Reading	
id	INTEGER
sensor_id	INTEGER
latitude	FLOAT
longitude	FLOAT
datetime	DATETIME
intensity	INTEGER

main.c

```
void createBody(struct reading_t reading, char *outputString){
	cJson to outputString
}

task createReading(){
while(true)
	getQueue()
	uint8_t populateReading()
	releaseSempahore()
	wait(20 seconds)
}


task onConnected()
{
while(true)
	fetchParams_t fetchParams;
	waitQueue()
	createBody(&reading, buffer)
	fetchParams.body = buffer
	fetch("SERVER_IP", &fetchParams);
	wifi_disconnect()
	if (fetchparams->status == 200){
		//Transmission OK! Send all NVMS
		//send all NVMS()'
	}
	else{
		Transmission Not OK
		store reading in NVMS text file
	}
}


void app_main(){
	wifiInit()
	//Will have to check space stack depth
	task(onConnected, priorty = 1)
	task(createReading, priorty = 2)
}
```

server.c

```
on_url_hit("/")
show .txt with all the data stored on the esp32 that has not been sent

RegisterEndPoints{}
```

connect.c

```
wifiInit(){
//starts wifi
}

event_handler{{
//manages IP acquisition
//releases semaphore used in onConnect()
}
```

fetch.c

```
//perform client only needed for GET requests
void clientEventHandler(){

}

createBody(char *number, char *message, char *outputString){
//creates the output string from a &reading
}


void fetch ( char *url, struct fetchParams_t fetchParams){
	//configure client
	//set method, header and body
	//perform client
	//check status code


}
```

reading.c

```
uint8_t populateSensorId(struct *reading_t) {
}

uint8_t readGPS(struct *reading_t) {
}

uint8_t readDateTime(struct *reading_t) {
}


uint8_t readIntensity(struct *reading_t) {
}

uint8 populateReading(struct *reading_t){
	populateSensorId()
	readGPS()
	populateDateTime()
	readIntensity()

}
```

memory.c

```
void printMemory()]
//print memory usage of the task
```

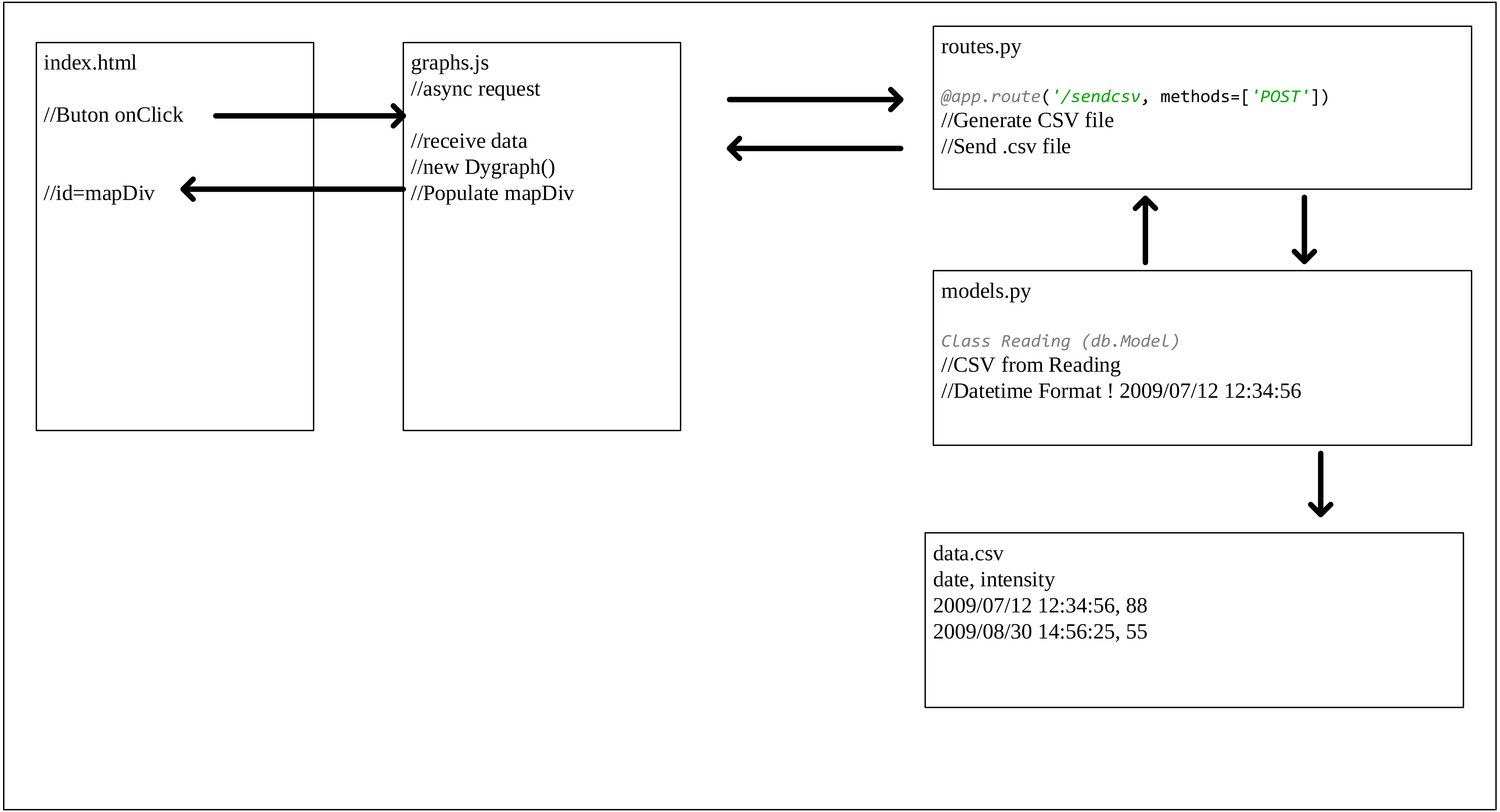
fetch.h

```
typedef struct{
	char *key;
	char *val;
}header_t;

typedef enum{
	GET,
	POST
}httpMethod;

struct fetchParams_t{
   void (*OnGotData) (char *incomingBuffer, char *output);
   char messageGET[500]; //500 byte array, we could do malloc and things like that
   header_t header[3];
   int headerCount;
   httpMethod method;
   char *bodyPOST;
   int status;
};

//Fetch executes post request with all the Fetch params
```

reading.h

```
typedef struct reading_t{
	int sensor_id;
	float latitude;
	float longitude;
	string datetime;
	int intensity;
}

//headers for all the functions go here
```

Dygraphs Data Visualization tool



Available GPIOs ESP32	
Signal Name	ESP32 GPIO
I2C_SDA	15
I2C_SCL	2
I2C_SDA	0
I2C_SCL	4
+ ANY GPIOS !!!	

boad_argon ADC wiring	
5V	5V
SDA ADC	33
SCL ADC	32
GND	GND

Online Course Wiring	
Temp. Sensor	ESP32 GPIO
3V3	3V3
SDA	26
SCL	25
GND	GND

boad_argon GPS wiring	
3.3V	3.3V
SDA ADC	26
SCL ADC	25
GND	GND

ADC2 pins will crash if wifi enabled!

