

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(){
    wifilnit()
    //Will have to check space stack depth
    task(onConnected, priority = 1)
    task(createReading, priority = 2)
}
```

```
server.c

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

RegisterEndPoints{}
```

```
connect.c

wifilnit(){
    //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

index.html

//Buton onClick

//id=mapDiv

graphs.js

//async request

//receive data

//new Dygraph()

//Populate mapDiv

routes.py

```
@app.route('/sendcsv', methods=['POST'])  
//Generate CSV file  
//Send .csv file
```

models.py

```
Class Reading (db.Model)  
//CSV from Reading  
//Datetime Format ! 2009/07/12 12:34:56
```

data.csv

```
date, intensity  
2009/07/12 12:34:56, 88  
2009/08/30 14:56:25, 55
```

Available GPIOs ESP32	
Signal Name	ESP32 GPIO
I2C_SDA	15
I2C_SCL	2
I2C_SDA	0
I2C_SCL	4
<b>+ ANY GPIOs !!!</b>	

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

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

Receives the script.js and style.css files

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

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

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);
```

6. Display it on index.html

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

5. Receive the JSON

```
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
}

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 /\* 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

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

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

spiffs

spiffs (html folder on the project

html

----- database.html

----- index.html

----- script.js

----- style.css

Index

0

1

2

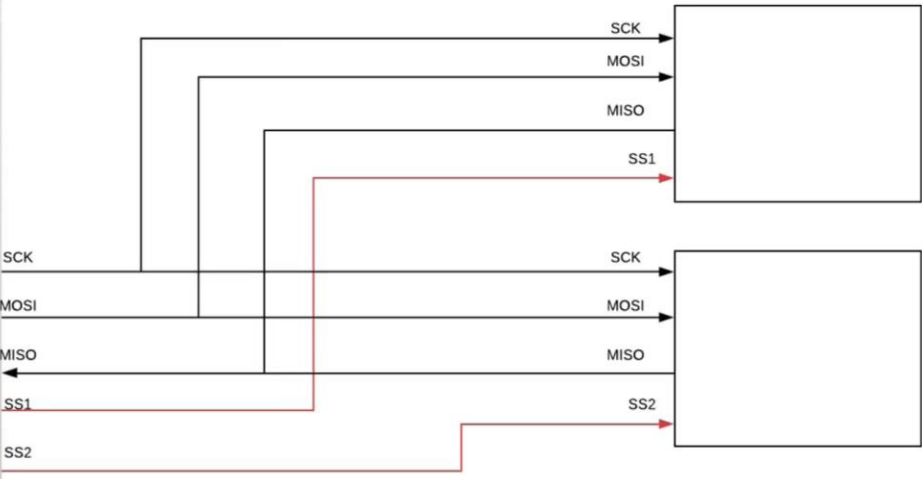
database.txt

databaseReport()      int databaseCountLines()  
databaseAppendJson(char \*destination)      databaseReadJson(char \*destination, int index)  
databaseDeleteLine(int index)



SPI = SERIAL PERIPHERAL INTERFACE

1. Requires 4 data lines
2. Each device on the bus required an additional GPIO pin
3. Faster than I2C



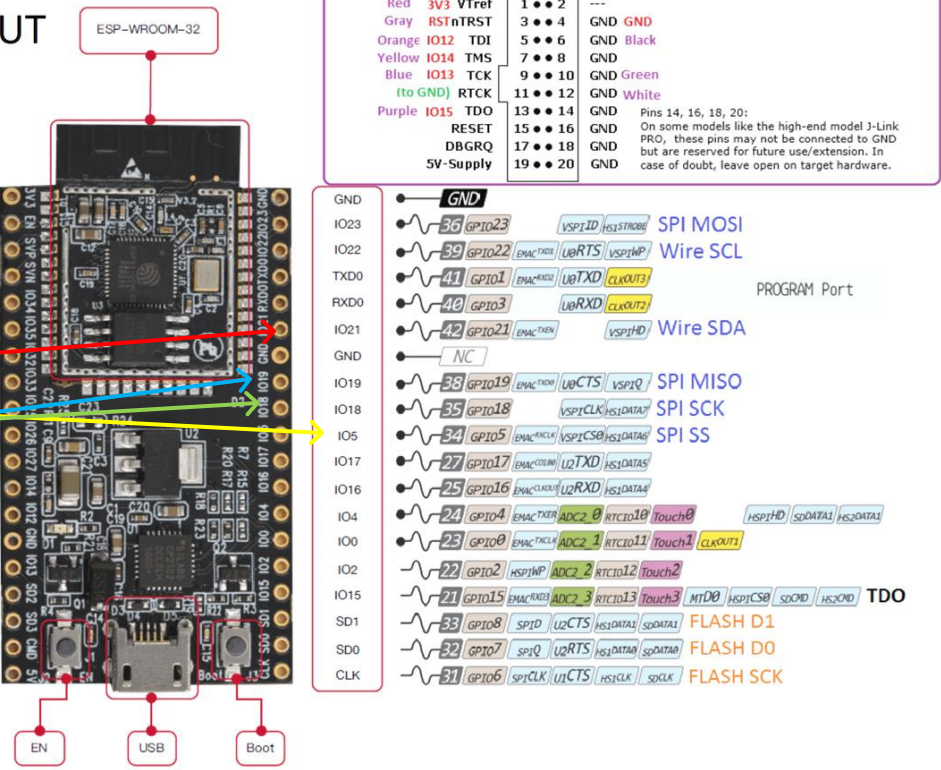
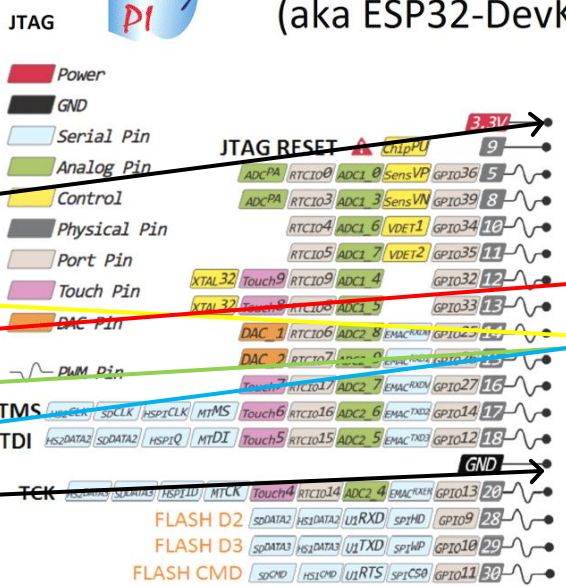
<https://darkmattertech.atlassian.net/wiki/spaces/LEAR/pages/326533133/Notes+on+ESP32+GPIOs>

board_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



ESP32-WROOM-32 PINOUT  
(aka ESP32-DevKitC)

