Coela Can’t!

KB WS35 Protogen Kit:

Getting Started

V1.0.2022 – Coela Can’t!

Contents

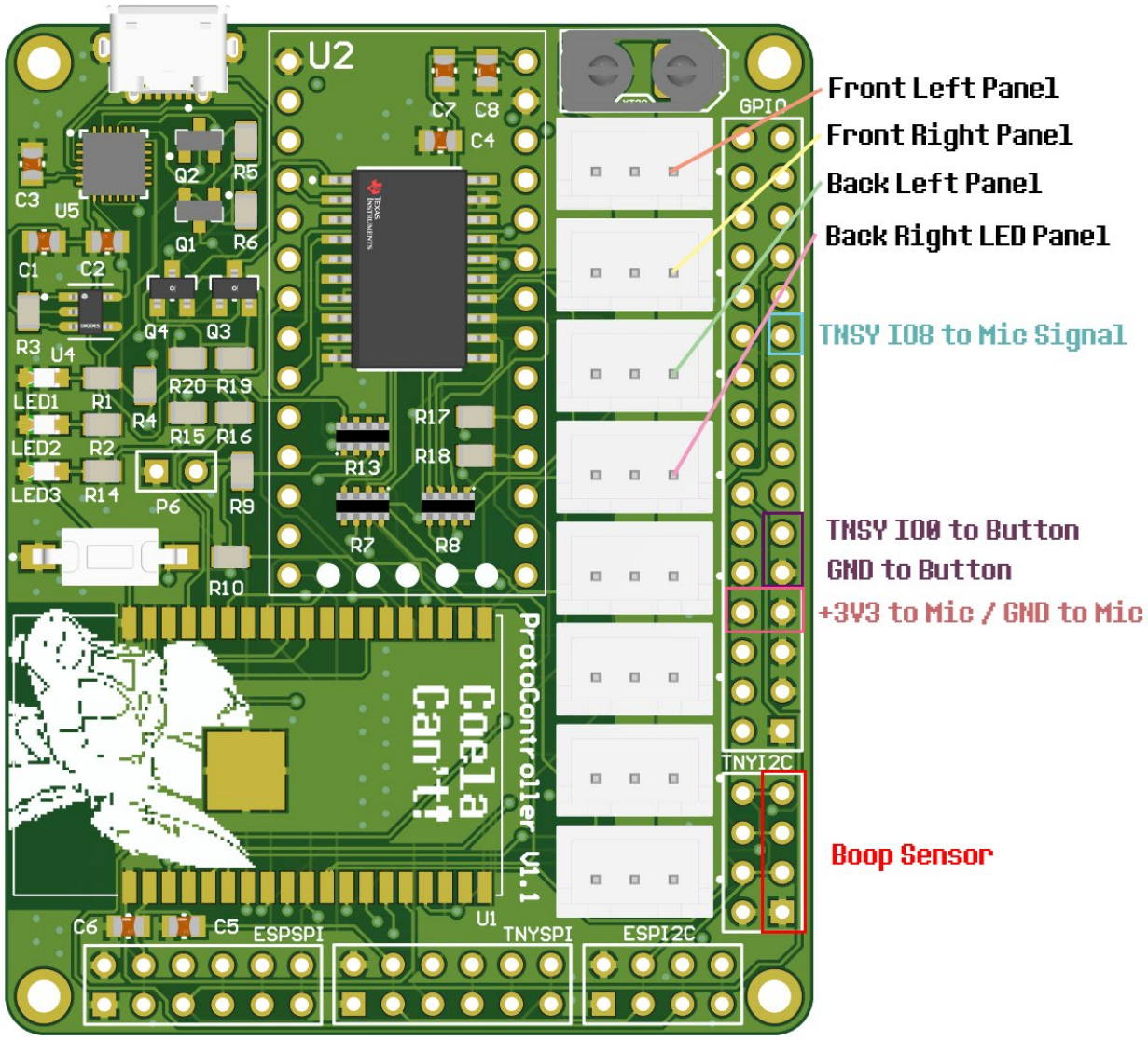
1 Wiring Your Protogen 4
1.1 Sensor Wiring Diagram 4
1.2 Wiring Pictures 5
2 KB WS35 LED Panels 6
2.1 General Information 6
2.2 Top-Side LED Panel 6
2.3 Bottom-Side LED Panel 7
3 Protocontroller V1.1 8
3.1 Pinout 8
3.2 Schematic 9
3.3 Physical Design 10
3.3.1 Top View 10
3.3.2 Bottom View 11
3.3.3 Isolated Top Layer View 12
3.3.4 Isolated Signal Layer 1 View 13
3.3.5 Isolated Signal Layer 2 View 14
3.3.6 Isolated Bottom Layer View 15
3.3.7 Isolated Top Layer Overlay 16
3.3.7 Isolated Bottom Layer Overlay (Flipped) 17
3.4 Header Pinouts 18
3.4.1 32 Pin Header 18
3.4.2 ESP32 SPI Header 18
3.4.3 ESP32 I2C Header 19
3.4.4 Teensy SPI Header 19
3.4.5 Teensy I2C Header 19
3.4.6 Battery Measurement Header 20
3.4.7 XT30 Power Input 20
3.5 Indicators 20
3.6 Power Setup 21
3.6.1 Powering the ESP32 for Programming 21
3.6.2 Powering the Teensy 4.0 for Programming 21
3.6.3 Powering the Controller for Usage 21
4 Sensors/Peripherals 22
4.1 Boop Sensor (APDS-9960) 22
4.2 MAX9814 Microphone 22
4.3 Control Button 22
4.4 MPU6050 22
4.5 I2C OLED Display 22
5 Programming the Teensy 23
5.1 Getting Started with ProtoTracer 23
5.1.1 Install Applications 23

5.1.2 Install Extensions 23
5.1.3 Download the Codebase 23
5.2 Loading a Controller 24
5.3 Loading or Modifying an Animation 25
5.4 Controller Example 26
5.5 Animation Example 29
6 Programming the ESP32 30
6.1 Programming in Arduino 30
6.1.1 Installing the ESP32 Boards in Board Manager 30
6.2 Example Communication Code for ESP32 33
6.2.1 ProtoControllerESP32.ino 33
6.2.2 MorphDecoder.h 35
6.3 Example Communication Code for Teensy 4 37
6.3.1 TeensyTest.ino 37
6.3.2 SerialInterpreter.h 37
7 General Recommendations and Notes 39

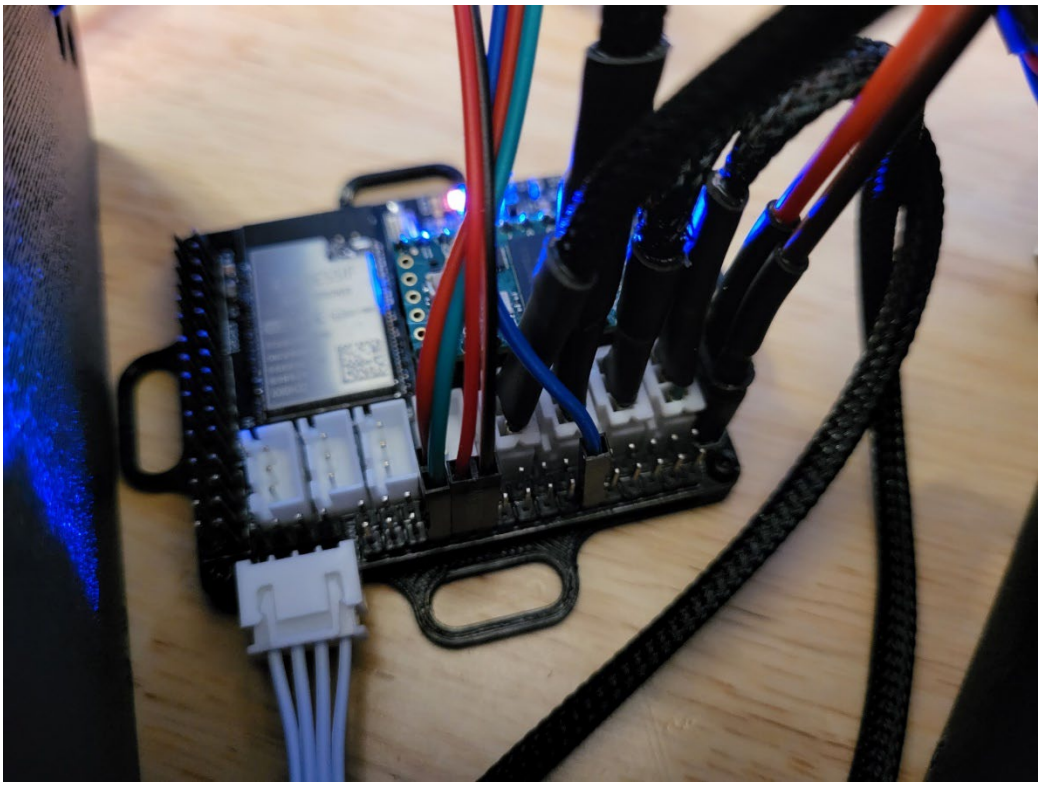
1 Wiring Your Protogen

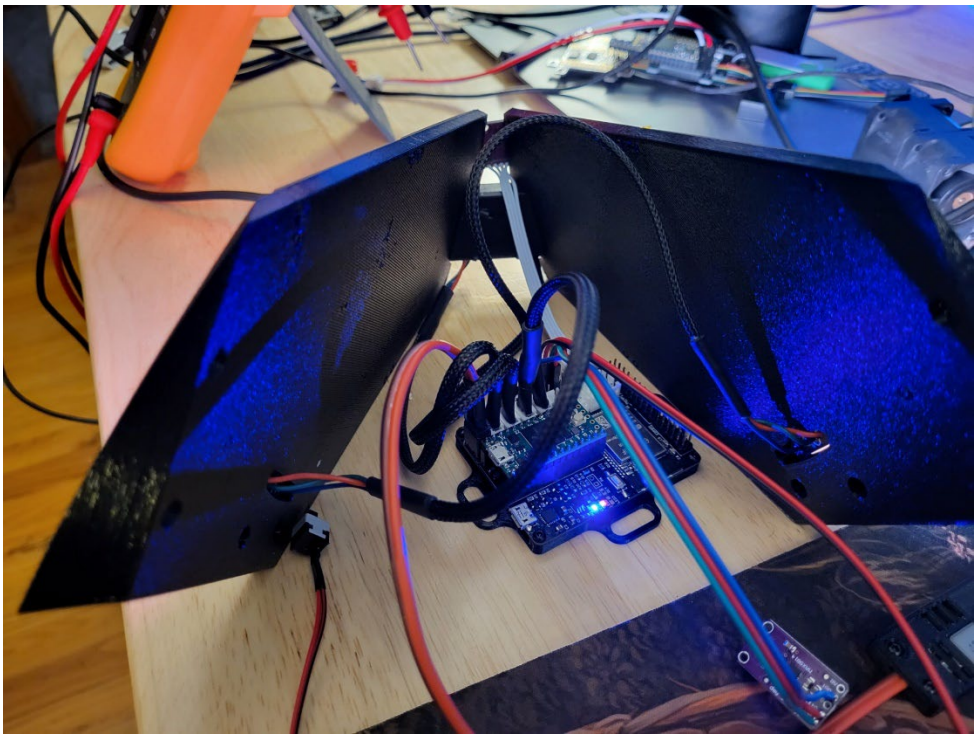
1.1 Sensor Wiring Diagram

The following diagram details the connections for all four separated LED panels for 100Hz operation along with the MAX9814 microphone, the APDS-9960 boop sensor, and the face control button.



