Hushcon Monkey Badge

August 2023 Version (3.4)

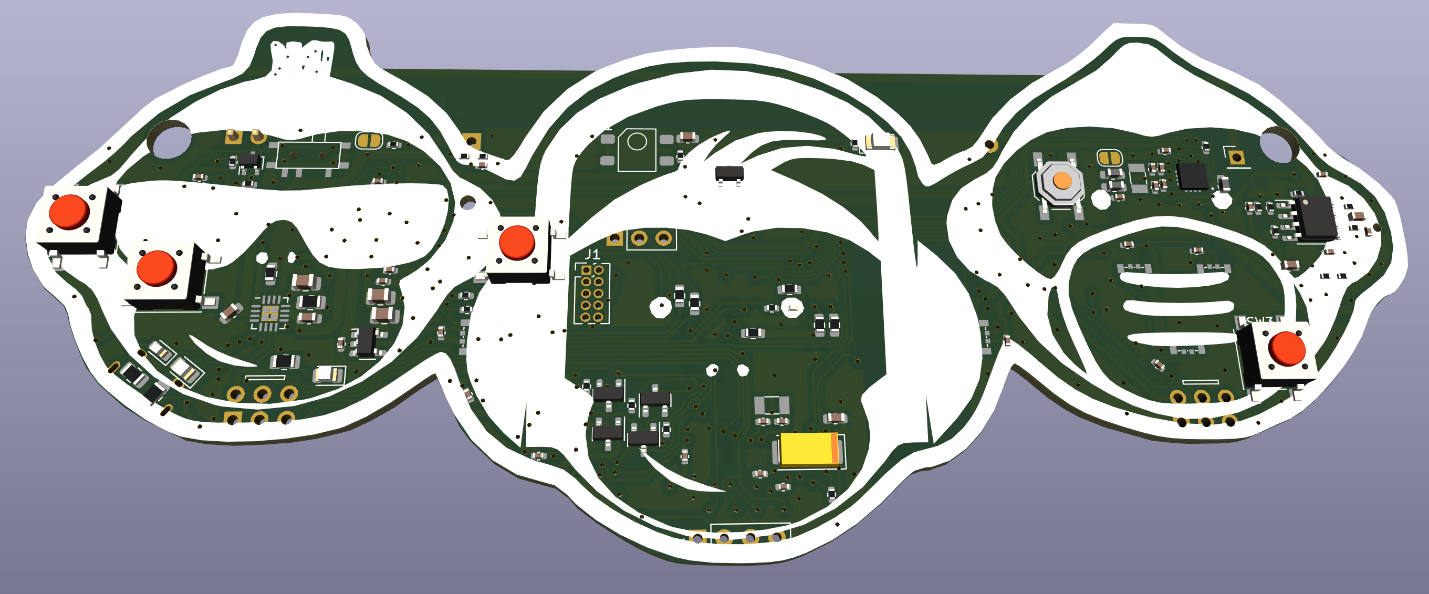
Datasheet, Programming Guide, wish-list

Jonathan Taylor (Temtel), Crackerjack and Redbeard

Copyright 2022-2023, PacketChat Inc.

Version 20230819

Designed with KICad 6.0



# Table of Contents

[1 Table of Contents 1](#_heading=h.30j0zll)

[2 High-Level Description 3](#_heading=h.1fob9te)

[3 Theory of Operation 4](#_heading=h.3znysh7)

[4 Badge Features 4](#_heading=h.2et92p0)

[5 Setting up your dev environment 5](#_heading=h.tyjcwt)

[5.1 Use the right cable 5](#_heading=h.3dy6vkm)

[5.2 Virtualbox Ubuntu 22.04 5](#_heading=h.1t3h5sf)

[5.2.1 Once in Virtualbox VM running ubuntu: 5](#_heading=h.4d34og8)

[5.2.2 Get rid of Apps that claim /dev/ttyUSB0 6](#_heading=h.2s8eyo1)

[5.2.3 add user to Dialout group 6](#_heading=h.17dp8vu)

[5.2.4 install Python3 & pip: 6](#_heading=h.3rdcrjn)

[5.2.5 Install & use mpremote: 6](#_heading=h.26in1rg)

[5.2.6 Create a filesystem on the badge 7](#_heading=h.lnxbz9)

[5.2.7 mpremote fs cp 7](#_heading=h.35nkun2)

[5.3 Interacting with the badge 7](#_heading=h.1ksv4uv)

[5.4 Built-in monkeybadge libraries 7](#_heading=h.44sinio)

[5.5 Obtaining a fresh monkeybadge image 8](#_heading=h.2jxsxqh)

[6 Technical Details 8](#_heading=h.z337ya)

[7 WS2812B-compatible Addressable RGB LEDs (SK6812D-EC3210R) 8](#_heading=h.3j2qqm3)

[7.1 Code to activate the addressable LED’s: 8](#_heading=h.1y810tw)

[7.2 Addressable LED Needs: 10](#_heading=h.4i7ojhp)

[8 Control Buttons 10](#_heading=h.1ci93xb)

[8.1 Code to configure control buttons: 11](#_heading=h.2bn6wsx)

[9 ESP32-WROOM-32E-8MB 12](#_heading=h.qsh70q)

[9.1 I2C on the ESP32 12](#_heading=h.3as4poj)

[9.2 Power Consumption 13](#_heading=h.49x2ik5)

[9.3 Wifi 13](#_heading=h.2p2csry)

[9.3.1 Default WiFi network 13](#_heading=h.147n2zr)

[9.3.2 Code to connect to a wifi network: 13](#_heading=h.3o7alnk)

[9.4 Bluetooth 14](#_heading=h.23ckvvd)

[9.5 Microphone Audio Input 14](#_heading=h.ihv636)

[9.6 ESP32 Programming 14](#_heading=h.32hioqz)

[9.7 ESP32 Needs 14](#_heading=h.1hmsyys)

[10 Power 15](#_heading=h.2grqrue)

[10.1 USB Charge Controller & Battery Unit 15](#_heading=h.vx1227)

[10.2 Size 14500 Lithium Ion Battery (AA Size but 3.7V) 15](#_heading=h.3fwokq0)

[10.3 Battery Monitor Pin 15](#_heading=h.1v1yuxt)

[10.3.1 Code to monitor the battery charge level 15](#_heading=h.4f1mdlm)

[10.4 3.3V Regulator 16](#_heading=h.19c6y18)

[10.5 5V Boost Converter 16](#_heading=h.3tbugp1)

[11 Shitty Add-On Connectors 16](#_heading=h.28h4qwu)

[12 0.96” SSD1306 based OLED Display 17](#_heading=h.37m2jsg)

[12.1 Code to activate the SSD1306 OLED display: 18](#_heading=h.46r0co2)

[13 SI4702 Stereo FM Broadcast Radio Receiver 18](#_heading=h.2lwamvv)

[13.1 FM Receive Audio monitoring by ESP32 18](#_heading=h.111kx3o)

[13.2 FM Broadcast Receive Antennas 18](#_heading=h.3l18frh)

[13.3 I2C bus and the SI4702 19](#_heading=h.4k668n3)

[13.4 SI4702 software Needs: 19](#_heading=h.2zbgiuw)

[13.5 Code to activate and use the SI4702 FM Receiver 19](#_heading=h.1egqt2p)

[14 LM4808 Headphone Amp & TRRS Jack (V 2.1 and later) 19](#_heading=h.3ygebqi)

[14.1 Amplifier Input 20](#_heading=h.2dlolyb)

[14.2 Amplifier Output 20](#_heading=h.3cqmetx)

[14.3 Microphone Amplifier (3.5 and later) 20](#_heading=h.4bvk7pj)

[14.4 4 position (TRRS) Headphone Jack 20](#_heading=h.2r0uhxc)

[15 Infrared Receiver and Emitter 21](#_heading=h.1664s55)

[15.1 Code to activate Infrared receiver and emitter 21](#_heading=h.25b2l0r)

[16 Monkeybadge Software 22](#_heading=h.kgcv8k)

[17 Monkey Badge version 3.4 Images 23](#_heading=h.34g0dwd)

[18 V1.0 Errata 28](#_heading=h.xvir7l)

[19 V1.1 Errata 28](#_heading=h.3hv69ve)

[1. USB D+ and D- are criss-crossed (RESOLVED) 28](#_heading=h.1x0gk37)

[2. Continuous machine noise in the earphone (REDESIGNED) 28](#_heading=h.4h042r0)

[3. No pullup resistors on the I2c Bus (FIXED) 28](#_heading=h.2w5ecyt)

[4. Battery holder pins slightly off position(FIXED) 28](#_heading=h.1baon6m)

[5. RST net was never connected to the EN pin on the ESP32. (RESOLVED) 28](#_heading=h.3vac5uf)

[6. This may still be related to #5, but Reset short is not working. 28](#_heading=h.2afmg28)

[7. Can’t get the FM Receiver chip working (FIXED in 2.1). 28](#_heading=h.pkwqa1)

[8. A 470ohm resistor was improperly added between the EN pin and ground. (Resolved) 29](#_heading=h.39kk8xu)

[20 V 2.0 Errata 29](#_heading=h.1opuj5n)

[21 V2.2 Errata 29](#_heading=h.48pi1tg)

[22 29](#_heading=h.2nusc19)

[23 Table of Figures 29](#_heading=h.1302m92)

# High-Level Description

See no evil. Hear no evil. Speak no evil.

This is the Huschcon Seattle 2023 Electronic badge by Temtel, Redbeard, Crackerjack, Faxanadu, theFreak, Ofir, and Dilligence, powered by PacketChat. Special mention of thanks to Faxanadu, thePhreak and Ofir for software & challenge help. Based on an 8MB version of the ESP32 Wifi/Bluetooth controller, this 2-layer badge packs in a whole crowd of rich multi-media hardware capability just screaming to be hacked.

There’s a full FM stereo radio on-board, with pins hooked up to the ESP32. After the con, you can trade Wifi for Bluetooth and allow possible connection to Bluetooth earphones or speaker, with the right code to support it (hear no evil). There’s also an audio amplifier and earphone jack with MIC support that could potentially be used for some kind of audio bridge conferencing solution (speak no evil). Side-mount multi-colored smart LED’s flank the reflective white silk of each monkey for the theme—Sunglasses, Headphones, and Mask allowing code to deliver themed light animations, integration with audio, or remote control by conference staff. (See no evil). 4 control buttons are connected to interrupt GPIO pins that are also capable of capacitive touch, and it’s equipped with a USB charge controller and a 14500 lithium-ion rechargeable battery (same size as a AA battery but 3.7v/1500mAh).

And just for fun, two Shitty Add-on connectors, an Infrared transceiver, and an I2C Oled interface are included for those badge-life interactions and add-ons we all adore. The hackable software and hardware add-on possibilities are many, but the designer, Temtel, is a terrible programmer and needs someone much more imaginative to come up with the perfect suite of hackable function and easter-egg.

