Coela Can't!

Protocontroller V2:

Getting Started

V1.0.2023 – Coela Can't!

Contents

1 Information

1.1 Overview

The Protocontroller V2 operates with the ProtoTracer codebase, which harnesses the SmartMatrix Shield Libraries and OctoWS2811 libraries for LED control. ProtoTracer 3D renders FBX files in real-time. The controller is equipped with I2C connections for sensor integration, a UART port designed for ESP32 communication enabling wireless control, IO headers with power, and PWM fan headers for thermal management.

This device draws inspiration from the SmartMatrix Shield for Teensy 4 and inherits its software compatibility, ensuring a broad range of existing libraries and applications can be utilized. As well as all software compatibility with the PJRC OctoWS2811 adaptor.

1.1.1 Teensy 4.0 Microcontroller Core:

At the core of the controller is a high-performance Teensy 4.0 board. It executes the control software for the LED panels and interfaces. Its extensive I/O capabilities and powerful processor allow for complex control schemes and rapid data handling.

1.1.2 HUB75 LED Panel Interface:

The HUB75 panel communication in this LED controller design draws its inspiration from the Smart LED Shield for Teensy 4. It facilitates the connection of a Teensy 4.0 or 4.1 to drive HUB75 panels, leveraging 5V buffers on all signal lines to ensure signal integrity, crucial for panels with 1/32 scan rates that require an additional address line. The controller utilizes nine I/O pins from the Teensy to manage the display and includes two pins specifically for APA102 LED connectivity. Power delivery to the Teensy and buffers is managed externally to avoid overloading the shield, ensuring it remains a compact yet capable platform for sophisticated LED control. Compatibility with the SmartMatrix Library is maintained, offering users the flexibility to adapt the pin assignments as needed for their projects.

1.1.3 Sensor Integration via I2C:

Five I2C ports with integrated pull-up resistors are provided, enabling multiple sensor connections for applications requiring environmental feedback or interactive control. Each port can handle standard I2C communication with sensors.

1.1.4 Serial Communication through UART:

There is a UART port with 470-ohm inline resistors for serial communication, allowing for reliable data transmission with external devices.

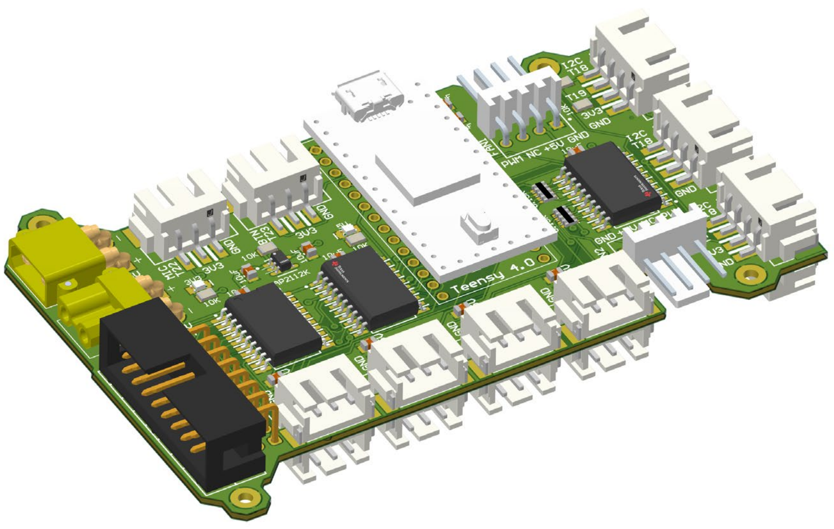
1.1.5 Individually Addressable LEDs:

In this controller design, the signal integrity for WS2812 RGB LEDs is ensured by a 74HCT245 buffer chip and 100-ohm impedance matching resistors. Unlike the OctoWS2811 Adaptor that uses CAT6 and RJ-45 connectors, this design opts for JST XH 2.54 3-pin headers for each LED connection, providing both 5V power and ground. This alternative connector choice simplifies individual LED strip connections, offering a more modular and customizable approach to LED array assembly, while maintaining high-quality signal transmission for precise lighting control.

1.1.6 Power Management:
The design incorporates an XT30 input for a 5V supply and an XT30 output, also at 5V, to power external devices. Voltage regulation ensures stable operation, and protection circuitry safeguards against electrical anomalies.

1.1.7 Analog and Digital I/Os:
Two analog inputs and two digital I/Os are broken out with 3.3V reference and ground connections, enabling the interfacing of various sensors and peripherals.

1.1.8 PWM Fan Control Headers:
There are two level-shifted PWM outputs from 3.3V to 5V for standard PWM fan control, providing versatility in thermal management.



1.2 Limitations
The design is tailored for LED control and may not be optimal for other purposes.

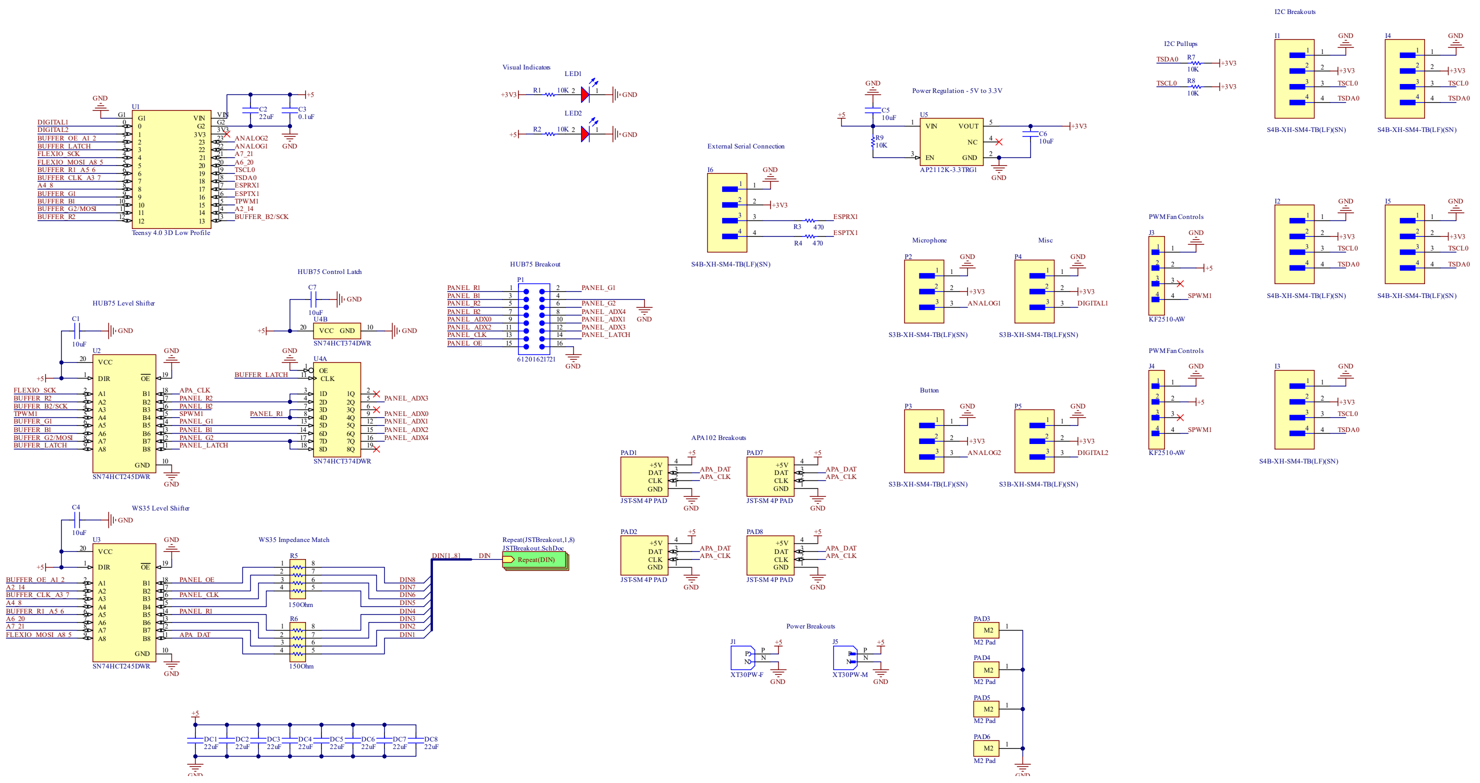
Power constraints of the AP2112K linear dropout regulator limit the number of directly connected I2C devices.