Addresses i2cdetect	
GPS	0x42
12bit ADC	0x48, 0x00
Temp sensor	0x48, 0x00

ADC cant share QWIIC cable

partitions_internet_server.csv

```
# Name,   Type, SubType, Offset,  Size, Flags
# Note: if you have increased the bootloader size, make sure to update the offsets to avoid overlap
nvs,      data, nvs,     ,        0x6000,
phy_init, data, phy,     ,        0x1000,
factory,  app,  factory, ,        1M,
spiffs,  data, spiffs,  ,        1M,
```

CMakeLists.txt

```
spiffs_create_partition_image(spiffs ../html FLASH_IN_PROJECT)
#spiffs is name of the partition on partitions_internet_server.csv
```

server.c

```
spiffs_create_partition_image(spiffs ../html FLASH_IN_PROJECT)
#spiffs is name of the partition on partitions_internet_server.csv

#We determine which file to send through by looking at the req->uri
strcpy(path, "/spiffs%s");
FILE *file = fopen(path,"r");
```

Opens the partition spiffs and send the file through

$idf.py erase_flash !!!!
$idf.py flash monitor clean

$idf.py erase_flash !!!!

We visit index.html 192.168.1.159:8080

index.html

```
...
  <link rel="stylesheet" type="text/css"  href="/style.css">
...
  <section class="main-body">
    <h3>Reading values on Database spiffs/db:</h3>
    <body><span id="database-val">No readings</span></body>
  </section>

  <script src="/script.js"></script>
```

1. on_url_hir gets triggered on server.c with a req->uri = index.html

Next, gets triggered with req->uri is = style.css and req->uri is = /script.js

server.c

```
...
  httpd_uri_t first_end_point_config = {
    .uri = "/*", // We add a * so we handle wildcards
    .method = HTTP_GET,
    .handler = on_url_hit
  };

static esp_err_t on_url_hit(httpd_req_t *req){
 //Open spiffs
//Check for path that was hit
// serve spiffs/index.html, then spiffs/style.css and spiffs/script.js
}
```

3. We register the /* endpoint with a on_url_hit handler

4. We implement the url_hit handler, open spiffs memory and send back index.html, script.js, style.css

spiffs

spiffs (html folder on the project

html
--------- database.html
--------- index.html
--------- script.js
--------- style.css

2. The browser receives index.html and later, all the files called on index.html (script.js and style.css)

Later /script.js and style.css get send back on the same handler

Recevies the script.js and style.css files

1. Calls script.js

script.js

```
...
  async function getDatabase() {
  const result = await fetch("/api/database");
  const database_text = await result.json();
  console.log(database);
  const el = document.getElementById("database-val");
  el.innerText = database_text.database;
 }
 setInterval(getDatabase, 1000);
```

2. Calls a fetch("api/database" function on server.c

server.c

```
...
 httpd_uri_t database_end_point_config = {
     .uri = "/api/database", // using /api is a convention we use
     .method = HTTP_GET,
     .handler = on_get_database
   };
   //We need to register the configuration with our server
   httpd_register_uri_handler(server, &database_end_point_config);
static esp_err_t on_get_database(httpd_req_t *req){

   ESP_LOGI(TAG, "url %s was hit", req->uri);
   char database[100];
   char message[116];
   memset(database, 0, sizeof(database));
   strcpy(database, "DatabaseTextWillGoHere");
   sprintf(message, "{\"database\":\"%s\"}", database);
   httpd_resp_send (req, message, strlen(message));
   return ESP_OK;
}
```

3. We register the /api/database endpoint with a handler

4. We implement the handler and send a response JSON to the script

6. Display it on index.html

5. Receive the JSON

database.txt

```
Index
0  '{\n   "datetime": "2020-04-11 07:06:28.205057",\n   "intensity": 10,\n   "latitude": 19.40417929694277,\n   "longitude": 64.53929710048422,\n   "sensor_id": 2\n}'
1  '{\n   "datetime": "2020-04-11 07:06:28.205057",\n   "intensity": 20,\n   "latitude": 19.40417929694277,\n   "longitude": 64.53929710048422,\n   "sensor_id": 2\n}'
2  '{\n   "datetime": "2020-04-11 07:06:28.205057",\n   "intensity": 30,\n   "latitude": 19.40417929694277,\n   "longitude": 64.53929710048422,\n   "sensor_id": 2\n}'
```

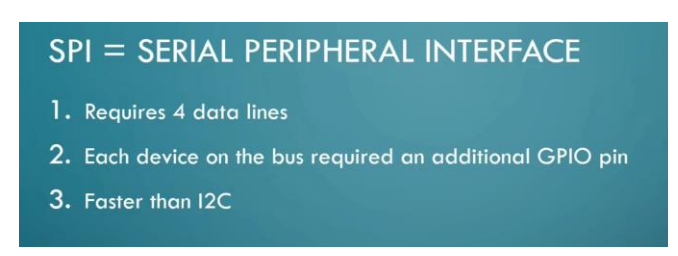
databaseReport()

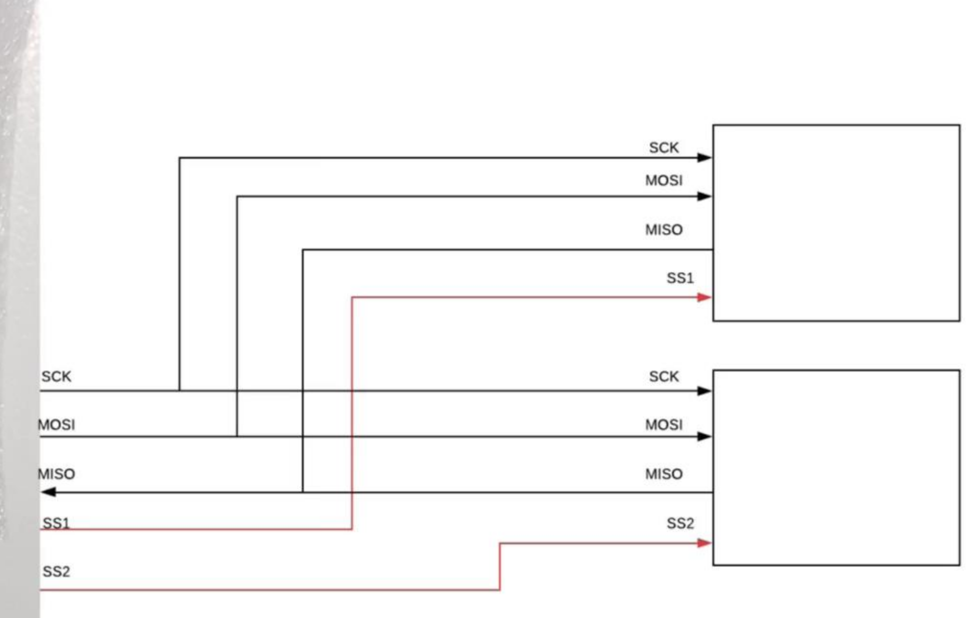
int databaseCountLines()

databaseReadJson(char *destination, int index)

databaseAppendJson(char *destination)

databaseDeleteLine(int index)

