MCT MINI PROJECT REPORT

1. **Group members:**

a. Amey Kumar	(23070123019)
b. Anushka Shinde	(23070123025)
c. Jiya Palod	(23070123070)
d. Maitreyee Gohad	(24070123509)

2. **Name/topic of project:**
   Morse Code Decoder

3. **Aim:**
   To implement a Morse Code Decoder using Arduino Uno Board

4. **Introduction:**
   This project aims to create an interactive Arduino Uno based Morse Code system. The system will process the Morse signals input by users through push buttons, decode them into text, and show the output on an LCD screen. This project highlights the application of microcontroller-based systems and Morse Code, acting as a functional communication tool and a learning tool.

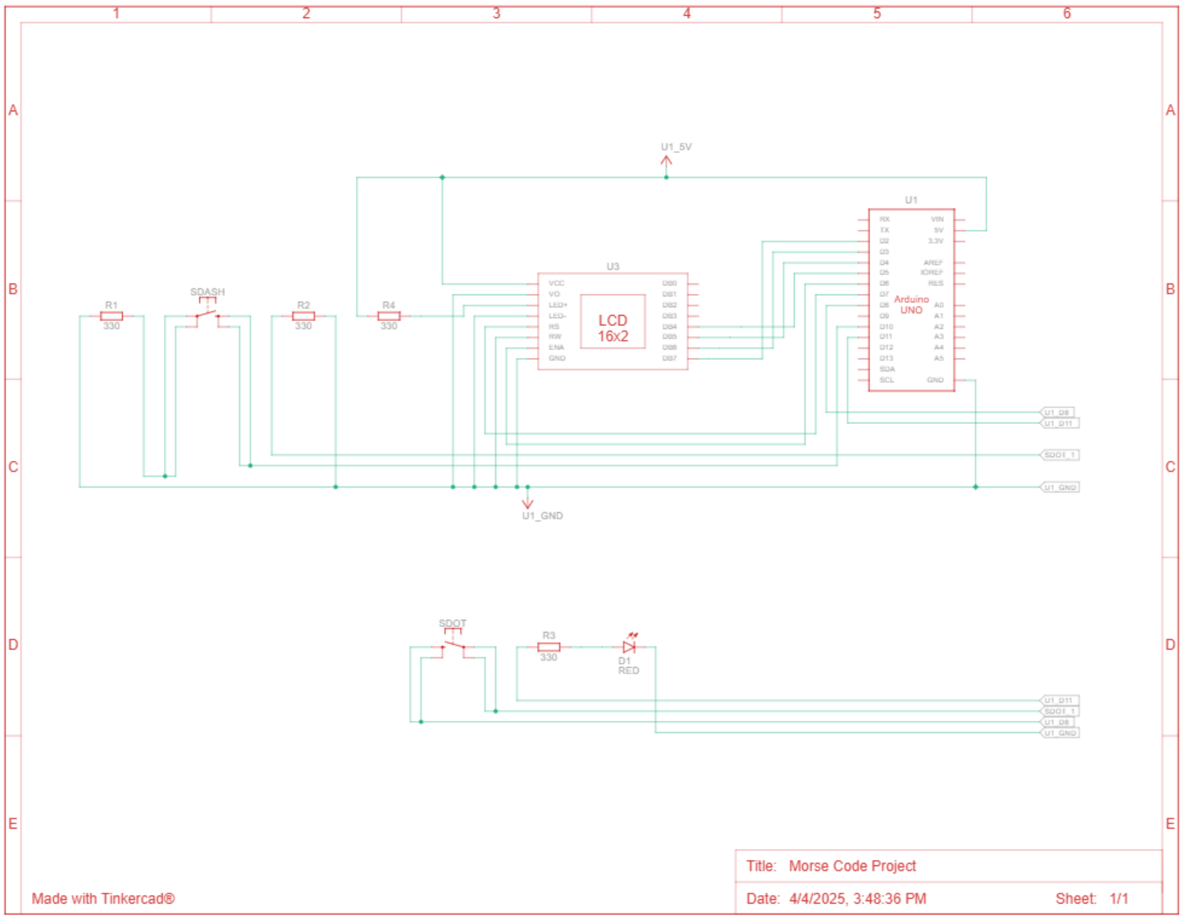
5. **Equipments:**
   Hardware -
   a.Arduino Uno Development Board
   b. Breadboard
   c. LCD Display
   d. LED
   e. Resistors
   f. Push Buttons
   g.Connecting wires
   Software -
   Arduino IDE

6. **Declaration:**
   We hereby declare that the project work titled "**Morse Code Decoder using Arduino UNO board"** submitted to, has been completed with all acknowledgements attached.