The onboard voltage regulation is designed for 5V input only; other voltages require external regulation.

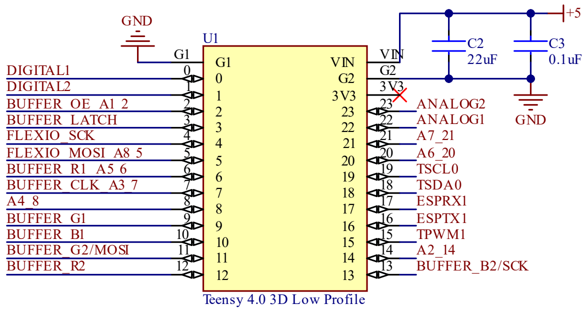
Thermal management through PWM fan headers is limited to devices compatible with the provided voltage levels and connectors.

2 Schematic

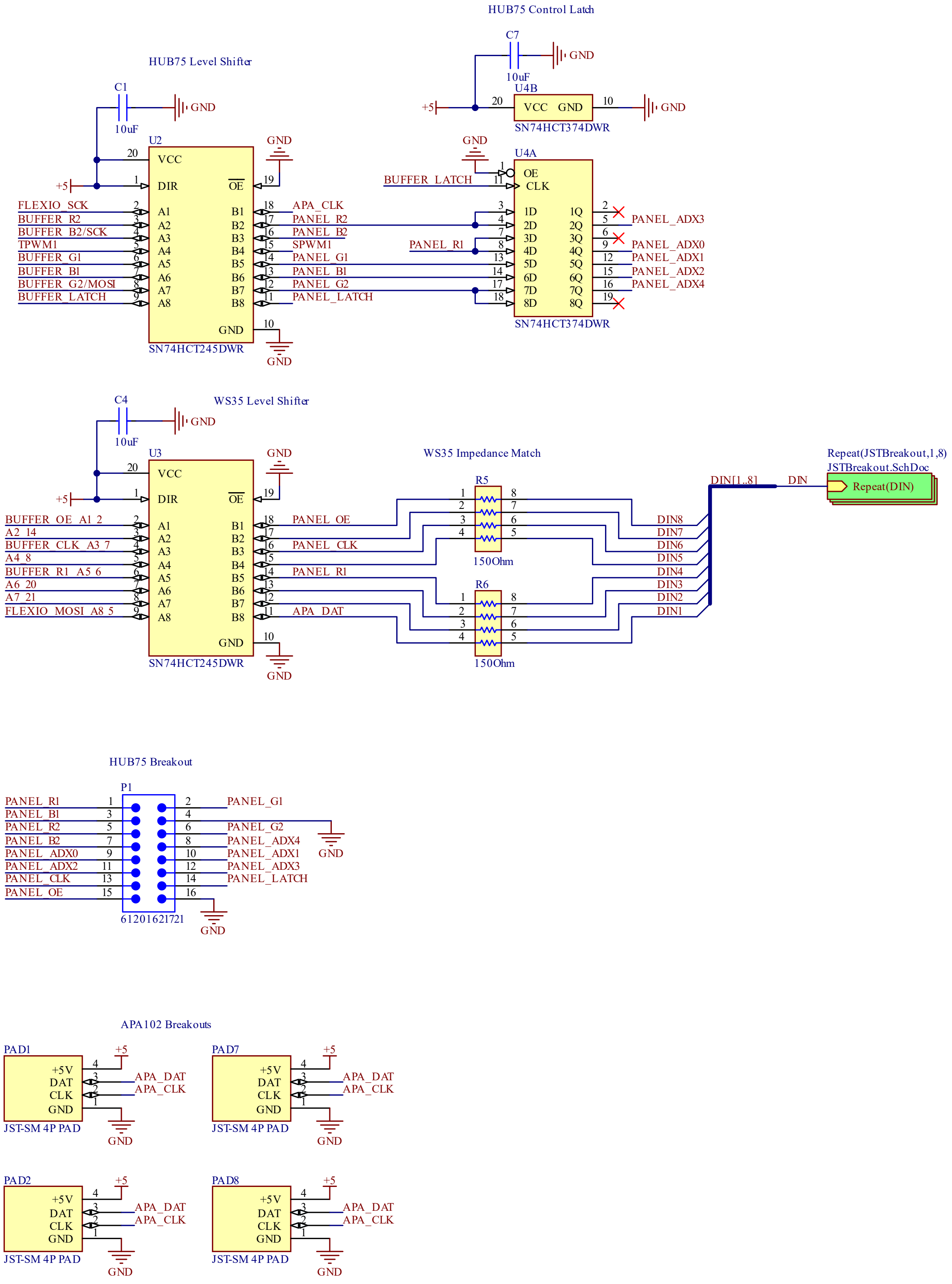
2.1 Full Schematic



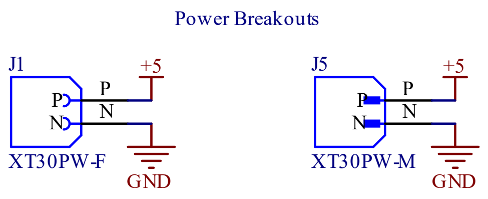
2.2 Teensy Pinout

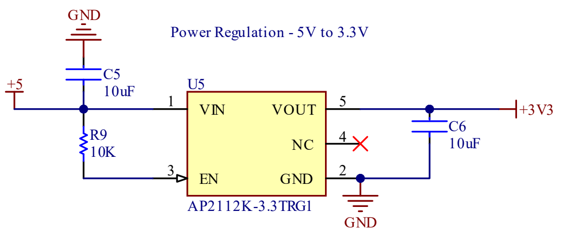


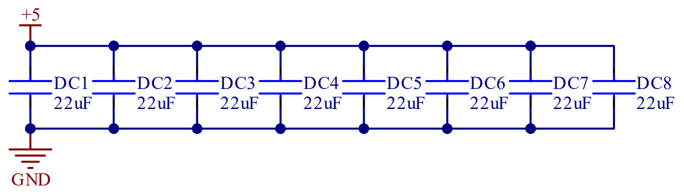
2.3 Logic Handling HUB75/APA102/WS2812

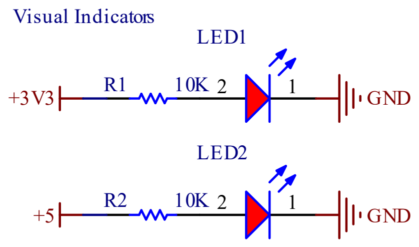


2.4 Power Handling and Voltage Regulation

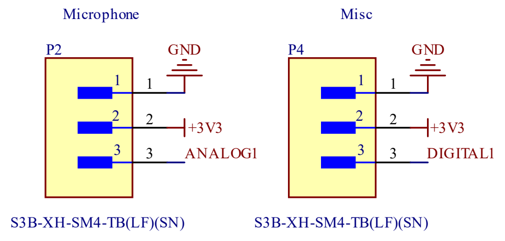


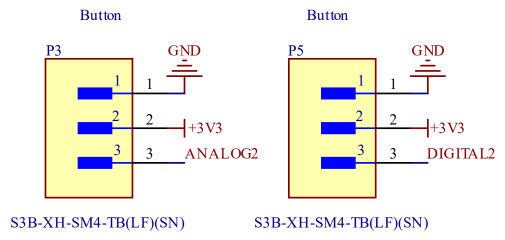




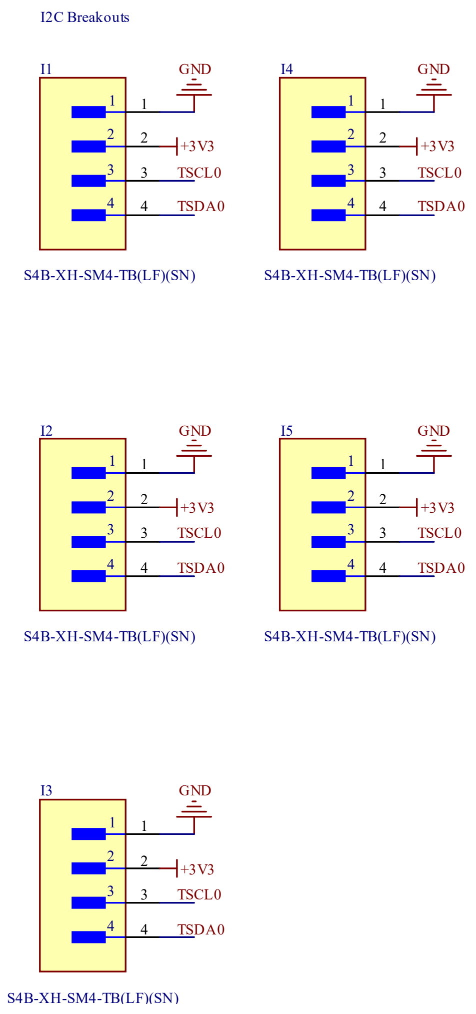


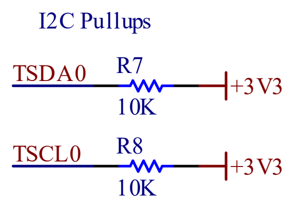
2.5 Sensors and Inputs

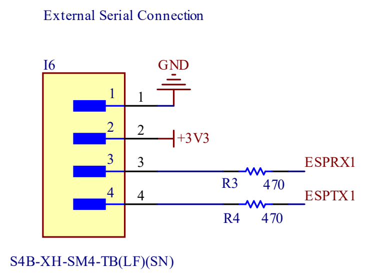




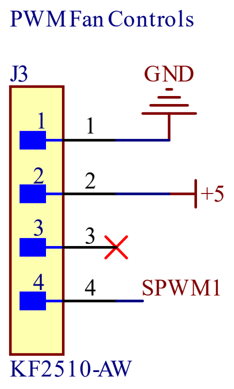
2.6 I2C and UART Communication

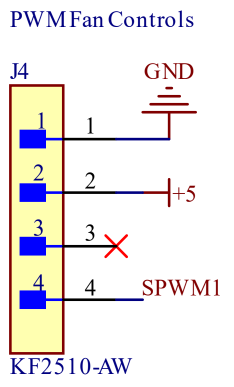




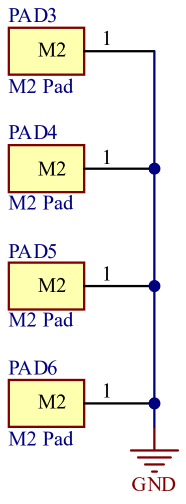


2.7 PWM Fan Control



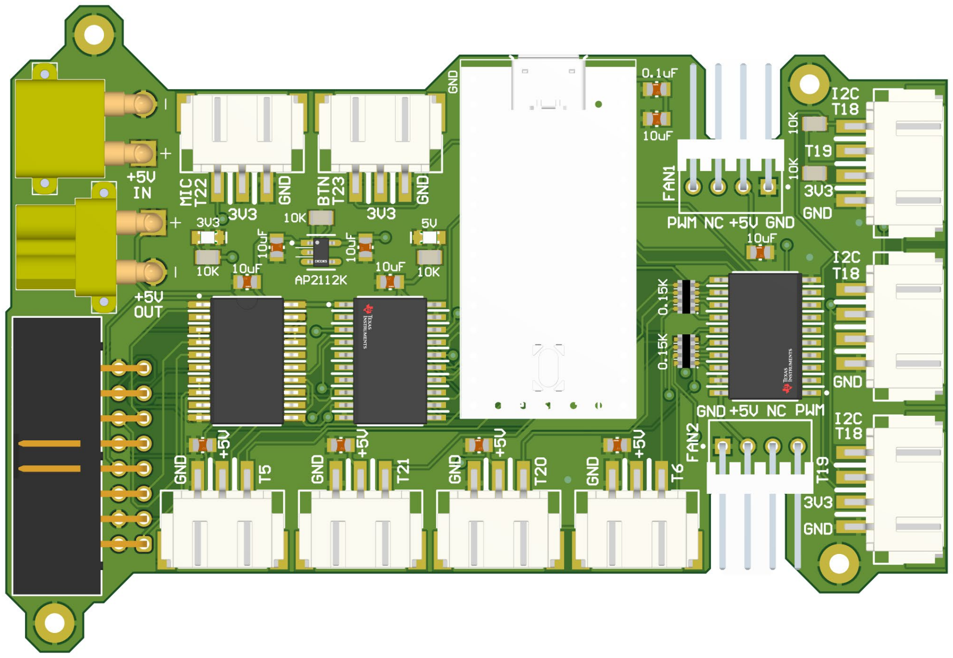


2.8 Mounting Points

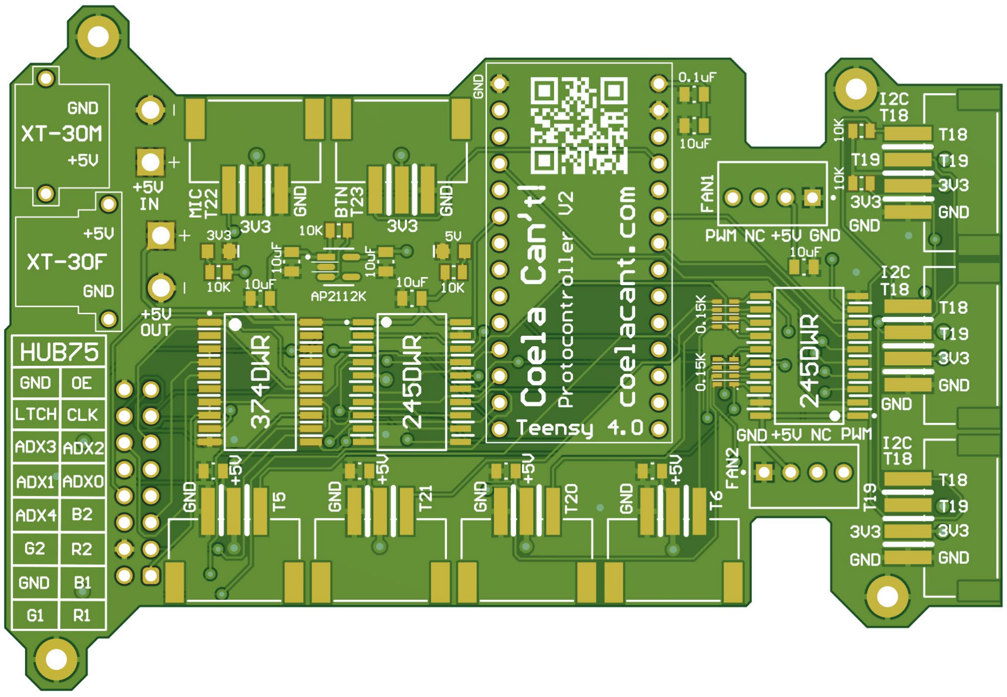


3 Board Layout

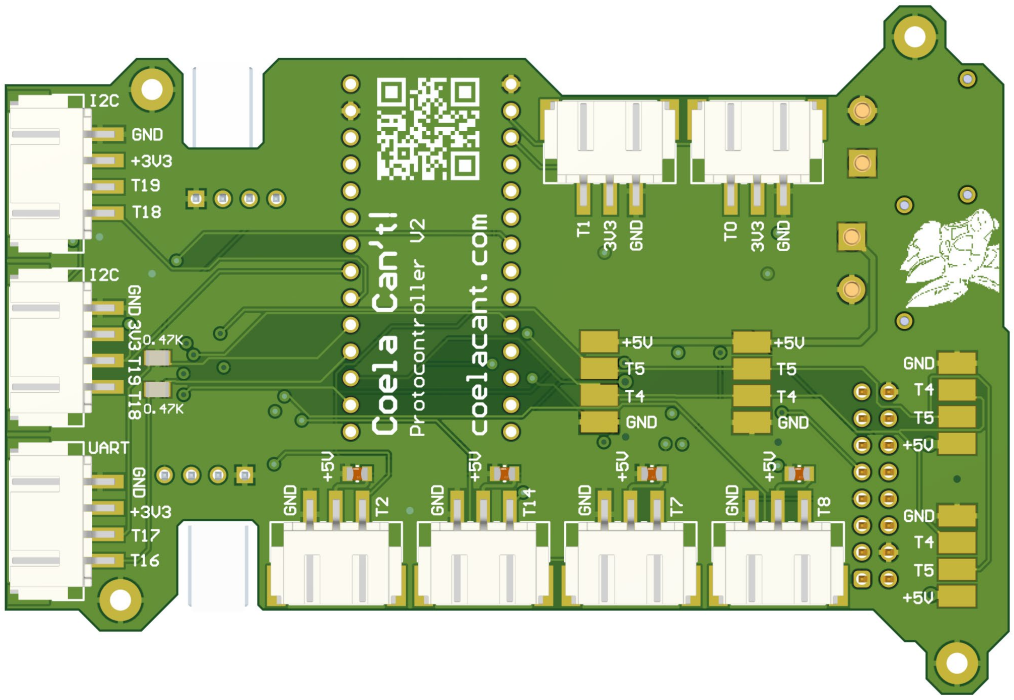
3.1 Top Layer 3D View with Bodies



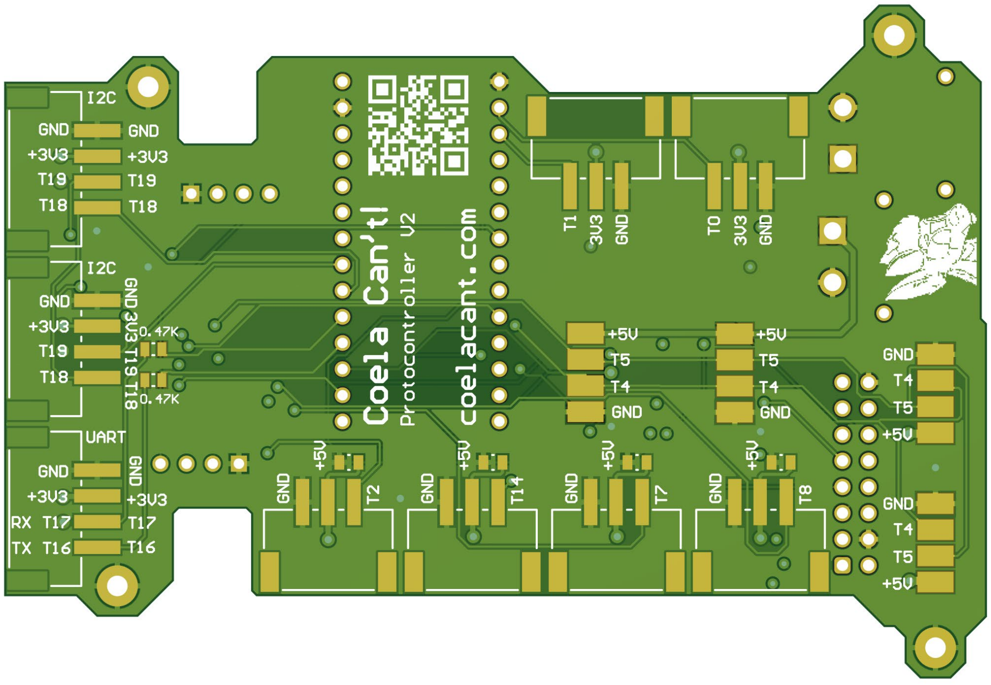
3.2 Top Layer 3D View without Bodies



