Destination Weather Station v4.5

Release 0.3.0

Destination SPACE Inc.

Feb 16, 2026

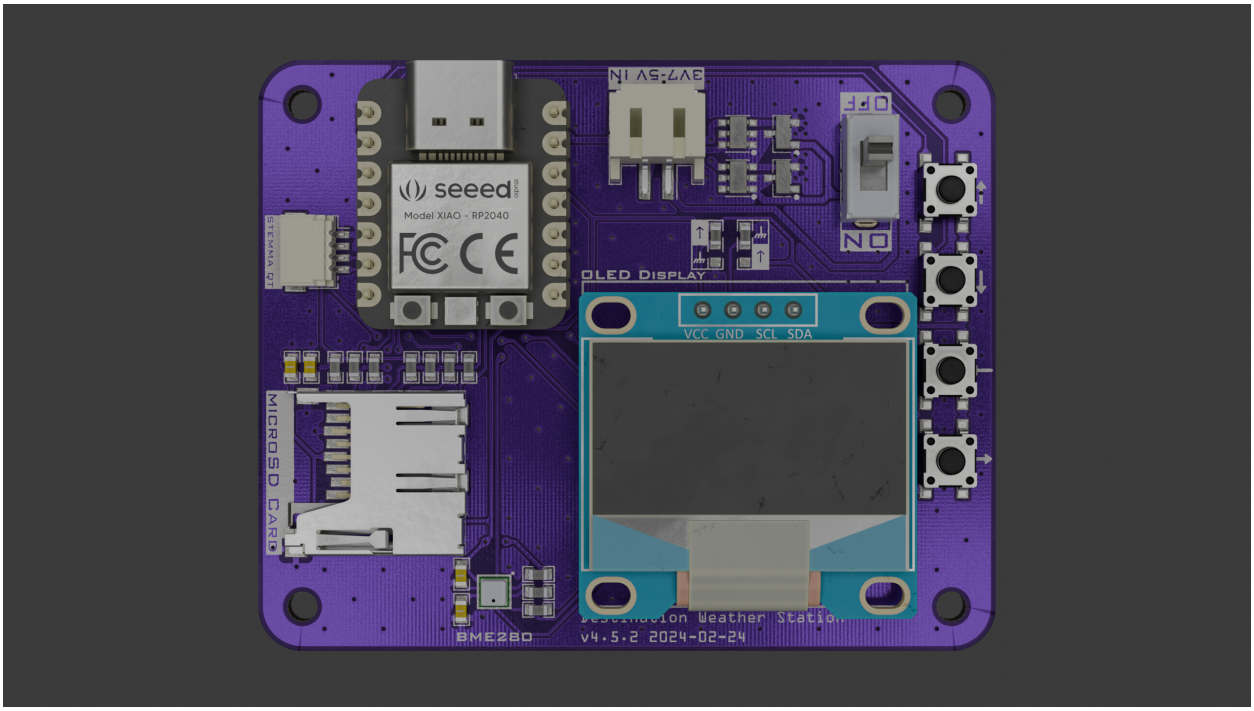
GETTING STARTED

An Open Source Remote Sensing Platform for National Taiwan Normal University

CHAPTER

ONE

CONTENTS

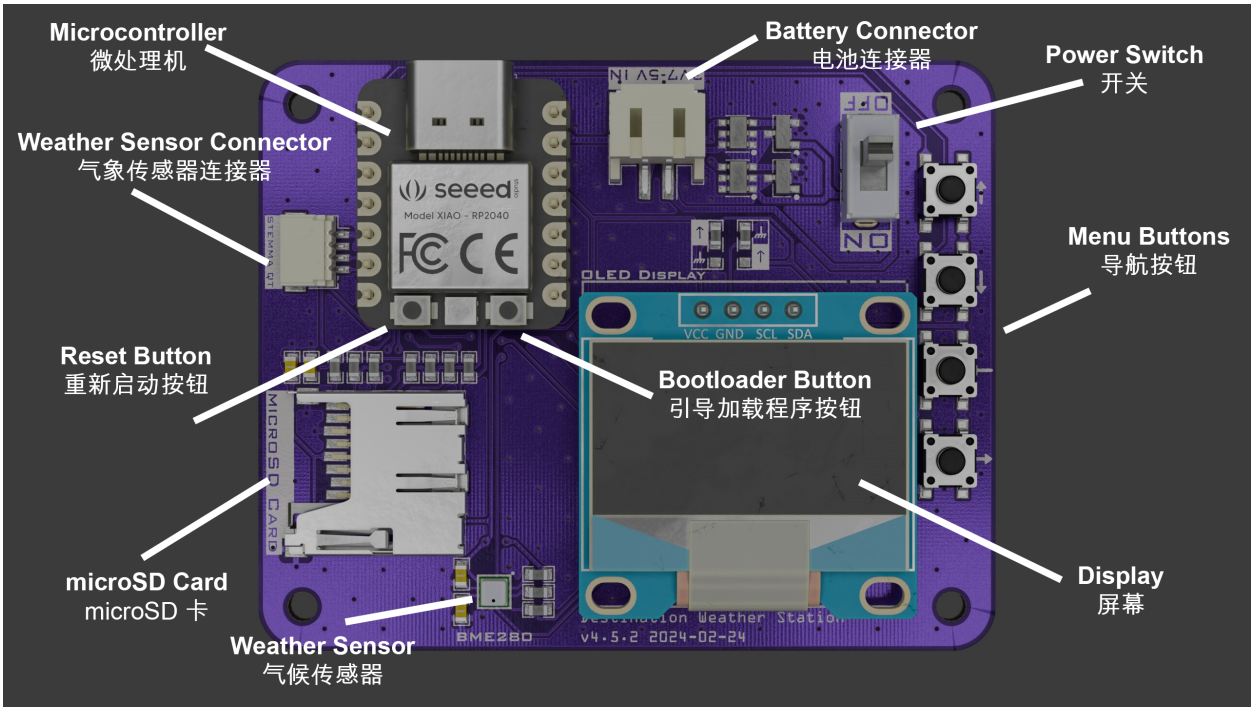


1.1 What is in the Kit?

1.1.1 Kit Contents

- Destination Weather Station v4.5.2
- Seeeduino XIAO RP2040 microcontroller w/ headers
- 128x64px OLED display w/ headers
- SPDT power switch
- AAA battery case
- 8GB micro-SD card
- USB-A to USB-C cable
- USB microSD card adapter

1.2 Destination Weather Station v4.5 Assembly



The Destination Weather Station v4.5 is preassembled with the majority of the circuit components.

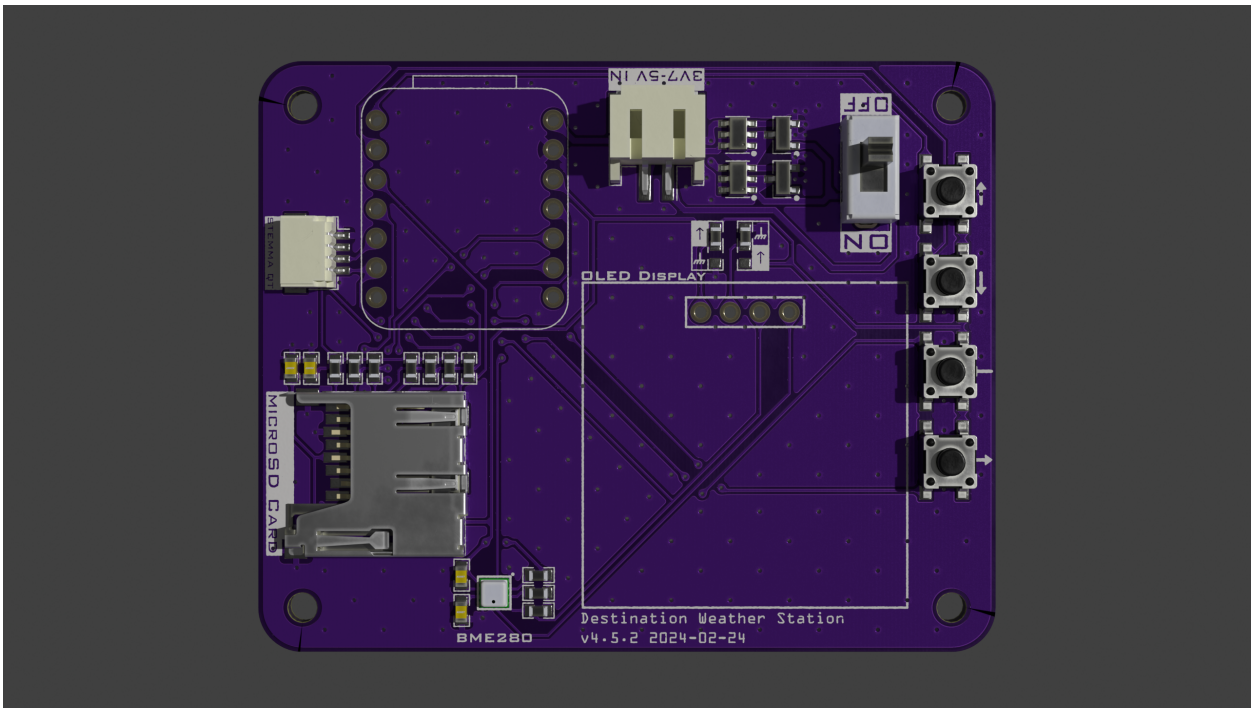
Before using the weather station, you will need to *solder* the *XIAO RP2040 microcontroller*, *OLED display*, and power switch. Follow the instructions below to solder your weather station. You can then *connect the batteries* and insert the microSD card.

1.2.1 Soldering

To solder the weather station, follow the instructions below. Tools required to complete these steps are:

- Soldering iron
- Solder wire

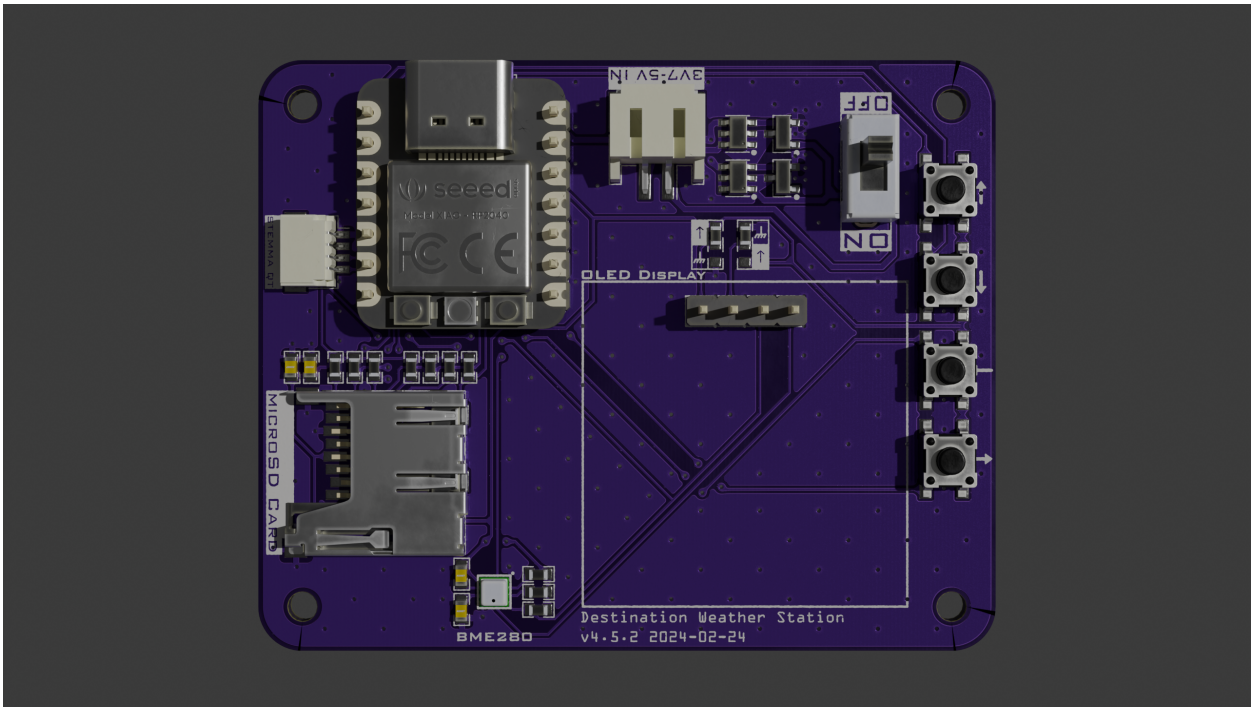
Power Switch



To solder the power switch to the weather station, insert it into the front side of the board. Flip the board over and solder the pins, starting with the two outer ones.

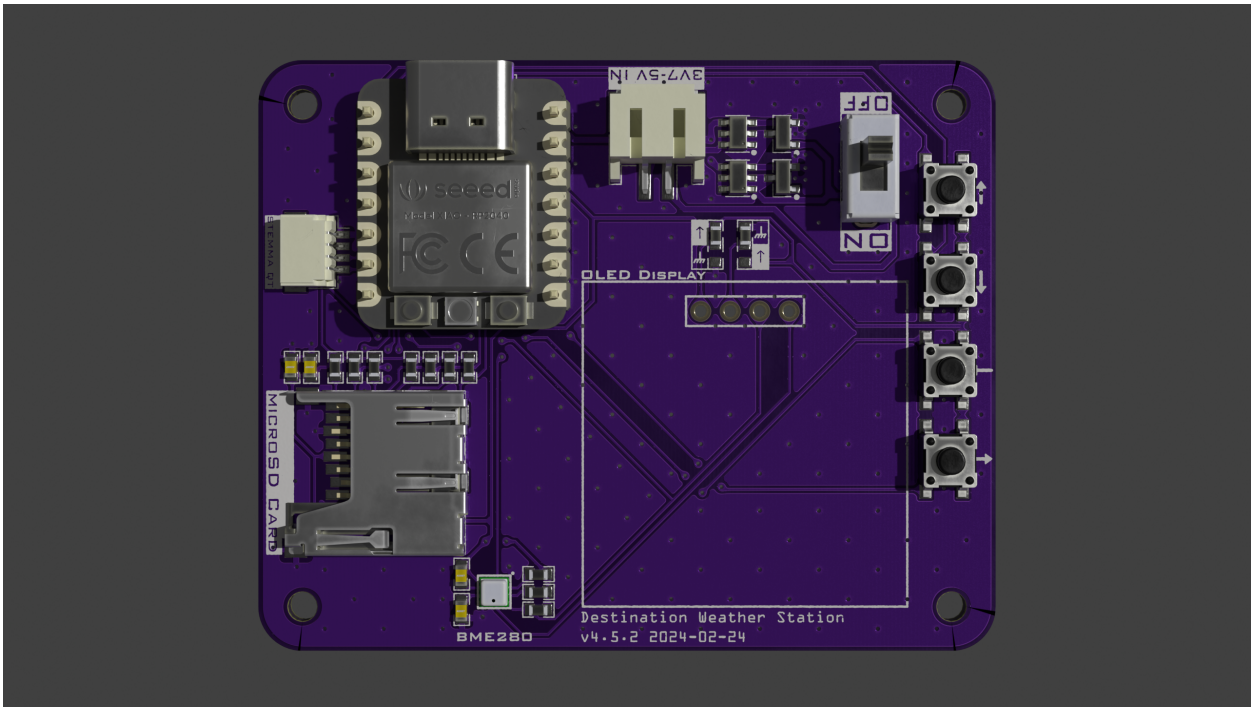
XIAO RP2040

To solder the XIAO RP2040 to the weather station, begin by inserting the long side of the 1x7 header pins into the weather station board as shown below.



Next, flip the weather station over and solder all 14 pins to the weather station.

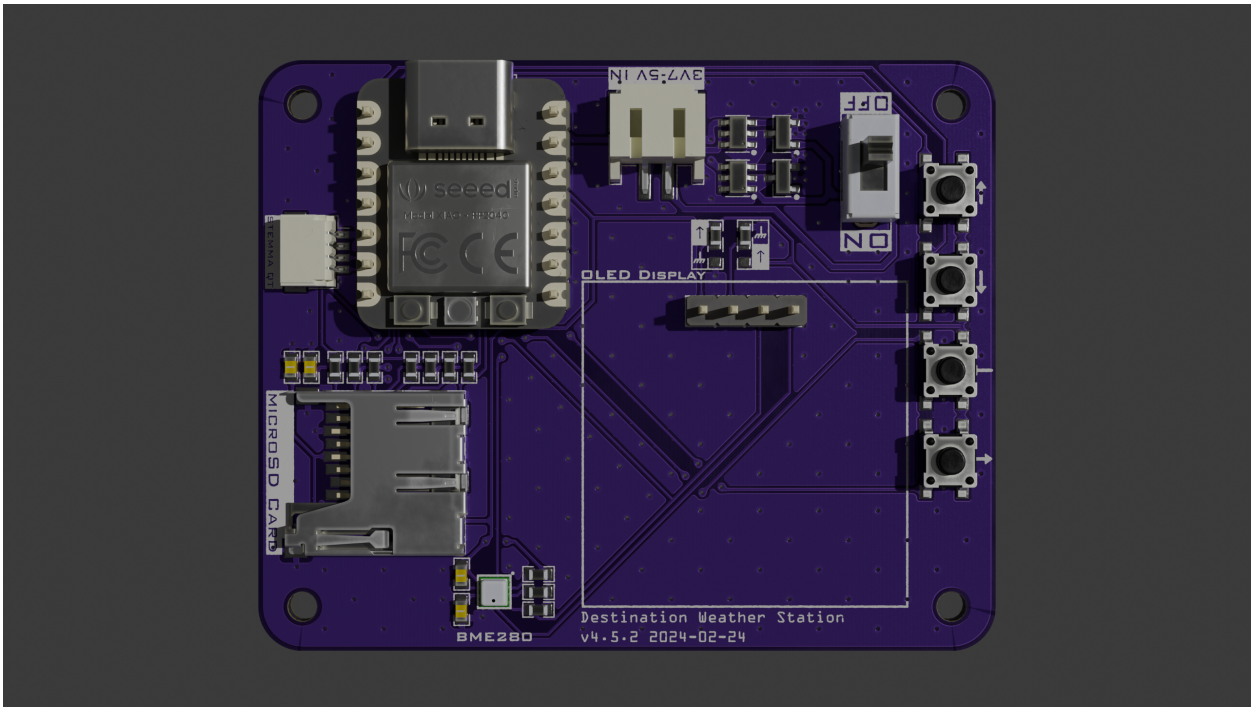
Finally, flip the weather station back over and solder the XIAO to the short side of the header pins. After you are done, it should look like the image below.