7. **Acknowledgements:**
   We would like to express our special thanks of gratitude to to our Hands-on Microcontrollers teacher for her able guidance and support throughout the course of our project.

8. Schematic Diagram:

Name	Quantity	Component
U1	1	Arduino Uno R3
U3	1	LCD 16 x 2
SDash SDot	2	Pushbutton
R1 R2 R3 R4	4	330 Ω Resistor
D1	1	Red LED



9. Simulation Screenshot:

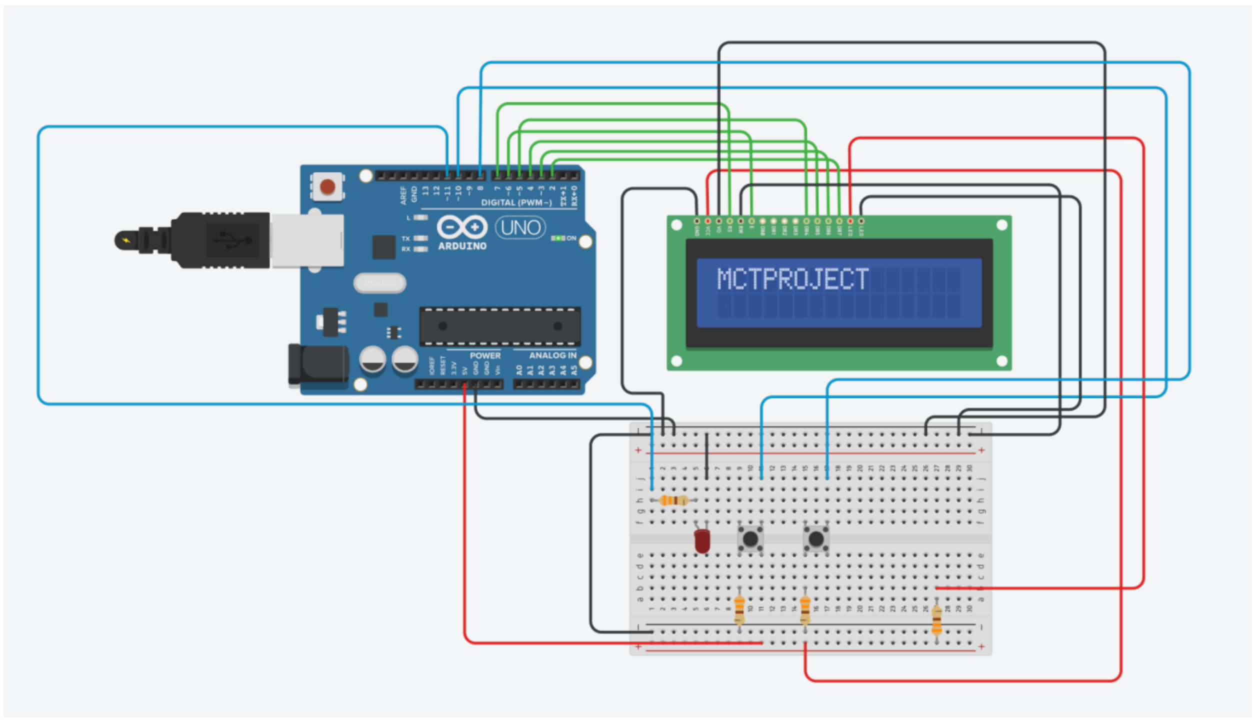


Fig. TinkerCAD Simulation

---

10. Learning objectives:

- **a.** Use Arduino Uno to detect and differentiate Morse code signals (dots and dashes) using 2 pushbuttons.
- **b.** Write and implement code to decode Morse patterns into readable characters using logical conditions and lookup tables.
- **c.** Display the decoded message on an LCD screen by interfacing it with the Arduino.

---

11. Theory:

Morse code is a method of communication where letters and numbers are represented using combinations of dots (·) and dashes (–). This binary signaling method has historical significance in telegraphy and remains an efficient way of encoding messages.

In this project, an **Arduino Uno** is used to decode Morse code entered through **two separate pushuttons**—one dedicated for **dots** and the other for **dashes**. Every time a button is pressed, the Arduino detects which button was used and stores the corresponding symbol (. or -) in a sequence.

After a character is completed , the Arduino compares the dot-dash sequence with a predefined **Morse code lookup table**. Once matched, the corresponding alphabet or number is identified.