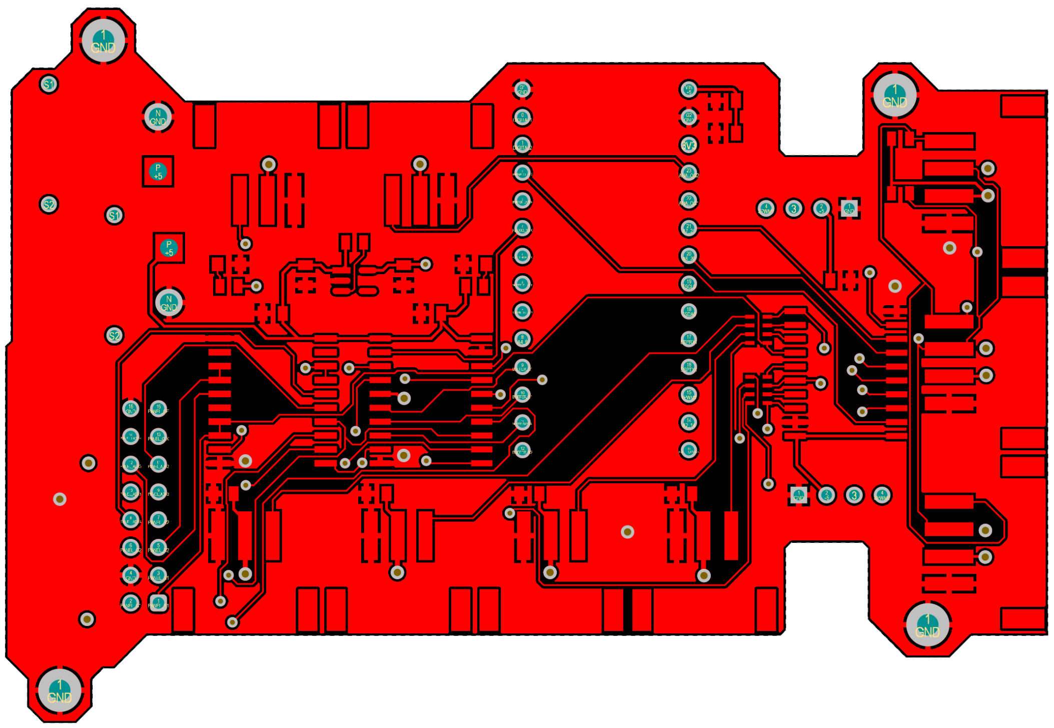
3.3 Bottom Layer 3D View with Bodies



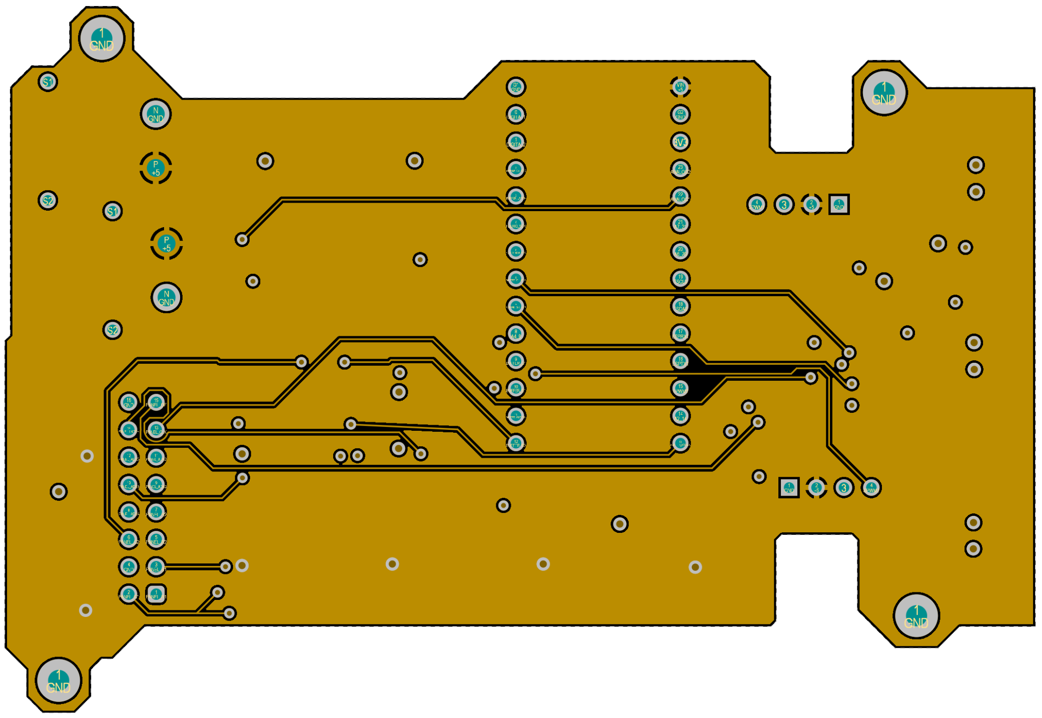
3.4 Bottom Layer 3D View without Bodies



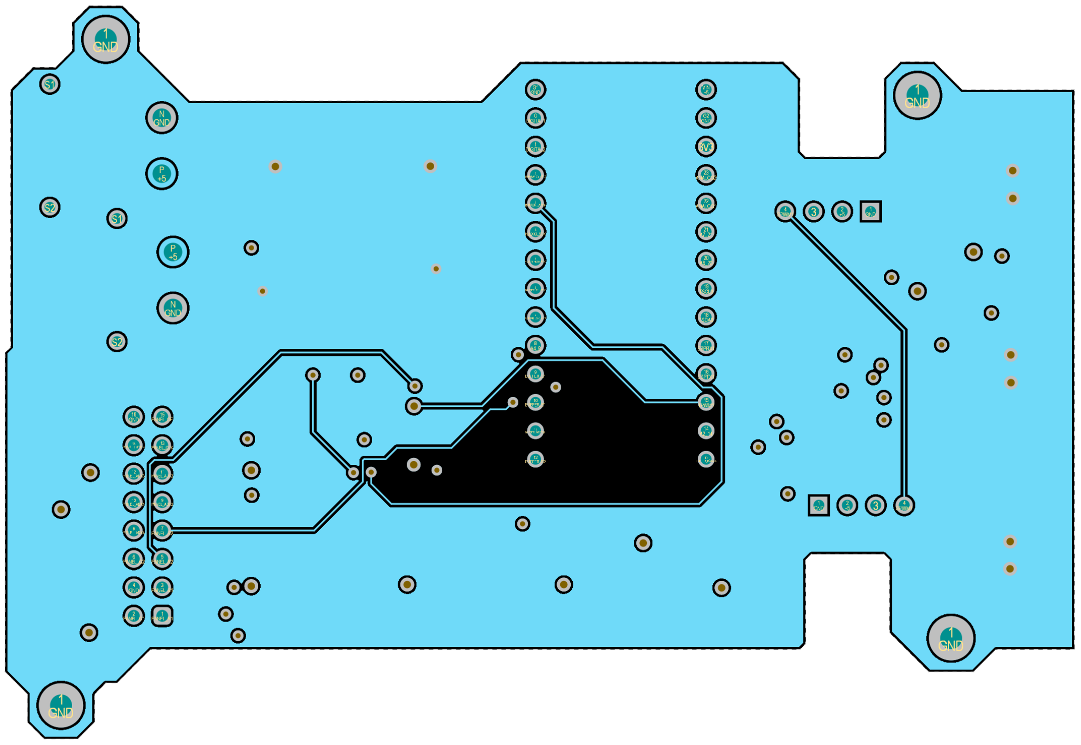
3.5 Top Copper Layer



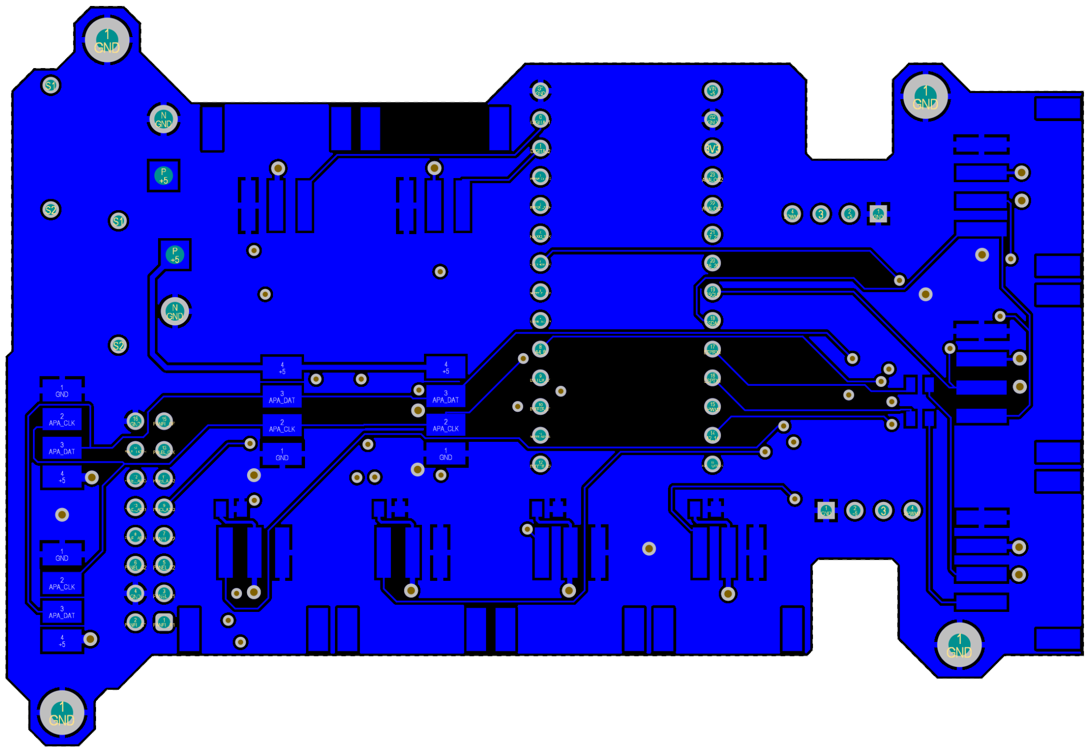
3.6 Middle Signal Copper Layer 1



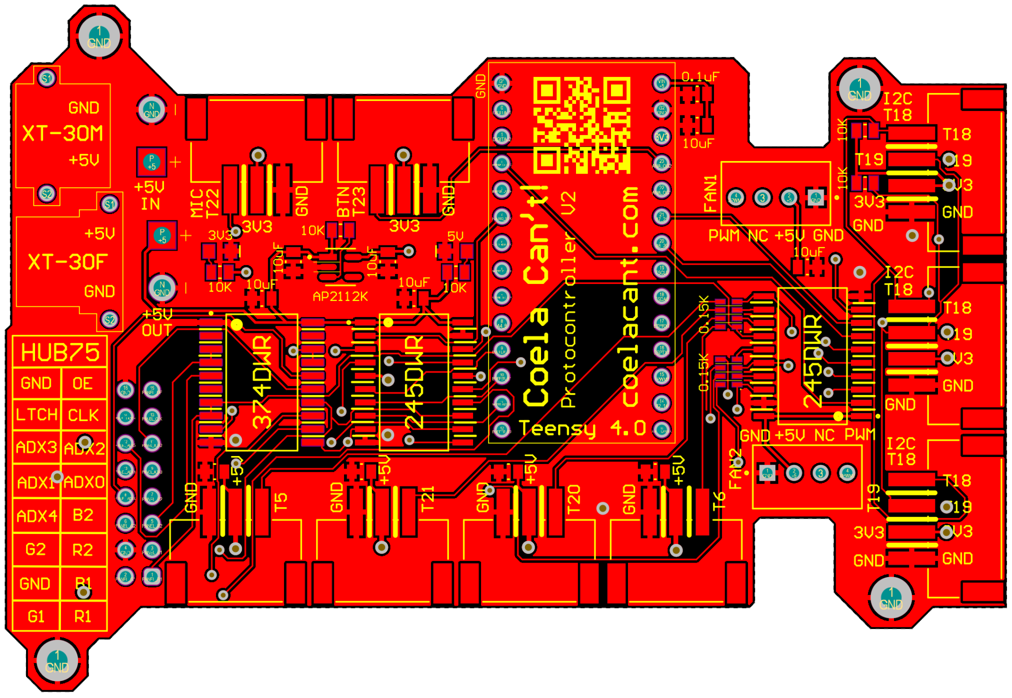
3.7 Middle Signal Copper Layer 2



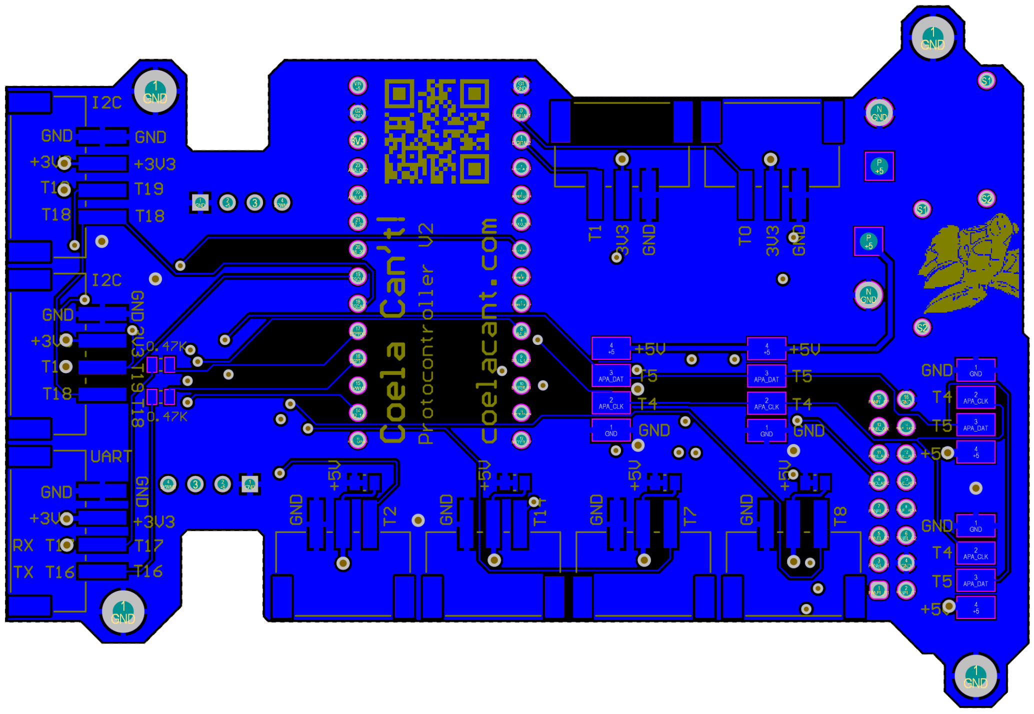
3.8 Bottom Copper Layer



3.9 Top Copper Layer with Top Overlay

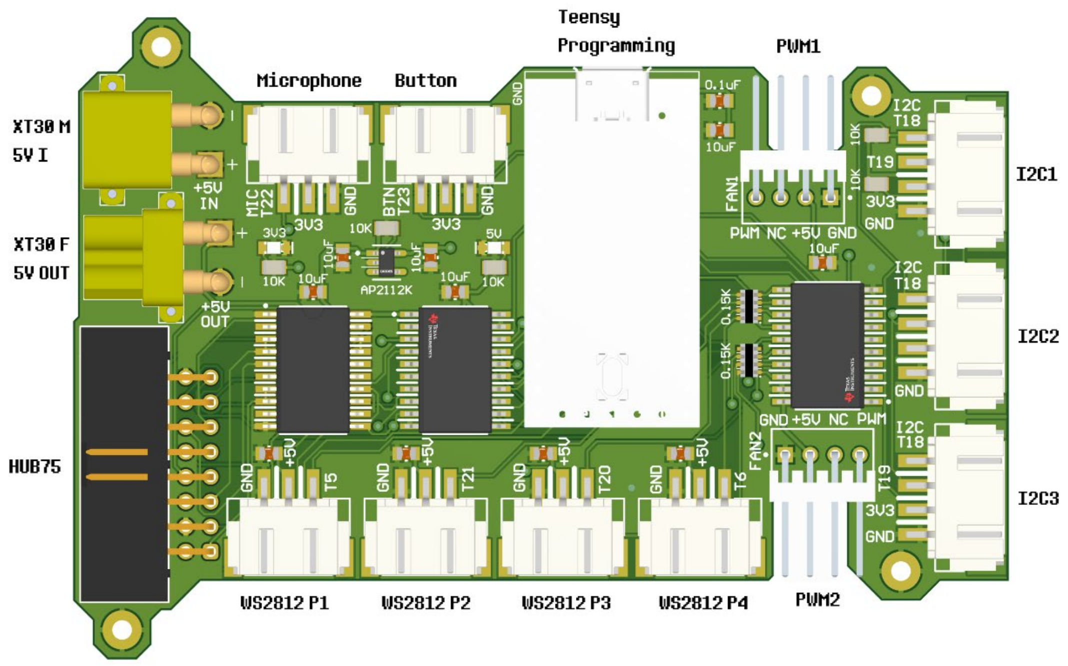


3.10 Bottom Copper Layer with Bottom Overlay

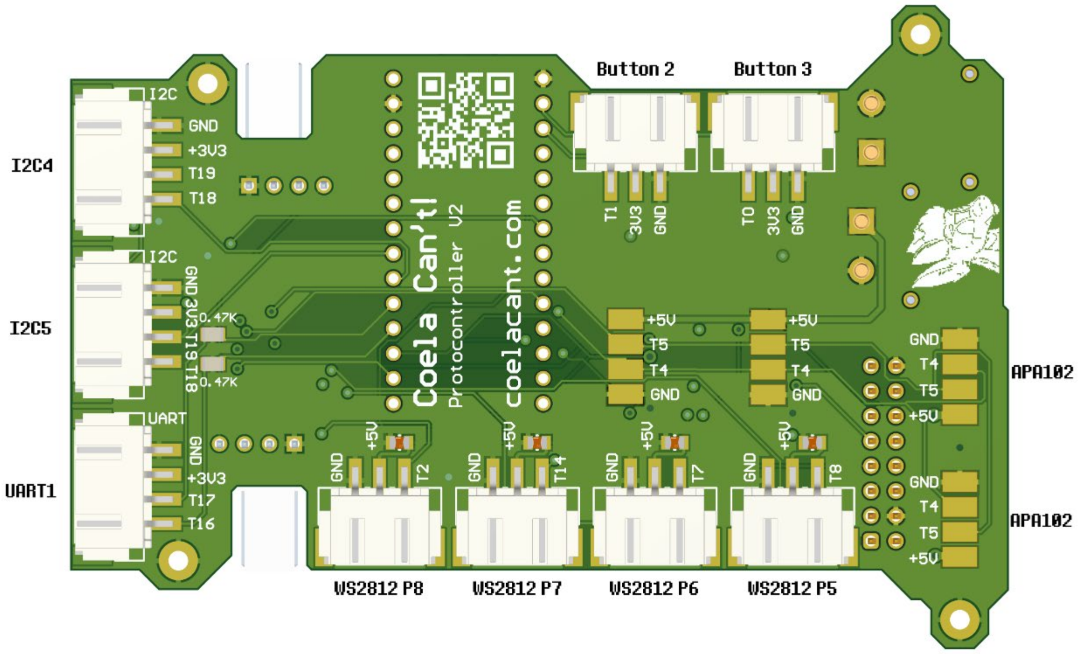


4 Wiring

4.1 Top Connector Pinout



4.2 Bottom Connector Pinout



5 Programming

5.1 Visual Studio Code and PlatformIO Setup

To download the ProtoTracer project from GitHub and set it up with PlatformIO for the Protocontroller V2 using the Teensy 4.0 microcontroller, you'll want to follow these steps:

5.1.1 Download ProtoTracer

Visit the ProtoTracer GitHub repository: github.com/coelacant1/ProtoTracer

Click the green "Code" button and select "Download ZIP", or use Git to clone the repository with the URL provided.

5.1.2 Install PlatformIO

Download and install Visual Studio Code. Open Visual Studio Code and install the PlatformIO extension from the marketplace.

