TESSA - Terrarium Environmental System with Smart Automation Documentation and Build Guide

Antonius Torode

September 21, 2023

Contents

Abstract

This project addresses the need for automated monitoring and periodic watering in a terrarium housing small geckos. The terrarium contains live plants and animals which will benefit from consistent monitoring of humidity and temperature. To ensure the geckos' well-being during extended absences, an automated system was developed to monitor and display environmental data. Additionally, the system includes a timer-based water dispensing mechanism for the plants and geckos. The project explores two implementations: one utilizing a Raspberry Pi and the other an Arduino Nano micro-controller. Both implementations involve the creation of a self-contained system adaptable to various scenarios, with a primary focus on automating environmental data display and periodic plant watering.

1 Introduction

Recently, I acquired two small gecko's (I caught them roaming about my living room). I created a terrarium enclosure for them to live, where I have provided them with a few plants, live crickets for food, and a large hollowed rock which I keep full of water. Every now and then I spray down the plants (to give them water) as well as filling the water dish. Unfortunately with the approach of a long needed vacation, a need for automation arose. I cannot leave the water dish for the length of my trip without it evaporating and them running out of water. Therefore, I decided to create an automated system for keeping their water full and plants watered. This opportunity serves as the perfect one for re-learning some circuitry skills as well as developing them further.

At the time of inception for this project, I had a few things laying around (including a Raspberry Pi). Therefore, a Raspberry Pi implementation was developed. With the desire to simplify the cost of the project as well as create a standalone system that doesn't rely on an OS, I explored using an Arduino Nano, and created an implementation using that as well. The primary goal of this project was both to automate and care for my gecko's enclosure while also furthering my skills and knowledge. It was decided (as I had never done it before) to create a complete integrated circuit (from a bare board and some pre-made components) for this project.

2 Project Overview

This is a simple project that utilizes an Arduino Nano (or a separate Raspberry Pi implementation) to setup an automated temperature/humidity monitoring for a terrarium as well as automating watering for filling a water dish and/or watering plants. The project involves created a self contained system. This system is highly versatile and the principals and components contained can be modified to fit a variety of situations and needs.

For more project details, files, and a complete set of supplemental material - see the git repository here:

https://github.com/torodean/Antonius-Personal/tree/master/TESSA

3 Components and Materials

At the heart of this project, some sort of micro-controller will be needed (i.e. Arduino Nano). This will store the basic timing and control components for the various components. These components include a temperature and humidity sensor (DHT11), a submersible low voltage water pump, and a display for output (OLED). the setup for this type of system is highly configurable and therefore the components can also vary widely. Basic circuit components will also be needed.

For testing the system, jumper cables and a breadboard are recommended. A complete components list is available for two various/possible setups.

Arduino Setup Components List

- 1 Arduino Nano
- 1 DHT11 Sensor
- 1 SSD1306 OLED Display
- 1 micro 5V submersible pump
- 1 10kΩ resistor
- 1 diode (1N4001-1N4007)
- 1 2N 2222A Transistor (or similar NPN transistor)
- 1 10×24 PCB
- Wires (varying sizes)
- Soldering kit (Solder Iron, Solder, Flux, etc)

Raspberry Pi Setup Components List

- 1 Raspberry Pi
- 1 DHT11 Sensor
- 1 SSD1306 OLED Display (optional)
- 1 micro 5V submersible pump
- 1 10kΩ resistor
- Wires (varying sizes)
- Jumper cables

Component Descriptions

This project can be accomplished using many different components. Below is a list and description of some of the components used for this specific implementation. Some components have subtleties in how they should be implemented. It is recommended to read documentation on the components before using them.

DHT11

The DHT11 sensor is a low-cost and easy to use sensor for measuring temperature and humidity levels. The sensor provides digital output, making it easy to interface with micro-controllers like Arduino or Raspberry Pi. It uses a single wire to transmit two environmental data points in a single package. The DHT11 has the following features:

- 3 to 5V power and I/O
- 2.5mA max current use during conversion (while requesting data)
- Good for 20-80% humidity readings with 5% accuracy
- Good for 0-50°C temperature readings ±2°C accuracy
- No more than 1 Hz sampling rate (once every second)
- Body size 15.5mm × 12mm × 5.5mm
- 4 pins with 0.1" spacing

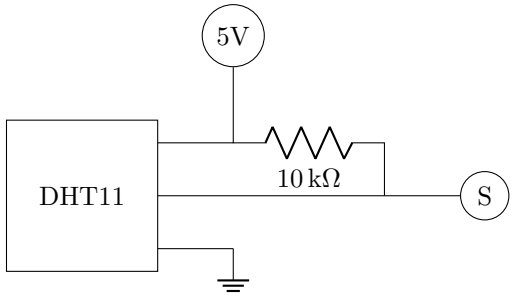


Figure 1: The DHT11 circuit setup. The GND pin needs connected to ground. The 5V pin needs connected to a constant 5V DC source. The data pin needs connected to the proper data channel to read from. A 10kΩ resistor is connected between the data pin and voltage pin that serves as a 'pull-up' for the data.

Mini Submersible Water Pump

The pump I found for this project was a micro submersible mini water pump. It has the following features:

- Rated voltage: 3.3V or 5V DC
- No load of water discharge capacity: 100L / H
- Load rated current: 150-250mA, Use: diving type

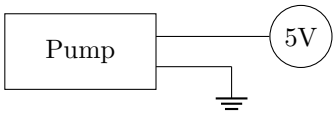


Figure 2: The water pump only has two connectors, one for voltage, and the other for ground.