Credit goes to Crackerjack for the monkey design concept and artwork, and MUCH gratitude to Redbeard for designing the core of the software, and even making a world-wide impact for the better by becoming a core contributor to the improvement of both BadgePython and MicroPython! We’ll let him tell that story though– buy him a beer and he’ll tell you all about it.

# Theory of Operation

Blinky-lights at activation, occasional wake-up, attach to wifi, listen for instructions, & animate to show it’s alive but need to conserve power. Buttons will also wake it from deep-sleep mode. We’ll have a low-power FM station running during the con, and a badge wifi network. When we need to, we can take control of everyone’s badges, flash the lights in sync, tune their FM receiver to a specific station and play a silent rave. The blinky lights in sync with others is a sign that it’s time to rave—plug in your earphones.

We want to code some interactions, animations, challenges, games & easter eggs into the badge. That’s where you come in!

# Badge Features

* ESP32-WROOM-32-8MB
  + 802.11n Wifi
  + Bluetooth Low-Energy
  + 8MB Flash
* FM Broadcast Receiver based on the SI4702 chip, I2C Controlled
* Op-amp based headphone amplifier (LM4808)
* 4 Position headphone+Mic (TRRS OMTP) 3.5mm Audio Connector
* 1500mAh Lithium Ion 3.7v 14500 size (AA size) Battery
* Texas Instruments BQ24072 Charge & Power Path Controller
* Super high efficiency RT9080 3.3V LDO Regulator
* 7 side-mount addressable RGB LED’s, compatible w/ WS2812B.
* 4 control buttons (Up, Down, Enter, Mute or whatever you want them to be)
* 2 Shitty Add-on V1.69bis connectors, each with I2C, 3.3V, and interrupt GPIO’s (Some are shared, see schematic for conflicts!)
* USB-Micro Serial & charge port
* CH340T USB Serial chip
* 0.96” I2C-based SSD1306 Monochrome OLED (external)
* Infrared receiver (IRM-H638T-TR2) and Infrared Emitter
* JTAG Interface
* High-quality TRRS earphones w/ microphone

# Setting up your dev environment

## Use the right cable

Programming the monkeybadge requires a USB Micro DATA cable, not a power-only cable. Be sure to get a USB Micro cable that includes the data pins, and is not Charge-only. I wasted a LOT of time trying to figure out what was wrong with my driver, when it was just a dumb power cable. Duh.

The USB serial chip on the monkeybadge doesn’t get power until you turn the power switch on. Your computer won’t see anything on the serial port until the badge is powered on. (Wasted a lot of time on that too, double-duh.)

My dev environment was an Ubuntu 22.04 virtual machine on Virtualbox on a Windows host.

## Virtualbox Ubuntu 22.04

The Monkeybadge has a CH340C USB to Serial adapter chip. When you run ubuntu on a Windows virtualbox host, it \*should\* auto-detect the device, but if it doesn't, you may need to install the

manufacturer's CH341 drivers first. The official driver is here: https://wch-ic.com/products/CH340.html

### Once in Virtualbox VM running ubuntu:

To attach the USB Serial adapter on the Monkeybadge:

1. right-click on the USB plug icon.
2. select QuinHeng Electronics USB Serial [8134]

A computer screen with a computer screen

Description automatically generated with medium confidence

This will detatch the COM port from windows, and attach it to the virtual host. The ubuntu kernel should detect the monkeybadge as /dev/ttyUSB0.

### Get rid of Apps that claim /dev/ttyUSB0

Next problem: Two programs like to claim /dev/ttyUSB0 at boot-- you'll need to mask or remove them. Easiest way:

sudo apt remove brltty

sudo apt remove modemmanager

### add user to Dialout group

Add your user to the dialout user's group so it can access /dev/ttyUSB0:

sudo usermod -a -G dialout temtel

### install Python3 & pip:

sudo apt install python3 python3-pip python-is-python3

### Install & use mpremote:

pip install --user mpremote

Use mpremote to connect to your monkeybadge via serial port:

/home/temtel/.local/bin/mpremote connect /dev/ttyUSB0

Press Enter a couple of times to get a python prompt:

>>>

Coolios! You can now start interacting with the badge within python.

However...

Manually entering commands or copy/paste is slow. You can use mpremote to upload a python script or library. First, disconnect from the badge, using CTRL+x.

### Create a filesystem on the badge

I DON’T THINK THIS WORKS YET—NEED TO TALK TO REDBEARD

*# ESP8266 and ESP32*

**import** **os**

os.umount('/')

os.VfsLfs2.mkfs(bdev)

os.mount(bdev, '/')

### mpremote fs cp

\*\*\*\*\*\*this is not working yet—create filesystem?

Next, use mpremote's filesystem copy command to upload your library.

mpremote fs cp myscript.py : myscript.py

To execute the script or load the library, use the standard Python import command:

import myscript

Boom, you're in business. Here’s a brief script to test some of the parts on the badge:

<https://drive.google.com/file/d/1CkMYsE-FcPje_vikxHWN_hh1BQW8gPuZ/view?usp=drive_link>

## Interacting with the badge

Your Monkeybadge’s base microcode is an improved version of Micropython and Badgepython. When you connect via mpremote (see 5.2.5), you’re looking at a live Python interactive prompt. Use the features of mpremote to upload libraries to the filesystem and import them. You can then execute your code, one line at a time, or by executing your own script.

## Built-in monkeybadge libraries

There's a bunch of libraries smashed onto the default monkeybadge. Many of them come from the standard Micropython ESP32 distribution, but they have been custom-curated by Redbeard to include the ones supported by our hardware.

firmware with super-tight compression See URL for API reference.

Micropython Documentation: <https://docs.micropython.org/en/latest/>

\*\*\*need a publication point for the MonkeyBadge-specific hardware API’s, such as the SI4702 FM Receiver

There’s a great deal of additional info out there on ESP32 programming. I especially appreciated this website while designing this badge:

<https://randomnerdtutorials.com/esp32-pinout-reference-gpios/>

## Obtaining a fresh monkeybadge image

(This may be outdated– check with Redbeard)

~~wget~~ [~~https://www.brianredbeard.com/projects/monkeypython-alpha.tgz~~](https://www.brianredbeard.com/projects/monkeypython-alpha.tgz)

wget <https://ultra.weir.do/monkeybadge_firmware-2023-09-08.tgz> #(updated 9/8/2023)

pip install esptool.py

pip install setuptools

esptool --chip esp32 erase\_flash #may not be necessary

tar -xf monkeybadge\_firmware-2023-09-08.tgz

cd monkeybadge\_firmware-2023-09-08/

./flash.sh

Edited: 2023-11-16

If you’re supplied with a firmware.bin file:

pip install esptool.py

pip install setuptools

esptool.py --chip esp32 erase\_flash

esptool.py --chip esp32 --port /dev/ttyUSB0 write\_flash -z 0x1000 firmware.bin

The results you should get:

temtel@ubuntu22:~/monkeybadge\_firmware-2023-09-08$ ./flash.sh

esptool.py v4.6.2

Found 1 serial ports

Serial port /dev/ttyUSB0

Connecting......

Detecting chip type... Unsupported detection protocol, switching and trying again...

Connecting...........

Detecting chip type... ESP32

Chip is ESP32-D0WD-V3 (revision v3.1)

Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None

Crystal is 40MHz

MAC: b0:a7:32:20:96:04

Uploading stub...

Running stub...

Stub running...

Configuring flash size...

Flash will be erased from 0x00001000 to 0x00006fff...

Flash will be erased from 0x00010000 to 0x001b7fff...

Flash will be erased from 0x00008000 to 0x00008fff...

Flash will be erased from 0x0000d000 to 0x0000efff...

Compressed 22672 bytes to 14456...

Wrote 22672 bytes (14456 compressed) at 0x00001000 in 1.4 seconds (effective 125.9 kbit/s)...

Hash of data verified.

Compressed 1733024 bytes to 1161955...

Wrote 1733024 bytes (1161955 compressed) at 0x00010000 in 102.5 seconds (effective 135.2 kbit/s)...

Hash of data verified.

Compressed 3072 bytes to 142...

Wrote 3072 bytes (142 compressed) at 0x00008000 in 0.1 seconds (effective 462.6 kbit/s)...

Hash of data verified.

Compressed 8192 bytes to 31...

Wrote 8192 bytes (31 compressed) at 0x0000d000 in 0.1 seconds (effective 818.4 kbit/s)...

Hash of data verified.

Leaving...

Hard resetting via RTS pin...

temtel@ubuntu22:~/

# Technical Details

Each of the badge components are described in detail below, including how they’re hooked up.

# WS2812B-compatible Addressable RGB LEDs (SK6812D-EC3210R)

All of the addressable LED’s are controlled via a single GPIO pin like a string of lights. This is on GPIO18, or pin 30 of the ESP32 chip. These LED’s usually want 5V, but they have been known to work at much lower voltages if there aren’t too many on the same string. To save costs, we’re running the RGB’s at the battery voltage delivered by the charge/power-path device, which should range between 4.1 and 3.2V under normal use. It’s technically under-spec for the lights, but it works and saves costs. The BSS138 is meant to allow WS2812B-4020’s to hear logic at the higher battery voltage instead of the 3.3V spoken by the ESP32.