OLED Display

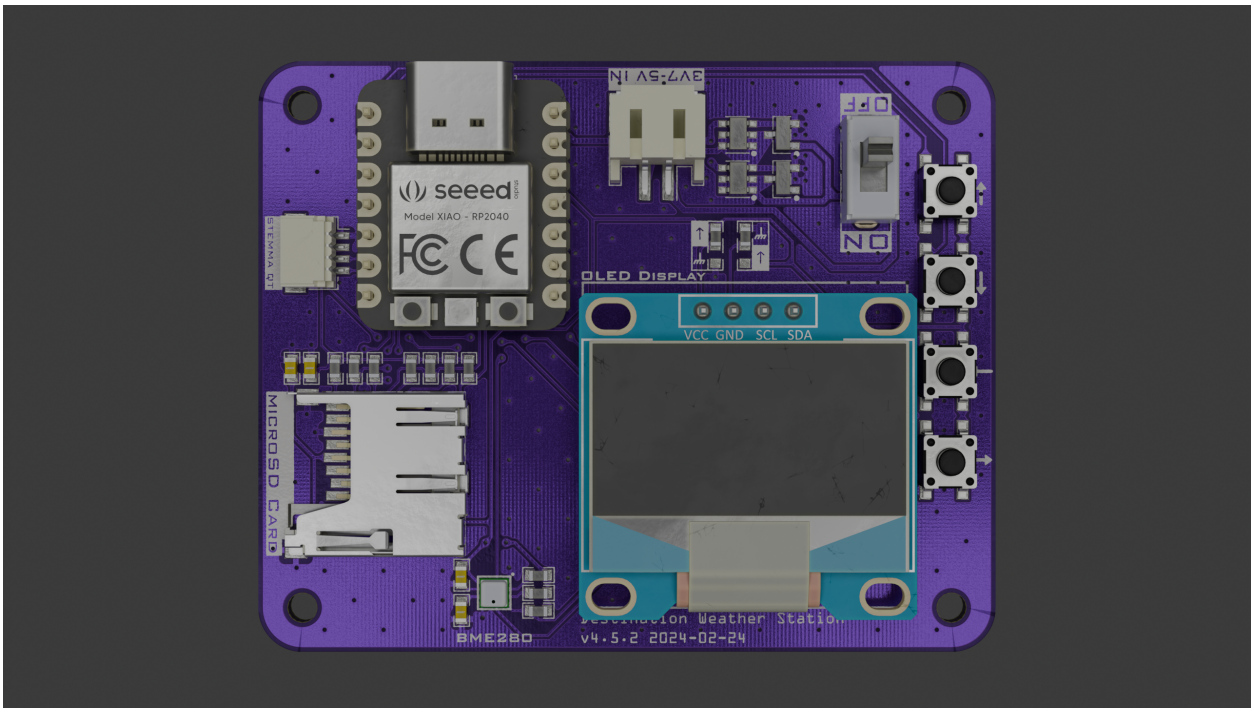
To solder the OLED display to the weather station, you will follow similar procedures to the steps above to solder the XIAO.

Begin by inserting the long side of the 1x4 header into the weather station board as shown below.

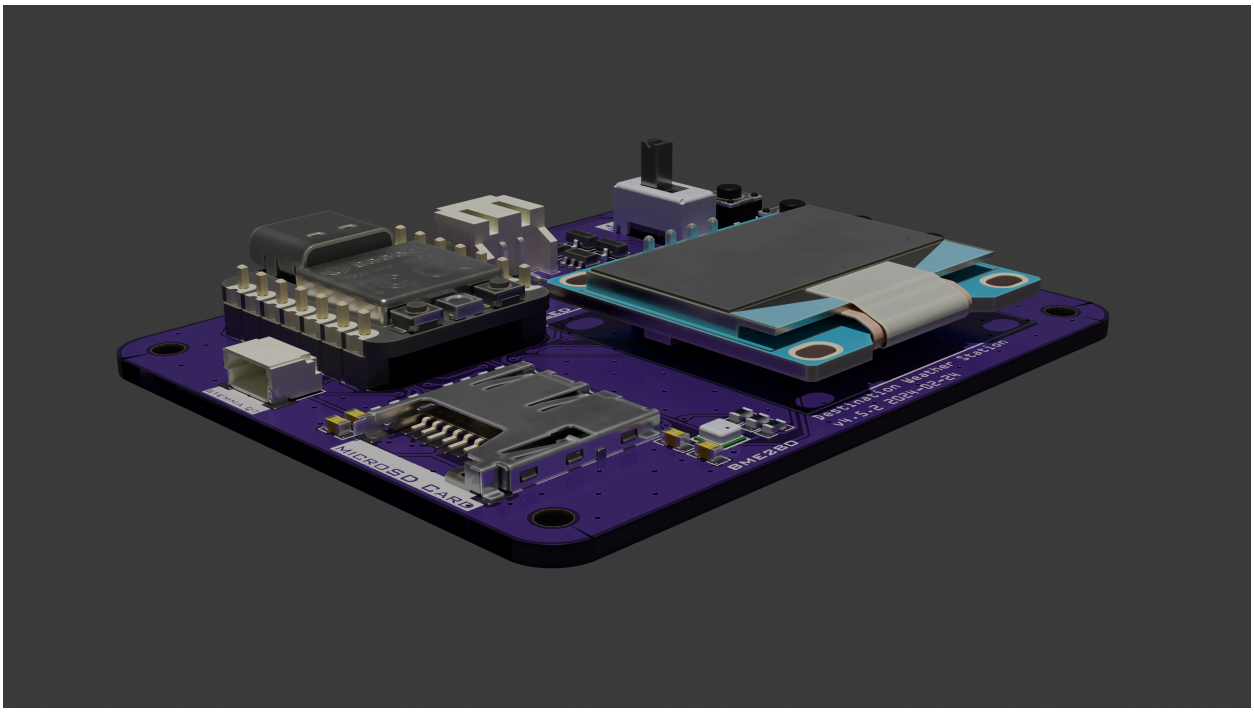


Next, flip the board over and solder the pins to the back.

Similarly, flip the board back over and solder the display to the front.



1.2.2 Final Assembly



Batteries

Insert three (3) AAA batteries into the battery pack. Connect the JST-PH connector to the matching connector on the weather station.

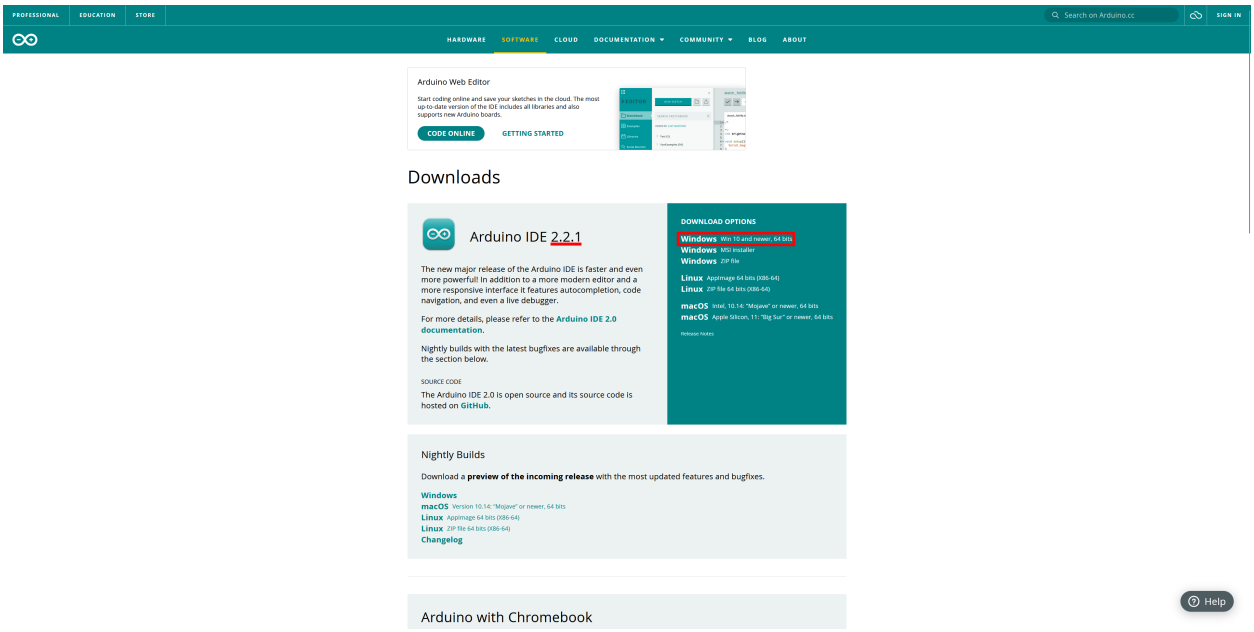
1.3 Installing Arduino IDE

To program the Destination Weather Station, you will need to download and install Arduino IDE.

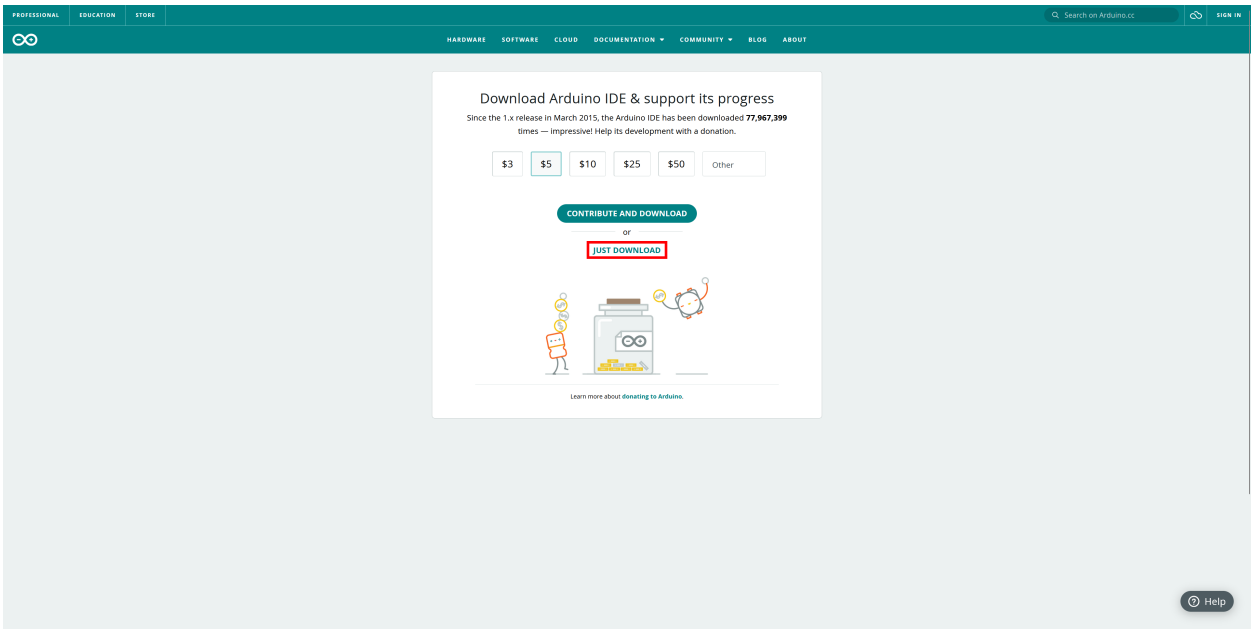
1.3.1 Download

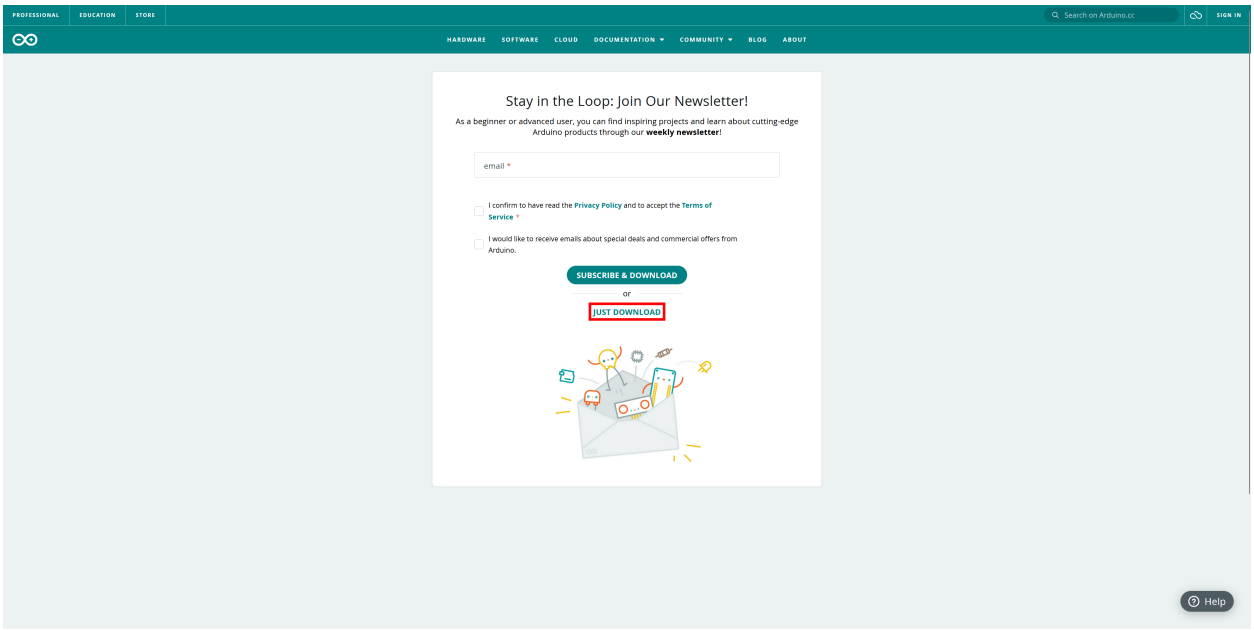
To begin, navigate to https://www.arduino.cc/en/software.

Click `Windows Win 10 and newer, 64 bits` under the most recent version of IDE. This may be different than the screenshot below



When the page loads, click next





1.3.2 Installation

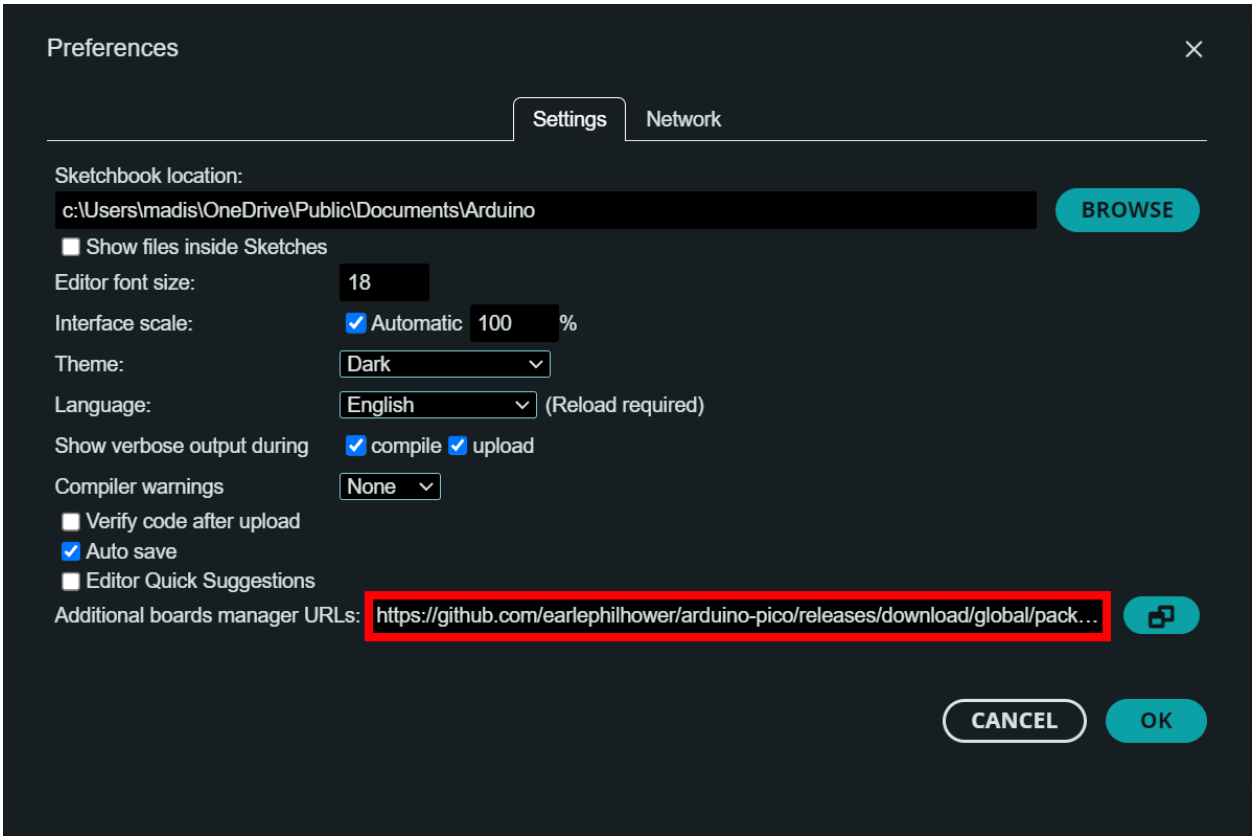
To install Arduino IDE, navigate to where you saved your `arduino-ide_2.x.x_Windows_64bit.exe` file and follow the instructions in the install.

