Objectives

- To design and develop a smart glove system equipped with five flex sensors, each mounted on one finger, to detect hand gestures based on finger movements.
- To accurately recognize static hand gestures by analyzing the unique bending patterns of the flex sensors and mapping them to predefined words or phrases.
- To convert recognized gestures into readable text displayed in real-time on a 16x2 LCD screen via an I2C interface.
- To provide simultaneous audio output for each recognized gesture using the MP3TF16P module and a speaker, enhancing communication through spoken words.
- To support bilingual output in both English and Hindi, allowing broader accessibility and communication in diverse linguistic settings.
- To promote inclusion and accessibility by bridging the communication gap between sign language users and those unfamiliar with it.

Literature Review

Over time, various solutions have emerged to support communication for the hearing-impaired, with sign language translation gloves showing strong potential as assistive tools. Early versions mainly converted hand gestures into text using microcontrollers and small LCDs, but often supported only a few words and lacked audio output. Flex sensors became a common choice due to their affordability and reliability in detecting finger movements.

More recent developments have improved these gloves by incorporating audio modules like the MP3TF16P, allowing spoken output for each recognized gesture. While some projects have explored camera-based gesture recognition, such systems face limitations in real-time use due to lighting issues and high processing needs. In contrast, glove-based solutions are more practical and portable.

This project enhances previous efforts by using five flex sensors to detect static hand gestures and map them to 31 words in both English and Hindi. The addition of an LCD for text display and an MP3TF16P module for voice output makes it a cost-effective, bilingual communication aid for the hearing-impaired.

System Architecture

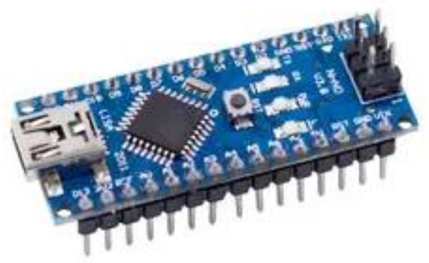
Components	Description
Arduino Nano (ATMega328p)	Central microcontroller that reads flex sensor data, processes gesture recognition, manages language selection, and controls outputs (LCD and audio).
16x2 LCD Panel+I2C module	Displays the recognized word in the selected language; I2C module simplifies wiring and communication with Arduino.
Flex Sensors (x5)	Detect the degree of bending for each finger, providing unique analog input patterns for gesture recognition.
9V Battery	Powers the entire system, including Arduino, sensors, LCD, and audio module.
Resistors (10kΩ + 1kΩ)	Used in series with each flex sensor to form voltage dividers, enabling accurate analog readings by the Arduino as well as to control the current going into the mp3 module.
Wires	Facilitate all electrical connections between components.

Solder	Ensures permanent, reliable electrical connections on the glove and circuit.
Glove	Physical base for mounting flex sensors, allowing natural hand movement and gesture input.
MP3TF16P Module + Sd card	Handles audio playback; plays pre-recorded MP3 files corresponding to recognized words in the selected language.
Speaker	Outputs the audio for the recognized word, enabling spoken communication.

Hardware Components:

Arduino Nano (ATMega328p)

➢ The Arduino Nano is a compact microcontroller board based on the ATmega328p chip.
➢ It features 14 digital I/O pins, 8 analog inputs, PWM outputs, and UART, SPI, and I2C communication interfaces.
➢ Its small size and USB connectivity make it ideal for embedded electronics and wearable technology.
➢ It is widely used in DIY electronics, robotics, sensor interfacing, and control systems for its ease of programming and low power consumption.



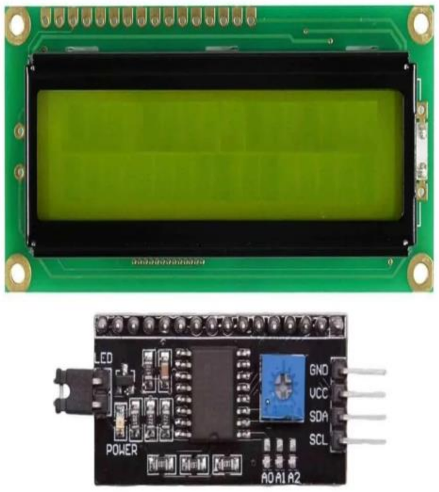
Arduino Nano

16x2 LCD Panel + I2C Module

➢ A 16x2 LCD is a display module capable of showing 2 lines of 16 characters each, commonly used to present textual information.

➢ The I2C module simplifies the interface by using only two wires (SDA and SCL) for communication, reducing pin usage on microcontrollers.

➢ It is commonly used in embedded systems for displaying status messages, sensor data, or user prompts in real time.



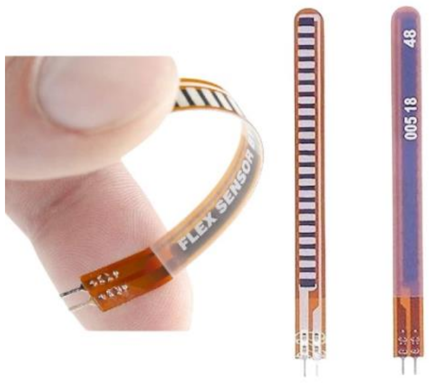
LCD Display + I2C interface

Flex Sensors (x5)

➢ Flex sensors are variable resistors that change their resistance based on the degree of bending.

➢ They are lightweight and flexible, making them suitable for capturing motion or physical deformation.

➢ These sensors are often used in wearables, robotics, virtual reality gloves, and other gesture-based applications.



Flex Sensor

9V Battery

➢ A 9V battery is a common portable power source that provides a nominal voltage of 9 volts.

➢ It is frequently used in low-power electronics projects, remote sensors, and handheld devices.

➢ Its compact size and ease of replacement make it suitable for powering small embedded systems.

9V Battery

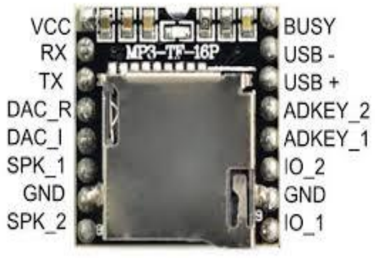
Glove

➢ A glove is a wearable fabric item designed to cover the hand and fingers.
➢ In electronics projects, gloves are often modified to serve as platforms for sensors or actuators.
➢ They are commonly used in motion capture, virtual reality, rehabilitation, and assistive technology devices.

Gloves

MP3TF16P Module + SD Card

➢ The MP3TF16P is an audio playback module that reads MP3 files stored on an SD card and outputs them through a speaker.
➢ It supports serial communication for control and offers onboard DAC for sound output.
➢ This module is widely used in voice alert systems, talking devices, and sound-enabled interactive electronics.



