Atlas Scientific™
Environmental Robotics

**V 4.5**

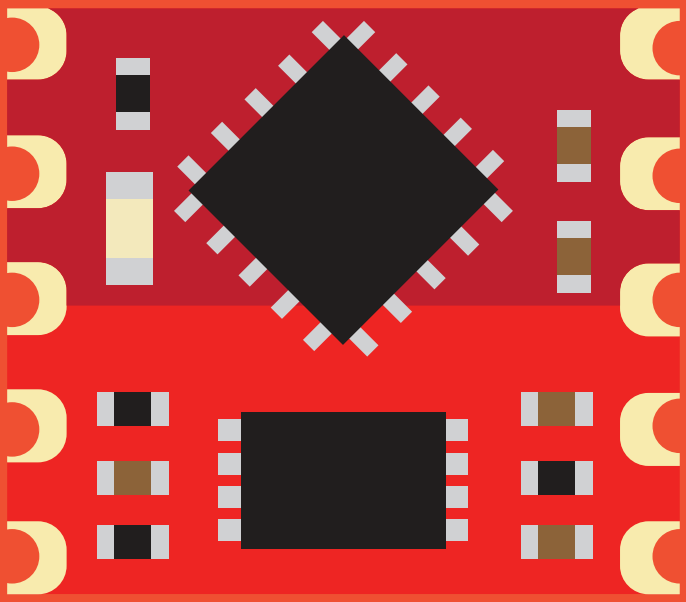
Revised 11/24

# OEM-pH™

**Embedded pH Circuit**

**ISO 10523 Compliant**

| | |
|---|---|
| Reads | **pH** |
| Range | **.001 – 14.000** |
| Resolution | **.001** |
| Accuracy | **+/– 0.002** |
| Response time | **1 reading every 420ms** |
| Supported probes | **Any type & brand** |
| Calibration | **1, 2, 3 point** |
| Temp compensation | **Yes** |
| Data protocol | **SMBus/I²C** |
| Default I²C address | **0x65** |
| Operating voltage | **3.0V – 5.5V** |
| Data format | **ASCII** |



RoHS COMPLIANT

**PATENT PROTECTED**

Written by Jordan Press
Designed by Noah Press

This is an evolving document, check back for updates.

# STOP

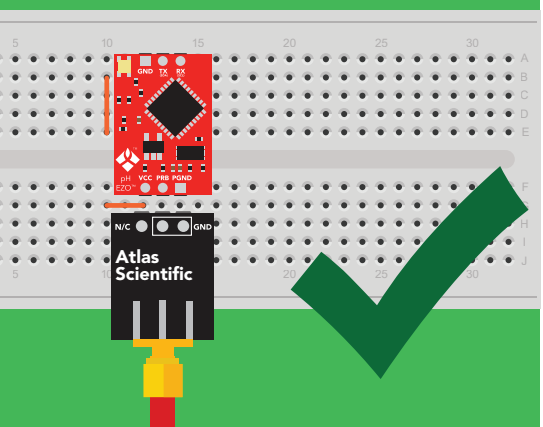
## SOLDERING THIS DEVICE VOIDS YOUR WARRANTY.

**Before purchasing the pH OEM™ read this data sheet in its entirety. This product is designed to be surface mounted to a PCB of your own design.**

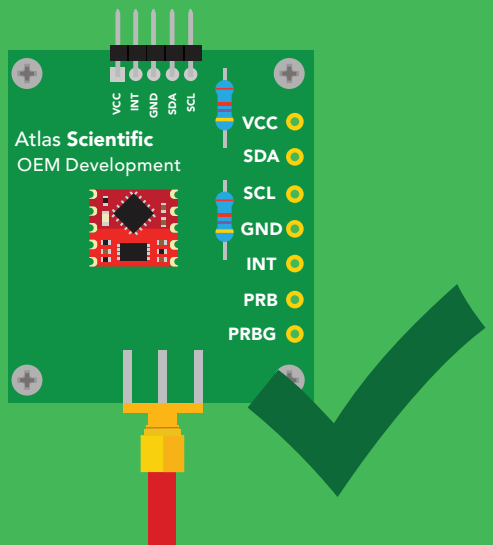
**This device is designed for electrical engineers who are familiar with embedded systems design and programing. If you, or your engineering team are not familiar with embedded systems design and programing, Atlas Scientific does not recommend buying this product.**

**Unfamiliar with pH sensing? Try our EZO™ pH circuit first.**

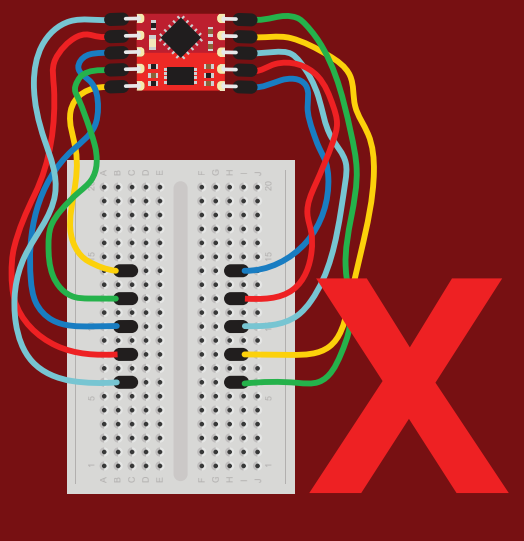
**It's much easier to use, and provides a good working reference.**