1.4 Software Configuration

1.4.1 Board Manager

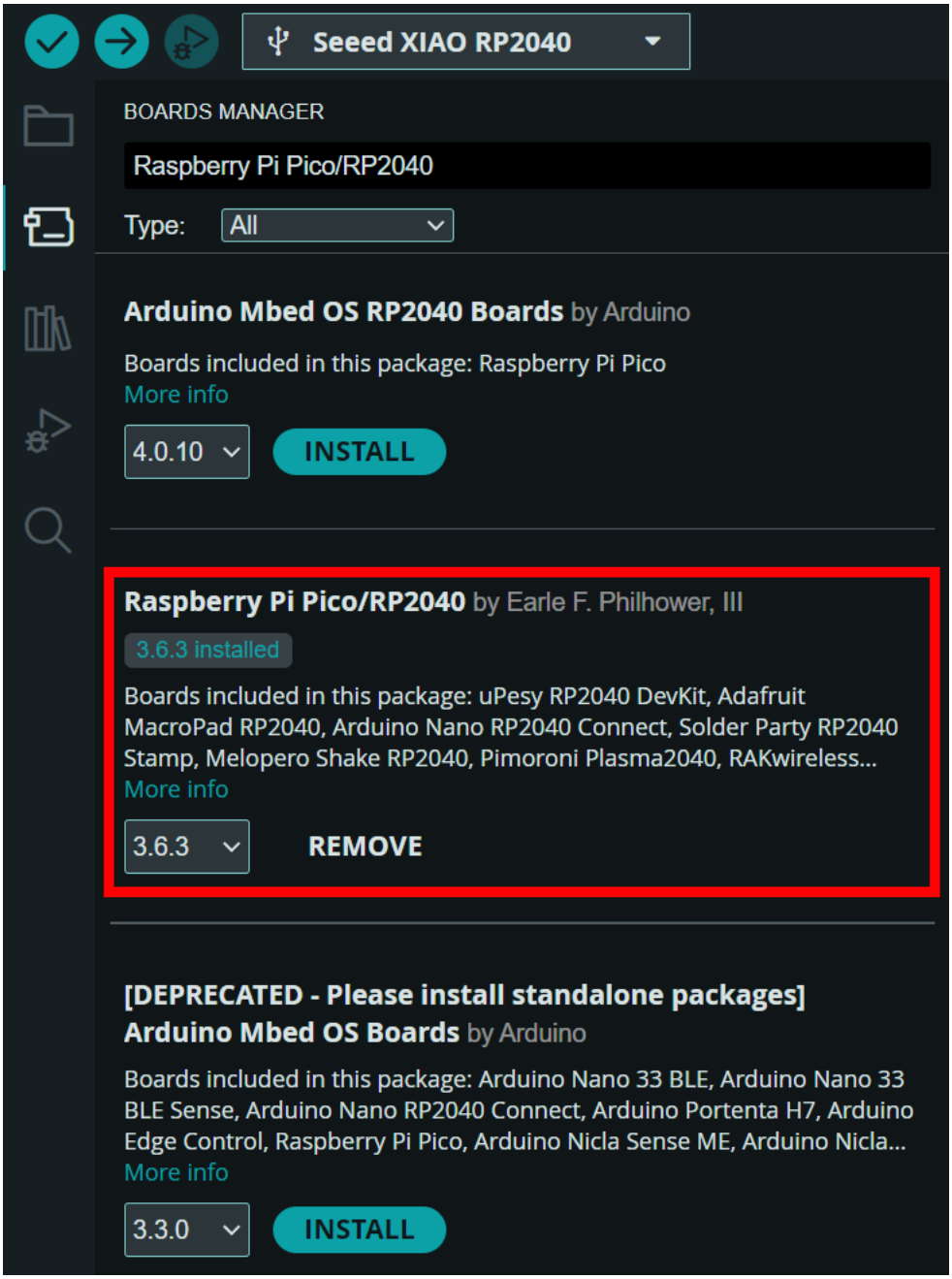
To properly configure IDE, the board manager for the XIAO RP2040 needs to be installed.

In Arduino IDE, navigate to **File > Preferences**. In the window, enter `https://github.com/earlephilhower/arduino-pico/releases/download/global/package_rp2040_index.json` into the **Additional boards manager URLs:** text box.



Click **OK**

Next, navigate to **Tools > Board: > Boards Manager...**. Search for `Raspberry Pi Pico/RP2040` and click **INSTALL** on the option published by Earle F. Philhower, III.



1.4.2 Code Libraries

Several libraries are needed to upload code to the weather station.

To install these libraries, navigate to **Tools > Manage Libraries...**

In the search box enter the following library names and install each including any required dependancies.

- `Adafruit BME280 Library`
- `Adafruit LTR390 Library`
- `Adafruit SSD1306`
- `Adafruit VEML7700 Library`

- `ENS160 - Adafruit Fork`
- `Sensirion I2C SCD4x`

1.5 Overview of Sensors

The Destination Weather Station is equipped with a *Bosch BME280 combined humidity and pressure sensor*.

In the expansion sensor kit, several more environmental sensors are provided to measure a wide variety of parameters. These include:

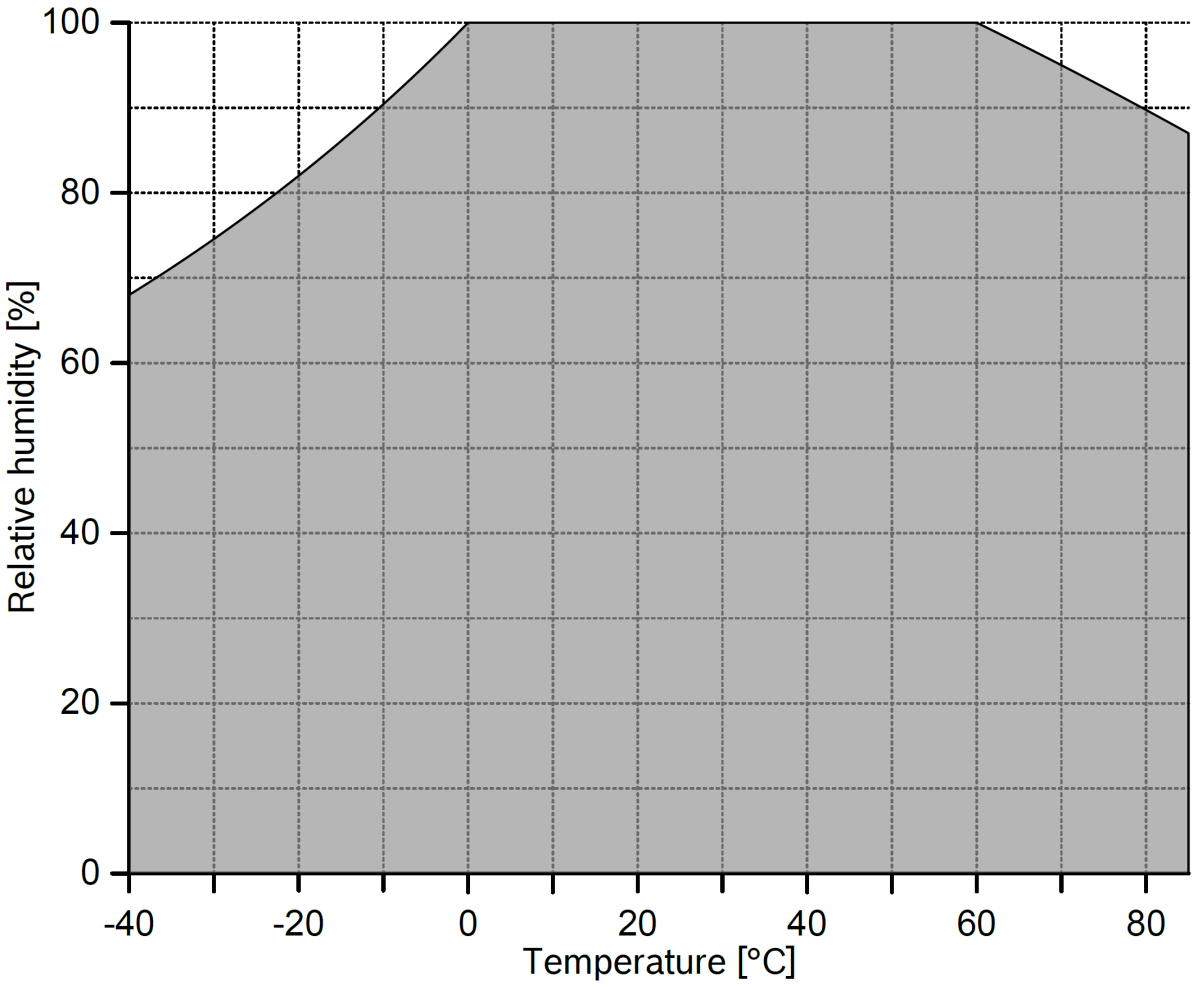
- *ScioSense ENS160 - Digital metal-oxide multi-gas sensor*
- *Lite-On LTR390 - Ambient light and ultravoiolet light sensor*
- *Sensirion SCD40 - Miniature CO2 sensor*
- *Vishay VEML7700 - High accuracy ambient light sensor*

1.6 BME280

The BME280 combined humidity and pressure sensor is a classic device used for accurately measuring temperature, pressure, and humidity.

1.6.1 Humidity

Parameter	Specifications
Operating Range	0 - 100 %RH
Absolute Accuracy	±3 %RH
Hysteresis	± %RH



1.6.2 Pressure

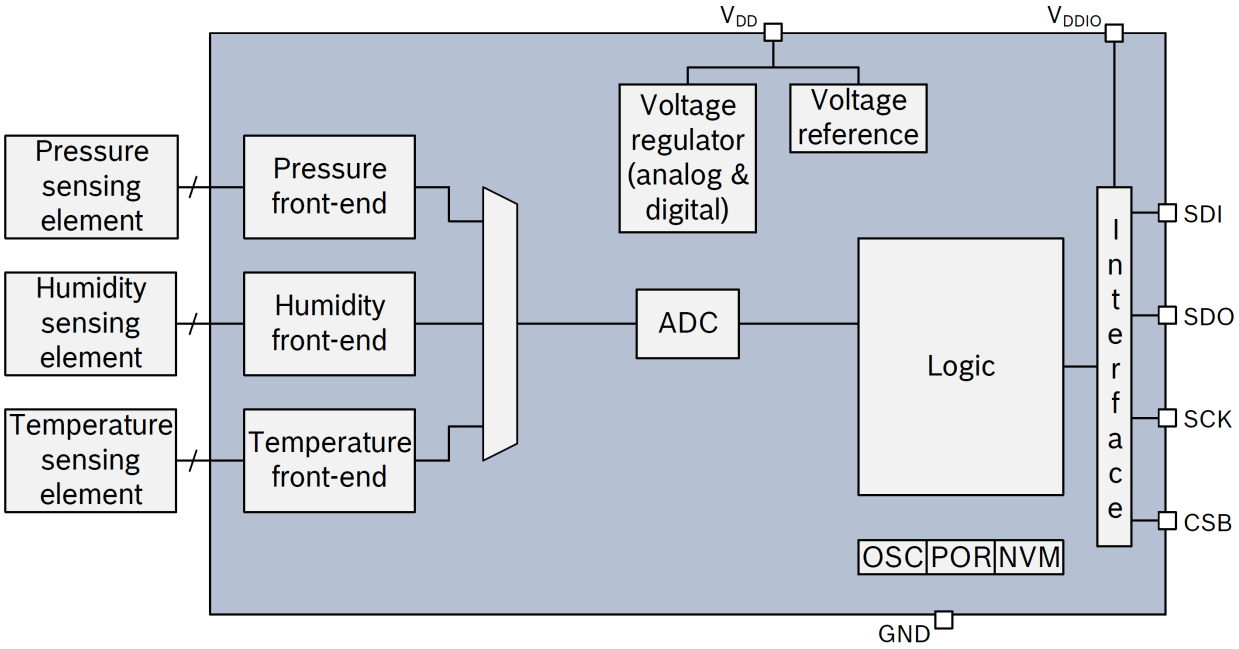
Parameter	Specifications
Operating Range	0 - 1100 hPa
Absolute Accuracy	± 1.0 hPa
Relative Accuracy (700 - 900 hPa & 25 - 40 °C)	± 0.12 hPa

1.6.3 Temperature

Parameter	Specifications
Operating Range	0 - 65 °C
Absolute Accuracy	0.5 °C

1.6.4 Functional Block Diagram

Below is a functional block diagram for how the BME280 sensor works. The data for each parameter (humidity, pressure, and temperature) are measured using an analog sensor. The data is then converted from an analog voltage signal to a digital data signal. This data is then passed to the logic side of the sensor which sends and receives data to/from the host device.



1.7 ENS160

The ENS160 digital metal-oxide multi-gas sensor measures volatile organic compounds (VOCs), which can be used to measure total VOC content (TVOC), air quality index (AQI), and calculate an approximate value for carbon-dioxide concentration (eCO2).

1.7.1 Specifications

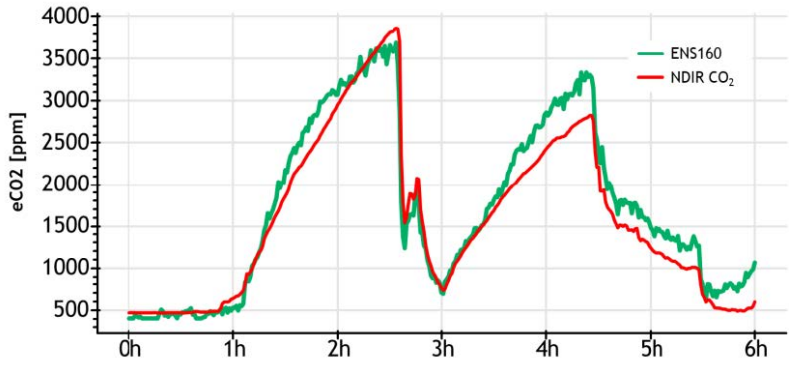
Parameter	Range	Resolution	Units
TVOC	0 - 65,000	1	ppb
eCO2	400 - 65,000	1	ppm CO2 equivalent
AQI-UBA	1 to 5	1	•
AQI-EPA	0 - 500	1	•

1.7.2 TVOC

Total Volatile Organic Compounds (TVOC) is a useful parameter used to determine the quality of the air in a given area. This is incredibly important for urban environments and in indoor settings. This is typically measured in parts-per-billion (ppb), but in polluted environments, can reach into the parts-per-million (ppm) range.

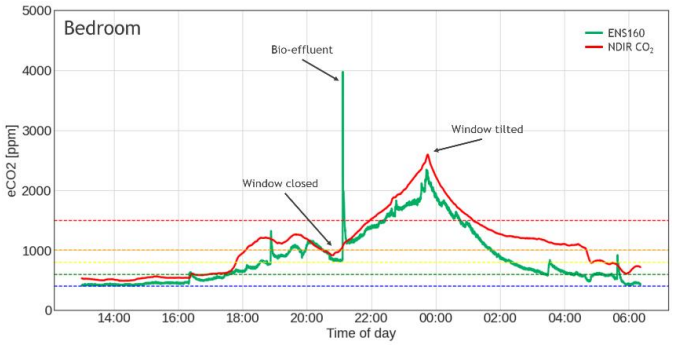
1.7.3 eCO2

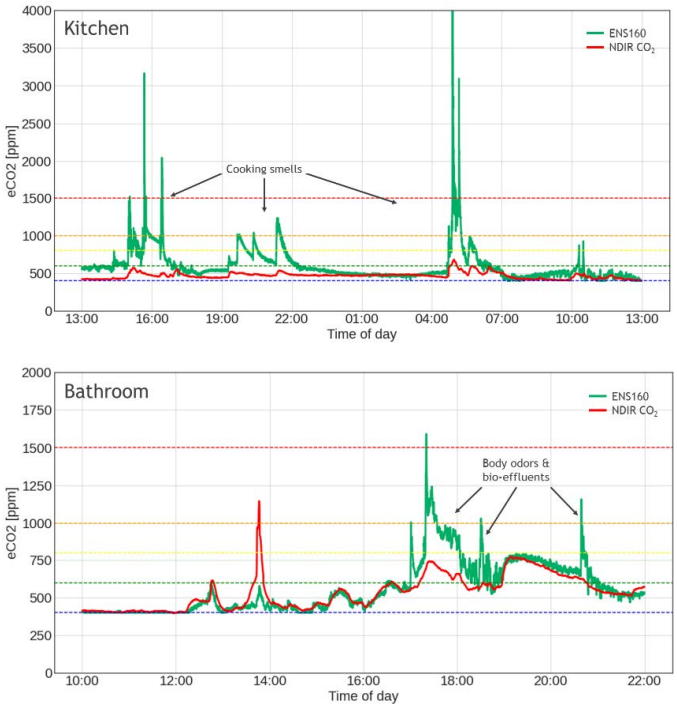
The ENS160 sensor uses an algorithm to calculate an approximate value for carbon-dioxide (CO2) concentration. While the sensor cannot directly measure CO2, it's algorithm can closely approximate this value, as seen in the graph below.



In the graph, the calculated eCO2 measurement is being compared to a non-dispersive infrared CO2 sensor. It can closely approximate the CO2 concentration, but it is not perfect.

In the graphs below, ScioSense has provided eCO2 measurement for different environments, such as in a bedroom, kitchen, and bathroom. These graphs show that different conditions, such as closing a window, cooking, or being in a room can increase CO2 concentrations.





1.7.4 AQI

Air Quality Index (AQI) is the typical parameter used to judge the quality of air and how polluted it is. There are several different scales used to measure this value, but they are all based on the concentration of VOCs. For example, the United States uses the NowCast algorithm developed by the Environmental Protection Agency (EPA) ranging from 0 to 500, the European Union's European Environment Agency (EEA) uses a scale ranging from 0 to 1250, and Taiwan's Ministry of Environment uses a similar scale to the United States, ranging from 0 to 500. The figures below show each of these indexes.