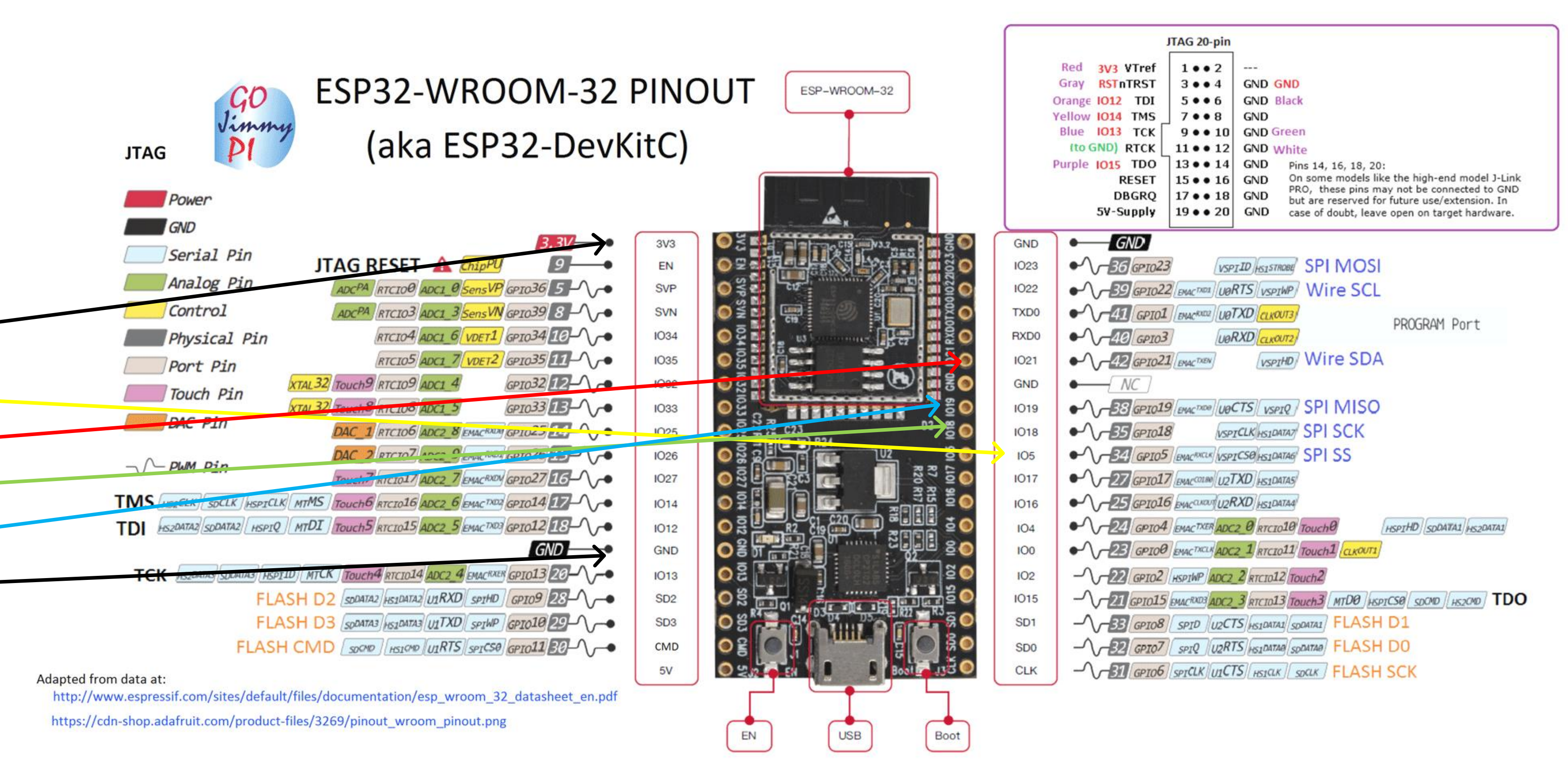


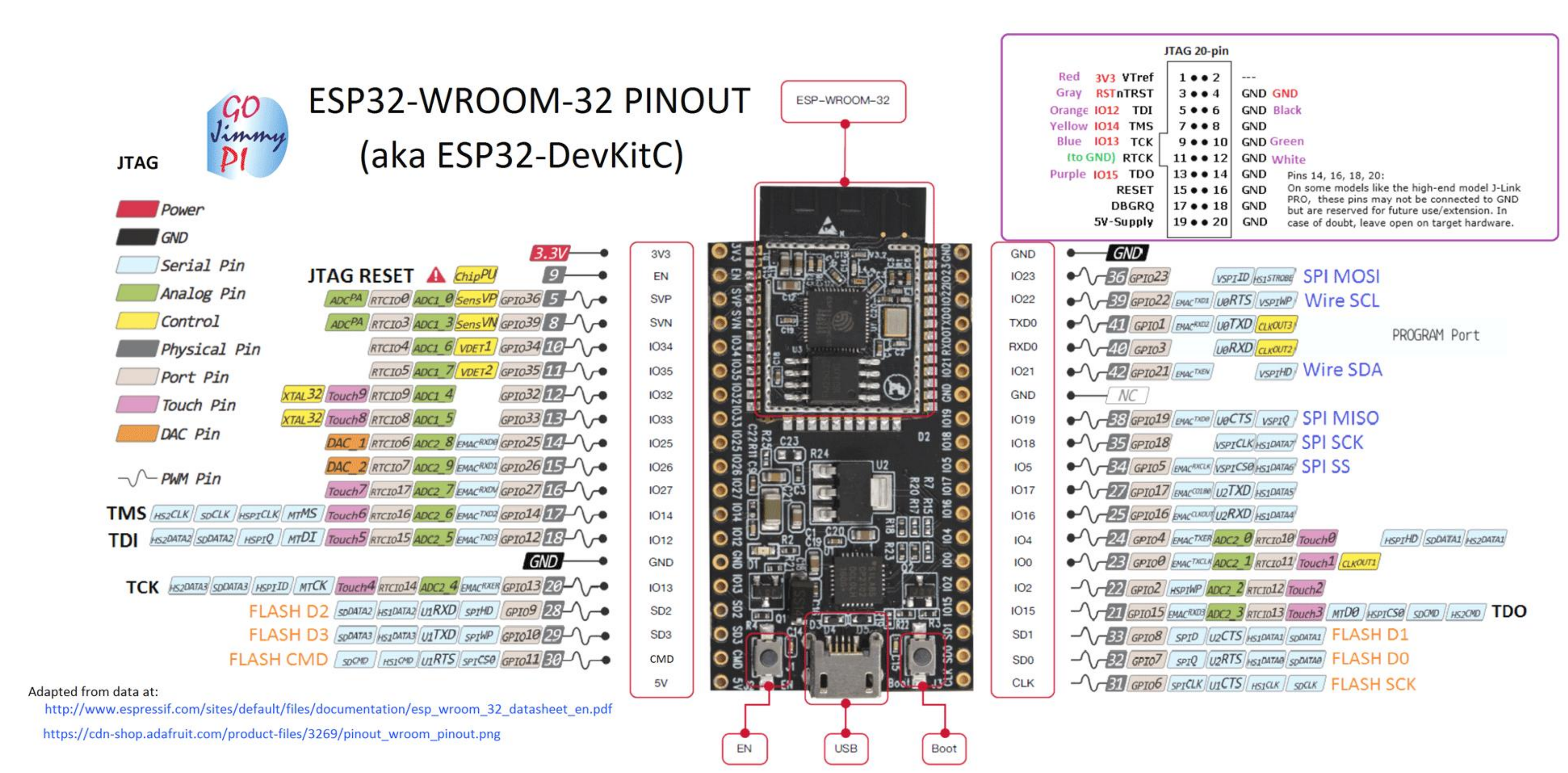


https://darkmattertech.atlassian.net/wiki/spaces/LEAR/pages/326533133/Notes+on+ESP32+GPIOs

boad_argon Wiring	
SSD	ESP32 GPIO
3V3	3V3
CS (Slave Select)	5
MOSI	23
CLK	18
MISO	19
GND	GND

Online Course Wiring	
SSD	ESP32 GPIO
3V3	3V3
CS (Slave Select)	5
MOSI	21
CLK	18
MISO	19
GND	GND



