Fundamentals of Applied Microcontrollers Laboratory Manual

Seth McNeill

Edition Fall 2022 (v0.5)
2022 October 04

Contents

Chapter 1

Introduction

This book is the accompanying lab manual to a class introducing microcontrollers to upper division, non-electrical engineering undergraduate students who have taken some C programming.

If you find this useful, please let me know. If you find any errors, areas that need improvement, or have any improvements to add please let me know.

The class textbook/manual is here.

1.1 License

Chapter 2

Arduino Startup

2.1 Installing the IDE

We want to try installing the IDE at least two different ways. First, on the lab computer, then on your personal computers if you have them. Both lab partners should try installing the software on the lab computer, each with their own login.

2.1.1 Lab Computer

This method may also work on your personal computers.

1. On the search bar, type in Microsoft Store.
2. Click on Microsoft Store
3. In the store search, type arduino
4. Arduino IDE App should appear, click on it
5. Click on Get
6. You do not need to sign into Microsoft to make this install work even if prompted
7. Once it has installed, run the Arduino IDE app.
8. It should load up with a window that looks like Figure 2.1.

9. Try rebooting the computer, logging in and seeing if the Arduino app is still present.

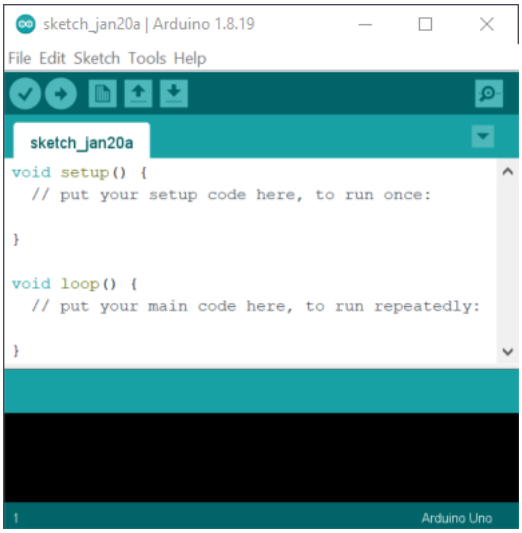

Figure 2.1: This is what the Arduino IDE should load up to.

2.1.2 Personal Computer

1. Go to software download page: https://www.arduino.cc/en/software
2. Download the Windows ZIP file (not the first link or the app)
3. Open the zip file and copy the folder inside (arduino-1.8.19 as of this writing) into your One Drive folder. This may take a while. If you are on your own computer, you can use any of the programs.
4. Once that transfer finishes, go into the folder and run arduino.exe. Windows will try to save you, but if you click More Info you can click Run Anyway.
5. Windows Defender Firewall will also complain. Uncheck the box that is checked and/or click Cancel.
6. It should load up with a window that looks like Figure 2.1.

2.2 Testing the Setup

2.2.1 Installing the Board Drivers

1. In order to get it to connect correctly to your board, you need to install the Arduino Nano Connect RP2040 board.

   (a) Navigate to Tools→Board: "Arduino Uno" (or similar)→Boards Manager

   (b) It should load as shown in Figure 2.2.

Figure 2.2: This what the Boards Manager loads up to.

   (c) In the search bar, type "arduino nano connect" (without the quotes)

   (d) The first item should be Arduino Mbed OS Nano Boards and should list the Arduino Nano RP2040 Connect.

   (e) Move your cursor over it and it should show an Install button. Click it to install the board library.

   (f) Wait for it to finish.

   (g) While you are waiting, plug your Nano Connect into your computer and let it install it.

(h) As it finished, I received a User Account Control warning asking if I wanted to let dpinst-amd64.exe make changes to my device. I said yes.

(i) Next it asked me if I wanted to install Arduino Universal Serial Bus devices. Again, click to Install.

(j) It popped up again and I clicked Install again. Now it should say that the Arduino Mbed OS Nano Boards has been installed.

(k) Close the Boards Manager.

2.2.1.1 Testing the Setup

This section can be done on either (or both) computers. The results from one run (between both lab partners) is all that needs to be turned in.