MP3TF16P

Speaker

➢ A speaker is a transducer that converts electrical audio signals into sound waves.
➢ It comes in various sizes and impedances, depending on the power and frequency response required.
➢ Commonly found in multimedia systems, alarm circuits, voice output modules, and embedded audio projects.

Speaker

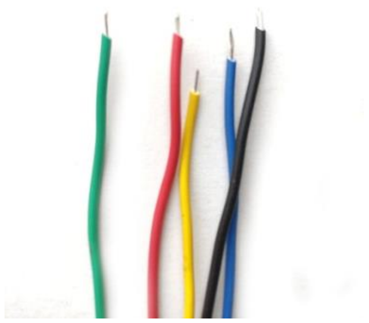
Resistors (10kΩ + 1kΩ)

➢ Resistors are passive electrical components used to limit current, divide voltages, or pull-up/down signals.
➢ The 10kΩ and 1kΩ values are standard in analog signal conditioning and microcontroller interfacing circuits.
➢ They are widely used in voltage dividers, filtering networks, and to protect components from excessive current.

Resistor

Wires

➢ Wires serve as conductors to connect different components in a circuit.
➢ Available in solid or stranded forms, they come in various colors and thicknesses for organized and reliable connections.
➢ They are essential in prototyping, breadboarding, and permanent circuit installations.


Wires

Solder

➢ Solder is a fusible metal alloy used to create permanent electrical connections between components and circuit boards.
➢ Typically composed of tin and lead (or lead-free alternatives), it ensures mechanical strength and conductivity.
➢ It is essential in both hand-soldering and automated electronics assembly processes.

Solder

Theory

Core Principle: This project operates on the principle of mapping unique combinations of finger flexion to specific words in sign language. The degree of bending in each finger, detected by the flex sensors, creates a distinct pattern that the Arduino interprets to identify a word.

Sensor Data Acquisition and Interpretation:

1. **Flex Sensor as Variable Resistor:** Each flex sensor acts as a variable resistor. When a finger is bent, the resistance of the corresponding sensor increases. The more the finger bends, the higher the resistance.
2. **Analog to Digital Conversion (ADC):** The Arduino Nano's built-in Analog-to-Digital Converter (ADC) converts the analog voltage readings from each flex sensor into digital values. These digital values represent the degree of bend for each finger.
3. **Multi-Finger Pattern Collection:** Five flex sensors, placed on the five fingers of a glove, provide a continuous stream of analog input. Each reading is sampled and interpreted in real-time to track hand gestures.

Word Recognition and Mapping:

1. **Defining Finger Bend Combinations:** For each of the 31 target words (for each language), a unique combination of finger bend states (e.g., straight, fully bent) is defined. Each word is essentially a "gesture signature."

2. **Thresholding or Range Mapping:** The digital values from the flex sensors are analysed to determine the state of each finger. This is done by setting specific threshold values or defining value ranges for different states:

   - **Straight:** Low resistance
   - **Fully bent:** High resistance

3. **Pattern Matching:** The Arduino continuously reads the sensor data and compares the current finger configuration against predefined gesture patterns. Matching is implemented through conditional statements or a lookup table (array or mapping structure).

4. **Language-Dependent Mapping:** Once a gesture is matched to a word, the Arduino checks the selected language (English or Hindi) and links the gesture to a specific text and audio output for that language.

System Workflow

1. **Startup & Language Selection:** When the system is powered on, the user is prompted (via both LCD and audio) to select a language using a gesture. Two predefined gestures are reserved - one for English and one for Hindi. The Arduino waits for a valid gesture before continuing.
2. **Gesture Detection Loop:** After language selection, the system enters its main loop where it constantly monitors the sensor values and matches them to the 31 predefined gestures for each language.

Output

Text Display: Once a matching finger combination is identified, the corresponding word (based on the selected language) is displayed on a 16x2 LCD panel via the I2C interface.

Audio Output: Simultaneously, the associated MP3 file for the recognized word is played using the MP3TF16P module. The audio files are stored on a microSD card in the module and are named in a structured manner.

Working Principle

The glove employs **flex sensors** mounted on each of the five fingers to detect the degree of bending. As the user forms a sign language gesture, the resistance of each sensor changes proportionally to the finger’s flexion. These resistance changes are read as **analog voltages** by the Arduino Nano.

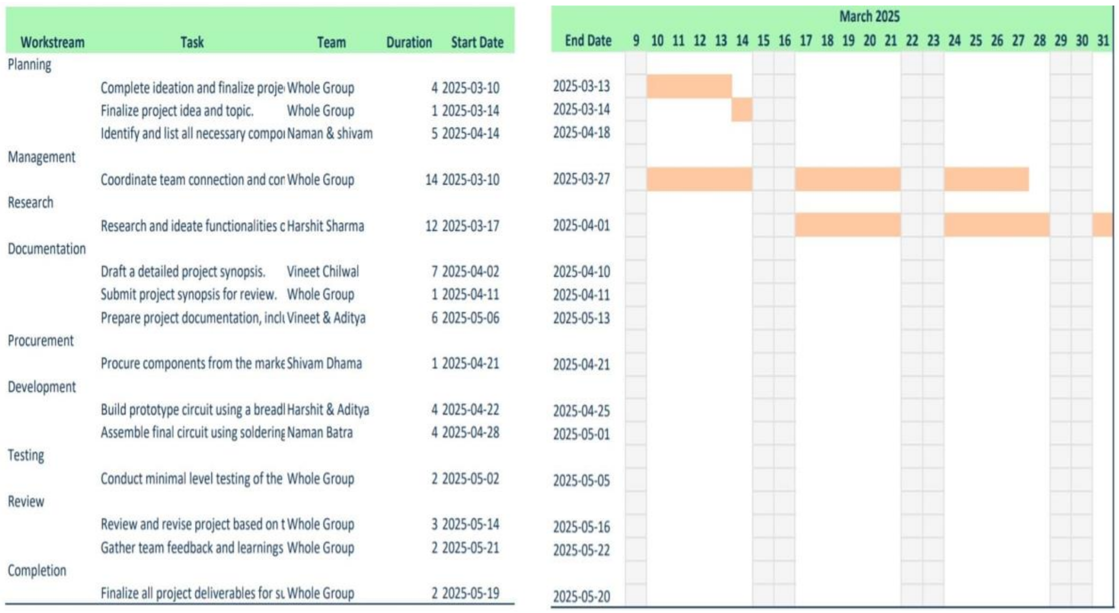
The Arduino’s **Analog-to-Digital Converter (ADC)** converts these analog voltages into digital values, effectively capturing a digital snapshot of the hand gesture. This snapshot, a combination of five digital values, is then compared against a **predefined library of gesture patterns** representing 31 specific words in each language.

