.0	RXD
P3.1	TXD
P3.2	N/A
P3.3	N/A
P3.4	RS
P3.5	RW
P3.6	ENABLE
P3.7	N/A

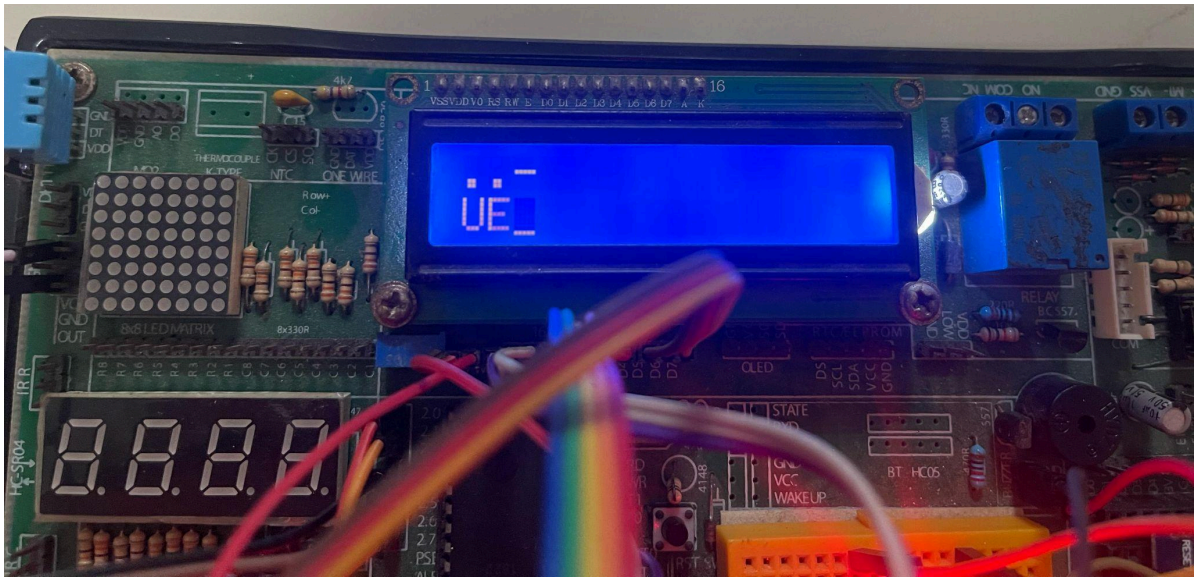
Complete Hardware Setup :



Mode 1:

Sending Morse

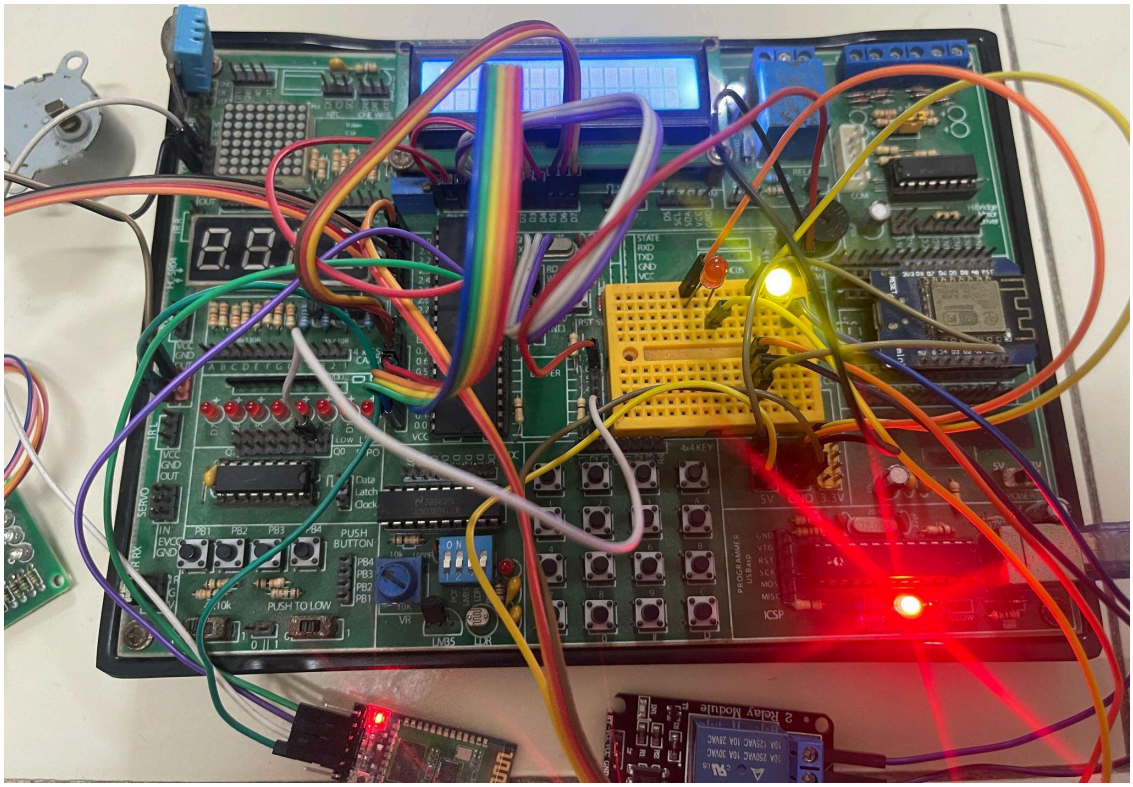




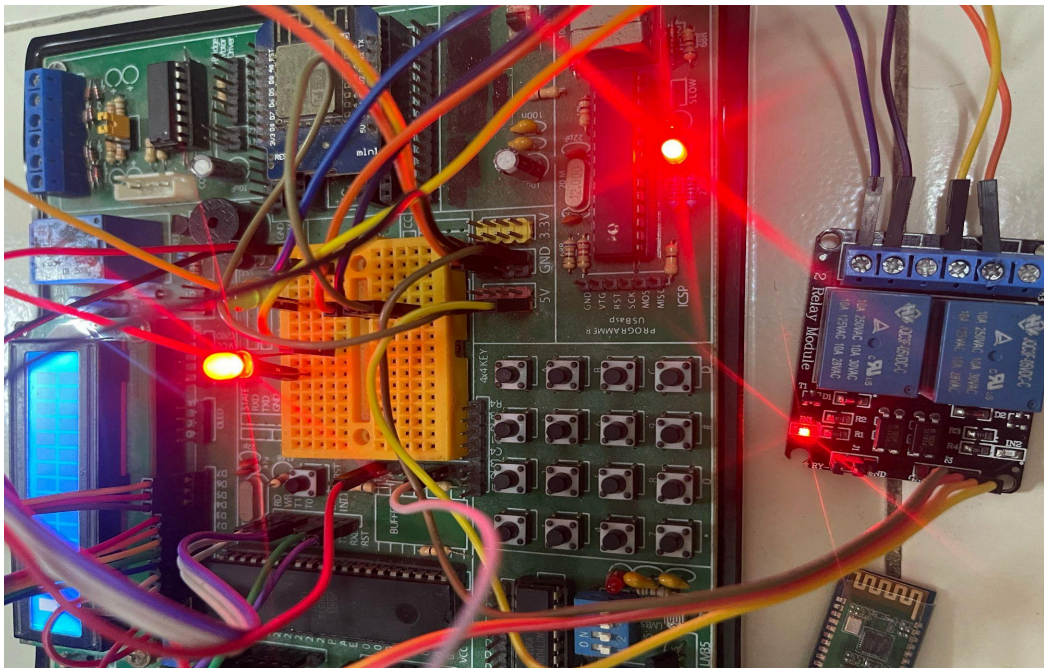
Mode 2:

Relay Control :