SSD1306 I^{2}C OLED Display

The display I used for this project is a SSD1306 0.96 inch I^{2}C organic light-emitting diode (OLED) display. The OLED display doesn't require a back light and the pixels only consume energy when on. The model I am using has 4 pins (V_{in}, GND, SCL, SDA). Some features of the display are as follows:

- Resolution: 128 x 64 dot matrix panel
- Support voltage: 3.3V-5V DC
- Wide range of operating temperature: -40°C to 85°C
- Embedded Driver IC: SSD1306. Communication: I2C/IIC Interface, only need two I / O ports

The I^{2}C driver means the bus consists of two signals: SDA and SCL. SDA (Serial Data) is the data signal and SCL (Serial Clock) is the clock signal.

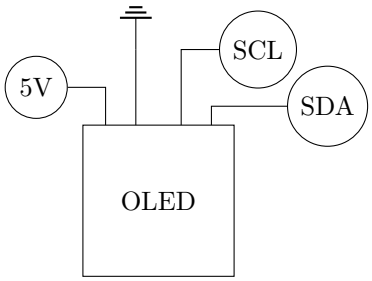


Figure 3: The OLED display has 4 inputs. The VCC is the power input, GND is ground, SDA is the serial data connector and SCL is the serial clock.

2N 2222A NPN Transistor

A transistor can be used as a switch by controlling the flow of current between its collector and emitter terminals. When a small current flows into the transistor's base terminal, it allows a larger current to pass from the collector to the emitter, effectively acting as an electronic switch that can be turned on and off based on the base current.

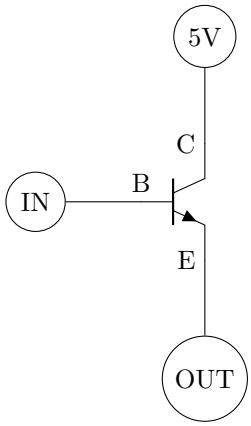


Figure 4: The 2N 2222A NPN Transistor. The IN is the triggering 3.3V input signal. The OUT is connected to the device being powered and is activated when the IN voltage is powered on. A common ground should be used among the connected components.

Arduino Nano

The Arduino Nano is a compact and versatile microcontroller board designed for embedded electronics projects and prototyping. The Nano is known for its small physical size, making it suitable for applications with limited space or when a compact design is essential. The Nano includes a set of digital and analog pins, allowing you to interface with sensors, actuators, and other electronic components. These pins can be programmed for input or output. It features a USB connector for easy programming and communication with a computer. This makes it convenient for uploading code and debugging. It is programmed using the Arduino IDE, which provides a user-friendly interface for writing, uploading, and debugging code. A large community and extensive libraries are available to simplify development. Like other Arduino boards, the Nano is open-source hardware and software. This means that the design files, schematics, and software are freely available for modification and customization.

Some features of the arduino nano that are important for this project are as follows:

- High-performance low-power 8-bit processor
- Has a 5V DC output pin.
- Has multiple programmable I/O pins.
- Has SDA and SCL connections for I^{2}C communication.

[1]This is important because the newer versions of the Arduino IDE are not pre-built for the ARM architecture.

Raspberry Pi

The Raspberry Pi is a versatile and powerful single-board computer designed for various computing and electronics projects. The Raspberry Pi is equipped with a high-performance ARM-based[1] processor (varies by model) that provides the processing power needed for a wide range of applications. It features a set of General-Purpose Input/Output (GPIO) pins, which can be programmed for digital input and output as well as hardware interfacing. These pins are crucial for connecting sensors, actuators, and other components. The Raspberry Pi supports communication protocols like I^2C and SPI, allowing you to connect and communicate with a variety of sensors and devices easily. Like Arduino, the Raspberry Pi is open-source hardware and software. It has a large and active community that offers support, tutorials, and a wide range of software packages.

Some Raspberry Pi models include built-in Wi-Fi and Bluetooth capabilities, making wireless communication and IoT projects convenient. The Raspberry Pi has an HDMI output for connecting to displays, making it suitable for applications that require visual feedback. Between these two features, a lot of possibilities open up with allowing the DHT11 sensor to be displayed on Pi screen rather than an OLED as well as potentially being able to monitor it remotely.

Some features of the Raspberry Pi has that are important for this project are as follows:

- Has a 5V DC output pin.
- Has multiple programmable GPIO pins.
- Has SDA and SCL connections for I^2C communication.

4 System Architecture

Given the two approaches outlined below for this project, there are a few things of importance to note when setting up and working with the various components. For the Raspberry Pi implementation, I used an Ubuntu 22.04.2 LTS install. Using the Raspian OS, things may vary slightly but shouldn't differ much.

When working with and programming the Arduino Nano, I used the Raspberry Pi system as my working/development machine. As an afterthought, this is not recommended. The Raspberry Pi uses an ARM processor architecture which requires a few extra steps for setting up the Arduino IDE. After setting this up on a separate laptop, it was much easier to get things going (as well as things working a lot faster on a faster machine) as far as setup and development.

5 Raspberry Pi Implementation

Circuit Diagram and Wiring

See figure 9 in section 14.2.

Software Setup

1. Setup and Install Python3 for development
    - sudo apt-get install python3
    - sudo apt-get install python3-pip
    - sudo apt-get install build-essential python3-dev
2. Setup and Install git
    - sudo apt-get install git-core
3. Install DHT11 Drivers
    - git clone https://github.com/adafruit/Adafruit_Python_DHT.git
    - cd Adafruit_Python_DHT
    - sudo python3 setup.py install
4. Install OLED Drivers
    - **THIS SECTION IN PROGRESS.**
    - sudo python3 setup.py install

Python Code

For testing, two separate programs were created. One which tests the DHT11 sensor and displays the output to the OLED and one which tests the pump timing. After these were working, another python script was made which combines these functions into one script. These can all be seen in section 14.3.