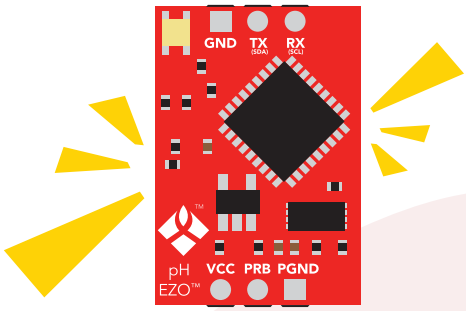



**Get this device working in our OEM Development board first!**



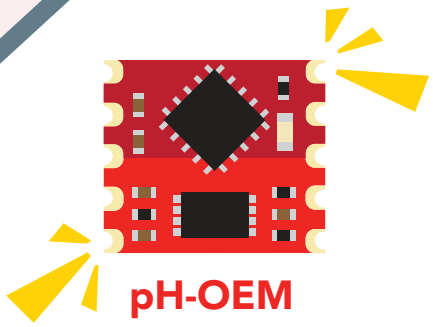

**Do not solder wires to this device.**





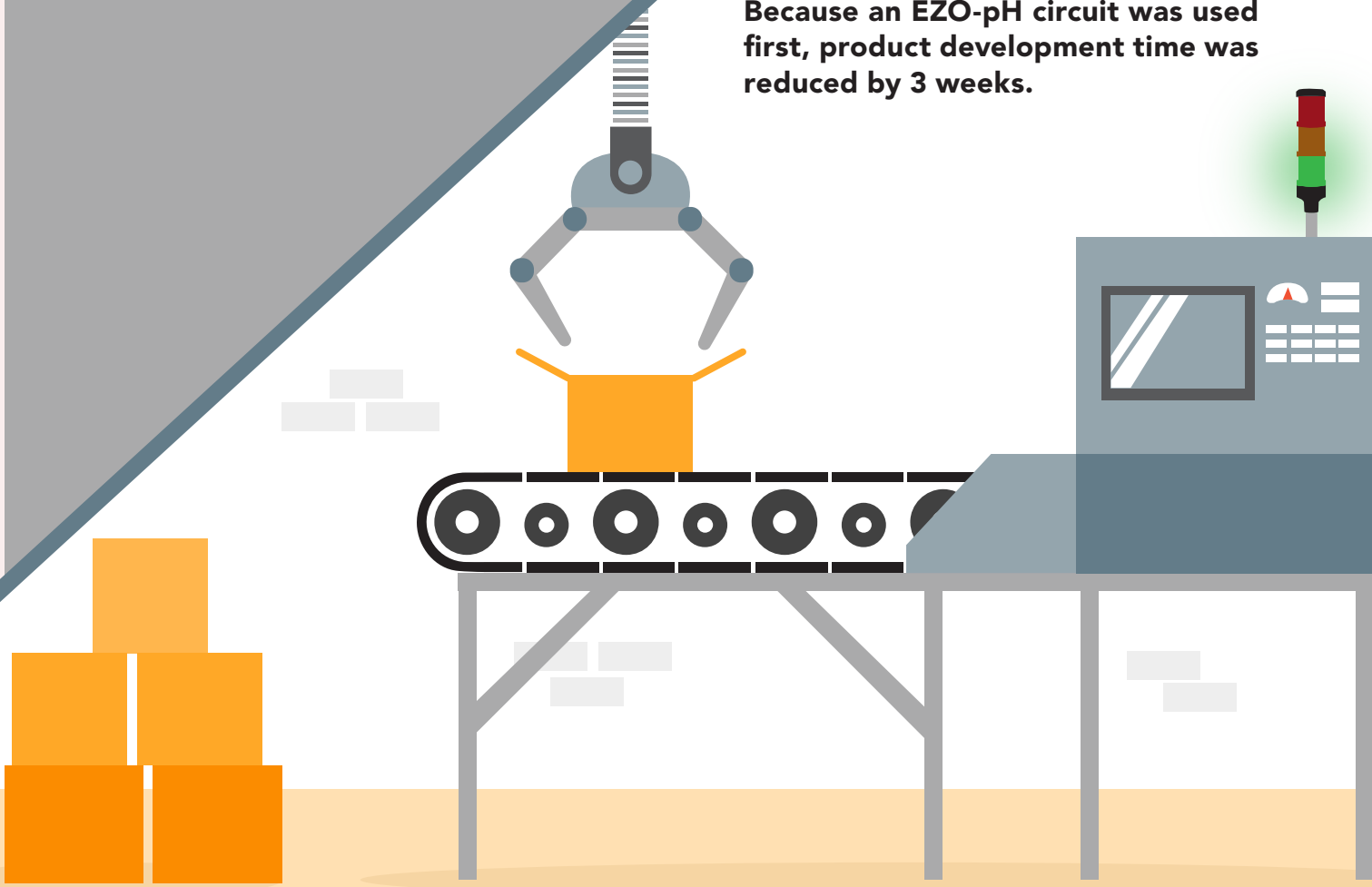

EZO-pH

**Getting an EZO-pH circuit working first will significantly speed up OEM development. It will act as a working reference model; making it easy to debug OEM issues.**



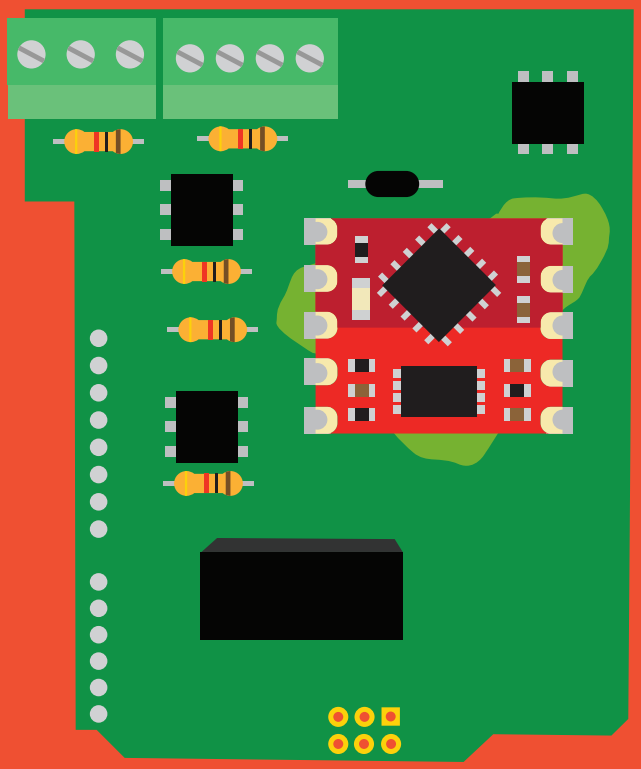
pH-OEM

**Because an EZO-pH circuit was used first, product development time was reduced by 3 weeks.**



# Attention

**After soldering the OEM circuit to your PCB, all flux residue MUST be removed. Failure to do so could lead to a high impedance short, making it impossible to get accurate readings.**



## Signs of a high impedance short.

pH readings will be locked at 0, 7 or 14 no matter what calibration solution the pH probe is in.

Readings will drift up or down until they eventually reach 0 or 14.

Getting an accurate, stable reading will be very difficult and it may take over an hour to get a stable reading.

