AMRITA VISHWA VIDYAPEETHAM

DEEMED TO BE UNIVERSITY UNDER SECTION 3 OF THE UGC ACT, 1956

AMRITA SCHOOL OF ENGINEERING, BENGALURU

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

B. TECH. IV SEMESTER AY 2024-2025

23ECE284 MICROCONTROLLER AND INTERFACING LABORATORY

SYSTEM DESIGN REPORT

JANUARY TO APRIL 2025

MORSE CODE GENERATOR USING LPC2148

Submitted by

REGISTER NUMBER	NAME	SIGNATURE WITH DATE
BL.EN.U4ECE23208	D.S.V.Sainath Reddy	
BL.EN.U4ECE23213	G.Srishanth	
BL.EN.U4ECE23223	L.Mokshagna	
BL.EN.U4ECE23249	S.Sahithi reddy	

BATCH NUMBER:02

FACULTY EXAMINER SIGNATURE WITH EVALUATION DATE

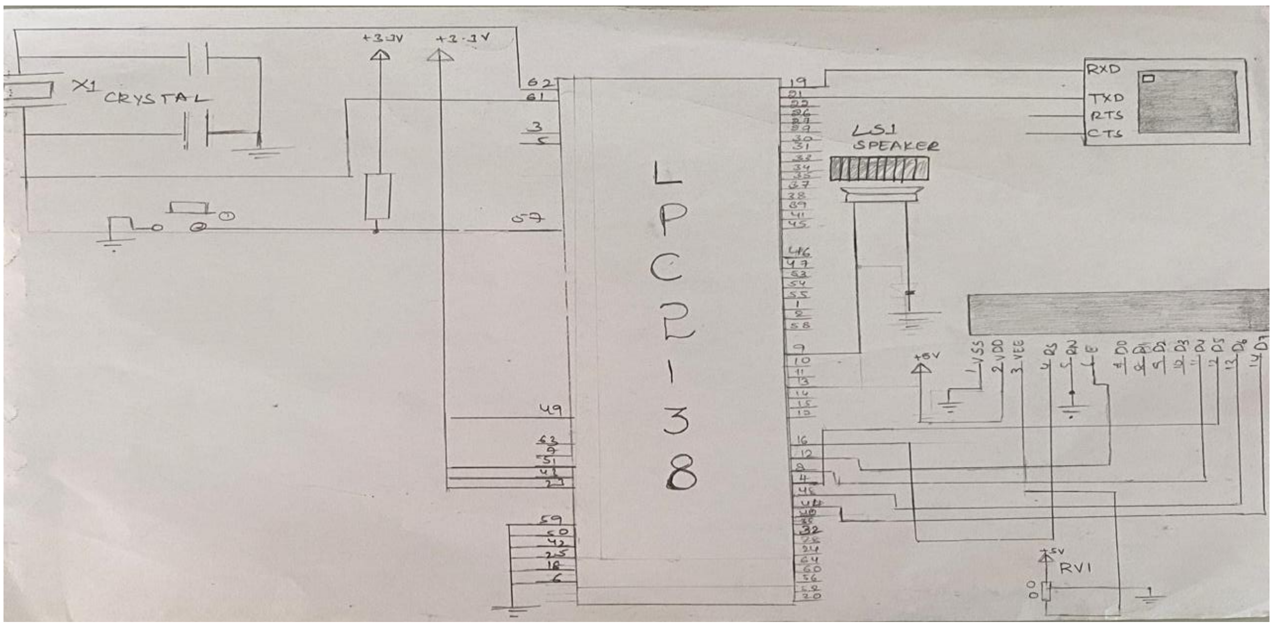
Aim:

To design and implement a Morse Code Generator using LPC2148

Components required along with their specifications:

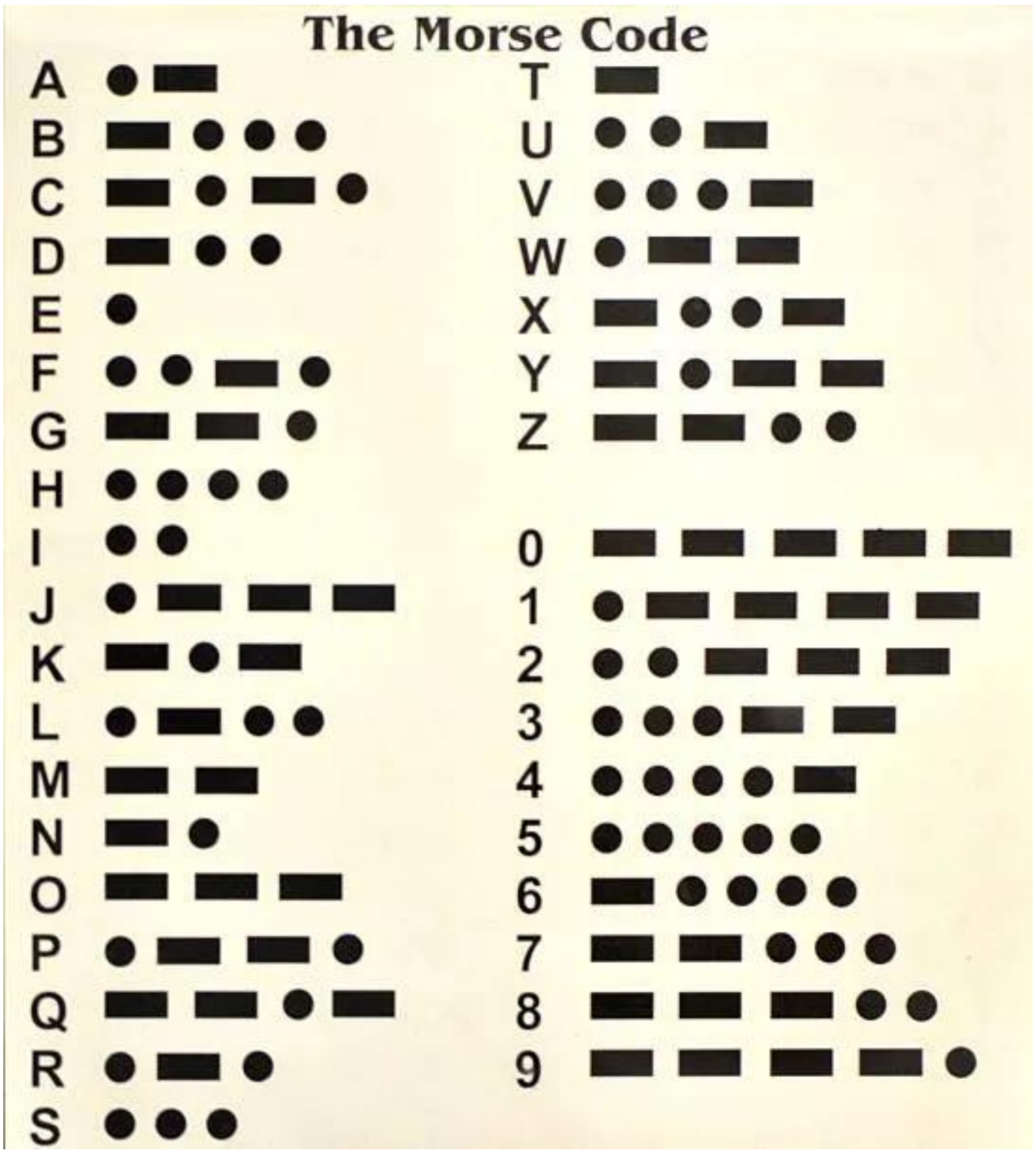
SL.NO	COMPONENT NAME	SPECIFICATION
1	Microcontroller	LPC2138 ARM7TDMI
2	Crystal Oscillator	12 MHz Crystal (X1)
3	Capacitor	22pF (for crystal stabilization)
4	Resistor	10kΩ (pull-up), 1kΩ (general use)
5	Push Button Switch	Tactile, Single Pole Single Throw
6	Speaker	General purpose 8Ω speaker
7	LCD	LM016L 16x2 Character LCD
8	Potentiometer	1kΩ Variable Resistor (RV1)
9	Virtual Terminal	UART Serial Communication Terminal
10	Power Supply	+5V and +3.3V DC
11	Ground	0V
12	Connecting Wires	Single strand jumper wires

Circuit diagram:



Theory and working principle:

The LPC2148 receives character input via UART from a virtual terminal. Each valid character is converted to uppercase and matched with a predefined Morse code from lookup tables for alphabets and numbers. The corresponding Morse code is output through a speaker by toggling P0.25 using Timer0 configured for approximately 1kHz tone generation. Dots and dashes are produced by varying the beep duration (100 ms for dot, 300 ms for dash). Simultaneously, the LCD displays the entered character and its Morse equivalent. Invalid characters are flagged with an "Invalid Input" message. Spaces between words and characters are managed through precise delay intervals, following Morse protocol standards.



