Coela Can’t!

HUB75 Protogen Kit:

Getting Started

V1.0.2022 – Coela Can’t!

Contents

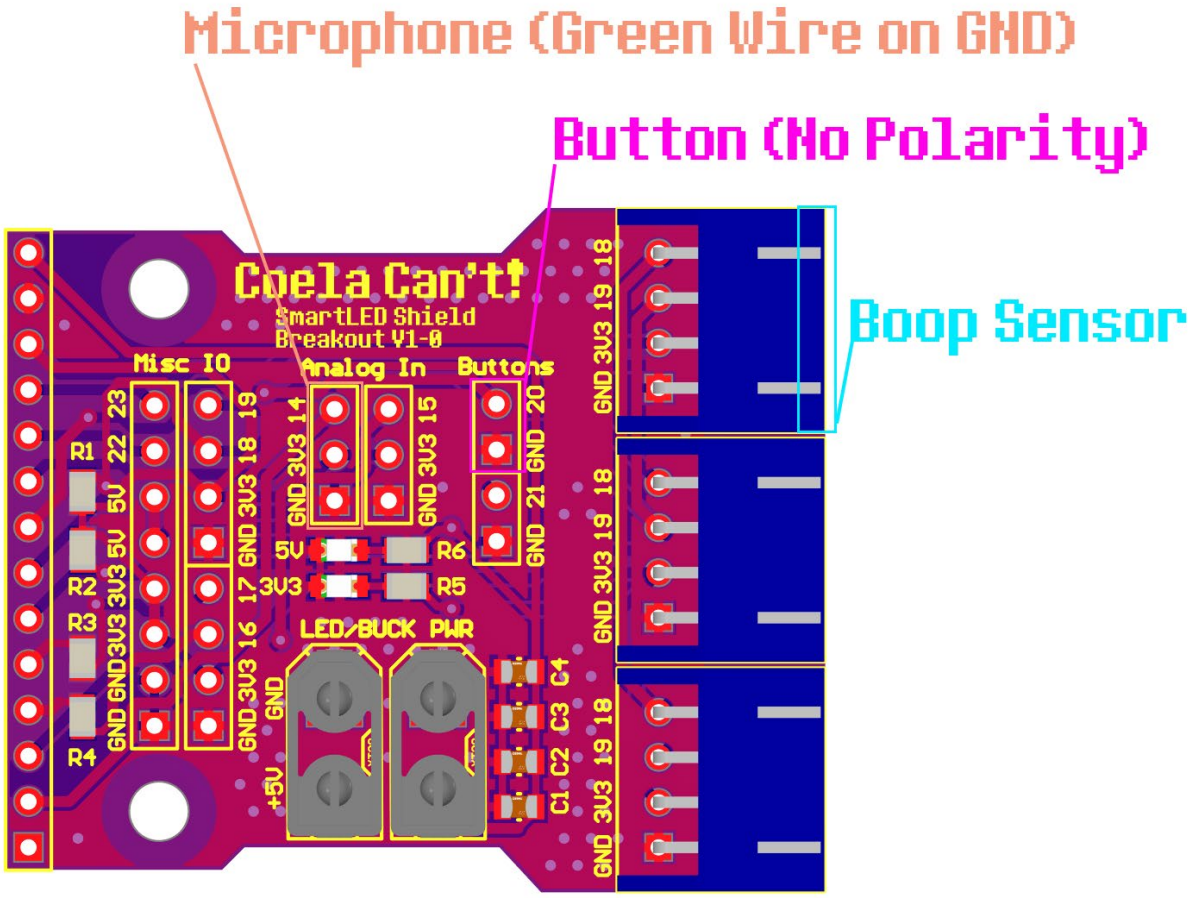
1 Wiring Your Protogen 4
1.1 Sensor Wiring Diagram 4
1.2 Wiring Pictures 5
1.2.1 Connecting the Controller 5
1.2.2 Wiring the USB Power Source 6
1.2.3 Wiring the Boop Sensor 6
1.2.4 Wiring the Microphone 7
1.2.5 Wiring the Control Button 7
1.2.6 Final Wiring 8
2 HUB75 LED Panels 9
2.1 General Information 9
2 SmartLED Shield and Breakout 9
3.1 General Information 9
3.1.1 Easy-to-implement LED Graphics 9
3.1.2 Features 10
3.1.3 HUB75 Panels 11
3.1.4 Teensy 11
3.1.5 Teensy Pin Usage 12
3.1.6 Documentation 13
3.2 Controller Breakout 13
3.2.1 Information 13
3.2.2 Schematic 13
3.2.3 Full PCB Design 14
3.2.4 Top Layer PCB Design 14
3.2.5 Bottom Layer PCB Design 15
3.2.6 Top Overlay PCB Design 15
3.4 Controller Breakout Schematic Breakdown 16
3.4.1 16 Pin Header 16
3.4.5 Teensy I2C Headers 17
3.4.7 XT30 Power Input and Output 17
3.4.8 Analog Headers 18
3.4.9 Digital Headers 18
3.5 Indicators 18
3.6 Power Setup 19
3.6.1 Powering the Teensy 4.0 for Programming 19
3.6.2 Powering the Controller for Usage 19
4 Sensors/Peripherals 20
4.1 Boop Sensor (APDS-9960) 20
4.2 MAX9814 Microphone 20
4.3 Control Button 20
4.4 MPU6050 20
4.5 I2C OLED Display 20

5 Programming the Teensy 21
5.1 Getting Started with ProtoTracer 21
5.1.1 Install Applications 21
5.1.2 Install Extensions 21
5.1.3 Download the Codebase 21
5.2 Loading a Controller 22
5.3 Loading or Modifying an Animation 23
5.4 Controller Example 24
5.5 Animation Example 27
6 General Recommendations and Notes 28