The device responds to rituals.
***"It only works when I'm standing."***

## DO NOT SKIP THIS STEP

**The PCB must be washed with an ultrasonic cleaner, OR cleaned with a commercial flux removing chemical, OR soaked in alcohol for ~20 mins.**



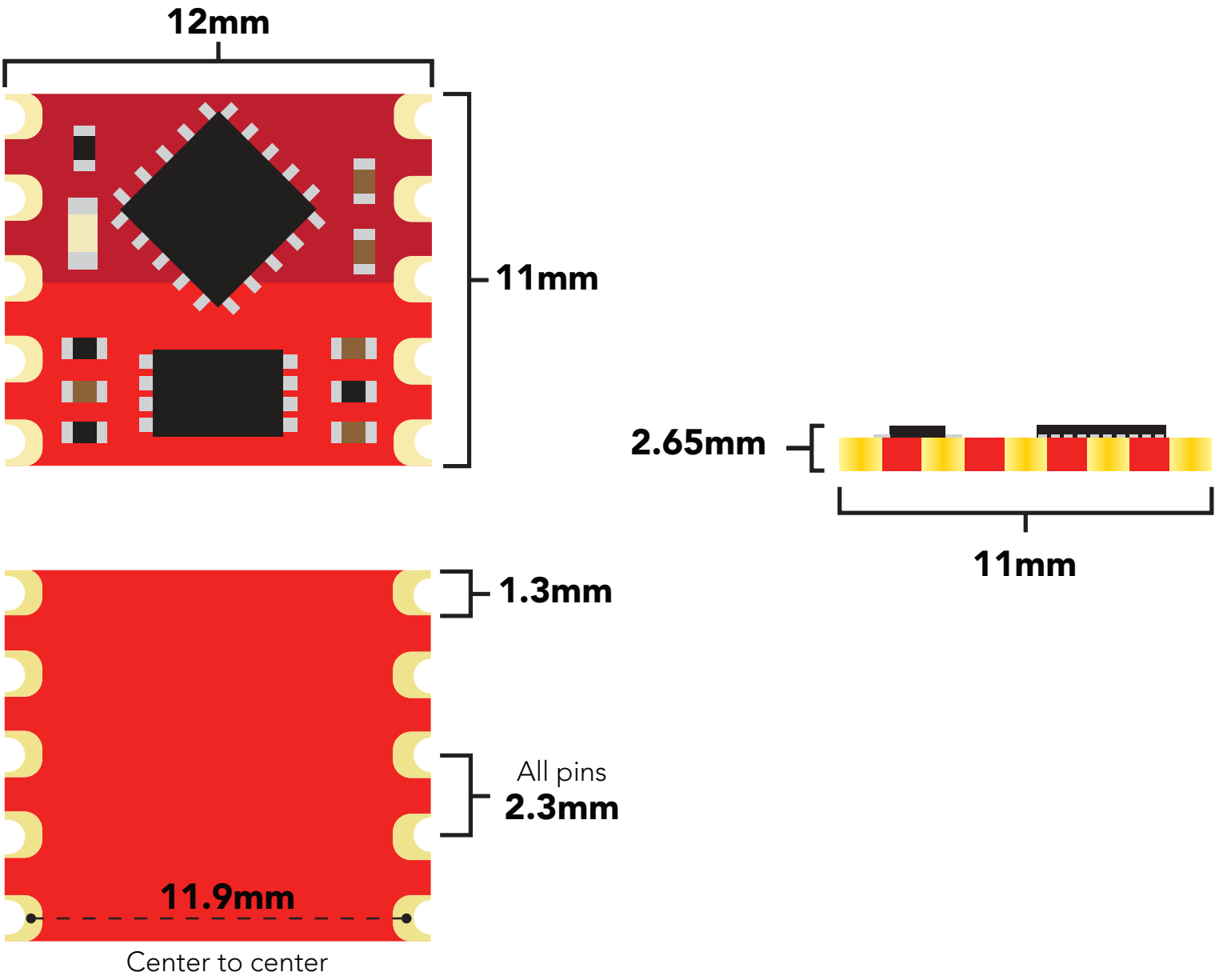

Atlas Scientific™ Environmental Robotics

# Table of contents



# REGISTERS

# OEM circuit dimensions



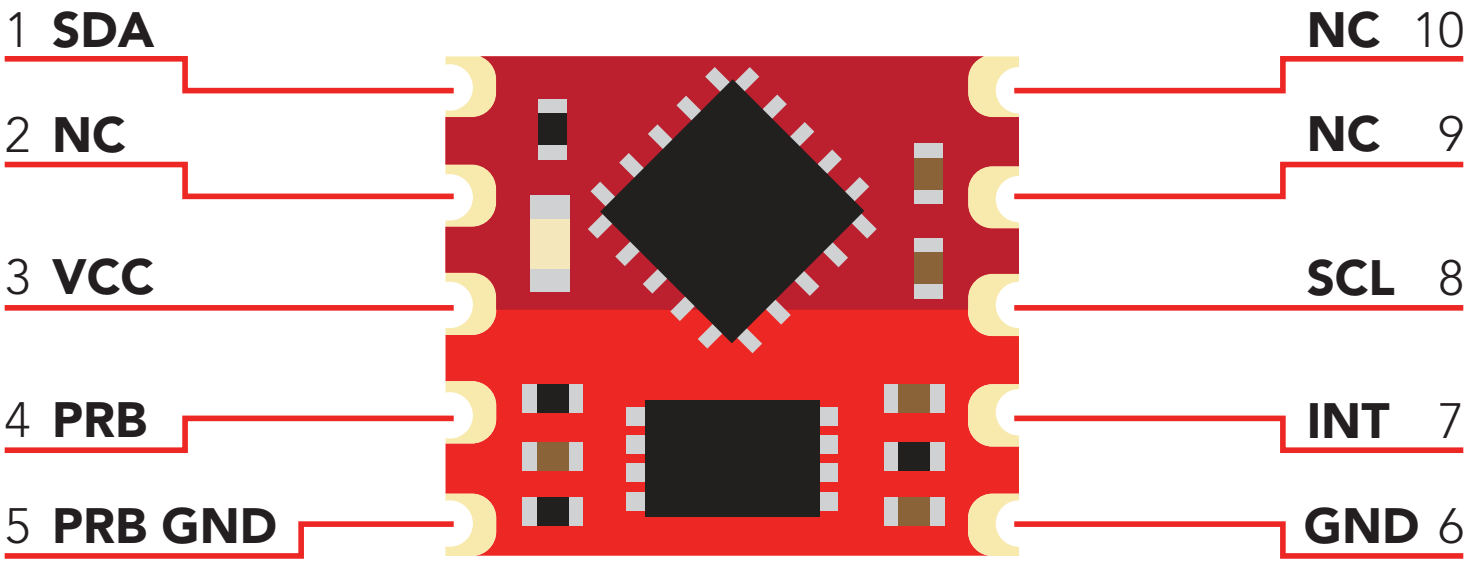

## Power consumption

| | LED | OPERATIONAL | HIBERNATION |
|---|---|---|---|
| 3.3V | ON | 3.46 mA | 3.43 mA |
| | OFF | 3.03 mA | 3.0 mA |

## Absolute max ratings

| Parameter | MIN | TYP | MAX |
|---|---|---|---|
| Storage temperature | -60 °C | | 150 °C |
| Operational temperature | -40 °C | 25 °C | 125 °C |
| VCC | 3.0V | 3.3V | 5.5V |

# Pin out




# Resolution

The resolution of a sensor is the smallest change it can detect in the quantity that it is measuring. The Atlas Scientific™ pH OEM™ will always produce a reading with a resolution of three decimal places.