Environmental Protection Agency (EPA)

AQI Basics for Ozone and Particle Pollution

Daily AQI Color	Levels of Concern	Values of Index	Description of Air Quality
Green	**Good**	**0 to 50**	**Air quality is satisfactory, and air pollution poses little or no risk.**
Yellow	**Moderate**	**51 to 100**	**Air quality is acceptable. However, there may be a risk for some people, particularly those who are unusually sensitive to air pollution.**
Orange	**Unhealthy for Sensitive Groups**	**101 to 150**	**Members of sensitive groups may experience health effects. The general public is less likely to be affected.**
Red	**Unhealthy**	**151 to 200**	**Some members of the general public may experience health effects; members of sensitive groups may experience more serious health effects.**
Purple	**Very Unhealthy**	**201 to 300**	**Health alert: The risk of health effects is increased for everyone.**
Maroon	**Hazardous**	**301 and higher**	**Health warning of emergency conditions: everyone is more likely to be affected.**

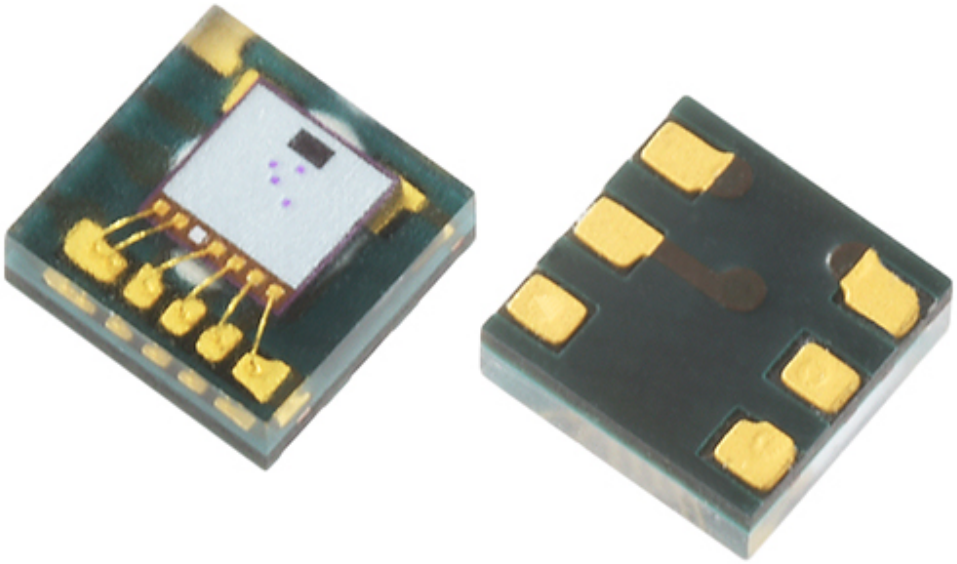
European Environment Agency (EEA)

Pollutant	Index level (based on pollutant concentrations in µg/m3)					
	Good	Fair	Moderate	Poor	Very poor	Extremely poor
Particles less than 2.5 µm ($PM_{2.5}$)	0-10	10-20	20-25	25-50	50-75	75-800
Particles less than 10 µm (PM_{10})	0-20	20-40	40-50	50-100	100-150	150-1200
Nitrogen dioxide (NO_2)	0-40	40-90	90-120	120-230	230-340	340-1000
Ozone (O_3)	0-50	50-100	100-130	130-240	240-380	380-800
Sulphur dioxide (SO_2)	0-100	100-200	200-350	350-500	500-750	750-1250

Ministry of Environment

Air Quality Index (AQI)							
AQI	O_3 (ppm) 8h	O_3 (ppm) 1-h (1)	$PM_{2.5}$ (µg/m³) 24-h	PM_{10} (µg/m³) 24-h	CO (ppm) 8-h	SO_2 (ppb) 1-h	NO_2 (ppb) 1-h
Good 0 ~ 50	0.000 - 0.054	-	0.0 - 15.4	0-50	0 - 4.4	0-20	0-30
Moderate 51 ~ 100	0.055 - 0.070	-	15.5 - 35.4	51-100	4.5 - 9.4	21-75	31-100
Unhealthy for Sensitive Groups 101 ~ 150	0.071 - 0.085	0.125 - 0.164	35.5 - 54.4	101-254	9.5 - 12.4	76-185	101-360
Unhealthy 151 ~ 200	0.086 - 0.105	0.165 - 0.204	54.5 - 150.4	255-354	12.5 - 15.4	186-304 (3)	361-649
Very Unhealthy 201 ~ 300	0.106 - 0.200	0.205 - 0.404	150.5 - 250.4	355-424	15.5 - 30.4	305-604 (3)	650-1249
Hazardous 301 ~ 400	(2)	0.405 - 0.504	250.5 - 350.4	425 - 504	30.5 - 40.4	605-804 (3)	1250-1649
Hazardous 401 ~ 500	(2)	0.505 - 0.604	350.5 - 500.4	505-604	40.5 - 50.4	805-1004 (3)	1650-2049

1.8 LTR390



The LTR390 optical sensor is a combined digital ambient light (ALS) and UVA sensor.

1.8.1 Ambient Light Specifications

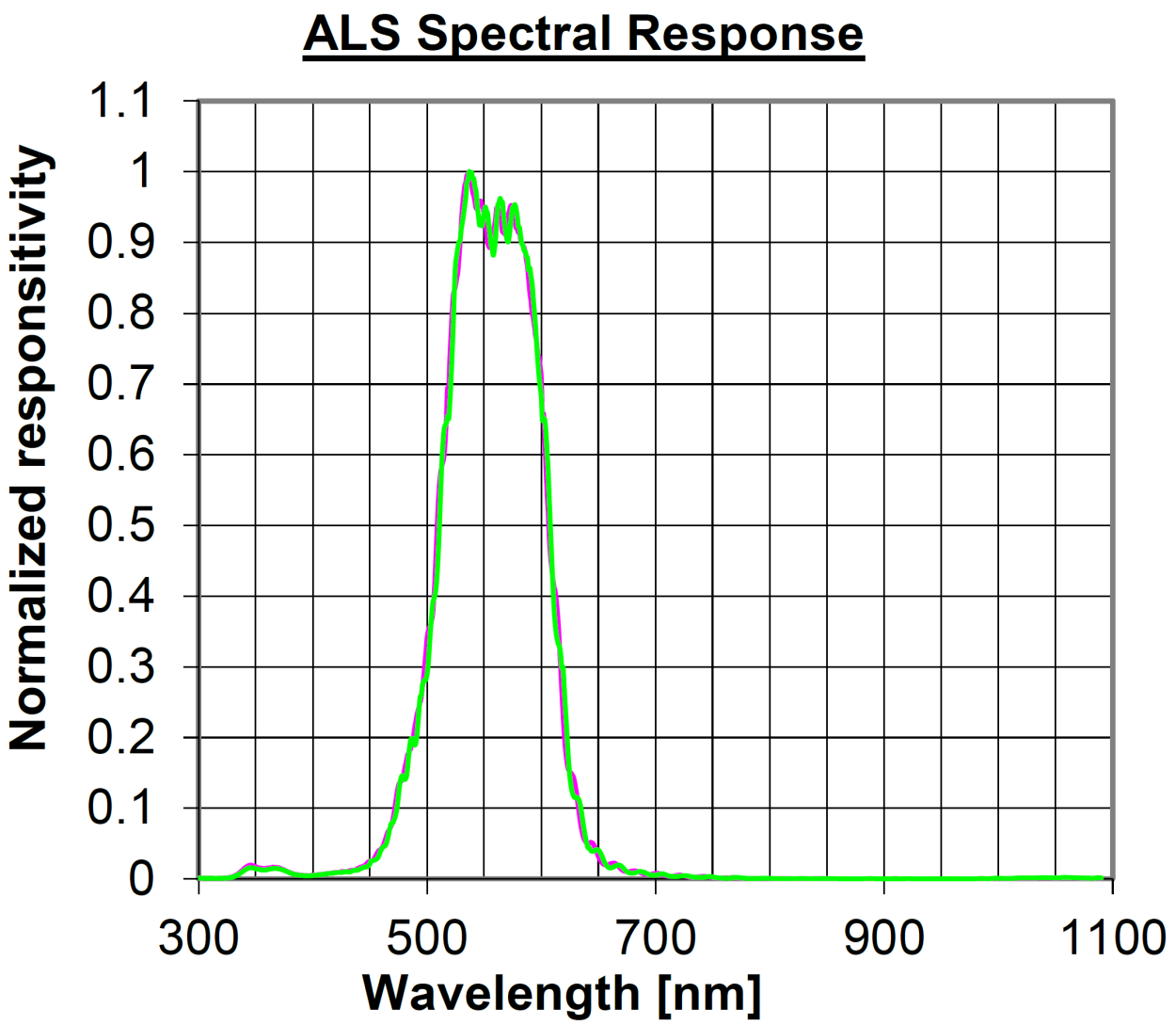
Parameter	Minimum	Typical	Maximum	Units	Condition
ALS Output Resolution	13	18	20	Bit	Programmable for 13, 16, 17, 18, 19, 20 bit
Dark Level Count	•	0	5	count	0 Lux, T_ope=25°C, 18-bit range
Calibrated Lux Error In Gain Range 3	-10	•	10	%	While LED, 5000K, T_ope=25°C
ALS Accuracy	-25	•	25	%	Across different light sources

1.8.2 UVS Specifications

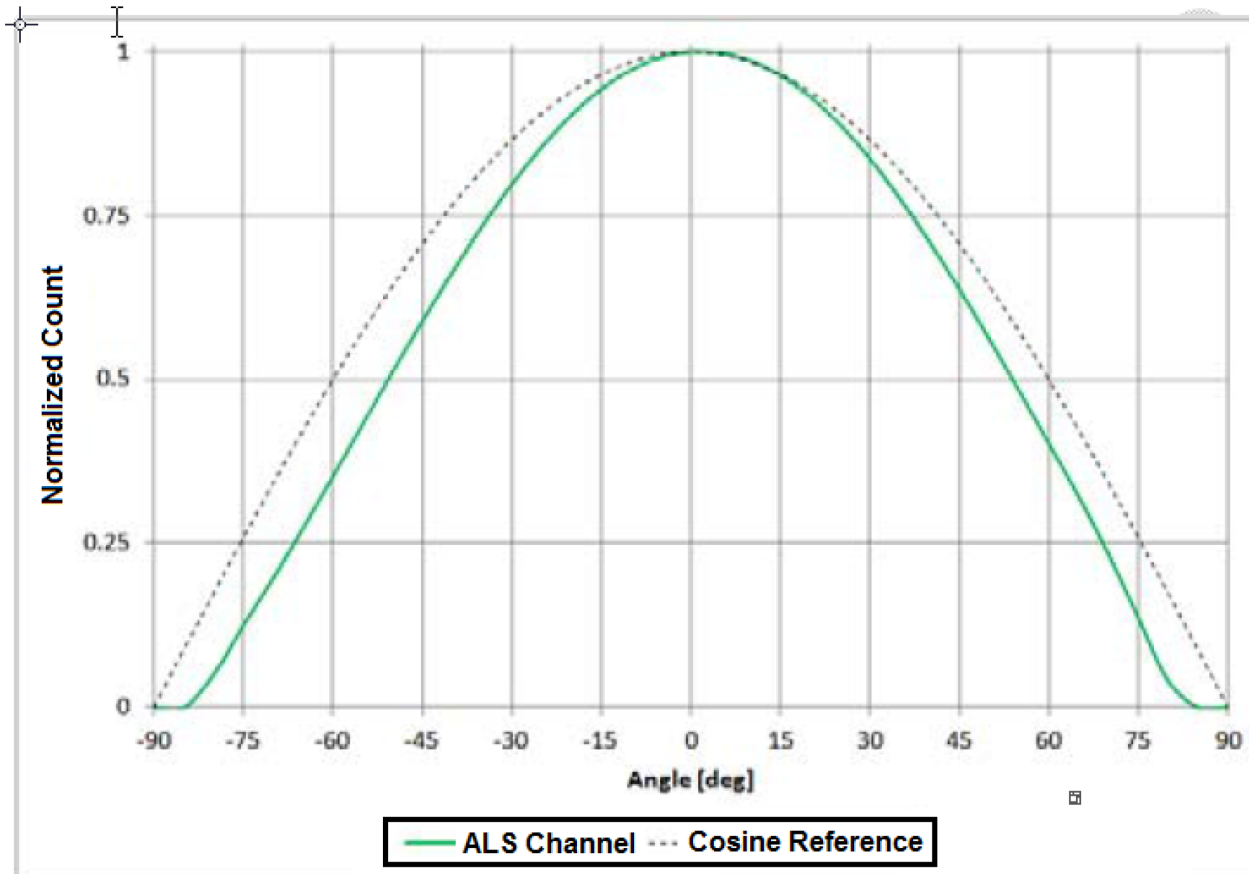
Parameter	Minimum	Typical	Maximum	Unit	Condition
UVS Output Resolution	13	18	20	Bit	Programmable for 13, 16,17, 18, 19, 20 bit
UV Count	•	160	•	count	UV LED 310nm, T_ope=25°C, 18-bit, Gain range = 18, Irradiance = 70uW/cm2
UV Sensitivity	•	2300	•	Counts/U	Gain range = 18, 20-bit
UVI Accuracy (UVI>5)	-20	•	20	%	Gain Range = 18, 20bit
UVI Accuracy (UVI<5)	-1	•	1	%	Gain Range = 18, 20bit

1.8.3 ALS Sensor

The ambient light sensor is sensitive to a range of 450nm - 650nm and is centered at 535nm as seen in the graph below.

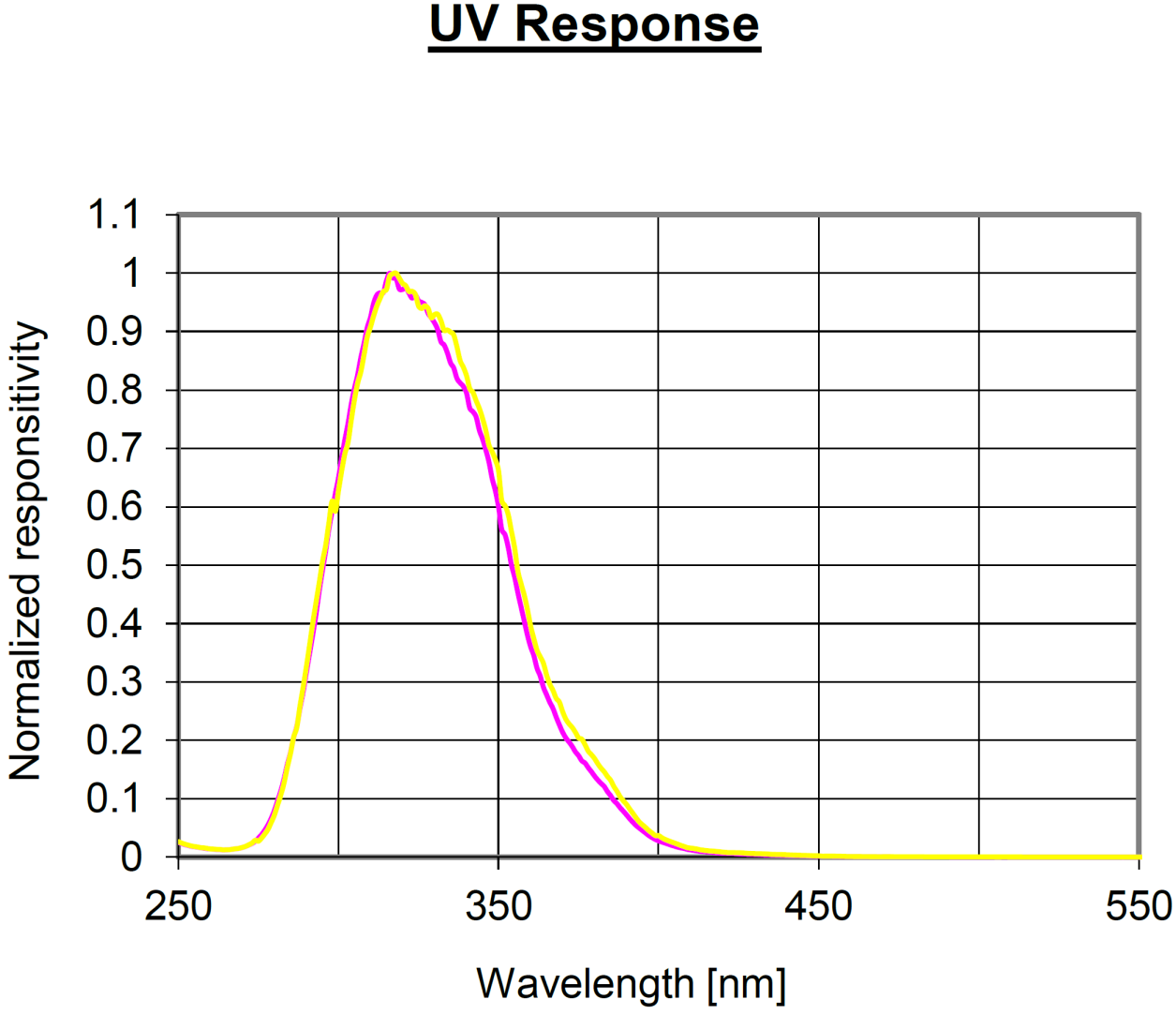


Additionally, below is a graph showing the angle of incidence of the sensor, where a normalized count of 1 indicates full sensitivity.



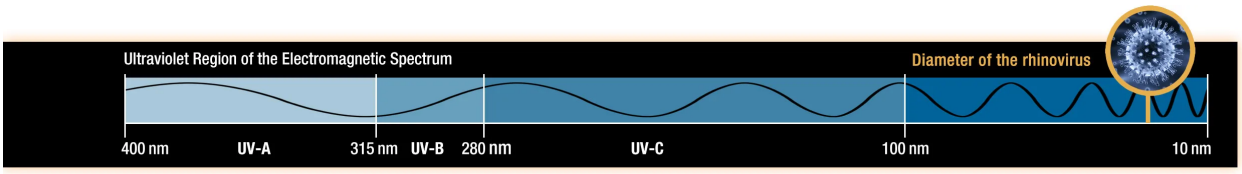
1.8.4 UVS Sensor

The UVS sensor peaks in sensitive for UVA wavelengths (315-400) with some moderate sensitivity to low-energy UVB wavelengths. The graph below indicates the the response of the UVS sensor, which has a range of 275nm - 400nm and is centered at 320nm.



1.8.5 The Ultra-Violet Spectrum

The ultra-violet spectrum ranges from 10nm to 400nm. This is further subdivided into different bands of intensity, which are used in remote sensing and to indicate their danger.



1.8.6 UVA

UVA is the longest wavelength of the UV spectrum, ranging from 400nm to 315nm. UVA is the primary cause of sunburns encountered from being in the sun too long. This is because the longer wavelength allows it to penetrate deeper into the skin.