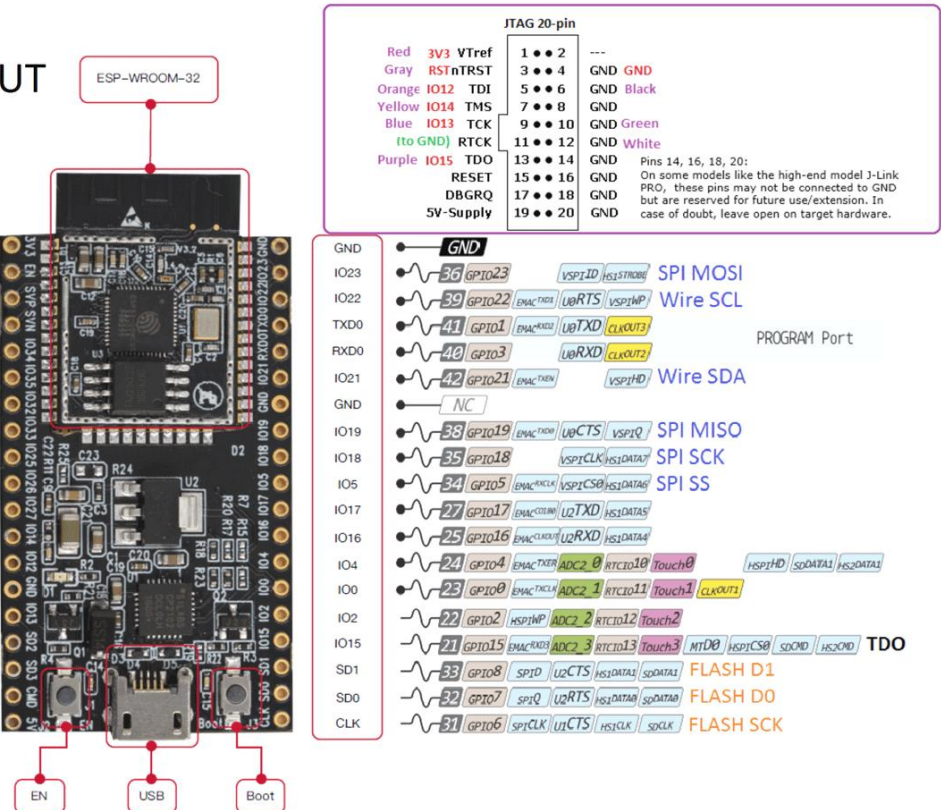
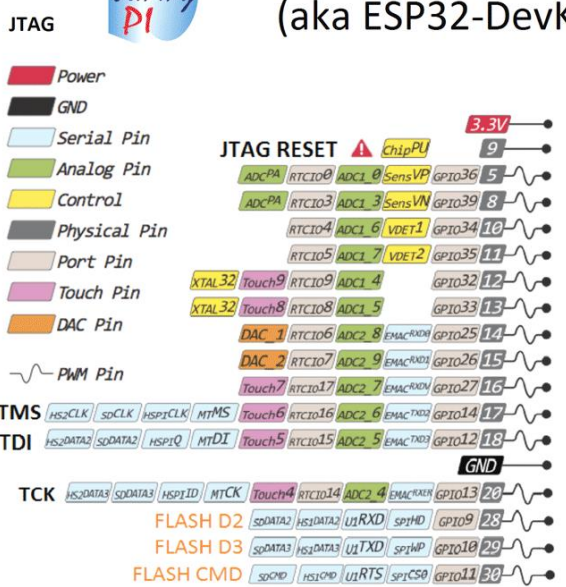


4. Peripherals and Sensors

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	



ESP32-WROOM-32 PINOUT  
(aka ESP32-DevKitC)





Circles in map

Colors [0:99]

RGB Gradient										
1	2	3	4	5	6	7	8	9	10	11
FF2400	FF2700	FF2800	FF2E00	FF3200	FF3500	FF3900	FF3000	FF4000	FF4400	FF4700
12	13	14	15	16	17	18	19	20	21	22
FF4800	FF4E00	FF5200	FF5600	FF5900	FF6D00	FF9000	FF6400	FF6700	FF6B00	FF6F00
23	24	25	26	27	28	29	30	31	32	33
FF7200	FF7800	FF7900	FF7D00	FF8000	FF8400	FF8800	FF8B00	FF8F00	FF9200	FF9600
34	35	36	37	38	39	40	41	42	43	44
FF9900	FF9D00	FFA100	FFA400	FFA800	FFAB00	FFAF00	FFB200	FFB600	FFBA00	FFBD00
45	46	47	48	49	50					
FFC100	FFC400	FFC800	FFCB00	FFCF00	FFD300					

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]