Testing and Troubleshooting

I did not really experience any issues with this setup, so I have no troubleshooting advice to give here.

6 Arduino Nano Implementation

This is the implementation using the Arduino Nano as the micro-controller. This setup is completely standalone and does not require a desktop monitor hookup once finished. Rather, it displays all the information on the OLED display and runs the pump based on a pre-programmed timer. A separate machine will be required to program the Arduino Nano. I recommend using a laptop or desktop running Linux (I have not tried this on a windows machine). If a Raspberry Pi is used as the development machine (this is what I used), a few extra steps will be needed to get the Arduino IDE software functioning.

Circuit Diagram and Wiring

See figure 5 and figure 6 in section 14.2 for circuit diagrams. For a PCB schematic, see figure 8.

Software Setup

1. Install Arduino IDE
    - The steps for this will vary depending on the development machine.
2. Install DHT11 Drivers
    - This should show as "" in the Arduino IDE Library Manager
3. Install SSD1306 Drivers
    - This should show as "Adafruit SSD1306" in the Arduino IDE Library Manager

Arduino Code

For testing, three separate programs were created. One which tests the OLED display on it's own, and one which tests the DHT11 sensor and displays the output to the OLED and one which tests the pump timing. After these were working, the latter two scripts were combined with the OLED code to add these functions into one script. The first program and final can be seen in section 14.3. Unfortunately, I forgot to save the code that only tested the DHT11 sensor and displays the output to the OLED as well as the simple timing code. Fortunately, those were quite simple and not needed.

Testing and Troubleshooting

If you are testing the setup using a breadboard, I recommend making sure the Arduino Nano jumper pins (at least the ones being used) are soldered on. If they are not soldered on, some of the jumper pins may not have a good connection and things can work unexpectedly.

7 3D Printed Enclosure

At the time of creating this project, no 3D printed enclosure was designed. This is, however, a plan for the future to keep everything protected and enclosed.

8 Power Supply

For the Raspberry Pi implementation, all power comes from the Raspberry Pi itself, therefore no extra power sources are needed[2]. For the Arduino Nano implementation, the power comes from a dedicated USB powered cable. This can be accomplished in any of the usual ways that a USB device is charged (I would not recommend trying fast chargers as I have no tested with any of those).

9 Usage and Operation

THIS SECTION IN PROGRESS.

10 Maintenance and Troubleshooting

THIS SECTION IN PROGRESS.

11 Full Enclosure Setup

The complete setup and implementation will vary for every setup. This is simply what I have used and designed for my specific setup and needs.

Equipment and Tools

THIS SECTION IN PROGRESS.

Setup and Implementation

THIS SECTION IN PROGRESS.

[2]If this project is adapted to use a more powerful pump, this will of course change.

12 Future Enhancements

There are many additions or future enhancements that can be made to this system. These can include (but are not limited to) the following:

- Add in an override switch for activating the pump outside of the normal programmed timing for the Arduino Nano setup.
- External monitoring for the Arduino implementation would be nice. Perhaps some sort of wireless signal output that can be monitored to display status messaging.

13 Author Contact Information

If you have any questions or comments about this project, you can find the authors contact information on his website at the following link:

```
https://torodean.github.io
```

References

[1] DHT11: https://www.adafruit.com/product/386

[2] SSD1306 Datasheet: https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf

[3] Arduino Nano: https://docs.arduino.cc/hardware/nano

14 Appendices

These Appendices include extra data and supplemental material for the sections above.

Appendix A: Bill of Materials

Most of the components for this project were purchased in kits of other components. Therefore, this is an approximate (overestimate in most cases) cost of components for items used in this project.

- Raspberry Pi Implementation
  - Raspberry Pi 4: $70
  - Mini Water Pump: $2
  - DHT11 sensor: $2
  - (optional) OLED display: $3
  - Jumper Cables/wiring: $2
  - Resistor/Transistor/Diode: $1
  - Breadboard: $3
  - **Total: $83**
- Arduino Nano Implementation
  - Arduino Nano V3.0: $6
  - Mini Water Pump: $2
  - DHT11 sensor: $2
  - OLED display: $3
  - 10×24 blank PCB: $1
  - Jumper Cables/wiring: $2
  - Resistor/Transistor/Diode: $1
  - (optional for testing) Breadboard: $3
  - Soldering iron, wire cutters, solder, flux, etc: Variable
  - **Total: $20**