**Example**
0.002 pH
13.476 pH

# Power on/start up

Once the Atlas Scientific™ pH OEM™ is powered on it will be ready to receive commands and take readings after 1ms. Communication is done using the SMBus/I²C protocol at speeds of 10 – 100 kHz.

**Settings that are retained if power is cut**

Calibration
I²C address

**Settings that are *NOT* retained if power is cut**

Active/Hibernation mode
LED control
Interrupt control

# System overview

The Atlas Scientific pH OEM™ Class Embedded Circuit is the core electronics needed to read the pH of water from any off the shelf pH probe. The pH OEM™ Embedded Circuit will meet, or exceed the capabilities and accuracy found in all models of bench top laboratory grade pH meters.

The pH OEM™ is an SMBus / I²C slave device that communicates to a master device at a speed of 10 to 100 kHz. Read and write operations are done by accessing **26** different 8 bit registers.

## Accessible registers

### Device information

| | |
|---|---|
| 0x00: Device type | R |
| 0x01: Firmware version | R |

### Device address

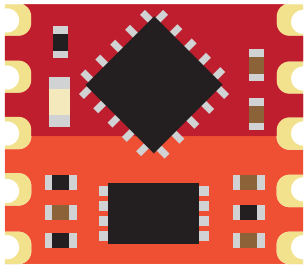
| | |
|---|---|
| 0x02: SMBus/I²C address lock/unlock | R/W |
| 0x03: SMBus/I²C address | R/W |

### Control

| | |
|---|---|
| 0x04: Interrupt control | R/W |
| 0x05: LED control | R/W |
| 0x06: Active/hibernate | R/W |
| 0x07: New reading available | R/W |

### Calibration

| | |
|---|---|
| 0x08: Calibration value MSB | R/W |
| 0x09: Calibration value high byte | R/W |
| 0x0A: Calibration value low byte | R/W |
| 0x0B: Calibration value LSB | R/W |
| 0x0C: Calibrate request | R/W |
| 0x0D: Calibration confirm | R |



**Read only** R
**Read and write** R/W

### Compensation

| | |
|---|---|
| 0x0E: Temperature compensation MSB | R/W |
| 0x0F: Temperature compensation high byte | R/W |
| 0x10: Temperature compensation low byte | R/W |
| 0x11: Temperature compensation LSB | R/W |

### Confirmation

| | |
|---|---|
| 0x12: Temperature confirm MSB | R |
| 0x13: Temperature confirm high byte | R |
| 0x14: Temperature confirm low byte | R |
| 0x15: Temperature confirm LSB | R |

### Sensor Data

| | |
|---|---|
| 0x16: pH reading MSB | R |
| 0x17: pH reading high byte | R |
| 0x18: pH reading low byte | R |
| 0x19: pH reading LSB | R |

**Each pH reading takes 420ms**

The default device address is **0x65**
This address can be changed.



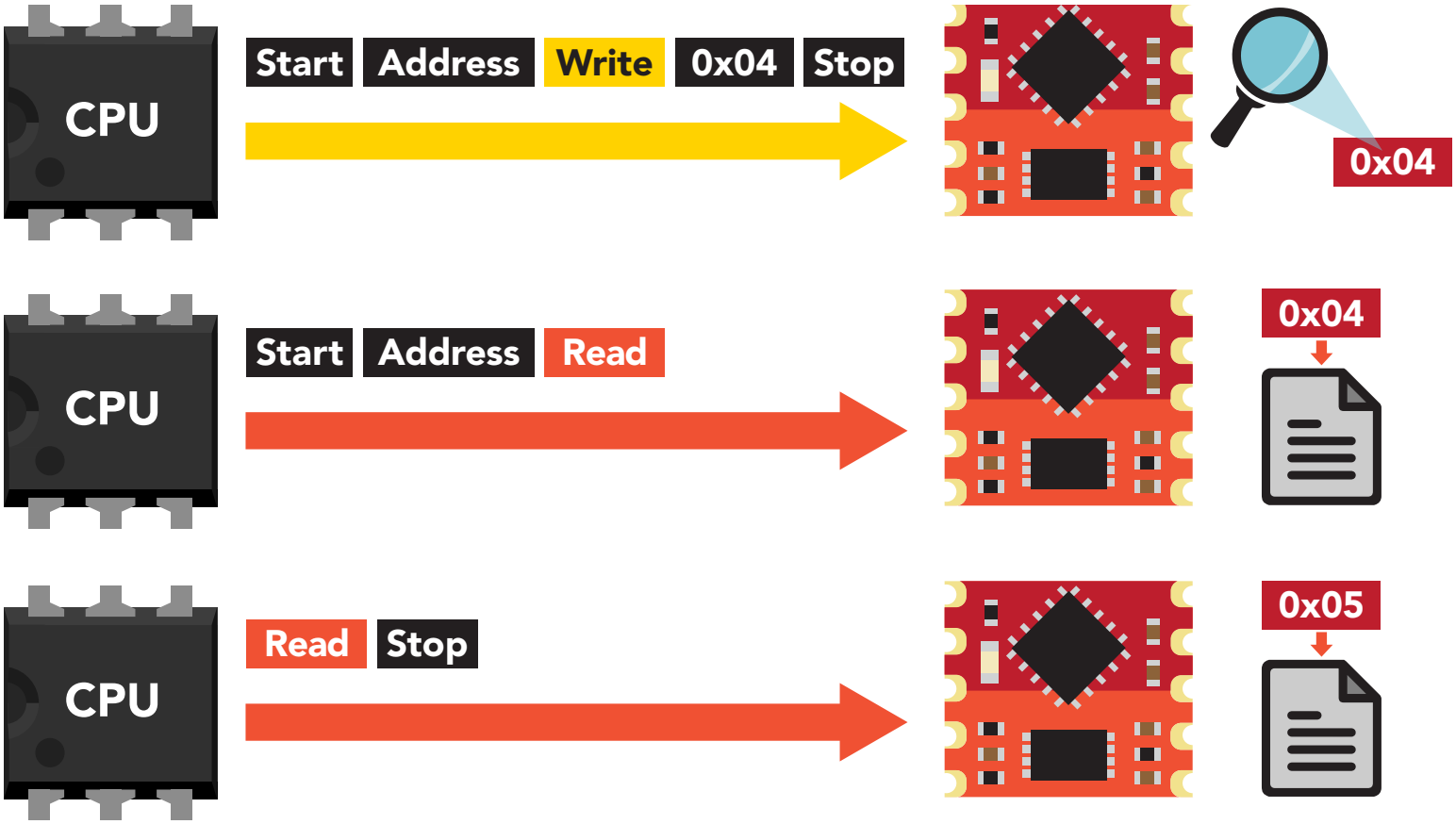

AtlasScientific™ Environmental Robotics

# Reading register values

To read one or more registers, issue a write command and transmit the register address that should be read from, followed by a stop command. Then issue a read command, the data read will be the value that is stored in that register. Issuing another read command will automatically read the value in the next register. This can go on until all registers have been read. After reading the last register, additional read commands will return 0xFF. Issuing a stop command will terminate the read event.