5.1.3 Open the Project in PlatformIO

Extract the ProtoTracer project if you downloaded a ZIP. Open PlatformIO in Visual Studio Code, go to the "Home" tab, and click on "Open Project". Navigate to the extracted ProtoTracer project folder and select it.

5.1.4 PlatformIO Libraries and Dependencies

PlatformIO should automatically resolve and install necessary libraries and dependencies when you open the project. If not, you can manually install required libraries through the PlatformIO library manager.

5.1.5 Compile and Upload

Connect your Teensy 4.0 to your computer. Use PlatformIO's built-in commands to compile and upload the project to your Teensy 4.0.

For detailed usage and setup instructions, including how to import files, set up controllers, and manipulate objects, you can refer to the links provided in the repository's README file.

5.2 WS35 LED Panel Protogen

5.2.1 Main.cpp Changes

Uncomment the #define WS35 and comment any other Protogen option:

```
//#define ALPHARIGHT
//#define ALPHALEFT
//#define BETAWS35
//#define GAMMAFRONT
//#define GAMMABACK
//#define HUB75
//#define HUB75Split
//#define HUB75Square
#define WS35
//#define WS35SPLIT
//#define ESP32HUB75
//#define CUSTOMHUB75
//#define CUSTOMWS35
//#define CUSTOMBETAWS35
//#define HARDWARETEST
```

