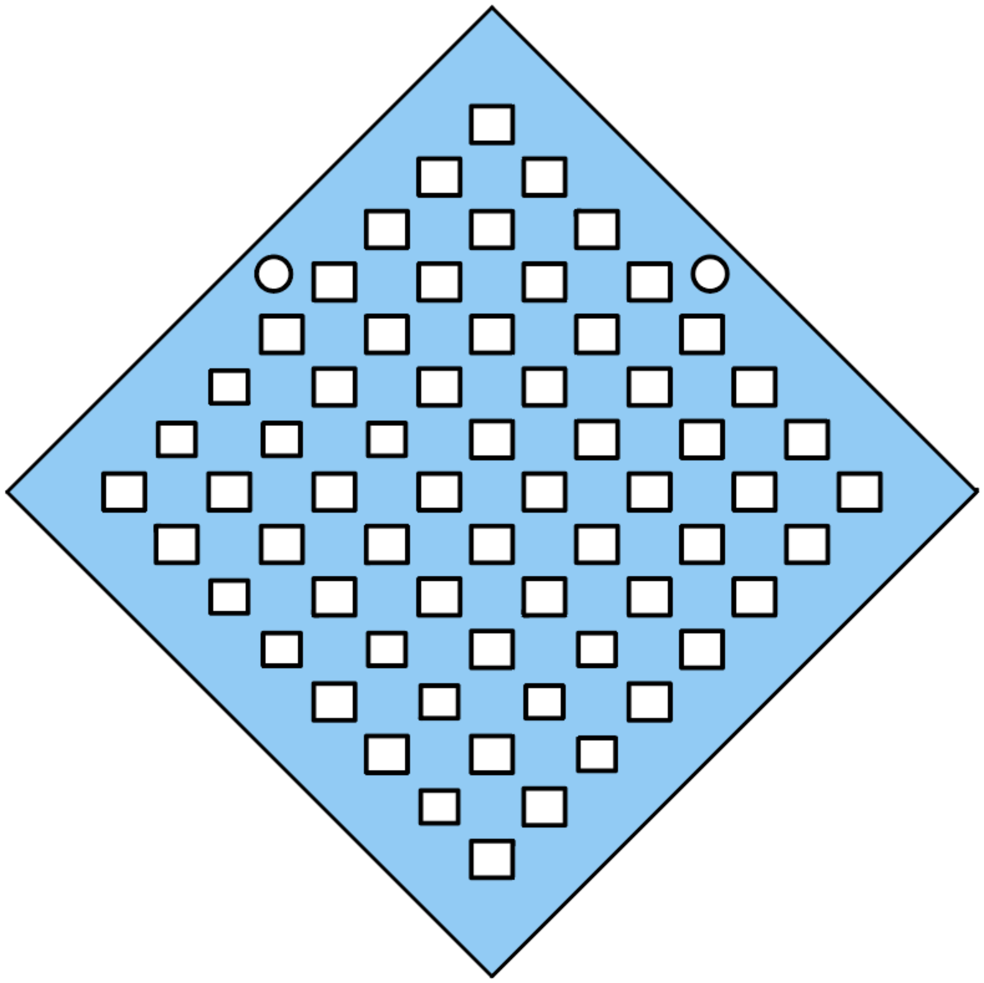
EightByEight Blinky Badge

Users Guide



Programming the ESP8266

Windows only: Install the drivers

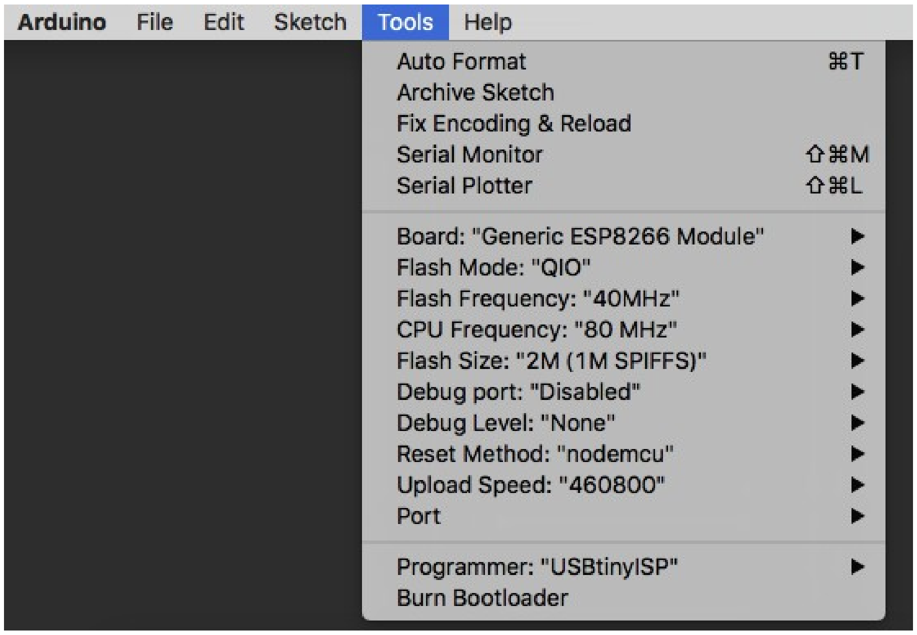
The eightbyeight badge acts USB CDC device, and requires a simple driver.