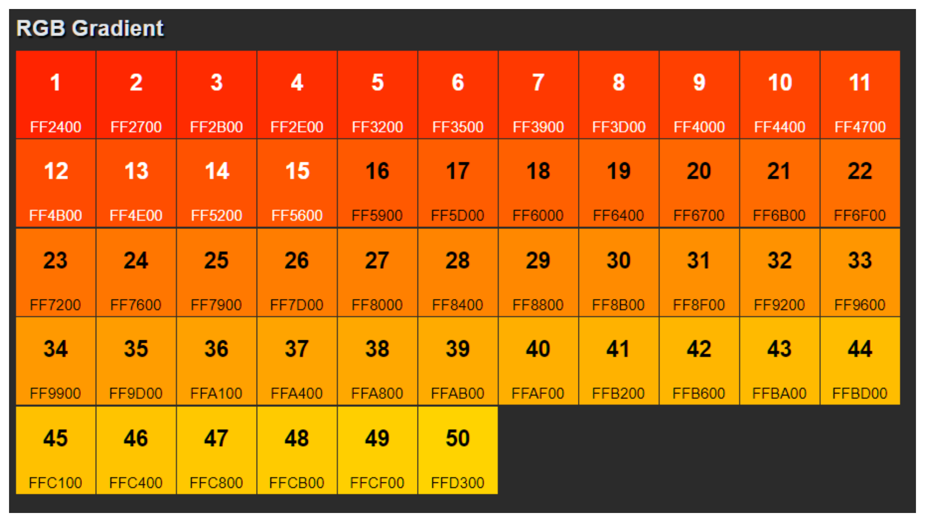


4. Peripherals and Sensors

Interface	Signal	Pin	Function
Parallel QSPI	SPIHD	SD_DATA_2	Supports Standard SPI, Dual SPI, and Quad SPI that can be connected to the external flash and SRAM
	SPIWP	SD_DATA_3	
	SPICS0	SD_CMD	
	SPICLK	SD_CLK	
	SPIQ	SD_DATA_0	
	SPID	SD_DATA_1	
	HSPICLK	MTMS	
	HSPICS0	MTDO	
	HSPIQ	MTDI	
	HSPID	MTCK	
	HSPIHD	GPIO4	
	HSPIWP	GPIO2	
	VSPICLK	GPIO18	
	VSPICS0	GPIO5	
	VSPIQ	GPIO19	
	VSPID	GPIO23	
	VSPIHD	GPIO21	
	VSPIWP	GPIO22	
	EMAC_TX_CLK	GPIO0	

Circles in map

Colors [0:99]



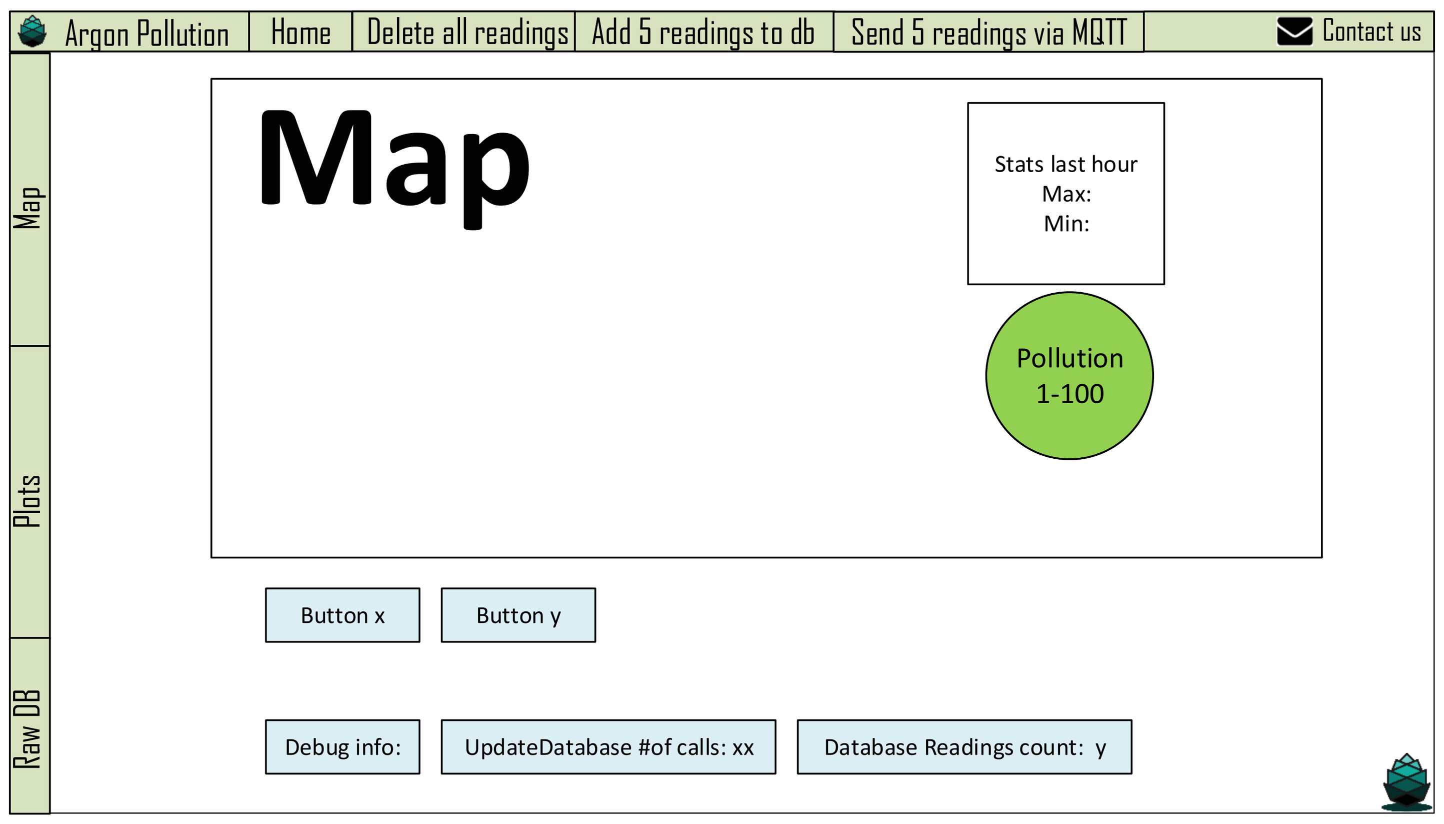
https://graf1x.com/shades-of-green-color-palette-html-hex-rgb-code/
http://www.perbang.dk/rgbgradient/