1.8.7 UVB

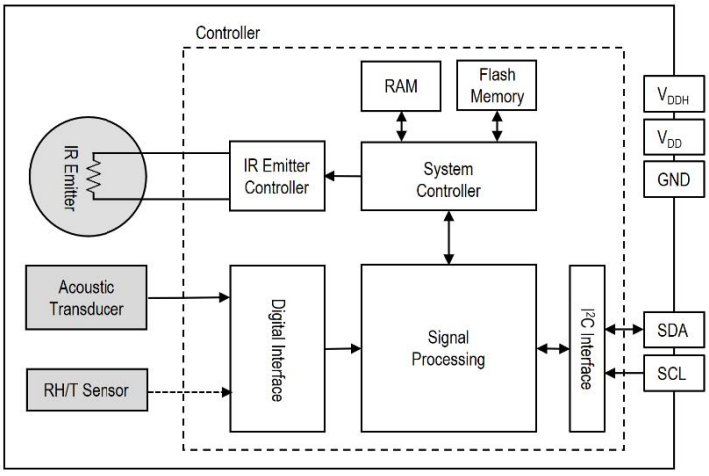
UVB is a the middle band of the UV spectrum and ranges from 315nm to 280nm. UVB is mostly absorbed by the ozone layer, but can still reach the earth's surface. This is primarily seen at higher latitudes and elevations. Because UVB has a shorter wavelength, this is the primary cause of skin cancer and blistering from sunburns.

1.8.8 UVC

UVB is the highest energy band in the UV spectrum, ranging from 280nm to 100nm. UVC is entirely filtered out by the Earth's atmosphere, so is no danger to life on the surface, but can become a risk during air travel or in space. UVC is a type of ionizing radiation, meaning it easily kills cells. This is useful for UV disinfection lights.

1.9 SCD40

THe SCD40 CO2 sensor uses photoacoustic non-dispersive infrared (NDIR) technology to measure actual CO2 concentration, unlike the eCO2 measurement on the *ENS160*. This works by measuring the attenuation of an infrared light shining through the gas onto an acoustic transducer. Along with the sample chamber there is a reference nitrogen sample used to compare the CO2 measurement to the known gas. This process works according to Beer-Lambert law which is this attenuation of the gas.



1.9.1 CO2 Sensing Performance

Parameter	Conditions	Value
Output range •		0 - 40,000ppm
Accuracy	400ppm - 2,000ppm	±(50ppm + 5% of reading)
Repeatability	Typical	±10ppm

1.9.2 Humidity Sensing Performance

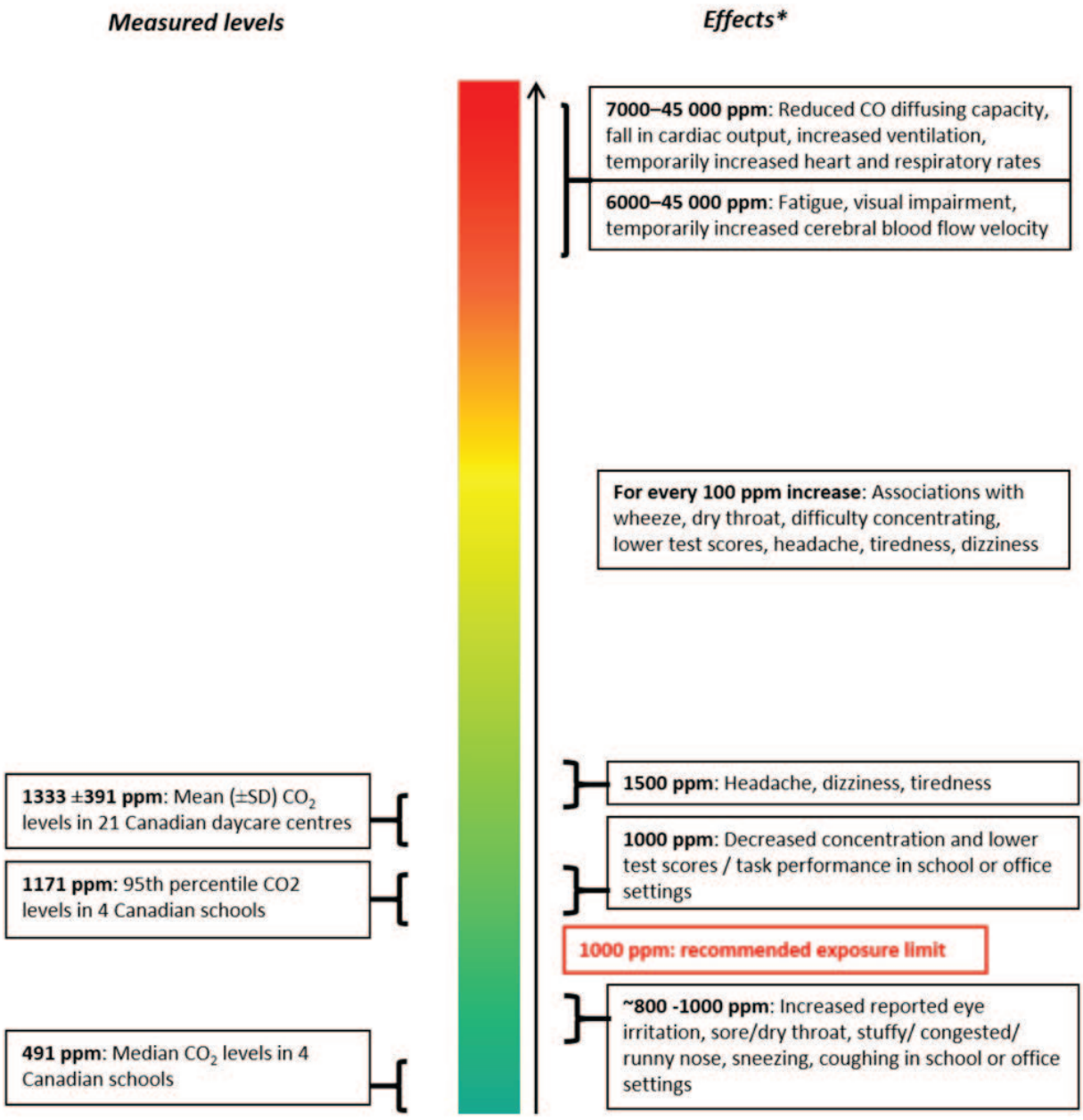
Parameter	Conditions	Value
Range •		0 %RH - 100 %RH
Accuracy	15°C - 35°C, 20 %RH - 65 %RH	±6 %RH
Accuracy	-10°C - 60°C, 0 %RH - 100 %RH	±9 %RH
Repeatability	Typical	±0.4 %RH

1.9.3 Temperature Sensing Performance

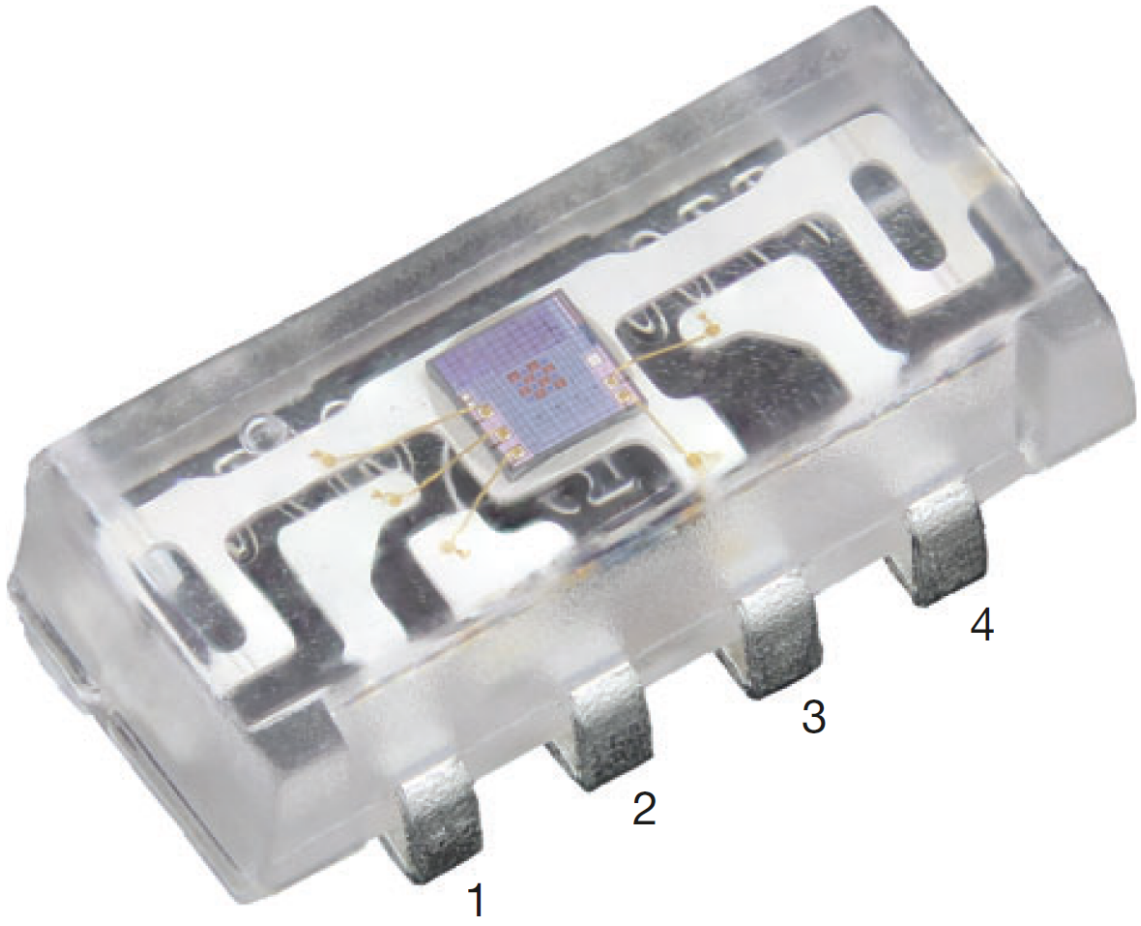
Parameter	Conditions	Value
Range •		-10°C - 60°C
Accuracy	15°C - 35°C	±0.8°C
Accuracy	-10°C - 60°C	±1.5°C
Repeatability	Typical	±0.1°C

1.9.4 Measuring CO2

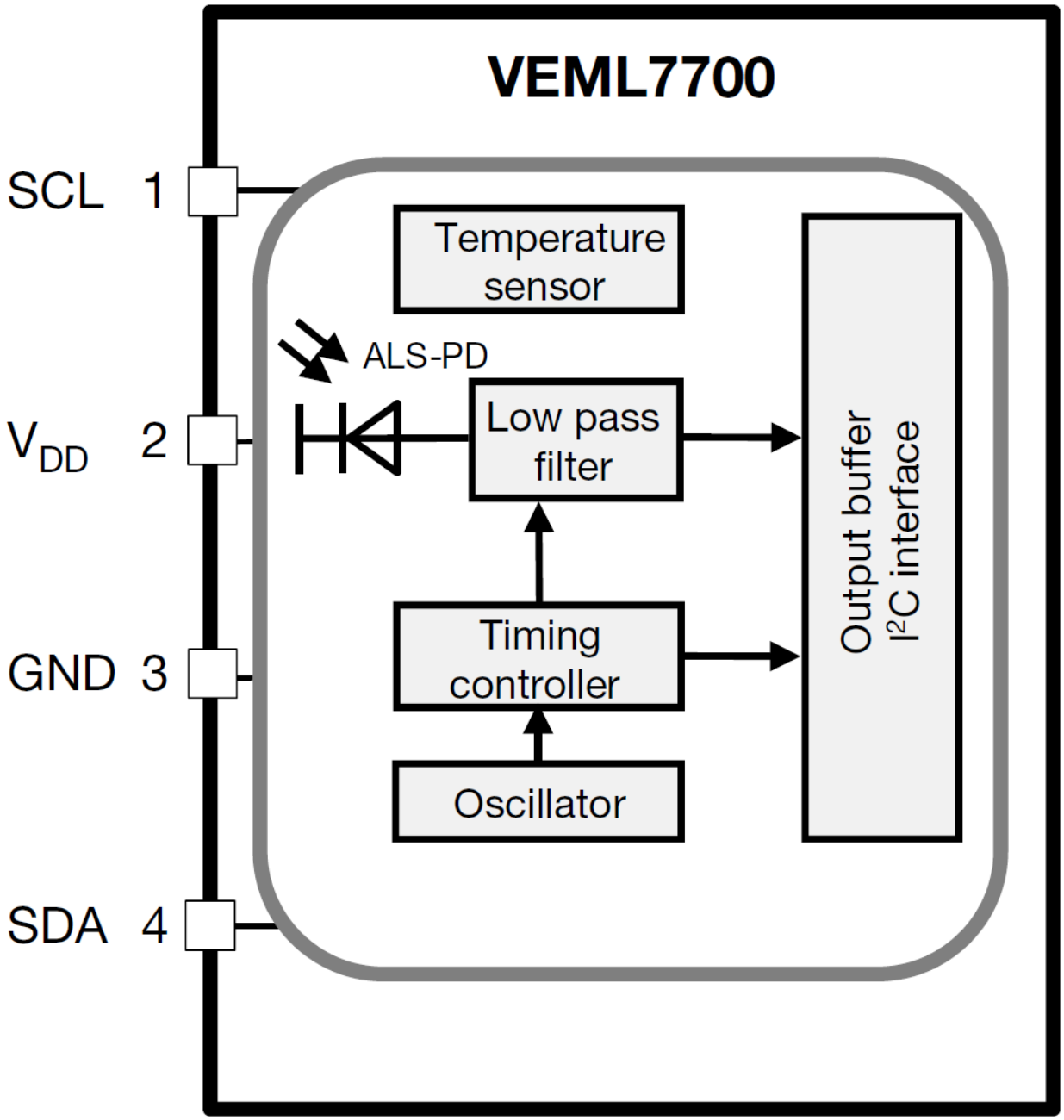
Measuring CO2 is another important parameter for indoor and outdoor air quality. Health Canada recommends not exceeding 1000ppm of CO2 in a 24 hour period. A high concentration of CO2 can increase the risk of respiratory symptoms, decreased cognitive function, and neurophysiological symptoms such as headaches, tiredness, and dizziness. The graph below shows different concentrations and what effects one may encounter at that concentration.


Measured levels
Effects*
7000–45 000 ppm: Reduced CO diffusing capacity, fall in cardiac output, increased ventilation, temporarily increased heart and respiratory rates
6000–45 000 ppm: Fatigue, visual impairment, temporarily increased cerebral blood flow velocity
For every 100 ppm increase: Associations with wheeze, dry throat, difficulty concentrating, lower test scores, headache, tiredness, dizziness
1333 ±391 ppm: Mean (±SD) CO2 levels in 21 Canadian daycare centres
1171 ppm: 95th percentile CO2 levels in 4 Canadian schools
491 ppm: Median CO2 levels in 4 Canadian schools
1500 ppm: Headache, dizziness, tiredness
1000 ppm: Decreased concentration and lower test scores / task performance in school or office settings
1000 ppm: recommended exposure limit
~800 -1000 ppm: Increased reported eye irritation, sore/dry throat, stuffy/ congested/ runny nose, sneezing, coughing in school or office settings

1.10 VEML7700



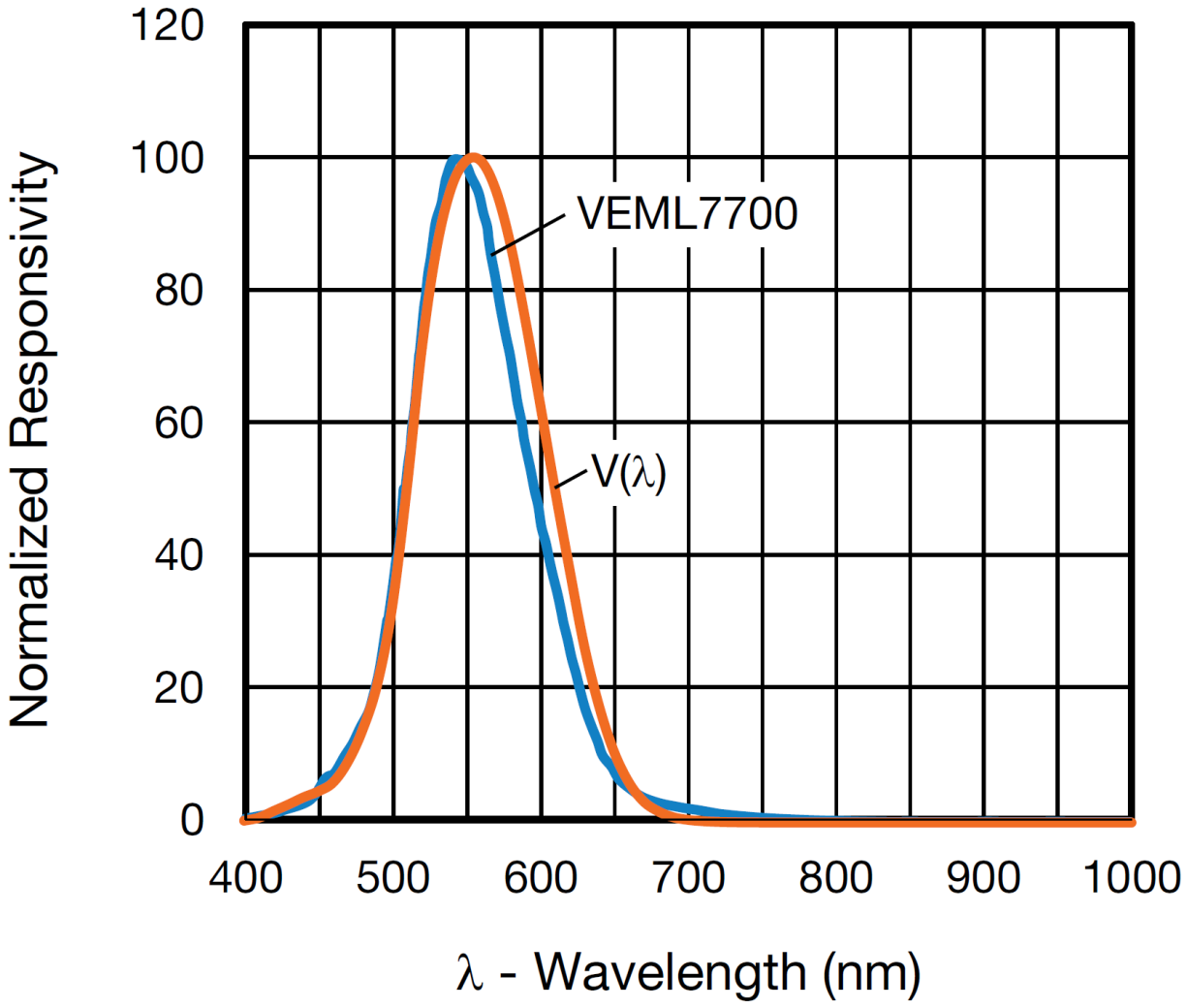
The VEML7700 high accuracy 16-bit light sensor can be used to precisely measure ambient light. Below is a functional block diagram of the device, where the sensor is an ALS photodiode.