1.2 Wiring Pictures





2 KB WS35 LED Panels

2.1 General Information

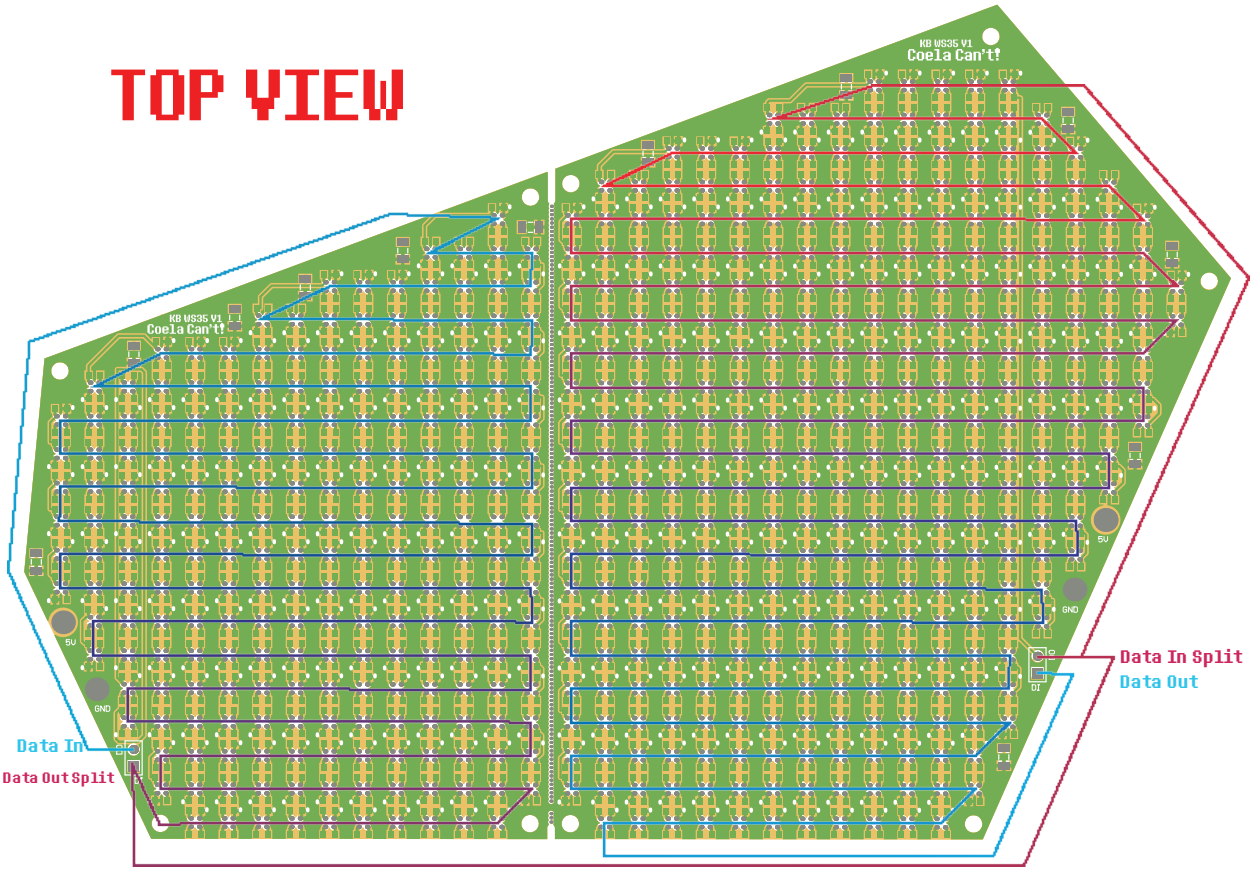
These LED boards use WS2812B-Mini serialized LEDs, these are glorified LED strips and can be controlled with the same controller! Each board has 571 LEDs which need to be controlled. This can be done through several means; I would recommend using my ProtoTracer program which will ray trace a live 3D model of a Protogen face to the LED panel in real time!

An alternative is to use FastLED with Arduino and manually define the pixels in order.

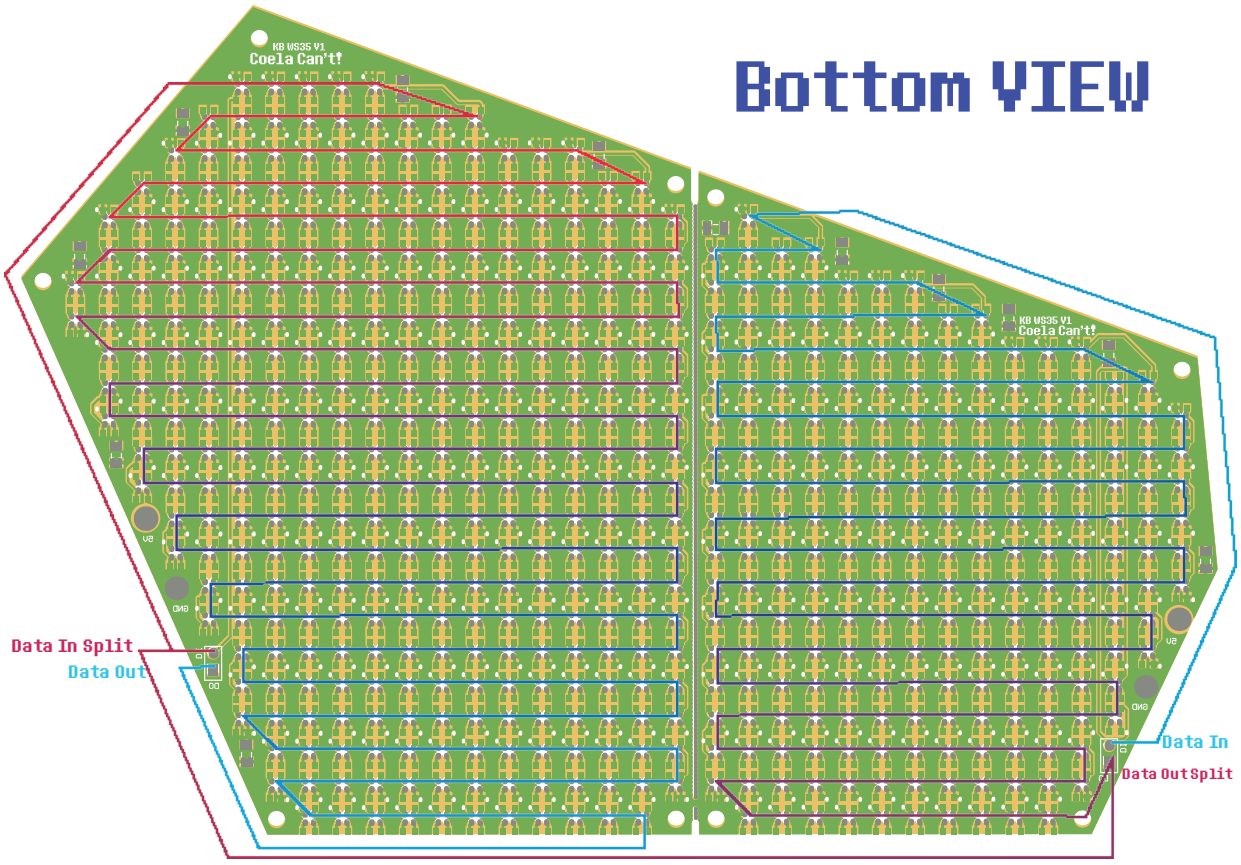
To increase the framerate the boards can be split in two with the perforation in the middle.

2.2 Top-Side LED Panel

As stated, before these panels are essentially LED strips, the order of the LEDs follows the pattern as below:

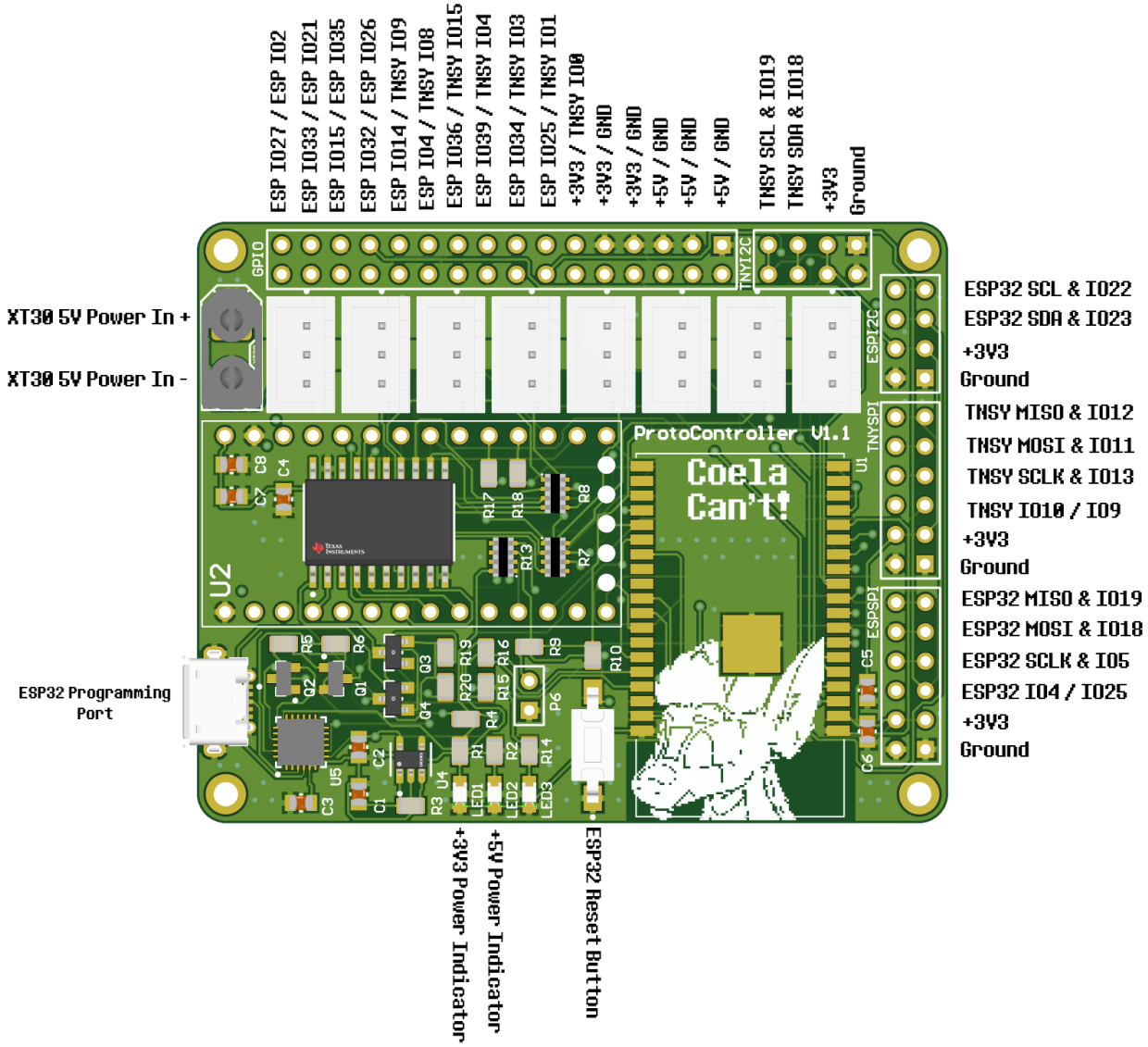


2.3 Bottom-Side LED Panel

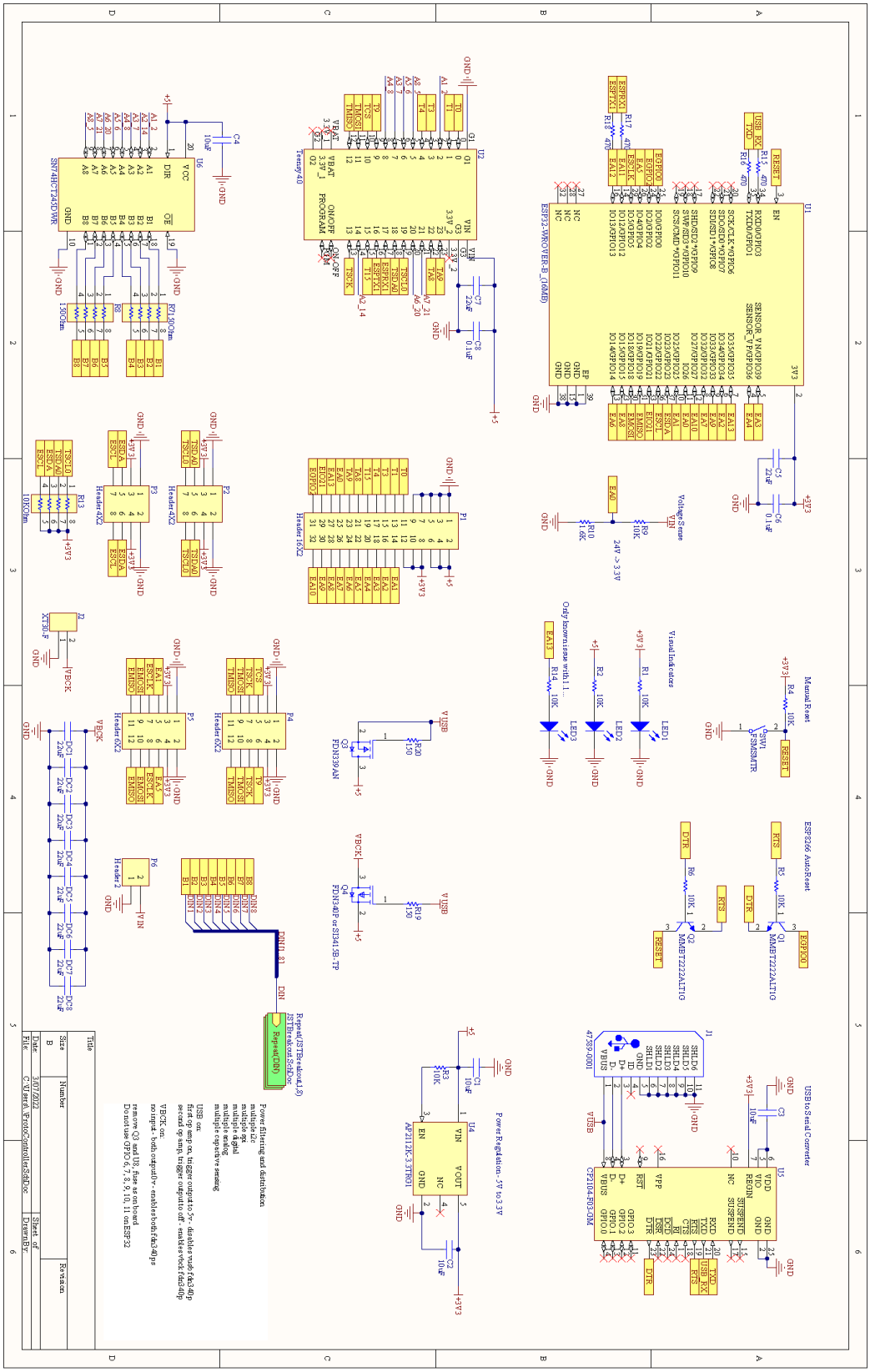


3 Protocontroller V1.1

3.1 Pinout

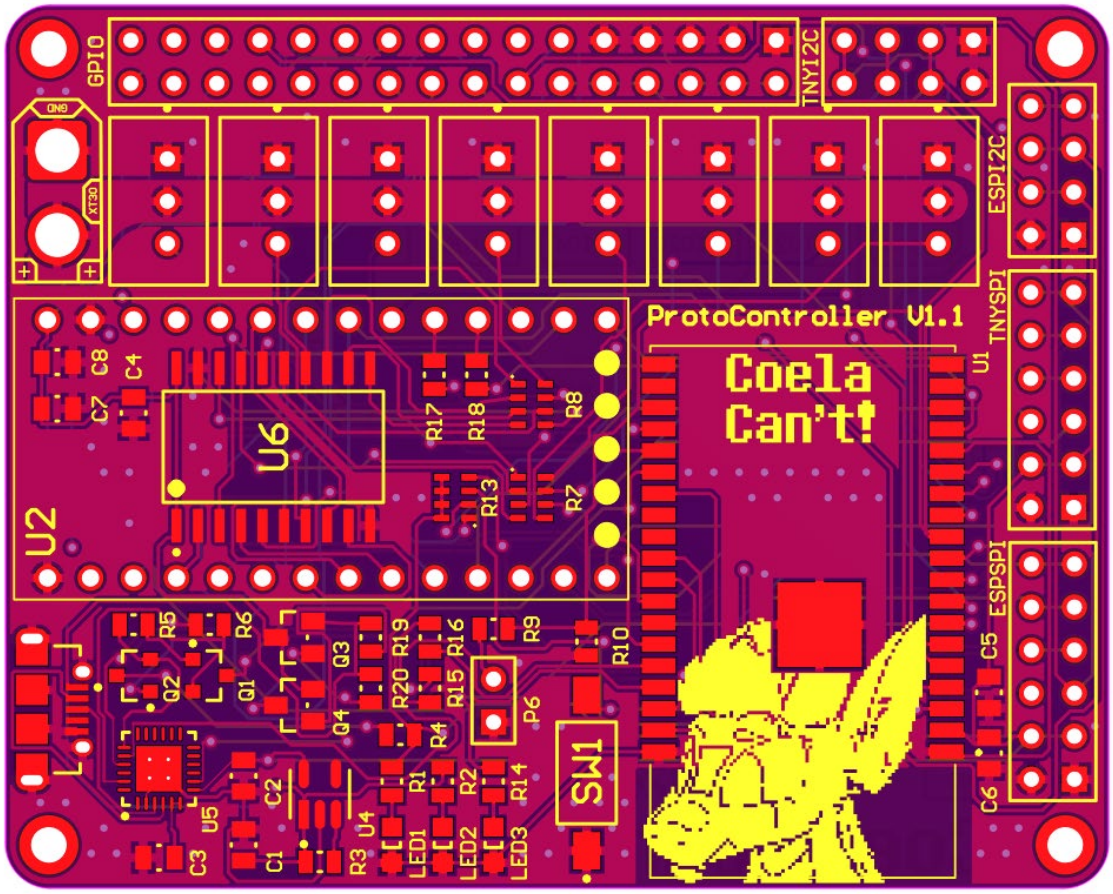


3.2 Schematic

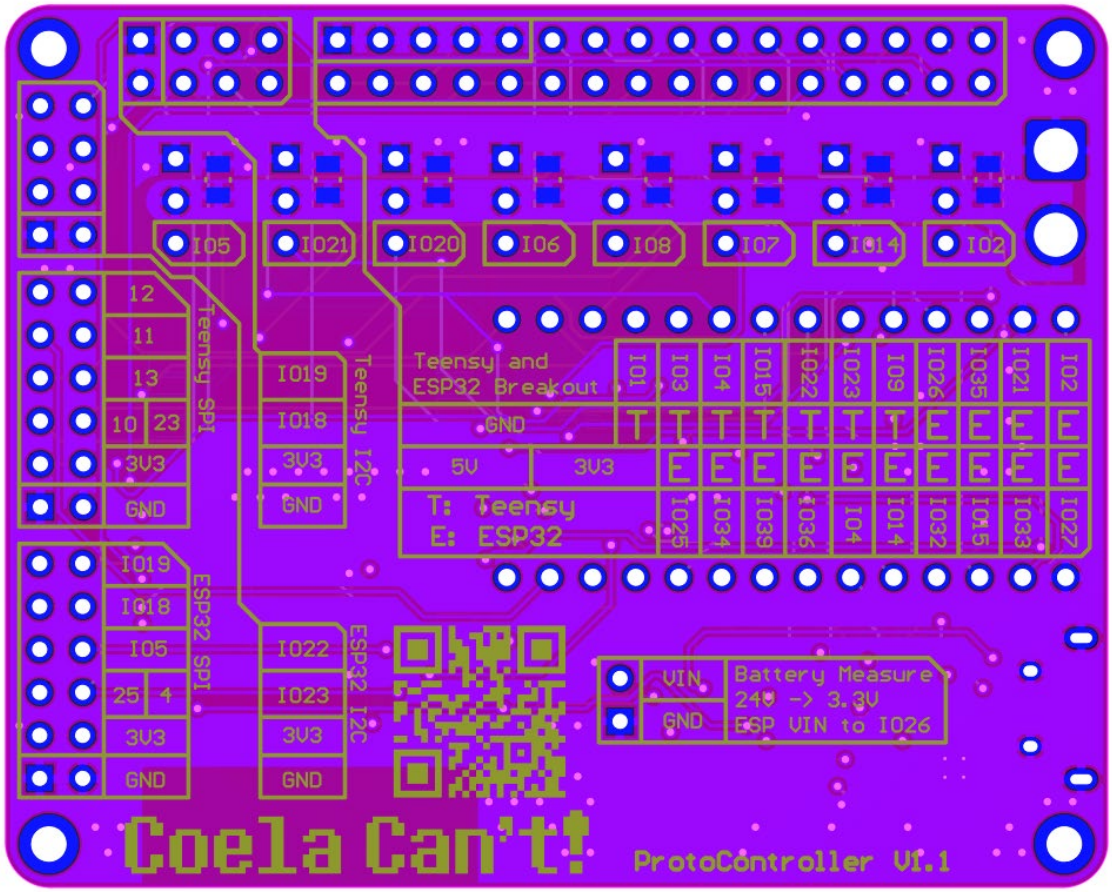


3.3 Physical Design

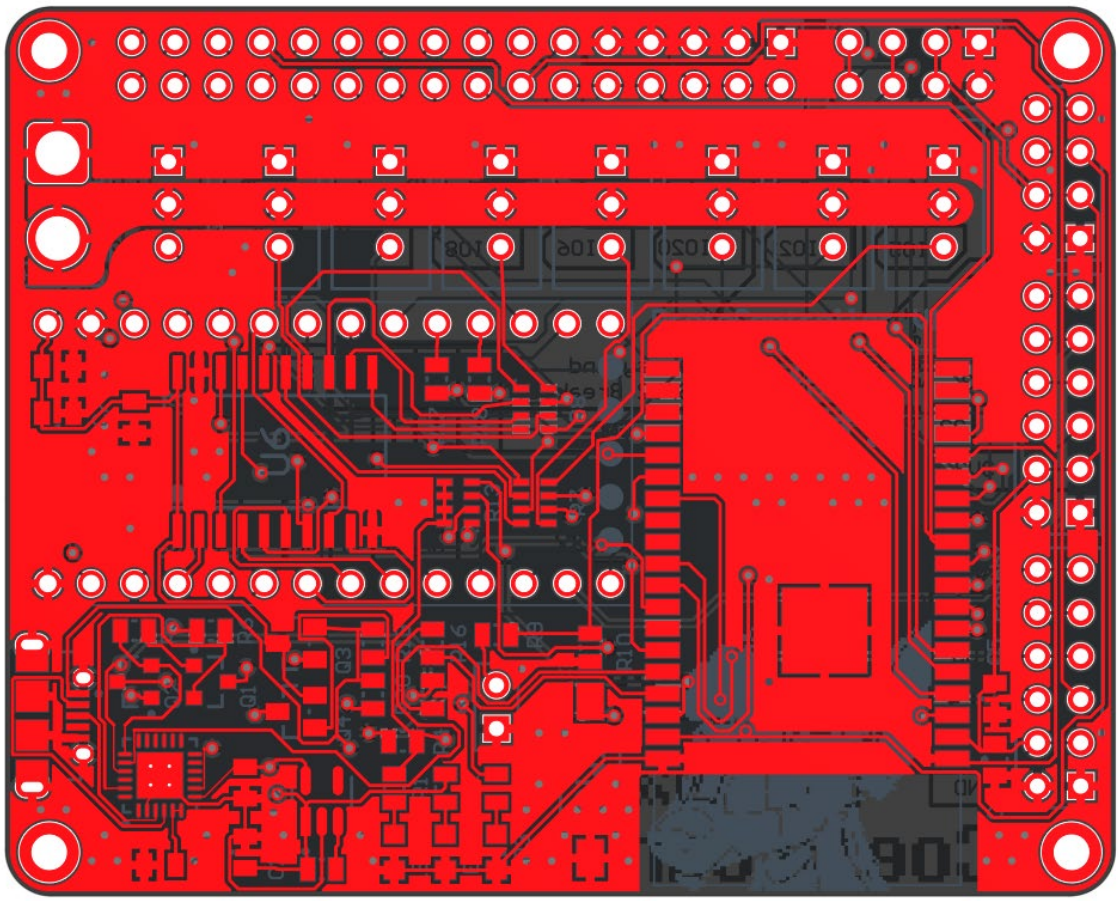
3.3.1 Top View



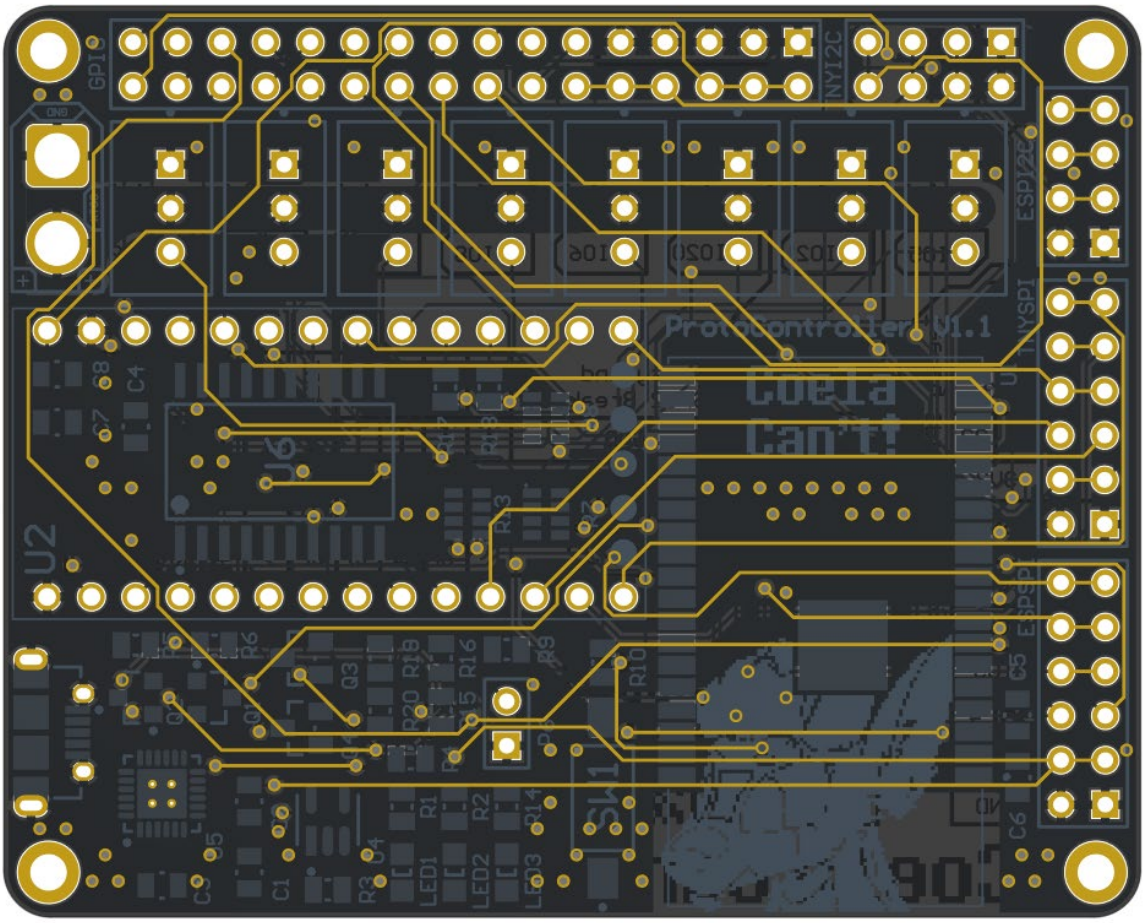
3.3.2 Bottom View



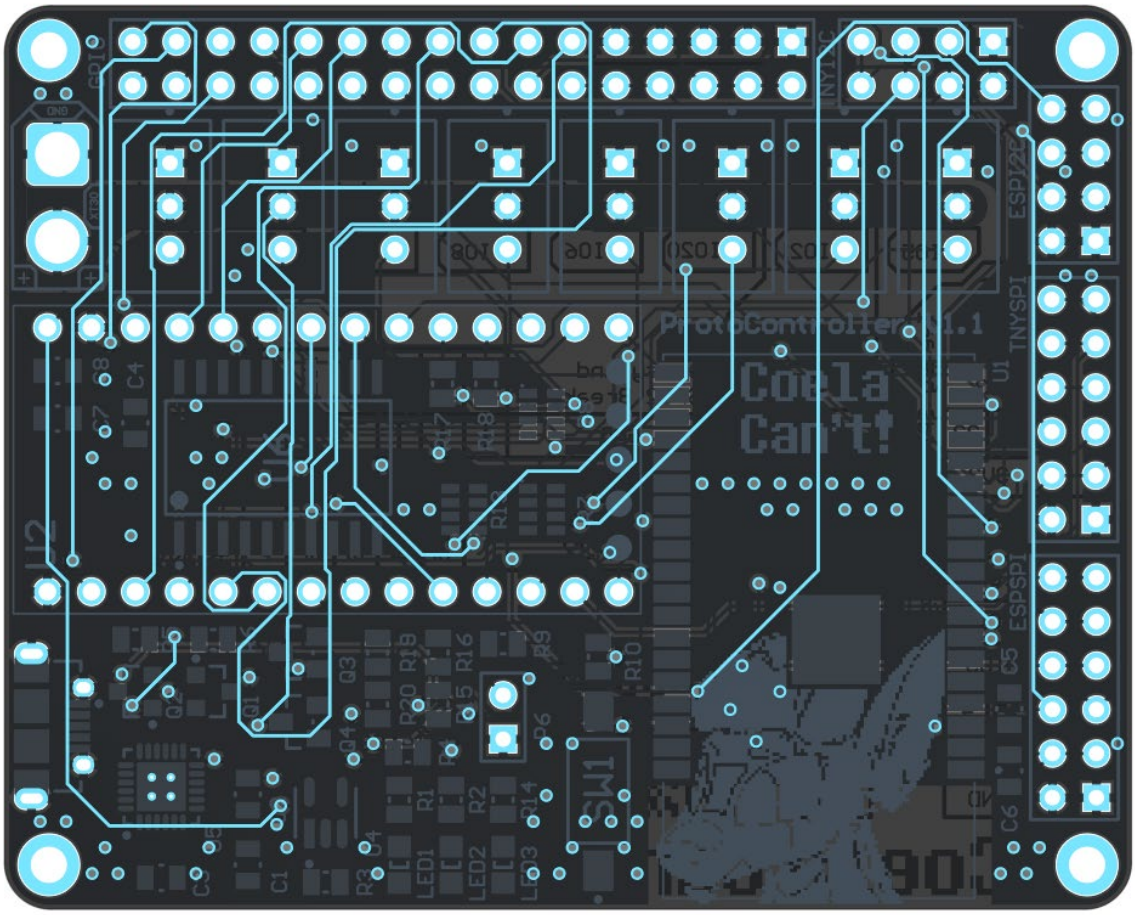
3.3.3 Isolated Top Layer View



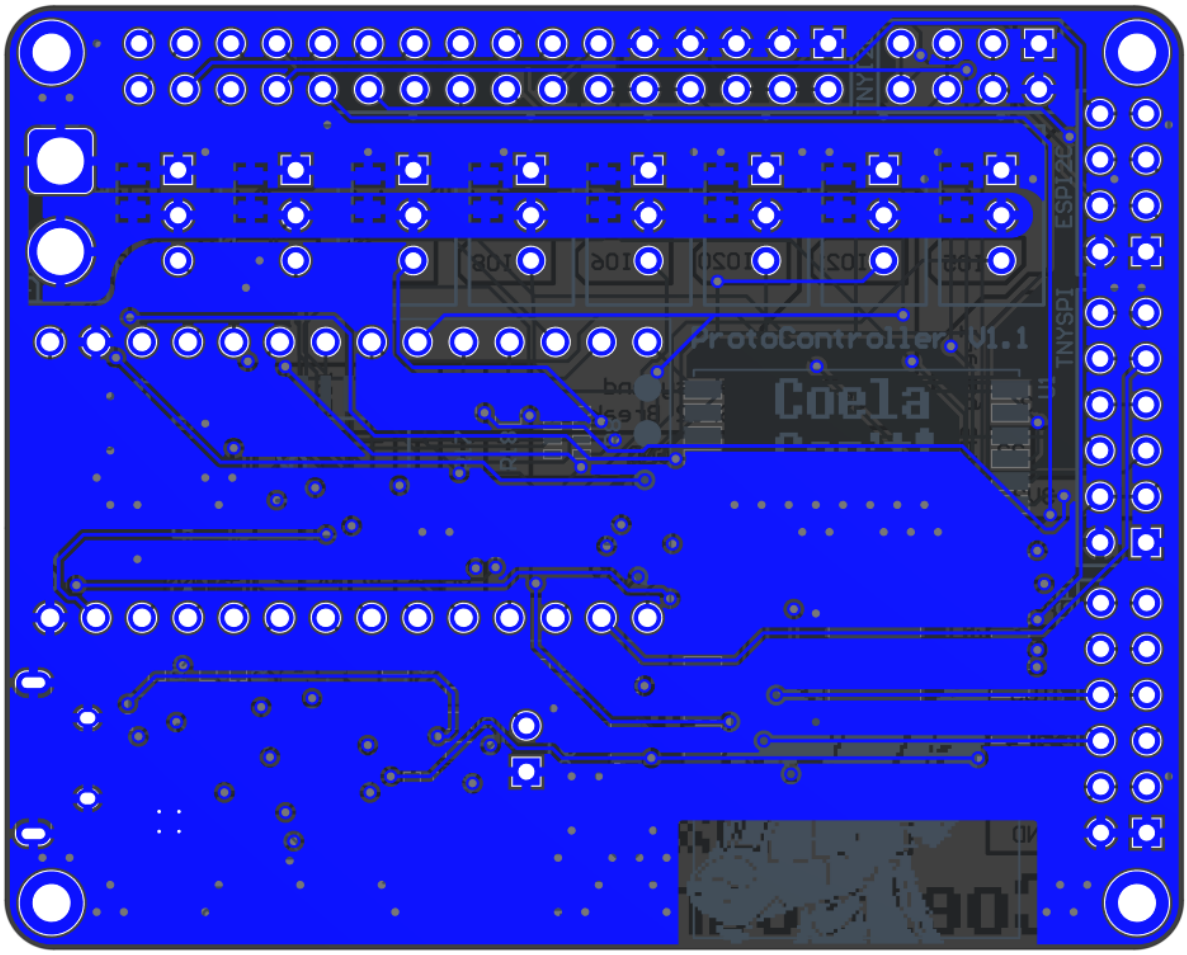