## Code to activate the addressable LED’s:

import machine, neopixel, time, network

from machine import Pin, I2C

import ssd1306

np = neopixel.NeoPixel(machine.Pin(18), 7)

slval = 50

#Addressable LED Test

def do\_all\_off():

for i in range(7):

np[i] = (0, 0, 0)

np.write()

def do\_canz\_spaz():

np[2] = (0, 0, 0)

np[3] = (0, 0, 0)

for i in range(255):

np[2] = (25, 32, i)

np[3] = (i, 55, 32)

#time.sleep\_ms(50)

np.write()

np[2] = (0, 0, 0)

np[3] = (0, 0, 0)

np.write()

do\_canz\_spaz()

#def do\_cans\_spaz():

#def do\_mic\_spaz():

def do\_greensweep():

for i in range(7):

np[i] = (0, 0, 0)

np.write()

time.sleep\_ms(slval)

for i in range(7):

np[i] = (0, 32, 0)

np.write()

time.sleep\_ms(slval)

do\_greensweep()

do\_all\_off()

def do\_redsweep():

for i in range(7):

np[i] = (0, 0, 0)

np.write()

time.sleep\_ms(slval)

for i in range(7):

np[i] = (32, 0, 0)

np.write()

time.sleep\_ms(slval)

do\_redsweep()

do\_all\_off()

def do\_bluesweep():

for i in range(7):

np[i] = (0, 0, 0)

np.write()

time.sleep\_ms(slval)

for i in range(7):

np[i] = (0, 0, 32)

np.write()

time.sleep\_ms(slval)

do\_bluesweep()

do\_all\_off()

def do\_purplesweep():

for i in range(7):

np[i] = (0, 0, 0)

np.write()

time.sleep\_ms(slval)

for i in range(7):

np[i] = (32, 0, 32)

np.write()

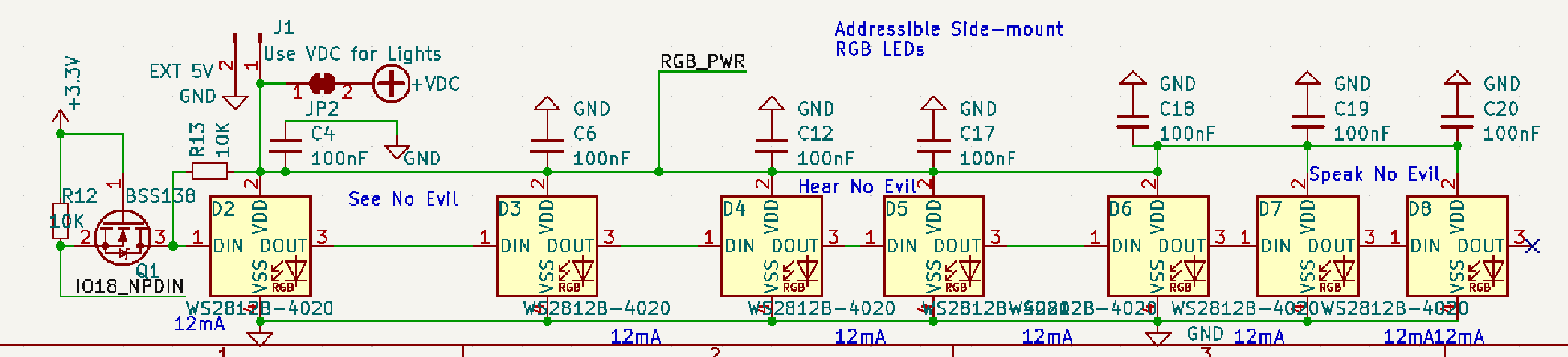
time.sleep\_ms(slval)

do\_purplesweep()

do\_all\_off()

## Addressable LED Needs:

1. Software to initialize RGB’s DONE!
2. Software animation sequences for the LED’s. NEED MORE!
3. Ability to take control of LED’s on all badges over WiFi
4. Software animation sequence for when badges become zombies to the overlord and tune to the same frequency
5. Animation that responds to audio received from the FM receiver.



*Figure 1 - Side-mount WS2812B RGB LED's*

# Control Buttons

This is going to take some coding. We’ve got 4 buttons to work with—I’ve labeled them Up, Down, Enter and Mute in the schematic. Each button has a GPIO pin on one side, and ground on the other side with no resistor. Up, Enter and Mute are also on realtime GPIO pins, that can be used to wake the badge from deep sleep.

| Physical Pin # | GPIO# | Button |
| --- | --- | --- |
| 26 | IO4 | Up |
| 13 | IO14 | Down |
| 23 | IO15 | Enter |
| 16 | IO13 | Mute |

A diagram of a radio control buttons

Description automatically generated

*Figure 2 - Control Buttons*

## Code to configure control buttons:

This needs work! Need to add software debounce—couldn’t do it in hardware because the buttons share GPIO’s with the JTAG interface.

#Buttons

#Button GPIO's are as follows: UP=4, DN=14, CANZ=15, mic=13

from machine import Pin

def up\_pressed(pin):

print('up button pressed')

do\_greensweep()

do\_all\_off()

def dn\_pressed(pin):

print('down button pressed')

do\_redsweep()

do\_all\_off()

def canz\_pressed(pin):

print('canz button pressed')

do\_bluesweep()

do\_all\_off()

def mic\_pressed(pin):

print('mic button pressed')

do\_purplesweep()

do\_all\_off()

p\_up = Pin(4, Pin.IN, Pin.PULL\_UP)

p\_dn = Pin(14, Pin.IN, Pin.PULL\_UP)

p\_canz = Pin(15, Pin.IN, Pin.PULL\_UP)

p\_mic = Pin(13, Pin.IN, Pin.PULL\_UP)

p\_up.irq(handler=up\_pressed, trigger=Pin.IRQ\_FALLING)

p\_dn.irq(handler=dn\_pressed, trigger=Pin.IRQ\_FALLING)

p\_canz.irq(handler=canz\_pressed, trigger=Pin.IRQ\_FALLING)

p\_mic.irq(handler=mic\_pressed, trigger=Pin.IRQ\_FALLING)

print('Buttons loaded. Please press each button one at a time to test.')

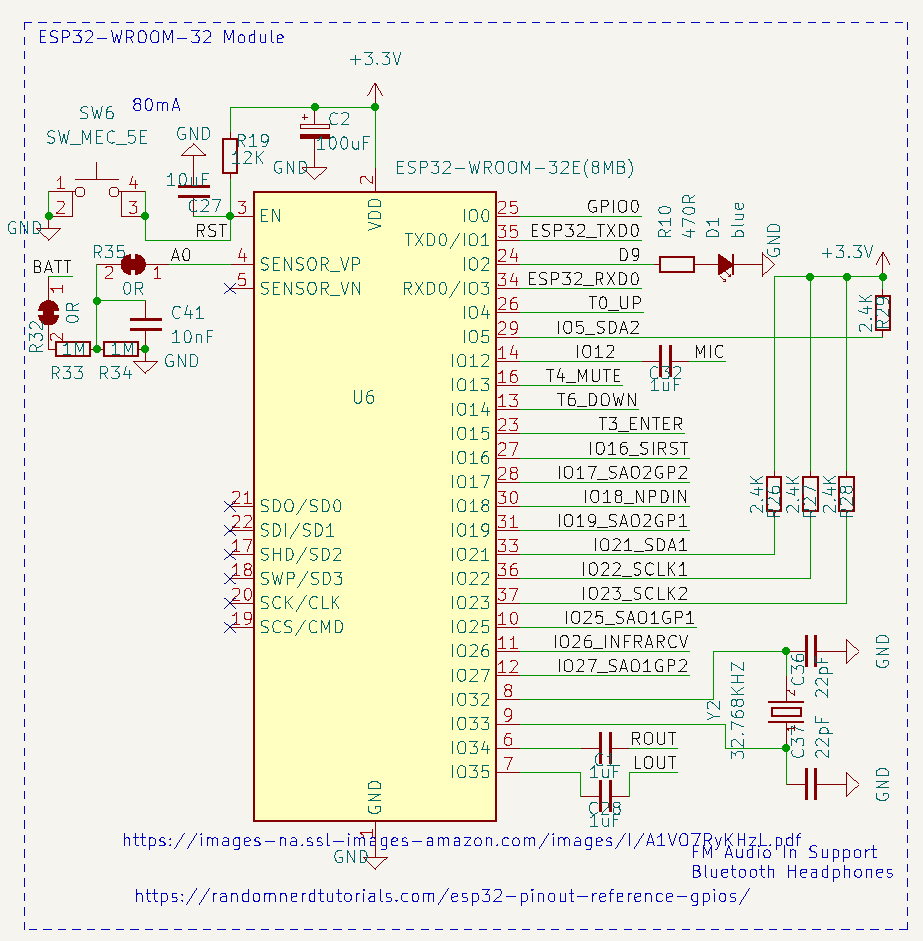
# ESP32-WROOM-32E-8MB

The main brains of the whole badge. This is the same popular ESP32 WiFi & Bluetooth module by Expressif, but with double the flash memory at 8MB instead of 4. The 32 “E” version means Eco extensions are included in the processor microcode.

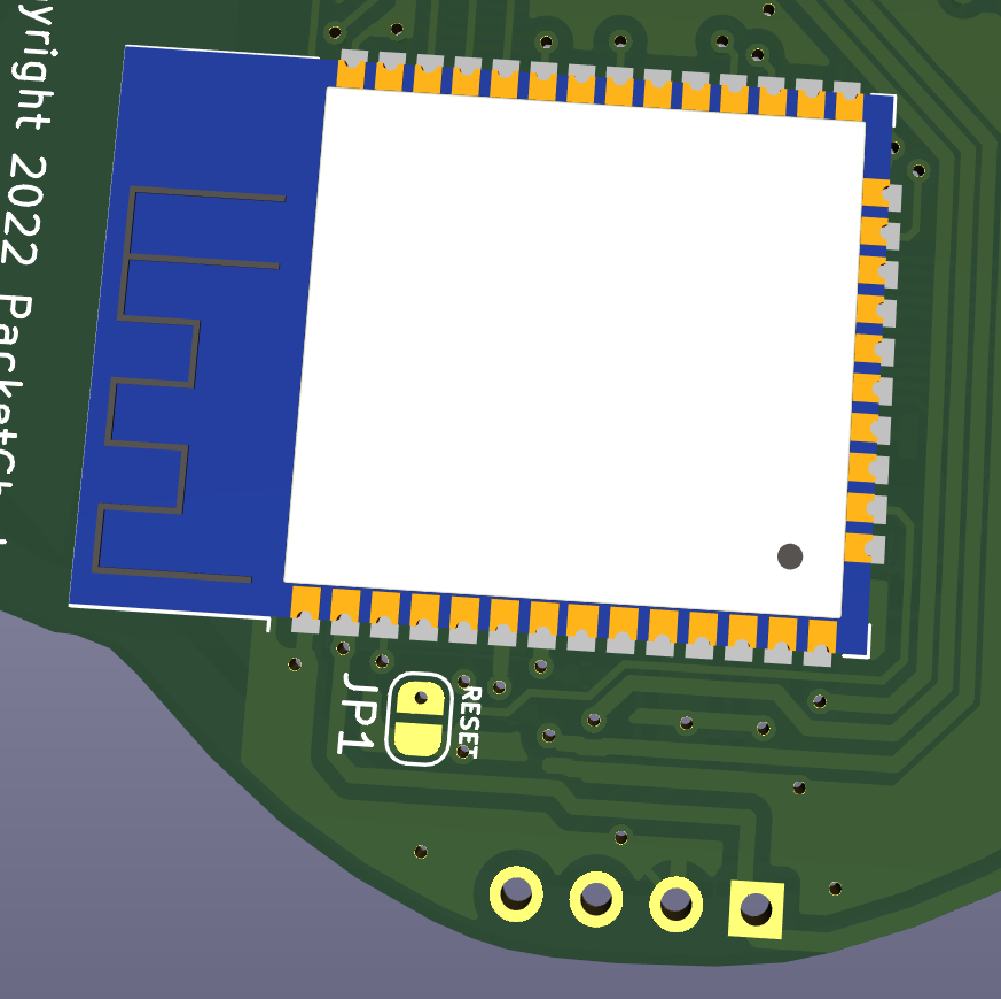
Hardware-wise the ESP32 on the Monkeybadge is setup similarly to a FireBeetle ESP32 Board, complete with automatic transistor switches to put it into programming mode with the DTR and RTS pins on the USB Serial port. Note however, we’re using a CH340T USB Serial adapter instead of an FTDI chip. Board definitions should take care of the differences. While they’re probably not applicable anymore, linked here are the original FireBeetle development libraries. <https://www.dfrobot.com/product-1590.html> or <https://images-na.ssl-images-amazon.com/images/I/A1VO7RyKHzL.pdf>

## I2C on the ESP32

For version 2.0 and higher of the monkey badge, the I2c devices on the right side of the monkey badge (FM Receiver and Shitty Add-on #2) are designed to use the ESP32’s second I2C bus, while those on the left side use the first I2C bus. Bus 2 is Pin 29 GPIO5 (SDA2) and Pin 37 GPIO23 (SCLK2). Bus 1 is Pin 33 GPIO21 (SDA1) and Pin 36 GPIO22(SCLK1). Software should ensure to activate the second I2C bus on these pins, as this is not a default state for the ESP32. Both I2C buses have 2.4K pullup resistors connecting them to the 3.3V rail.