Appendix B: Schematics and PCB Designs

Arduino Nano Circuit Design

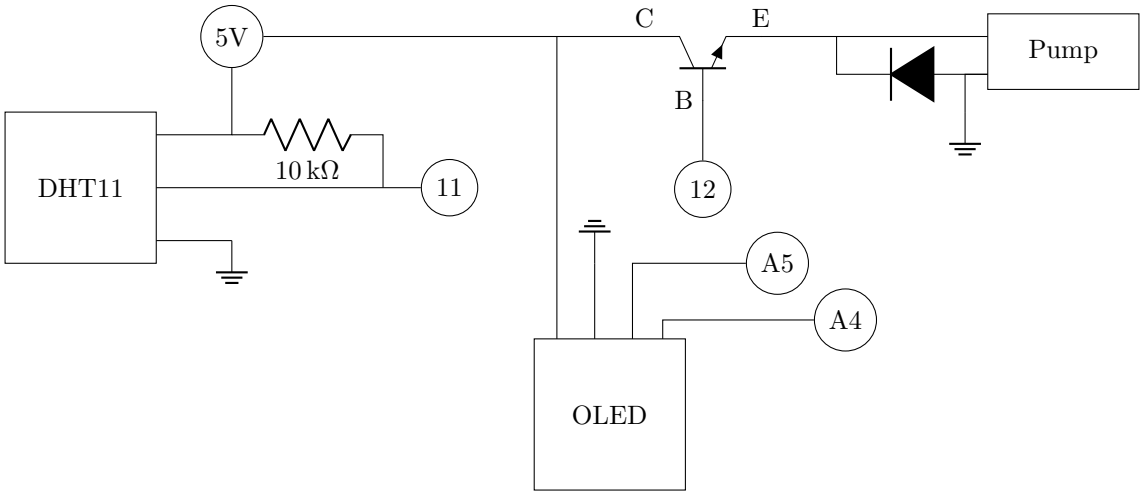


Figure 5: The Circuit diagram for the Arduino Nano setup. All grounds must be common. The A4 and A5 pins correspond to the Arduino Nano's SDA/SCL pins and should not be modified. The other circle connectors 11 and 12 correspond to the I/O pins on the Nano and can be changed to any of the other I/O pins if desired (the code and circuitry would need updated appropriately).

Arduino Nano Circuit Design With Extra Pump(s)

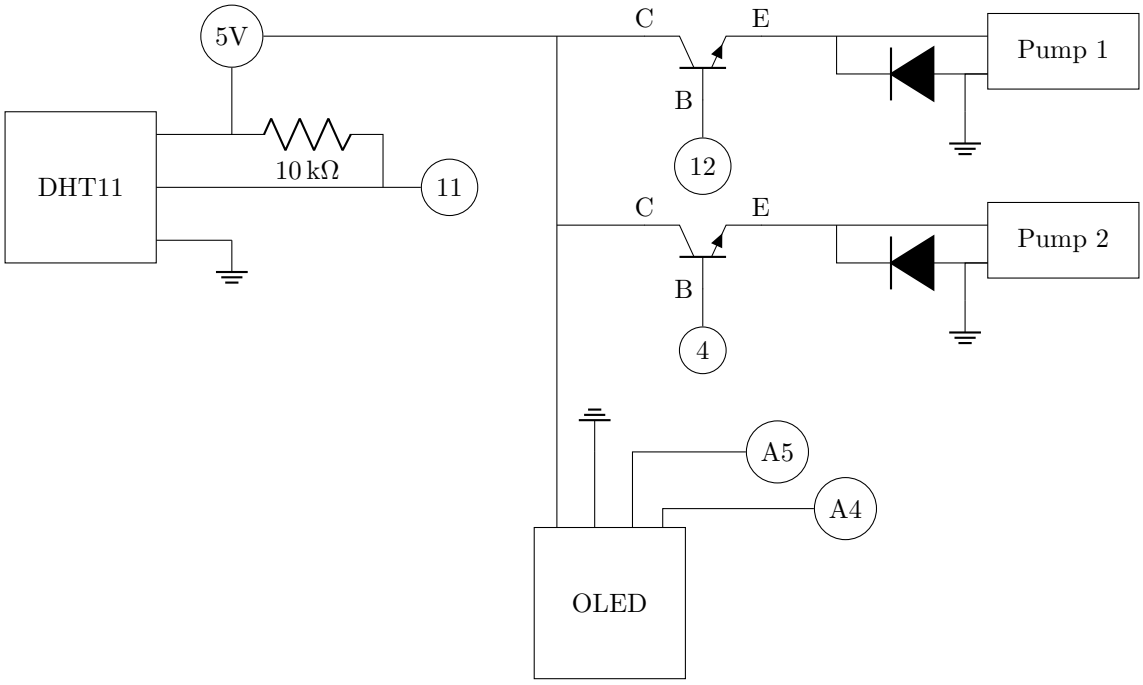


Figure 6: The Circuit diagram for the Arduino Nano setup. All grounds must be common. The A4 and A5 pins correspond to the Arduino Nano's SDA/SCL pins and should not be modified. The other circle connectors 4, 11 and 12 correspond to the I/O pins on the Nano and can be changed to any of the other I/O pins if desired (the code and circuitry would need updated appropriately).