1 Wiring Your Protogen

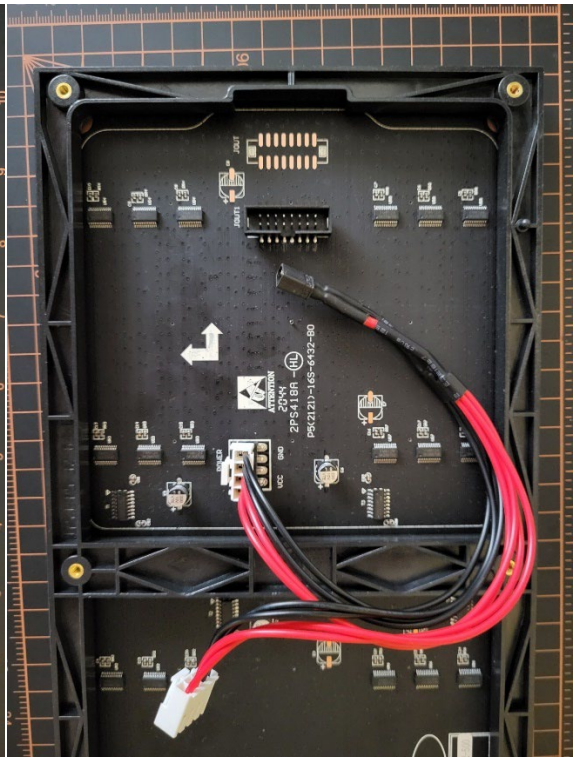
1.1 Sensor Wiring Diagram

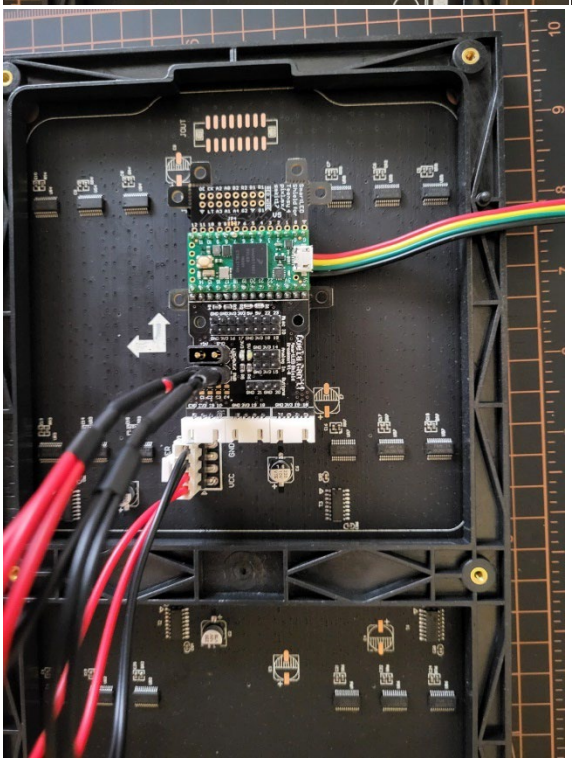
The following diagram details the connections for the MAX9814 microphone, the APDS-9960 boop sensor, and the face control button. The power does not matter but the XT30 connections can only be inserted in one way. Power from the USB Buck Converter to the breakout board, from the breakout board to the LED Panels. Do not bypass the breakout board or it will not provide power to the Teensy.



