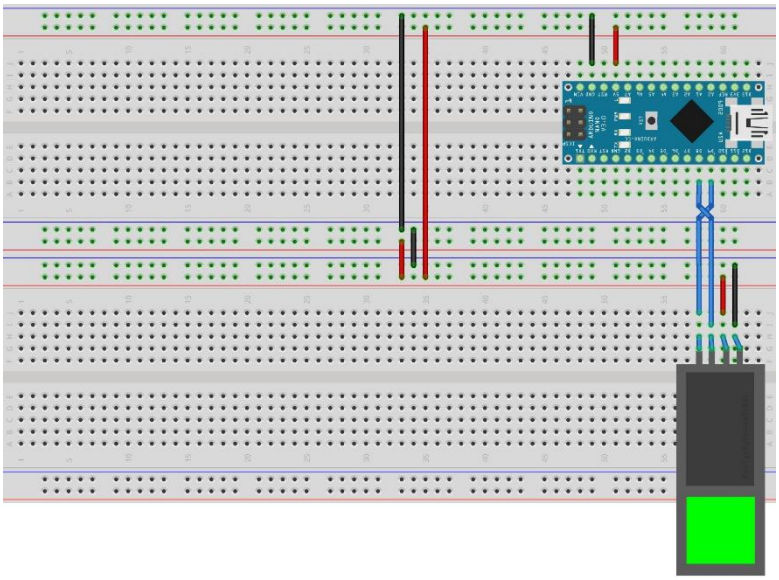
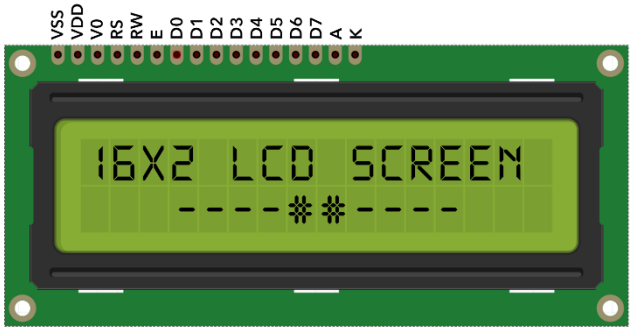
Project – Biometric Attendance System

- **Objective**: To store, and mange fingerprints and verify them for attendance system
- **Equipment**:
    - Arduino board - Nano
    - Fingerprint sensor
    - LCD
    - HC-05 Bluetooth module
    - Jumper wires.
- **Procedure**:
    1. Wire the Fingerprint sensor to the Arduino board.

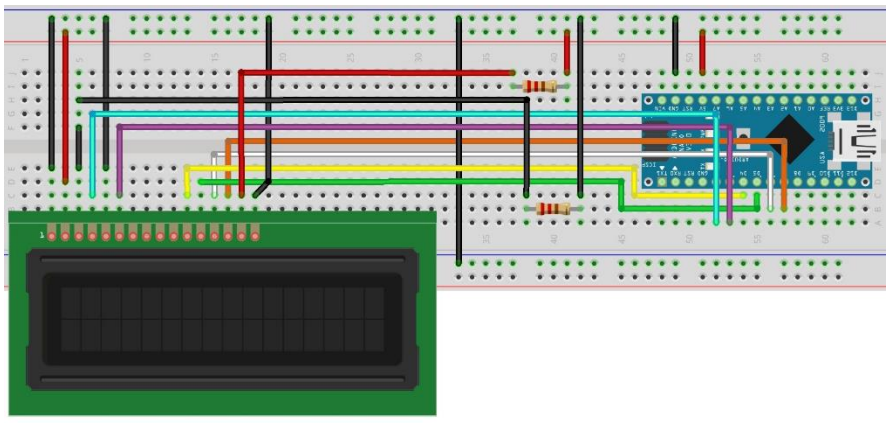
Pin Nmuber	Name	Type	Function Description
1	5V	in	Power input (DC4.2V - 6V)
2	GND	-	Signal ground. Connected to power ground
3	TXD	out	Data output. TTL logical level
4	RXD	in	Data input. TTL logical leve
5	Touch	out	Finger detection signal（maximum output current: 50mA）
6	3.3V	in	Finger detection power (DC3.3V - 5V、about 5uA)