RGB Gradient										
1	2	3	4	5	6	7	8	9	10	11
FFD300	FBD200	F7D200	F4D101	F0D101	EDD002	E9D002	E5CF03	E2CF03	DECE04	DBCE04
12	13	14	15	16	17	18	19	20	21	22
D7CD05	D4CD05	D0CC05	CCCC08	C9CB09	C5CB07	C2CA07	BECA08	BAC908	B7C909	B3CB09
23	24	25	26	27	28	29	30	31	32	33
B0CB0A	ACC70A	A9C70B	A5C70B	A1C60B	9EC60C	9AC50C	97C50D	93C40D	90C40E	8CC30E
34	35	36	37	38	39	40	41	42	43	44
88C30F	85C20F	81C210	7EC110	7AC111	76C011	73C011	6FBF12	6CBF12	68BE13	65BE13
45	46	47	48	49	50	51				
61BD14	5DBD14	5ABC15	56BC15	53BB16	4FB816	4CBB17				

Argon Pollution

Home

Delete all readings

Add 5 readings to db

Send 5 readings via MQTT

Contact us

Map

Plots

Raw DB

Map

Stats last hour  
Max:  
Min:

Pollution  
1-100

Button x

Button y

Debug info:

UpdateDatabase #of calls: xx

Database Readings count: y

Argon Pollution

Home

Delete all readings

Add 5 readings to db

Send 5 readings via MQTT

Contact us

Map

Plots

Raw DB

Map

Pontiac

NASA

Email alert

SMS alert

Download

Greenbelt  
Park

White  
House

Argon Pollution

Home

Delete all readings

Add 5 readings to db

Send 5 readings via MQTT

Contact us

Map

Plots

Raw DB

Map

Raw Database

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

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

COLOR THESAURUS										SHADES OF 9 COLORS	
A	B	C	D	E	F	G	H	I			
YELLOW	ORANGE	RED	PINK	VIOLET	BLUE	GREEN	BROWN	GRAY			
1 YELLOW #FFF200	ORANGE #FC6600	RED #D30000	PINK #FC0FC0	VIOLET #8200ED	BLUE #0018F9	GREEN #3BB143	BROWN #7C4700	GRAY #828282			
2 MELLOW #F8DE7E	GOLD #F9A602	SALMON #FAB072	RUBY #E0115F	HIBISCUS #B43757	DENIM #131E3A	FOREST #0B6623	CEDAR #4B3A26	FOSSIL #787276			
3 CYBER #FFD300	GOLDENROD #D8A520	SCARLET #FF2400	ULTRA #FF6FFF	MAUIVE #784B84	PIGEON #7285A5	SAGE #9DC183	CINNAMON #622A0F	MINK #88807B			
4 ROYAL #FADASE	PINKPIN #FF7417	BARN RED #7C0A02	THULIAN #FDE6FA1	MULBERRY #C64B8C	SKY #95C8D8	OLIVE #708238	BRUNETTE #3A1F04	PEARL RIVER #D9DDDC			
5 TROMBONE #D2B55B	FIRE #FDAS0F	IMPERIAL #ED3939	MAGENTA #FF009D	ORCHID #E4A0F7	INDEPENDENCE #4D516D	LIME #C7EA46	TAWNY #7E481C	ABALONE #D6CFC7			
6 BANANA #FCF4A3	OCHRE #CC7722	INDIAN RED #CD5C5C	ROSE PINK #FF66CC	ORCHID #AF69EE	AIR FORCE #598BAF	HUNTER #3F704D	UMBER #362312	HARBOR GRAY #C7C6C1			
7 TUSCANY #FCD12A	AMBER #FFBF00	CHILI #C21807	LAVENDER #F8AED2	LILAC #B660CD	BABY BLUE #B9CFF7	JADE #00A868	TORTILLA #99795D	SMOKE #B6B0B8			
8 LEMON #FFD95F	DIJON #CA9102	FIRE BRICK #B22222	CREAMY #FF69B4	ELECTRIC #8F00FF	NAVY #000080	ARTISHOKE #8F9778	CHOCOLATE #3B1700	THUNDER #BDB7A8			
9 BUMBLEBEE #FFC205	TANGERINE #F9812A	MAROON #800000	FUCHSIA #FF00FF	AFRICAN #8264BE	STEEL #4682B4	FERN #4F7342	SYRUP #592000	PEWTER #999999			
10 FLAX #EEDCB2	TIGER #F06A02	REDWOOD #A43A52	FRENCH ROSE #F04A8A	GRAPE #8F2DA8	CAROLINA #57A0D2	JUNGLE #23A837	GINGERBREAD #5C4C05	STEEL #777B7E			
11 CREAM #FFFDD0	HONEY #FDB905	RASPBERRY #D21F3C	CERISE #DE3163	AMETHYST #9966CB	TURKISH #5097A4	LAUREL #A9BA9D	CARAMEL #751613	STONE #87777D			
12 PEACH #FFE5B4	CARROT #E77215	CANDY APPLE #FF0800	CARNATION #FF69B4	BYZANTINE #702963	MAYA #73C27B	MOSS #8A9A5B	WALNUT #43270F	IRON #484948			
13 LAGUNA #F8E473	APRICOT #FDD800	CARMINE #990019	BRICK #FF660F	FANDANGO #85338A	CORNFLOWER #6699FF	MINT #98FB98	PECAN #48260D	RHINO #B998B6			
14 MUSTARD #FEDC56	BRONZE #B1560F	PERSIAN #CA3433	AMARANTH #F19CBB	HELIO #DE73FF	OLYMPIC #008ECC	PINE #01796F	WOOD #402F1D	TROUT #97978F			
15 ECRU #FCEB180	CIDER #B3672B	U.S. FLAG #8F0A3D	TARTY #F987C5	FLORAL #B47EDE	SAPPHIRE #0F52BA	TEA #D0F0C0	HICKORY #351E1D	SEAL #81838D			
16 CORN #E4CD05	CLAY #B13F0B	FERRARI #FF2800	BUBBLE GUM #FF69B4	ROSE #D78FDC	AZURE #0080FF	ARMY #4B532D	ESPRESSO #4B382A	LAVA #808588			
17 PINEAPPLE #FEE12B	RUST #8B4000	BURGUNDY #8D021F	HOT PINK #FF1894	ROYAL #852A9	EGYPTIAN #1134A6	EMERALD #00C878	PEANUT #795C32	SHADOW #363636			
18 FLAXEN #D5985A	AMBER 2 #8B3D00	CRIMSON #8B000A	PUNCH #EC5578	LOLLIPOP #81007F	VALE #0EAC92	KELLY #4CBB17	MOCHA #3B270C	ASH #544C4A			
19 EGG NOG #F9E29C	SPICE #793802	SANERIA #5E1914	LIMONADE #F08080	PLUM #8D4585	PRUSSIAN #003151	SACRAMENTO #043927	COFFEE #4B3619	ANCHOR #3E424B			
20 SERIA #F26778	BURN'T OR. #5C4000	MAHOAGANY #520000	FLAMINGO #FC3357	EGGPLANT #311432	SPACE #1C9551	SEA #2E8B57	RUSSET #734618	CHARCOAL #222021			
YELLOW #FFF200	ORANGE #FC6600	RED #D30000	PINK #FC0FC0	VIOLET #8200ED	BLUE #0018F9	GREEN #3BB143	BROWN #7C4700	GRAY #828282			