*Figure 3 - Version 2.0 schematic of the ESP32 on the Monkey badge, showing pullups on both I2c Busses.*



## Power Consumption

1. Software that takes advantage of sleep mode to extend battery life should be implemented.

## Wifi

### Default WiFi network

Badge should default to connecting to a specific Wifi network for the con. This should be defined by the software developer and recorded here.

### Code to connect to a wifi network:

Note: This code relies on functions like redsweep() and greensweep() to indicate wifi success. See section for these functions.

#Wifi Tests

import network

wlan = network.WLAN(network.STA\_IF)

wlan.active(True)

if not wlan.isconnected():

print('connecting to network...')

do\_redsweep()

wlan.connect('StrongFooW', 'easydoesit')

while not wlan.isconnected():

pass

do\_greensweep()

print('network config', wlan.ifconfig())

do\_all\_off()

## Bluetooth

1. Output audio received on IO34 Pin 6 (Right Channel) and IO35 Pin 7 (Left Channel) to a Bluetooth sink, such as a Bluetooth speaker or Bluetooth headphones. It’s a weak audio signal coming directly from the radio through a 1uF DC blocking capacitor, so it’s likely microphone level. Possible libraries and source code: <https://www.pschatzmann.ch/home/2021/04/25/esp32-sound-processing-library/>
2. Possible way to send Bluetooth Audio: <https://www.pschatzmann.ch/home/2020/09/15/sending-sound-from-an-esp32-to-a-bluetooth-sink-e-g-bluetooth-speaker/>

## Microphone Audio Input

The wired microphone pin coming from the TRRS audio connector is attached to Pin 14 GPIO12 ADC2 Channel 5. 0through a 1uF DC Blocking capacitor.

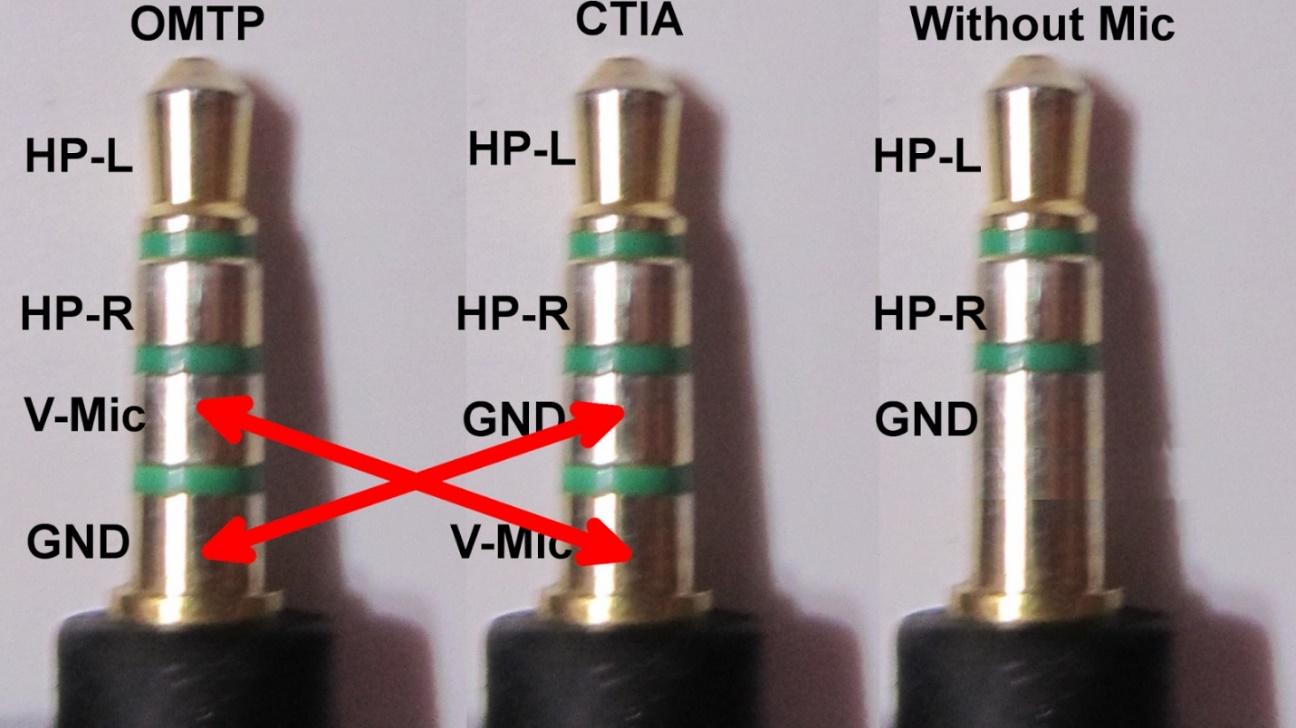
## ESP32 Programming

As of the second prototype (Version 1.1), programming can be done via the USB Serial port. The DTR and RTS pins are connected to transistor switches that control same pins on the ESP32 used by the FireBeetle ESP32, so fingers crossed, use the Firebeetle definition files and you should be able to program the monkeybadge.

Programming the ESP32 on the Monkeybadge is done via the USB port. Treat it like a LUA, Firebeetle, or other ESP32 board. (It’s most like the Firebeetle). The reset pin is attached to the Serial DTR, so it can be pulsed in software with Picocom to do a soft reset. If for some reason the USB serial is not working, the serial pins of the ESP32 are also broken out with 3 through-hole pins in the center of the badge, and you can short the reset pin with the low-profile Button SW6 on the front near the eye of the 3rd monkey.

## ESP32 Needs

1. Configure physical pins 6(Right Audio), 7(Left Audio) and 12 (wired headset micrphone) to receive analog audio and,
   1. convert it to Bluetooth audio
   2. Trigger animations or light patterns
   3. Make some kind of badge-based VOIP or bluetooth audio chat bridge
2. Games or animations that bring people together. Maybe a Bluetooth proximity detection feature that exchanges points for being close by and socializing?
3. Bluetooth text chat or fighting game on the OLED



*Figure 4 - OMTP vs CTIA vs TRS Pinouts*

The TRRS Microphone pin connects to the ESP32 chip through a 1uF DC blocking capacitor at pin 14, which is GPIO12.

# Power

Badge power is handled by a full powerpath controller and battery charger chip.

## USB Charge Controller & Battery Unit

Typical 14500 sized Lithium Ion batteries call for charge between 0.5 and 0.75 Amps. As designed with control resistors, the BQ24072RGT based charge controller is programmed to charge at a rate of 0.67 Amps. ~~There’s a 6.25hr limit on the charge time~~, and the USB power limits itself to 1-Amp of power draw from the USB bus. If there are any troubles with these settings, let Temtel know before we order the production run.

## Size 14500 Lithium Ion Battery (AA Size but 3.7V)

Batteries on order are 1500mAh capacity. Testing on samples show their actual capacity ranges between 1400 and 1600 and seems to last longer when draw rate is between 150mA and 500mA continuous.

## Battery Monitor Pin

Battery Power can be directly monitored by the ESP32, using the SensVP pin (GPIO36) of the ESP32. It is connected through a dual 1Mohm voltage divider. Solder Jumper R32 can be cut to detach the battery from the voltage divider, and Solder Jumper R35 can be cut to detach the SensVP pin if problems arise.

### Code to monitor the battery charge level

#battery Voltage

#The 5th pin down on the left hand side (GPIO 36 or ADC1\_0) with the antenna up is connected to

#a voltage divider (two 1M resistors) and the + battery pin. My voltmeter

#shows that when the battery is fully-charged,its at 1.94v. For this to work however,

#you'll need a remote terminal.

print('Testing the battery charge detector with a potentiometer on Pin 36. Potentiometer value is:')

from machine import ADC

Potentiometer = ADC(Pin(36) #GPIO Pin 26 defined for input

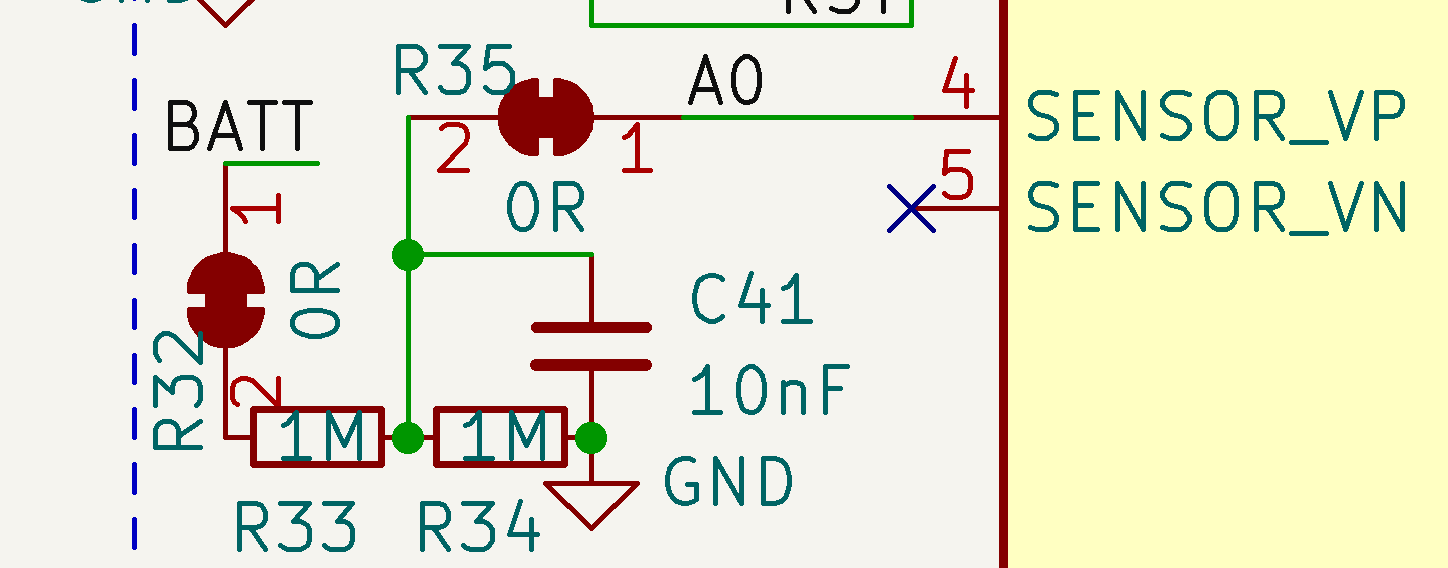
#Potentiometer.atten(ADC.ATTN\_11DB) #TODO: This is for 3.3V, need better value