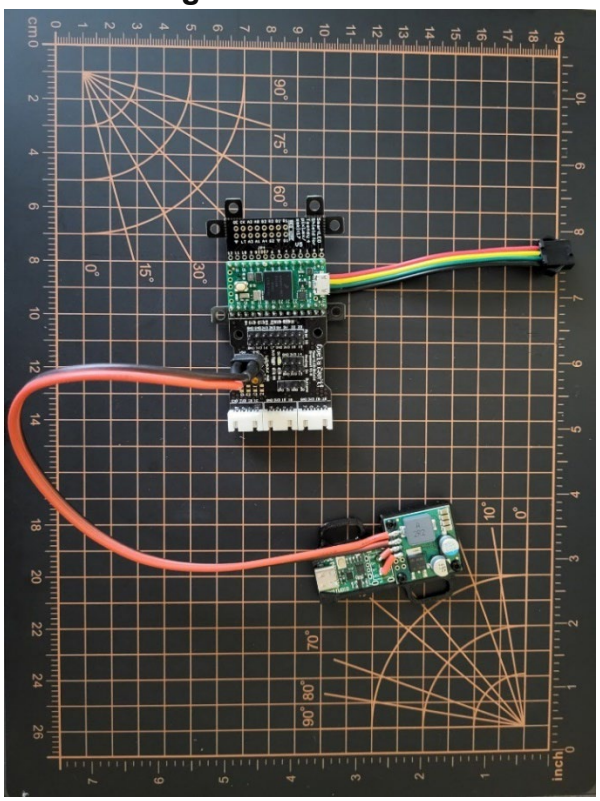
1.2 Wiring Pictures

1.2.1 Connecting the Controller

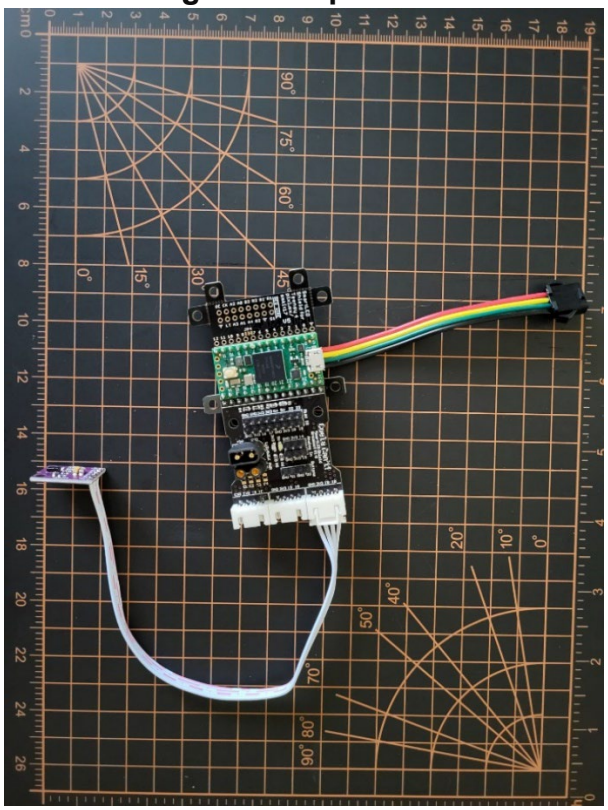




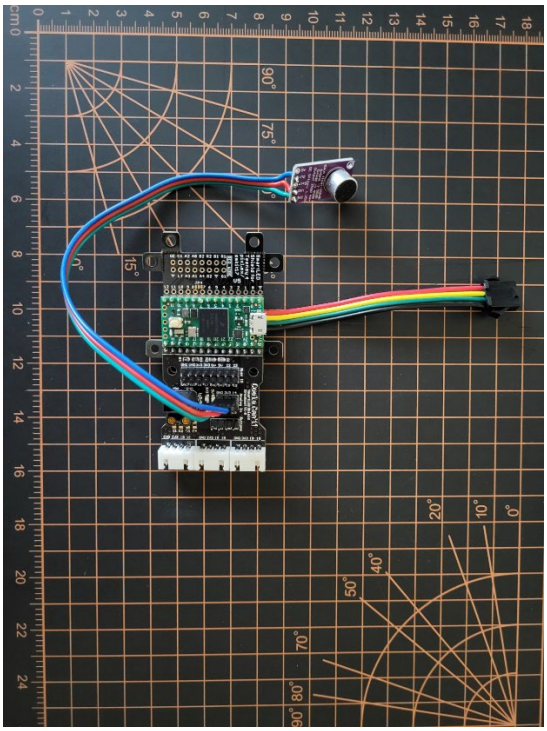
1.2.2 Wiring the USB Power Source

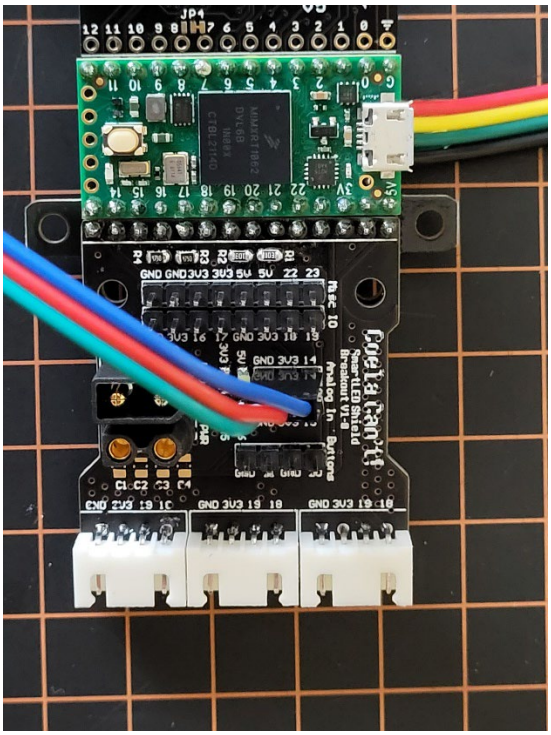


1.2.3 Wiring the Boop Sensor

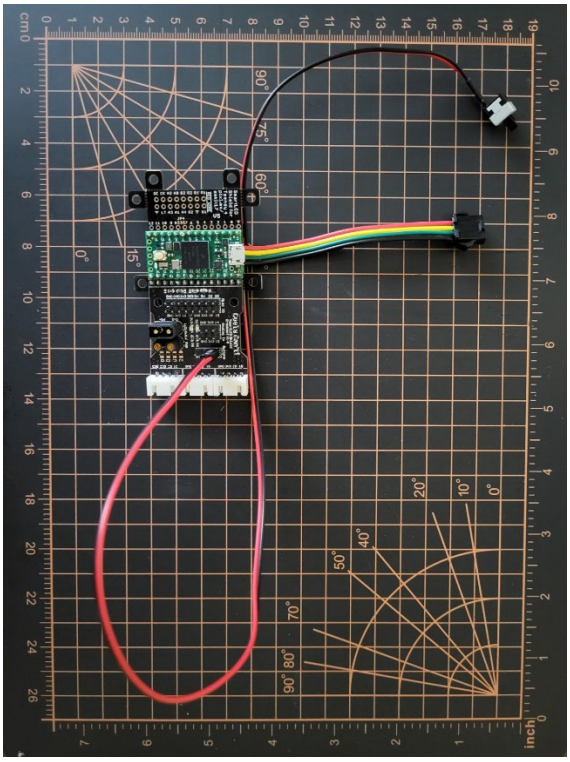


1.2.4 Wiring the Microphone

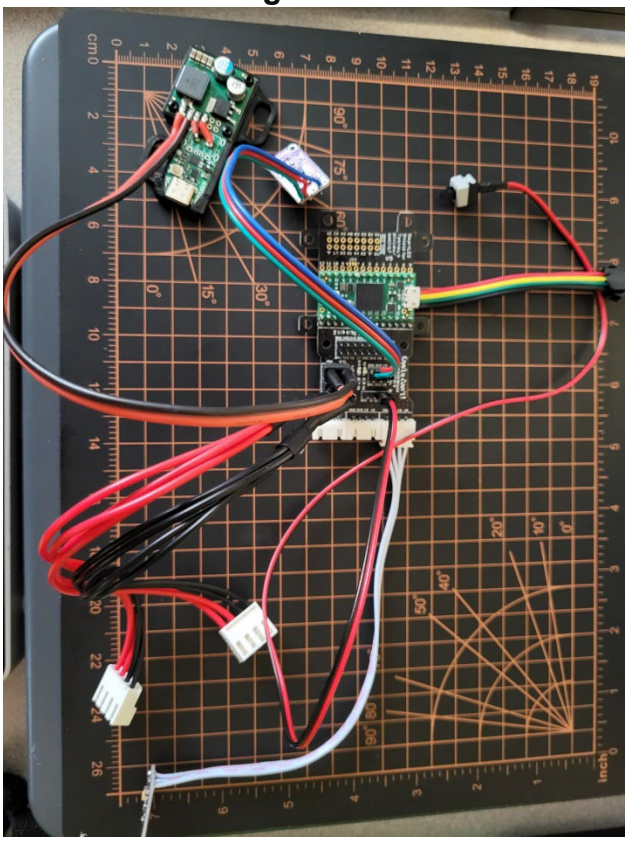




1.2.5 Wiring the Control Button



1.2.6 Final Wiring



2 HUB75 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 SmartLED Shield and Breakout

3.1 General Information

The SmartLED Shield for the Teensy 4 is used as the backbone to this design, this allows you to use open-source code to be easy to implement if you decide against using ProtoTracer to display graphics on your Protogen!

The following information is pulled directly from their Crowd Supply page as a means of maintaining a local copy incase they take the posting down:
https://www.crowdsupply.com/pixelmatix/smartled-shield-for-teensy-4

3.1.1 Easy-to-implement LED Graphics

SmartLED Shield enables the Teensy 4 to drive high-quality graphics to HUB75 RGB LED panels, with 36-bit color and 240 Hz refresh rate across large panels (e.g. 128x64 pixels). A Teensy 4.0 or Teensy 4.1 with pins fits into the socket on the shield, and the shield can attach directly to the HUB75 panel or through a ribbon cable. The SmartMatrix library for Arduino makes it easy to draw basic graphics, create scrolling and static text, draw beautiful patterns using FastLED, and play animated GIFs on the panel. Example code is provided so you can get started as quickly as possible. The Shield and library use special features and peripherals of the Teensy 4 processor to send graphics data to your display with minimal CPU usage, so you can use the processor to do other tasks in parallel such as SPI communication, file decoding, or complex rendering.

Using SmartLED Shield with SmartMatrix library and the Teensy 4 is the easiest way to drive high-quality and high pixel count graphics to RGB LED panels with a microcontroller. Use a simple API to tell the library what to draw on the screen, and the library takes care of refreshing in the background. Advanced features like these are enabled automatically:

- 36-bit Color Refresh - See the full color range in the image or pattern you're displaying, with no noticeable brightness steps when dimming pixels down to black. Up to 48-bit color refresh is available.
- Color (Gamma) Correction - Your source graphics are probably 24-bit color, but SmartMatrix library applies automatic color correction so they have good contrast, smooth gradients, and don't look washed out.
- Global Brightness Control - When you don't need the full brightness of the LED panel, lower the brightness without having to sacrifice color depth of your graphics.

The shield is easy to assemble and connect to a panel, and there’s no soldering required beyond adding pins to the Teensy. The Teensy is removable, so you can swap between the 4.0 and 4.1 if you want. All long edge Teensy signals are brought out to expansion rows for easy prototyping.

3.1.2 Features

- Ease-of-use:
    - SmartLED Shield is fully assembled. If your Teensy has pins, then no soldering is required.
    - Teensy can easily be inserted into and removed from the shield.
    - SmartMatrix library for Arduino provides an easy development platform, along with others such as FastLED.
    - Example code is included for a quick start.
- Flexibility:
    - Drives displays with up 9k pixels (e.g. 96 x 96) with high quality settings, and even larger displays with reduced quality settings.
    - Signals on the long edges of the Teensy are brought out to expansion rows for easy prototyping.
    - The 4-pin JST-SM connector may be used to provide power to the Teensy separate from the USB connector.
    - Optionally drive DotStar/APA102-compatible LEDs using the onboard 5 volt buffers and 4-pin JST-SM connector. Mating JST-SM cable is included.
    - Panels can be daisy-chained to make large, bright, high-resolution displays.
    - Drives all 14 signals on HUB75 panels using 5 volt buffered outputs, using only 9 GPIO pins on the Teensy 4.0 or 4.1.
- Quality:
    - Provides up to a 240 Hz refresh rate.
    - Up to 48-bit color refresh is available.
    - Color (gamma) correction, and global brightness control features allow for a high level of visual quality control.

3.1.3 HUB75 Panels

HUB75 RGB panels are typically used for LED billboards (e.g., Times Square), making them cost-effective and readily available. They're much cheaper per-pixel than addressable LEDs, and available in a wide range of pixel pitch (as of now, 2 mm spacing up to 10 mm spacing per LED). They do require an external controller to continually send data to the panels to refresh them line by line, and that's where the SmartLED Shield and SmartMatrix library come in. Adafruit, Sparkfun, and other distributors carry panels that are known to be compatible with SmartLED Shield and the SmartMatrix library, but most panels on AliExpress and other sources are compatible as well.

Note: HUB75 panels have a separate power connector and require an appropriate 5-volt power supply. The SmartLED Shield does not provide power to the panel, only data signals.

The pixel pitch and "RGB" are good search terms on Aliexpress, e.g. "P6 RGB" for a 6 mm pitch RGB HUB75 panel.

