

# Lab 3 report

CSE121

Inhle Cele

## Contents

<b>Lab 3 Overview</b> .....	1
<b>Lab3.1: Solder pins</b> .....	1
ChatGPT chat export.....	2
<b>Lab3.2: Display library</b> .....	2
ChatGPT chat export.....	3
<b>Lab3.3: Integrate humidity and temperature</b> .....	9
ChatGPT chat export.....	9
<b>Conclusion</b> .....	9

## Lab 3 Overview

- **Objective:** To develop a C++ library on the ESP32 for controlling a display over I2C and using the same interface to read and display data from a temperature/humidity sensor.
- **Overview:** three key tasks:
  - Soldering pins to enable I2C and Vdd control.
  - Creating a display library compatible with the ESP32.
  - Integrating the temperature and humidity sensor data into the display.

## Lab3.1: Solder pins

**Task:** Solder the pin headers onto the ESP32 board to enable I2C and Vdd connections required for the display.

### Materials and Tools:

- ESP32 board
- Pin headers
- Soldering iron and solder

### Steps:

1. Go to the BELs soldering workshop
2. Align pin headers with the ESP32 board.
3. Carefully solder each pin to ensure a secure and functional connection.
4. Perform a continuity test to verify all connections are intact.

**Verification:** Asked the tutor and got the part checked off

## ChatGPT chat export:

No ChatGPT assistance needed

## Lab3.2: Display library

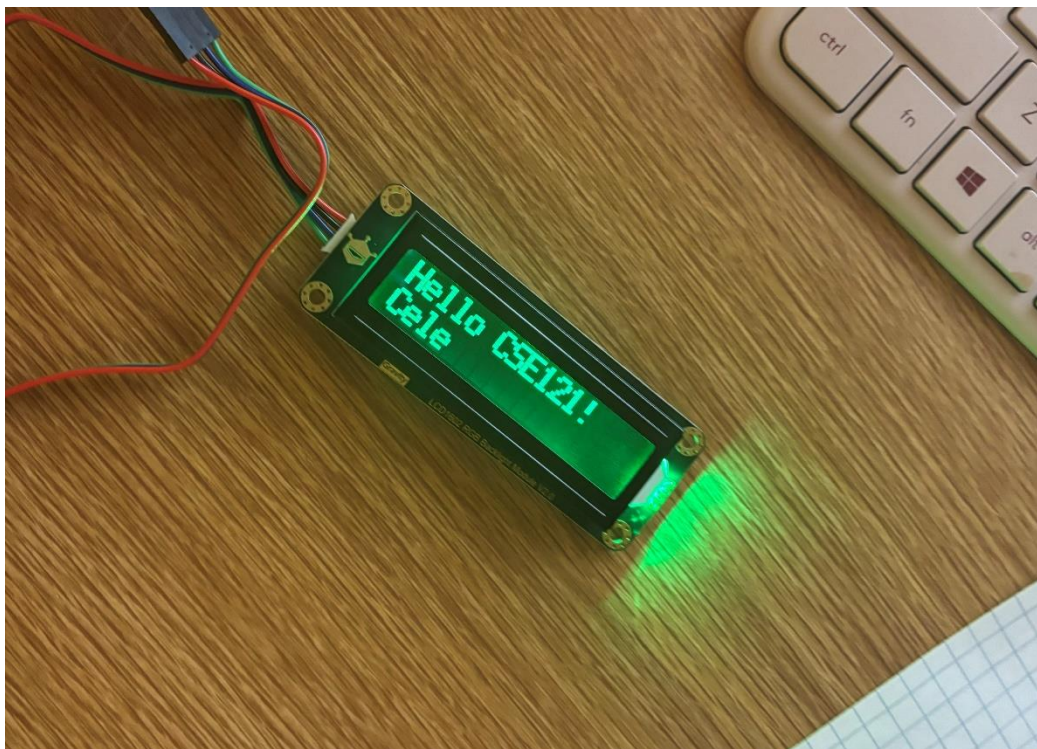
**Objective:** Adapt the DFRobot\_RGBLCD1602 library for ESP32 compatibility, maintaining the same API structure but replacing unsupported Arduino calls with ESP32 equivalents.