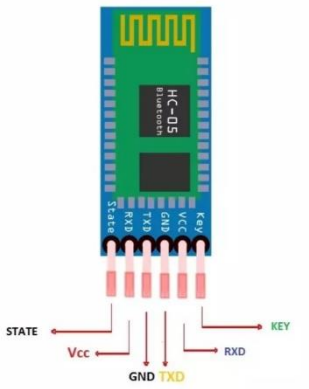
2. Interface the LCD to the Arduino Board

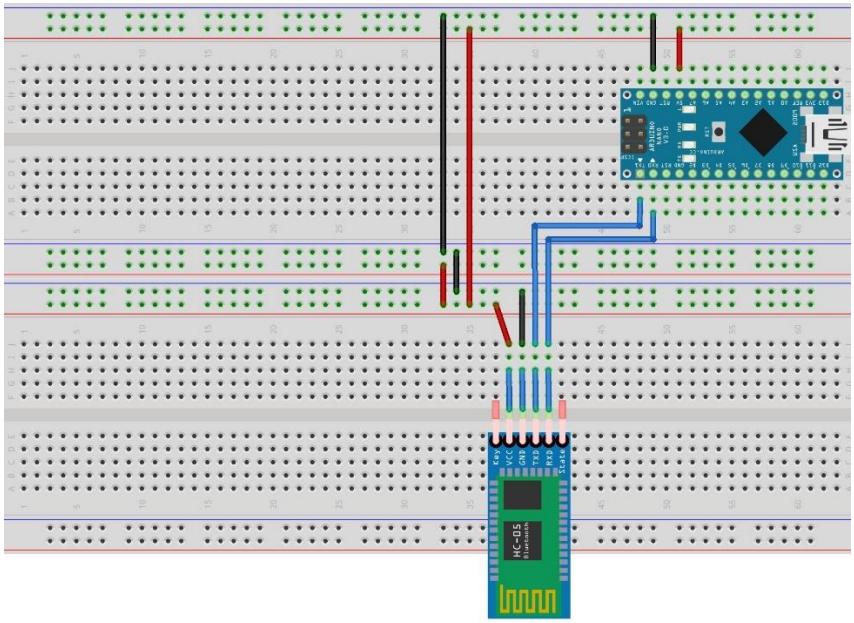


No	Symbol	Function
1	VSS	Ground
2	VDD	5V +
3	V0	Contrast
4	RS	Register
5	RW	Read/Write
6	E	Enable
7	D0	Data bus
8	D1	Data bus
9	D2	Data bus
10	D3	Data bus
11	D4	Data bus
12	D5	Data bus
13	D6	Data bus
14	D7	Data bus
15	A	Anode (5V+)
16	K	Cathode (GND)