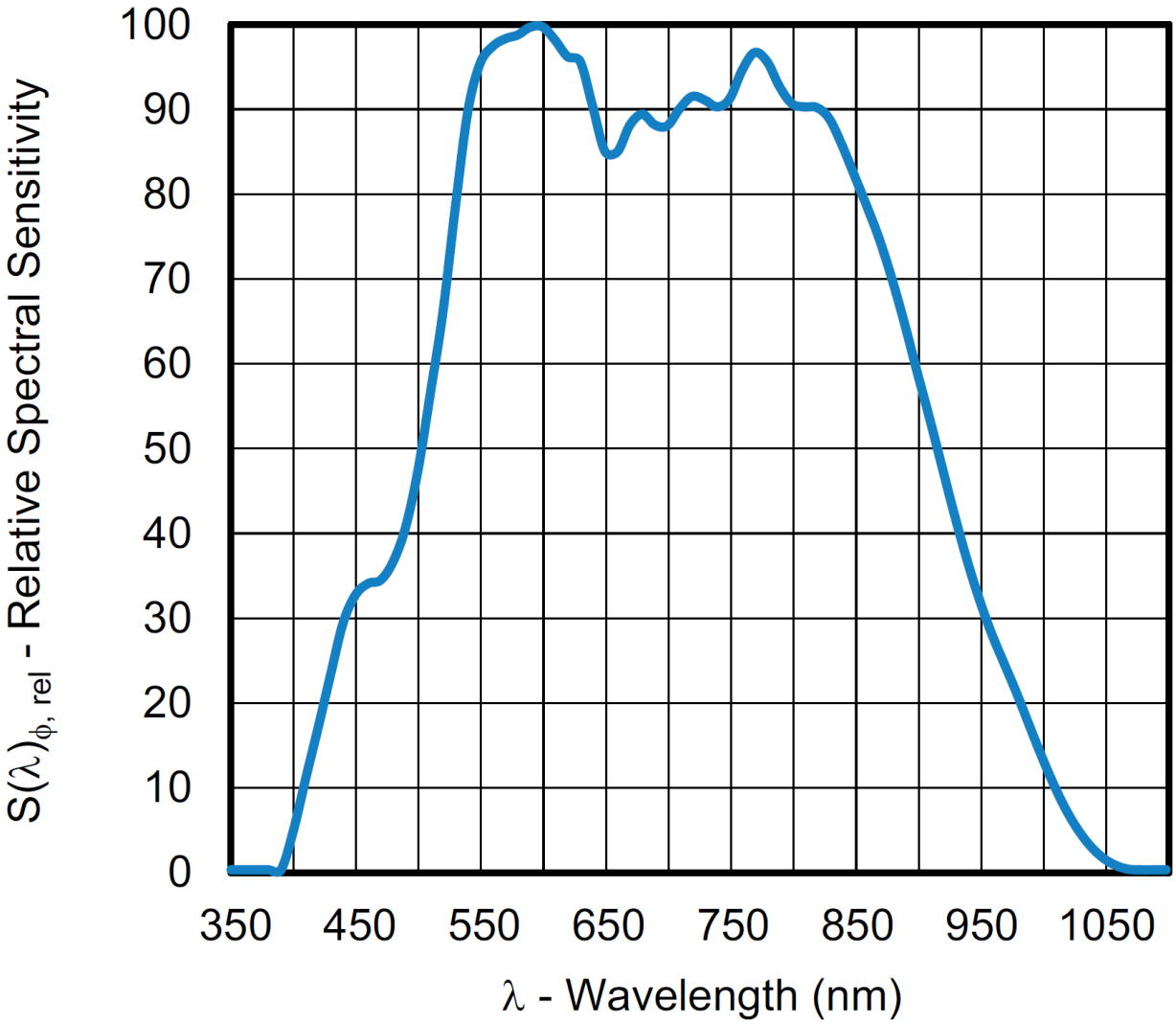
1.10.1 Specifications

Parameter	Typical Value	Units
Digital resolution	0.0036	lux/bit
Detectable minimum illuminance	0.0072	lux
Detectable maximum illuminance	120,000	lux
Dark offset	3	steps

Below is a graph showing the spectral response of the sensor. It is sensitive to wavelengths ranging from 450nm - 650nm, normalized at 550nm.



Additionally, below is a graph showing the sensitivity of the white light channel on the sensor.

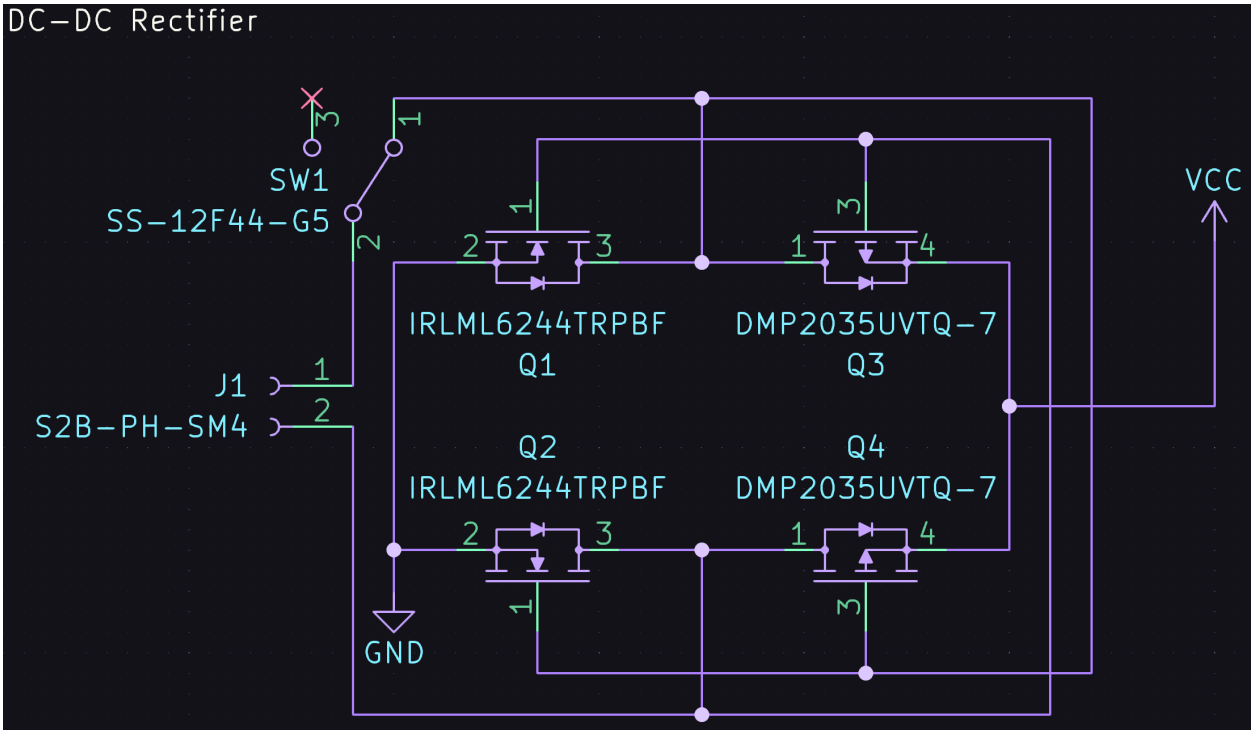


1.11 Overview of Hardware

The Destination Weather Station was designed in KiCad and assembled by hand. the weather station includes several features that improve user experience. These are:

1.11.1 Power

The power subsystem of the weather station was designed with reverse voltage protection in mind. Typically, if a battery is connected to a circuit backwards, it will not work, or worse, fry the entire system. This is why reverse current protection is often added to circuits to prevent such user error. This allows the user to not have to worry about the polarity of their LiPo battery. On the Destination Weather Station v4.5 we have taken this one step further, by not only having reverse voltage/polarity protection, but also correcting this polarity. This works using the same principal as an AC rectifier circuit. This DC rectifier consists of two N-Channel MOSFETs and two P-Channel MOSFETs. These MOSFETs combine to essentially create a rectifier circuit, which will only allow the correct polarity to pass through the FETs. A schematic of this circuit can be found below.



Note: There is no over-current protection, so make sure to not exceed 5V in.

1.11.2 Storage

The storage on the Destination Weather station is simply a microSD card slot to store recorded sensor data for further evalulation. There is also chip-detect functionality which allows the code to know if a card has been inserted or not.

1.11.3 Human Interface Devices

On the Destination Weather Station there are 4 buttons, which are the primary way to interact with the device. These buttons are each labeled with arrows ↑, ↓, ←, →. These buttons are used to scroll through the menus and to switch between the different data displays.

1.12 Troubleshooting

Coming soon!

1.13 Overview of Software

Software for the Destination Weather Station v4.5.

1.13.1 Blink Sketch

The Blink sketch is used as a beginners program and to test the on-board RGB LED and NeoPixel.

Libraries

The Blink Sketch has only one required library. This library is the Adafruit NeoPixel Library. This is used to send commands to the NeoPixel RGB LED on the Seeeduino XIAO.

```
#include <Adafruit_NeoPixel.h>
```

Hardware Definitions

Several hardware definitions need to be defined to use the RBG LED and the NeoPixel. These are defined by assigning each variable a General Purpose Input-Output (GPIO) pin on the XIAO and any other criteria specified by the NeoPixel library. What these definitions do is define the red, green, and blue pins of the normal RGB LED a GPIO pin, define the power and signal pin for the NeoPixel, and define the number of NeoPixels on the board.

```
#define RGB_R 17
#define RGB_G 16
#define RGB_B 25
#define NEOPIXEL_PWR 11
#define NEOPIXEL_PIN 12
#define NEOPIXEL_NUM 1
```

Following the hardware definitions, the information specified above needs to be defined in the NeoPixel definition function

```
Adafruit_NeoPixel pixels(NEOPIXEL_NUM, NEOPIXEL_PIN, NEO_GRB + NEO_KHZ800);
```

Setup Function

The next step is to initialize the LEDs for the primary loop function. This is done by starting the NeoPixel function, initializing the power and signal pins, and defining the RGB LEDs pins as outputs.

```
void setup() {
   //Initialize NeoPixel
   pixels.begin();
   pinMode(NEOPIXEL_PWR, OUTPUT);
   digitalWrite(NEOPIXEL_PWR, HIGH);

   //Initialize RGB LED
   pinMode(RGB_R, OUTPUT);
   pinMode(RGB_G, OUTPUT);
   pinMode(RGB_B, OUTPUT);
}
```

Loop Function

The final step is to send the commands to the NeoPixel and RGB LED. This is done by first clearing the NeoPixel's stored data, defining the color, and sending the neopixel the data. Additionally, there is a custom function to easily control the color of the RGB LED. This works by passing the red, green, and blue values to the function `setColor()`. An example of the NeoPixel code and the RGB LED can be found below

```
pixels.clear();
pixels.setPixelColor(0, pixels.Color(15, 25, 205));
delay(400); // How long the NeoPixel stays on
pixels.show();
```

```
setColor(15, 25, 205);

void setColor(int VAL_GRN, int VAL_RED, int VAL_BLU){
   analogWrite(RGB_R, 255 - VAL_RED);
   analogWrite(RGB_G, 255 - VAL_GRN);
   analogWrite(RGB_B, 255 - VAL_BLU);
}
```

Full Code

Below is the full Blink example sketch

```
#include <Adafruit_NeoPixel.h> // Import Adafruit NeoPixel library

//Define Hardware
#define RGB_R 17
#define RGB_G 16
#define RGB_B 25
#define NEOPIXEL_PWR 11
#define NEOPIXEL_PIN 12
#define NEOPIXEL_NUM 1

Adafruit_NeoPixel pixels(NEOPIXEL_NUM, NEOPIXEL_PIN, NEO_GRB + NEO_KHZ800); // Define␣
↪NeoPixel

void setup() {
   //Initialize NeoPixel
   pixels.begin();
   pinMode(NEOPIXEL_PWR, OUTPUT);
   digitalWrite(NEOPIXEL_PWR, HIGH);

   //Initialize RGB LED
   pinMode(RGB_R, OUTPUT);
   pinMode(RGB_G, OUTPUT);
   pinMode(RGB_B, OUTPUT);
}

void loop() {
   pixels.clear();
   pixels.setPixelColor(0, pixels.Color(15, 25, 205));
   setColor(15, 25, 205);
   delay(400);

   pixels.show();
   pixels.clear();
   pixels.setPixelColor(0, pixels.Color(103, 25, 205));
   setColor(15, 25, 205);
   delay(400);

   pixels.show();
   pixels.clear();
   pixels.setPixelColor(0, pixels.Color(233, 242, 205));
   setColor(233, 242, 205);
```

(continues on next page)

(continued from previous page)

```
   delay(400);

   pixels.show();
   pixels.clear();
   pixels.setPixelColor(0, pixels.Color(233, 23, 23));
   setColor(233, 23, 23);
   delay(400);

   pixels.show();
   pixels.clear();
   pixels.setPixelColor(0, pixels.Color(12, 66, 101));
   setColor(12, 66, 101);
   delay(400);

   pixels.show();
   delay(500);
}

//Function to change color of RGB LED
void setColor(int VAL_GRN, int VAL_RED, int VAL_BLU){
   analogWrite(RGB_R, 255 - VAL_RED);
   analogWrite(RGB_G, 255 - VAL_GRN);
   analogWrite(RGB_B, 255 - VAL_BLU);
}
```

1.13.2 Sensor Test Sketch

The Sensor Test sketch is used to make sure the sensors on the weather station are working correctly. Use this sketch as the first program to upload after the Blink sketch.

Libraries

For the Sensor Test sketch, many more libraries are required. These can be split into two categories; System libraries and sensor libraries. In this example only one system library is required, which is the Wire Library. This is a default Arduino library and is used to communicate with sensors on the Inter-Integrated Circuit (I2C) protocol. More details about the protocol can be found here.

The libraries required to use all of the sensors on your weather stations and for the classroom expansion sensors have been pre-installed. A list of the libraries are below.

```
//Import system libraries
#include <Wire.h>

//Import sensor libraries
#include <Adafruit_BME280.h>
#include <Adafruit_LTR390.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_VEML7700.h>
#include <ScioSense_ENS160.h>
#include <SensirionI2CScd4x.h>
```

Hardware Definitions

Only one hardware definition is required for this example. This defines mean sea-level pressure, which is used to calculate an approximate value for altitude using temperature and pressure. By default, this value is set to 1013.25 hPa. You are welcome to change this value to the current sea-level pressure in your region. Click `here <>`_ to find out more about calculating barometric altitude.

```
#define SEALEVELPRESSURE_HPA (1013.25)
```

Define Sensor Functions

Several sensor functions need to be defined to get the sensors to work. This ensures that each sensor is communicating to the correct address. These are also used to define which variable you will be using to communicate with the sensor.

```
Adafruit_BME280 BME280;
Adafruit_LTR390 LTR390 = Adafruit_LTR390();
Adafruit_VEML7700 VEML7700 = Adafruit_VEML7700();
ScioSense_ENS160 ENS160(ENS160_I2CADDR_0);
SensirionI2CScd4x scd4x;
```

Define Data Types

The next step is to define the data types of each variable. A data type is a way to standardize the formatting of a variable, such as if it is a text value, integer, boolean (True/False), or number with decimal places. Click here for the definitions of C++ fundamental types.

Below you can find the data types of the variables used in this example program. The version in the final code file is condensed to save on line-count.

```
float BME280_TEMP;      // Temperature
float BME280_PRES;      // Pressure
float BME280_HUMD;      // Humidity
float BME280_HI;        // Heat-index
float BME280_ALT;       // Barometric altitude
float ENS160_AQI;       // Air Quality Index
float ENS160_AQI_PREV;  // Previous AQI value
float ENS160_TVOC;      // Total Volatile Organic Compounds
float ENS160_eCO2;      // Calculated CO2 concentration
float LTR390_UVI;       // Ultra-Violet Index
float LTR390_RAW;       // Raw UV data
float VEML7700_ALS;     // Ambient light data
float VEML7700_WHITE;   // How much of the incoming light is white
float VEML7700_LUX;     // Ambient light intensity
float SCD40_CO2;        // Measured CO2 concentration
float SCD40_TEMP;       // Temperature data from the CO2 sensor
float SCD40_HUMD;       // Humidity data from the CO2 sensor

int clock_timer;        // Time between data recordings
int hh;                 // hours elapsed
int mm;                 // minutes elapsed
int ss;                 // seconds elapsed

uint16_t SCD40_CO2_RAW; // Raw CO2 data
```

Setup Function

The next section is for the setup function. This is where all of the sensors are initialized and confirmed if they are connected or not. Each sensor has a slightly different initialization process, which is determined by the sensor libraries. Essentially, each sensor is checked to see if it is attached; if it is not, then it's status indicator is set to 0/False. Lastly, a header for all of the data parameters is printed to the serial monitor.

```
void setup() {
   Serial.begin(115200); // Set serial stream to 115200bits/s
   while(!Serial); // Wait until serial monitor is open

   //Initialize BME280 sensor
   if(!BME280.begin(0x76, &Wire)){
      Serial.print("\n\nBME280 not found");
   }

   //Initialize ENS160 sensor
   if(ENS160.begin()){
      if(!ENS160.setMode(ENS160_OPMODE_STD)){
         Serial.print("\n\nENS160 failed to init");
         ENS160_STS = 0;
      }
      delay(10);
      if(ENS160.available()){
         ENS160.measure(true);
         ENS160_AQI_PREV = ENS160.getAQI();
      }
   }
   else{
      Serial.print("\n\nENS160 not found");
      ENS160_STS = 0;
   }

   //Initialize LTR390 sensor
   if(!LTR390.begin()){
      Serial.print("\n\nLTR390 not found");
      LTR390_STS = 0;
   }
   else{
      LTR390.setMode(LTR390_MODE_UVS);
      LTR390.setGain(LTR390_GAIN_3);
      LTR390.setResolution(LTR390_RESOLUTION_16BIT);
      LTR390.setThresholds(100,1000);
      LTR390.configInterrupt(true, LTR390_MODE_UVS);
   }

   //Initialize VEML7700 sensor
   if(!VEML7700.begin()){
      Serial.print("\n\nVEML7700 not found");
      VEML7700_STS = 0;
   }
   else{
      VEML7700.setLowThreshold(10000);
      VEML7700.setHighThreshold(20000);
```

(continues on next page)

(continued from previous page)

```
      VEML7700.interruptEnable(true);
   }

   //Initialize SCD40 sensor
   scd4x.begin(Wire);
   if(scd4x.stopPeriodicMeasurement() || scd4x.startPeriodicMeasurement()){
         Serial.print("\n\nSCD40 failed to respond");
         SCD40_STS = 0;
   }
   //Print data table header
   Serial.print("\n\
↪n+=============================================================================================+\n| ␣
↪TIME  | TEMP | HUM |  HI  | PRES | ALT | CO2 | TVOC |  AQI  |  UVI  |  LUX  |\
↪n|hh:mm:ss| (°C) | (%) | (°C) | hPa  | (m) |(ppm)|(ppb.)|(0-300)|(0-+11)|(k-lux)|\
↪n+=============================================================================================+");

   //Set timer variables to zero
   clock_timer = 0;
   hh = 0;
   mm = 0;
   ss = 0;
}
```

Loop Function

The loop function is the main portion of the software where the data is collected and displayed. First, the variable for the delay_timer is set to be equal to the current processor clock time (in milliseconds). This is to help the code make sure it is refreshing at the correct rate.

```
int delay_timer = millis();
```

The next step is to read from all the individual sensors and preform any relevant calculations. Again, each sensor has it's own method to collect data.