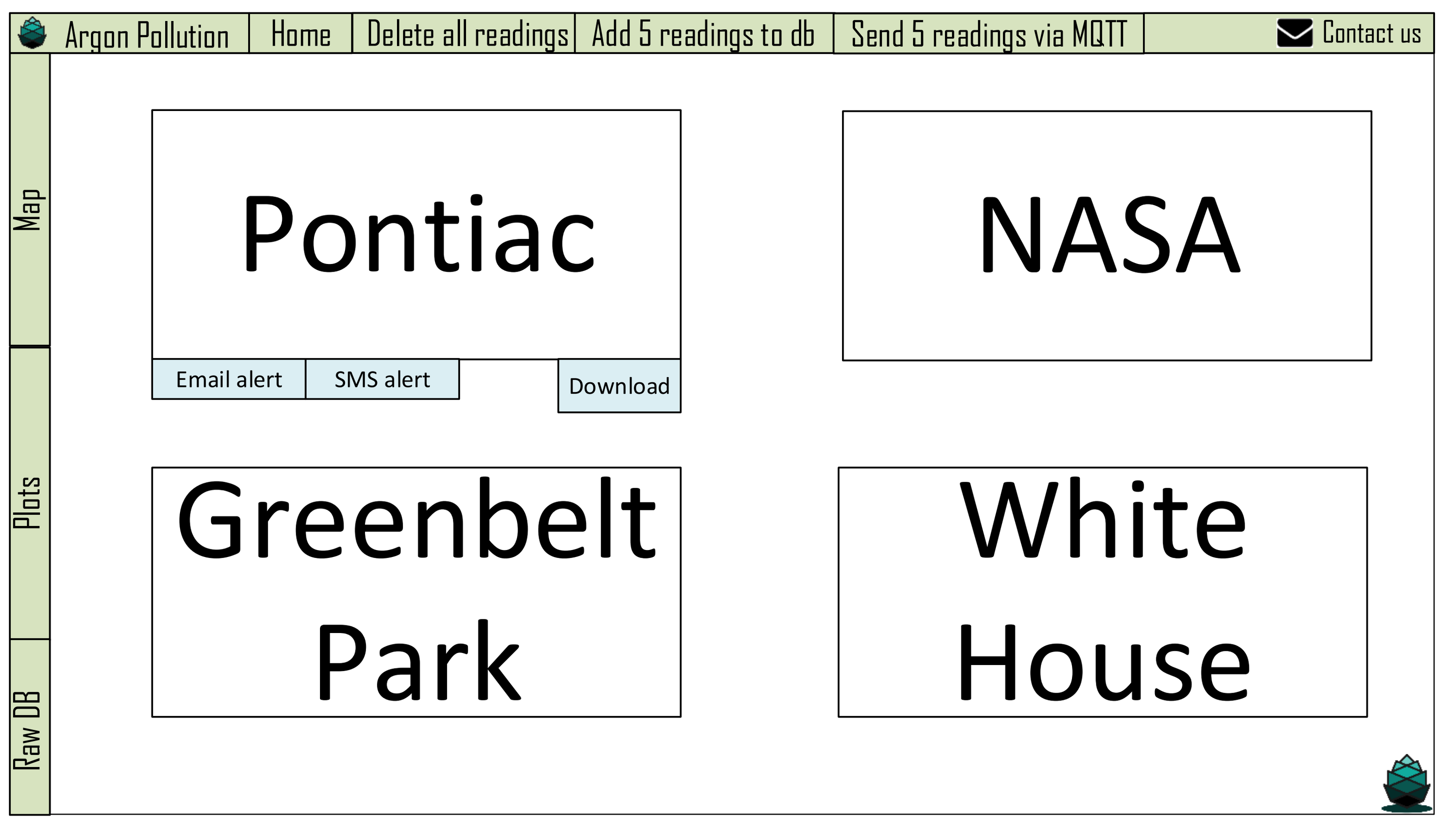
Scarlet — Red FF2400

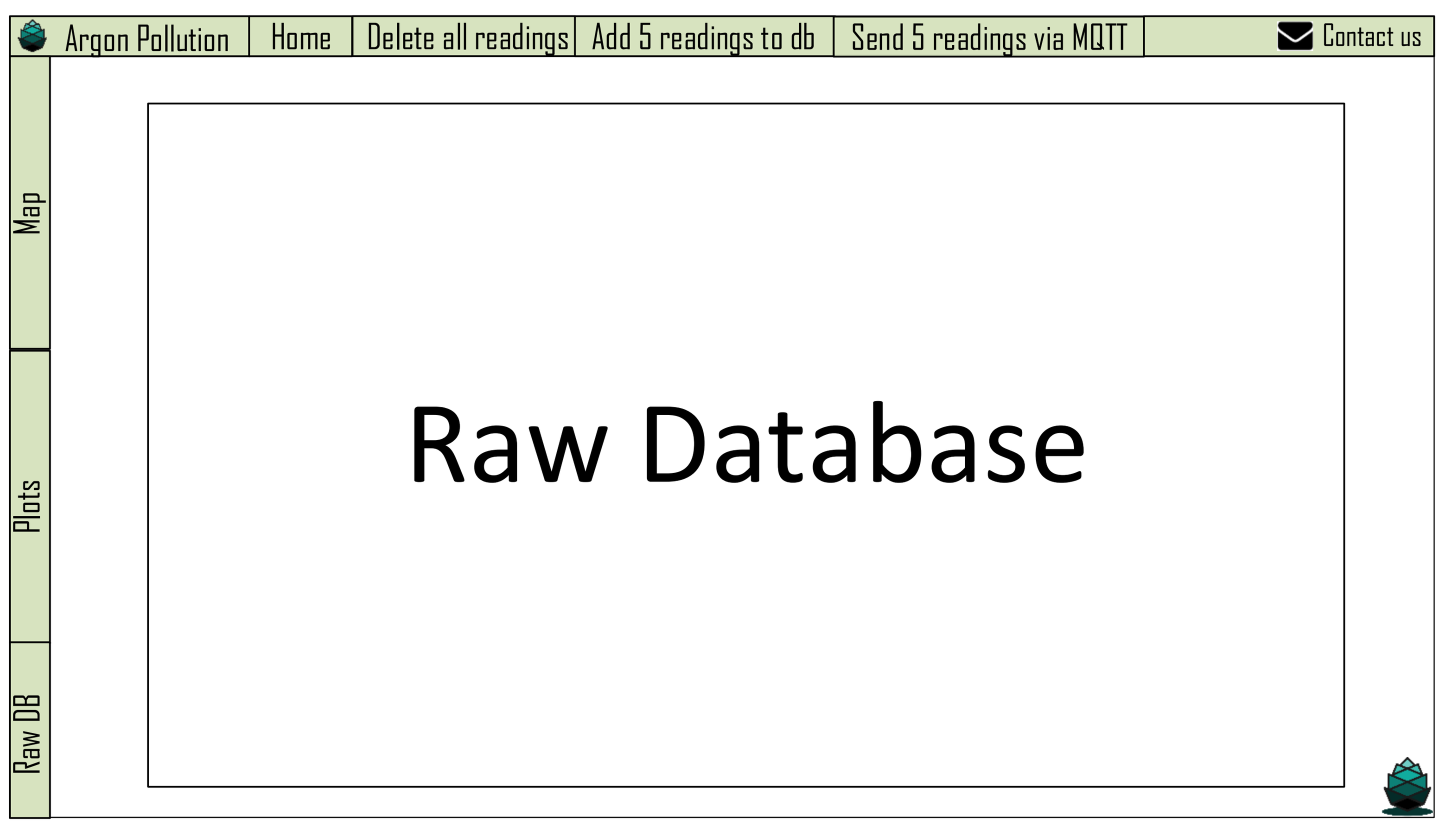
Cyber — Yellow FFD300

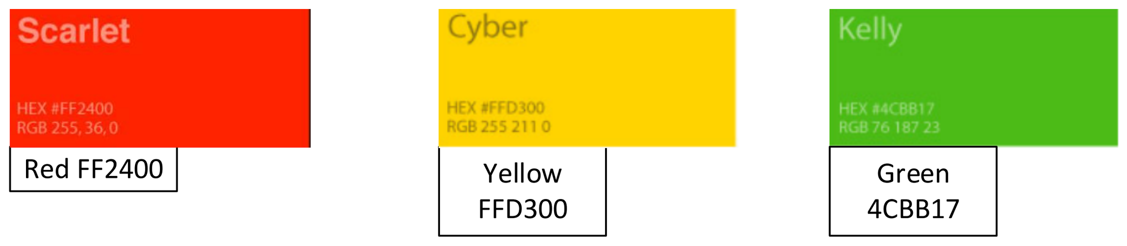
Kelly — Green 4CBB17

Radius [0:100]