The Morse Code

A	● ▬	T	▬
B	▬ ● ● ●	U	● ● ▬
C	▬ ● ▬ ●	V	● ● ● ▬
D	▬ ● ●	W	● ▬ ▬
E	●	X	▬ ● ● ▬
F	● ● ▬ ●	Y	▬ ● ▬ ▬
G	▬ ▬ ●	Z	▬ ▬ ● ●
H	● ● ● ●		
I	● ●	0	▬ ▬ ▬ ▬ ▬
J	● ▬ ▬ ▬	1	● ▬ ▬ ▬ ▬
K	▬ ● ▬	2	● ● ▬ ▬ ▬
L	● ▬ ● ●	3	● ● ● ▬ ▬
M	▬ ▬	4	● ● ● ● ▬
N	▬ ●	5	● ● ● ● ●
O	▬ ▬ ▬	6	▬ ● ● ● ●
P	● ▬ ▬ ●	7	▬ ▬ ● ● ●
Q	▬ ▬ ● ▬	8	▬ ▬ ▬ ● ●
R	● ▬ ●	9	▬ ▬ ▬ ▬ ●
S	● ● ●		

Program:

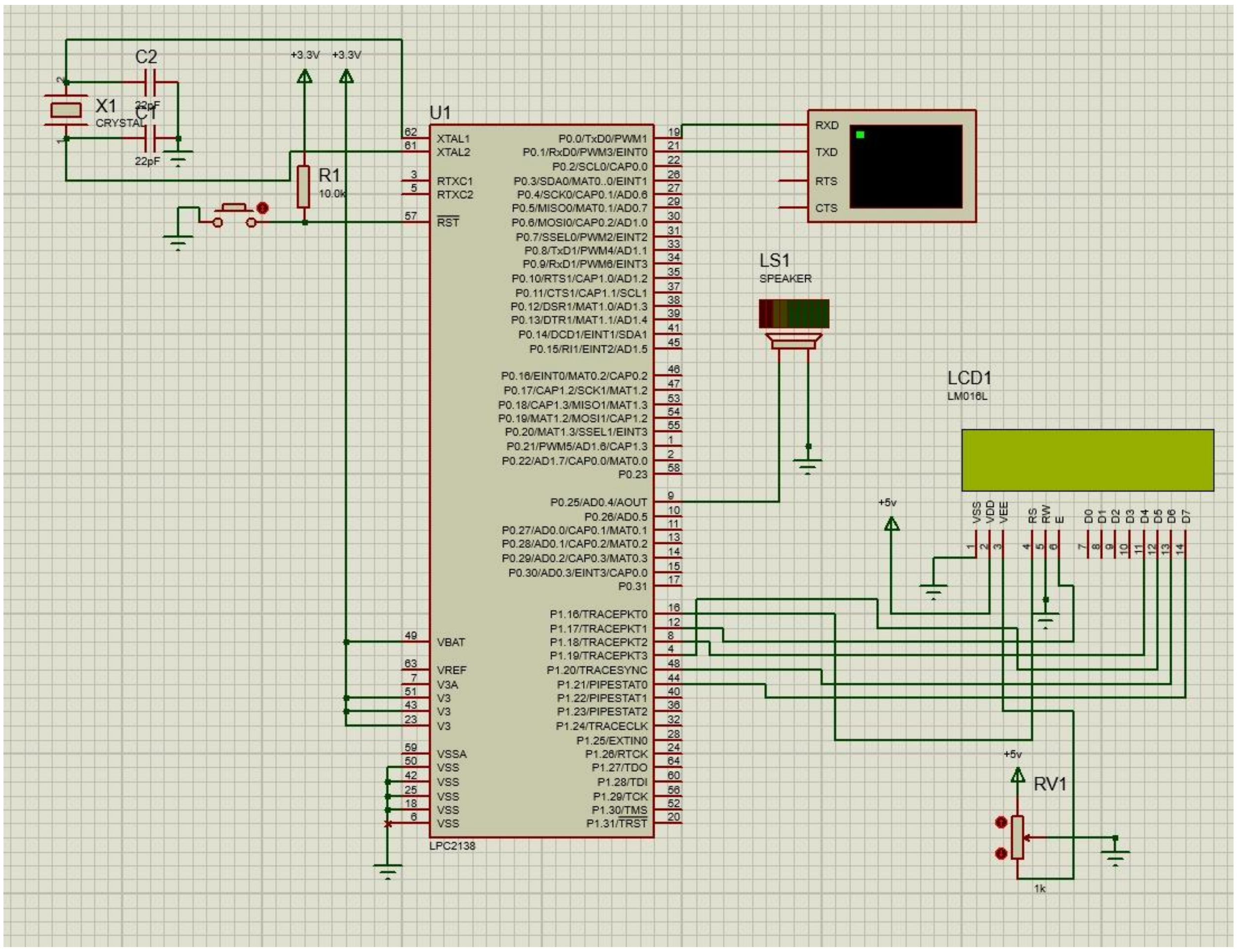
```
#include <lpc214x.h>
#include <string.h>
// Delay function (~1ms delay)
void delay_ms(unsigned int ms) {
    unsigned int i, j;
    for (i = 0; i < ms; i++)
        for (j = 0; j < 6000; j++);
}
// Define Speaker pin (P0.25)
#define SPEAKER (1 << 25)
// UART0 Initialization
void uart0_init() {
    PINSEL0 |= 0x00000005;  // Enable TxD0 and RxD0
    U0LCR = 0x83;           // Enable DLAB and set 8-bit character
    U0DLM = 0x00;
    U0DLL = 78;             // 9600 baud rate
    U0LCR = 0x03;           // Disable DLAB
}
// UART0 Receive
char uart0_getchar() {
    while (!(U0LSR & 0x01));
    return U0RBR;
}
// LCD Macros and Pins (4-bit mode, P1.18 - P1.21)
#define LCD_DATA_MASK 0x003F0000
#define RS (1 << 16)
#define EN (1 << 17)
void lcd_strobe() {
    IOSET1 = EN;
    delay_ms(1);
    IOCLR1 = EN;
    delay_ms(1);
}
void lcd_cmd(unsigned char cmd) {
    IOCLR1 = LCD_DATA_MASK;
    IOCLR1 = RS;
    IOCLR1 = 0x003C0000;
    IOSET1 = ((cmd >> 4) & 0x0F) << 18;
    lcd_strobe();
    IOCLR1 = 0x003C0000;
    IOSET1 = (cmd & 0x0F) << 18;
    lcd_strobe();
    delay_ms(2);
}
void lcd_data(unsigned char data) {
    IOCLR1 = LCD_DATA_MASK;
    IOSET1 = RS;
    IOCLR1 = 0x003C0000;
    IOSET1 = ((data >> 4) & 0x0F) << 18;
    lcd_strobe();
    IOCLR1 = 0x003C0000;
    IOSET1 = (data & 0x0F) << 18;
    lcd_strobe();
    delay_ms(2);
}
```

```
void lcd_init() {
    IO1DIR |= LCD_DATA_MASK;
    delay_ms(20);
    lcd_cmd(0x02);