Once a match is identified, the corresponding word is:

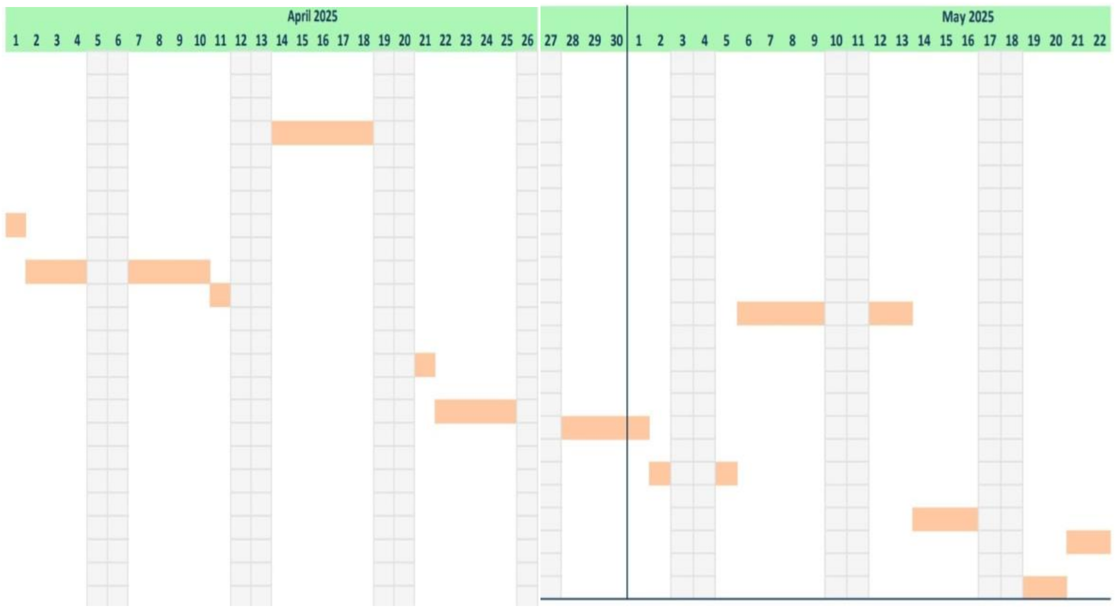
- **Displayed as text** on a 16x2 LCD screen via the I2C interface.
- **Spoken aloud** through a speaker using the MP3TF16P audio module in the **language selected by the user** (English or Hindi).

This process enables seamless **real-time translation of sign language into both textual and audible communication**, making interaction inclusive and accessible.