5.2.2 ProtogenWS35.h Breakdown

Once you modify the main.cpp, no other changes are necessary. Here is a commented breakdown of the Animation class used with this display type:

```
// Prevent multiple inclusions of this header file
#pragma once

// Include the necessary headers for the animation and object classes
#include "ProtogenAnimation.h"
#include "..\Objects\Background.h"
#include "..\Objects\LEDStripBackground.h"
#include "..\Morph\NukudeFlat.h"

// Define the ProtogenWS35 class, which is a type of ProtogenAnimation with 1 animated face
object
class ProtogenWS35 : public ProtogenAnimation<1> {
private:
    // Declare a NukudeFace object for managing facial morphs
    NukudeFace pM;

    // An array of strings representing different facial expressions for use on the HUD
    const __FlashStringHelper* faceArray[10] = {F("DEFAULT"), F("ANGRY"), F("DOUBT"), F("FROWN"),
F("LOOKUP"), F("SAD"), F("AUDIO1"), F("AUDIO2"), F("AUDIO3")};

    // Override the LinkControlParameters method to link UI controls with morph parameters
    void LinkControlParameters() override {
        // The following AddParameter calls link UI controls to specific facial morph weights
        // The parameters include the morph type, a reference to the morph weight, and the
control sensitivity
        // Some parameters use cosine interpolation for smooth transitions
        // ... (repeated for various facial morph parameters)

        // AddViseme calls link viseme shapes to morph weights for lip syncing
        // ... (repeated for various viseme types)

        // AddBlinkParameter links the blink morph weight to the control system
    }

    // Define methods for each facial expression that modify the morph parameters and, in some
cases, the material color
    // These methods are called based on the current mode of the animation
    // ... (methods for Default, Angry, Sad, Surprised, Doubt, Frown, LookUp, LookDown)

public:
    // Constructor for the ProtogenWS35 class
    ProtogenWS35() : ProtogenAnimation(Vector2D(), Vector2D(192.0f, 105.0f), 22, 23, 9){
        // Add the NukudeFace object to the scene
        scene.AddObject(pM.GetObject());

        // Set the material of the object to the face material
        pM.GetObject()->SetMaterial(GetFaceMaterial());

        // Link the control parameters to the UI
        LinkControlParameters();

        // Set up the HUD with the face expressions array
        hud.SetFaceArray(faceArray);
    }

    // Override the Update method which is called each frame to update the animation state
    void Update(float ratio) override {
        // Reset the morph weights to their initial state
        pM.Reset();

        // Get the current mode based on user input or other stimuli
        uint8_t mode = Menu::GetFaceState();//change by button press

        // Apply the Surprised expression if the model is booped and not in AUDIO1 mode
        if (IsBooped() && mode != 6){
            Surprised();
        }
        else{
            // Apply different expressions based on the mode
            // ... (conditional logic for setting the facial expression)

        }

        // Update the face based on the current animation ratio
        UpdateFace(ratio);

        // Set specific morph weights for the nose and eye movement
        pM.SetMorphWeight(NukudeFace::BiggerNose, 1.0f);
        pM.SetMorphWeight(NukudeFace::MoveEye, 1.0f);

        // Update the NukudeFace object
        pM.Update();

        // Align the object in the scene
        AlignObject(pM.GetObject(), -7.5f);

        // Set animation properties such as wiggle speed and menu appearance
        // ... (setting various animation and menu properties)

        // Update the object's transform with a calculated wiggle offset and apply the transform
        pM.GetObject()->GetTransform()->SetPosition(GetWiggleOffset());
        pM.GetObject()->UpdateTransform();
    }
};
```