    lcd_cmd(0x28);
    lcd_cmd(0x0C);
    lcd_cmd(0x06);
    lcd_cmd(0x01);
}
void lcd_string(char *str) {
    while (*str) {
        lcd_data(*str++);
    }
}
// Morse Lookup Tables
const char* morse_table_alpha[26] = {
    ".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....", "..", ".---",
    "-.-", ".-..", "--", "-.", "---", ".--.", "--.-", ".-.", "...", "-",
    "..-", "...-", ".--", "-..-", "-.--", "--.."
};
const char* morse_table_number[10] = {
    "-----", ".----", "..---", "...--", "....-", ".....",
    "-....", "--...", "---..", "----."
};

// Initialize Timer0 for ~1kHz tone generation
void timer0_init() {
    T0CTCR = 0x0;
    T0PR = 0;
    T0MR0 = 6000;  // Match for ~1kHz
    T0MCR = 3;
    T0TCR = 1;
}

// Toggle speaker output using Timer0
void tone_on() {
    IOSET0 = SPEAKER;
    while (!(T0IR & 0x01));
    T0IR = 1;
    IOCLR0 = SPEAKER;
    while (!(T0IR & 0x01));
    T0IR = 1;
}

// Generate tone for duration_ms
void beep(unsigned int duration_ms) {
    unsigned int t;
    for (t = 0; t < duration_ms; t += 2) {
        tone_on();  // Approx 2ms per cycle
    }
}
// Play Morse code sequence
void play_morse(const char* code) {
    while (*code) {
        if (*code == '.') {
            beep(100); // Dot
        } else if (*code == '-') {
            beep(300); // Dash
        }
        delay_ms(200);  // Element gap
        code++;
    }
    delay_ms(400);  // Character gap
}
// Read full word via UART with display
void uart0_getstring_with_display(char* buffer, int max_len) {
    char ch;
    int i = 0;
    int j;
    int start;
    lcd_cmd(0x01);               // Clear LCD
    lcd_string("Enter:");
    lcd_cmd(0xC0);               // Move to 2nd line
    while (i < max_len - 1) {
        ch = uart0_getchar();

        if (ch == '\r' || ch == '\n') {
            break;
        }