Arduino Nano PCB schematic

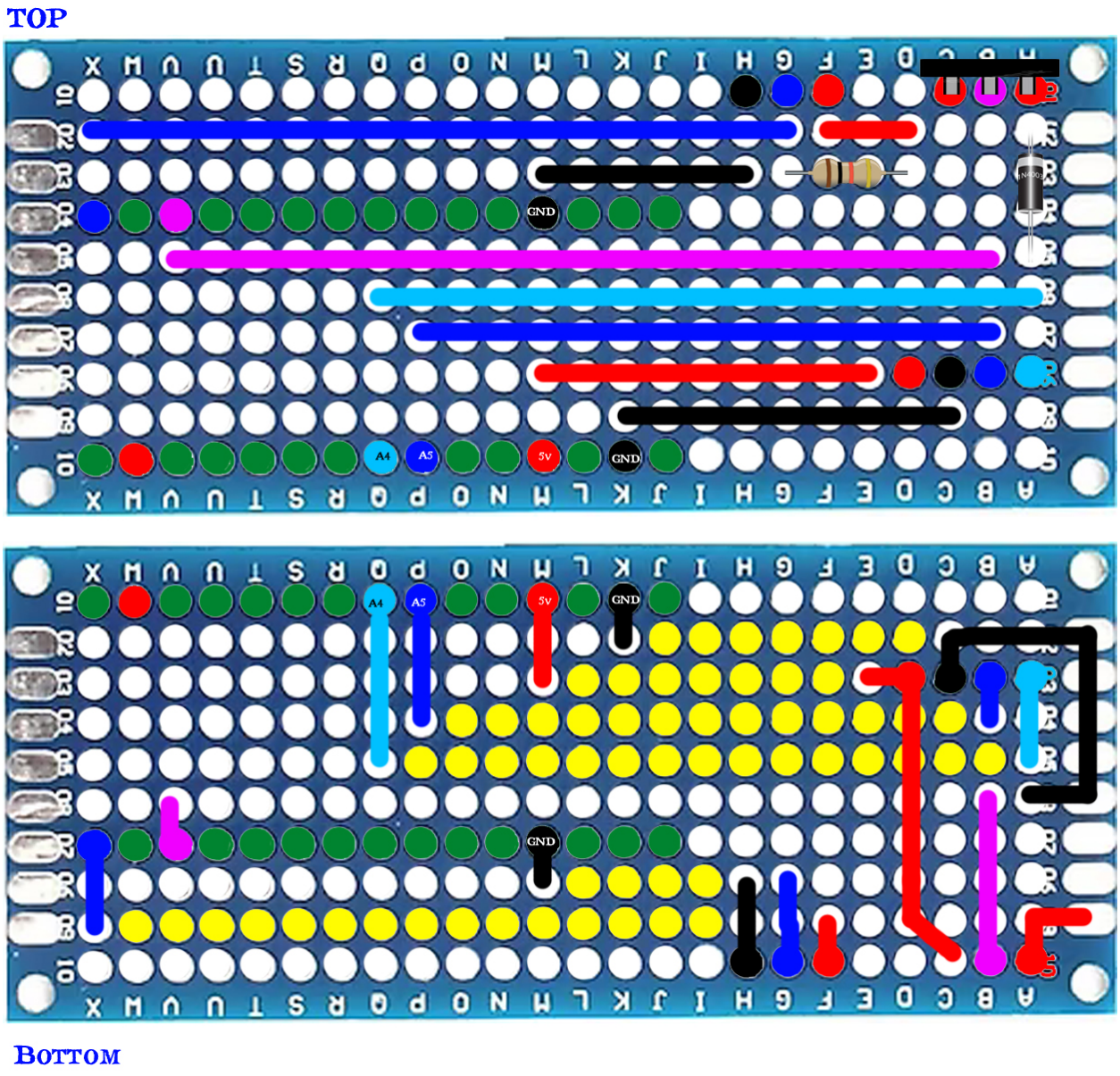


Figure 7: The Circuit PCB schematic for the Arduino Nano setup (one pump displayed). The colored lines represent soldered connections or wires. The yellow circles indicate wires on the other side of the board. The green dots are the placement of the Arduino Nano.