*chart.js*

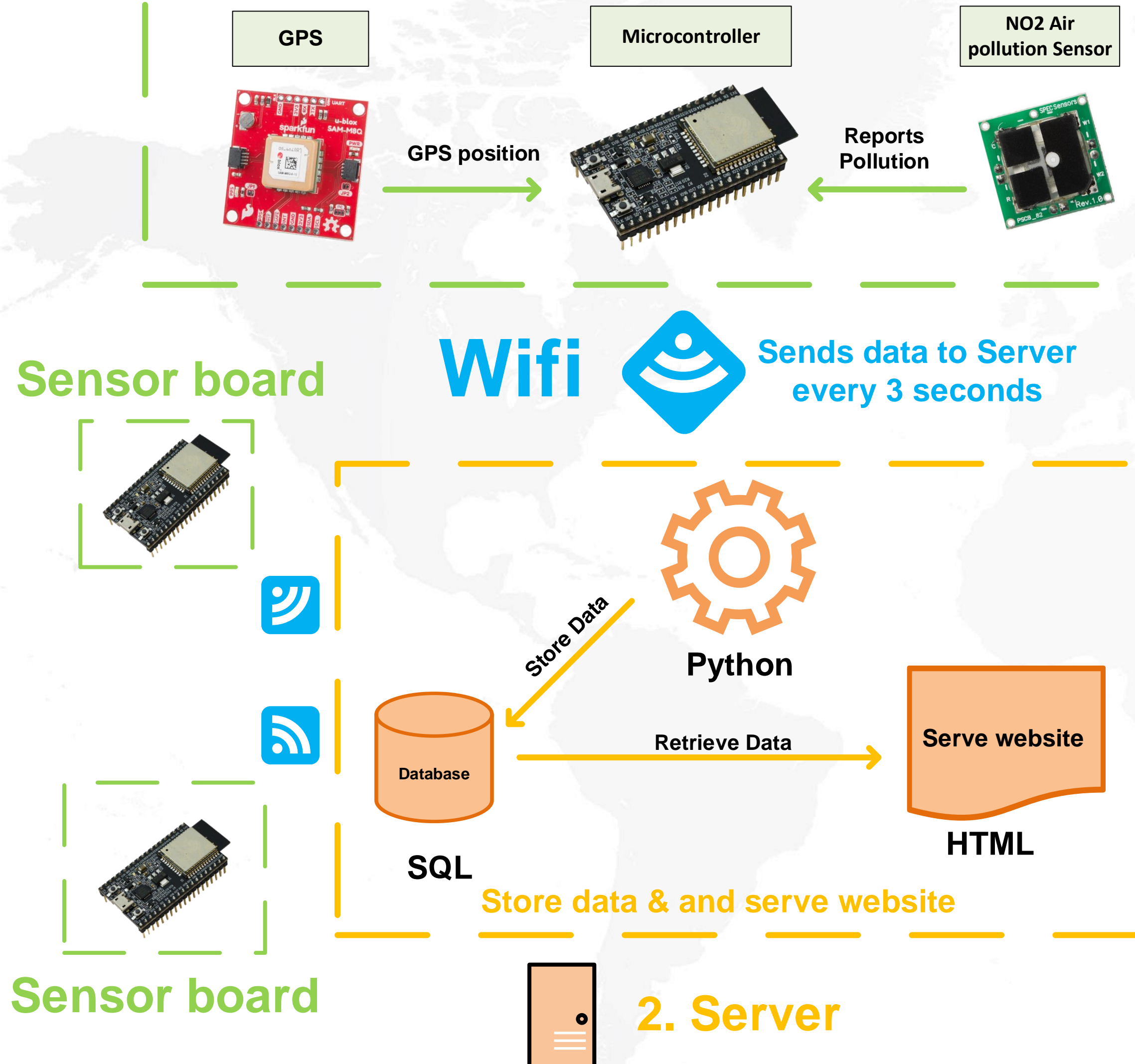
```
var config = {
  ...
  options: {
    title: {
      text: 'Pontiac Street'
    }
  }
  scales: {
    xAxes: {
      realtime: {
        ...
        onRefresh: onRefresh
      }
    }
  }
};

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

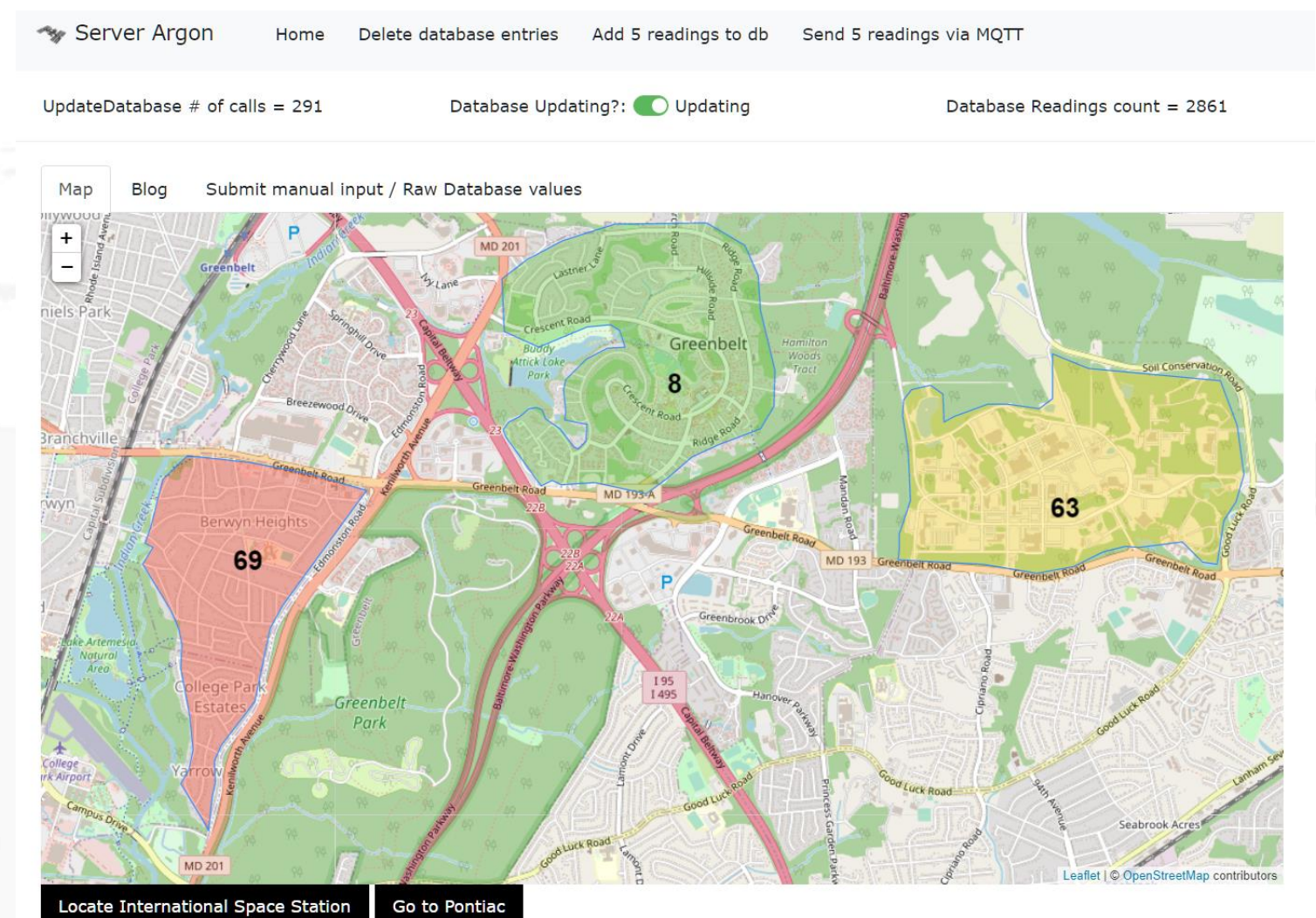