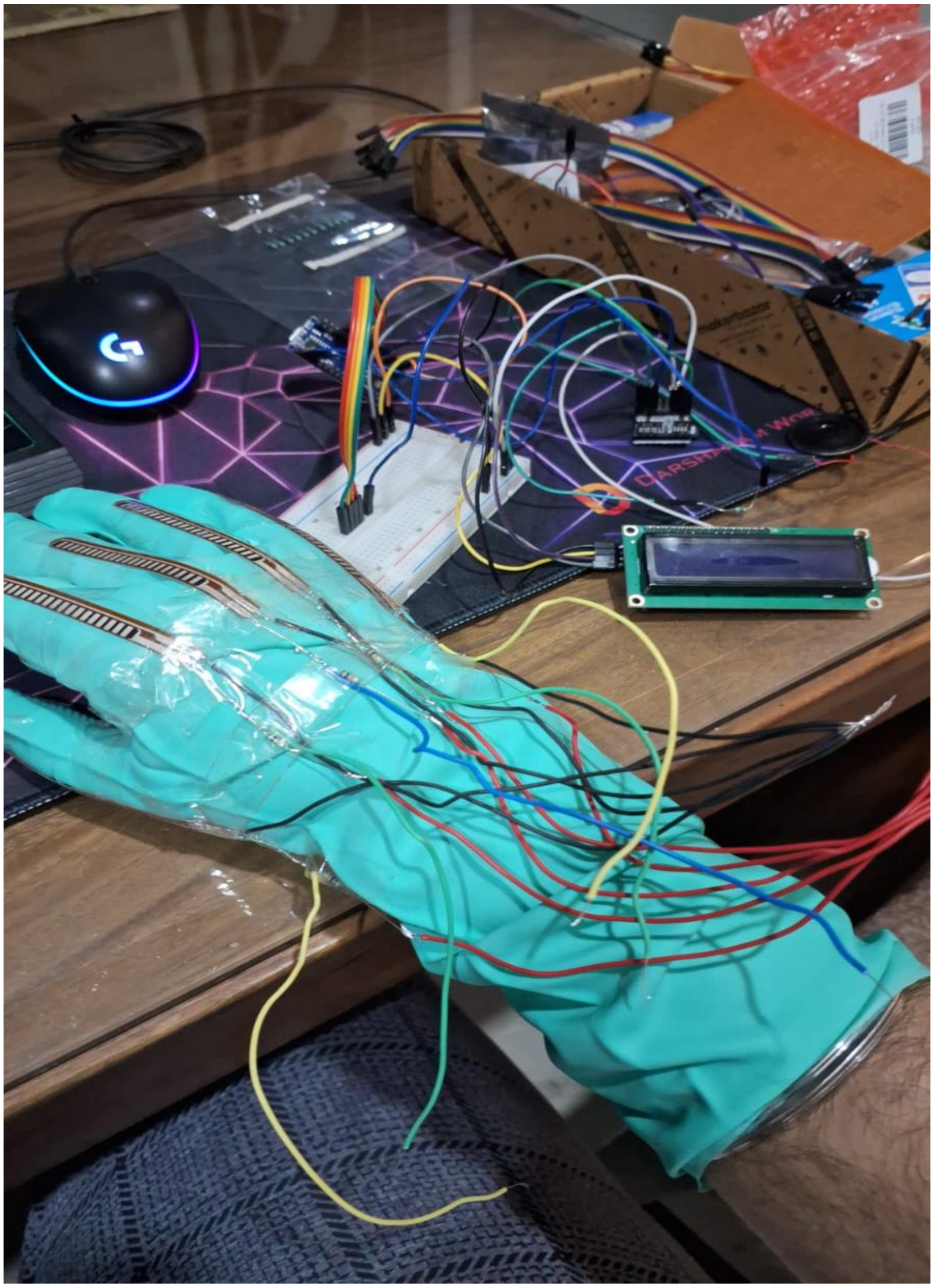
Gantt Chart



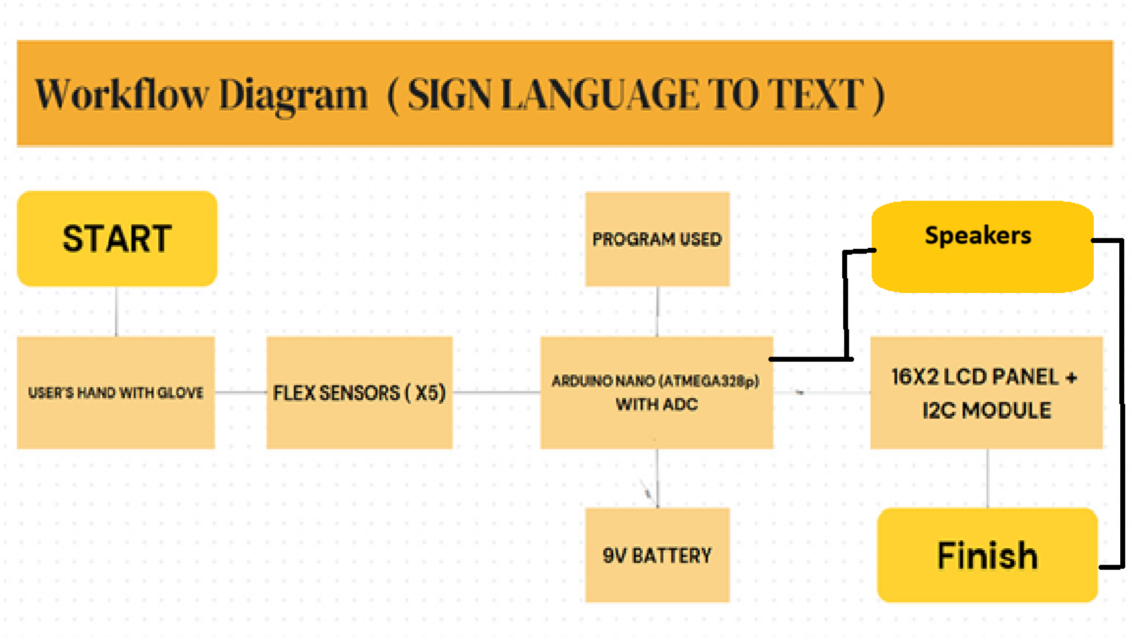
Workstream	Task	Team	Duration	Start Date	End Date
Planning					
	Complete ideation and finalize proje	Whole Group	4	2025-03-10	2025-03-13
	Finalize project idea and topic.	Whole Group	1	2025-03-14	2025-03-14
	Identify and list all necessary compo	Naman & shivam	5	2025-04-14	2025-04-18
Management					
	Coordinate team connection and cor	Whole Group	14	2025-03-10	2025-03-27
Research					
	Research and ideate functionalities c	Harshit Sharma	12	2025-03-17	2025-04-01
Documentation					
	Draft a detailed project synopsis.	Vineet Chilwal	7	2025-04-02	2025-04-10
	Submit project synopsis for review.	Whole Group	1	2025-04-11	2025-04-11
	Prepare project documentation, incl	Vineet & Aditya	6	2025-05-06	2025-05-13
Procurement					
	Procure components from the marke	Shivam Dhama	1	2025-04-21	2025-04-21
Development					
	Build prototype circuit using a bread	Harshit & Aditya	4	2025-04-22	2025-04-25
	Assemble final circuit using soldering	Naman Batra	4	2025-04-28	2025-05-01
Testing					
	Conduct minimal level testing of the	Whole Group	2	2025-05-02	2025-05-05
Review					
	Review and revise project based on t	Whole Group	3	2025-05-14	2025-05-16
	Gather team feedback and learnings	Whole Group	2	2025-05-21	2025-05-22
Completion					
	Finalize all project deliverables for su	Whole Group	2	2025-05-19	2025-05-20