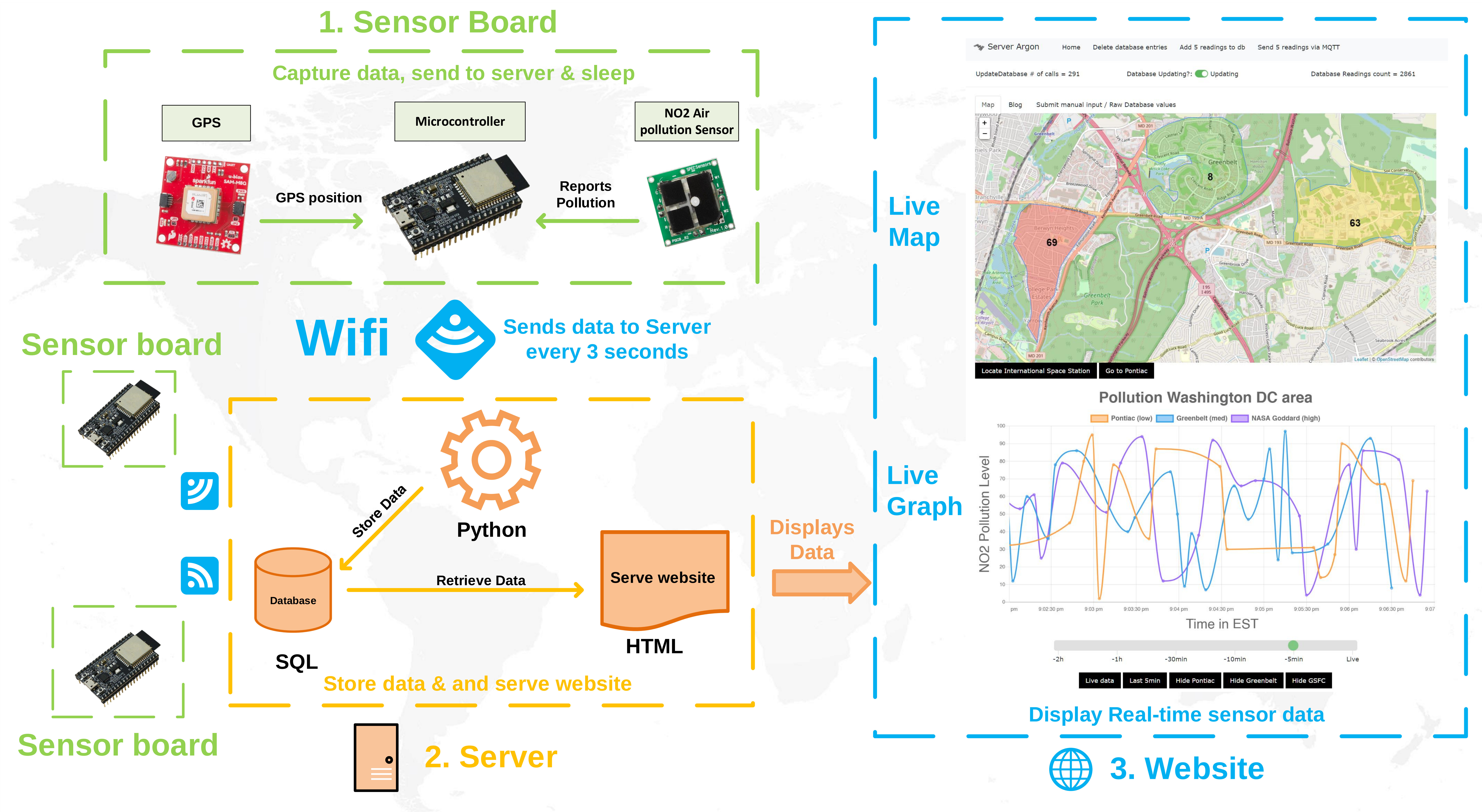


https://graf1x.com/list-of-colors-with-color-names/

DOWNLOAD CHART
https://github.com/NoeMelo/Chart.js/blob/master/images/chart%20in%20html.PNG

chart.js

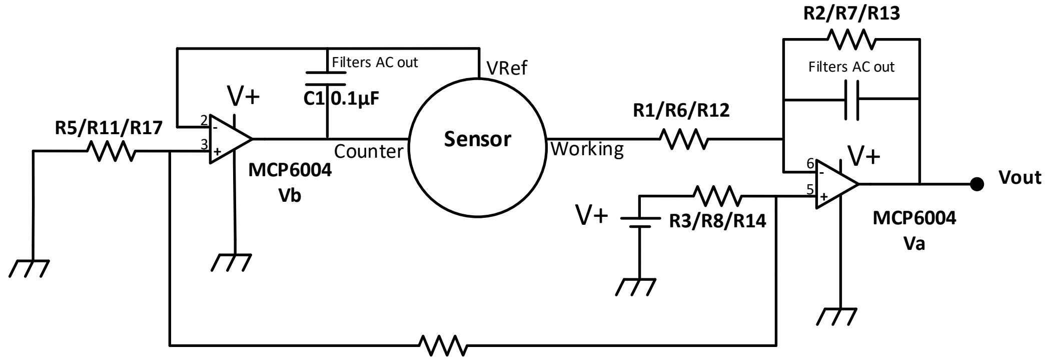
```
var config ={
	...
	options: {
		title:{
			text: ‘Pontiac Street’
		}
	}
		scales:{
			xAxes:{
				realtime:{
					…
					onRefresh: onRefresh
					}
				}
			}
};


window.onload = function() {
	var ctx_pontiac = document.getElementById('chartLive_pontiac').getContext('2d');
	window.chartLive_pontiac = new Chart(ctx_pontiac, config);
};
```



- **Written in 5 different programming languages**
- **3,500 lines of code**

	Language	Lines of code
Board	C	700
Server	Python	450
	HTML	350
	CSS	200
Website	Javascript	1800

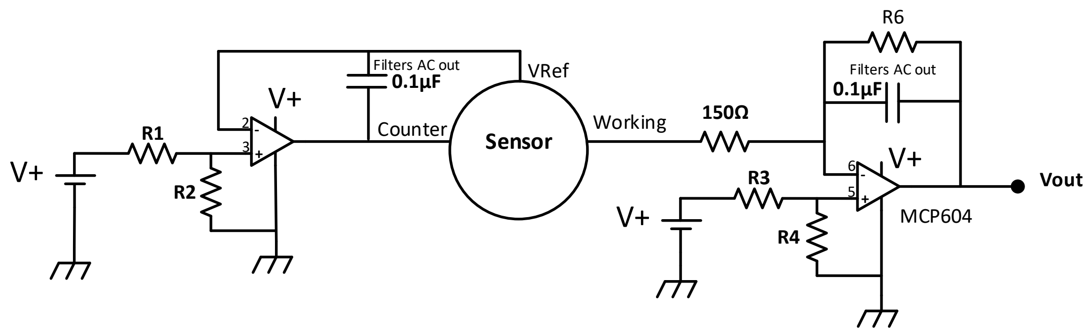
Circuit Proposed by Galvin



0Ω (zero bias circuit)
4.7KΩ (+8mV bias)
12KΩ (-20mV bias)

Comments:
Vbias sensor is = -25mV [from 110-507 datasheet]
C2 Optional, right?
Why the circuits are different?
Looked into LMP91000
MCP604 vs MCP6004 ?