5.3 HUB75 Protogen

5.3.1 Main.cpp Changes

Uncomment the #define HUB75 and comment any other Protogen option:

```
//#define ALPHARIGHT
//#define ALPHALEFT
//#define BETAWS35
//#define GAMMAFRONT
//#define GAMMABACK
#define HUB75
//#define HUB75Split
//#define HUB75Square
//#define WS35
//#define WS35SPLIT
//#define ESP32HUB75
//#define CUSTOMHUB75
//#define CUSTOMWS35
//#define CUSTOMBETAWS35
//#define HARDWARETEST
```

5.3.2 ProtogenHUB75.h Breakdown

```
// Ensure that the header file is included only once during the compilation
process
#pragma once

// Include dependencies from the ProtogenAnimation framework and other local
headers
#include "ProtogenAnimation.h"
#include "..\Objects\Background.h"
#include "..\Objects\DeltaDisplayBackground.h"
#include "..\Morph\NukudeFlat.h"

// Declare the ProtogenHUB75 class, inheriting from ProtogenAnimation with 2
objects
class ProtogenHUB75 : public ProtogenAnimation<2> {
private:
    NukudeFace pM;  // Private instance of NukudeFace to manage facial
animations
    DeltaDisplayBackground deltaDisplayBackground;  // Instance to manage the
background display

    // Array of flash-stored strings representing facial expression names
    const __FlashStringHelper* faceArray[10] = {F("DEFAULT"), F("ANGRY"),
F("DOUBT"), F("FROWN"), F("LOOKUP"), F("SAD"), F("AUDIO1"), F("AUDIO2"),
F("AUDIO3")};

    // Override of a virtual method to link control parameters to facial
morphs and expressions
    void LinkControlParameters() override {
        // The following AddParameter calls bind UI controls to facial morphs
with specific configurations
        // ... (code that binds parameters to controls)
        // The following AddViseme calls bind mouth shapes to sound visemes
for lip-syncing
        // ... (code that binds visemes to controls)

        // A special parameter for blinking
    }

    // Methods corresponding to facial expressions that can be triggered
    // Each method sets the appropriate morph parameters and material colors
    // ... (methods for Default, Angry, Sad, Surprised, Doubt, Frown, LookUp,
LookDown)

public:
    // Constructor for ProtogenHUB75, initializing the base ProtogenAnimation
with specific parameters
    ProtogenHUB75() : ProtogenAnimation(Vector2D(), Vector2D(192.0f, 96.0f),
22, 23, 9){
        // Add the NukudeFace and DeltaDisplayBackground objects to the scene
        scene.AddObject(pM.GetObject());
        scene.AddObject(deltaDisplayBackground.GetObject());

        // Set materials for the objects
        pM.GetObject()->SetMaterial(GetFaceMaterial());
        deltaDisplayBackground.GetObject()->SetMaterial(GetFaceMaterial());

        // Link UI controls to animation parameters
        LinkControlParameters();

        // Initialize the HUD with the array of face expressions
        hud.SetFaceArray(faceArray);
    }

    // Override the Update method to handle animation updates
    void Update(float ratio) override {
        // Reset the NukudeFace to its base state
        pM.Reset();

        // Retrieve the current mode based on user interaction
        uint8_t mode = Menu::GetFaceState();//change by button press

        // Apply expressions based on the mode, with special behavior for
boops
        // ... (conditional logic for expression modes)

        // General update procedures for the face and object alignment
        UpdateFace(ratio);
        pM.Update();
        AlignObject(pM.GetObject(), -7.5f);

        // Set various animation and menu properties
        // ... (setting wiggle speeds and menu properties)

        // Update the object's position and apply transformations
        pM.GetObject()->GetTransform()->SetPosition(GetWiggleOffset());
        pM.GetObject()->UpdateTransform();
    }
};
```

5.4 Uploading ProtoTracer

5.4.1 Connect the Teensy 4.0 to Your Computer

Since the Vin line is cut, ensure that the Teensy is powered through its external power supply.

5.4.2 Open Your Project in PlatformIO

Launch Visual Studio Code with PlatformIO installed.

Open your project that you want to upload to the Teensy.

5.4.3 Build the Project

Use the PlatformIO: Build command from the PlatformIO IDE's toolbar or the sidebar to compile your code.

5.4.4 Upload the Firmware

Click on the PlatformIO: Upload button, which should also be on the toolbar or sidebar. This will start the upload process.

If the Teensy Loader does not start automatically or the upload doesn't begin, you may need to manually reset the Teensy by pressing the pushbutton on the board.

5.4.5 Monitor the Upload Process

PlatformIO will indicate the progress of the upload in the terminal at the bottom of the Visual Studio Code window.

Wait for the process to complete successfully before disconnecting the Teensy or attempting to upload again.

6 General Recommendations and Notes

Please ensure that the XT30 connector is supplied with 5V directly; the controller is designed to operate within a voltage range of 4.5V to 5.5V, and any voltage outside this range is beyond the specified limits.

Avoid short-circuiting the I/O pins and refer to the guide for active pin designations.

When connecting external power to the controller via the XT30 connector, ensure the supply is strictly 5V, within the operational range of 4.5V to 5.5V. Any voltage outside this range falls outside the device's specifications.

Avoid short-circuiting the I/O pins by referring to the guide for detailed pin usage.

When making or modifying connections, always verify the polarity to prevent damage, even though the controller has reverse voltage protection.

The controller is designed for operation in temperatures ranging from 0°C to +40°C; however, for reliable functioning, maintain environmental temperatures between 5°C and 30°C.

Exercise caution with humidity, as it can damage the electronics.

If you encounter any issues with the controller, contact support before attempting repairs to avoid further damage.

In the event of water exposure, promptly disconnect power. This controller is not rated for high humidity environments exceeding 90% and is not waterproof—any exposure to water can cause irreversible damage.