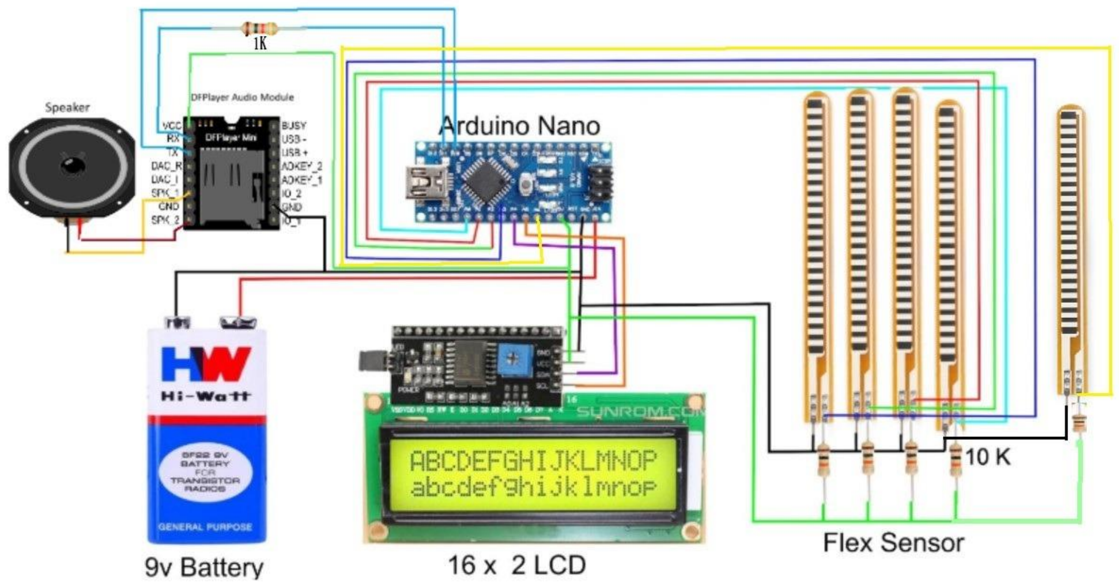
Prototype



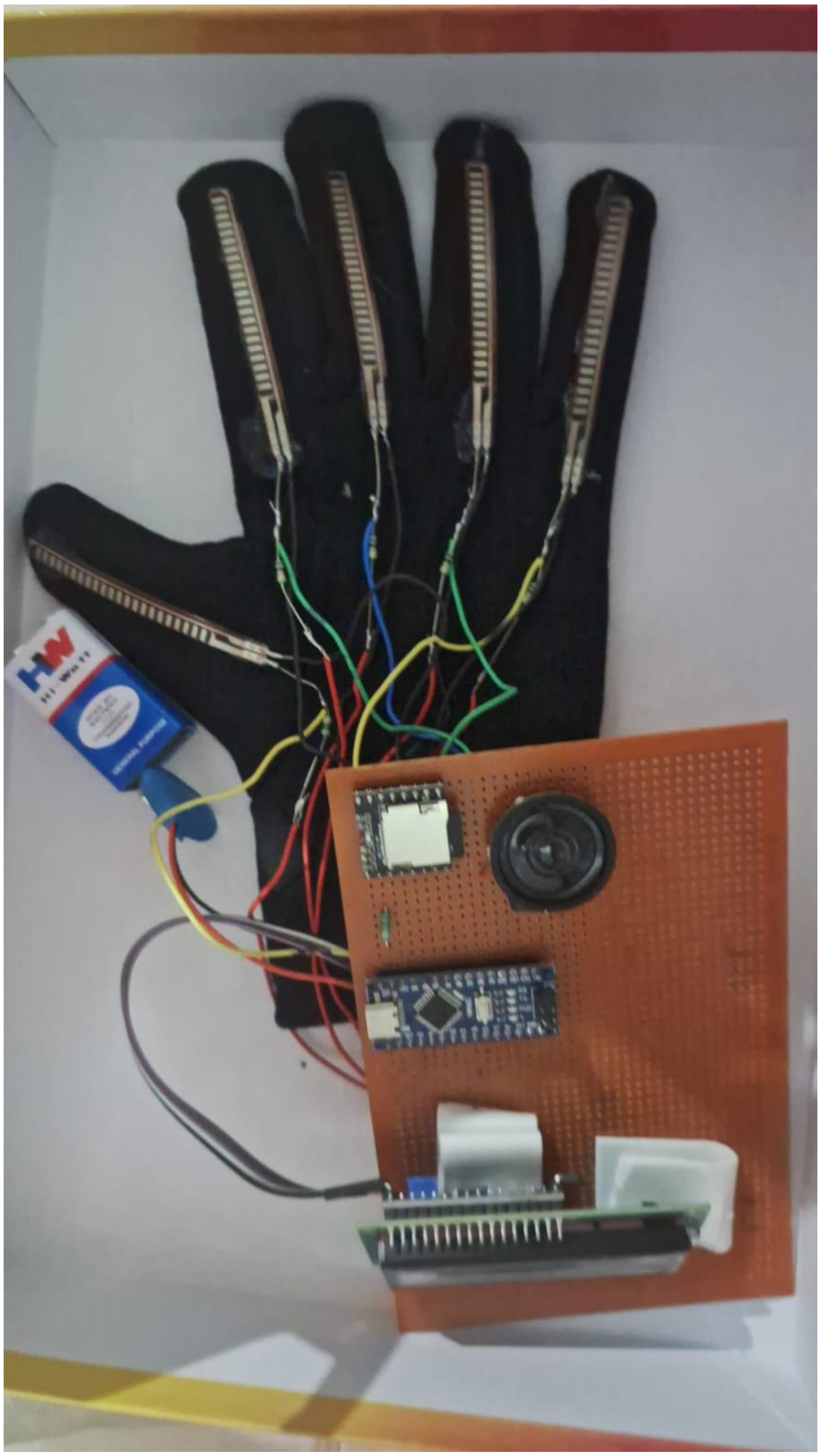
Block Diagram:


Workflow Diagram (SIGN LANGUAGE TO TEXT)
START
PROGRAM USED
Speakers
USER'S HAND WITH GLOVE
FLEX SENSORS (X5)
ARDUINO NANO (ATMEGA328p) WITH ADC
16X2 LCD PANEL + I2C MODULE
9V BATTERY
Finish

Connection Diagram:



Final Project



Arduino Programming

The programming for the Arduino nano microcontroller will be done in the Arduino IDE, then the program will be loaded in the flash memory of the ATMega328p chip. The code will then continuously run on it while it is powered on and keep detecting the flexion of the sensors to give suitable output.