3.3.4 Isolated Signal Layer 1 View



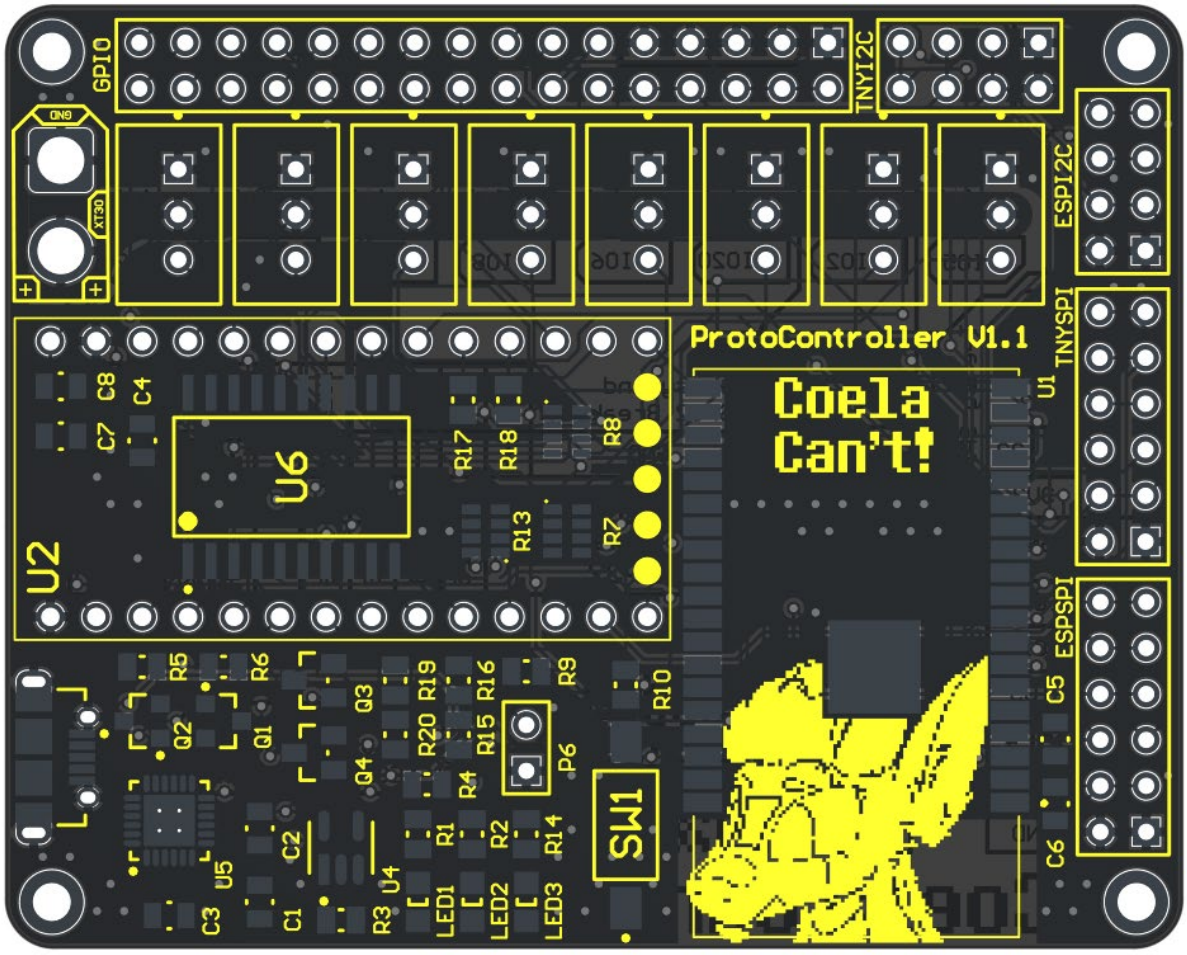
3.3.5 Isolated Signal Layer 2 View



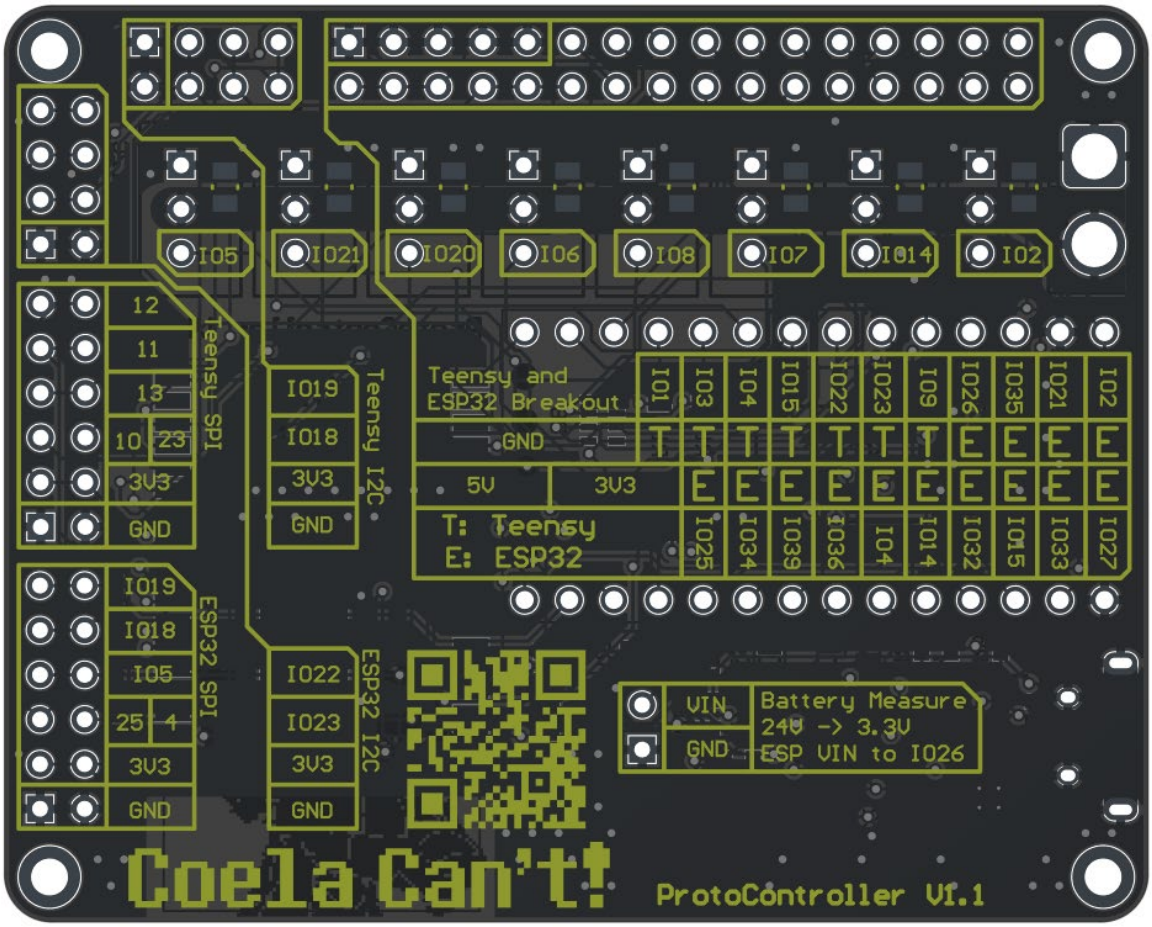
3.3.6 Isolated Bottom Layer View



3.3.7 Isolated Top Layer Overlay

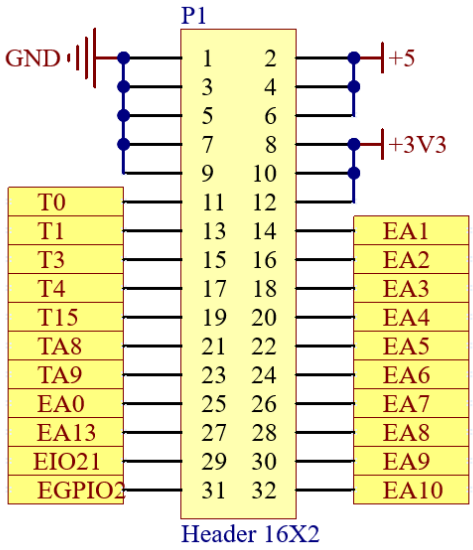


3.3.7 Isolated Bottom Layer Overlay (Flipped)

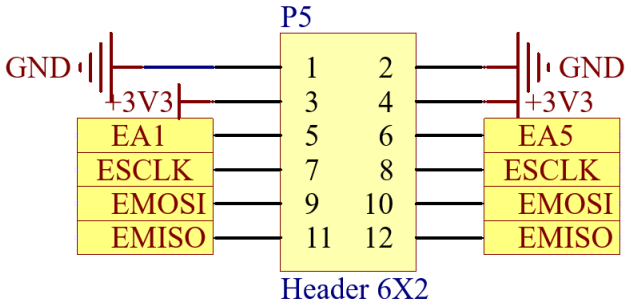


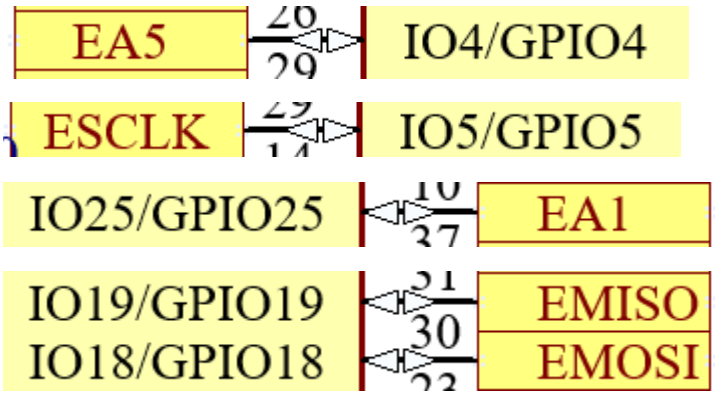
3.4 Header Pinouts

3.4.1 32 Pin Header

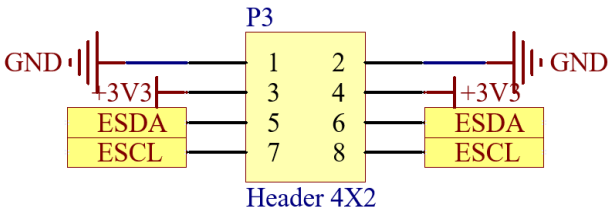


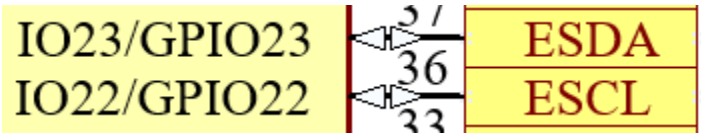