# 1. Sensor Board

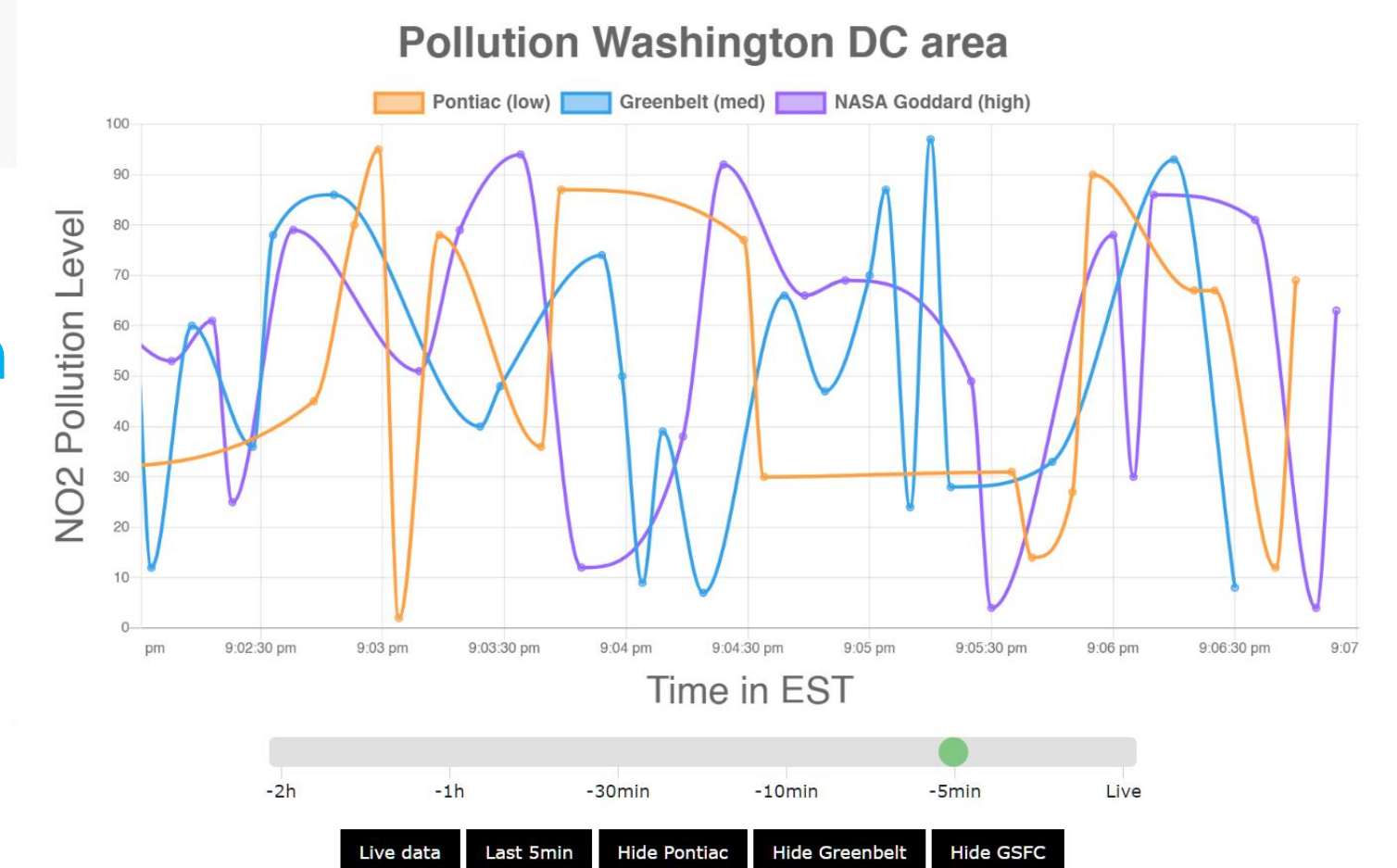
Capture data, send to server & sleep



Live Map



Live Graph



Display Real-time sensor data

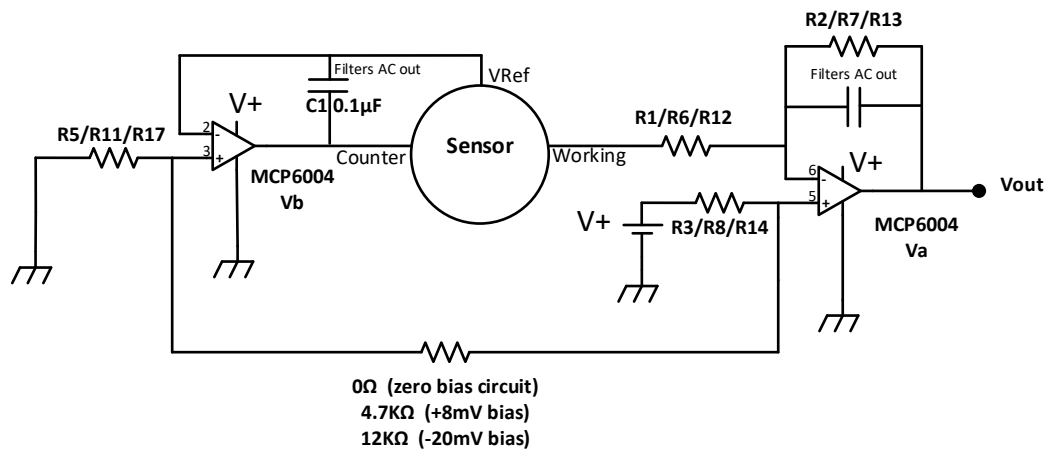
3. Website

- 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

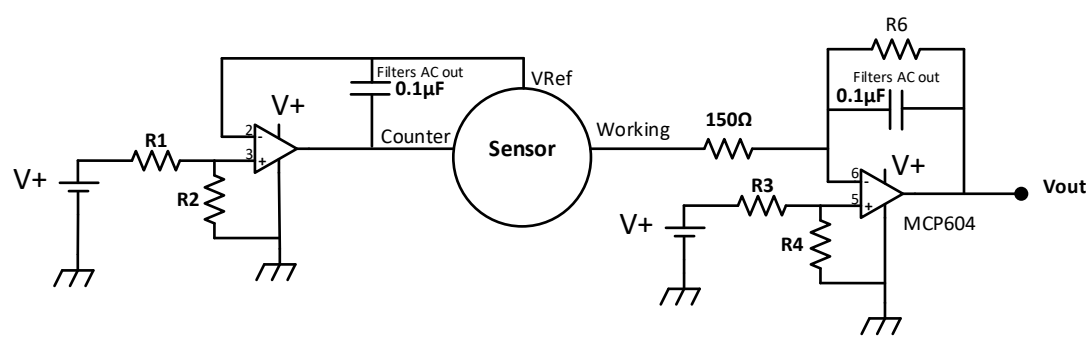


# Circuit Proposed by Galvin



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.