The decoded characters are then displayed on an **LCD screen** connected to the Arduino. The entire system runs using code written in the **Arduino C/C++** language within the Arduino IDE. Logical conditions, arrays (or maps), and button handling are used to create an interactive and functional Morse code decoder.
This project helps understand **digital input reading**, **character mapping**, and **hardware interfacing**, making it a great hands-on learning experience in embedded systems.

<u>Working Principle:</u>

- **Input Mechanism:**
  Two push buttons are connected to digital input pins on the Arduino Uno.
  - **Button 1** is assigned for the **dot (·)** input.
  - **Button 2** is assigned for the **dash (–)** input.
- **Input Handling in Code:**
  Each time the user presses a button, the Arduino reads the digital state of the pin. Depending on which button is pressed, it appends a dot or dash to a string (or character array) representing the current Morse sequence.
- **Sequence Completion:**
  After completing a character (usually using a **third button** for "enter" or a delay/pause), the stored dot-dash sequence is compared against a **Morse code lookup table**.
  The lookup table is implemented using arrays, if-else conditions, or a switch-case structure in the Arduino code.
- **Character Mapping and Output:**
  Once a match is found, the corresponding alphabet or digit is displayed on a **16x2 LCD screen** connected via parallel or I2C communication. This allows the user to see the decoded text in real time as they enter each Morse symbol.

---

12. Program:

```
#include <LiquidCrystal.h>

// Pin configuration
const int dotBtn = 8;
const int dashBtn = 10;
const int ledPin = 11;
const int RS = 13, EN = 12, D4 = 5, D5 = 4, D6 = 3, D7 = 2;
LiquidCrystal lcd(RS, EN, D4, D5, D6, D7);

// Morse data
String morseInput = "";
unsigned long lastPressTime = 0;
unsigned long startTime;

String morseCode[] =
{
  ".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....",
  "..", ".---", "-.-", ".-..", "--", "-.", "---", ".--.",
  "--.-", ".-.", "...", "-", "..-", "...-", ".--", "-..-",
  "-.--", "--.."
};
char letters[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

void setup() {
  pinMode(dotBtn, INPUT);     // Using external pull-down resistors
  pinMode(dashBtn, INPUT);
  pinMode(ledPin, OUTPUT);

  Serial.begin(9600);
  lcd.begin(16, 2);
  lcd.print("Morse Decoder");
  Serial.println("Morse Decoder");
  delay(3000);  // Let message display

  lcd.clear();
  lcd.setCursor(0, 0);
  startTime = millis();  // Capture initial time to avoid false decoding
}

void loop()
{
  // Detect dot button press
  if (digitalRead(dotBtn) == HIGH) {
    morseInput += ".";
    blinkDot();
    lastPressTime = millis();
    while (digitalRead(dotBtn) == HIGH); // Debounce - wait for release
  }

  // Detect dash button press
  if (digitalRead(dashBtn) == HIGH)
{

    morseInput += "-";
    blinkDash();
    lastPressTime = millis();
    while (digitalRead(dashBtn) == HIGH); // Debounce - wait for release
  }

  // Decode if 1s has passed since last press and at least 3s after startup
  if (morseInput.length() > 0 && millis() - lastPressTime > 1000 && millis() - startTime > 3000)
{
    String decoded = decodeMorse(morseInput);
    Serial.print(decoded);
    lcd.print(decoded);
    morseInput = "";
  }
}

// LED feedback
void blinkDot()
{
  digitalWrite(ledPin, HIGH);
  delay(200);
  digitalWrite(ledPin, LOW);
  delay(50);
}

void blinkDash()
{
  digitalWrite(ledPin, HIGH);
  delay(600);
  digitalWrite(ledPin, LOW);
  delay(50);
}

// Translate Morse code to letter
String decodeMorse(String morse) {
  for (int i = 0; i < 26; i++) {
    if (morse == morseCode[i]) {
      return String(letters[i]);
    }
  }
  return "?";  // Unknown input
}
```

- The code is written in **Arduino C/C++**, and uploaded via the **Arduino IDE**.
- setup() initializes pin modes, serial communication (if needed), and the LCD display.
- loop() constantly checks the state of both buttons and constructs the Morse sequence accordingly.
- Debouncing is applied in the code to avoid false readings caused by mechanical noise from button presses.
- The message can be cleared or reset using another button or after a specific number of characters.