        buffer[i++] = ch;
        buffer[i] = '\0';
        start = (i <= 32) ? 0 : i - 32;
        lcd_cmd(0x01);  // Clear display
        // First line
        lcd_cmd(0x80);
        for (j = 0; j < 16 && buffer[start + j] != '\0'; j++) {
            lcd_data(buffer[start + j]);
        }
        // Second line
        lcd_cmd(0xC0);
        for (j = 16; j < 32 && buffer[start + j] != '\0'; j++) {
            lcd_data(buffer[start + j]);
        }
    }

    buffer[i] = '\0';  // Null-terminate
}
// Main function
int main() {
    char input[64];
    int i;
    char c;
    const char* morse;
    // Speaker pin setup
    PINSEL1 &= ~(3 << 18);
    IO0DIR |= SPEAKER;
    lcd_init();
    uart0_init();
    timer0_init();
    lcd_string("Morse Generator");
    delay_ms(1500);
    lcd_cmd(0x01);
    while (1) {
        lcd_cmd(0x01);
        lcd_string("Enter Word:");
        lcd_cmd(0xC0);
        uart0_getstring_with_display(input, 64);
        delay_ms(1000);
        lcd_cmd(0x01);
        for (i = 0; input[i] != '\0'; i++) {
            c = input[i];
            if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <= '9')) {
                if (c >= 'a' && c <= 'z') c -= 32;
                lcd_cmd(0x01);
                lcd_data(c);
                lcd_cmd(0xC0);
                if (c >= 'A' && c <= 'Z') {
                    morse = morse_table_alpha[c - 'A'];
                } else {
                    morse = morse_table_number[c - '0'];
                }
                lcd_string((char*)morse);
                play_morse(morse);
            } else if (c == ' ') {
                delay_ms(600);  // Word gap
            } else {
                lcd_cmd(0x01);
                lcd_string("Invalid Input");
                delay_ms(1000);
            }
        }

        delay_ms(1000);
        lcd_cmd(0x01);
    }
}
```