$$Cx = \frac{1}{M} \times (Vgas - Vgas_0)$$

Cx = gas concentration in ppm

Vgas = output voltage from Vgas

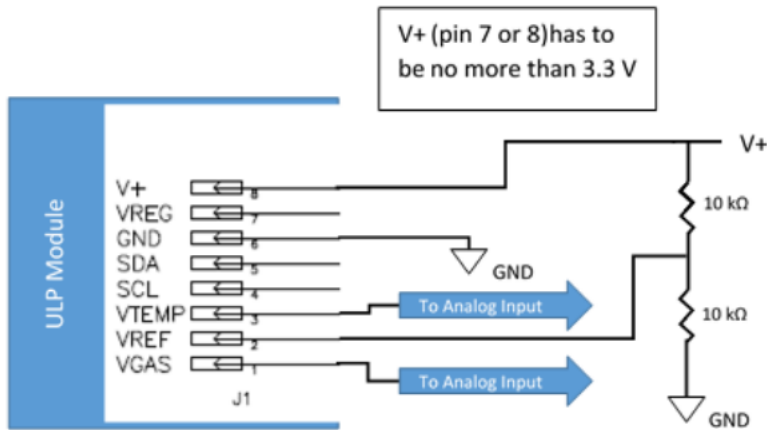
Vgas\_0 = output voltage from Vgas in clean-air environment

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

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