3. Interface the HC-05 module with Arduino





- **Program:**

```
#include <LiquidCrystal.h>  // for LCD
#include <Adafruit_Fingerprint.h>
#define CODE 1234
const int rs = 2,
  en = 3,
  d4 = 4,
  d5 = 5,
```

```
  d6 = 6,
  d7 = 7;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
#if(defined(__AVR__) || defined(ESP8266)) && !defined(__AVR_ATmega2560__)
SoftwareSerial mySerial(8, 9); //check
#else
#define mySerial Serial1
#endif

Adafruit_Fingerprint finger = Adafruit_Fingerprint( & mySerial);
int mode;
uint8_t getFingerprintEnroll(uint8_t);

void Display(String line1, String line2 = "", int del = 300) {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(line1.substring(0, 16)); // Limit to 16 characters per line
  lcd.setCursor(0, 1);
  lcd.print(line2.substring(0, 16)); // Limit to 16 characters per line
  if (del) delay(del);
}

// Function to read a number from Serial
int readnumber(void) {
  int num = 0;
  while (num == 0) {
    while (!Serial.available());
    num = Serial.parseInt();
  }
  return num;
}

// Function to handle enrollment process
void enroll() {
```

```
  Display("Enroll Mode", "Enter ID:");
  int id = readnumber();
  if (id < 0) {
    mode = 2;
    return;
  }
  Display("Enrolling ID #" + String(id), "", 1000);
  while (!getFingerprintEnroll(id));

}

uint8_t getFingerprintID() {
  int p = -1;
  while (p != FINGERPRINT_OK) {
    uint8_t num = 2;

    if (Serial.available())
      num = Serial.parseInt();
    if (num != 2 && num > 0) {
      mode = num;
      return -1;
    }
    p = finger.getImage();
  }
  switch (p) {
  case FINGERPRINT_OK:
    Serial.println("Image taken");
    Display("Image taken", "", 1000);
    break;
  case FINGERPRINT_NOFINGER:
    Serial.println("No finger detected");
    Display("Print not", "Detected!! b", 1000);
    return p;
```

```
case FINGERPRINT_PACKETRECIEVEERR:
  Serial.println("Communication error");
  Display("Communication error", "", 1000);
  return p;
case FINGERPRINT_IMAGEFAIL:
  Serial.println("Imaging error");
  Display("Imaging error", "", 1000);
  return p;
default:
  Serial.println("Unknown error");
  Display("Unknown error", "", 1000);
  return p;
}
p = finger.image2Tz();
switch (p) {
case FINGERPRINT_OK:
  Serial.println("Image converted");
  break;
case FINGERPRINT_IMAGEMESS:
  Serial.println("Image too messy");
  Display("Image too messy", "", 500);
  return p;
case FINGERPRINT_PACKETRECIEVEERR:
  Serial.println("Communication error");
  Display("Communication error", "", 1000);
  return p;
case FINGERPRINT_FEATUREFAIL:
  Serial.println("Could not find fingerprint features");
  Display("Bad Read", "Retry", 1000);
  return p;
case FINGERPRINT_INVALIDIMAGE:
  Serial.println("Could not find fingerprint features");
  Display("Bad Read", "Retry", 1000);
  return p;
```

```
  default:
    Serial.println("Unknown error");
    Display("Unknown error", "", 1000);
    return p;
  }
  p = finger.fingerSearch();
  if (p == FINGERPRINT_OK) {
    Serial.println("Found a print match!");
    Display("Match Found!", "", 1000);
  } else if (p == FINGERPRINT_PACKETRECIEVEERR) {
    Serial.println("Communication error");
    Display("Communication error", "", 1000);
    return p;
  } else if (p == FINGERPRINT_NOTFOUND) {
    Serial.println("Did not find a match");
    Display("Match Not Found!", "", 1000);
    return p;
  } else {
    Serial.println("Unknown error");
    Display("Unknown error");
    return p;
  }
  Serial.print("Found ID #");
  Serial.print(finger.fingerID);

  Serial.print(" with confidence of ");
  Serial.println(finger.confidence);
  Display("Found ID #" + String(finger.fingerID), "Confidence : " + String(finger.confidence) + "%", 1000);
  return finger.fingerID;

}

void attendance() {
  Display("Attendance Mode", "Waiting...   (m)");
```

```
  getFingerprintID();
  delay(50);
}

void setup() {
  Serial.begin(19200); // Initialize serial communication
  lcd.begin(16, 2); // Initialize the LCD with 16x2 dimensions