Getting Started with Arduino

1. Follow the setup instructions for ESP8266 on Arduino:

https://github.com/esp8266/Arduino/blob/master/README.md

2. Use the following settings to connect to the board:

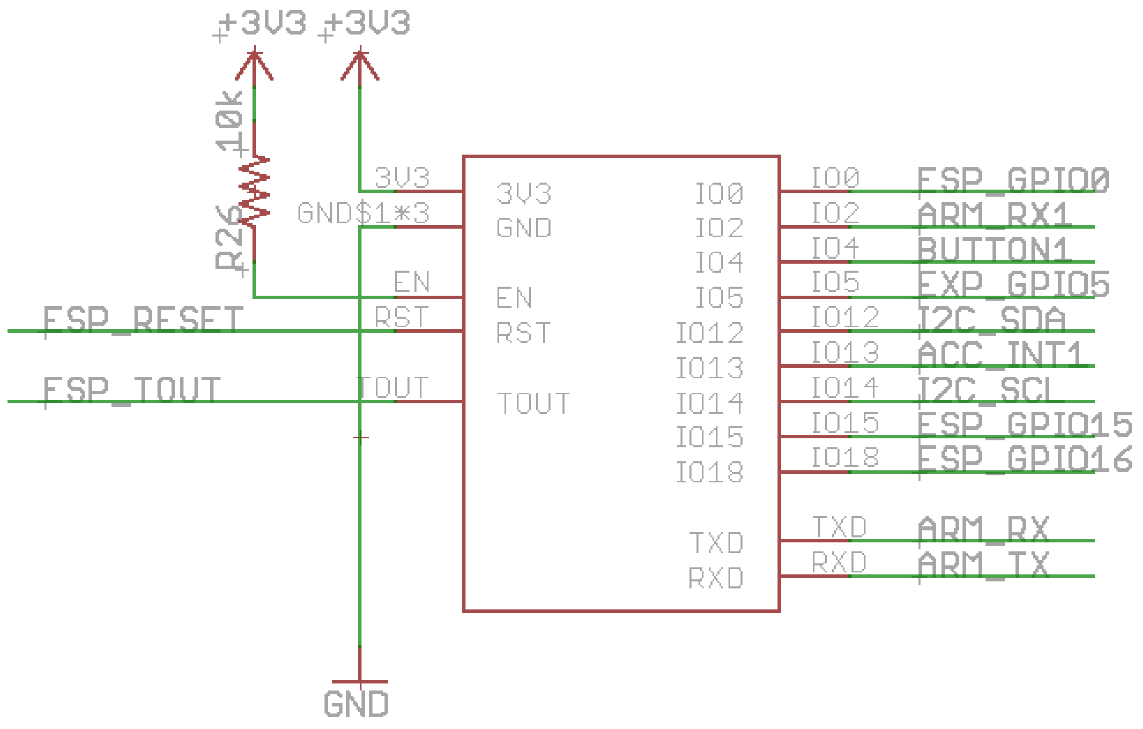


Board: "Generic ESP8266 Module"
Flash Mode: "QIO"
Flash Frequency: "40MHz"
CPU Frequency: "80 MHz"
Flash Size: "2M (1M SPIFFS)"
Debug port: "Disabled"
Debug Level: "None"
Reset Method: "nodemcu"
Upload Speed: "460800"
Programmer: "USBtinyISP"