Arduino Nano Implementation Final Board

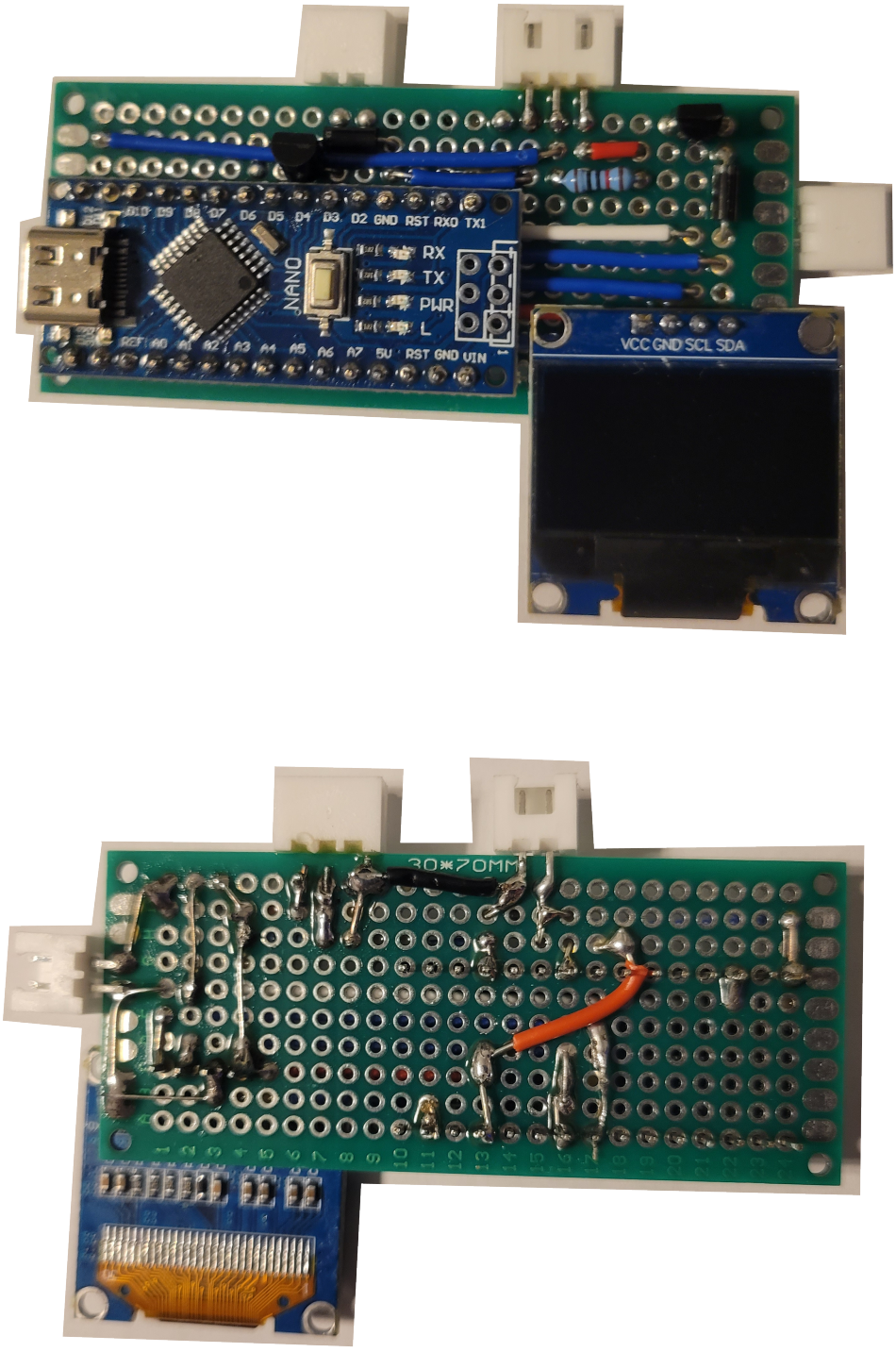


Figure 8: The Circuit PCB board for the Arduino Nano setup (two pumps supported). Front and back view

Raspberry Pi Circuit Design

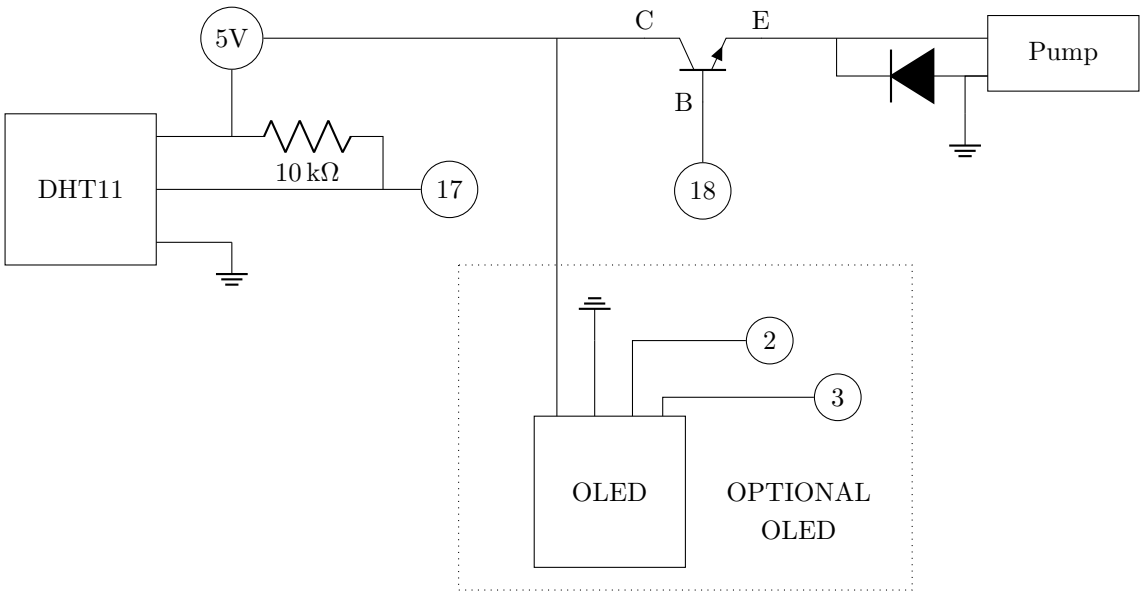


Figure 9: The Circuit diagram for the Raspberry Pi setup. All grounds must be common. The OLED section is optional and can be omitted if the user would rather monitor from the same display as the Raspberry Pi. the circles with numbers correspond to the GPIO pins on the Raspberry Pi. Pins 2 and 3 are the SDA/SCL pins respectively and should be kept as they are but the others can be used with other GPIO pins if desired (the code will need updated appropriately).

Appendix C: Code Samples

Raspberry Pi Python Code: Run all features (one pump installed)

```
import Adafruit_DHT
import Adafruit_SSD1306
from PIL import Image, ImageDraw, ImageFont
import RPi.GPIO as GPIO
import time
import threading

# Set the GPIO mode to BCM
GPIO.setmode(GPIO.BCM)

# Define the GPIO pin you want to use
gpio_pin = 18  # Replace with the actual GPIO pin number

# Set the GPIO pin as an output
GPIO.setup(gpio_pin, GPIO.OUT)

# Constants for DHT11 sensor and GPIO pin
DHT_SENSOR = Adafruit_DHT.DHT11
DHT_PIN = 4

# Initialize the OLED display
disp = Adafruit_SSD1306.SSD1306_128_64(rst=None)
disp.begin()

# Clear the display
disp.clear()
```

```
disp.display()

# Create a blank image for drawing
width = disp.width
height = disp.height
image = Image.new("1", (width, height))

# Create a drawing object
draw = ImageDraw.Draw(image)

# Load a font
font = ImageFont.load_default()

# Initialize the device status variable
device_running = False

def device_ON(t=5):
    """
    Turns the device on for t seconds.
    """
    global device_running
    # Set the device status to running
    device_running = True

    # Turn on the 5V output
    GPIO.output(gpio_pin, GPIO.HIGH)
    print("Device is ON")

    # Wait for some time (you can perform other tasks here)
    time.sleep(t)  # Wait for t seconds

    # Turn off the 5V output
    GPIO.output(gpio_pin, GPIO.LOW)
    print("Device is OFF")

    # Set the device status to not running
    device_running = False

def device_control_thread():
    while True:
        device_ON()
        time.sleep(30)  # Run device_ON() every 30 seconds

# Create a separate thread for device control
device_thread = threading.Thread(target=device_control_thread)
device_thread.daemon = True  # This allows the thread to exit when the main
    program exits
device_thread.start()

try:
    while True:
        # Read data from the DHT11 sensor
        humidity, temperature = Adafruit_DHT.read(DHT_SENSOR, DHT_PIN)

        # Check if data is valid
        if humidity is not None and temperature is not None:
            # Clear the previous content on the OLED display
            draw.rectangle((0, 0, width, height), outline=0, fill=0)