1. Now go to Files→Examples→01.Basics→Blink.
2. This will open another window with the Blink program.
3. Go to Tools→Board→Arduino Mbed OS Nano Boards and select the Arduino Nano RP2040 Connect
4. Go to Tools→Port and select the COM that isn't COM1 (mine showed up as COM5)
5. Click the right arrow under the word edit in the menu to Uplaod the sketch to the Arduino board.
6. It should say "Compiling sketch..." in the lower left and show a progress bar on the lower right.
7. Then it should switch to Uploading... and finally Done Uploading.
8. An orange light near the USB port on your board should be blinking.
9. Congratulations! You have programmed your board!
10. Now look in the program for the two delay statements. Try changing the values inside the parentheses and re-uploading it. Does the blinking change?

11. In order to save files and have it portable, you need to change the directory where the Arduino IDE stores it's sketchbooks
    (a) Go to File→Preferences
    (b) Change the Sketchbook location to your OneDrive and a folder named arduino (lowercase is good)
    (c) My OneDrive was in C:\Users\mcneils2\OneDrive - Embry-Riddle Aeronautical University\arduino
12. Now try saving the blink sketch with your changed values.
13. Demonstrate your working blink and it's storage location to your instructor/TA
14. Here are some other Examples to test:
    (a) Basics → fade: change the variable led to be LED_BUILTIN, watch the red/orange LED pulse
    (b) Digital → DigitalInputPullup: Change the first pinMode call to use A0 instead of 2. The same for the digitalRead command (2→A0). Press the right button (SW1) and see the LED blink. Note that this program isn't written as well as the others since you have to change a number in two places. Could you rewrite it better?
15. Finally, create a sketch called `getIDs` using the code at https://github.com/semcneil/CEC325Examples/blob/main/getIDs/getIDs.ino
16. You will need to install two libraries to make this script run.
    (a) Go to Sketch → Include Library → Manage Libraries
    (b) Install the ArduinoECCX08 and OneWireNG libraries
17. Run `getIDs` and submit the results in the end of lab Canvas quiz.

2.3 Turn In

1. Make sure that the TA or instructor has signed off on your modified blink sketch.

2. Fill out the end of lab quiz prior to leaving. Note that it includes asking you for the output of the `getIDs` sketch.

Chapter 3

Multiplexer, LED Display, Binary, HEX

3.1 Purpose

The goal of this lab is to use binary and hexadecimal numbers in an applied setting. This is achieved by having you (the student) use the Arduino Nano Connect RP2040 to setup the PCA9535 I^2C Input/Output (IO) port chip to drive a 4-digit LED display.

3.2 Resources

1. PCA9535 datasheet
2. LED display datasheet
3. PCA9535 library (McNeill version)
4. PCB Schematic and Layout - see class manual in the Arduino Startup → Schematics and PCB section

3.3 Procedure

3.3.1 Add PCA95x5 library

1. Open the Arduino IDE

2. Open preferences
3. Note the Sketchbook location
4. Go to the website for the PCA9535 library (see 3.2 Resources section)
5. Click the green Code button and then Download ZIP
6. Save the zip file to your Downloads folder or somewhere else you can find it again
7. In the Arduino IDE select Sketch → Include Library → Add .ZIP Library
8. Select the zip file you downloaded earlier
9. Wait for the IDE to say "Library added to your libraries"
10. Check that the library is indeed installed one of two ways:
    (a) Go to Sketch → Include Libraries and look for PCA95x5
    (b) Go to File → Examples and look for the PCA95x5 examples
11. The PCA95x5 library is now added successfully.

3.3.2 Turn on some LEDs

An example script is shown in Listing 3.1. Use this as a starting point for your code. Be sure to change the header information to include all lab partner names and the correct date. This script displays an 8. in the first digit and zeros in the rest. Note the clearing lines to make it so that there is not ghosting of numbers to the right of where they are displayed.