3.4.2 ESP32 SPI Header



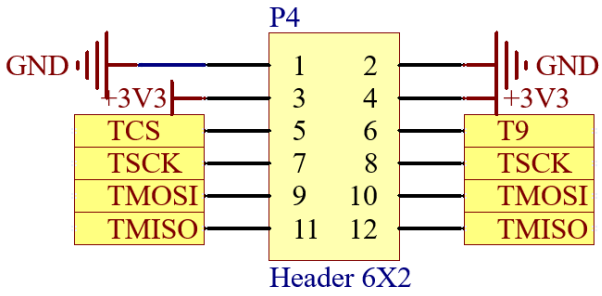


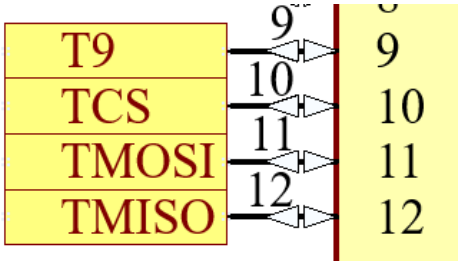
3.4.3 ESP32 I2C Header



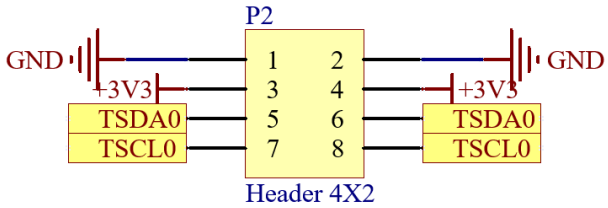


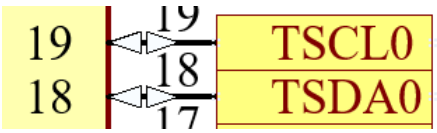
3.4.4 Teensy SPI Header





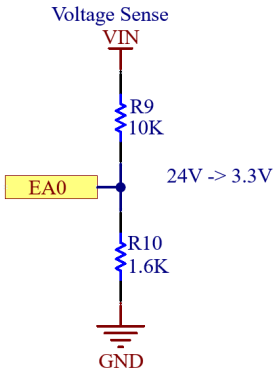
3.4.5 Teensy I2C Header



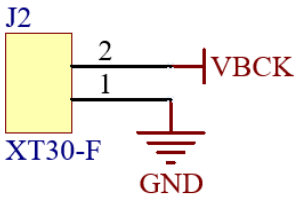


3.4.6 Battery Measurement Header

The battery input measurement uses a simple voltage divider to scale an input of 0V -> 24V down to 0V ->3.3V.