### Methodology:

- **Library Setup:** The Arduino-based DFRobot\_RGBLCD1602 library was cloned to start the development.
- **Modifications:**
  - Removed Print functionality.
  - Replaced Wire library calls with ESP32-specific I2C handling.
- **API Consistency:** The API (from DFRobot\_LCD.h) was preserved for consistency with the original library.

**Implementation:** The primary function setup to initialize and configure the display to show “Hello CSE121” on the first line and my last name on the second line:

**Challenges and Solutions:** Adapting Arduino code for the ESP32 environment required debugging I2C communication and testing different ESP32 I2C functions to replace the Wire library.



## ChatGPT chat export:

<https://chatgpt.com/share/672300f3-5a40-8008-8be6-5e3f0cb5cf83>

<https://chatgpt.com/share/6722ffdb-5740-8008-82f6-997b61cd4d64>

### Chat Export

User: /\*!

\* @file DFRobot\_LCD.cpp

\* @brief DFRobot's LCD

\* @n High Accuracy Ambient Light Sensor

\*

\*/

ChatGPT: To adapt this `DFRobot\_LCD.cpp` code for the ESP32C3, ...

User: It shouldnt have a wire or arduino it should be compatible with esp32c3 and in c++

ChatGPT: To make this code compatible with I2C on the ESP32C3...

User: make it compatible with I2C

ChatGPT: The `Wire` library in Arduino is typically used to...

User: create the updated code

ChatGPT: Here?s the complete code with the I2C compatibility...

User:

```
void DFRobot_LCD::begin(uint8_t cols, uint8_t lines, uint8_t dotsize)
```

```
{
```

```
    if (lines > 1) {
```

```
        _showfunction |= LCD_2LINE;
```

```
    }
```

```
    _numlines = lines;
```

```
    _currline = 0;
```

```
    ///< for some 1 line displays you can select a 10 pixel high font
```

```
    if ((dotsize != 0) && (lines == 1)) {
```

```
        _showfunction |= LCD_5x10DOTS;
```

```
    }
```

```
    ///< SEE PAGE 45/46 FOR INITIALIZATION SPECIFICATION!
```

```
    ///< according to datasheet, we need at least 40ms after power rises above 2.7V
```

```
    ///< before sending commands. Arduino can turn on way before 4.5V so we'll wait 50
```

```
    delay(50);
```

```
    ///< this is according to the hitachi HD44780 datasheet
```

```
    ///< page 45 figure 23
```

```
    ///< Send function set command sequence
```

```
    command(LCD_FUNCTIONSET | _showfunction);
```

```
delay(5); // wait more than 4.1ms
```

```
///  
//< second try
```

```
command(LCD_FUNCTIONSET | _showfunction);
```

```
delay(5);
```

```
///  
//< third go
```

```
command(LCD_FUNCTIONSET | _showfunction);
```

```
///  
//< turn the display on with no cursor or blinking default
```

```
_showcontrol = LCD_DISPLAYON | LCD_CURSOROFF | LCD_BLINKOFF;
```

```
display();
```

```
///  
//< clear it off
```

```
clear();
```

```
///  
//< Initialize to default text direction (for romance languages)
```

```
_showmode = LCD_ENTRYLEFT | LCD_ENTRYSHIFTDECREMENT;
```

```
///  
//< set the entry mode
```

```
command(LCD_ENTRYMODESET | _showmode);
```

```
///  
//< backlight init
```

```
setReg(REG_MODE1, 0);
```

```
///  
//< set LEDs controllable by both PWM and GRPPWM registers
```

```

setReg(REG_OUTPUT, 0xFF);

///< set MODE2 values

///< 0010 0000 -> 0x20 (DMLNK to 1, ie blinky mode)

setReg(REG_MODE2, 0x20);

setColorWhite();

}

```

Assistant:

The `begin` function in your `DFRobot\_LCD` class is quite detailed, but I'll break down the process and point out where adjustments may improve initialization consistency:

### 1. **Initialization Timing**:

The sequence is correct in waiting for the display to stabilize (with delays after each function command), as recommended by the HD44780 datasheet.