Circuit on the Spec Analog Sensor Developer Kit datasheet

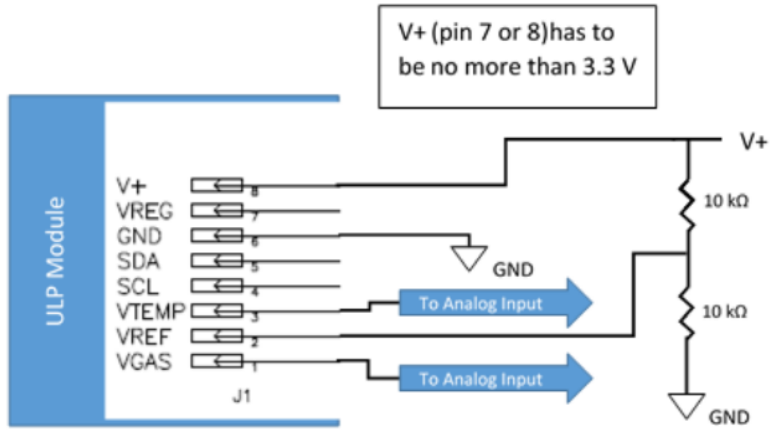


Notes:
- A positive bias for the electromechanical cell is established by setting the voltage at U2, pin 5 with respect to U1, pin 3
- The gain of the transimpedance amplifier is set with R6
- Capacitors C1 and C2 and Resistor R5 can be adjusted to match the characteristics of the electrochemical cell.
- Analog or digital filtering can be implemented to improve signal-to-noise characteristics of the circuit

Coulombs = Voltage * Farads

```
//  Using resistor values from board R1, R2, R3 are for setting _Vref and Bias, while R6 sets the gain
//  If using modified or custom boards set Vref and Gain like this
//long int R1 = 61900, R2 = 1000, R3 = 1000000;
//int bias = 1; //alternatively bias=-1; for negative bias.
//sensor1.setVref(R1, R2, R3, bias); //will set the new Vref for custom sensor voltage ladder. bias is necessary to set the correct arrangement
//sensor1._Gain = 49900; //resistor R6
```

MCP604: Open loop gain 100 dB from http://ww1.microchip.com/downloads/en/DeviceDoc/21314g.pdf



For the video I only showed the output voltages from Vgas, Vref, and Vtemp, but there is an equation that can be used to derive the concentration of the target gas.

$$Cx = \frac{1}{M} \times (Vgas - Vgas_{\circ})$$

Cx = gas concentration in ppm

Vgas = output voltage from Vgas

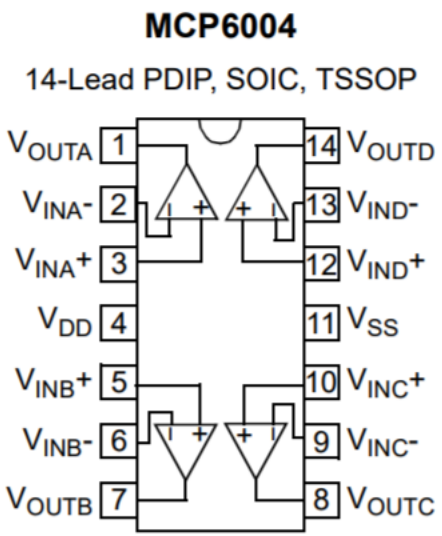
$Vgas_{\circ}$ = output voltage from Vgas in clean-air environment

M = sensor calibration factor – to calculate M use the following:

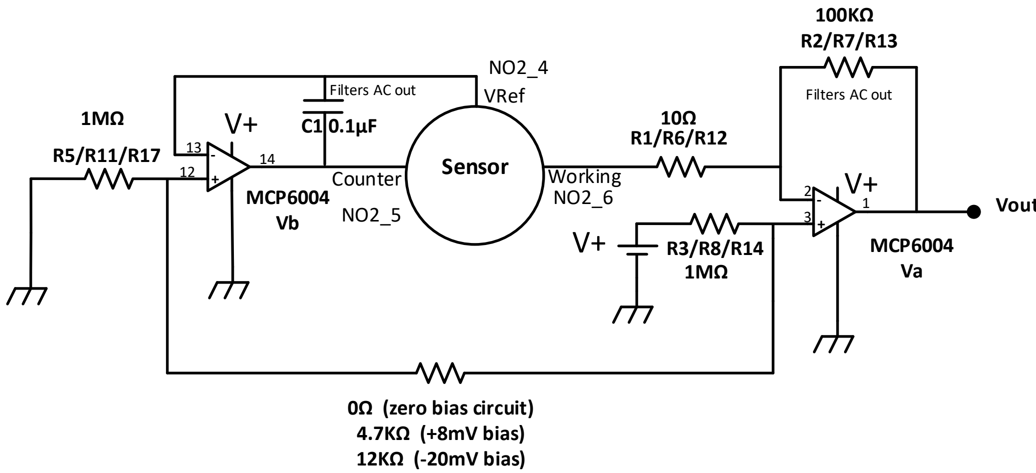
$$M\left(\frac{V}{ppm}\right) = SensitivityCode\left(\frac{nA}{ppm}\right) \times TIA\left(\frac{kV}{A}\right) \times 10^{-9}\left(\frac{A}{nA}\right) \times 10^{3}\left(\frac{V}{kV}\right)$$

Sensitivity Code = provided on sensor label (see below)

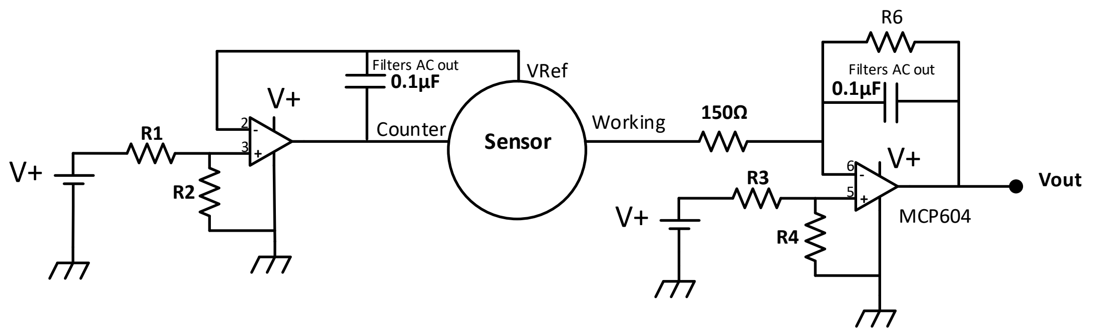
TIA Gain = gain of the trans-impedance amplifier stage of ULPSM circuit