Code:-

```
#include <SoftwareSerial.h>
#include <DFRobotDFPlayerMini.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <avr/pgmspace.h> //for PROGMEM

// --- Pin Definitions and Constants ---
const int LCD_I2C_ADDRESS = 0x27; //LCD's I2C address
const int MP3_RX_PIN = 10;       //Arduino RX for MP3 (connect to MP3 TX with 1k resistor)
const int MP3_TX_PIN = 11;       //Arduino TX for MP3 (connect to MP3 RX)

//Flex Sensor Analog Pins
const int flexPin_f1 = A0;
const int flexPin_f2 = A1;
const int flexPin_f3 = A2;
const int flexPin_f4 = A3;
const int flexPin_f5 = A6;

const int BEND_THRESHOLD = 800  ; //Threshold to consider finger bent

// --- MP3 Player and LCD Objects ---
SoftwareSerial mySerial(MP3_RX_PIN, MP3_TX_PIN);
DFRobotDFPlayerMini myPlayer;
LiquidCrystal_I2C lcd(LCD_I2C_ADDRESS, 16, 2); // 16 columns, 2 rows

// --- Language State ---
enum Language { ENGLISH, HINDI };
Language currentLanguage = ENGLISH; // Default language
```

```
// --- Word List (Stored in PROGMEM) ---

// English Word Table
const char en_string_0[] PROGMEM = "hello";
const char en_string_1[] PROGMEM = "food";
const char en_string_2[] PROGMEM = "water";
const char en_string_3[] PROGMEM = "washroom";
const char en_string_4[] PROGMEM = "help";
const char en_string_5[] PROGMEM = "please";
const char en_string_6[] PROGMEM = "thank you";
const char en_string_7[] PROGMEM = "come here";
const char en_string_8[] PROGMEM = "bye";
const char en_string_9[] PROGMEM = "sorry";
const char en_string_10[] PROGMEM = "hot";
const char en_string_11[] PROGMEM = "cold";
const char en_string_12[] PROGMEM = "day";
const char en_string_13[] PROGMEM = "night";
const char en_string_14[] PROGMEM = "yesterday";
const char en_string_15[] PROGMEM = "tomorrow";
const char en_string_16[] PROGMEM = "today";
const char en_string_17[] PROGMEM = "time?";
const char en_string_18[] PROGMEM = "date";
const char en_string_19[] PROGMEM = "like";
const char en_string_20[] PROGMEM = "dislike";
const char en_string_21[] PROGMEM = "happy";
const char en_string_22[] PROGMEM = "sad";
const char en_string_23[] PROGMEM = "angry";
const char en_string_24[] PROGMEM = "how";
const char en_string_25[] PROGMEM = "why";
const char en_string_26[] PROGMEM = "pain";
const char en_string_27[] PROGMEM = "forward";
const char en_string_28[] PROGMEM = "right";
const char en_string_29[] PROGMEM = "left";
const char en_string_30[] PROGMEM = "backward";
const char en_string_31[] PROGMEM = "****"; // 31 words

const char* const en_word_table[] PROGMEM = {
  en_string_0, en_string_1, en_string_2, en_string_3, en_string_4, en_string_5, en_string_6,
en_string_7,
  en_string_8, en_string_9, en_string_10, en_string_11, en_string_12, en_string_13,
en_string_14, en_string_15,
  en_string_16, en_string_17, en_string_18, en_string_19, en_string_20, en_string_21,
en_string_22, en_string_23,
  en_string_24, en_string_25, en_string_26, en_string_27, en_string_28, en_string_29,
en_string_30, en_string_31
};
```

```
// Hindi Word Table (Replace with your actual Hindi translations)
const char hi_string_0[] PROGMEM = "namaste"; //32nd
const char hi_string_1[] PROGMEM = "khana";//33
const char hi_string_2[] PROGMEM = "pani";//34
const char hi_string_3[] PROGMEM = "shauchalay";//35
const char hi_string_4[] PROGMEM = "sahayeta kariye";//36
const char hi_string_5[] PROGMEM = "kripya";//37
const char hi_string_6[] PROGMEM = "dhanyawad";//38th - 36
const char hi_string_7[] PROGMEM = "idhar aao";
const char hi_string_8[] PROGMEM = "alvida";
const char hi_string_9[] PROGMEM = "maaf kijiye";
const char hi_string_10[] PROGMEM = "garam";
const char hi_string_11[] PROGMEM = "thanda";
const char hi_string_12[] PROGMEM = "din";
const char hi_string_13[] PROGMEM = "raat";
const char hi_string_14[] PROGMEM = "beeta hua kal";
const char hi_string_15[] PROGMEM = "aane wala kal";
const char hi_string_16[] PROGMEM = "aaj";
const char hi_string_17[] PROGMEM = "samay?";
const char hi_string_18[] PROGMEM = "tareekh";//50th - 48
const char hi_string_19[] PROGMEM = "pasand";//51st- 57
const char hi_string_20[] PROGMEM = "na pasand";//52nd-49
const char hi_string_21[] PROGMEM = "khush";//53rd-50
const char hi_string_22[] PROGMEM = "dukhhi";
const char hi_string_23[] PROGMEM = "gussa";//55th - 52
const char hi_string_24[] PROGMEM = "kaise";//56th - 58
const char hi_string_25[] PROGMEM = "kyun"; //57th-53
const char hi_string_26[] PROGMEM = "dard"; //27th
const char hi_string_27[] PROGMEM = "aage"; //28th
const char hi_string_28[] PROGMEM = "daayein"; //29th
const char hi_string_29[] PROGMEM = "baayein";
const char hi_string_30[] PROGMEM = "peeche";
const char hi_string_31[] PROGMEM = "****";


const char* const hi_word_table[] PROGMEM = {
  hi_string_0, hi_string_1, hi_string_2, hi_string_3, hi_string_4, hi_string_5, hi_string_6,
hi_string_7,
  hi_string_8, hi_string_9, hi_string_10, hi_string_11, hi_string_12, hi_string_13,
hi_string_14, hi_string_15,
  hi_string_16, hi_string_17, hi_string_18, hi_string_19, hi_string_20, hi_string_21,
hi_string_22, hi_string_23,
  hi_string_24, hi_string_25, hi_string_26, hi_string_27, hi_string_28, hi_string_29,
hi_string_30, hi_string_31
};
```

```
// --- Audio File Mapping (Stored in PROGMEM - Single List) ---
// Maps word index to audio file number (1-based).
// English words 0-29 map to audio files 1-30.
// Hindi words 0-29 map to audio files 31-60.
// Power-on = 61, English Select = 62, Hindi Select = 63.
const uint8_t audio_map[] PROGMEM = {
  1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, // English
0-22 -> Audio 1-23
  23, 24, 25, 26, 66, 27, 28, 29, 30,                                               //
English 23-29 -> Audio 24-30
  65, 31, 32, 33, 56, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 57, 49, 50,
// Hindi 0-19 -> Audio 31-50
  51, 52, 58, 53, 54, 55, 59, 60, 61, 65                                       // Hindi 20-29
-> Audio 51-60
};

const int NUM_WORDS = sizeof(en_word_table) / sizeof(en_word_table[0]); // Number of
words/phrases (30)
const int POWER_ON_AUDIO = 64;
const int ENGLISH_SELECT_AUDIO = 63;
const int HINDI_SELECT_AUDIO = 62;



// --- Helper functions to read from PROGMEM ---
char wordBuffer[30]; //Buffer for reading strings from Flash (size fits longest string + 1)

void readWordFromFlash(int wordIndex, Language lang) {
  char* wordAddress;
  if (lang == ENGLISH) {
    wordAddress = (char*)pgm_read_word(&(en_word_table[wordIndex]));
  } else { // HINDI
    wordAddress = (char*)pgm_read_word(&(hi_word_table[wordIndex]));
  }
  strcpy_P(wordBuffer, wordAddress);
}

uint8_t getAudioNumberForWord(int wordIndex, Language lang) {
    // This logic needs to map the word index and language to the correct audio file number
    if (lang == ENGLISH) {
        return pgm_read_byte(&(audio_map[wordIndex])); // English words map to audio_map
indices
    } else { // HINDI
        return pgm_read_byte(&(audio_map[wordIndex + NUM_WORDS])); // Hindi words map to
audio_map indices
    }
}
```

```
// --- Variables for Gesture Detection and Cooldown ---
static int lastDetectedWordIndex = -1;
static unsigned long lastGestureTime = 0;
const unsigned long gestureCooldown = 3000; //Cooldown in ms

// --- Language Selection Gesture Variables ---
static unsigned long lastLanguageGestureTime = 0;
const unsigned long languageGestureCooldown = 2000; //Cooldown for language selection gesture
const int ENGLISH_GESTURE_WORD_INDEX = -2; //special index for gesture types
const int HINDI_GESTURE_WORD_INDEX = -3;


void setup() {
  Serial.begin(115200); // Use a higher baud rate for debugging
  lcd.init();
  lcd.backlight();

  // --- 1) Power On Sequence ---
  lcd.setCursor(0, 0);
  lcd.print(F("  Powering On ")); // Print "Powering On" from Flash

  mySerial.begin(9600);
  Serial.println(F("Initializing DFPlayer Mini...")); // Print from Flash

  if (!myPlayer.begin(mySerial)) {
    Serial.println(F("DFPlayer Mini Error:"));
    Serial.println(F("1. Check connection!"));
    Serial.println(F("2. Insert SD card!"));
    while(true); // Halt if MP3 player fails
  }
  Serial.println(F("DFPlayer Mini online."));

  //myPlayer.volume(30); // Set a default volume

  // Play power-on sound (assuming 0061.mp3)
  myPlayer.play(63);
  delay(4000); // Wait for power-on sound to play

  // Clear LCD after power on message
  lcd.clear();

  // --- 2) Language Selection (Gesture-based) ---
  lcd.setCursor(0, 0);
  lcd.print(F("Select Language:")); // Print from Flash
  lcd.setCursor(0, 1);
  lcd.print(F("English / Hindi")); // Print from Flash