# while True:

Potentiometer\_val = Potentiometer.read()

print(Potentiometer\_val)

# sleep(1)

 A green circuit board with white text and white letters

Description automatically generated

*Figure 5 - Battery Monitor Pin on ESP32*

## 3.3V Regulator

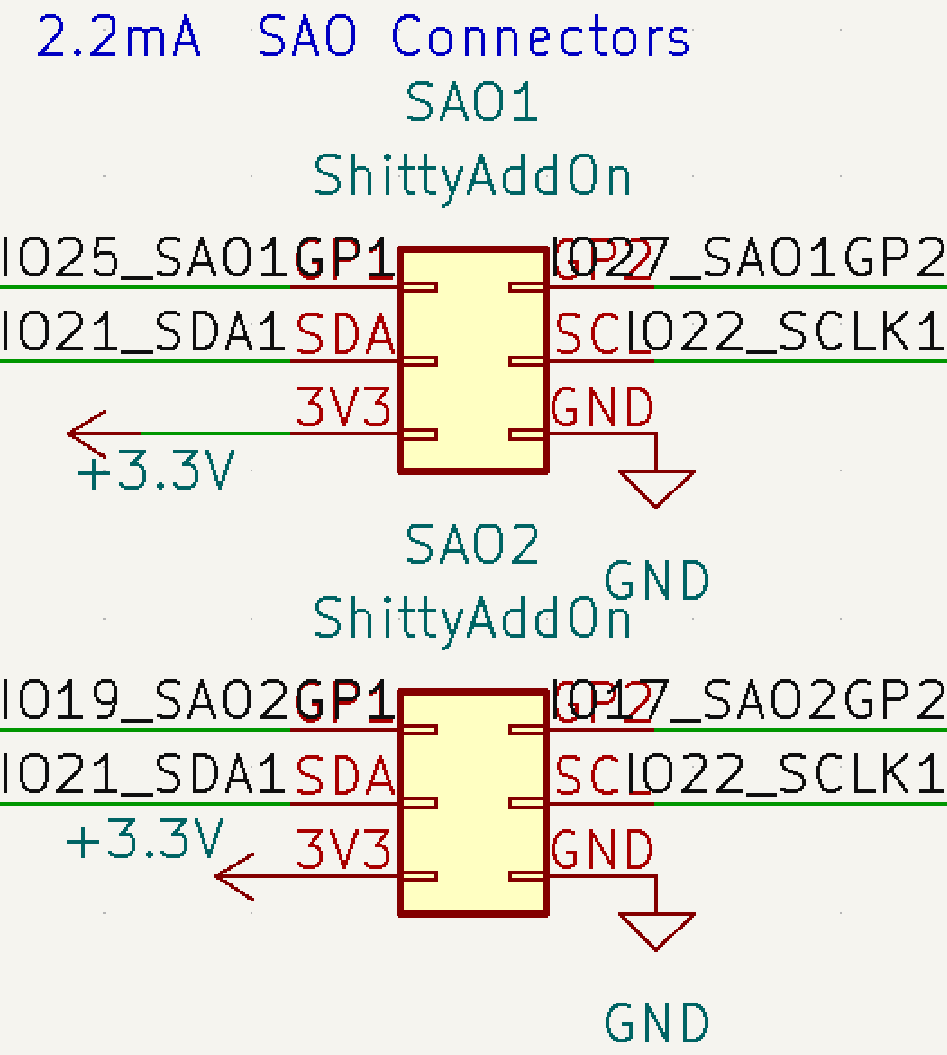
The RT9080-33GJ5 regulator is a super-efficient Low drop-out 3.3v regulator capable of @600mA, which should handle the spikes from the ESP32 at startup, and shitty add-ons with little fuss.

## 5V Boost Converter

As of version 3.4 of the badge, the audio amplifier alone is attached to a 5V Boost Converter to provide better quality audio.

# Shitty Add-On Connectors

These are built to the new Shitty Add-On V92bis standard, complete with both I2c and two GPIO pins. Note that SAO2 shares its GPIO pins with the Infrared transmitter and receiver! It’s unlikely we’ll have any add-ons that take advantage of this, but what’s an over-achiever to do? For this reason, we should have some way for add-on badges to install control code on the ESP32. I don’t even know where to begin with that.

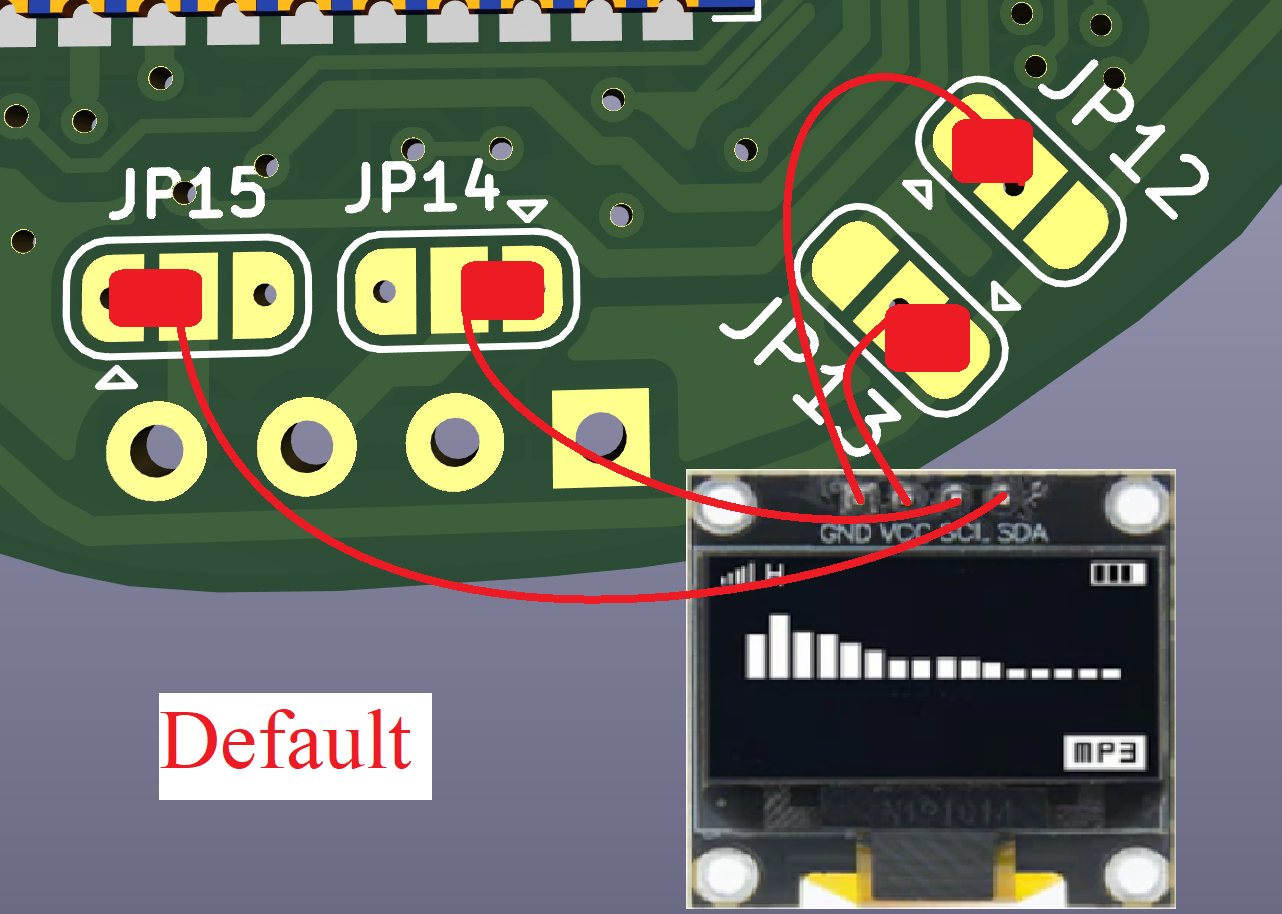


*Figure 6 - Shitty Add-On Connectors*

# 0.96” SSD1306 based OLED Display

This is a 4-pin I2c-controlled black and white SSD1306 display device. It is attached to GPIO23 (SCLK2) and GPIO5 (SDA2) on the second of the two hardware I2c buses on the ESP32.

The device may be pre-installed, or soldered by each user at the conference, depending on logistics. The badge should be coded to operate fine without it, but if present, should display things like radio frequency, IP address on WiFi, Bluetooth paring, menus, games, etc. Whatever we want to do with it—we do it!



*Figure 7 - OLED Pin Swap Jumpers*

Solder Jumpers OLED1-OLED4 allow swapping of Ground/3.3V, or SCLK/SDA pins if necessary, however we have purchased OLED displays that match the defaults so hardware changes shouldn’t be necessary.

## Code to activate the SSD1306 OLED display:

The SSD1306 driver is hard-coded into the firmware and doesn’t need to be uploaded, just imported. Example:

**from** **machine** **import** Pin, I2C  
**import** **ssd1306**i2c = I2C(sda=Pin(5), scl=Pin(23))  
display = ssd1306.SSD1306\_I2C(128, 64, i2c)

display.fill(0) # set everything to black

display.show() # Activate  
display.text('Hushcon-Seattle', 0, 0, 1)  
display.text('2023', 45, 15, 1) *# draw some text at x=0, y=0, color=1*

display.show() # Activate with new parameters

# SI4702 Stereo FM Broadcast Radio Receiver

The FM Receiver chip is found here: <https://datasheet.lcsc.com/lcsc/2202032030_SILICON-LABS-SI4702-C19-GMR_C2155666.pdf> .

* What address on which i2c bus?