### 2. **Explanation of Key Sections**:

- **Function Set**: This command sets up the display mode (number of lines and font size).

Sending this three times ensures that the display is fully configured.

- **Display Control**: The `\_showcontrol` flags determine whether the display is on and if the cursor is visible or blinking.

- **Entry Mode**: `\_showmode` sets the text direction and whether characters shift as you type.

- **Backlight Initialization**: Uses `setReg` to set the registers controlling the backlight and LED modes, with `setColorWhite` setting a default color.

### 3. **Error Handling and Robustness**:

If you're noticing instability in initializing certain LCDs, try adding slight delays after each `command` call (e.g., `delayMicroseconds(100);`), which can stabilize the communication.

Here's a slightly enhanced version with additional comments:

```
```cpp
void DFRobot_LCD::begin(uint8_t cols, uint8_t lines, uint8_t dotsize)
{
    // Configure function based on line count and font size
    if (lines > 1) {
        _showfunction |= LCD_2LINE;
    }

    _numlines = lines;
    _currline = 0;

    // Set font size if 1-line display and dotsize not 0
    if ((dotsize != 0) && (lines == 1)) {
        _showfunction |= LCD_5x10DOTS;
    }

    // Wait for power stabilization
    delay(50);

    // Send function set command sequence, retrying three times
    command(LCD_FUNCTIONSET | _showfunction);
    delay(5); // wait for 4.1ms
}
```



```

command(LCD_FUNCTIONSET | _showfunction);

delay(5);

command(LCD_FUNCTIONSET | _showfunction);


// Set initial display settings (display on, no cursor or blinking)
_showcontrol = LCD_DISPLAYON | LCD_CURSOROFF | LCD_BLINKOFF;

display();


// Clear display

clear();


// Set default text direction and entry mode

_showmode = LCD_ENTRYLEFT | LCD_ENTRYSHIFTDECREMENT;

command(LCD_ENTRYMODESET | _showmode);


// Initialize backlight and LEDs

setReg(REG_MODE1, 0);      // Basic mode

setReg(REG_OUTPUT, 0xFF);  // Allow control via PWM and GRPPWM

setReg(REG_MODE2, 0x20);   // Enable blinking mode


// Set default color

setColorWhite();

}

...

```

This enhanced version doesn't fundamentally change the logic but adds further comments and a

structure to clarify each step in the initialization. Let me know if this aligns with your hardware requirements or if any further adjustments are needed!

## Lab3.3: Integrate humidity and temperature

**Objective:** Read and display temperature and humidity values from the sensor, updating every second on the LCD.

**Implementation:** The code below reads temperature and humidity, displaying temperature on the first line and humidity on the second line, refreshing every second:

```
    vTaskDelay(pdMS_TO_TICKS(100));

    float temp = read_temperature(); // Read temperature
    float hum = read_humidity();     // Read humidity

    if (temp != -1 && hum != -1) {
        char temp_str[16], hum_str[16];
        sprintf(temp_str, "Temp: %.2f C", temp);
        sprintf(hum_str, "Hum: %.2f %%", hum);

        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.printstr(temp_str); // Display temperature
        lcd.setCursor(0, 1);
        lcd.printstr(hum_str); // Display humidity
    }

    sleep_shtc3(); // Put the sensor to sleep
    vTaskDelay(pdMS_TO_TICKS(2000)); // Update every 2 se
```

**Challenges and Solutions:** Ensuring compatibility between the sensor and the display update rates was resolved by managing delays and checking I2C communication stability.

### ChatGPT chat export:

Here I just asked chatgpt to convert my lab 2.3 code to cpp and added my LCD display code

## Conclusion

Each task of the lab was completed successfully:

1. **Soldering:** Pin headers were soldered securely, enabling I2C and Vdd control for the display.
2. **Display Library Development:** The library was adapted for ESP32, maintaining API consistency with DFRobot\_LCD.h.

3. **Sensor Integration:** Temperature and humidity values were successfully read and displayed on the LCD with a one-second update interval.

**Key Learnings:** This lab reinforced knowledge on I2C communication with ESP32, modifying libraries, and embedding sensor data display in C++.