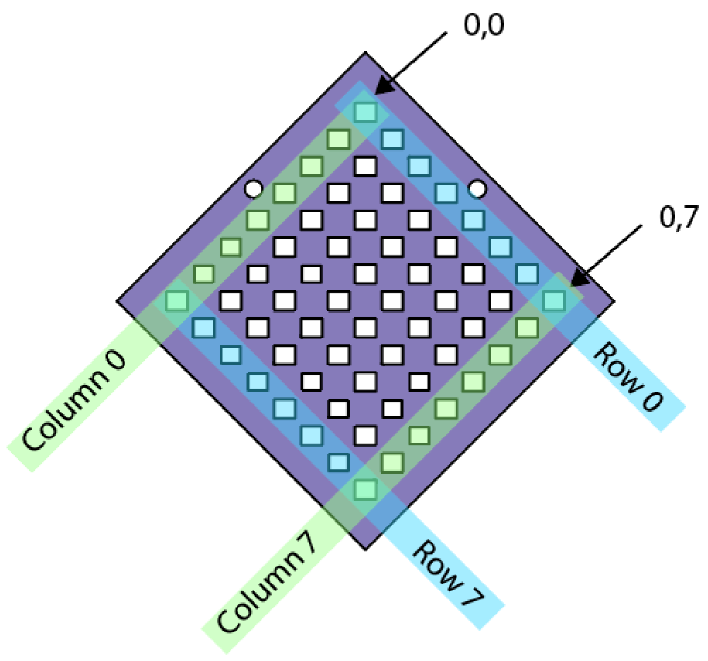
Note that only a baud rate of 460800 is currently supported. The ARM part emulates the 'nodemcu' method for resetting the ESP chip.

3. Load the example sketch and upload it. If all goes well, the LEDs on the badge should start showing a color pattern. You're ready to go!

ESP pin assignments



Controlling the LEDs



TODO: Encapsulate this into a library, expose functions to change display brightness, speed (see matrix_setPixelColor() and matrix_show() functions in the BadgeDemo sketch)

The LEDs are controlled by sending data to the serial1 port on the ESP8266 chip. The ARM chip does the busy work of generating the PWM signals, and supports a simple serial protocol for updating the LEDs.

For each LED, you'll need to send an 8-bit value for the red, green, and blue brightness. The eight bit value can vary from 0 (fully off) to 254 (fully on). There are 64 LEDs, so

Note: Internally, the matrix driver expands this to a brightness corrected, 11-bit space. (TODO: More bits)

First, initialize the port with a baud rate of 230400:

```
Serial1.begin( 230400 );
```

Next, send the data for each LED (this example sets the entire display to red):

```
for(int i = 0; i < 64; i++) {
      Serial1.print(char(255));     // Red
      Serial1.print(char(0));       // Green
      Serial1.print(char(0));       // Blue
}
```

Finally, send a '255' character to tell the matrix to start displaying the new data:

```
Serial1.print(char(255));
```

Reading the button

There is a single button on the badge, that does nothing by default. It is connected to GPIO pin 4 on the ESP8266 wifi chip.

To read the status, first put the pin in input mode:

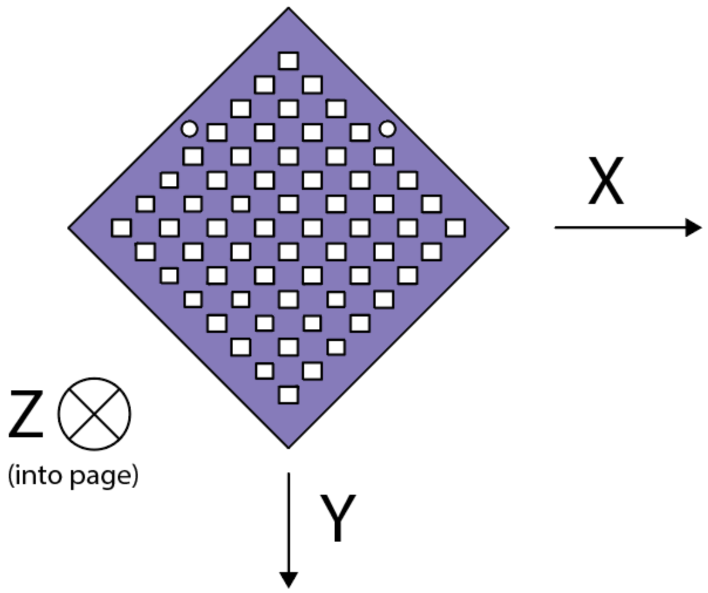
```
const int button = 4;
pinMode(button, INPUT_PULLUP);
```

Then, the button status can be read using the digitalRead() function:

```
bool buttonPressed = !digitalRead(button);
```

Note that the value of the button input is low when the button is pressed, and high when it is not pressed.