Simulator used for simulator:

- proetus
- keil

Screenshot of simulator-based circuit implementation:



Inputs applied to the simulator:

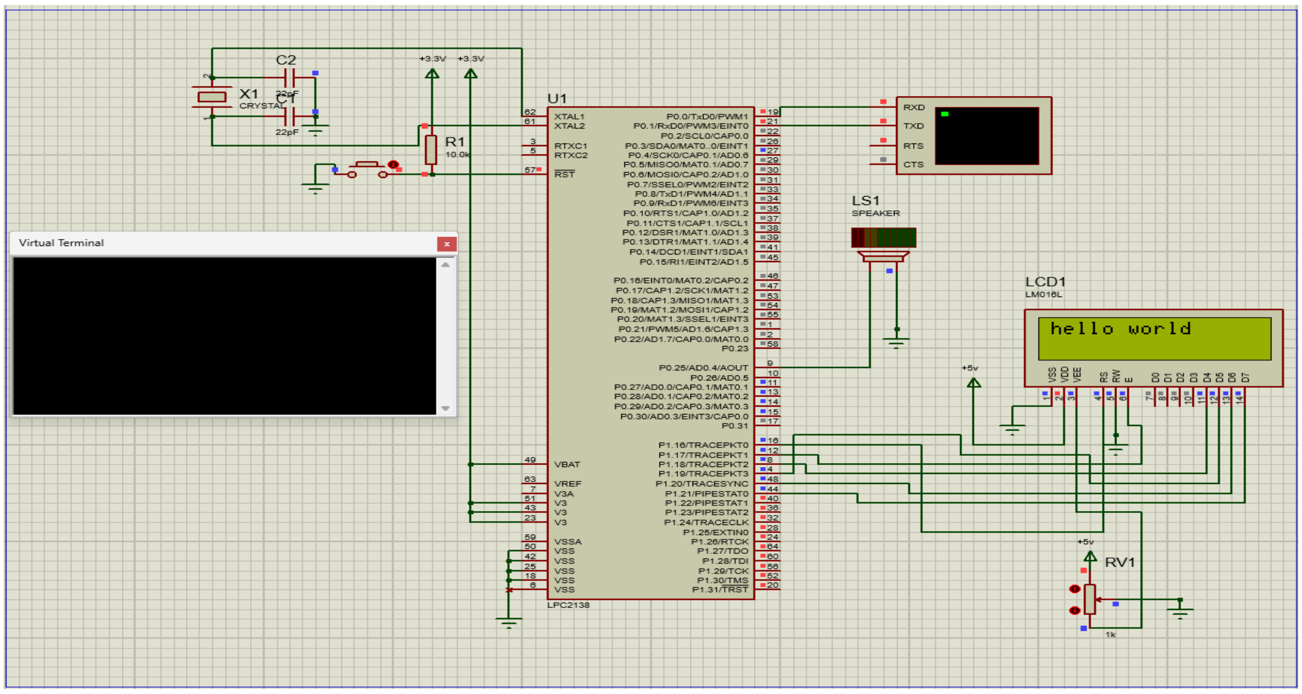
- "HELLO WORLD"
- "ARM7 LPC2148"
- "SOS"
-

Output obtained from the simulator:

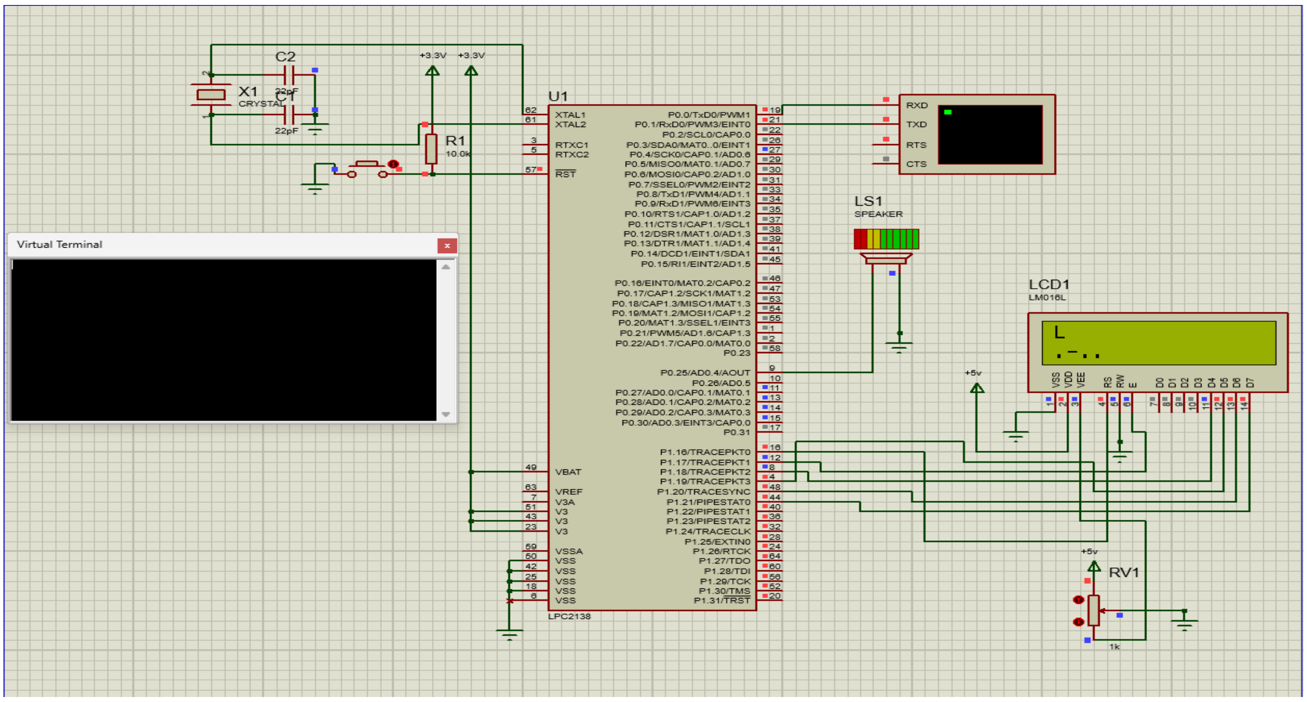
- Each character typed in the virtual terminal appears on the LCD.
- Morse code translation of each character is shown on the second line of the LCD.
- Speaker generates beeps with correct timing for dots and dashes.
- Word gaps and letter gaps are distinctly handled using appropriate delays.
- Invalid characters are identified and reported.