The default device address is **0x65**
This address can be changed.

**Example**
Start reading at register 0x04 and read 2 times.




**Example code**
**reading two registers**

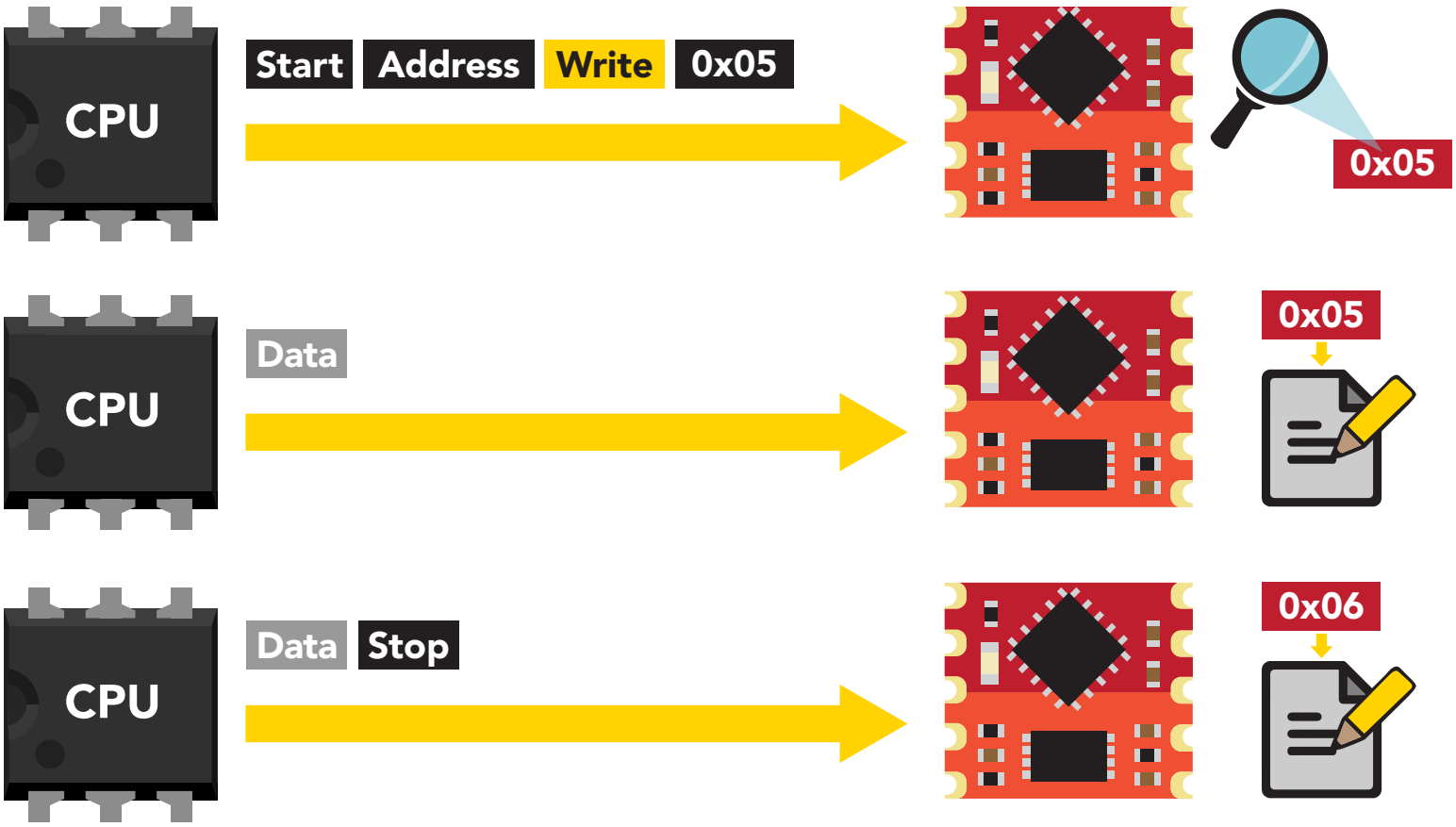
```
byte i2c_device_address=0x65;
byte reg_4, reg_5;

Wire.beginTransmission(i2c_device_address);
Wire.write(0x04);
Wire.endTransmission();

Wire.requestFrom(i2c_device_address,2);
reg_4=Wire.read();
reg_5=Wire.read();
Wire.endTransmission();
```

# Writing register values

All registers can be read, but only registers marked read/write can be written to.

To write to one *(or more)* registers, issue a write command and transmit the register address that should be written to, followed by the data byte to be written. Issuing another write command will automatically write the value in the next register. This can go on until all registers have been written to. After writing to the last register, additional write commands will do nothing.

**Example**
Start writing at address 0x05 and write 2 values.




**Example code**
**writing the number 1 in register 0x05 – 0x06**

```
byte i2c_device_address=0x65;
byte starting_register=0x05
byte data=1;

Wire.beginTransmission(i2c_device_address);
Wire.write(starting_register);
Wire.write(data);
Wire.write(data);
Wire.endTransmission();
```

# Sending floating point numbers

**For ease of understanding we are calling fixed decimal numbers "floating point numbers." We are aware they are not technically floating point numbers.**

It is not possible to send/receive a floating (fixed decimal) point number over the SMBus/ $I^2C$ data protocol. Therefore, a multiplier/divider is used to remove the decimal point. Do not transmit a floating point number without property formatting the number first.

2 blocks of registers require the master to transmit a floating point number.

**Calibration** **Compensation**

When transmitting a floating point number to any of these 2 register blocks, the number must first be multiplied by ***1000* for pH calibration values** and ***100* for temperature compensation values**. This would have the effect of removing the floating point. Internally the pH OEM™ will divide the number by 100 or 1000 (depending on type), converting it back into a floating point number.

### Example

Setting a pH calibration midpoint of: 7.123
7.123 X **1000** = 7123
Transmit the number 7123 to the Calibration Value Registers

Setting a pH calibration low point of: 4.00
4.000 X **1000** =4000
Transmit the number 4000 to the Calibration Value Registers

Setting a temperature compensation value of 99.06˚C
99.06 X **100** = 9906
Transmit the number 9906 to the Temperature Compensation Registers

When reading back a value stored in one of these 2 register blocks the value must be divided by 100 or 1000 (depending on type) to return it to its originally intended value.

# Receiving floating point numbers

2 blocks of registers require the master to transmit a floating point number.