  bool languageSelected = false;
  while (!languageSelected) {
    // Read flex sensors
    int f1_value = readFlexSensor(flexPin_f1);
    int f5_value = readFlexSensor(flexPin_f5);

    unsigned long currentTime = millis();

    if (f5_value >= BEND_THRESHOLD && currentTime - lastLanguageGestureTime >
languageGestureCooldown) {
      // English selected
      currentLanguage = ENGLISH;
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print(F(" Language: Eng ")); // Print from Flash
      // Play English selection audio
      myPlayer.play(64);
      delay(2000); // Wait for audio
      languageSelected = true;
      lastLanguageGestureTime = currentTime; // Update cooldown timer
    } else if (f1_value >= BEND_THRESHOLD && currentTime - lastLanguageGestureTime >
languageGestureCooldown) {
      // Hindi selected
      currentLanguage = HINDI;
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print(F(" Language: Hindi")); // Print from Flash
       // Play Hindi selection audio
      myPlayer.play(62);
      delay(2000); // Wait for audio
      languageSelected = true;
      lastLanguageGestureTime = currentTime; // Update cooldown timer
    }
    // Add a small delay in the selection loop to avoid reading too fast
     delay(50);
  }

  // Clear LCD after language selection
  lcd.clear();
  // --- Set up main operation display ---
  lcd.setCursor(0, 0);
  lcd.print(F("Gesture Ready!  ")); // Standard top line text from Flash
}


void loop() {
  // --- Main Operation: Read Sensors, Detect Gestures, Output ---

  // Read all flex sensor values (You might want to use averaging here)
  /*
  int f1_value = analogRead(flexPin_f1);
  int f2_value = analogRead(flexPin_f2);
  int f3_value = analogRead(flexPin_f3);
  int f4_value = analogRead(flexPin_f4);
  int f5_value = analogRead(flexPin_f5);
  */

  int f1_value = readFlexSensor(flexPin_f1);
  int f2_value = readFlexSensor(flexPin_f2);
  int f3_value = readFlexSensor(flexPin_f3);
  int f4_value = readFlexSensor(flexPin_f4);
  int f5_value = readFlexSensor(flexPin_f5);

  // Determine which fingers are bent based on the threshold
  bool is_f1_bent = (f1_value >= BEND_THRESHOLD);
  bool is_f2_bent = (f2_value >= BEND_THRESHOLD);
  bool is_f3_bent = (f3_value >= BEND_THRESHOLD);
  bool is_f4_bent = (f4_value >= BEND_THRESHOLD);
  bool is_f5_bent = (f5_value >= BEND_THRESHOLD);

  // --- Gesture Detection Logic ---
  // Define combinations of bent fingers mapped to word indices (0-29)

  int detectedWordIndex = -1; // -1 means no word gesture detected yet


 if (is_f1_bent && !is_f2_bent && !is_f3_bent && !is_f4_bent && !is_f5_bent) {
    detectedWordIndex = 0;
  } else if (!is_f1_bent && is_f2_bent && !is_f3_bent && !is_f4_bent && !is_f5_bent) {
    detectedWordIndex = 1;
  } else if (!is_f1_bent && !is_f2_bent && is_f3_bent && !is_f4_bent && !is_f5_bent) {
    detectedWordIndex = 2;
  } else if (!is_f1_bent && !is_f2_bent && !is_f3_bent && is_f4_bent && !is_f5_bent) {
```

```
  detectedWordIndex = 3;
} else if (!is_f1_bent && !is_f2_bent && !is_f3_bent && !is_f4_bent && is_f5_bent) {
  detectedWordIndex = 4;
}

// Pair of Finger Gestures (Indices 5-14) - 10 Combinations
else if (is_f1_bent && is_f2_bent && !is_f3_bent && !is_f4_bent && !is_f5_bent) {
  detectedWordIndex = 5;
} else if (is_f1_bent && !is_f2_bent && is_f3_bent && !is_f4_bent && !is_f5_bent) {
  detectedWordIndex = 6;
} else if (is_f1_bent && !is_f2_bent && !is_f3_bent && is_f4_bent && !is_f5_bent) {
  detectedWordIndex = 7;
} else if (is_f1_bent && !is_f2_bent && !is_f3_bent && !is_f4_bent && is_f5_bent) {
  detectedWordIndex = 8;
} else if (!is_f1_bent && is_f2_bent && is_f3_bent && !is_f4_bent && !is_f5_bent) {
  detectedWordIndex = 9;
} else if (!is_f1_bent && is_f2_bent && !is_f3_bent && is_f4_bent && !is_f5_bent) {
  detectedWordIndex = 10;
} else if (!is_f1_bent && is_f2_bent && !is_f3_bent && !is_f4_bent && is_f5_bent) {
  detectedWordIndex = 11;
} else if (!is_f1_bent && !is_f2_bent && is_f3_bent && is_f4_bent && !is_f5_bent) {
  detectedWordIndex = 12;
} else if (!is_f1_bent && !is_f2_bent && is_f3_bent && !is_f4_bent && is_f5_bent) {
  detectedWordIndex = 13;
} else if (!is_f1_bent && !is_f2_bent && !is_f3_bent && is_f4_bent && is_f5_bent) {
  detectedWordIndex = 14;
}

// Triplet Finger Gestures (Indices 15-24) - 10 Combinations
else if (is_f1_bent && is_f2_bent && is_f3_bent && !is_f4_bent && !is_f5_bent) {
  detectedWordIndex = 15;
} else if (is_f1_bent && is_f2_bent && !is_f3_bent && is_f4_bent && !is_f5_bent) {
  detectedWordIndex = 16;
} else if (is_f1_bent && is_f2_bent && !is_f3_bent && !is_f4_bent && is_f5_bent) {
  detectedWordIndex = 17;
} else if (is_f1_bent && !is_f2_bent && is_f3_bent && is_f4_bent && !is_f5_bent) {
  detectedWordIndex = 18;
} else if (is_f1_bent && !is_f2_bent && is_f3_bent && !is_f4_bent && is_f5_bent) {
  detectedWordIndex = 19;
} else if (is_f1_bent && !is_f2_bent && !is_f3_bent && is_f4_bent && is_f5_bent) {
  detectedWordIndex = 20;
} else if (!is_f1_bent && is_f2_bent && is_f3_bent && is_f4_bent && !is_f5_bent) {
  detectedWordIndex = 21;
} else if (!is_f1_bent && is_f2_bent && is_f3_bent && !is_f4_bent && is_f5_bent) {
    detectedWordIndex = 22;
  } else if (!is_f1_bent && is_f2_bent && !is_f3_bent && is_f4_bent && is_f5_bent) {
    detectedWordIndex = 23;
  } else if (!is_f1_bent && !is_f2_bent && is_f3_bent && is_f4_bent && is_f5_bent) {
    detectedWordIndex = 24;
  }

  // Quadruplet Finger Gestures (Indices 25-29) - 5 Combinations
  else if (is_f1_bent && is_f2_bent && is_f3_bent && is_f4_bent && !is_f5_bent) {
    detectedWordIndex = 25;
  } else if (is_f1_bent && is_f2_bent && is_f3_bent && !is_f4_bent && is_f5_bent) {
    detectedWordIndex = 26;
  } else if (is_f1_bent && is_f2_bent && !is_f3_bent && is_f4_bent && is_f5_bent) {
    detectedWordIndex = 27;
  } else if (is_f1_bent && !is_f2_bent && is_f3_bent && is_f4_bent && is_f5_bent) {
    detectedWordIndex = 28;
  } else if (!is_f1_bent && is_f2_bent && is_f3_bent && is_f4_bent && is_f5_bent) {
    detectedWordIndex = 29;
  }
  else if (is_f1_bent && is_f2_bent && is_f3_bent && is_f4_bent && is_f5_bent) {
    detectedWordIndex = 30;
  }
  // All Five Fingers (Index 30) - 1 Combination
  else if (is_f1_bent && is_f2_bent && !is_f3_bent && is_f4_bent && is_f5_bent) {
     // This is the specific gesture you wanted to keep for index 31, now mapped to 30
    detectedWordIndex = 31;
  }



  // --- Output if a Word Gesture is Detected ---
  unsigned long currentTime = millis();

  // Check if a word gesture is detected AND it's a new gesture OR enough time has passed
  if (detectedWordIndex != -1 && (detectedWordIndex != lastDetectedWordIndex || currentTime -
lastGestureTime > gestureCooldown)) {

    // Clear the previous word on the bottom line
    lcd.setCursor(0, 1);
    lcd.print("                "); // Clear the entire second line

    // Read the word from Flash (correct language) and display it on the bottom line
    readWordFromFlash(detectedWordIndex, currentLanguage);
    lcd.setCursor(0, 1);
    lcd.print(wordBuffer);
    // Get the corresponding audio file number (language-dependent)
    uint8_t audioFileNumber = getAudioNumberForWord(detectedWordIndex, currentLanguage);

    // Play the audio file (from the root or folder 01 if you put everything there)
      myPlayer.play(audioFileNumber);



    // --- For Debugging ---
    Serial.print(F("Detected: "));
    Serial.println(wordBuffer);
    Serial.print(F("Playing audio file: "));
    Serial.println(audioFileNumber);
    // --- End Debugging ---



    // Update last detected gesture and time
    lastDetectedWordIndex = detectedWordIndex;
    lastGestureTime = currentTime;

  } else if (detectedWordIndex == -1 && lastDetectedWordIndex != -1 && currentTime -
lastGestureTime > gestureCooldown) {
      // No word gesture detected currently, and enough time has passed since the last one
      // Optionally clear the bottom line or show a "Ready" message again
      // lcd.setCursor(0, 1);
      // lcd.print("                ");
      lastDetectedWordIndex = -1; // Reset the last detected gesture state
  }



  // Add a small delay in the loop to avoid reading too fast and reduce CPU load
  delay(1000);


}


// --- Averaging function ---


int readFlexSensor(int pin) {
  int sum = 0;
  for (int i = 0; i < 10; i++) { // Take 10 readings
    sum += analogRead(pin);
    // delay(1); // Small delay between readings (can remove if needed)
  }
  return sum / 10; // Return the average}
```