By entering HELLO WORLD:

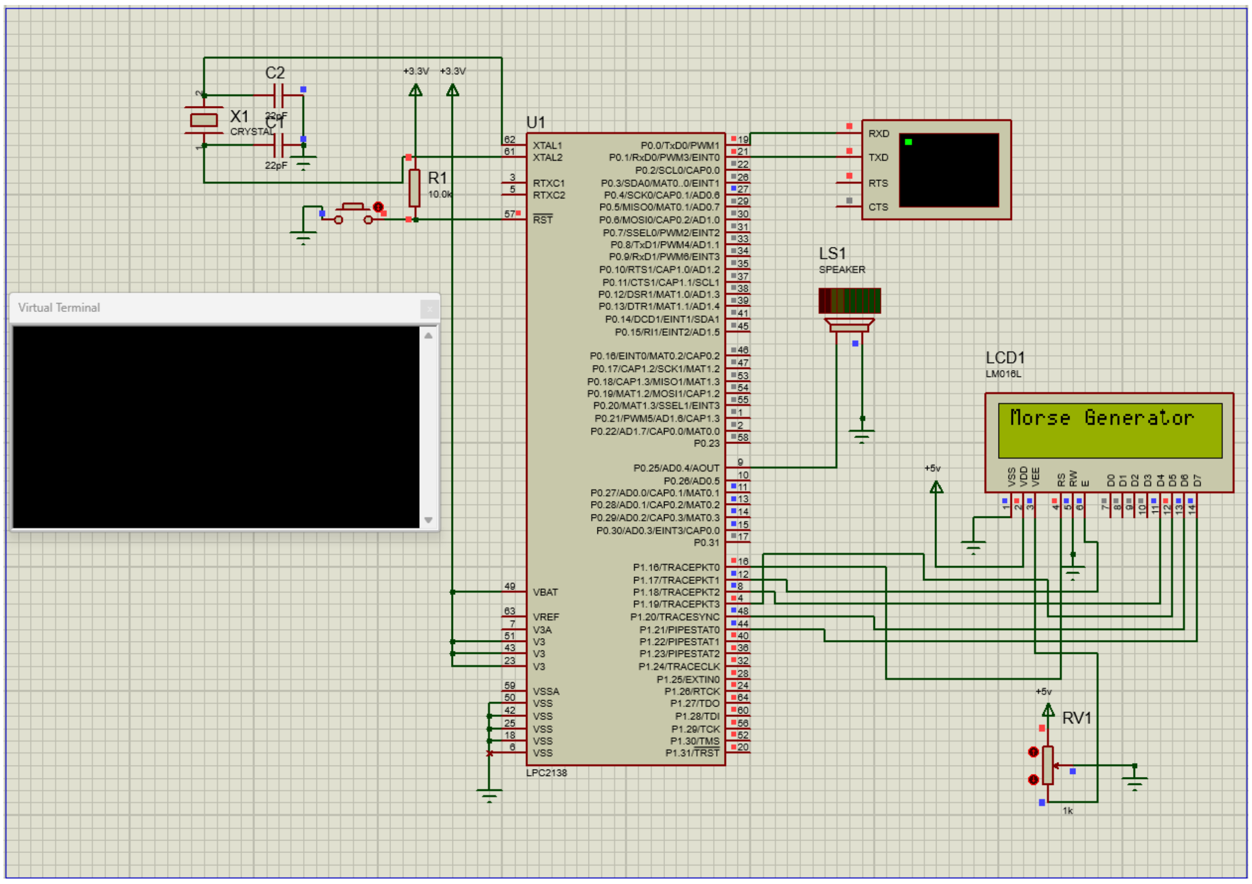
Received



OUTPUT:



Screenshot of the simulator-based circuit



Video link for the working model:

[BL.EN.U4ECE23208_BL.EN.U4ECE23213_BL.EN.U4ECE23223_BL.EN.U4ECE23249.mp4]

Any other relevant information:

- **Power Supply**: The LPC2138 is powered by a 3.3V supply, which is appropriately configured in the Proteus simulation.
- **Crystal Oscillator**: A 12 MHz crystal with 22pF capacitors ensures stable operation of the LPC2138 clock.
- **Timing Precision**: Delay functions are based on nested loops and might vary slightly in real hardware; for precise timing, hardware timers should be fully utilized.
- **Code Modularity**: The program is written in a modular structure, separating hardware initialization, LCD handling, UART functions, and Morse generation, which enhances readability and maintainability.

Conclusion:

This project effectively demonstrates a Morse Code Generator implemented using the LPC2138 ARM7 microcontroller. It receives characters through UART, displays the character and its Morse code equivalent on a 16x2 LCD, and generates corresponding tones through a speaker. The design integrates UART serial communication, LCD interfacing in 4-bit mode, and tone generation using a timer-controlled GPIO pin. The system was rigorously tested with both valid and invalid inputs, and all core functionalities were verified. With the initial implementation challenges resolved, the project now operates reliably and serves as a solid example of embedded system design combining communication, signal processing, and user feedback

Problems faced during the implementation:

- During the implementation phase, several issues arose. The LCD initially failed to display any data or showed random symbols, indicating an issue with its initialization.

- The speaker connected to P0.25 did not produce any sound even though the code was generating Morse patterns. Some alphabetic characters were not translating to the correct Morse code sequences or were not handled at all.