Reading accelerometer data



The accelerometer is connected to the ESP using an I2C bus. The ESP doesn’t actually have a hardware I2C peripheral, however the ROM can emulate one. First, set up the Wire library to start the I2C bus:

```
const int i2c_scl = 14;
const int i2c_sda = 12;
Wire.begin(i2c_sda, i2c_scl);
```

after Wire is configured correctly, set up the mma8653 library:

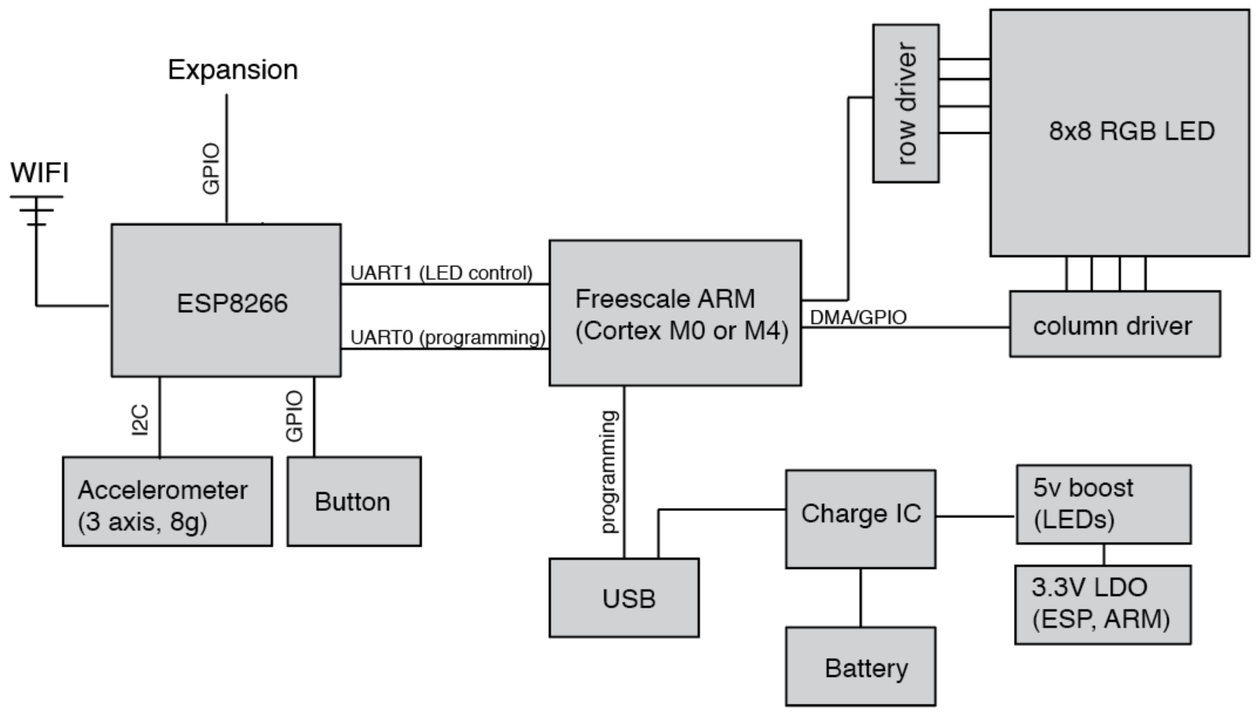
```
mma8653.setup();
```

Once everything is configured correctly, the most current values can be read from the accelerometer:

```
float x, y, z;
mma8653.getXYZ(x, y, z);
```

TODO: Implement interrupt to automatically poll accelerometer data, implement implement accessor functions for individual axis data to avoid references in function call.

System Diagram



Everything fits together like this!

Updating the ARM firmware

The ARM chip acts as a USB-to-Serial converter for the ESP8266 Wifi module, and as a sophisticated LED controller for the 8x8 RGB LED matrix. The firmware should normally not need to be upgraded, however it supports the DFU protocol for loading new firmware.

TODO: Directions for updating firmware (see: https://github.com/Blinkinlabs/BlinkyTile/tree/master/firmware)

TODO: Directions for entering firmware recovery mode using test pin.