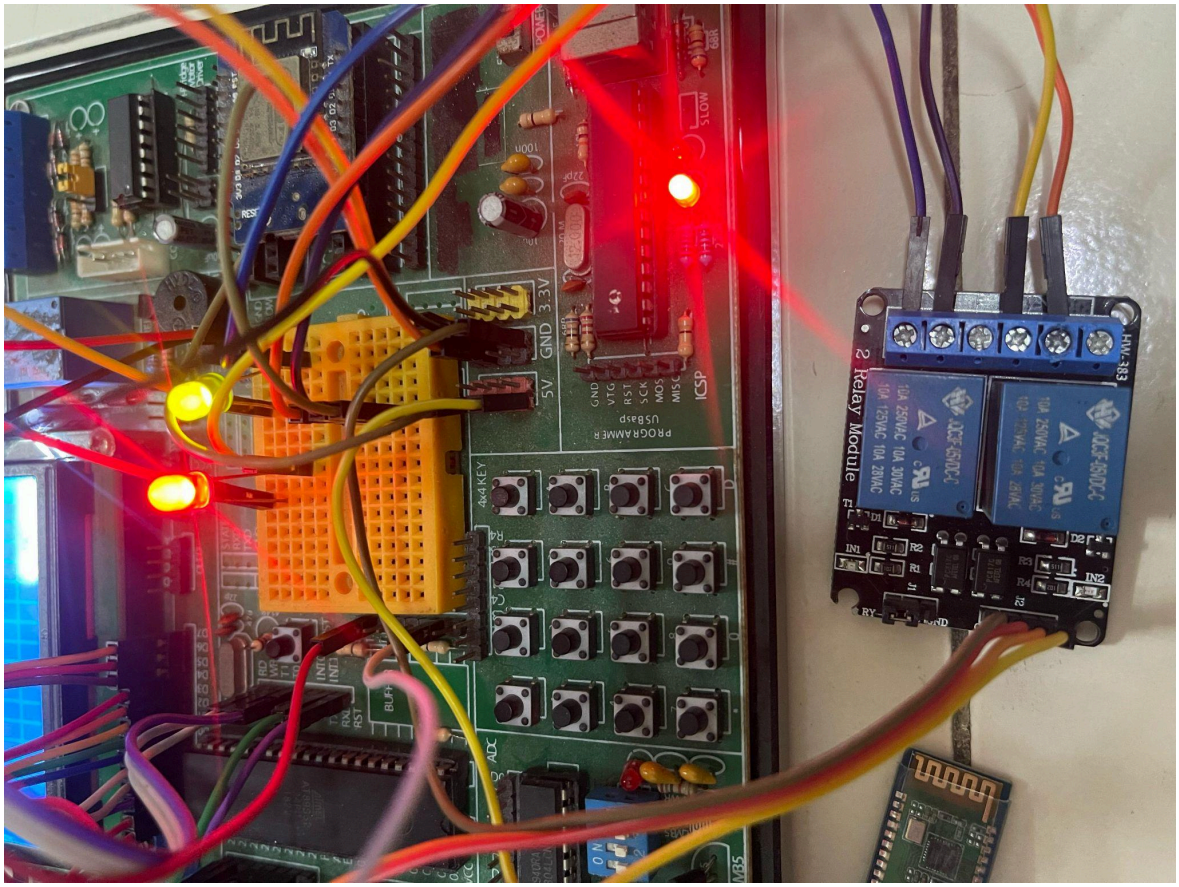
Input 1:



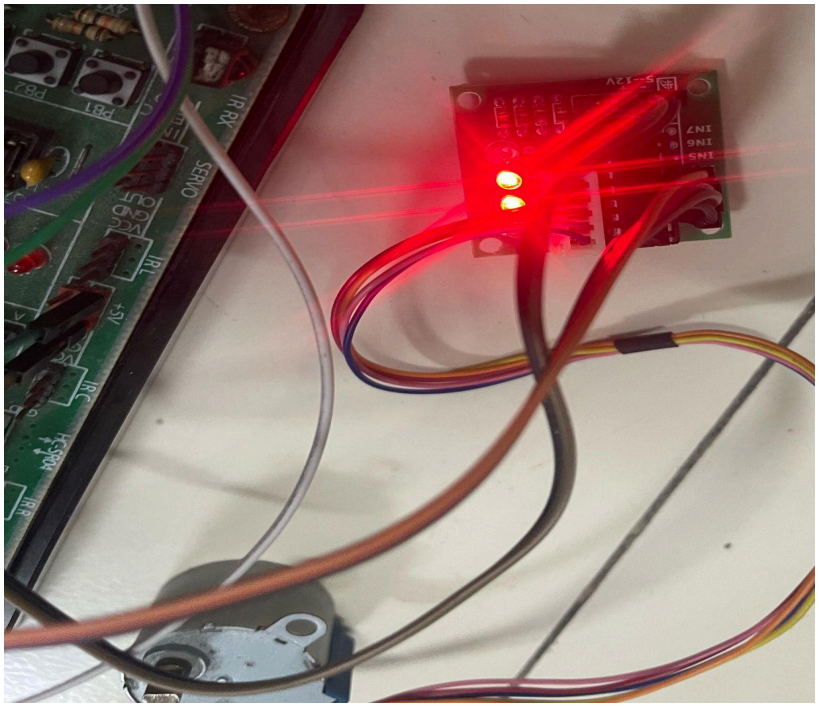
Input 2:

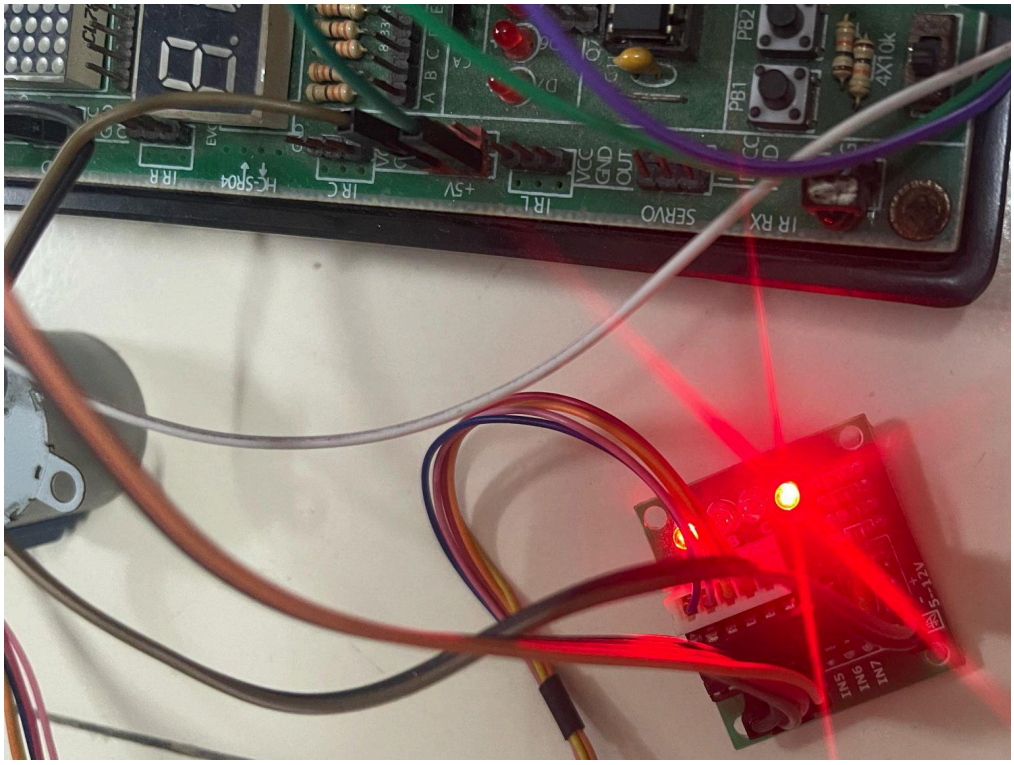


Input 3:

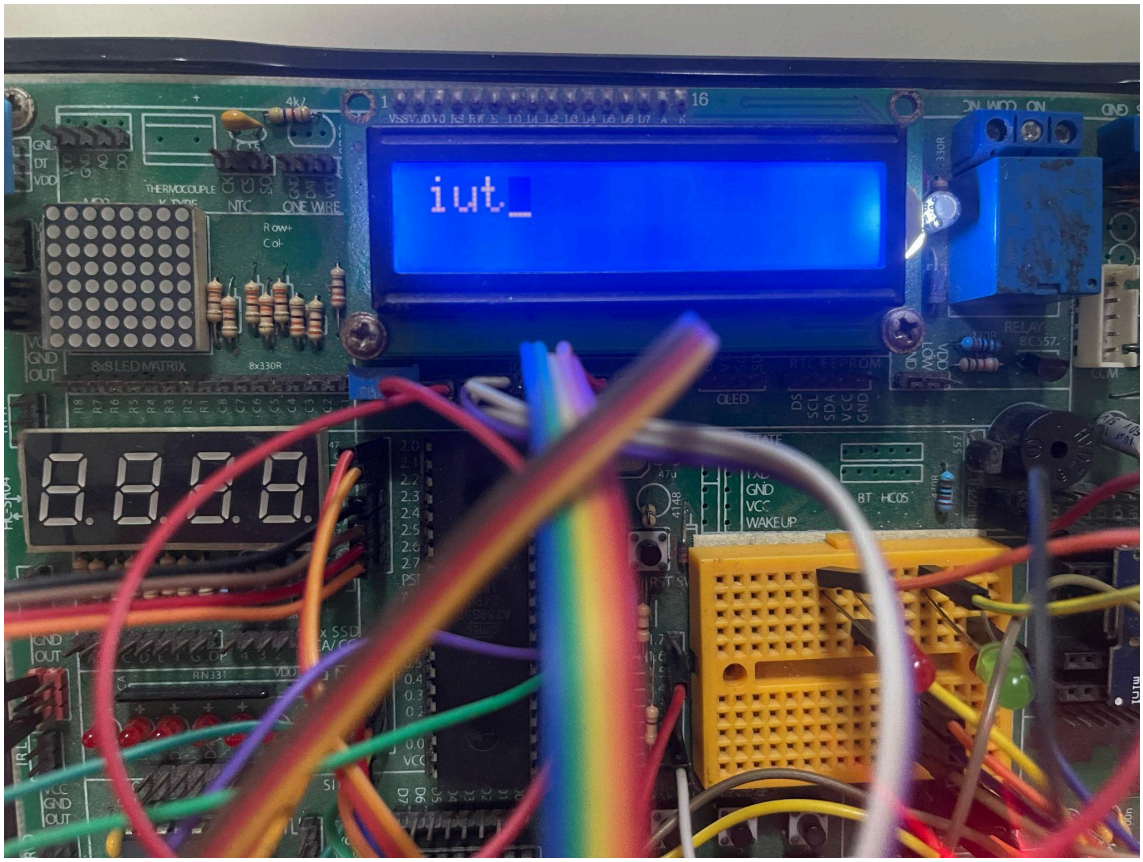


Stepper Motor Control :





Mode 3: Caesar Cipher



9.Problems Faced:

1. Since Proteus does not provide a compatible Bluetooth module, we had to utilize the Virtual Terminal (Serial Monitor) in Proteus as an alternative medium for UART communication.

2. The LCD was not functioning properly when connected to Port 0 in Proteus and Port 1 of the microcontroller board during hardware implementation. This required troubleshooting and adjustment of connections.

3. While sending mode selections via the Bluetooth module, we encountered difficulties in keeping the modes distinct, as continuous inputs were being received and the system was unable to properly differentiate between them.

4. As the motor gear was not receiving enough power from the microcontroller, it was not rotating. Hence, we had to monitor the motor driver's LEDs to check the stepping sequence of the motor.

10. Conclusion :

In this project we successfully integrated the serial communication with 8051 microcontroller and performed multiple different tasks separated by different modes. The multi functional system is capable of interpreting Morse Code inputs, controlling relay and stepper motor operations and performing simple Caesar cipher encryption based on real time UART communication. Using serial interrupts made the code efficient and the input handling smooth allowing the microcontroller to respond promptly to user commands. Additionally the integration of the LCD module provided an interface for real time display of information. The project demonstrated the versatility of 8051 microcontroller in handling multiple tasks simultaneously.