For the controller you should utilize boards with the 6124 chip to ensure functionality. This is the only panel that I have tested this with, others should work but will not be supported in software by default.

3.1.4 Teensy

Teensys are small, well-featured, low-cost microcontroller boards designed by PJRC. Best of all, the boards have excellent development support and a large community. They're the perfect solution to build an LED board around.

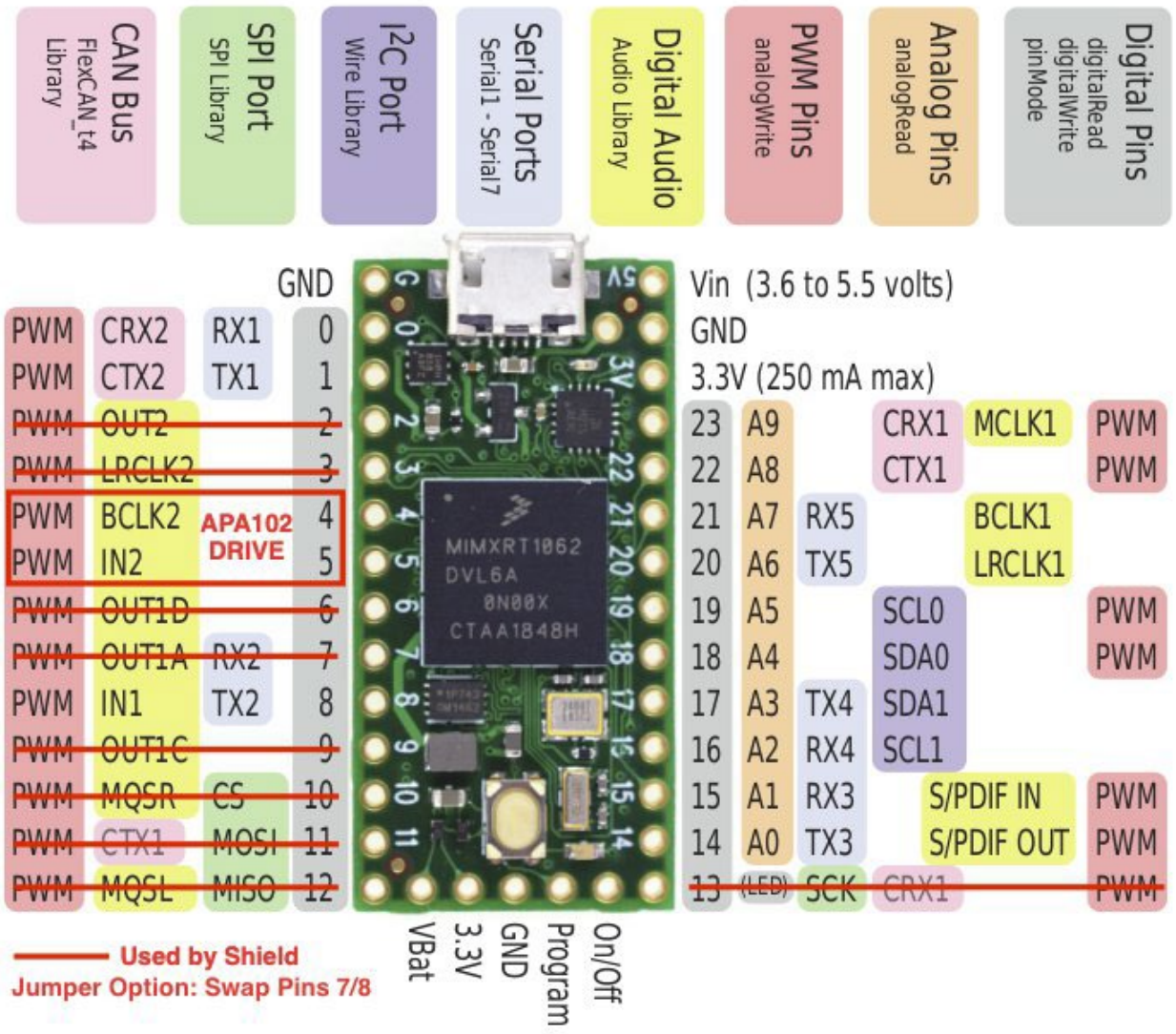
SmartLED Shield works with two variants of Teensy 4, the 4.0 and 4.1, with the 4.1 being the higher-powered option. Teensy units are available for purchase along with shields during the campaign. Please note that SmartLED Shield kits come with headers required to mate your Teensy to the board, but if you buy a Teensy through the campaign, you will need to solder the Teensy headers yourself. Units with pre-soldered headers can be found on Sparkfun.

As stated on the PJRC website, these are the main differences between the two units:

Feature	Teensy 4.1	Teensy 4.0
Ethernet	10 / 100 Mbit DP83825 PHY (6 pins)	-none-
USB Host	5 Pins with power management	2 SMT Pads
SDIO (4-bit data)	Micro SD Socket	8 SMT Pads
PWM Pins	35	31
Analog Inputs	18	14
Serial Ports	8	7
Flash Memory	8 Mbyte	2 Mbyte

QSPI Memory	2 chips Plus Program Memory	Program memory only
Breadboard Friendly I/O	42	24
Bottom SMT Pad Signals	7	16
SD Card Signals	6	0
Total I/O Pins	55	40

3.1.5 Teensy Pin Usage



The additional pins on the right – 14 to VIN - are used to provide the additional functionality to the controller breakout.

3.1.6 Documentation

SmartMatrix library 4 GitHub: https://github.com/pixelmatix/SmartMatrix/

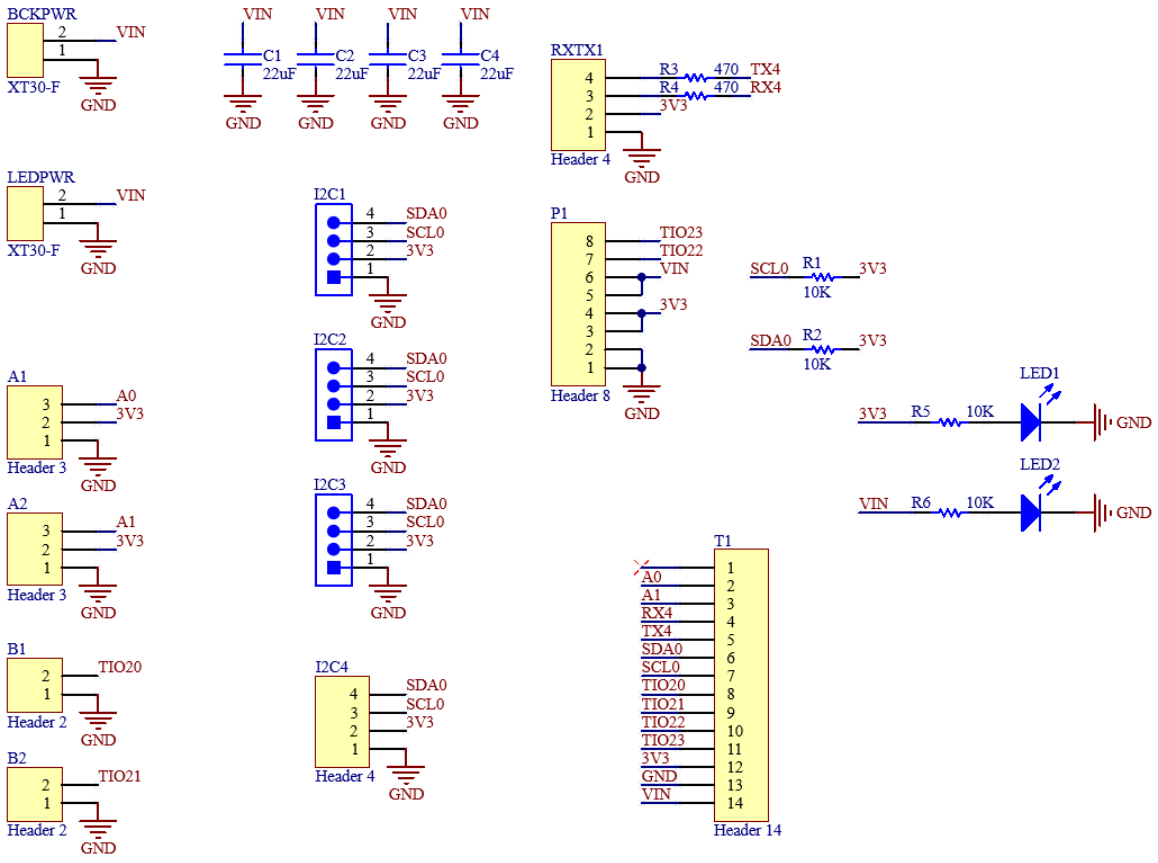
SmartLED Shield for Teensy 4 Documentation:
http://docs.pixelmatix.com/SmartMatrix/shield-t4.html