1. Run the script in Listing 3.1 to make sure it runs
2. Change the script so that it displays four consecutive numbers such as 1234. You can use any 4 consecutive, single digit numbers.
3. Demonstrate your sketch working to a TA or instructor.

```
/* 20220907LEDsClass1.ino
*
* Demo of LED display in class.
*
* Seth McNeill
* 2022 September 07
*/

#include <PCA95x5.h>  // include library

PCA9535 LEDmux;  // create instance (object) of
library

void setup() {
  Serial.begin(115200);
  while(!Serial);
  Serial.println("Starting...");

  // initialize object
  Wire.begin();
  LEDmux.attach(Wire, 0x21);
  LEDmux.polarity(PCA95x5::Polarity::ORIGINAL_ALL);
  uint16_t mux_direction = 0;
  mux_direction = 0x0010;  // mostly outputs
  LEDmux.direction(mux_direction);

  Serial.println("Everything setup");
}


void loop() {
  int delayTime = 0;
  // use object
  LEDmux.write(0x000F);  // required to remove
ghosting on other digits
  LEDmux.write(0xFF0E);
  delay(delayTime);
  LEDmux.write(0x000F);  // required to remove
```

```
  ghosting on other digits
    LEDmux.write(0x3F0D);
    delay(delayTime);
    LEDmux.write(0x000F);  // required to remove
  ghosting on other digits
    LEDmux.write(0x3F0B);
    delay(delayTime);
    LEDmux.write(0x000F);  // required to remove
  ghosting on other digits
    LEDmux.write(0x3F07);
    delay(delayTime);
  }
```

Listing 3.1: This listing is a starting point for driving the LED display. This sketch may also be available on Canvas.

3.3.3 Count

Make a new sketch, based off your first sketch (meaning copy and paste it into the new sketch). This new sketch should count up from 0 to 9 (or 0000 to 0009) incrementing once per second.

3.3.4 Extra credit

Have your counting system count higher than 9.

3.4 Turn In

Turn in the following:

1. A video of your board counting that includes both of your faces and you saying your names.
2. A PDF of your first sketch that displays 4 consecutive numbers.
3. A PDF of your second sketch that counts.
4. .ino versions of both sketches.

5. Fill out the end of lab quiz prior to leaving. Note that it includes asking you for the output of the `getIDs` sketch.

Chapter 4

Buttons and Serial (UART)

4.1 Purpose

The goal of this lab is to gain a better understanding of the serial interface between the board and the computer, get the buttons all working, and add a few other fun interfaces.

Note that the most time consuming part of this lab in 2022 was getting the non-SW1 switches working.

4.1.1 Serial Library

4.1.1.1 Initialization

So far we have used the Serial library without much explanation of how to use it best. In the sketches we have used we just have used the two lines shown in Listing 4.1 to start Serial.

```
Serial.begin(115200);
while(!Serial);
```

Listing 4.1: If serial is required for a sketch this method of starting Serial blocks until a serial monitor is started.

However, if a serial connection to the computer is not required, the code in Listing 4.1 prevents the rest of the sketch from executing. The Serial library does take some time to start so a simple solution is to just delay for a few seconds after calling `Serial.begin()` as shown in Listing 4.2.

```
Serial.begin(115200);
delay(3000);
```

Listing 4.2: If serial is not required for a sketch this method of starting Serial waits a bit in hopes it connects.

If you want to have the program quit waiting as soon as the Serial connects but not delay forever like in Listing 4.1, you can keep checking for a serial connection a fixed number of times as shown in Listing 4.2.

```
const int maxNoSerial = 300;
int noSerialCount = 0;
while(!Serial && noSerialCount < maxNoSerial) {
  delay(10);
  noSerialCount++;
}
```

Listing 4.3: This snippet tries connecting to Serial a fixed number of times so that it will delay less than Listing 4.2 if a serial connection exists.

4.1.1.2 print vs println

If you want to print a string with an endline, use `Serial.println()`. This is handy for statements like `Serial.println("Starting...")`. However, if you want to print out the value of variables you often don't want newline at the end of the print. For this you use `Serial.print()`. Some examples are shown in Listing 4.4.

```
Serial.print("Number of Names: ");
Serial.println(nNames);
\dots
Serial.print(F("You're connected to the network,
IP = "));
Serial.println(WiFi.localIP());
```

Listing 4.4: This snippet shows using print and println

4.1.1.3 Reading data from the Serial

The serial connection goes both ways. Information can be sent from the computer to your board. Listing 4.5 is an example of using input from the serial

port. The call to `Serial.available()` tells the sketch whether or not any characters have been received by the serial port. If characters have been received by the serial port, `Serial.read()` reads them in one character at a time. The `switch` function allows different responses depending on what character is received.