```

```
            # Format and display the temperature and humidity
            message = f"Temp: {temperature:.1f}C\nHumidity: {humidity:.1f}%"
            draw.text((5, 5), message, font=font, fill=255)

            # Display the device status
            status_message = "Pump ACTIVE" if device_running else "Pump IDLE"
            draw.text((5, height - 15), status_message, font=font, fill=255)

            # Display the updated content on the OLED
            disp.image(image)
            disp.display()
except KeyboardInterrupt:
    pass
finally:
    # Clear the display and turn it off
    disp.clear()
    disp.display()
    # Clean up GPIO configuration
    GPIO.cleanup()
```

Raspberry Pi Python Code: Test DHT11 with OLED display

```
import Adafruit_DHT
import Adafruit_SSD1306
from PIL import Image, ImageDraw, ImageFont

# Constants for DHT11 sensor and GPIO pin
DHT_SENSOR = Adafruit_DHT.DHT11
DHT_PIN = 4

# Initialize the OLED display
disp = Adafruit_SSD1306.SSD1306_128_64(rst=None)
disp.begin()

# Clear the display
disp.clear()
disp.display()

# Create a blank image for drawing
width = disp.width
height = disp.height
image = Image.new("1", (width, height))

# Create a drawing object
draw = ImageDraw.Draw(image)

# Load a font
font = ImageFont.load_default()

try:
    while True:
        # Read data from the DHT11 sensor
        humidity, temperature = Adafruit_DHT.read(DHT_SENSOR, DHT_PIN)

        # Check if data is valid
        if humidity is not None and temperature is not None:
            # Clear the previous content on the OLED display
            draw.rectangle((0, 0, width, height), outline=0, fill=0)
```

```
            # Format and display the temperature and humidity
            message = f"Temp: {temperature:.1f}C\nHumidity: {humidity:.1f}%"
            draw.text((5, 5), message, font=font, fill=255)

            # Display the updated content on the OLED
            disp.image(image)
            disp.display()
except KeyboardInterrupt:
    pass
finally:
    # Clear the display and turn it off
    disp.clear()
    disp.display()
```

Raspberry Pi Python Code: Test the motor

```
import RPi.GPIO as GPIO
import time

# Set the GPIO mode to BCM (Broadcom SOC channel)
GPIO.setmode(GPIO.BCM)

# Set the pin 12 (GPIO18) as an output
gpio_pin = 18  # This corresponds to GPIO18
GPIO.setup(gpio_pin, GPIO.OUT)

try:
    # Turn on the GPIO pin
    GPIO.output(gpio_pin, GPIO.HIGH)
    print(f"GPIO pin {gpio_pin} is turned ON.")

    # Run for 5 seconds
    time.sleep(5)

    # Turn off the GPIO pin
    GPIO.output(gpio_pin, GPIO.LOW)
    print(f"GPIO pin {gpio_pin} is turned OFF.")

except KeyboardInterrupt:
    # Clean up GPIO configuration on Ctrl+C
    GPIO.cleanup()

finally:
    # Clean up GPIO configuration on program exit
    GPIO.cleanup()
```

Arduino Nano Code: Complete 2 pump setup

```
#include <Wire.h>             /*Wire Communication library*/
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h> /*OLED Adafruit library*/
#include <Adafruit_Sensor.h>
#include <DHT.h>              /*DHT sensor library*/

#define SCREEN_WIDTH 128   /*128 width OLED in pixels*/
```

```
#define SCREEN_HEIGHT 64   /*64 height OLED in pixel*/

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1); /*I2C
    Display initialization*/

#define DHTPIN 12   /*DHT11 signal pin*/
#define DHTTYPE DHT11
#define PUMPPIN1 10 /*Pump 1 signal pin*/
#define PUMPPIN2 4  /*Pump 2 signal pin*/

DHT dht(DHTPIN, DHTTYPE);

void setup() {
    pinMode(PUMPPIN1, OUTPUT);
    pinMode(PUMPPIN2, OUTPUT);
    Serial.begin(9600);
    dht.begin();
    if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {   /*OLED I2C Address*/
        Serial.println(F("SSD1306 allocation failed"));
        for(;;);
    }
    // Show initial display buffer contents on the screen --
    // the library initializes this with an Adafruit splash screen.
    display.display();
    delay(2000);
    display.clearDisplay();
    display.setTextColor(WHITE);  /*Text color*/
}

long count = 0;

// The time to keep the first pump on.
long onTime = 30;

// The time to keep the second pump on.
long onTime2 = 30;

// The delay for turning the pump on.
long delayTime = 24L*60L*60L - onTime - onTime2; // one day - on times.

void loop() {
    delay(1000);
    count++;
    if (count >= delayTime + onTime + onTime2) {
        digitalWrite(PUMPPIN2, LOW);
        displayText(false);
        count = 0;
    } else if(count >= delayTime + onTime){
        digitalWrite(PUMPPIN2, HIGH);
        digitalWrite(PUMPPIN1, LOW);
        displayText(true);
    } else if(count >= delayTime){
        digitalWrite(PUMPPIN1, HIGH);
        displayText(true);
    } else {
        displayText(false);
    }
}