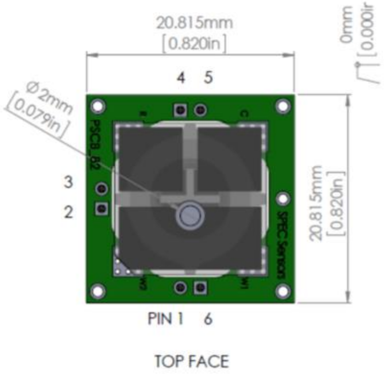


Circuit Proposed by Galvin

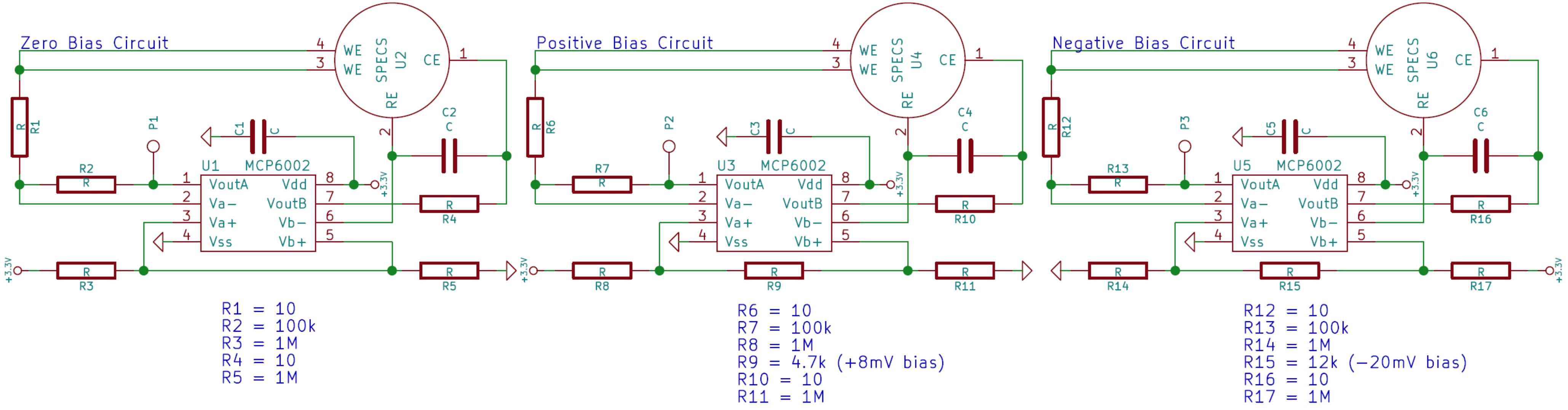


Circuit on the Spec Analog Sensor Developer Kit datasheet



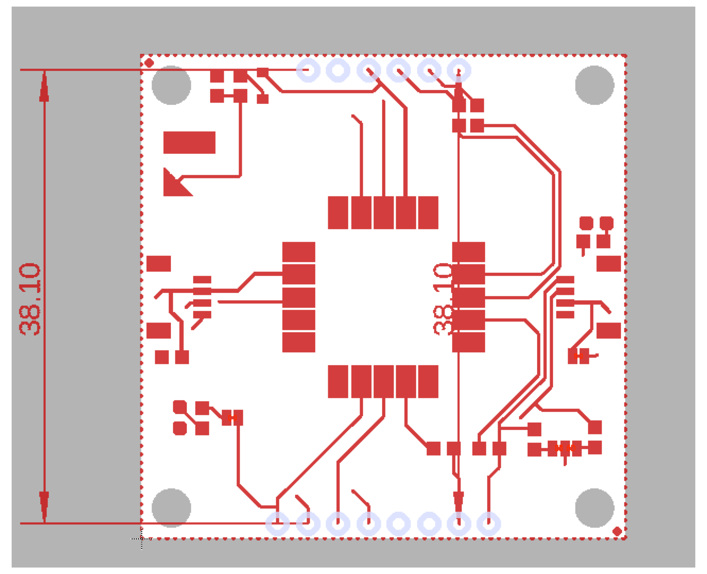


PIN	CONNECTION
1	WORKING
2	NC
3	NC
4	REFERENCE
5	COUNTER
6	WORKING



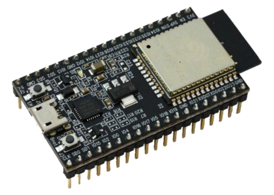
GPS Dimensions

1.5" distance holes

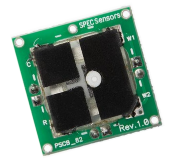


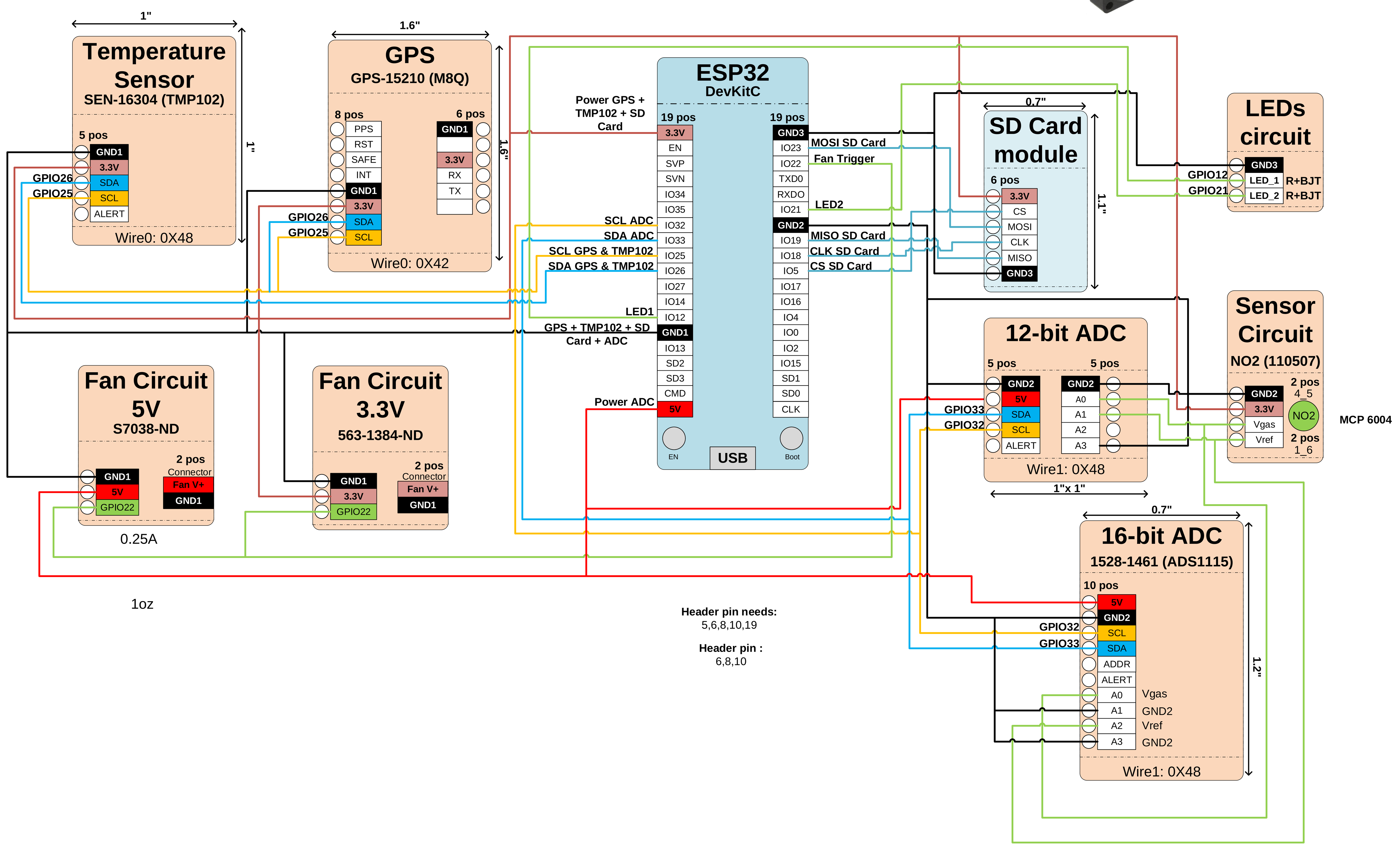
boad_argon ADC wiring	
5V	5V
SDA ADC	33
SCL ADC	32
GND	GND

boad_argon GPS wiring	
3.3V	3.3V
SDA ADC	26
SCL ADC	25
GND	GND



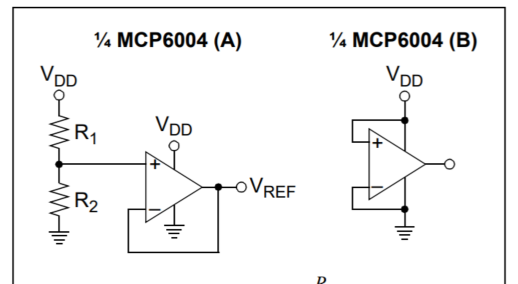
Limit Switches





4.5 Unused Op Amps

An unused op amp in a quad package (MCP6004) should be configured as shown in Figure 4-5. These circuits prevent the output from toggling and causing crosstalk. Circuit A sets the op amp at its minimum noise gain. The resistor divider produces any desired reference voltage within the output voltage range of the op amp; the op amp buffers that reference voltage. Circuit B uses the minimum number of components and operates as a comparator, but it may draw more current.



$$V_{REF} = V_{DD} \bullet \frac{R_2}{R_1 + R_2}$$

FIGURE 4-5: *Unused Op Amps.*