Note the layout of the sketch. It is important to define the pins for accessories so that they can be used later.

```
/*  CEC325-SerialRcv.ino
 *
 *  Demonstrates receiving and reacting to serial
input.
 *  Uses the ERAU CEC325 board v0.3 which has an
 *  Arduino Nano RP2040 Connect on board.
 *
 *  Seth McNeill
 *  2022 February 09
 *  2022 September 20 modified for v0.5 board
 *
 *  This code in the public domain.
 */

#include <WiFiNINA.h>  // for RGB LED and A7 (right
 button)

// pin definitions:
#define RIGHT_BUTTON_PIN  A0
#define BUZZ_PIN          2 // buzzer


void setup() {
  // initialize serial:
  Serial.begin(115200);
  while(!Serial) delay(100); // wait for serial to
begin
  Serial.println("Starting...");

  // make the pins outputs:
```

```
  pinMode(LEDR, OUTPUT);  // WiFiNINA RGB LED red
  pinMode(LEDG, OUTPUT);  // WiFiNINA RGB LED green
  pinMode(LEDB, OUTPUT);  // WiFiNINA RGB LED blue
  pinMode(RIGHT_BUTTON_PIN, INPUT);
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(BUZZ_PIN, OUTPUT);  // buzzer pin is
output
}

void loop() {
  while(Serial.available() > 0) {  // characters
have been received
    char inChar = Serial.read();
    switch(inChar) {
      case 'a': digitalWrite(LED_BUILTIN,HIGH);
break;
      case 'A': digitalWrite(LED_BUILTIN,LOW);
break;
      case 'b': digitalWrite(LEDB, HIGH); break;
      case 'r': digitalWrite(LEDR, HIGH); break;
      case 'g': digitalWrite(LEDG, HIGH); break;
      case '+': add(); break;
      case 'o': ledsOff(); break;
      case 'z': tone(BUZZ_PIN, 1000, 100); break;
      case '\n': break;
      default: Serial.print("Unknown character: ");
 Serial.println(inChar);
    }
  }
}

// turns off all 4 LEDs
void ledsOff() {
  digitalWrite(LED_BUILTIN,LOW);
  digitalWrite(LEDR,LOW);
  digitalWrite(LEDG,LOW);
  digitalWrite(LEDB,LOW);
}
```

```
// Adds characters received subsequent to +
void add() {
  Serial.println("Adding single digit numbers");
  int a = Serial.read() - 48; // subtract off value
 of ASCII 0
  int b = Serial.read() - 48; // subtract off value
 of ASCII 0
  Serial.print(a);
  Serial.print('+');
  Serial.print(b);
  Serial.print('=');
  Serial.println(a+b);
}
```

Listing 4.5: This sketch shows controlling parts of the board using input from the serial port. Remember, don't copy and paste from a PDF since that process garbles some of the characters.

4.1.2 Buttons

4.1.2.1 I/O Setup

To use general I/O pins on a processor in Arduino the pin has to be defined as an input or an output. This is done in the `setup()` function using the `pinMode()` function. It has the form of `pinMode(pinNumber, direction)`. The pinNumber is typically the number defined in the Arduino specifications as D1 or A0. If it is a D1 type number, then just pass the number to pinMode. If it is an A0 type number, you have to specify both the letter and the number. So for the righthand button (SW1) on the v0.5 board that is attached to the A0 pin you would use `pinMode(A0, INPUT)` since it is an input. The buzzer is attached to pin D2 so we would set it up as `pinMode(2, OUTPUT)` since it is an output.

4.1.2.2 Reading and Writing Pins

Once the pinMode has been setup, you can read the value from an input using the `digitalRead(pinNum)` function which takes a pin number as

an argument and returns a 1 or 0 (LOW or HIGH) depending on the read value.

To write a value to an output pin, use the `digitalWrite(pinNum, value)` function. It also takes a pin number as an input along with whether you want the output LOW or HIGH.

Since you will be using the pin number in multiple places, it is best to define it with a name at the top of the program so that you can change it at all places in your program by changing it once.

Examples of reading and writing pins after setting their mode can be seen in Listing 4.5.

4.1.2.3 Other buttons

The v0.5 board has 4 buttons. SW1 is attached to A0 and can be read directly using `digitalRead`. The other three buttons (SW2, FRONT_SW, and BACK_SW) are connected to the PCA9535 I2C to GPIO chip with an address of 0x20 and have to be accessed using the PCA95x5 library.