- UART communication was unresponsive at first, and no characters appeared to be received by the microcontroller.

- Finally, when characters were input too quickly in succession, the system became erratic, either skipping inputs or overlapping tones and display output

Problems resolved using solutions

- To address these problems, several corrective steps were taken. For the LCD, proper 4-bit initialization commands were issued with adequate delays to stabilize the display.

- The speaker issue was resolved by correctly configuring the I/O pin (P0.25) using PINSEL1 and setting its direction with IO0DIR, along with implementing a timer-based square wave using Timer0 to drive the speaker.

- Incorrect Morse mappings were fixed by converting lowercase inputs to uppercase before accessing the lookup tables, and by validating character ranges.

- UART communication was restored by verifying the PINSEL0 settings for TxD0 and RxD0 and setting the correct baud rate (9600 bps) using U0DLL.

- Lastly, to prevent glitches from fast inputs, carriage return, and newline characters were filtered out, and delays were added between operations to ensure the system had time to process each input completely.

References:

- https://embetronicx.com/tutorials/microcontrollers/lpc2148/keypad-interfacing-with-lpc2148/
- https://embetronicx.com/tutorials/microcontrollers/lpc2148/lpc2148-gpio- tutorial-led-interfacing/
- https://github.com/VaibhavDeshpande2199/RTOS-based-Home-Automation-System
- https://github.com/Suraksha-Rajagopalan/HomeAutomation_Proteus
- http://hc05-interfacing-with-lpc2138-using-keiluvision-and-proteus- main.zip/
- https://youtu.be/NlxskIs-0Bc?si=uvDInDJApiJJn7bn
- https://github.com/Suraksha-Rajagopalan/HomeAutomation_Proteus
- https://github.com/Suraksha-Rajagopalan/HomeAutomation_Proteus
- https://youtu.be/O3TlS5ztvBo?si=zeiRu75G-sgQR