**Sensor Data** **Confirmation**

After receiving a value from any of these 2 register blocks, the number must be divided by ***1000*** ***for the pH Read Register*** or ***100*** ***for the Temperature Confirmation Register*** to convert it back into a floating point number.

**Example**
Reading an pH value of 14.563
Value received = 14563
14563 / **1000** =14.563

Reading a Temperature confirmation value of 99.06˚C
Value received = 9906
9906 / **100** = 99.06˚C

# Registers

# Device information



| | |
|---|---|
| 0x00: Device type | R |
| 0x01: Firmware version | R |

## 0x00 – Device type register

1 unsigned byte
Read only value = 1
1 = pH

This register contains a number indicating what type of OEM device it is.

## 0x01 – Firmware version register

1 unsigned byte
Read only value = 2
2 = firmware version

This register contains a number indicating the firmware version of the OEM device.

**Example code**
**reading device type and device version registers**

```
byte i2c_device_address=0x65;
byte starting_register=0x00
byte device_type;
byte version_number;

Wire.beginTransmission(i2c_device_address);
Wire.write(staring_register);
Wire.endTransmission();

Wire.requestFrom(i2c_device_address,(byte)2);
device_type = Wire.read();
version_number = Wire.read();
Wire.endTransmission();
```

# Changing I²C address

| | |
|---|---|
| 0x02: SMBus/I²C address lock/unlock | R/W |
| 0x03: SMBus/I²C address | R/W |

**This is a 2 step procedure**

To change the I²C address, an unlock command must first be issued.

## Step 1
Issue unlock command

### 0x02 – I²C address unlock register

1 unsigned byte
Read only value = 0 or 1
0 = **unlocked**
1 = **locked**

To unlock this register it must be written to twice.

| | | | |
|---|---|---|---|
| Start | unlock register | 0x55 | Stop |
| Start | unlock register | 0xAA | Stop |


The two unlock commands must be sent back to back in immediate succession. No other write, or read event can occur. Once the register is unlocked it will equal 0x00 (unlocked).

**To lock the register**
Write any value to the register other than 0x55;
or, change the address in the Device Address Register.

**Example code**
**address unlock**

```
byte i2c_device_address=0x65;
byte unlock_register=0x02;

Wire.beginTransmission(bus_address);
Wire.write(unlock_register);
Wire.write(0x55);
Wire.endTransmission();

Wire.beginTransmission(bus_address);
Wire.write(unlock_register);
Wire.write(0xAA);
Wire.endTransmission();
```



# Step 2
Change address

## 0x03 – I²C address register

1 unsigned byte
Default value = **0x65**
Address can be changed **0x01 – 0x7F (1–127)**

Address changes outside of the possible range **0x01 – 0x7F (1–127)** will be ignored.

After a new address has been sent to the device the Address lock/unlock register will lock and the new address will take hold. It will no longer be possible to communicate with the device using the old address.

⚠ Settings to this register are retained if the power is cut.

**Example code**
**changing device address**

```
byte i2c_device_address=0x65;
byte new_i2c_device_address=0x60;
byte address_reg=0x03;

Wire.beginTransmission(bus_address);
Wire.write(address_reg);
Wire.write(new_i2c_device_address);
Wire.endTransmission();
```

0x00
0x01
0x02
0x03
0x04
0x05
0x06
0x07
0x08
0x09
0x0A
0x0B
0x0C
0x0D
0x0E
0x0F
0x10
0x11
0x12
0x13
0x14
0x15
0x16
0x17
0x18
0x19

# Control registers

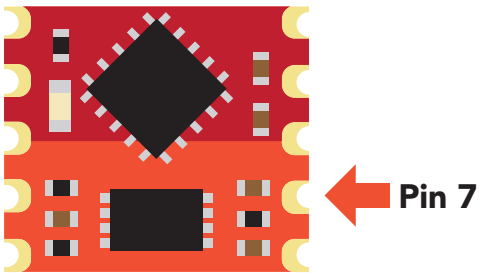
| | |
|---|---|
| 0x04: Interrupt control | R/W |
| 0x05: LED control | R/W |
| 0x06: Active/hibernate | R/W |
| 0x07: New reading available | R/W |

0x00
0x01
0x02
0x03
0x04
0x05
0x06
0x07
0x08
0x09
0x0A
0x0B
0x0C
0x0D
0x0E
0x0F
0x10
0x11
0x12
0x13
0x14
0x15
0x16
0x17
0x18
0x19

## 0x04 – Interrupt control register

1 unsigned byte
Default value = 0 (disabled)

**Command values**
0 = disabled
2 = pin high on new reading (manually reset)
4 = pin low on new reading (manually reset)
8 = invert state on new reading (automatically reset)



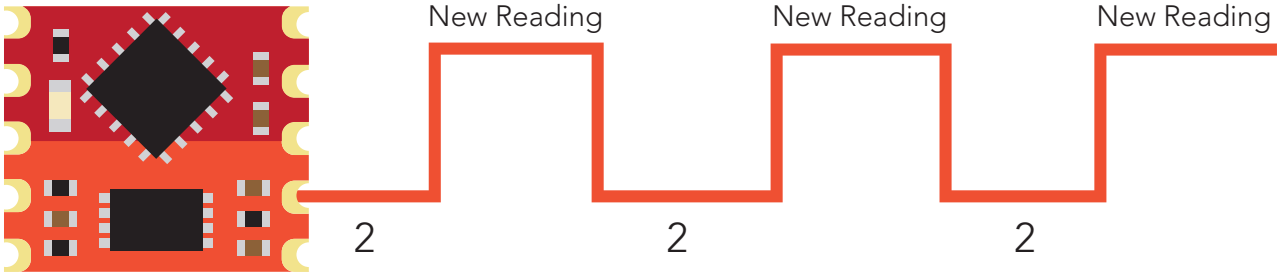

The Interrupt control register adjusts the function of pin 7 (the interrupt output pin).

⚠ Settings to this register are **not** retained if the power is cut.

## Pin high on new reading

### Command value = 2

By setting the interrupt control register to 2 the pin will go to a low state (0 volts). Each time a new reading is available the INT pin (pin 7) will be set and output the same voltage that is on the VCC pin.




The pin will not auto reset. 2 must be written to the interrupt control register after each transition from low to high.