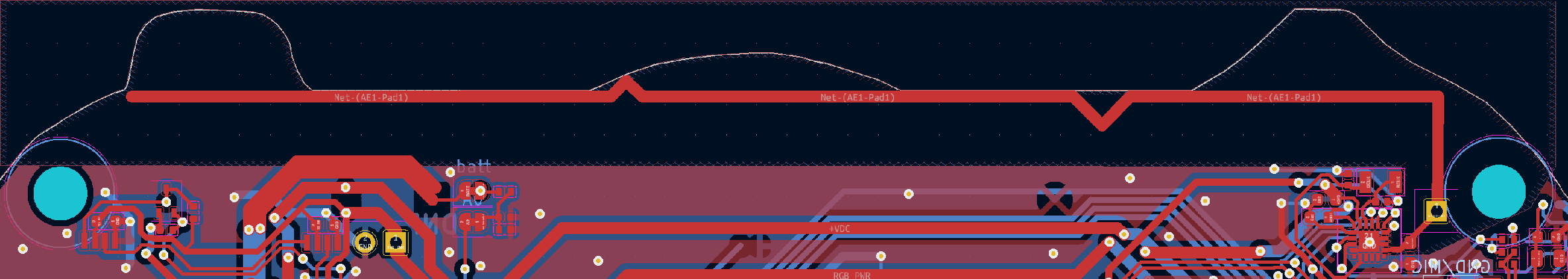
## FM Receive Audio monitoring by ESP32

LOUT and ROUT of the SI4702 chip are also connected to the ESP32 chip at GPIO34 and GPIO35 respectively, through separate 1uF capacitors. Because these GPIO’s are input-only, they cannot be used to output to the amplifier (‘wish I had more pins to do that.) They should however, be capable of monitoring the audio data, perhaps for light shows that listen to FM receive audio.

## FM Broadcast Receive Antennas

We have a few FM receive options here. An antenna for the FM receiver is embedded into the front side of the badge’s printed circuit board on version 2.0 and higher. It’s designed roughly to the vendor’s specs for an onboard antenna, but may not be very effective and can be cut if interference is suspected. The ground/mic wire of the earphone also doubles as an FM Receive antenna. An inductor coil is placed between the FM antenna and ground, and Ferrite beads are placed on the audio inputs and ground of the amplifier to create a high-impedance for RF signals, while allowing an easy pass for audio signals.

there’s also a solder point at the base next to the lanyard hole, that you can solder a 27" thin wire and run it inside or along the lanyard to deliver a decent 1/4 wave whip antenna for 100.5Mhz. http://www.csgnetwork.com/antennagenericfreqlencalc.html



*Figure 8 - FM Receive Antenna embedded in the circuit board*

## I2C bus and the SI4702

The FM receiver chip is controlled by the I2C bus on the ESP32. See I2C Bus under ESP32 section for more details.

## SI4702 software Needs:

1. We’ll need to write some software to control the FM radio (See VERY basic code below to activate the SI470x driver Redbeard wrote).
2. We’ll need some kind of a menu system that allows frequency change—perhaps up and down buttons when in radio “mode” automatically scan up or down for frequencies. It would be nice if this menu system works with the optional i2c add-on, center-bottom of the badge.
3. Recommend using the Mute button as an interrupt to immediately quiet the radio
4. We’ll need to allow a remote signal during the conference to be sent via wifi to all badges, to set their FM receiver to a specific frequency. Talk to Crackerjack for details about this one—his request.
5. Code to initialize the radio, put it on a default frequency, set volume
6. Code allowing FM broadcast radio to be switched to a frequency sent from conference leaders via WiFi.
7. Code to allow buttons to change frequency, scan or search or something.
8. If there’s a way to make the lights blink, glow, or animate based on the music heard on the FM receiver, bonus!

Possible SI4702 libraries: <https://pu2clr.github.io/SI470X/>

## Code to activate and use the SI4702 FM Receiver

Redbeard wrote and documented a detailed library to control the SI4702 called si47x.py. Upload the library first, then import it via your main script.

import si470x

rd=si470x.SI470X()

rd.tuneFreq(90.9) #(Pick a frequency that’s strong in your area)

rd.setVolume(12) #Range is 0-15)

rd.getStatus() #Displays all registers on the SI4702 chip

If the ability to upload libraries is not working, this library can be run by itself remotely and used via repl:

mpremote

radio=SI470X()

radio.tuneFreq(90.9)

radio.setVolume(12)

radio.getStatus()

# LM4808 Headphone Amp & TRRS Jack (V 2.1 and later)

An LM4808 based headphone amplifier (specialized op-amp), powered by a separate 5v boost converter, has been designed to minimize popping, and calculated external parts for better than -5db response from 20hz to 20Khz. This required no more than 470nf input capacitors (max to reduce popping), and also a pair of 330uF output tantalum capacitors on the backside. An on/off switch was added in version 3.4 to allow powering off the amplifier.

## Amplifier Input

By default, Input to the amp comes from the LOUT and ROUT pins of the SI4702 chip. The left channel however, can be hardware-hacked to accept input from GPIO26 of the ESP32 chip. JP3 can be cut and switched to route an audio signal generated at GPIO26 to the left channel of the audio amplifier. (THIS HAS NOT BEEN TESTED as of 7/26/2023.)

A close up of a computer

Description automatically generated

*Figure 9 - Jumper locations for GPIO26 Audio Out Hack*

## Amplifier Output

LOUT and ROUT of the SI4702 chip are also connected to the ESP32 chip at GPIO34 and GPIO35 respectively, through separate 1uF capacitors.

Amplifier audio output is attached to the Left and Right positions on the TRRS headphone jack.

A close up of a computer chip

Description automatically generated

*Figure 10 - Jumper locations for CTIA vs OMTP Standard TRRS Settings*

## Microphone Amplifier (3.5 and later)

An optional Microphone amplifier circuit was added at the very end to better support two-way voice applications on the badge. It’s not enabled by default—to enable it, solder a bridge over JP???.

## 4 position (TRRS) Headphone Jack

Traces under the soldermask default to CTIA style (Apple/Motorola) for Mic & ground, but JP9 and JP10, on the back-side of the badge, can be cut and re-soldered to do OMTP (Samsung/Others) style. See “**Error! Reference source not found.**” for details. Note that this includes MIC support.

Also don’t forget, it’s a Bluetooth device, so if someone writes the code for it, Bluetooth headphones and speakers can be paired as well!

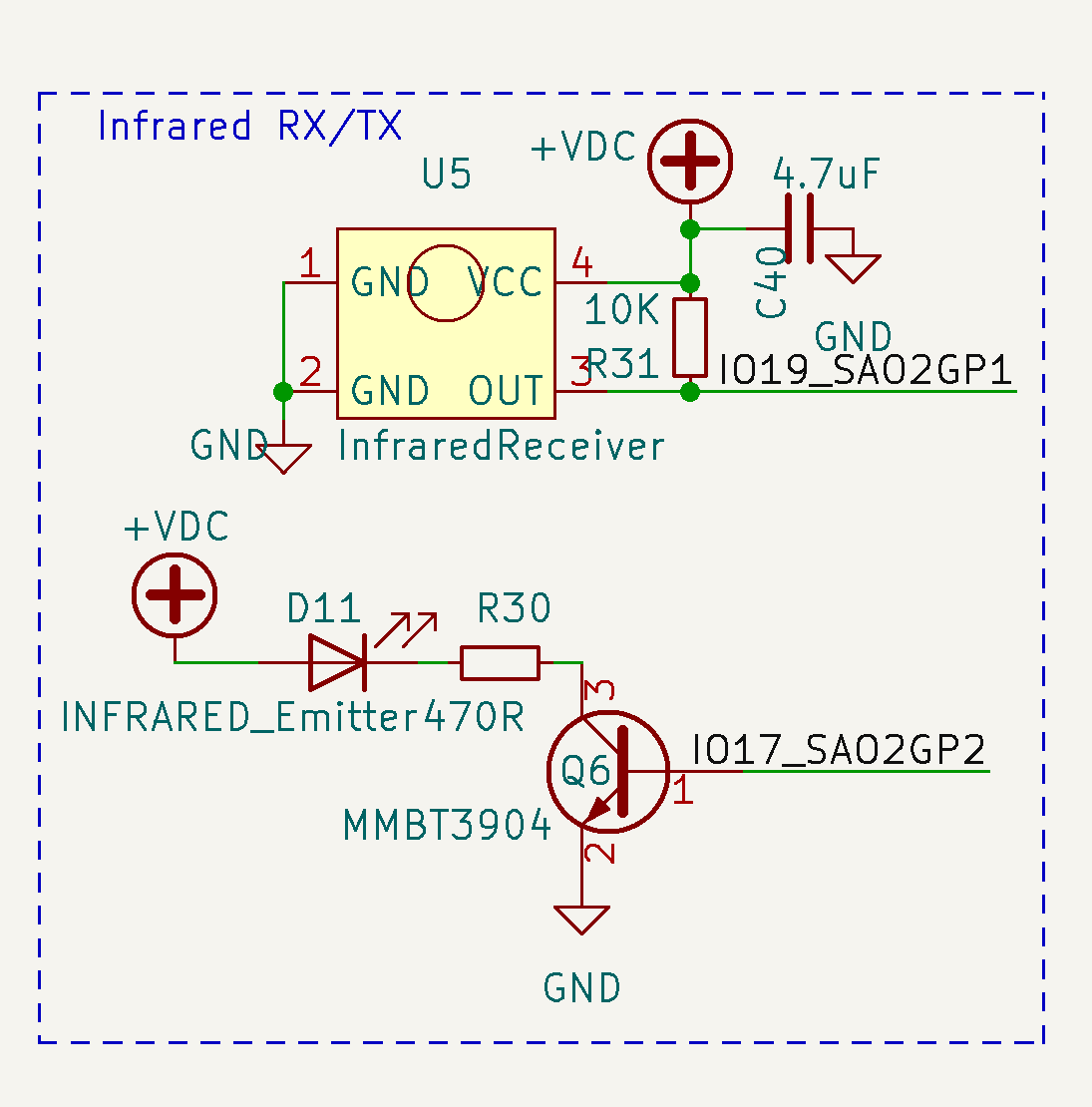
# Infrared Receiver and Emitter

*NOTE: GPIO’s for Infrared are shared with the Shitty Add-On Connector 2, which is on the chin of the monkey on the right with a mask—Speak no evil.*

Beginning with version 2.0, an infrared receiver and emitter have been added. This is intended to be used as a badge interaction program of some kind. Software will need to be written to make use of this capability in a fun and engaging way.

Infrared Receiver line is pulled high with a 10K resistor and is connected to Pin 31 GPIO19 on the ESP32. NOTE—this pin is also shared with the right Shitty Ad-On’s first GPIO pin.