1. Include the PCA95x5 library
2. Create a variable of type PCA9535 named something like `muxU31` (for the chip listed as U31 on the schematic)
3. Initialize the variable/object:
   (a) Start the wire library: `Wire.begin()`
   (b) Attach to the mux using the attach method and correct address: `muxU31.attach(Wire, 0x20)`
   (c) Set the polarity: `muxU31.polarity(PCA95x5::Polarity::ORIGINAL_ALL)`
   (d) Set the direction: `muxU31.direction(0x????)` remembering that a 1 is an input and a 0 is an output. Set non-connected pins as outputs. Change the question marks appropriately.
4. Once the object is properly setup, read values with the `read()` method. This returns a 16-bit number representing the 16 inputs on the chip.
5. A quick way to check if a bit is high is to bitwise AND (&) it with a number where only the bit of interest is 1: `mux31.read() & 0x0004` reads the third input.

Note that switches SW2, FRONT_SW and BACK_SW are are HIGH when not pressed and LOW when pressed while SW1 is LOW when not pressed and HIGH when pressed.

4.1.3 Making Noise (buzzer)

The board for lab has a buzzer attached to pin D2. Arduino has a handy function called `tone(pin, frequency, duration)`. This is an easy way to make your board beep. The duration is in milliseconds and the function blocks until the duration expires. Note that there is another version of the function with the form `tone(pin, frequency)` that does not specify a duration. This is a non-blocking way to make tones. It will continue to make the tone until a call to `noTone(pinNum)` is called.

NOTE: If the tone function is all that is in a loop, it needs a short delay (10 ms will do) after it to keep the Nano Connect RP2040 from crashing.

4.1.4 NeoPixels

The board has 18 multicolor LEDs around its periphery. These are called NeoPixels by the Adafruit company. They get called other things by other companies. They are all based on the WS2812 type chip. We will use the Adafruit NeoPixel library. Be sure to install it in the usual manner in the Library Manager. It requires the usual:

1. #include the library
2. Initialize an object
3. Setup the library
4. Use the library

A minimal sketch to do all these is shown in Listing 4.6

```
/* NeoPixelSetup.ino
*
* A minimum sketch to setup NeoPixels.
*
* Seth McNeill
* 2022 September 20
```

```
 */

#include <Adafruit_NeoPixel.h> // NeoPixel library

#define NEO_PIN           17  // WARNING! THIS IS
GPIO NOT D NUMBER for NeoPixels
#define NEO_COUNT         18  // number of
NeoPixels

// Declare our NeoPixel strip object:
Adafruit_NeoPixel strip(NEO_COUNT, NEO_PIN, NEO_GRB
 + NEO_KHZ800);

void setup() {
  Serial.begin(115200);

  // Setup NeoPixels
  strip.begin();
  strip.clear();  // Set all pixel values to zero
  strip.show();   // Write values to the pixels
}

void loop() {
  for(int ii = 0; ii < NEO_COUNT; ii++) {
    // set all the pixels to purple
    strip.setPixelColor(ii, strip.Color(255,0,255))
;
  }
  strip.show();
  delay(1000);
  // turn off all the LEDs
  strip.clear();
  strip.show();
  delay(1000);
}
```

Listing 4.6: This snippet shows how to setup and run the NeoPixels on the board.

4.2 Resources

1. Serial Library
2. digitalRead
3. tone
4. notone
5. Adafruit NeoPixel Library
6. PCA9535 datasheet
7. PCA9535 library (McNeill version)
8. PCB Schematic and Layout - see class manual in the Arduino Startup → Schematics and PCB section

4.3 Procedure

Write a sketch with the following functionality:

1. Reads in values from the serial port and does something in response.
2. Writes information to the serial port in response to some stimulus/stimuli.
3. Reacts to all 4 buttons on the board in some way.
   (a) The other buttons are the most difficult part of this lab. START HERE
   (b) Playing a tone
   (c) Lighting up the LED display from the last lab
   (d) NeoPixels
4. Creates a tone in response to some stimulus.
5. Make the NeoPixels do something.

4.4 Turn In

Turn in the following:

1. A video of your board fulfilling the requirements in the Procedure section that includes both of your faces and you saying your names.
2. A PDF of your sketch.
3. .ino versions of your sketch.
4. Fill out the end of lab quiz prior to leaving. Note that it includes asking you for the output of the `getIDs` sketch.