**Example code**
**Setting pin high on new reading**

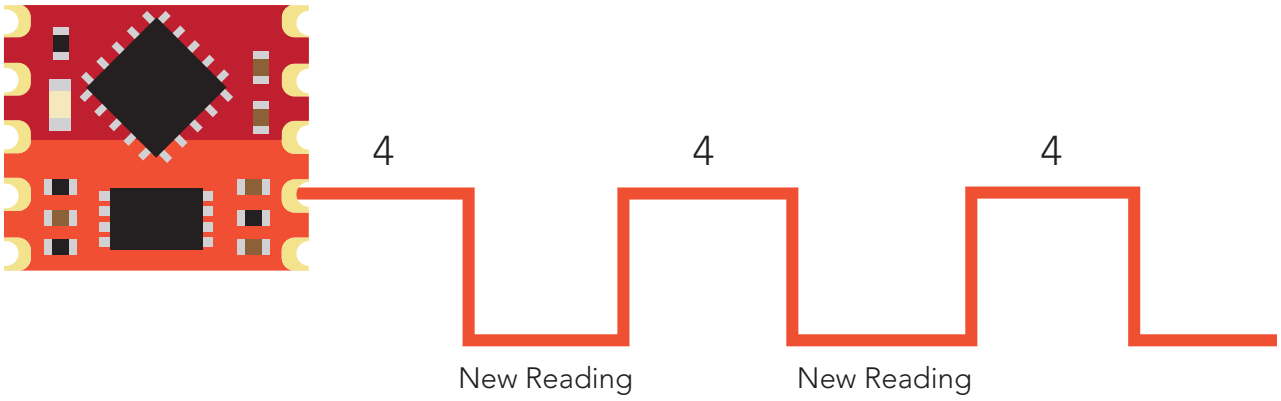
```
byte i2c_device_address=0x65;
byte int_control=0x04;

Wire.beginTransmission(i2c_device_address);
Wire.write(int_control);
Wire.write(0x02);
Wire.endTransmission();
```

# Pin low on new reading

## Command value = 4

By setting the interrupt control register to 4 the pin will go to a high state (VCC). Each time a new reading is available the INT pin (pin 7) will be reset and the pin will be at 0 volts.




The pin will not auto set. 4 must be written to the interrupt control register after each transition from high to low.

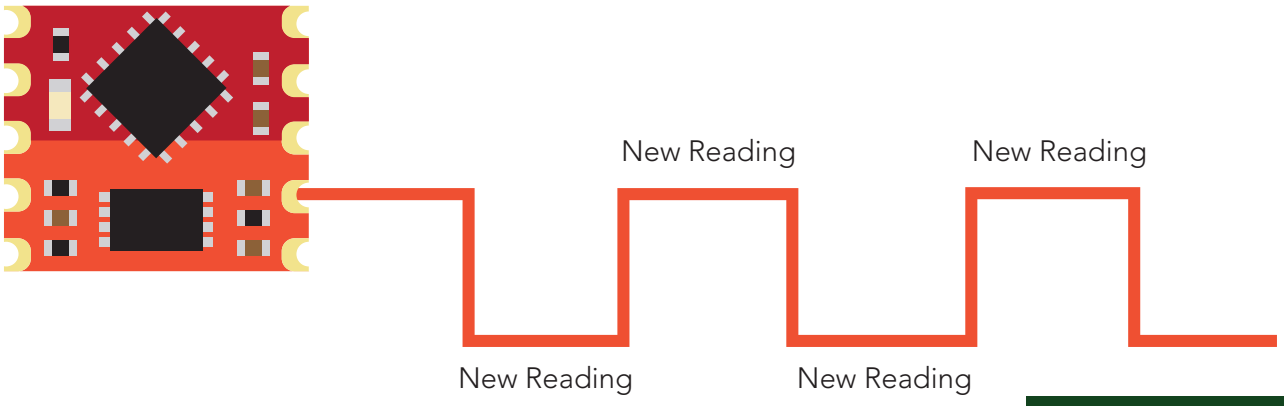
**Example code**
**Setting pin low on new reading**

```
byte I2C_device_address=0x65;
byte int_control=0x04;

Wire.beginTransmission(I2C_device_address);
Wire.write(int_control);
Wire.write(0x04);
Wire.endTransmission();
```

# Invert state on new reading

## Command value = 8

By setting the interrupt control register to 8 the pin will remain in whatever state it is in. Each time a new reading is available the INT pin (pin 7) will invert its state.




The pin will automatically invert its state each time a new reading is available. This setting has been specifically designed for a master device that can use an interrupt on change function.

**Example code**
**Inverting state on new reading**

```
byte i2c_device_address=0x65;
byte int_control=0x04;

Wire.beginTransmission(i2c_device_address);
Wire.write(int_control);
Wire.write(0x08);
Wire.endTransmission();
```



# 0x05 – LED control register

1 unsigned byte

**Command values**
1 = Blink each time a reading is taken
0 = Off

The LED control register adjusts the function of the on board LED. By default the LED is set to blink each time a reading is taken.

**Example code**
**Turning off LED**

```
byte i2c_device_address=0x65;
byte led_reg=0x05;

Wire.beginTransmission(i2c_device_address);
Wire.write(led_reg);
Wire.write(0x00);
Wire.endTransmission();
```

Settings to this register are **not** retained if the power is cut.

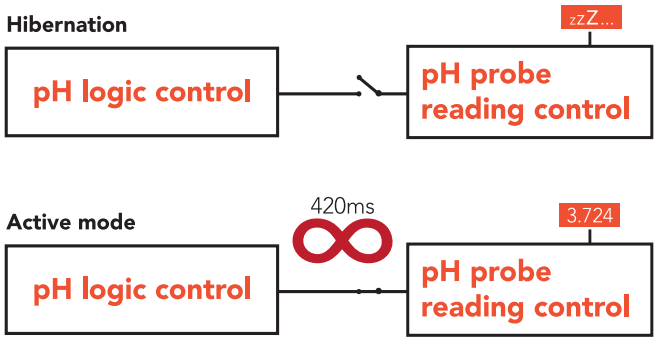
# 0x06 – Active/hibernate register

1 unsigned byte

**To wake the device**
Transmit a 0x01 to register 0x06

**To hibernate the device**
Transmit a 0x00 to register 0x06

This register is used to activate, or hibernate the sensing subsystem of the OEM device.




**Example code**
**Activate pH readings**

```
byte i2c_device_address=0x65;
byte active_reg=0x06;

Wire.beginTransmission(i2c_device_address);
Wire.write(active_reg);
Wire.write(0x01);
Wire.endTransmission();
```

Once the device has been woken up it will continuously take readings every 420ms. **Waking the device is the only way to take a reading. Hibernating the device is the only way to stop taking readings.**

# 0x07 – New reading available register

1 unsigned byte
Default value = 0 (no new reading)
New reading available = 1

**Command values**
0 = reset register



This register is for applications where the interrupt output pin cannot be used and continuously polling the device would be the preferred method of identifying when a new reading is available.

When the device is powered on, the New Reading Available Register will equal 0. Once the device is placed into active mode and a reading has been taken, the New Reading Available Register will move from 0 to 1.