Infrared emitter LED is connected directly to VDC which will range between 4.7v and 3.2v depending on battery charge or if USB is attached. It is current-limited by a 470-ohm resister going into the collector of an NPN transistor, who’s base is controlled by GPIO17 of the ESP32. NOTE – this pin is also shared with the right Shitty Add-On’s second GPIO pin.



*Figure 11 - Infrared Receiver and emitter circuits (Version 2.0+)*

## Code to activate Infrared receiver and emitter

This code has only been tested at a VERY basic level– if you’ve got a better way of doing this, go for it!

#Infrared Transmit & Receiver

print('Experimental code to test the Infrared Transmit/Receive Capability. dunno if this works yet')

# From:

#https://github.com/gamefunc/micropython\_espX\_IR\_Transceiver

#upload driver code to badge:

# ~/monkeys4/mb-v1.20/external/micropython/tools/mpremote/mpremote.py fs cp irGetCMD.py :irGetCMD.py

# ~/monkeys4/mb-v1.20/external/micropython/tools/mpremote/mpremote.py fs cp irSelectCMD.py :irSelectCMD.py

# ~/monkeys4/mb-v1.20/external/micropython/tools/mpremote/mpremote.py fs cp irSendCMD.py :irSendCMD.py

#logIRCMD(time list) -> auto save to /buttomCMD.txt, i am use VS/HX1838B:

from irGetCMD import \*

a = irGetCMD(19) #(xx) is GPIO Pin number. Monkeybadge IR receiver is on GPIO 19.

#get log code line -> json.loads -> get the listObject .

from irSelectCMD import \*

irCMDList = irSelectCMD(0)

# pwmObject

import machine

irLed = machine.Pin(17, machine.Pin.OUT) # 17 is GPIO PIN number

irLedPwmObject = machine.PWM(irLed, freq=38000, duty=0)

# ir send

from irSendCMD import \*

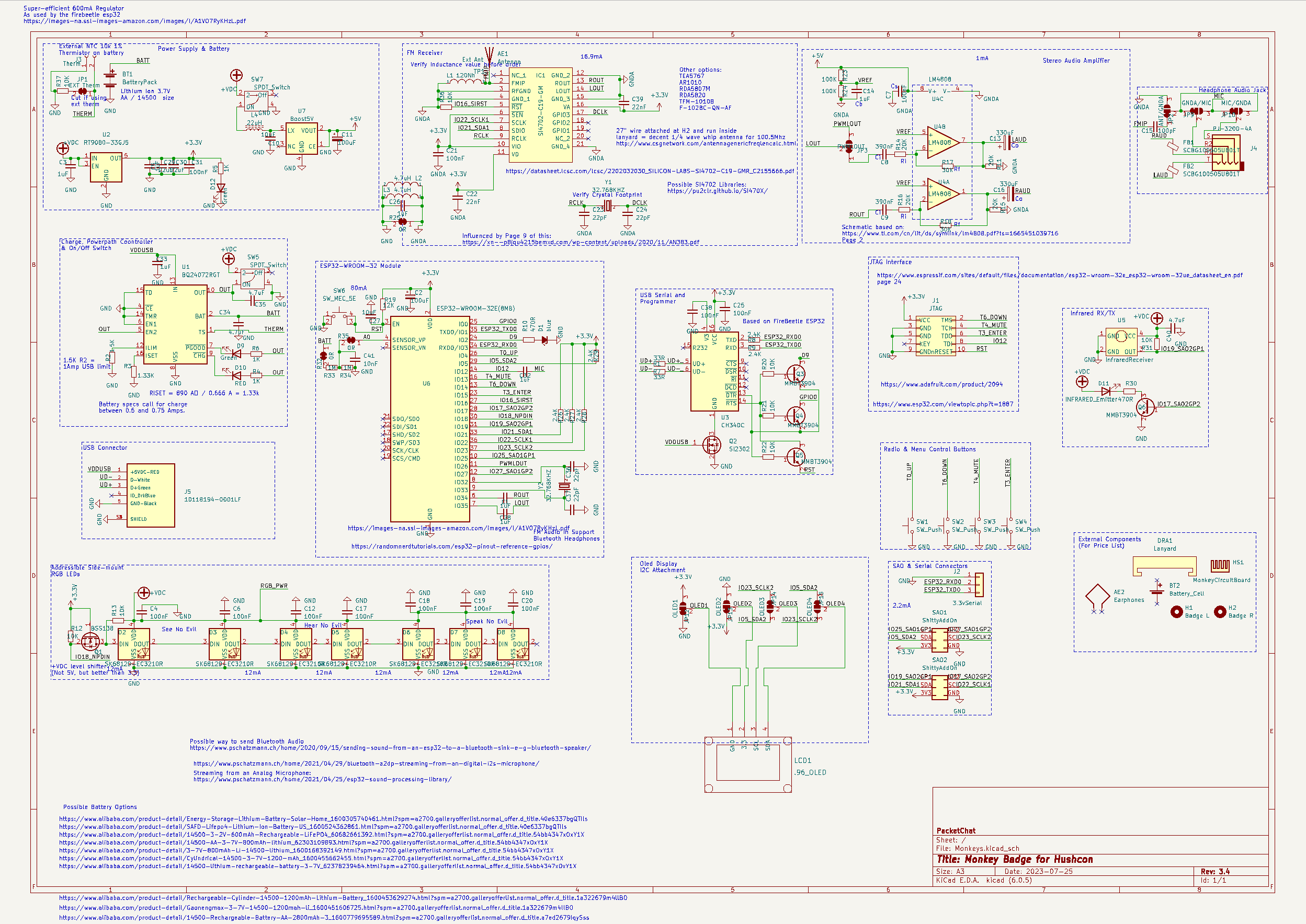
irSendCMD(irLedPwmObject, irCMDList, duty=360) irSendCMD(irLedPwmObject, irCMDList, duty=360)

# Monkeybadge Software

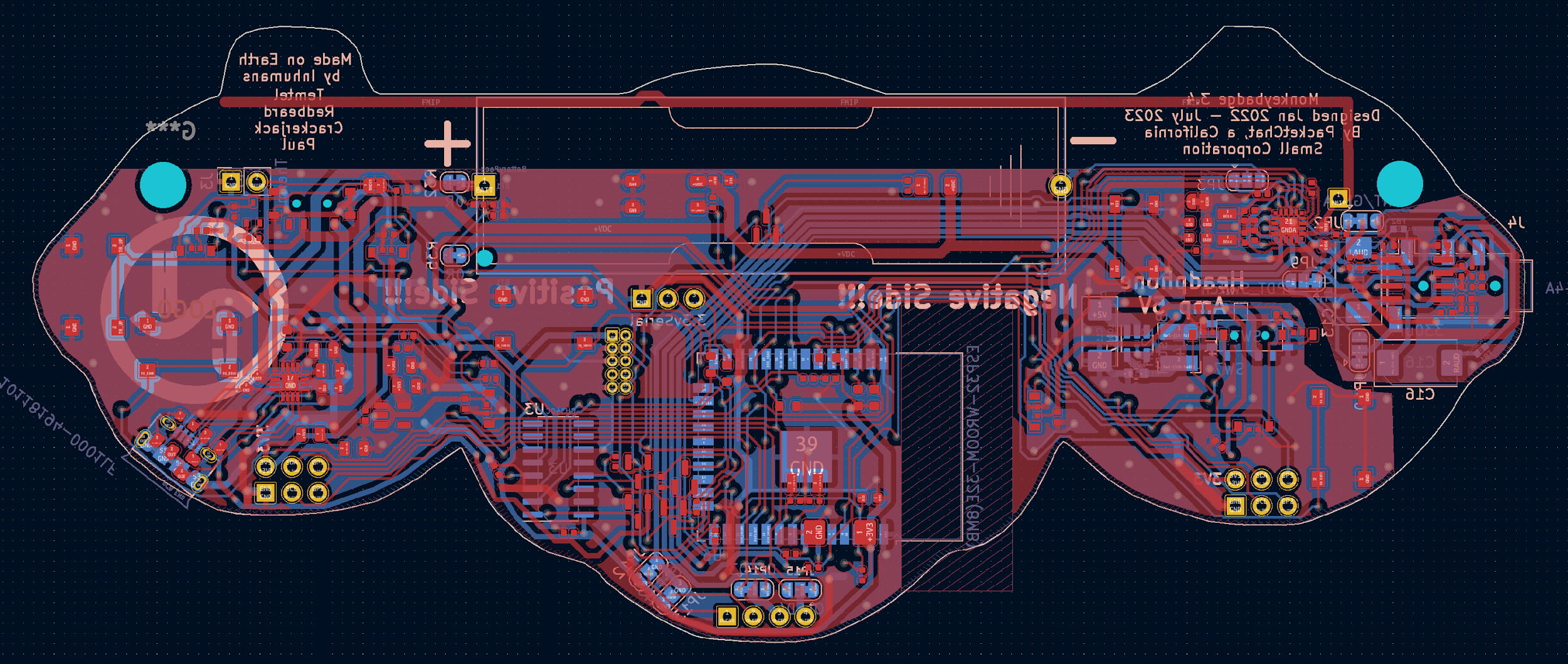
Here’s where we get to showcase all the wonderment and amazing work that is Redbeard’s Monkeybadge additions to BadgePython! We’ll go over how to install the modified and GREATLY improved version of MicroPython onto your IDE workstation. Then we’ll go over uploading the base development image where you can manually control parts of the badge for testing & development purposes.

See [Obtaining a fresh monkeybadge image](#_heading=h.2jxsxqh) for step-by-step instructions to get the latest Redbeard firmware.