BME280

The Adafruit library for the BME280 makes it simple to collect data. It is a simple read function where you define the target variable and assign the data value to it.

```
BME280_TEMP = BME280.readTemperature();
BME280_PRES = BME280.readPressure() / 100.0F;
BME280_HUMD = BME280.readHumidity();
BME280_ALT = BME280.readAltitude(SEALEVELPRESSURE_HPA);
```

Now, the data can be interpreted to get the desired weather parameters. With the BME280 weather sensor several parameters can be calculated. The ones featured in this program are dew point, absolute humidity, and heat index. Only the calculation for heat index is active by default, but you can uncomment the dew point and absolute humidity calculations to include them.

Heat index is calculated using dry-bulb temperature and relative humidity. The formula used to calculate heat index was first derived in The Assessment of Sultriness. Part I: A Temperature-Humidity Index Based on Human Physiology and Clothing Science by R.G. Steadman in 1979. His results were then interpreted by the National Weather Service in technical report SR 90-23 in 1990 to be approximated into an equation. $HI = -42.379 + 2.04901523 \cdot T +$

$10.14333127 \cdot RH - 0.22375541 \cdot T \cdot RH - 6.83783 \cdot 10^(-3) \cdot T^2 - 5.481717 \cdot 10^(-2) \cdot RH^2 + 1.22874 \cdot 10^(-3) \cdot T^2 \cdot RH + 8.5282 \cdot 10 \cdot (-4) \cdot T \cdot RH^2 - 1.99 \cdot 10^(-6) \cdot T^2 \cdot RH^2$ Where T is temperature in Fahrenheit and RH is relative humidity.

Because this is a regression fit, the equation has an error of ± 1.3°F.

```
float h = (log10(BME280_HUMD)-2.0)/0.4343+(17.62*BME280_TEMP)/(243.12+BME280_TEMP);

//Uncomment to calculate dew point and absolute humidity
//float BME280_DEW_POINT = 243.12*h/(17.62-h); // Calculate dew point
//float BME280_ABSOLUTE_HUMIDITY = 216.7*(BME280_HUMD/100.0*6.112*exp(17.62*BME280_TEMP/
↪(243.12+BME280_TEMP))/(275.15+BME280_TEMP)); // Calculate absolute humidity

float BME280_TEMP_F = (1.8*BME280_TEMP) + 32.0; // Convert Celsius to Fahrenheit

//Calculate heat-index
float BME280_HI_F = -42.379 + 2.04901523*BME280_TEMP_F + 10.14333127*BME280_HUMD - 0.
↪22475541*BME280_TEMP_F*BME280_HUMD - 0.00683783*BME280_TEMP_F*BME280_TEMP_F - 0.
↪05481717*BME280_HUMD*BME280_HUMD + 0.00122874*BME280_TEMP_F*BME280_TEMP_F*BME280_HUMD␣
↪+ 0.00085282*BME280_TEMP_F*BME280_HUMD*BME280_HUMD - 0.00000199*BME280_TEMP_F*BME280_
↪TEMP_F*BME280_HUMD*BME280_HUMD; // Initial heat-index value

if(BME280_HUMD < 13.0 && BME280_TEMP_F > 80.0 && BME280_TEMP_F < 112.0){ // If RH < 13%␣
↪and 80 < T < 112
   BME280_HI_F = BME280_HI_F + (13.0 - BME280_HUMD)/4.0*sqrtf((17.0 - abs(BME280_TEMP_F -
↪ 95.0))/17);
}
else if(BME280_HUMD > 85 && BME280_TEMP_F > 80.0 && BME280_TEMP_F < 87.0){ // If RH > 85
↪% and 80 < T < 87
   BME280_HI_F = BME280_HI_F + (BME280_HUMD - 85.0)/10*(87 - BME280_TEMP_F)/5;
}
else if(BME280_HI_F < 80.0){ // If HI < 80
   BME280_HI_F = 0.5 * (BME280_HI_F + 61.0 + 1.2*(BME280_HI_F - 68.0) + BME280_HUMD*0.
↪094);
}

BME280_HI = (5.0/9.0)*(BME280_HI_F - 32.0); // Convert back to Celsius
```

ENS160

The data collecting method for the ENS160 is very similar to the BME280. The parameters being collected are Air Quality Index (AQI) calculated from the German Federal Environmental Agency model, which is on a scale from 1 - 5, total volatile organic compounds (TVOC), and a calculated measurement of approximate Carbon-Dioxide concentration (eCO2).

```
ENS160.measure(true);
ENS160_AQI = ENS160.getAQI();
ENS160_TVOC = ENS160.getTVOC();
ENS160_eCO2 = ENS160.geteCO2();
```

No further calculations need to be completed for this sensor.

LTR390

Just like before, only the data needs to be pulled from the sensor with minimal calculations.

```
LTR390_RAW = LTR390.readUVS();
LTR390_UVI = LTR390_RAW / 2300.00;
```

VEML7700

```
VEML7700_LUX = VEML7700.readLux();
```

SCD40

```
bool isDataReady = false; // Reset Data-Ready flag
if(scd4x.getDataReadyFlag(isDataReady)){
}

if(isDataReady){
   if(scd4x.readMeasurement(SCD40_CO2_RAW, SCD40_TEMP, SCD40_HUMD)){
   }
   else if(SCD40_CO2_RAW == 0){
   }
   else{
      SCD40_CO2 = static_cast<float>(SCD40_CO2_RAW);
   }
}
else{
}
```

Full Code

Below is the full Sensor Test example sketch

```
//Import system libraries
#include <Wire.h>

//Import sensor libraries
#include <Adafruit_BME280.h>
#include <Adafruit_LTR390.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_VEML7700.h>
#include <ScioSense_ENS160.h>
#include <SensirionI2CScd4x.h>

#define SEALEVELPRESSURE_HPA (1013.25) //Define sea-level pressure as 1013.25hPa

//Set sensor functions
Adafruit_BME280 BME280;
Adafruit_LTR390 LTR390 = Adafruit_LTR390();
Adafruit_VEML7700 VEML7700 = Adafruit_VEML7700();
ScioSense_ENS160 ENS160(ENS160_I2CADDR_0);
SensirionI2CScd4x scd4x;
```

(continues on next page)

(continued from previous page)

```
//Define data types
float BME280_TEMP, BME280_PRES, BME280_HUMD, BME280_HI, BME280_ALT, ENS160_AQI, ENS160_
↪AQI_PREV, ENS160_TVOC, ENS160_eCO2, LTR390_UVI, LTR390_RAW, VEML7700_ALS, VEML7700_
↪WHITE, VEML7700_LUX, SCD40_CO2, SCD40_TEMP, SCD40_HUMD;
bool ENS160_STS = 1;
bool LTR390_STS = 1;
bool VEML7700_STS = 1;
bool SCD40_STS = 1;
int clock_timer, hh, mm, ss;
uint16_t SCD40_CO2_RAW;

float TVOC_CONCENTRATION_MIN = 5000;
float TVOC_CONCENTRATION_MAX = 0;

void setup() {
Serial.begin(115200); // Set serial stream to 115200bits/s
while(!Serial); // Wait until serial monitor is open

//Initialize BME280 sensor
if(!BME280.begin(0x76, &Wire)){
   Serial.print("\n\nBME280 not found");
}

//Initialize ENS160 sensor
if(ENS160.begin()){
   if(!ENS160.setMode(ENS160_OPMODE_STD)){
      Serial.print("\n\nENS160 failed to init");
      ENS160_STS = 0;
   }
   delay(10);
   if(ENS160.available()){
      ENS160.measure(true);
      ENS160_AQI_PREV = ENS160.getAQI();
   }
}
else{
   Serial.print("\n\nENS160 not found");
   ENS160_STS = 0;
}

//Initialize LTR390 sensor
if(!LTR390.begin()){
   Serial.print("\n\nLTR390 not found");
   LTR390_STS = 0;
}
else{
   LTR390.setMode(LTR390_MODE_UVS);
   LTR390.setGain(LTR390_GAIN_3);
   LTR390.setResolution(LTR390_RESOLUTION_16BIT);
   LTR390.setThresholds(100,1000);
   LTR390.configInterrupt(true, LTR390_MODE_UVS);
```

(continues on next page)

(continued from previous page)

```
}

//Initialize VEML7700 sensor
if(!VEML7700.begin()){
   Serial.print("\n\nVEML7700 not found");
   VEML7700_STS = 0;
}
else{
   VEML7700.setLowThreshold(10000);
   VEML7700.setHighThreshold(20000);
   VEML7700.interruptEnable(true);
}

//Initialize SCD40 sensor
scd4x.begin(Wire);
if(scd4x.stopPeriodicMeasurement() || scd4x.startPeriodicMeasurement()){
      Serial.print("\n\nSCD40 failed to respond");
      SCD40_STS = 0;
}

//Print data table header
Serial.print("\n\
↪n+=========================================================================================+\n| ␣
↪TIME  | TEMP | HUM |  HI  | PRES | ALT | CO2 | TVOC |  AQI  |  UVI  |  LUX  |\
↪n|hh:mm:ss| (°C) | (%) | (°C) | hPa  | (m) |(ppm)|(ppb.)| (1-5) |(0-+11)|(k-lux)|\
↪n+=========================================================================================+");

//Set timer variables to zero
clock_timer = 0;
hh = 0;
mm = 0;
ss = 0;
}

void loop() {
int delay_timer = millis(); // Reset delay timer

//Collect Data from BME280
BME280_TEMP = BME280.readTemperature();
BME280_PRES = BME280.readPressure() / 100.0F;
BME280_HUMD = BME280.readHumidity();
BME280_ALT = BME280.readAltitude(SEALEVELPRESSURE_HPA);

float h = (log10(BME280_HUMD)-2.0)/0.4343+(17.62*BME280_TEMP)/(243.12+BME280_TEMP);

//Uncomment to calculate dew point and absolute humidity
//float BME280_DEW_POINT = 243.12*h/(17.62-h); // Calculate dew point
//float BME280_ABSOLUTE_HUMIDITY = 216.7*(BME280_HUMD/100.0*6.112*exp(17.62*BME280_TEMP/
↪(243.12+BME280_TEMP))/(275.15+BME280_TEMP)); // Calculate absolute humidity

float BME280_TEMP_F = (1.8*BME280_TEMP) + 32.0; // Convert Celsius to Fahrenheit
```

(continues on next page)

(continued from previous page)

```
//Calculate heat-index
float BME280_HI_F = -42.379 + 2.04901523*BME280_TEMP_F + 10.14333127*BME280_HUMD - 0.
↪22475541*BME280_TEMP_F*BME280_HUMD - 0.00683783*BME280_TEMP_F*BME280_TEMP_F - 0.
↪05481717*BME280_HUMD*BME280_HUMD + 0.00122874*BME280_TEMP_F*BME280_TEMP_F*BME280_HUMD␣
↪+ 0.00085282*BME280_TEMP_F*BME280_HUMD*BME280_HUMD - 0.00000199*BME280_TEMP_F*BME280_
↪TEMP_F*BME280_HUMD*BME280_HUMD; // Initial heat-index value

if(BME280_HUMD < 13.0 && BME280_TEMP_F > 80.0 && BME280_TEMP_F < 112.0){ // If RH < 13%␣
↪and 80 < T < 112
   BME280_HI_F = BME280_HI_F + (13.0 - BME280_HUMD)/4.0*sqrtf((17.0 - abs(BME280_TEMP_F -
↪ 95.0))/17);
}
else if(BME280_HUMD > 85 && BME280_TEMP_F > 80.0 && BME280_TEMP_F < 87.0){ // If RH > 85
↪% and 80 < T < 87
   BME280_HI_F = BME280_HI_F + (BME280_HUMD - 85.0)/10*(87 - BME280_TEMP_F)/5;
}
else if(BME280_HI_F < 80.0){ // If HI < 80
   BME280_HI_F = 0.5 * (BME280_HI_F + 61.0 + 1.2*(BME280_HI_F - 68.0) + BME280_HUMD*0.
↪094);
}

BME280_HI = (5.0/9.0)*(BME280_HI_F - 32.0); // Convert back to Celsius

// Retrieve data from ENS160 VOC sensor
if(ENS160_STS && ENS160.available()){
   ENS160.measure(true);
   ENS160_AQI = ENS160.getAQI(); // Get air quality index
   ENS160_TVOC = ENS160.getTVOC(); // Get total volatile organic compound concentration␣
↪in parts per billion
   ENS160_eCO2 = ENS160.geteCO2(); // Get eCO2 measurement, derived from TVOC
}
else{
   ENS160_AQI = 0;
   ENS160_TVOC = 0;
   ENS160_eCO2 = 0;
}

//Retrieve data from LTR390 UVA sensor
if(LTR390_STS && LTR390.newDataAvailable()){
   LTR390_RAW = LTR390.readUVS();
   LTR390_UVI = LTR390_RAW / 2300.00; // Calculate UV-index from raw values
}
else{
   LTR390_UVI = 0;
}

if(VEML7700_STS){
   VEML7700_LUX = VEML7700.readLux(); // Retrieve data from VEML7700 ALS sensor
}
else{
   VEML7700_LUX = 0;
}
```

(continues on next page)

(continued from previous page)

```
//Retrieve data from SCD40 CO2 sensors
if(SCD40_STS){
   bool isDataReady = false; // Reset Data-Ready flag
   if(scd4x.getDataReadyFlag(isDataReady)){ // Check if there is data available to read
   }

   if(isDataReady){ // If data is ready to be read, read data
      if(scd4x.readMeasurement(SCD40_CO2_RAW, SCD40_TEMP, SCD40_HUMD)){
      }
      else if(SCD40_CO2_RAW == 0){ // If CO2 data is zero, keep data
      }
      else{
         SCD40_CO2 = static_cast<float>(SCD40_CO2_RAW); // Cast CO2 unsigned 16bit-
↪integer to type float
      }
   }
   else{ // If data is not ready, skip measurement
   }
}
else{
   SCD40_CO2 = 0;
}

//Clock timer
ss = (millis() - clock_timer)/1000; // Set seconds to how much time has elapsed since␣
↪first data read
if(ss >= 60){ // If seconds is 60+, add 1 minutes and reset seconds variable
   clock_timer = millis();
   ss = ss - 60;
   mm = mm + 1;
   if(mm >= 60){ // If minutes is 60+, add 1 hours and reset minutes and seconds
      ss = ss - 60;
      mm = mm - 60;
      hh = hh + 1;
   }
}

char buffer[1024]; // Create 1024-bit buffer for data output
sprintf(buffer, "\n|%02d:%02d:%02d| %5.2f|%5.2f| %5.2f|%6.1f|%5.1f| %4.0f| %5.0f|  %5.
↪0f|  %5.2f|%7.2f|", hh, mm, ss, BME280_TEMP, BME280_HUMD, BME280_HI, BME280_PRES,␣
↪BME280_ALT, SCD40_CO2, ENS160_TVOC, ENS160_AQI, LTR390_UVI, VEML7700_LUX);
Serial.print(buffer); // Print buffer

//Idle until it is time for next data read
while(true){
   if(millis() - delay_timer >= 1000) break;
   delay(5);
   }
}
```

0.

1.13.3 Demo Sketch

The Demo sketch is used as an introductory program to most of the features on the weather station. This includes scrollable menus for all sensors.

There is no data recording in this example.

Button Menu

On the weather station there are 4 buttons, ↑, ↓, ←, →.

The ← and → buttons are used to scroll between the weather menu, air quality menu, and the light menu.

1.13.4 Full Code Sketch

The Full Code sketch incorporates all of the components of the weather station, including data recording. This has all the menus of the Demo sketch as well as a menu used to start and stop data recording.

Program Overview

Button Menu

On the weather station there are 4 buttons, ↑, ↓, ←, →.

The ← and → buttons are used to scroll between the weather menu, air quality menu, and the light menu.

The ↑ arrow is used to navigate to the data recording menu. Here you can choose to either start recording data or pause data recording. **data recording is paused by default**. Press the → button to confirm your selection.

The ↓ is used to navigate the data recording menu above. When in either the air quality or light menu, pressing it will return you to the weather menu.

1.14 Overview of Sensor Activities

Each activity should take approximately 10 minutes to complete.

1.14.1 Temperature Activity - Microclimates

Duration: 15 minutes

Background

Materials

- Destination Weather Station
- Paper
- Pencil

Procedure

1. Make a map of the classroom on a sheet of paper. Make sure to label any windows, doors, air vents, and significant heat sources.
2. Make a prediction for which areas of the classroom you think will be hot or cold.
3. On your map, mark locations in the classroom you want to collect data.
4. Using your weather station, record the average temperature at each location you marked on your map.

Discussion Questions

1. Did your predictions match the data you collected? Why might they be different?
2. What do you think caused that part of the classroom to be that temperature?