3.2 Controller Breakout

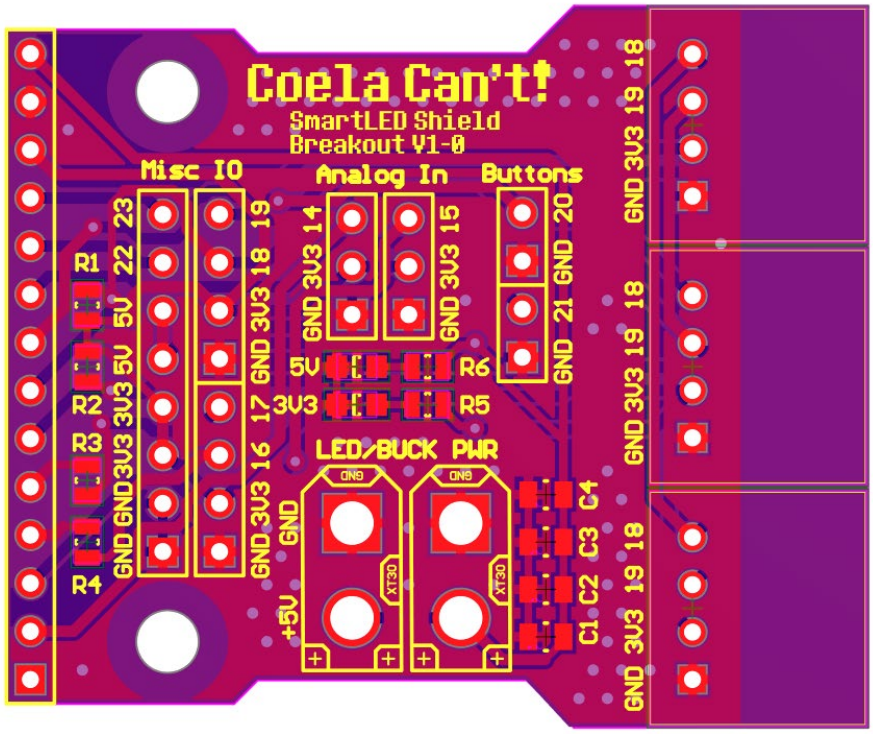
3.2.1 Information

The controller breakout provides sensor connectivity, power distribution, and indication methods for power.