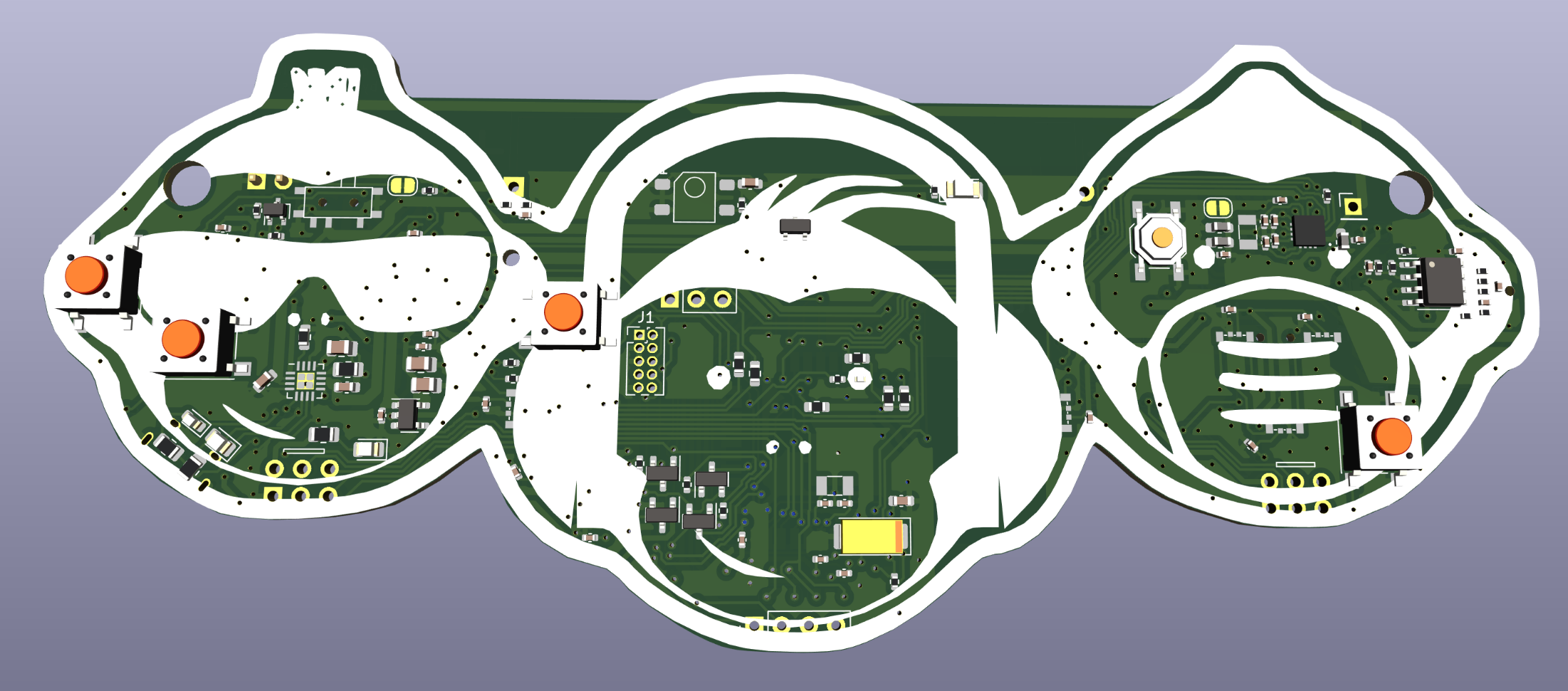
# Monkey Badge version 3.4 Images

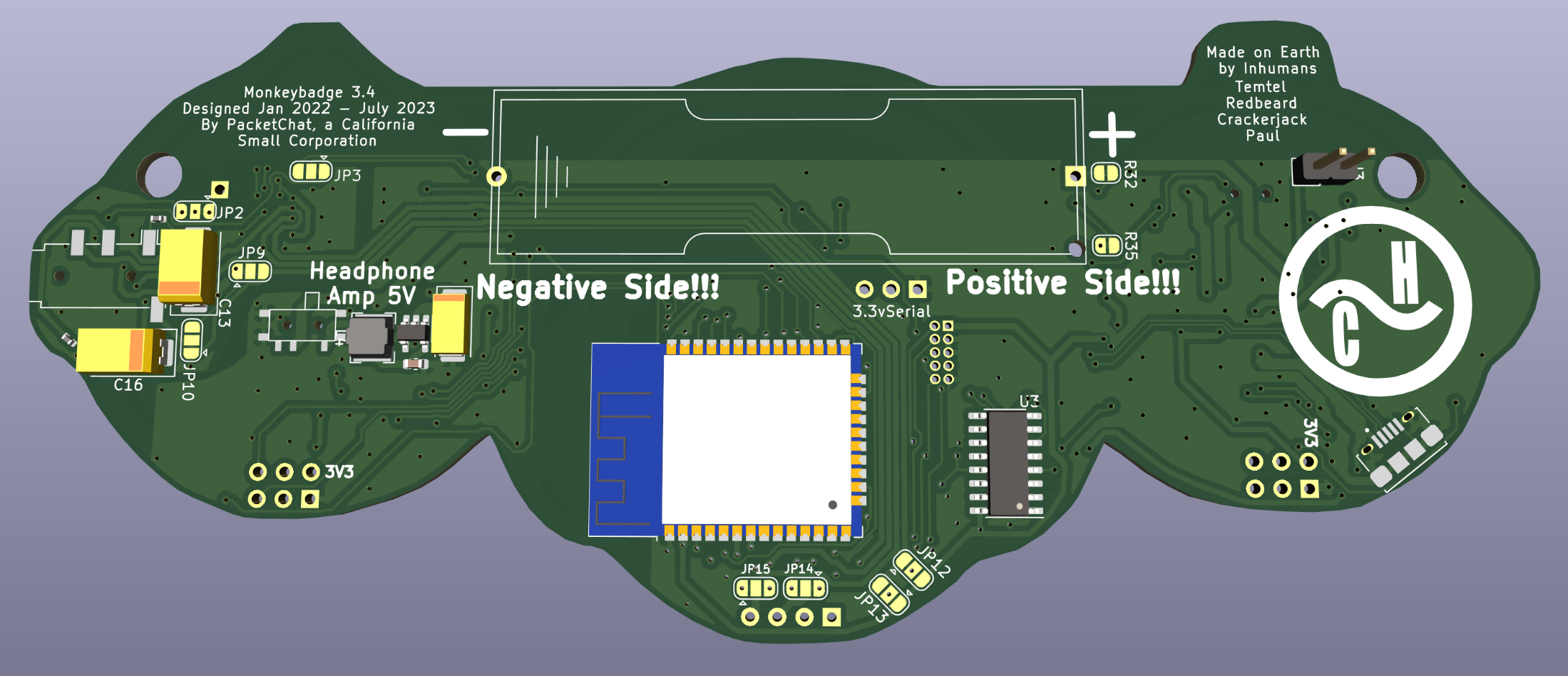


*Figure 12 Schematic*



*Figure 13 - Version 3.4 PCB Traces*





*Figure 14 - Version 3.4 3D rendering*

# V1.0 Errata

1. Pin numbers & footprint wrong on schematic for WS2812B-4020. Fixed in V1.1. No realistic hardware hack to get them to work on the prototype.
2. No control pins to set the ESP32 into programming mode. Fixed in v1.1

# V1.1 Errata

## USB D+ and D- are criss-crossed (RESOLVED)

* 1. To fix on 1.1 prototype, remove R1 and R7 with an air rework station (they’re not really necessary). Criss-cross solder jumpers to the pads.

## Continuous machine noise in the earphone (REDESIGNED)

* 1. Noise appears to be coming from the LMV3558 opamp earphone amplifier hearing the ESP32 idle and process.
  2. Cutting the traces leading to the ESP32 did nothing to quiet the noise.
  3. Cutting power to the amp stops the noise.
  4. Bypassing the amp altogether by using the jumpers is also silent, but, is it actually controlling the radio chip still?
  5. CONCLUSION: amp design was bad. Re-designed using vendor specs.

## No pullup resistors on the I2c Bus (FIXED)

* 1. Fix on prototype by bridging the 3.3V pin with SCK and SCL via 10k resistors. Results? No change. Hmm….
  2. 8/18/22 – Noticed that <https://randomnerdtutorials.com/esp32-i2c-communication-arduino-ide/> says pullups should be 2.4K for 3.3v I2C circuits, so 10K was too high. Test w/ 2.4K resistors instead…
  3. Fix on next prototype? YES.

## Battery holder pins slightly off position(FIXED)

* 1. Fix on next prototype? YES

## RST net was never connected to the EN pin on the ESP32. (RESOLVED)

* 1. This is needed to program the ESP32. The DTR and RTS serial control pins are used by the firebeetle to set pins for programming mode. Since we’re using the fire-beetle ESP32 board file, this pin is very necessary.
  2. Fix on 1.1 prototype by soldering a thin wire from the emitter (pin 2) of Q5 to the ESP32’s castleated pin 3.
  3. Fix on next prototype? YES.

## This may still be related to #5, but Reset short is not working.

* 1. I think it might be the 470 ohm resistor there—don’t think it’s necessary. See #8 below.

## Can’t get the FM Receiver chip working (FIXED in 2.1).

* 1. I2c Device at 0x10 appears. Responds to commands May be a firmware problem. Still troubleshooting.
  2. 8/18/22 – Might be that pullup resistor 10k is too high—see 3B above. Trying 2.4K instead of 10K…
  3. SOLVED: Turns out we forgot to put 3.3v on VA pin 16. (Analog Power In)

## A 470ohm resistor was improperly added between the EN pin and ground. (Resolved)

* 1. On version 1.1 remove R11 and make a solder bridge to short it.
  2. Fixed on next protype? YES.

# V 2.0 Errata

1. SI4702 Chip Analog VCC was not connected—fixed.
2. Internet sources report low audio quality from the opamp. Switched to LM4808, which is designed for headphones.
3. Infrared transmitter & receiver not on same wavelength. Switched transmitter to match receiver.
4. Added 10K pulldown resistor for SI4702 RST pin, which is active low.

# V2.2 Errata

1. Need to replace the infrared LED emitter with one that is front-facing, and doesn’t require hand-soldering.
2. Make a true reset button
3. Replace the audio connector with a thru-hole for strength
4. Tent the vias so white is white
5. Drag a few more parts off of the silk
6. Figure out a better way to do powered-off re-charge

# 

# Table of Figures

[Figure 1 - Side-mount WS2812B RGB LED's 10](#_heading=h.2xcytpi)

[Figure 2 - Control Buttons 11](#_heading=h.3whwml4)

[Figure 3 - Version 2.0 schematic of the ESP32 on the Monkey badge, showing pullups on both I2c Busses. 12](#_heading=h.1pxezwc)

[Figure 4 - OMTP vs CTIA vs TRS Pinouts 15](#_heading=h.41mghml)

[Figure 5 - Battery Monitor Pin on ESP32 16](#_heading=h.2u6wntf)

[Figure 6 - Shitty Add-On Connectors 17](#_heading=h.nmf14n)

[Figure 7 - OLED Pin Swap Jumpers 17](#_heading=h.1mrcu09)

[Figure 8 - FM Receive Antenna embedded in the circuit board 18](#_heading=h.206ipza)

[Figure 9 - Jumper locations for GPIO26 Audio Out Hack 20](#_heading=h.sqyw64)

[Figure 10 - Jumper locations for CTIA vs OMTP Standard TRRS Settings 20](#_heading=h.1rvwp1q)

[Figure 11 - Infrared Receiver and emitter circuits (Version 2.0+) 21](#_heading=h.3q5sasy)

[Figure 12 Schematic 25](#_heading=h.1jlao46)

[Figure 13 - Version 3.4 PCB Traces 25](#_heading=h.43ky6rz)

[Figure 14 - Version 3.4 3D rendering 27](#_heading=h.2iq8gzs)