1.14.2 Humidity Activity - Relative vs Absolute

measure humidity, idk.

Duration: 15 minutes

Background

Materials

- Destination Weather Station
- Humidifier

Procedure

Discussion Questions

1.14.3 Carbon-Dioxide Activity - Photosynthesis

plant in bag. measure CO2 inside bag and outside bag.

Duration: 15 minutes

Background

Materials

plant

Procedure

Discussion Questions

1.14.4 Volatile Organic Compounds Activity - Methane / Ethane TBD

measure VOCs. relate to IRL VOC measurements.

Duration: 15 minutes

Background

Materials

Procedure

Discussion Questions

1.14.5 Ultraviolet Light Activity - Material Adsorption

Measure how much UV passes through different materials like glass, tinted glass, plexiglass, whatever.

Duration: 15 minutes

Background

Materials

Procedure

Discussion Questions

1.14.6 Ambient Light Activity - Light abortion related to Albedo and climate

Reflect light from lamp off of paper and measure reflected light. white paper vs brown? dirt? idk.

Duration: 15 minutes

Background

intensity to luminous flux to radiant flux.

Materials

Procedure

Discussion Questions

1.15 Manufacturing and Ordering Kits

- *Ordering PCBs*
- *Adafruit Expansion Sensors*
- *3D printing Ballon Mounts*
- *Building the Kits*

1.16 Manufacturing PCBs

To manufacture the Destination Weather Station v4.5, files for JLCPCB have been provided.

1.16.1 Download Files

JLCPCB handles the manufacturing and assembly of the weather station PCB.

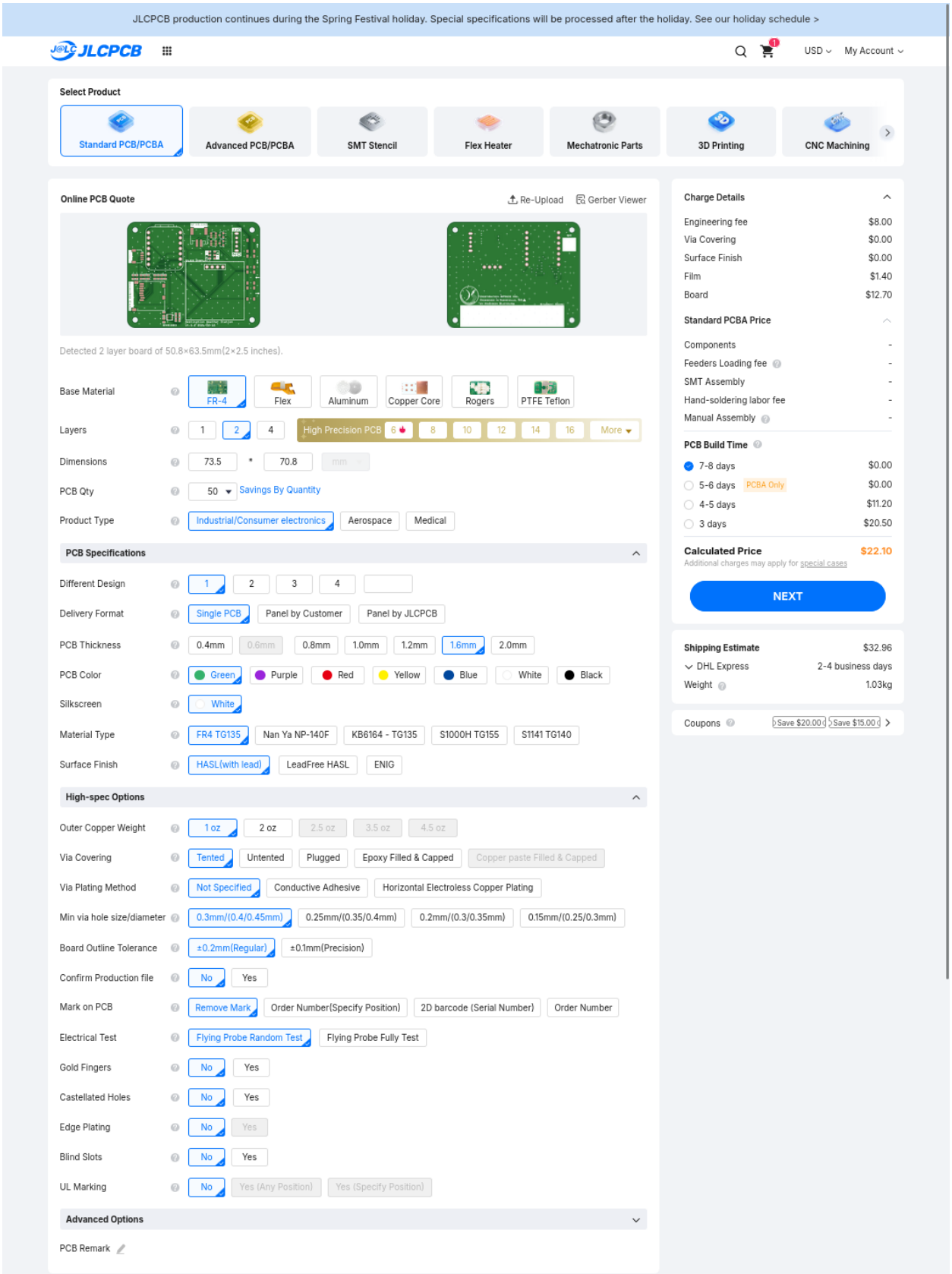
Download the following files from the GitHub repository.

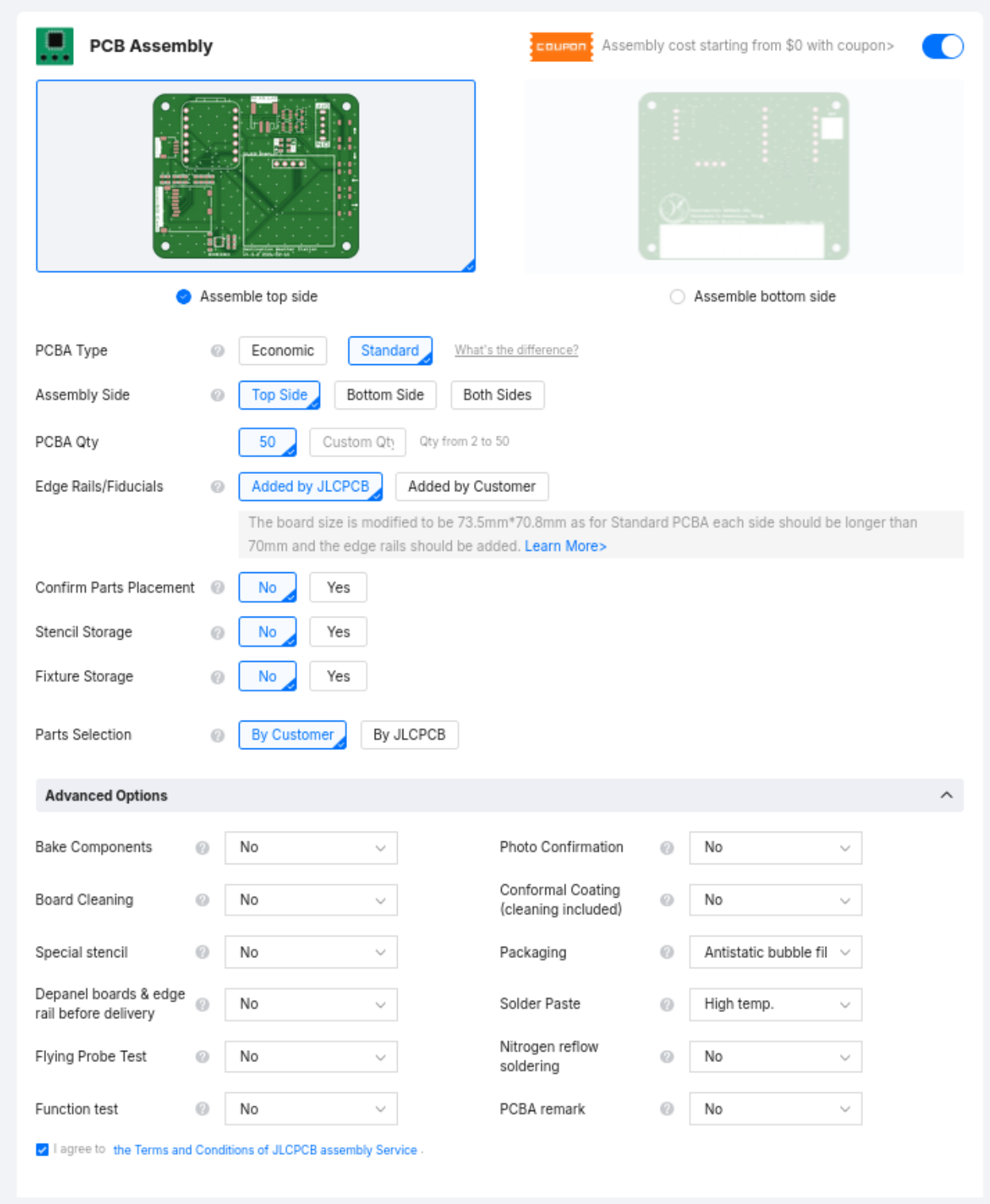
- Gerbers
- Bill of materials
- Component placement

1.16.2 Ordering

1. Go to JLCPCB.com/quote
2. Click **Add gerber file** and upload the `jlcpcb_gerbers.zip` file
3. Change the following parameters
    - PCB Qty: Select a quantity
    - PCB Color: Select any color (this may change cost or manufacturing time)
    - Surface Finish: LeadFree HASL

- PCB Assembly > PCB Type: Standard

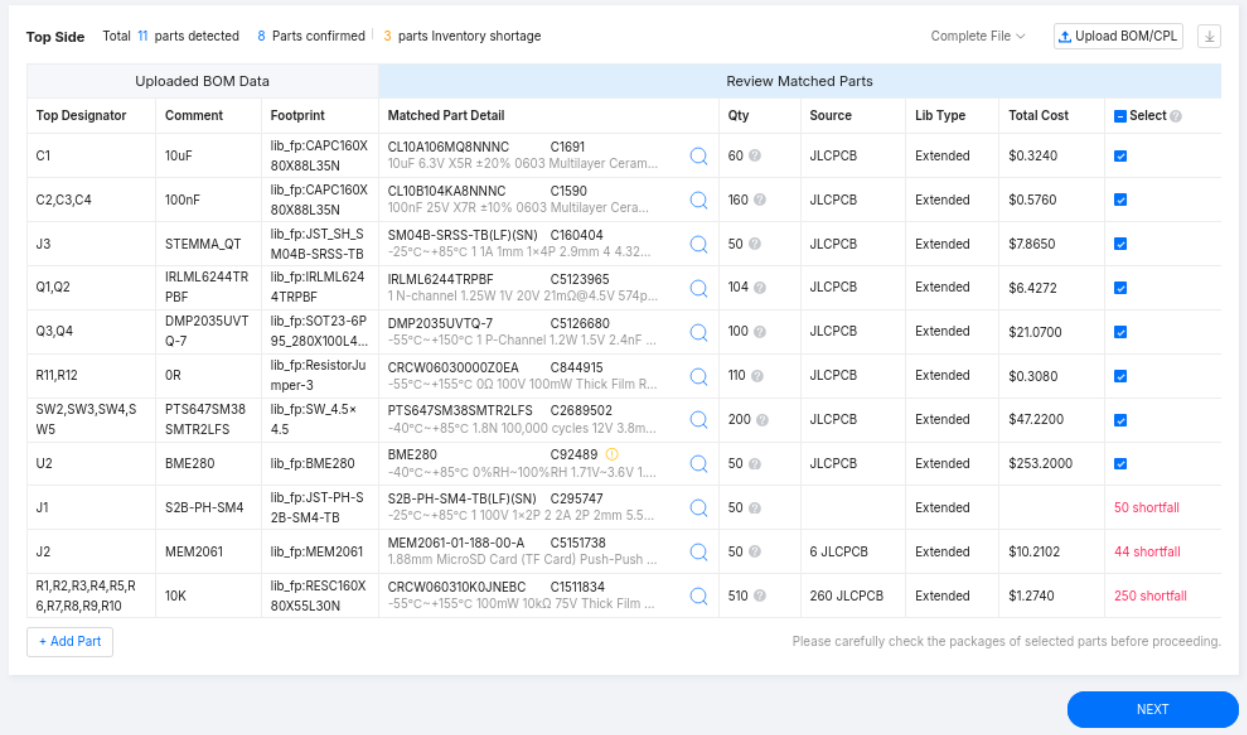




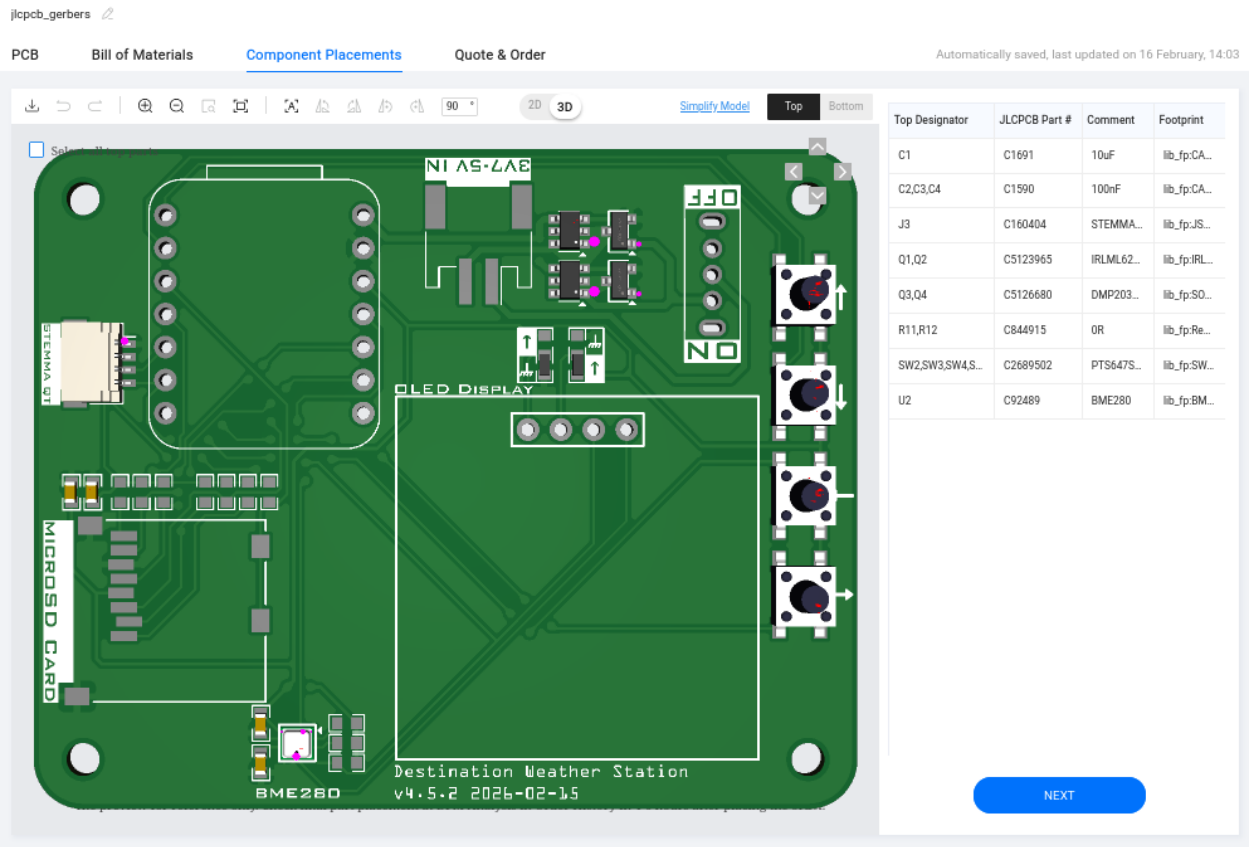
4. Click **Next**
5. Click **Next**
6. Upload the `jlcpcb_bom.csv` and `jlcpcb_cpl.csv` files.
7. Click **Process BOM & CPL**

8. Click **Next**

   If components are unavailable, make substitutions.



9. Click **Next**



10. If asked, select *ResearchEducationDIYEntertainment > Development Board - HS Code 847330* for the product description.
11. Click **Save to Cart**
12. Complete purchase

1.17 Adafruit Expansion Sensors

Below is a list of the components needed to build a classroom set of environmental sensors

Component	Manufacturer	Manufacturer Part Number	Quantity	Link
SCD-40 - True CO2 Sensor	Adafruit	5187	7	*https://www.adafruit.com/product/5187*
ENS160 - MOX Gas Sensor	Adafruit	5606	8	*https://www.adafruit.com/product/5606*
VEML7700 - Lux Sensor	Adafruit	4162	8	*https://www.adafruit.com/product/4162*
LTR390 - UV Light Sensor	Adafruit	4831	8	*https://www.adafruit.com/product/4831*
STEMMA QT JST SH 4-Pin Cable - 50mm	Adafruit	4399	24	*https://www.adafruit.com/product/4399*
STEMMA QT JST SH 4-Pin Cable - 100mm	Adafruit	4210	12	*https://www.adafruit.com/product/4210*
Adafruit Swirly Aluminum Mounting Grid - 5x5	Adafruit	5779	8	*https://www.adafruit.com/product/5779*

1.18 3D Printing Balloon Mounts

1.19 Additional Materials

To complete the kit, a few additional components will need to be sourced as these will be soldered by the students. These components can be found in the table below.

Component	Manufacturer	Manufacturer Part Number	Link
XIAO RP2040	Seeed Studio	XIAO RP2040	*https://www.seeedstudio.com/XIAO-RP2040-v1-0-p-5026.html*
SPDT slide switch	C&K	OS102011MS2QN1	*https://www.lcsc.com/product-detail/C221829.html*
128x64 SSD1315 OLED display	HS	HS96L03W2C03	*https://www.lcsc.com/product-detail/C5248080.html*
Pin header 4 position 2.54mm pitch	XFCN	PZ254V-11-04P	*https://www.lcsc.com/product-detail/C2691448.html*
3x AAA Battery holder	MYOUNG	BH-AAA-B1AJ022	*https://www.lcsc.com/product-detail/C5290194.html*

1.20 Frequently Asked Questions

Coming soon!