---

13. Project Methodology:

a. **Component Selection:**
Selected essential components like Arduino Uno, two push buttons for dot and dash input, a 16x2 LCD display, resistors, breadboard, and jumper wires.

b. **Circuit Design:**
Designed the hardware setup by connecting the push buttons to digital input pins and interfacing the LCD with the Arduino.

c. **Pin Configuration:**
Assigned Arduino pins for dot and dash inputs, LCD data/control lines, and defined them in the code using pinMode() in the setup() function.

d. **Code Development:**
Wrote the Arduino program using C/C++ to detect button presses, identify dot/dash input, and store them in a sequence.

e. **Morse Code Decoding Logic:**
Created a lookup mechanism in code to compare stored dot-dash patterns with standard Morse code and decode them into readable characters.

f. **LCD Output Display:**
Displayed each decoded character on the LCD in real time as the user inputs Morse code, giving live feedback.

g. **Testing and Debugging:**
Tested each module (buttons, decoding, LCD), handled software debouncing, and fine-tuned the code for reliable performance.

---

14. Testing of the Project:

Once the circuit was assembled and the code was uploaded to the Arduino Uno, the project was tested to ensure proper functionality. The push buttons were checked individually to confirm accurate detection of dot and dash inputs. Various combinations were entered to form Morse code sequences, which were then decoded and compared against the expected characters. The decoding logic was validated by matching multiple input patterns with their corresponding alphabets or numbers. The LCD display was monitored throughout to ensure that the decoded characters were correctly shown in real-time. Debouncing was also tested and handled in software to avoid false triggers caused by rapid or shaky button presses. Additionally, sequences involving multiple characters were entered to evaluate how well the system handled continuous input. Overall, the system performed as expected, accurately detecting, decoding, and displaying Morse code messages.

15. Photos:

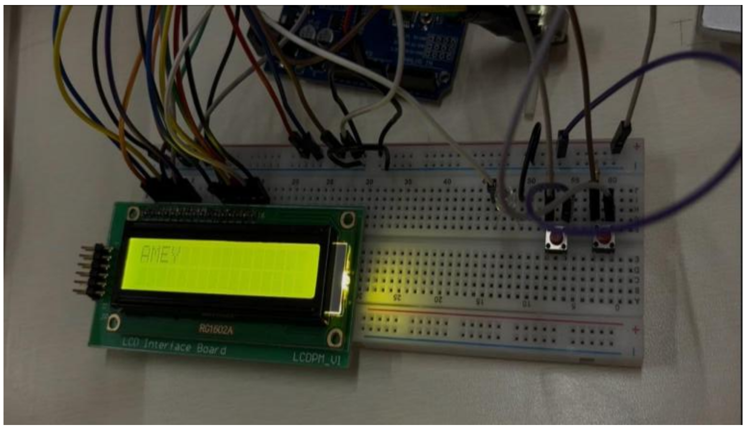
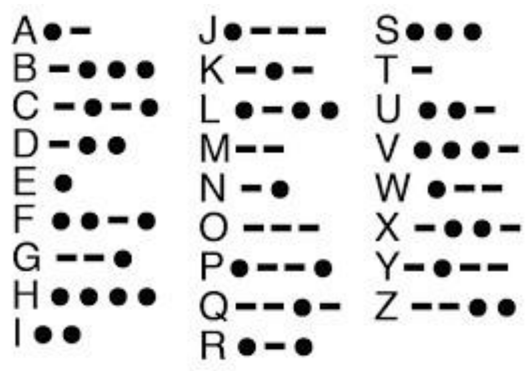


Fig.(a)Breadboard



A	•–	J	•–––	S	•••
B	–•••	K	–•–	T	–
C	–•–•	L	•–••	U	••–
D	–••	M	––	V	•••–
E	•	N	–•	W	•––
F	••–•	O	–––	X	–••–
G	––•	P	•––•	Y	–•––
H	••••	Q	––•–	Z	––••
I	••	R	•–•		

Fig(b)Morse Code reference

16. Conclusion:

The Morse Code Decoder using Arduino Uno was successfully designed and implemented using two push buttons for manual input and a 16x2 LCD for output display. The project demonstrated how Morse code can be decoded through user-friendly hardware and efficient programming. By detecting dot and dash inputs, mapping them to characters using a lookup table, and displaying them in real-time, the system provided a simple yet effective solution for decoding Morse code. This project helped in understanding the fundamentals of embedded systems, digital input handling, character mapping, and hardware-software integration. It also reinforced practical skills in coding, circuit building, and debugging. Overall, the project served as a strong learning experience and a stepping stone toward more advanced microcontroller-based applications.