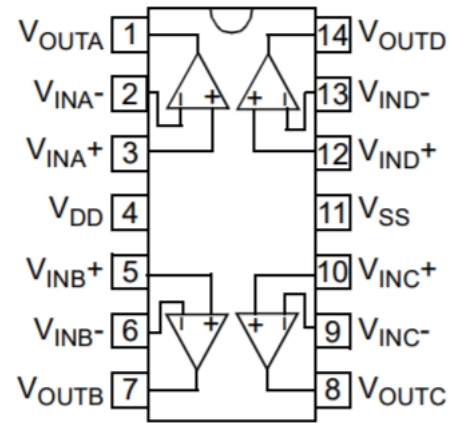
Sensitivity Code = provided on sensor label (see below)

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

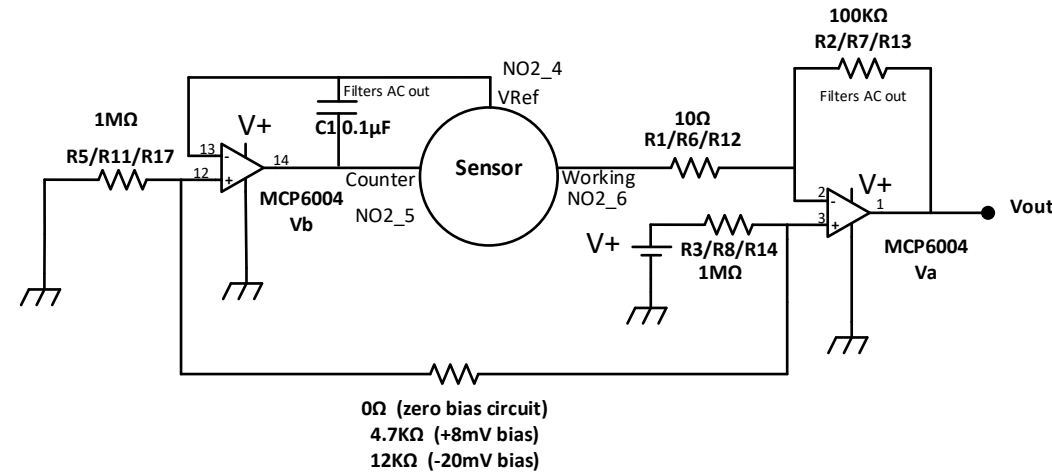


## MCP6004

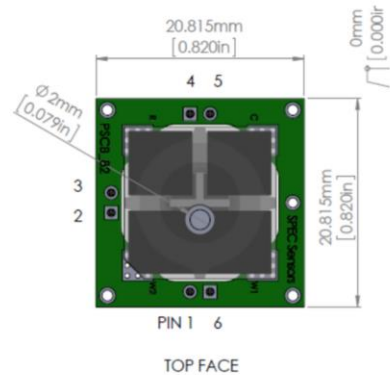
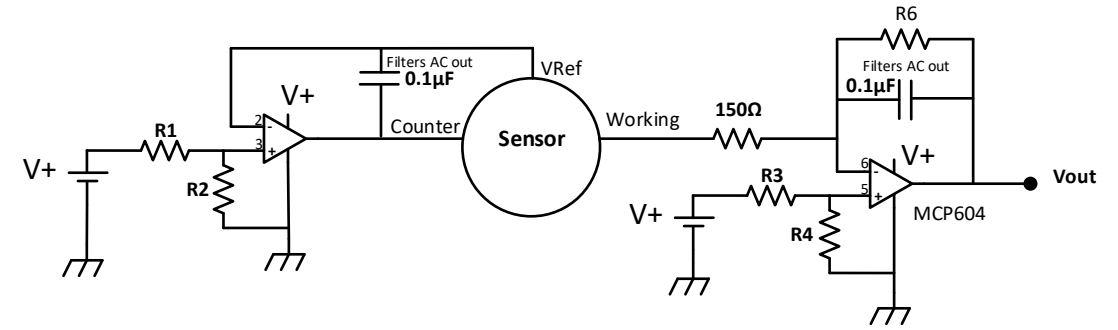
14-Lead PDIP, SOIC, TSSOP



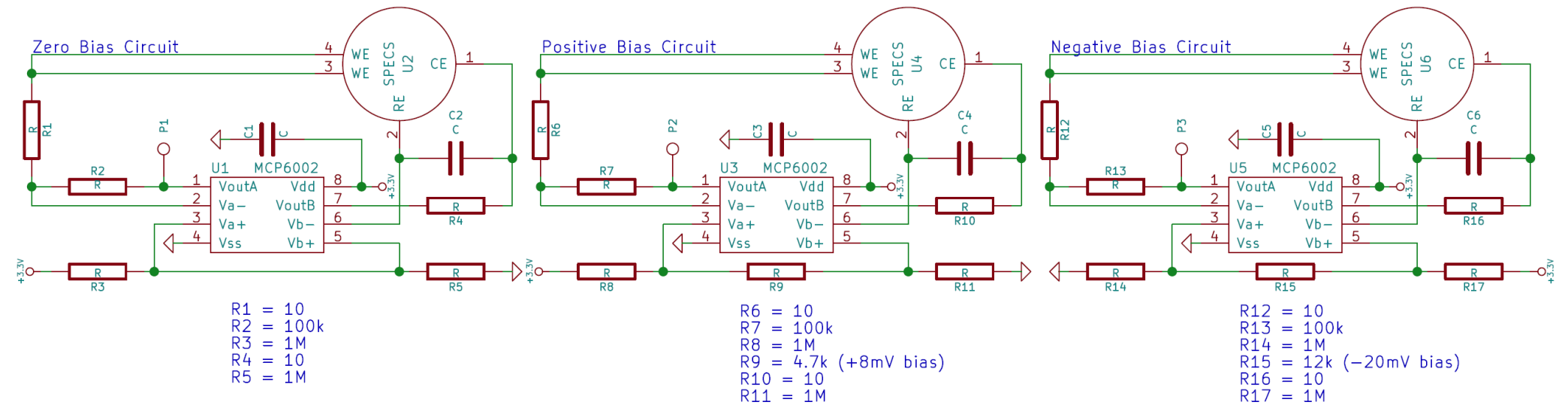