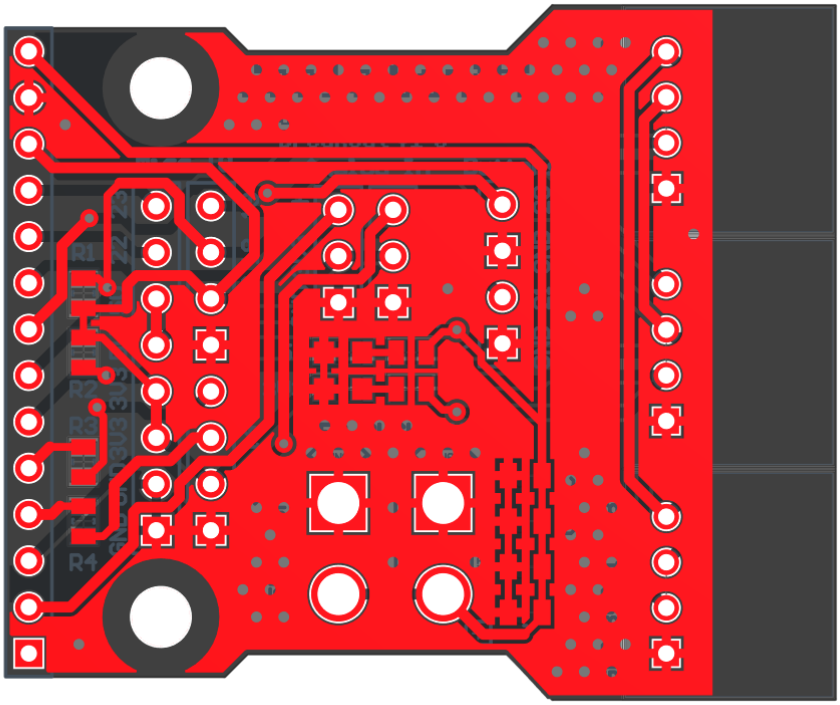
3.2.2 Schematic



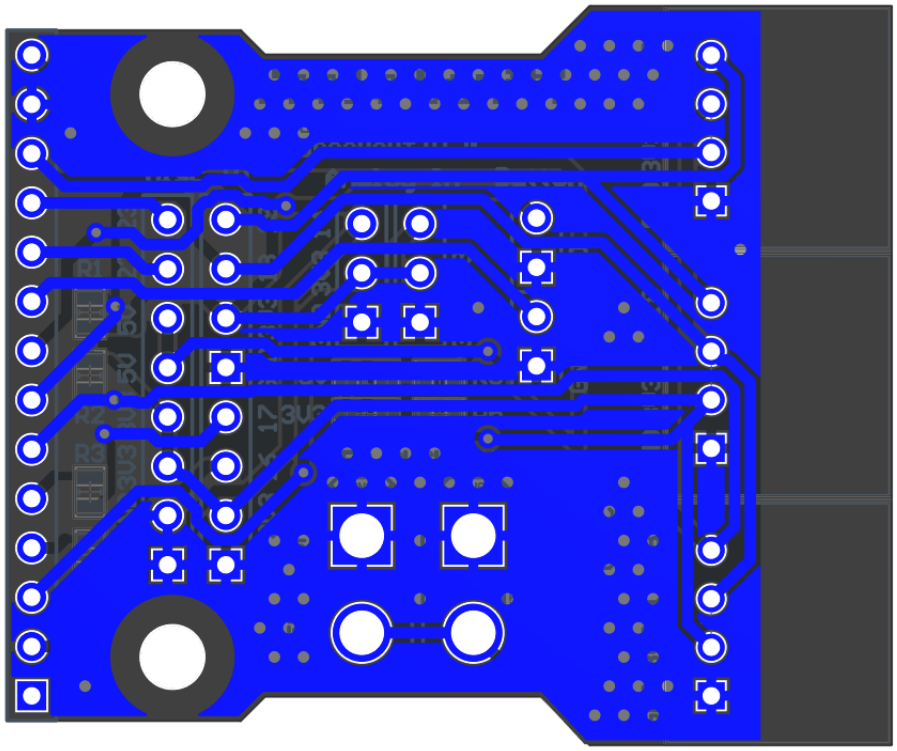
3.2.3 Full PCB Design



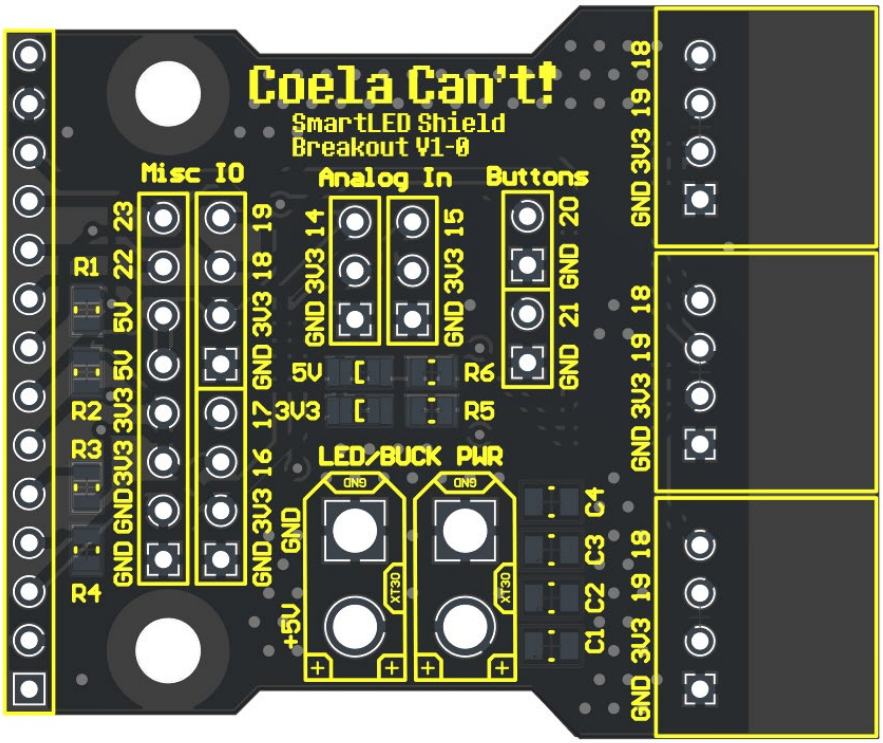
3.2.4 Top Layer PCB Design



3.2.5 Bottom Layer PCB Design



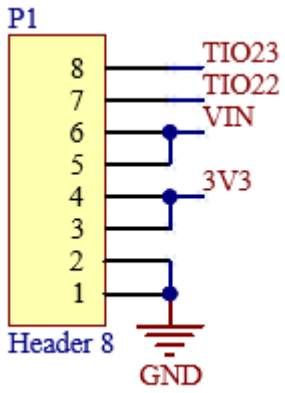
3.2.6 Top Overlay PCB Design



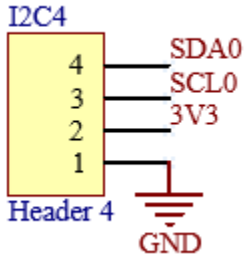
3.4 Controller Breakout Schematic Breakdown

3.4.1 16 Pin Header

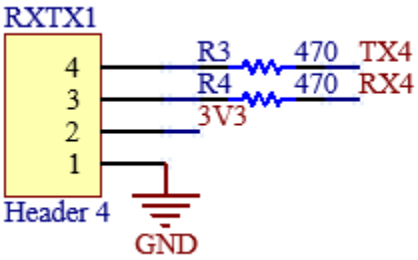
Left:



Right Bottom:

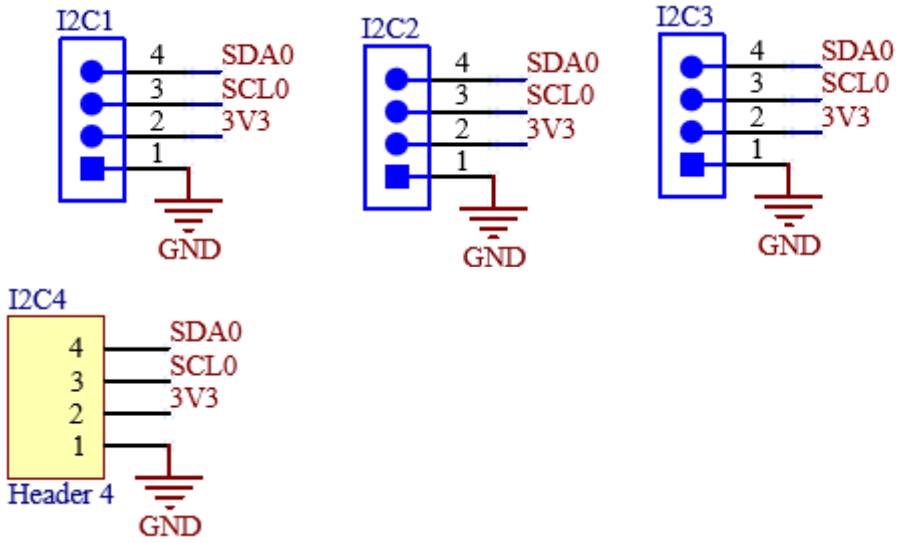


Right Top:

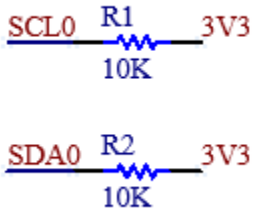


3.4.5 Teensy I2C Headers

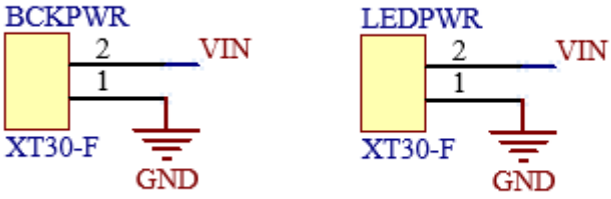
All four I2C headers are the same connections and can be used interchangeably



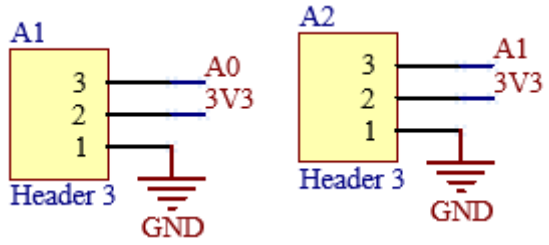
Built in pullups:



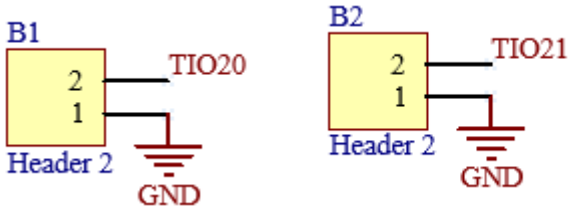
3.4.7 XT30 Power Input and Output



3.4.8 Analog Headers

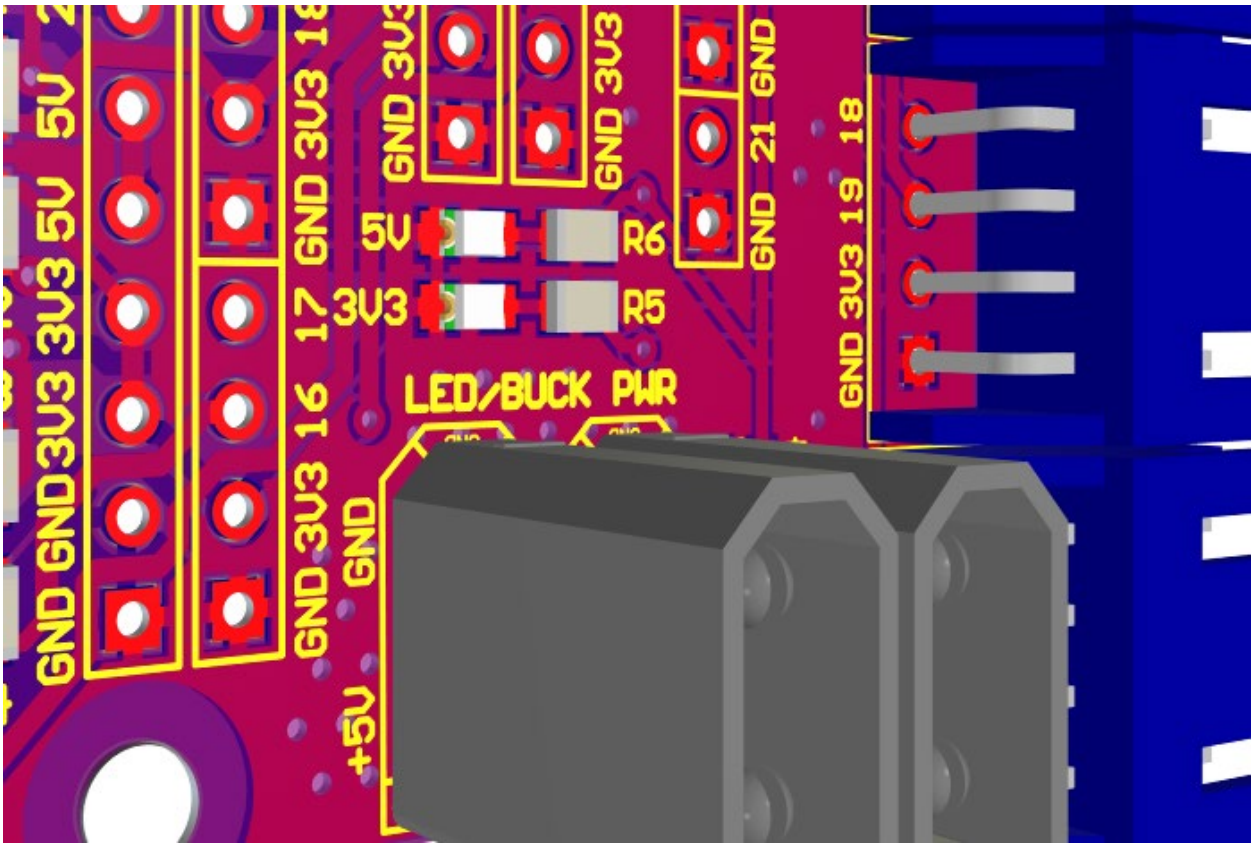


3.4.9 Digital Headers

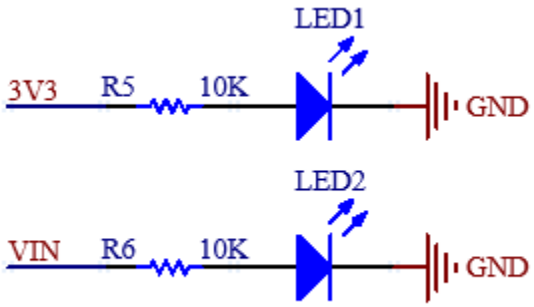


3.5 Indicators

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



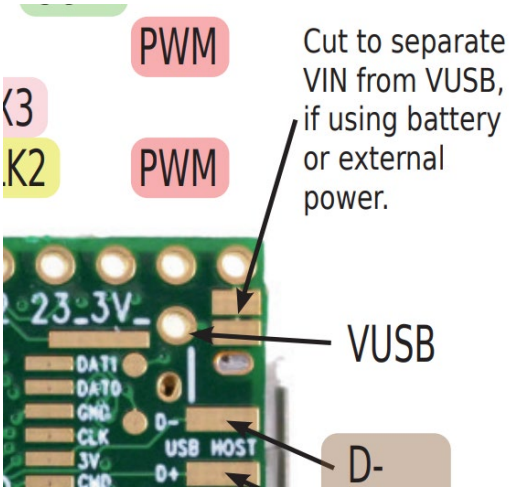
Schematic view of LED wiring:



3.6 Power Setup

3.6.1 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.

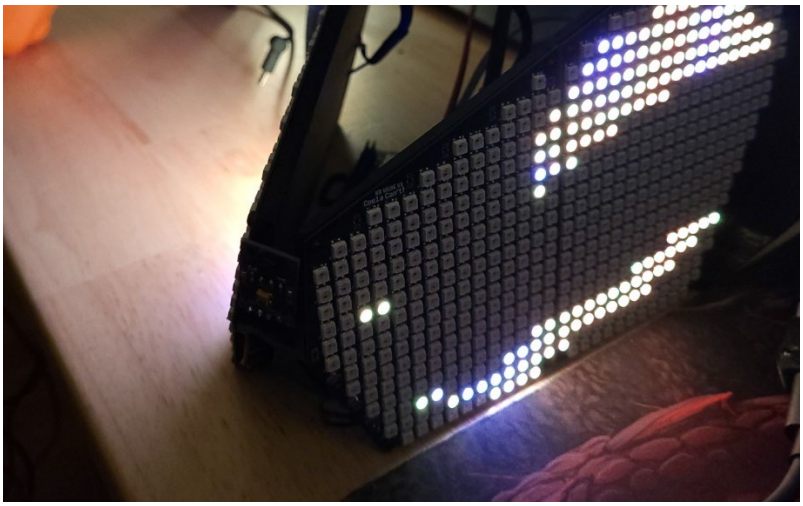
3.6.2 Powering the Controller for Usage

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

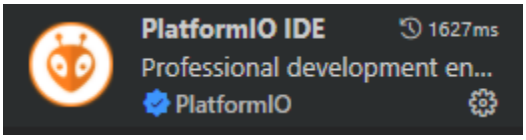
5.1.1 Install Applications

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

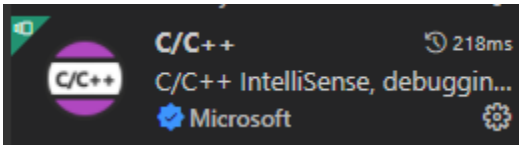
Install TeensyDuino from here: https://www.pjrc.com/teensy/td_download.html

5.1.2 Install Extensions

Install PlatformIO IDE under the extensions in Visual Studio Code:

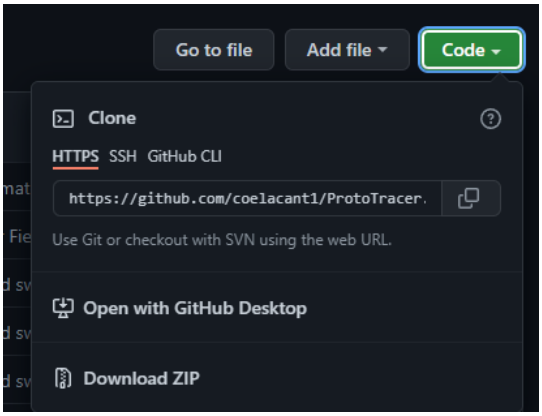


Install C/C++ extension:



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:



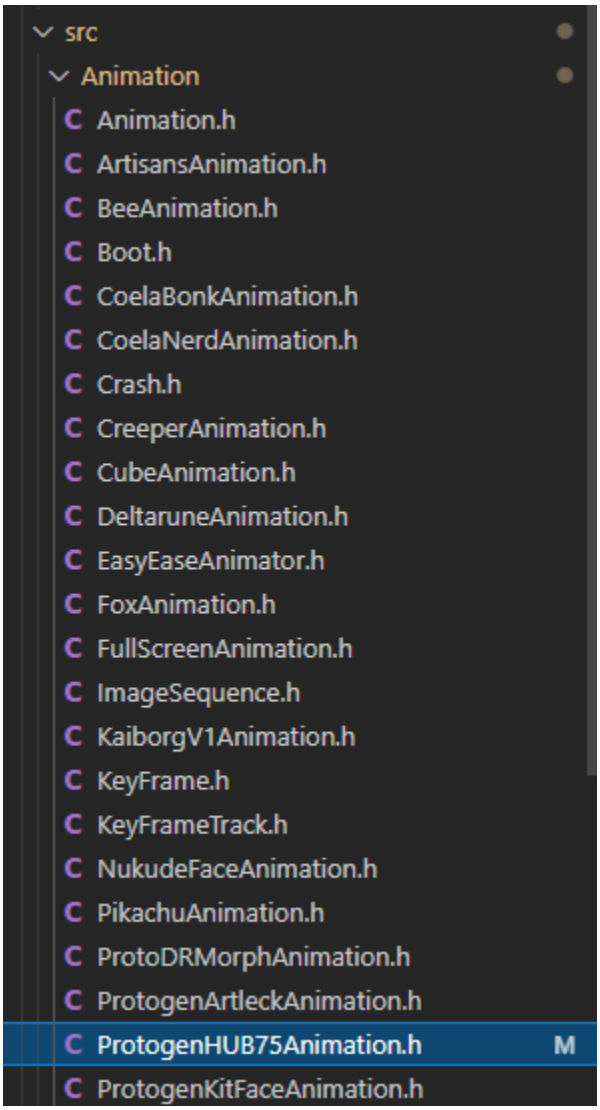
5.2 Loading a Controller

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 SmartMatrixHUB75 controller:

```
1. //---------------- ANIMATIONS ---------------
2. #include "Animation\ProtogenHUB75Animation.h"
3.
4. //---------------- CONTROLLERS ---------------
5. #include "Controllers\SmartMatrixHUB75.h"
6.
7. const uint8_t maxBrightness = 20;
8. Controller* controller = new SmartMatrixHUB75(maxBrightness);
9. Animation* animation = new ProtogenHUB75Animation();
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 ProtogenHUB75Animation.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:

```
#include <Arduino.h>
//#include <MatrixHardware_Teensy3_ShieldV4.h>        // SmartLED Shield for Teensy 3 (V4)
#include <MatrixHardware_Teensy4_ShieldV5.h>        // SmartLED Shield for Teensy 4 (V5)
//#include <MatrixHardware_Teensy3_ShieldV1toV3.h>    // SmartMatrix Shield for Teensy 3 V1-V3
//#include <MatrixHardware_Teensy4_ShieldV4Adapter.h> // Teensy 4 Adapter attached to SmartLED Shield for Teensy 3 (V4)
//#include <MatrixHardware_ESP32_V0.h>                // This file contains multiple ESP32 hardware configurations, edit the file to define GPIOPINOUT (or add #define GPIOPINOUT with a hardcoded number before this #include)
//#include "MatrixHardware_Custom.h"                  // Copy an existing MatrixHardware file to your Sketch directory, rename, customize, and you can include it like this
#include <SmartMatrix.h>

#include "Controller.h"
#include "Render/Camera.h"
#include "Flash/PixelGroups/P3HUB75.h"


#define COLOR_DEPTH 24                  // Choose the color depth used for storing pixels in the layers: 24 or 48 (24 is good for most sketches - If the sketch uses type `rgb24` directly, COLOR_DEPTH must be 24)
const uint16_t kMatrixWidth = 64;       // Set to the width of your display, must be a multiple of 8
const uint16_t kMatrixHeight = 64;      // Set to the height of your display
const uint8_t kRefreshDepth = 12;       // Tradeoff of color quality vs refresh rate, max brightness, and RAM usage.  36 is typically good, drop down to 24 if you need to.  On Teensy, multiples of 3, up to 48: 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48.  On ESP32: 24, 36, 48
const uint8_t kDmaBufferRows = 4;       // known working: 2-4, use 2 to save RAM, more to keep from dropping frames and automatically lowering refresh rate.  (This isn't used on ESP32, leave as default)
const uint8_t kPanelType = SM_PANELTYPE_HUB75_32ROW_MOD16SCAN;   // Choose the configuration that matches your panels.  See more details in MatrixCommonHub75.h and the docs: https://github.com/pixelmatix/SmartMatrix/wiki
const uint32_t kMatrixOptions = (SM_HUB75_OPTIONS_NONE);        // see docs for options: https://github.com/pixelmatix/SmartMatrix/wiki
const uint8_t kBackgroundLayerOptions = (SM_BACKGROUND_OPTIONS_NONE);

SMARTMATRIX_ALLOCATE_BUFFERS(matrix, kMatrixWidth, kMatrixHeight, kRefreshDepth, kDmaBufferRows, kPanelType, kMatrixOptions);
SMARTMATRIX_ALLOCATE_BACKGROUND_LAYER(backgroundLayer, kMatrixWidth, kMatrixHeight, COLOR_DEPTH, kBackgroundLayerOptions);

class SmartMatrixHUB75 : public Controller {
private:
    CameraLayout cameraLayout = CameraLayout(CameraLayout::ZForward, CameraLayout::YUp);

    Transform camTransform = Transform(Vector3D(), Vector3D(0, 0, -500.0f), Vector3D(1, 1, 1));

    PixelGroup camPixels = PixelGroup(P3HUB75, 2048);

```

```
    Camera cam = Camera(&camTransform, &cameraLayout, &camPixels);

    Camera* cameras[1] = { &cam };

public:
    SmartMatrixHUB75(uint8_t maxBrightness) : Controller(cameras, 1, maxBrightness){}

    void Initialize() override{
        matrix.addLayer(&backgroundLayer);
        matrix.begin();

        matrix.setBrightness(255);
        matrix.setRefreshRate(240);

        backgroundLayer.swapBuffers();//for ESP32 - first is ignored
    }

    void Display() override {
        for (int i = 0; i < 2048; i++){
            if (camPixels.GetPixel(i)->Color.R == 0 && camPixels.GetPixel(i)->Color.G == 0 && camPixels.GetPixel(i)->Color.B == 0){
                camPixels.GetPixel(i)->Color = camPixels.GetPixel(i)->Color.Scale(brightness).Add(16);
            }
        }

        for (uint16_t y = 0; y < 32; y++) {
            for (uint16_t x = 0; x < 64; x++){
                uint16_t pixelNum = y * 64 + x;

                rgb24 rgbColor = rgb24((uint16_t)camPixels.GetPixel(pixelNum)->Color.R, (uint16_t)camPixels.GetPixel(pixelNum)->Color.G, (uint16_t)camPixels.GetPixel(pixelNum)->Color.B);

                backgroundLayer.drawPixel(x, y, rgbColor);
                backgroundLayer.drawPixel(63 - x, y + 32, rgbColor);
            }
        }

        backgroundLayer.swapBuffers();
    }
};
```

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 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 short circuit the IO pins. Be careful about which pins are in use and are listed at the beginning of the guide.

Be careful leaving the electronics in a humid environment.

If you have a faulty device, please contact me before attempting repairs. Repairs are easy with the proper equipment but without this equipment you could cause more damage.