  finger.begin(57600);

  Display("Setup Complete");
  delay(100);
  if (finger.verifyPassword()) {
    Serial.println("Found fingerprint sensor!");
    Display("Sensor Found ", "", 1000);
  } else {
    Serial.println("Did not find fingerprint sensor :(");
    Display("Sensor Not", "Found!!");
    while (1) {
      delay(1);
    }

  }

  //setMode();
  mode=2;    //Default to attendance mode
}

void setMode() {
  Display("MODE : ", "", 0);

  while (!Serial.available());
  int temp = Serial.read();
  if (temp != 10)
    mode = temp - 48;
```

```
}

void loop() {
  Serial.print(mode);
  switch (mode) {
  case 1:
    enroll();
    break;
  case 2:
    attendance();
    break;
  case 3:
    Delete();
    break;
  case 4:
    EmptyDB();
    break;
  }

}

uint8_t getFingerprintEnroll(uint8_t id) {
  int p = -1;
  Serial.print("Waiting for valid finger to enroll as #");
  Serial.println(id);
  Display("Waiting for Finger", "#" + String(id));
  while (p != FINGERPRINT_OK) {
    p = finger.getImage();
    switch (p) {
    case FINGERPRINT_OK:
      Serial.println("Image taken");
      Display("Image taken", "", 400);
      break;
    case FINGERPRINT_NOFINGER:
```

```
      Serial.print(".");
      break;
    case FINGERPRINT_PACKETRECIEVEERR:
      Serial.println("Communication error");
      Display("Communication", "error", 1000);
      break;
    case FINGERPRINT_IMAGEFAIL:
      Serial.println("Imaging error");
      Display("Imaging error", "", 1000);
      break;
    default:
      Serial.println("Unknown error");
      Display("Unknown error", "", 1000);
      break;
    }
  }

  // OK success!

  p = finger.image2Tz(1);
  switch (p) {
  case FINGERPRINT_OK:
    Serial.println("Image converted");
    break;
  case FINGERPRINT_IMAGEMESS:
    Serial.println("Image too messy");
    Display("Messy Image", "Retry", 1000);
    return p;
  case FINGERPRINT_PACKETRECIEVEERR:
    Serial.println("Communication error");
    Display("Comm. error", "Retry", 1000);
    return p;
  case FINGERPRINT_FEATUREFAIL:
    Serial.println("Could not find fingerprint features");
```

```
  Display("Bad Read", "Retry", 1000);
  return p;
case FINGERPRINT_INVALIDIMAGE:
  Serial.println("Could not find fingerprint features");
  Display("Bad Read", "Retry", 1000);
  return p;
default:
  Serial.println("Unknown error");
  Display("Unknown error", "Retry", 1000);
  return p;
}

Serial.println("Remove finger");
Display("Remove finger", "", 2000);
p = 0;
while (p != FINGERPRINT_NOFINGER) {
  p = finger.getImage();
}
Serial.print("ID ");
Serial.println(id);
Display("ID #" + String(id));
p = -1;
Serial.println("Place same finger again");
Display("Place Finger", "Again", 500);
while (p != FINGERPRINT_OK) {
  p = finger.getImage();
  switch (p) {
  case FINGERPRINT_OK:
    Serial.println("Image taken");
    Display("Image taken", "", 400);
    break;
  case FINGERPRINT_NOFINGER:
    Serial.print(".");
    break;
```

```
  case FINGERPRINT_PACKETRECIEVEERR:
    Serial.println("Communication error");
    Display("Comm. Error", "", 1000);
    break;
  case FINGERPRINT_IMAGEFAIL:
    Serial.println("Imaging error");
    Display("Imaging Error", "", 1000);
    break;
  default:
    Serial.println("Unknown error");
    Display("Unknown Error", "", 1000);
    break;
  }
}

// OK success!

p = finger.image2Tz(2);
switch (p) {
case FINGERPRINT_OK:
  Serial.println("Image converted");
  break;
case FINGERPRINT_IMAGEMESS:
  Serial.println("Image too messy");
  Display("Messy Image", "Retry", 1000);
  return false;
case FINGERPRINT_PACKETRECIEVEERR:
  Serial.println("Communication error");
  Display("Comm. error", "Retry", 1000);
  return false;
case FINGERPRINT_FEATUREFAIL:
  Serial.println("Could not find fingerprint features");
  Display("Bad Read", "Retry", 1000);
  return false;
```

```
case FINGERPRINT_INVALIDIMAGE:
  Serial.println("Could not find fingerprint features");
  Display("Bad Read", "Retry", 1000);
  return false;
default:
  Serial.println("Unknown error");
  Display("Unknown error", "Retry", 1000);
  return false;
}

// converted!
Serial.print("Creating model for #");
Serial.println(id);
Display("Creating Image", "#" + String(id), 1500);
p = finger.createModel();
if (p == FINGERPRINT_OK) {
  Serial.println("Prints matched!");
  Display("Prints Matched!", "", 600);
} else if (p == FINGERPRINT_PACKETRECIEVEERR) {
  Serial.println("Communication error");
  Display("Comm. Error", "", 1000);
  return false;
} else if (p == FINGERPRINT_ENROLLMISMATCH) {
  Serial.println("Fingerprints did not match");
  Display("Fingerprint not", "Matched", 1000);
  return false;
} else {
  Serial.println("Unknown error");
  Display("Unknown error", "", 1000);
  return false;
}

Serial.print("ID ");
Serial.println(id);
```

```
  Display("ID #" + String(id));
  p = finger.storeModel(id);
  if (p == FINGERPRINT_OK) {
    Serial.println("Stored!");
    Display("Stored!", "", 600);
  } else if (p == FINGERPRINT_PACKETRECIEVEERR) {
    Serial.println("Communication error");
    Display("Comm. Error", "", 1000);
    return false;
  } else if (p == FINGERPRINT_BADLOCATION) {
    Serial.println("Could not store in that location");
    Display("Location not", "Available", 1000);
    return false;
  } else if (p == FINGERPRINT_FLASHERR) {
    Serial.println("Error writing to flash");
    Display("Couldn't write", "to flash", 1000);
    return false;
  } else {
    Serial.println("Unknown error");
    Display("Unknown Error", "", 1000);
    return false;
  }
  mode = 2;
  return true;
}
void Delete() {
  Display("Delete Mode", "Enter ID:");
  int id = readnumber();
  if (id < 0) {
    mode = 2;
    return;
  }
  Display("Deleting ID #" + String(id), "", 1000);
```

```
  Display("Enter Code : ");
  int x = readnumber();
  if (x == CODE) {
    Display("Deleting", "...", 1500);
    int p = finger.deleteModel(id);

    if (p == FINGERPRINT_OK) {
      Display("ID #" + String(id), "Deleted", 1000);
      mode = 2;
    } else {
      Display("Error Deleting", "ID #" + String(id), 1000);
      mode = 2;
    }
  } else {
    Display("Wrong Code!", "", 1000);
    mode = 2;
    return;
  }
}
void EmptyDB() {
  Display("Deleting", "Database", 1000);
  Display("Enter Code : ");
  int x = readnumber();
  if (x == CODE) {
    Display("Deleting", "...", 1500);
    finger.emptyDatabase();
    Display("Database", "Emptied", 1000);
    mode = 2;
  } else {
    Display("Wrong Code!", "", 1000);
    mode = 2;
    return;
  }
}
```

- **Description of Modes:**

1. **Enroll Mode:**

   This mode is used to enroll new fingerprints into the sensor.
   There is a prompt to enter an id to give to the new finger in the database. At this screen, if an negative number is entered, the model resets to Attendance mode which is also the default mode, the user needs to scan the finger twice for it to be registered successfully.

2. **Attendance Mode:**

   This mode is the default mode of the model, used to verify fingerprint and grant attendance.
   There is a prompt to put the finger on the sensor. At this screen, the lower right corner shows "(m)", this denotes that the model will take input to change modes, if provided. On successful recognition of a finger the screen shows the ID and the confidence %. This data can be sent to further monitoring systems easily to maintain attendance records.

3. **Delete Mode:**

   This is a mode for administrators, this mode is used to delete single prints from the sensor database, so that old, unrequired or faulty prints can be deleted, the mode prompts for inputting an id to delete, then asks for the passcode, for simplicity the code is set to 1234, this can be changed n the code easily, on entering the correct code, the print will be deleted.

4. **Empty Database:**
   This mode/command can be used to empty the entire database of prints stored in the sensor, this mode also requires authentication via a passcode to execute.

Figure 15: Complete circuit diagram