## Circuit Proposed by Galvin



## Circuit on the Spec Analog Sensor Developer Kit datasheet

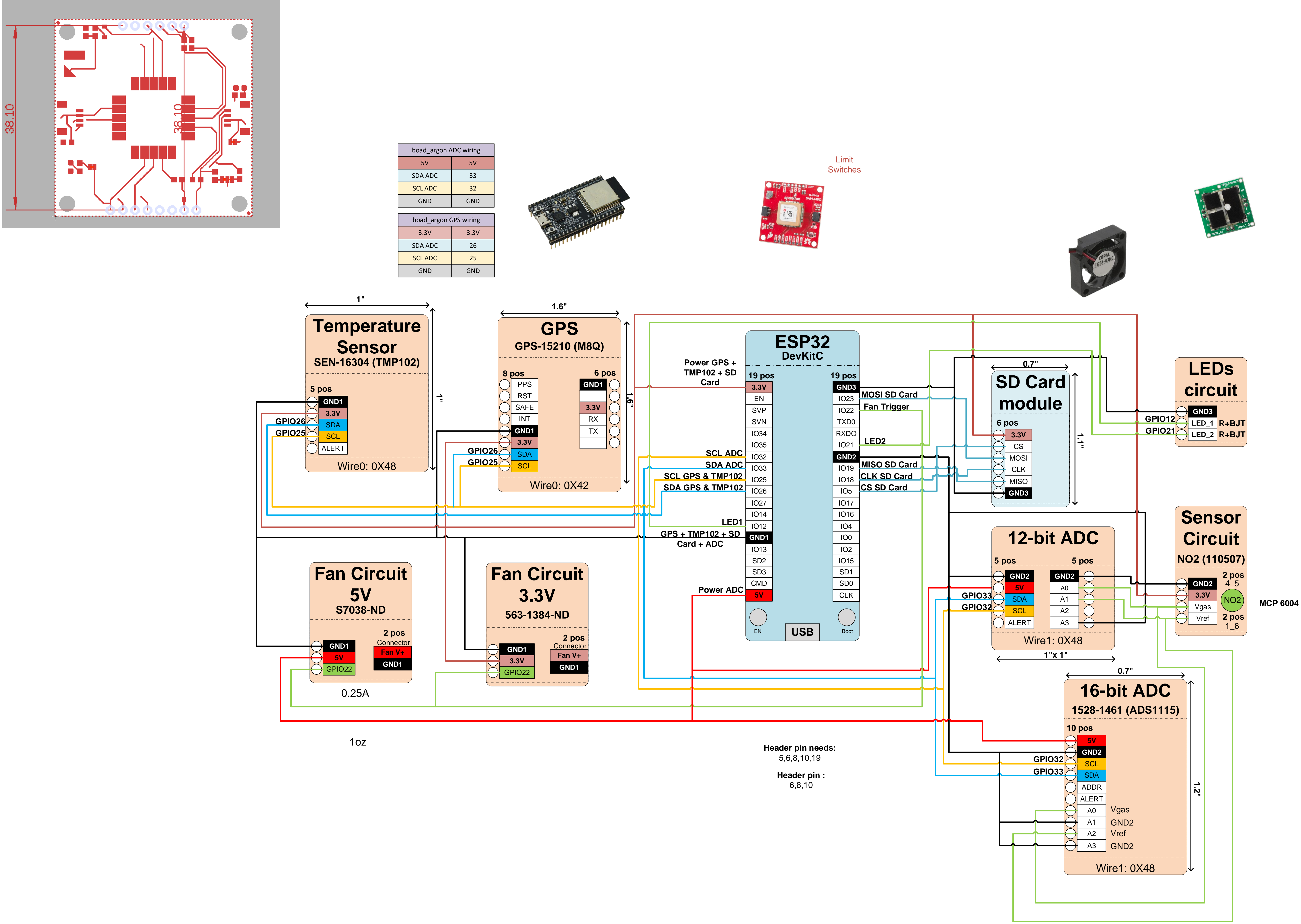


PIN	CONNECTION
1	WORKING
2	NC
3	NC
4	REFERENCE
5	COUNTER
6	WORKING





GPS Dimensions 1.5" distance holes



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.

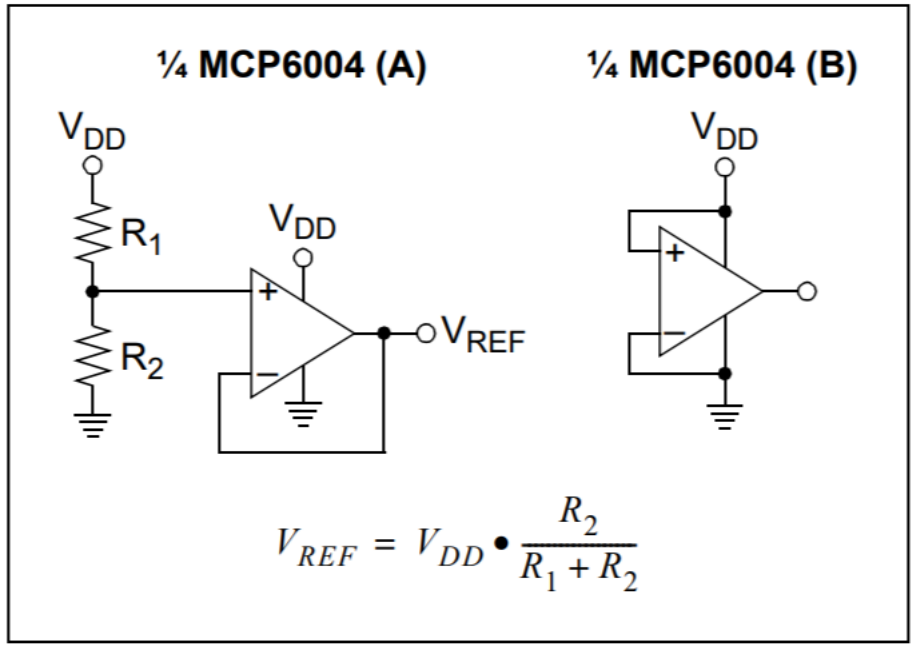


FIGURE 4-5: Unused Op Amps.