Chapter 5

Displays

5.1 Purpose

The goal of this lab is to learn to use the TFT display.

5.2 Procedure

5.2.1 Main Requirements

Write a sketch with the following functionality:

1. Choose a favorite character and have it move left 10 pixels for each press of the left button and right 10 pixels for each press of the right button.
   (a) Choose the Y value to one that looks good to you
   (b) If 10 pixels seems to not be a good value feel free to change it, just note that you changed the value.
   (c) This builds on last week's making all the buttons function
   (d) The specifications do require software debouncing of the buttons
   (e) Add logic so that the character doesn't go far off the edge of the screen on either side.
2. Display a custom message for 3 seconds indicating the start of the program when your program starts

3. Draw some of the drawing primitives (line, circle, rectangle, etc.)

5.2.2 Extra Credit

The following are extra credit options with increasing value:

1. Make the front and back switches move your character up and down along with the left and right from the Main Requirements. (1 point)
2. Make an animation of some kind that lasts at least two seconds. (2 points)
3. Make a game that uses the buttons and the screen (5 points)

5.3 Turn In

Turn in the following:

1. A video of your board fulfilling the requirements in the Procedure section that includes both of your faces and you saying your names.
2. A PDF of your sketch.
3. .ino versions of your sketch.
4. Fill out the end of lab quiz prior to leaving. Note that it includes asking you for the output of the `getIDs` sketch.

5.4 Resources

1. Adafruit 1.14" 240x135 Color TFT Display + MicroSD Card Breakout - ST7789
2. Adafruit ST7789 library
3. Adafruit GFX library
4. See the chapter in the class manual about displays
5. PCB Schematic and Layout - see class manual in the Arduino Startup → Schematics and PCB section

Chapter 6

Data Collection

6.1 Purpose

The goal of this lab is to learn to collect data with an ADC and other sensors.

6.2 Procedure

6.2.1 Main Requirements

Write a sketch with the following functionality:

Collect the following data and display it once a second on the display with the appropriate labels and units if you can make them fit.

1. The output from `millis()` (ms)
2. Both light intensities as a number between 0 and 1023 (unitless)
3. Potentiometer value as a voltage between 0 and 3.3 V
4. Temperature in Fahrenheit from TEMP0 (U27 attached to input 0 of U37)

The light sensors, potentiometer, and temperature sensor are connected to ADS7142 analog-to-digital (ADC) sensors. There is a library in the normal library adding method. Look for the library for the ADS7142 by Seth McNeill. After you install it, there should now be examples under Anitracks ADS7142. Follow the read2Ch example combined with your previous work to complete this lab.

6.2.2 Suggestions

Use a String object to accumulate your display string and then call `display.println(yourString)` to display it. Note a few things:

1. The String type starts with a capital S.
2. You can add to the String object using += or just +, but with only the plus operator, all arguments have to be of the same type.
3. The String object also allows you to limit the number of decimal places for `float` types. `String(tF,1)` displays tF to 1 decimal place.

An example is shown in Listing 6.1.

```
String dispStr;
dispStr = "T(F): ";
dispStr += String(tF,1);
dispStr += "\n";
// Control the display
tft.fillScreen(ST77XX_BLACK);  // clear display
tft.setTextColor(ST77XX_YELLOW);  // set text color
tft.setTextSize(1);  // Normal 1:1 pixel scale
tft.setCursor(0,0);  // Start at top-left corner
tft.println(dispStr);
```

Listing 6.1: This is an example of using a String object to display text and float variables. The floats are limited to 1 decimal place such that 7.123 would be displayed as 7.1.

However, if you want to have different portions of the text in different colors you will need to call `tft.setTextColor` between each `tft.println`.

6.3 Turn In

Turn in the following:

1. A video of your board fulfilling the requirements in the Procedure section that includes both of your faces and you saying your names.
2. A PDF of your sketch.

3. .ino versions of your sketch.

4. Fill out the end of lab quiz prior to leaving. Note that it includes asking you for the output of the `getIDs` sketch.

6.4 Resources

1. Adafruit 1.14" 240x135 Color TFT Display + MicroSD Card Breakout - ST7789

2. Adafruit ST7789 library

3. Adafruit GFX library

4. See the chapter in the class manual about displays

5. PCB Schematic and Layout - see class manual in the Arduino Startup → Schematics and PCB section