```

```
void displayText(bool pumpStatus){
    float t = dht.readTemperature();  /*read temp*/
    float h = dht.readHumidity();    /*read humidity*/
    if (isnan(h) || isnan(t)) {
        Serial.println("Failed to read from DHT sensor!");
    }
    display.clearDisplay();  /*clear display*/
    display.setTextSize(1); /*OLED font size*/
    display.setCursor(0,0);
    display.print("Temp: ");
    display.print(t);       /*print temp in Celsius*/
    display.print(" ");
    display.cp437(true);
    display.write(167);
    display.print("C");
    display.setCursor(0, 10);
    display.print("Humidity: ");
    display.print(h);      /*prints humidity percentage*/
    display.print(" %");
    if (pumpStatus) {
        display.setCursor(0, 35);
        display.print("Pump ACTIVE");
    } else {
        display.setCursor(0, 35);
        display.print("Pump IDLE");
    }
    display.display();
}
```

Arduino Nano Code: OLED display testing

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 /*128 width of OLED in pixels*/
#define SCREEN_HEIGHT 64 /*64 height of OLED in pixels*/

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1); /*OLED
    display connected at I2C pins (SDA, SCL)*/

void setup() {
    Serial.begin(115200);  /*Baud rate UART communication */
    if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { /*I2C Address at which
        OLED will communicate*/
        Serial.println(F("SSD1306 allocation failed"));
        for(;;);
    }
    delay(2000);
    display.clearDisplay();  /*Clear display*/
    display.setTextSize(2);  /*OLED screen text size defined*/
    display.setTextColor(WHITE); /*OLED screen text color*/
    display.setCursor(0, 10); /*Display static text*/
    display.println("HELLO WORLD!!");  /*String to represent on OLED display
        */
    display.display();
}

```

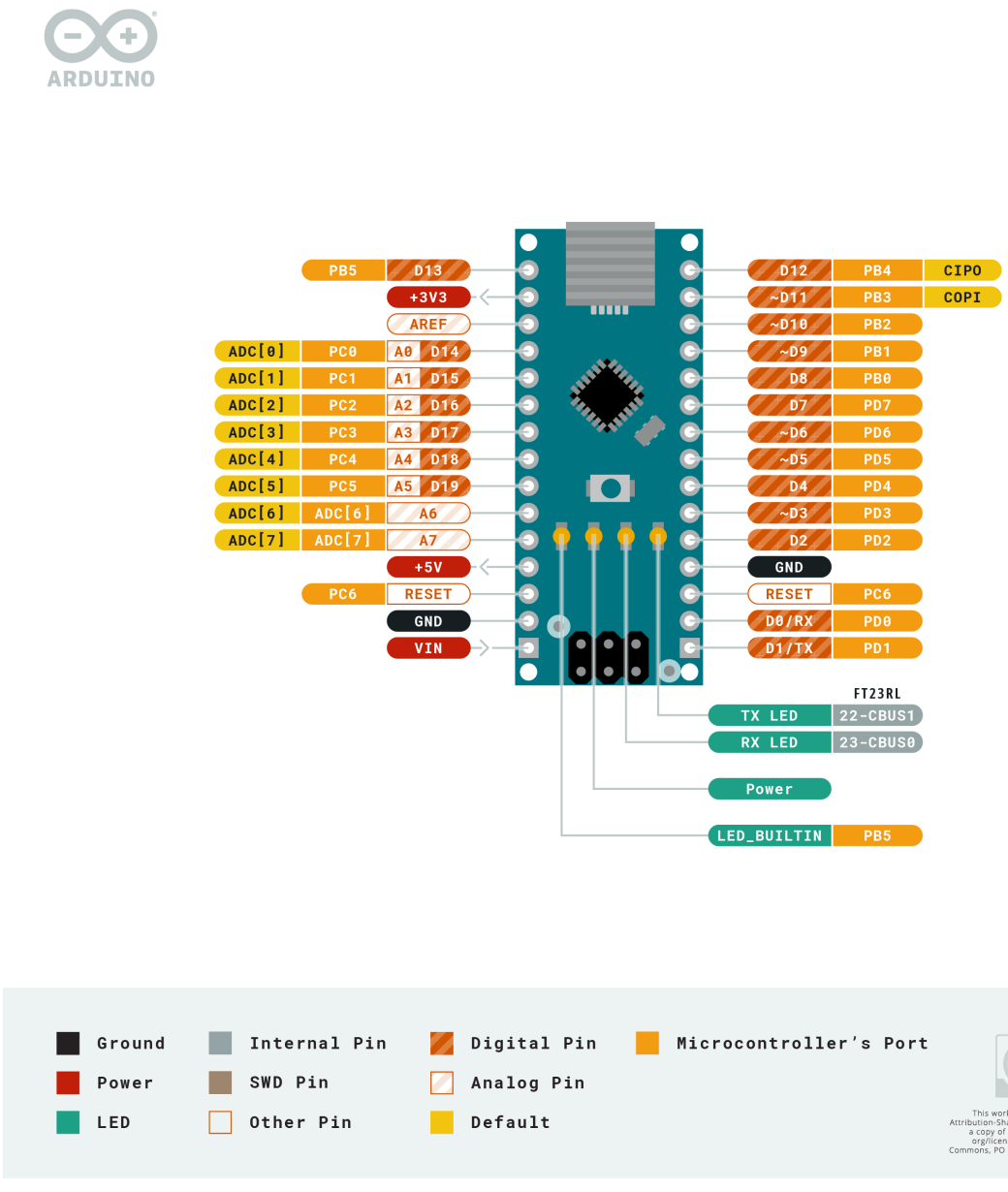
```
25 void loop() {}
```

Appendix D: 3D Printing Files

THIS SECTION IN PROGRESS.

Appendix E: Micro-Controller Pin Diagrams

Arduino Nano Pinout



Raspberry Pi Pinout