Applications

- **Communication Aid for the Hearing Impaired:** The primary application is to facilitate communication between individuals who use sign language and those who do not understand it. The glove can translate hand gestures into text displayed on the LCD, enabling more seamless interaction in everyday situations.
- **Educational Tool for Sign Language Learners:** This glove can serve as a valuable tool for individuals learning sign language. By providing immediate text feedback for their gestures, it can help learners practice and reinforce their understanding of different signs.
- **Accessibility in Public and Private Settings:** The glove could be used in various settings like shops, hospitals, banks, or homes to improve accessibility for hearing-impaired individuals when interacting with people who don't know sign language.
- **Assistive Technology for Non-Verbal Communication:** While focused on sign language, the underlying technology of gesture recognition could potentially be adapted for other forms of non-verbal communication for individuals with speech impairments or other communication challenges.
- **Real-time Basic Translation:** In situations where a professional sign language interpreter is not immediately available, this glove could offer a basic level of real-time translation for a limited vocabulary, aiding in essential communication.

Scope and Impact

While the current implementation provides a foundational system, it opens the door to several promising upgrades:

- **Advanced Microcontrollers**: Incorporating more powerful boards (e.g., ESP32, Raspberry Pi) could improve speed, responsiveness, and storage capacity.
- **Artificial Intelligence (AI) Integration**: Machine learning models could be trained to recognize a broader and more complex set of gestures, including **dynamic hand motions** and **full sentence translation**, significantly enhancing the glove's real-world usability.
- **Wireless Communication**: Adding Bluetooth/Wi-Fi capabilities for integration with smartphones or computers.
- **Multilingual Expansion**: Supporting more languages to improve accessibility globally.

This project exemplifies how **low-cost, open-source components** can be combined to create meaningful, real-world solutions that promote inclusion and accessibility.

Conclusion

In conclusion, this project successfully demonstrates a functional **Sign Language to Text and Speech Glove** using an **Arduino Nano**, **flex sensors**, an **LCD display**, and an **MP3TF16P audio module**. By mapping distinct combinations of finger bends to a predefined vocabulary of **31 words for 2 different languages each**, the glove offers a simple yet effective means to translate sign language gestures into both **readable text** and **audible speech**. This can serve as a practical assistive technology to help bridge communication gaps between hearing and non-hearing individuals.