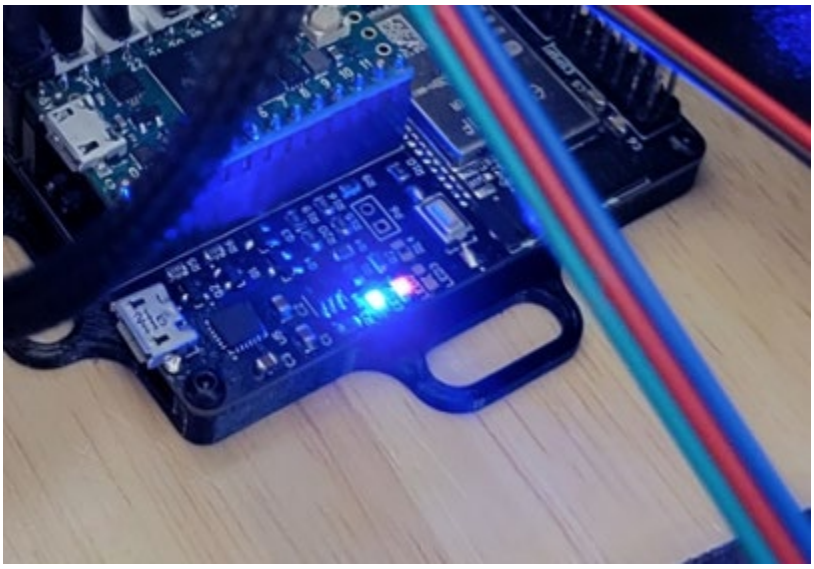


3.4.7 XT30 Power Input



3.5 Indicators

There are two in-use visual indicators on the controller, the left LED in this image indicates that the board's 3.3V supply is powered. The right LED indicates that the board's 5V supply is powered.



3.6 Power Setup

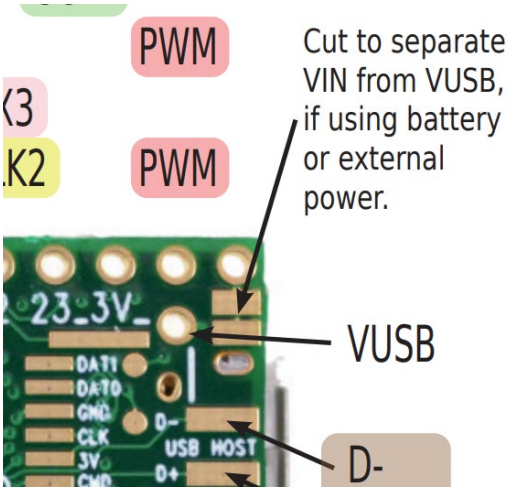
The full controller can be powered 3 different ways, two of which should be done, one should be prevented.

3.6.1 Powering the ESP32 for Programming

The ESP32 can be connected directly with the onboard USB Micro port, no changes need to be made to use it. It can be plugged in and programmed as-is.

3.6.2 Powering the Teensy 4.0 for Programming

For programming the Teensy 4.0, the bottom VIN from VUSB trace needs to be cut as to not provide power to the LED boards and cause damage while programming:



This is detailed in the top right corner of this diagram.

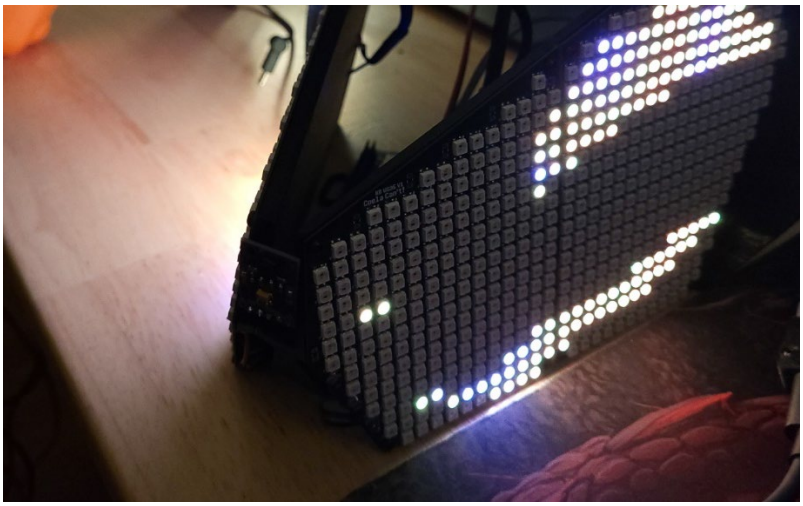
3.6.3 Powering the Controller for Usage

To use the controller, 5V must be provided to the XT30 connector, make sure you follow the standards for the XT30 polarity!

4 Sensors/Peripherals

4.1 Boop Sensor (APDS-9960)

The APDS-9960 is a Time of Flight sensor that uses an IR light to measure the distance to the object in front of it using the I2C communication protocol. This can be connected on either the Teensy I2C breakout or the ESP32 I2C breakout. The following picture shows the recommended location for the device:



4.2 MAX9814 Microphone

The MAX9814 electret microphone is a standard electret microphone with automatic gain compensation via an amplifier. This will pick up a varying range of sounds and not just your voice, so it is best to tune the software gain appropriately.

4.3 Control Button

The control button is a simple button that allows you to toggle between faces. There is no analog filtering on the button with the kit as the button debouncing is handled in code within the ButtonHandler class.

4.4 MPU6050

The MPU6050 is a motion processing unit that allows for reading from a 3-axis accelerometer and a 3-axis gyroscope. These inputs can be read in over I2C with either the Teensy or the ESP32 and be used to map and give more motion to the face.

4.5 I2C OLED Display

The OLED display is a small display for mounting within the visor for the wearer to see, it is used to display the current face being displayed on the outside, status of a battery, or anything else you like. This will communicate over I2C with either the Teensy or the ESP32.

5 Programming the Teensy

5.1 Getting Started with ProtoTracer

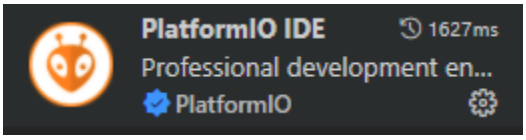
5.1.1 Install Applications

Install Visual Studio Code from here: https://code.visualstudio.com/

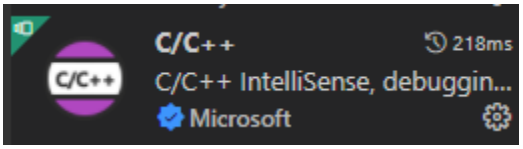
Install TeensyDuino from here: https://www.pjrc.com/teensy/td_download.html

5.1.2 Install Extensions

Install PlatformIO IDE under the extensions in Visual Studio Code:

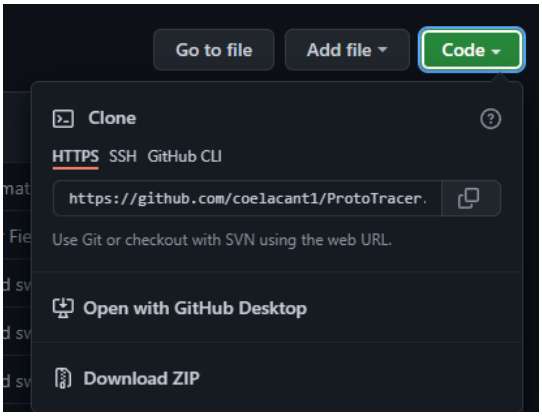


Install C/C++ extension:



5.1.3 Download the Codebase

Download ProtoTracer from here: https://github.com/coelacant1/ProtoTracer You can use either the Zip download or clone it to a local Git repository:



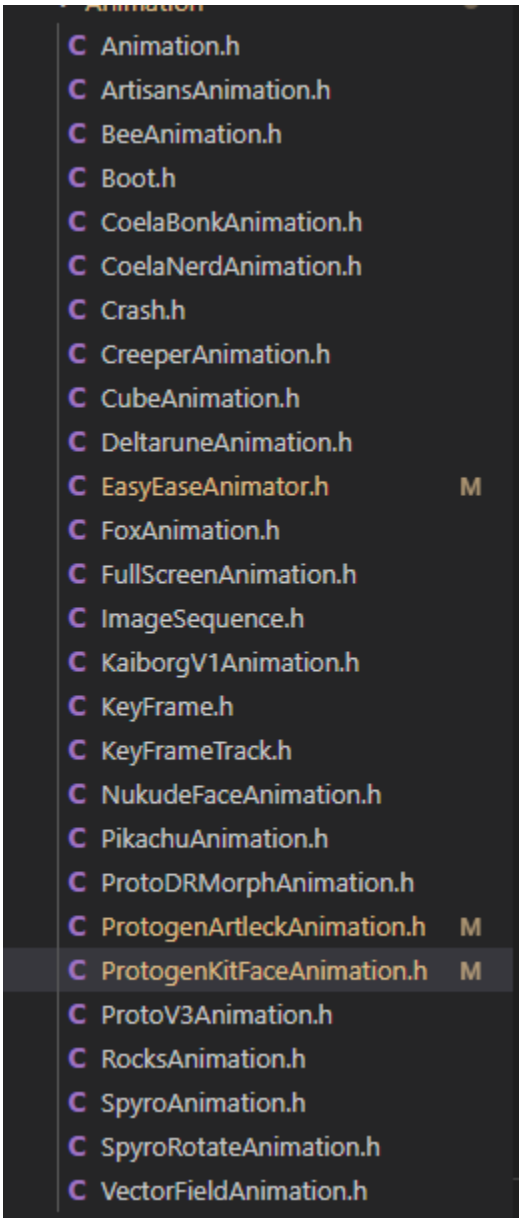
5.2 Loading a Controller

Depending on the status of the repository you will need to change the Main.cpp file in your repository. Use the following Main.cpp to load for your Protogen Controller using the ProtoController V1.1 or relevant controller:

```
1. //---------------- ANIMATIONS ---------------
2. #include "Animation\ProtogenKitFaceAnimation.h"
3.
4. //---------------- CONTROLLERS ---------------
5. #include "Controllers\KaiborgV1D1Controller.h"
6.
7. const uint8_t maxBrightness = 20;
8. Controller* controller = new KaiborgV1D1Controller(maxBrightness);
9. Animation* animation = new ProtogenKitFaceAnimation();
10.
11.      void setup() {
12.          Serial.begin(115200);
13.          Serial.println("\nStarting...");
14.
15.          controller->Initialize();
16.      }
17.
18.      void loop() {
19.          float ratio = (float)(millis() % 5000) / 5000.0f;
20.          animation->UpdateTime(ratio);
21.
22.          controller->Render(animation->GetScene());
23.
24.          controller->Display();
25.
26.          Serial.print("Animated in ");
27.          Serial.print(animation->GetAnimationTime(), 4);
28.
29.          Serial.print("s, Rendered in ");
30.          Serial.print(controller->GetRenderTime(), 4);
31.          Serial.println("s");
32.      }
```

5.3 Loading or Modifying an Animation

There are several previously created animations that you can load and modify into your Protogen. The recommended pre-created file is the ProtogenKitFaceAnimation.h which will work from the start with your included sensors. Alternative animations can be found under the Animations folder:



The animation includes everything to import your 3D face file, set the coloring, time the keyframes, map parameters to generators, listen for your inputs, and modify everything that will then be passed to the controller to render to your displays.

5.4 Controller Example

The following is an example controller that is used to import the cameras – which include the location of each pixel – the transform specifying where the camera is and which direction it is facing, as well as a list of the output pixels which stored the rendered information. Here is the parent class you must use to define a controller:

```
#pragma once

#include "..\Render\Camera.h"

class Controller {
private:
    const float softStart = 3000000;//microseconds
    long previousTime;
    Camera** cameras;
    uint8_t count = 0;
    float renderTime = 0.0f;
    uint8_t maxBrightness;
    bool isOn = false;

protected:
    uint8_t brightness;

    Controller(Camera** cameras, uint8_t count, uint8_t maxBrightness){
        this->cameras = cameras;
        this->count = count;
        this->maxBrightness = maxBrightness;
        previousTime = micros();
    }

public:
    void Render(Scene* scene){
        previousTime = micros();

        if (!isOn && previousTime < softStart){
            brightness = map(previousTime, 0, softStart, 0, maxBrightness);
        }
        else if (!isOn){
            brightness = maxBrightness;
            isOn = true;
        }

        for (int i = 0; i < count; i++){
            cameras[i]->Rasterize(scene);
        }

        renderTime = ((float)(micros() - previousTime)) / 1000000.0f;
    }

    virtual void Initialize() = 0;
    virtual void Display() = 0;

    float GetRenderTime(){
        return renderTime;
    }

};
```

This class requires you to override the Initialize and Display functions to specify how your controller is set up and how it updates your display:

```
1.  #include <Arduino.h>
2.  #include <OctoWS2811.h>
3.
4.  #include "Controller.h"
5.  #include "Render/Camera.h"
6.  #include "Flash/PixelGroups/KaiborgV1Pixels.h"
7.
8.  const int ledsPerStrip = 346;
9.  DMAMEM int displayMemory[346 * 6];
10. int drawingMemory[346 * 6];
11. const int config = WS2811_GRB | WS2811_800kHz;
12.
13. OctoWS2811 leds(ledsPerStrip, displayMemory, drawingMemory, config);
14.
15. class KaiborgV1D1Controller : public Controller {
16. private:
17.     CameraLayout cameraLayout = CameraLayout(CameraLayout::ZForward,
    CameraLayout::YUp);
18.
19.     Transform camRghtTransform = Transform(Vector3D(), Vector3D(0, 0, -500.0f),
    Vector3D(1, 1, 1));
20.     Transform camLeftTransform = Transform(Vector3D(), Vector3D(0, 0, -500.0f),
    Vector3D(1, 1, 1));
21.
22.     PixelGroup camRghtPixels = PixelGroup(KaiborgV1Pixels, 571, PixelGroup::ZEROTOMAX);
23.     PixelGroup camLeftPixels = PixelGroup(KaiborgV1Pixels, 571, PixelGroup::MAXTOZERO);
24.
25.     Camera camRght = Camera(&camRghtTransform, &cameraLayout, &camRghtPixels);
26.     Camera camLeft = Camera(&camLeftTransform, &cameraLayout, &camLeftPixels);
27.
28.     Camera* cameras[2] = { &camRght, &camLeft };
29.
30. public:
31.     KaiborgV1D1Controller(uint8_t maxBrightness) : Controller(cameras, 2,
    maxBrightness){}
32.
33.     void Initialize() override{
34.         leds.begin();
35.         leds.show();
36.     }
37.
38.     void Display() override {
39.         int offset;
40.
41.         for (int i = 0; i < 571; i++){
42.             camLeftPixels.GetPixel(i)->Color = camLeftPixels.GetPixel(i)-
    >Color.Scale(brightness);
43.             camRghtPixels.GetPixel(i)->Color = camRghtPixels.GetPixel(i)-
    >Color.Scale(brightness);
44.         }
45.
46.         for (int i = 0; i < 571; i++) {
47.             if (i < 346){
48.                 offset = i + 225;
49.
```

```
                leds.setPixel(i + 346 * 4, camLeftPixels.GetPixel(offset)->Color.R, camLeftPixels.GetPixel(offset)->Color.G, camLeftPixels.GetPixel(offset)->Color.B);//Pin 7
                leds.setPixel(i + 346 * 5, camRghtPixels.GetPixel(i)->Color.R, camRghtPixels.GetPixel(i)->Color.G, camRghtPixels.GetPixel(i)->Color.B);//Pin 8
            }
            else{
                offset = i - 346;

                leds.setPixel(i + 346 * 6 - 346, camLeftPixels.GetPixel(offset)->Color.R, camLeftPixels.GetPixel(offset)->Color.G, camLeftPixels.GetPixel(offset)->Color.B);//Pin 8
                leds.setPixel(i + 346 * 7 - 346, camRghtPixels.GetPixel(i)->Color.R, camRghtPixels.GetPixel(i)->Color.G, camRghtPixels.GetPixel(i)->Color.B);//Pin 8
            }
        }

        leds.show();
    }

    void SetPixels(uint8_t strip, uint16_t led, RGBColor rgb){
        leds.setPixel(346 * strip + led, rgb.R, rgb.G, rgb.B);//Pin 8
    }
};
```

5.5 Animation Example

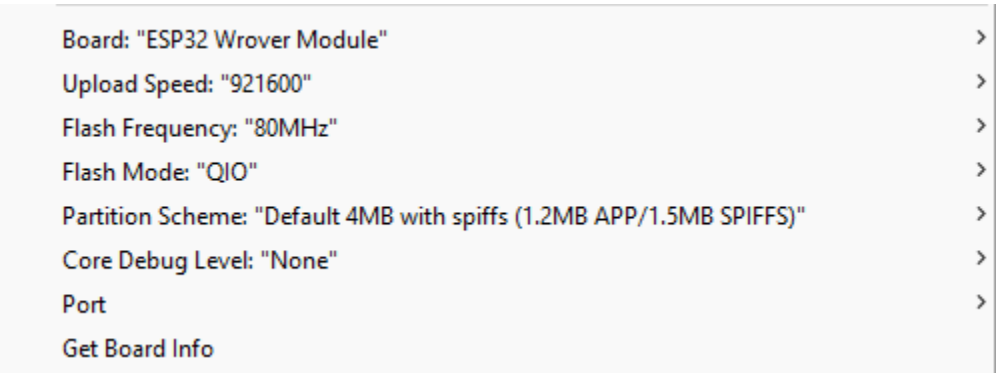
The following is an example controller which loads in a cube with a depth material and rotates in space based on the animation completion ratio input to the Update function:

```
#pragma once

#include "Animation.h"
#include "..\Objects\Cube.h"
#include "..\Materials\DepthMaterial.h"
#include "..\Materials\LightMaterial.h"
#include "..\Math\FunctionGenerator.h"

class CubeAnimation : public Animation{
private:
    Cube cube;
    DepthMaterial dMat = DepthMaterial(DepthMaterial::Z, 100.0f, 600.0f);
    LightMaterial lMat = LightMaterial();
    FunctionGenerator fGenRotation = FunctionGenerator(FunctionGenerator::Sine, -360.0f, 360.0f, 6.0f);
    FunctionGenerator fGenScale = FunctionGenerator(FunctionGenerator::Sine, 0.25f, 0.75f, 4.0f);

public:
    CubeAnimation() : Animation(1) {
        scene->AddObject(cube.GetObject());

        cube.GetObject()->SetMaterial(&dMat);
    }

    void FadeIn(float stepRatio) override {}
    void FadeOut(float stepRatio) override {}

    void Update(float ratio) override {
        float x = fGenRotation.Update();
        float sx = fGenScale.Update();

        Quaternion rotation = Rotation(EulerAngles(Vector3D(x, ratio * 720.0f, 0), EulerConstants::EulerOrderXZYS)).GetQuaternion();

        cube.GetObject()->ResetVertices();

        cube.GetObject()->GetTransform()->SetRotation(rotation);
        cube.GetObject()->GetTransform()->SetScale(Vector3D(sx, sx, sx));
        cube.GetObject()->GetTransform()->SetPosition(Vector3D(125.0f, 75.0f, 600.0f));

        cube.GetObject()->UpdateTransform();
    }
};
```

6 Programming the ESP32

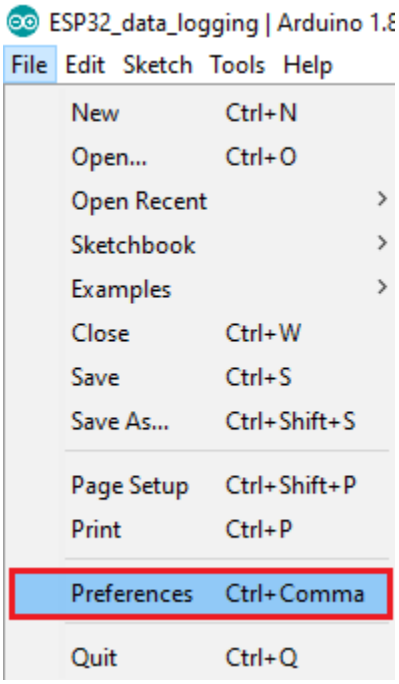
6.1 Programming in Arduino

Install the ESP32 boards in the board manager. Use the following settings for uploading on the ESP32:



6.1.1 Installing the ESP32 Boards in Board Manager

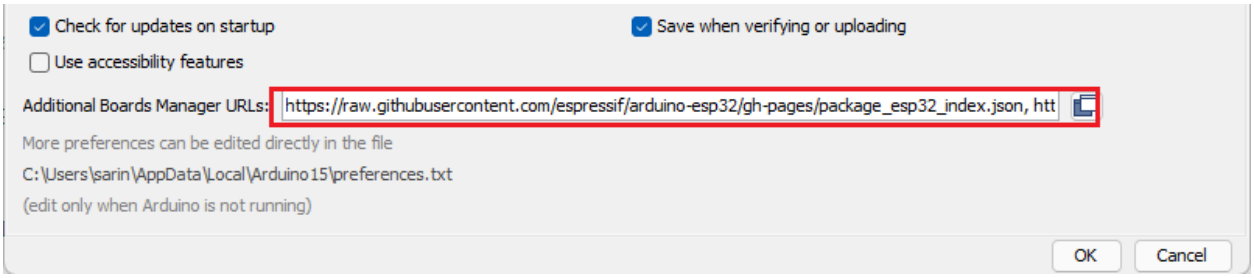
Go to File -> Preferences:



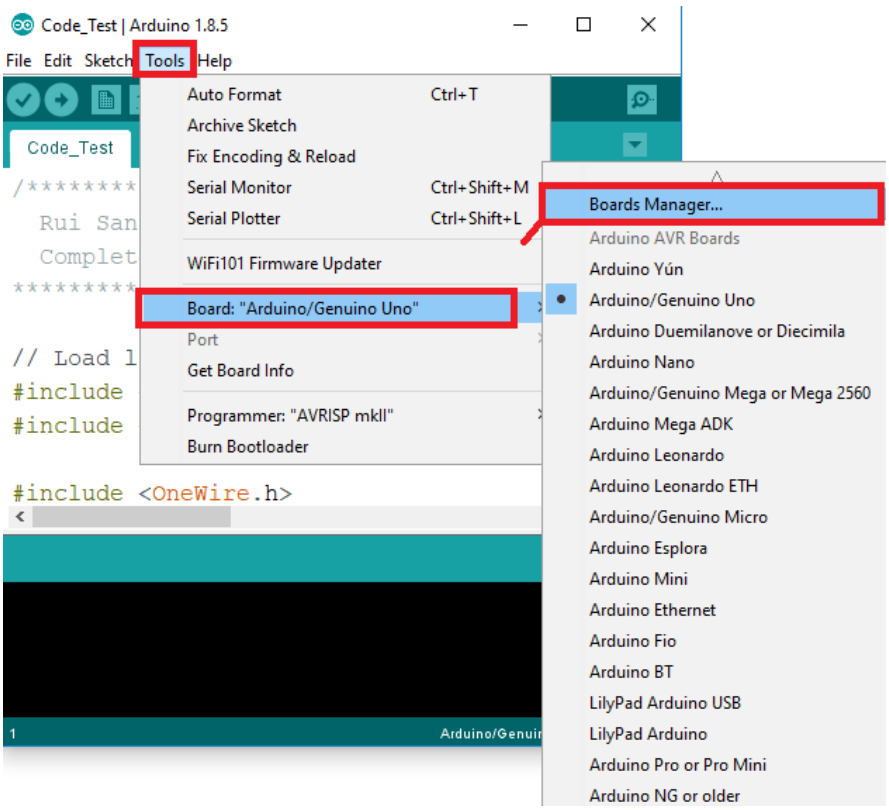
Enter the following into the Additional Board Manager URL field:

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

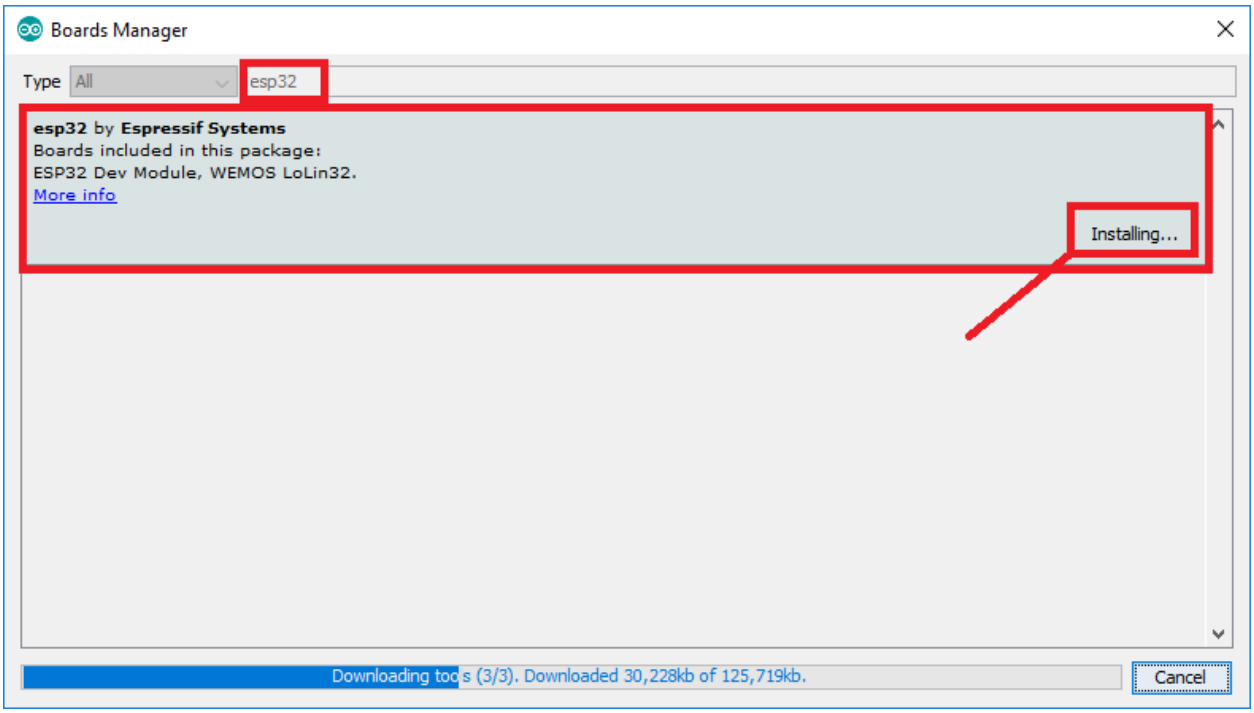
Click the OK button:



Open the Boards manager:



Search for ESP32 and install button for the “ESP32 by Espressif Systems”:


Boards Manager
Type
All
esp32
esp32 by Espressif Systems
Boards included in this package:
ESP32 Dev Module, WEMOS LoLin32.
More info
Installing...
Downloading tools (3/3). Downloaded 30,228kb of 125,719kb.
Cancel

6.2 Example Communication Code for ESP32

This codebase requires you to install Adafruit Sensor, Adafruit BNO055, SerialTransfer, and Adafruit APDS9960. The most up-to-date version of this code can be found at: https://github.com/coelacant1/ProtoControllerESP32

6.2.1 ProtoControllerESP32.ino

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <SerialTransfer.h>
#include <Adafruit_APDS9960.h>
#include "MorphDecoder.h"

#define PROTOCONTROLLERV1_1

MorphDecoder morphDecoder;

Adafruit_APDS9960 apds;

uint16_t BNO055_SAMPLERATE_DELAY_MS = 10;

Adafruit_BNO055 bno = Adafruit_BNO055(55, 0x28);

static struct ESP32Data {
    float oW;
    float oX;
    float oY;
    float oZ;
    uint16_t r;
    uint16_t g;
    uint16_t b;
    uint16_t c;
    uint8_t m;
    uint8_t d;
    float ratio;
    uint8_t mode;
} e32Data;

const uint8_t button1 = 32;
long previousMillis = 0;
bool wasUsed = false;
bool useApds = false;
bool useBno = false;

void IRAM_ATTR isr() {
  if(millis() - previousMillis > 250){
    if(!wasUsed){
      e32Data.mode += 1;
      if (e32Data.mode > 8) e32Data.mode = 0;

      previousMillis = millis();
      wasUsed = true;
    }
    else{
      wasUsed = false;
    }
  }
```

```
}

SerialTransfer dataTransfer;

void setup(void){
  Wire.begin(23, 22);
  Serial.begin(115200);

  #ifdef PROTOCONTROLLERV1_1
  Serial1.begin(9600, SERIAL_8N1, 12, 13);//For V1-1
  #else
  Serial1.begin(9600, SERIAL_8N1, 12, 19);
  #endif
  Serial.println("Orientation Sensor Test"); Serial.println("");

  pinMode(button1, INPUT_PULLUP);
  attachInterrupt(button1, isr, RISING);

  dataTransfer.begin(Serial1);

  if (!bno.begin()){
    Serial.println("Ooops, no BNO055 detected ... Check your wiring or I2C ADDR!");
  }
  else{
    Serial.println("BNO initialized!");
    useBno = true;
  }

  if(!apds.begin()){
    Serial.println("Failed to initialize APDS! Please check your wiring.");
  }
  else{
    Serial.println("APDS initialized!");
    useApds = true;

    //enable color sensign mode
    apds.enableColor(true);
    apds.enableProximity(true);
  }

  delay(1000);
}

void loop(void){
  if(useBno){
    imu::Quaternion quat = bno.getQuat();

          e32Data.oW = quat.w();
          e32Data.oX = quat.x();
          e32Data.oY = quat.y();
          e32Data.oZ = quat.z();
        }

        if(useApds){
          apds.getColorData(&e32Data.r, &e32Data.g, &e32Data.b, &e32Data.c);
          e32Data.d = apds.readProximity();
        }


        e32Data.ratio = (float)(millis() % 5000) / 5000.0f;

```

```
        Serial.print(e32Data.ratio); Serial.print('\t'); Serial.println(e32Data.mode);
        morphDecoder.Fetch();

        uint16_t sendSize = 0;

        sendSize = dataTransfer.txObj(e32Data, sendSize);
        dataTransfer.sendData(sendSize);

        delay(20);
      }
```

6.2.2 MorphDecoder.h

```
#pragma once

#include <arduinoFFT.h>
#include "KalmanFilter.h"

class MorphDecoder{
public:
  enum MouthShape{
    EE,
    E,
    AE,
    AH,
    OO,
    OPEN
  };

private:
  static const uint16_t sampleFreqency = 5000;
  static const uint16_t samplePeriod = round(1000000 * (1.0f / sampleFreqency));
  static const uint16_t samples = 512;//Multiple of 2 only
  static const uint8_t micPin = 34;//25
  uint16_t lowThreshold = 50;

  arduinoFFT FFT = arduinoFFT();
  SemaphoreHandle_t fftMutex;
  double noisData[samples];
  double realData[samples];
  double imagData[samples];
  unsigned long previousTime = 0;

  void SampleMicrophone(){
    for(uint16_t i = 0; i < samples; i++){
      realData[i] = analogRead(micPin);
      imagData[i] = 0;
    }

    previousTime = micros();

    while (micros() < (previousTime + samplePeriod)){ delay(0); }
  }

  void CalculateFFT(){
    FFT.DCRemoval(realData, samples);
    FFT.Windowing(realData, samples, FFT_WIN_TYP_FLT_TOP, FFT_FORWARD);
    FFT.Compute(realData, imagData, samples, FFT_FORWARD);
    FFT.ComplexToMagnitude(realData, imagData, samples);
  }
```

```
48.
49.   void RemoveNoise(){
50.     for(uint16_t i = lowThreshold; i < samples / 2; i++){
51.       noisData[i] = noisData[i] * 0.95 + realData[i] * 0.05;
52.       realData[i] = realData[i] - noisData[i];
53.       realData[i] = realData[i] < 0 ? 0 : realData[i];
54.     }
55.   }
56.
57.   void SmoothFFT(){
58.     KalmanFilter kf = KalmanFilter(0.05, 20);
59.
60.     for(uint16_t i = lowThreshold; i < samples / 2; i++){
61.       realData[i] = kf.Filter(realData[i]);
62.     }
63.   }
64.
65.   void FFTUpdate(){
66.     while(true){ // infinite loop
67.       if(xSemaphoreTake(fftMutex, (TickType_t)5) == pdTRUE){
68.         SampleMicrophone();
69.         //CalculateFFT();
70.         //CombineFFT();
71.         //RemoveNoise();
72.         //SmoothFFT();
73.
74.         xSemaphoreGive(fftMutex);
75.       }
76.
77.       vTaskDelay(50 / portTICK_PERIOD_MS);//5ms delay
78.     }
79.
80.     vTaskDelete(NULL);
81.   }
82.
83.   static void FFTWrapper(void* _this){
84.     ((MorphDecoder*)_this)->FFTUpdate();
85.   }
86.
87.   void CreateFFTTask(){
88.     xTaskCreate(this->FFTWrapper, "Update FFT", 16384, this, 1, NULL);
89.   }
90.
91. public:
92.   MorphDecoder(){
93.     pinMode(micPin, INPUT);
94.     fftMutex = xSemaphoreCreateMutex();
95.
96.     CreateFFTTask();
97.   }
98.
99.   void Fetch(){
100.          if(xSemaphoreTake(fftMutex, (TickType_t)5) == pdTRUE){
101.            for(uint16_t i = lowThreshold; i < samples / 2; i++){
102.              //Serial.println(realData[i]);
103.            }
104.
105.            xSemaphoreGive(fftMutex);
106.          }
107.        }
108.      };
```

6.3 Example Communication Code for Teensy 4

This code will read the struct that was transferred over serial to the Teensy 4 from the ESP32.

6.3.1 TeensyTest.ino

```
#include "SerialInterpreter.h"

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

  SerialInterpreter::Initialize();
}

void loop() {
  SerialInterpreter::Update();

  String dataString = SerialInterpreter::GetQuaternion() + "," +
SerialInterpreter::GetColor() + "," + SerialInterpreter::GetRatio() + "," +
SerialInterpreter::GetMode() + "," + SerialInterpreter::GetMorph();

  Serial.println(dataString);
}
```

6.3.2 SerialInterpreter.h

```
#pragma once

#include <SerialTransfer.h>

class SerialInterpreter{
private:
    static bool baseRotationSet;

    static struct ESP32Data {
        float oW;
        float oX;
        float oY;
        float oZ;
        uint16_t r;
        uint16_t g;
        uint16_t b;
        uint16_t c;
        uint8_t m;
        uint8_t d;
        float ratio;
        uint8_t mode;
    } e32Data;

    static SerialTransfer dataTransfer;

public:
    static void Initialize(){
        Serial4.begin(9600);
        dataTransfer.begin(Serial4, true);//_debug = true
        baseRotationSet = false;
    }

```

```
33.     static String GetColor(){
34.         return String(e32Data.r) + "," + String(e32Data.g) + "," + String(e32Data.b);
35.     }
36.
37.     static String GetQuaternion(){
38.         return String(e32Data.oW) + "," + String(e32Data.oX) + "," + String(e32Data.oY)
    + "," + String(e32Data.oZ);
39.     }
40.
41.     static float GetRatio(){
42.         return e32Data.ratio;
43.     }
44.
45.     static uint8_t GetMode(){
46.         return e32Data.mode;
47.     }
48.
49.     static uint8_t GetMorph(){
50.         return e32Data.m;
51.     }
52.
53.     static void Update(){
54.         if(dataTransfer.available()){
55.             uint16_t receiveSize = 0;
56.
57.             receiveSize = dataTransfer.rxObj(e32Data, receiveSize);
58.         }
59.     }
60.
61. };
62.
63. bool SerialInterpreter::baseRotationSet;
64.
65. SerialTransfer SerialInterpreter::dataTransfer;
66. SerialInterpreter::ESP32Data SerialInterpreter::e32Data;
```

7 General Recommendations and Notes

Do not use external power to the XT30 connector, unless it is 5V directly, the controller should be able to handle 4.5V to 5.5V but anything else is outside of specification.

Do not drop the LED panels. They should be able to handle a reasonable amount of wear-and-tear but only within reason.

Do not short circuit the IO pins. Be careful about which pins are in use and are listed at the beginning of the guide.

Make sure to leave the Kapton tape between the split panel edges, this ensures that the power planes do not touch and short out. There should be a reasonable gap without this tape, but it is there in case.

If you de-solder the connections going to the LED boards, ensure that the polarity is correct. The LEDs have reverse voltage protection, but it is best not to test it!

The LED boards are only meant for use between -25C and +85C but the recommended range would be 5C to 30C for tested operation.

Be careful leaving the electronics in a humid environment.

If you have a faulty LED board, please contact me before attempting repairs. The LEDs are easy to replace with the proper equipment but without this equipment you could cause more damage.

If you’re electronics get submerged in water disconnect power as soon as possible. The LED boards should not be damaged but will draw excess current from the rest of your circuit. I have tested the LED boards submerged entirely under water without issue but do not rely on this. The controllers and sensors will be damaged by water.