**This register will never automatically reset itself to 0. The master must reset the register back to 0 each time.**

**Example code**
**Polling new reading available register**

```
byte i2c_device_address=0x65;
byte new_reading_available=0;
byte nra=0x07;

while(new_reading_available==0){
Wire.beginTransmission(i2c_device_address);
Wire.write(nra);
Wire.endTransmission();

Wire.requestFrom(i2c_device_address,(byte)1);
new_reading_available = Wire.read();
Wire.endTransmission();
delay(10);
}

if(new_reading_available==1){
call read_pH();
Wire.beginTransmission(i2c_device_address);
Wire.write(nra);
Wire.write(0x00);
Wire.endTransmission();
}
```

# Calibration

| | |
|---|---|
| 0x08: Calibration value MSB | R/W |
| 0x09: Calibration value high byte | R/W |
| 0x0A: Calibration value low byte | R/W |
| 0x0B: Calibration value LSB | R/W |

## 0x08 – 0x0B Calibration registers

Signed long
0x08 = MSB
0x0B = LSB
Units = pH

Calibration values can be whole number, or floating point.
The first calibration point must always be pH 7.

**Both pH 4 and pH 10 calibration commands can be sent as many times as you like. However, each time a ph 7 calibration command is sent both the 4 and 10 calibration points are deleted. This is because pH 7 is the foundation of the calibration protocol.**

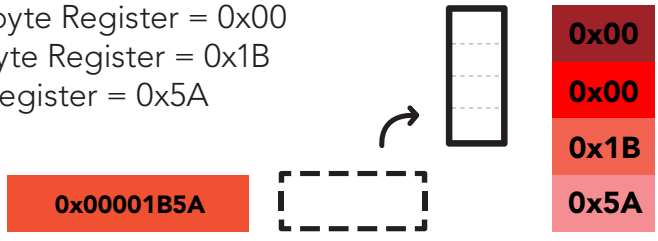
After sending a value to this register block, calibration is **not** complete. The calibration request register must be set after loading a calibration value into this register block.

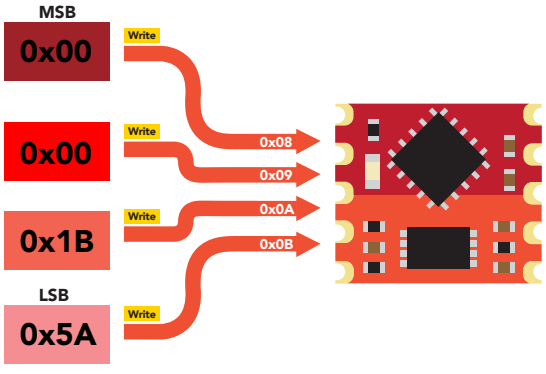
To send a new calibration value to the pH OEM™ the value of the calibration solution must be multiplied by 1000 and then transmitted to the pH OEM™. The calibration value will be divided by 1000 internally. Move the value from a float to an unsigned long. Break up the unsigned long into its 4 individual bytes. Send the bytes (MSB to LSB) to registers 0x08, 0x09, 0x0A and 0x0B.

### Example

Calibrating to a pH of 7.002
calibration value = 7.002
7.002 x 1000 = 7002
7002 to HEX = 0x00001B5A

calibration MSB Register = 0x00
calibration high byte Register = 0x00
calibration low byte Register = 0x1B
calibration LSB Register = 0x5A



# 0x0C – Calibration request register

1 unsigned byte

**Command values**
1 Clear calibration = (delete all calibration data)
2 Low point calibration = (typically this is pH 4.0)
3 Midpoint calibration = (typically this is pH 7.0)
4 High point calibration low = (typically this is pH 10.0)

Once a calibration value has been transmitted to the previous registers (0x08 – 0x0B) the calibration request register is used to apply the calibration value.

By default this register will read 0x00. When a calibration request command has been sent and a stop command has been issued, the pH OEM™ will perform that calibration requested. Once the calibration has been done the calibration request registers value will return to 0x00.

After setting this register to one of the four possible values, calibration will commence once an $I^2C$ stop bit has been transmitted.

# 0x0D – Calibration confirmation register

1 unsigned byte

**Command values**
0 = low point calibration
1 = midpoint calibration
2 = high point calibration

After a calibration event has been successfully carried out, the calibration confirmation register will reflect what calibration has been done, by setting bits 0 – 2.

| **Bit 2** (High) | **Bit 1** (Mid) | **Bit 0** (Low) | **Decimal** |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

Settings to this register are retained if the power is cut.

# Temperature compensation

| | |
|---|---|
| 0x0E: Temperature compensation MSB | R/W |
| 0x0F: Temperature compensation high byte | R/W |
| 0x10: Temperature compensation low byte | R/W |
| 0x11: Temperature compensation LSB | R/W |

## 0x0E – 0x11 Temperature compensation registers

Unsigned long
0x0E = MSB
0x11 = LSB
Default value = 25 °C
Units = °C

The pH OEM™ Embedded pH Circuit can take temperature compensated pH readings. Any temperature value from 0.01 °C to 200.0 °C can be entered into the device. The default temperature is 25.0 °C

To send a new temperature to the pH OEM™ the value of the temperature must be multiplied by 100 and then transmitted to the pH OEM™. Internally the temperature will be divided by 100.

Settings to this register are **not** retained if the power is cut.

### Example

Setting the register to 34.26°C
34.26 x 100 = 3,426
3,426 ➡ Unsigned long
Unsinged long = Hex (0x00, 0x00, 0x0D, 0x62)

| 0x0E | 0x1F | 0x10 | 0x11 |
|---|---|---|---|



# Temperature confirmation

| | |
|---|---|
| 0x12: Temperature confirm MSB | R |
| 0x13: Temperature confirm high byte | R |
| 0x14: Temperature confirm low byte | R |
| 0x15: Temperature confirm LSB | R |

## 0x12 – 0x15 Temperature confirmation registers

Unsigned long
0x12 = MSB
0x15 = LSB
Default value = 25 °C
Units = °C

**The value in this register is only updated when actively taking readings.**

This read only data is the temperature compensation value that was used to take the pH readings. This register can be used to be sure that the pH readings that are being taken are at the correct temperature.

If the temperature compensation register is changed from 25 °C to 30 °C, reading this register will show what temperature the pH reading was taken at. If a reading is being taken each time the interrupt pin fires, the first reading may still be at the old temperature of 25 °C while all other subsequent readings would then be at 30 °C.

To read the value in this register, read the bytes MSB to LSB and assign them to an unsigned long, cast to a float and divide that number by 100.



# Sensor data

| | |
|---|---|
| 0x16: pH reading MSB | R |
| 0x17: pH reading high byte | R |
| 0x18: pH reading low byte | R |
| 0x19: pH reading LSB | R |

## 0x16 – 0x19 pH reading registers

Signed long
0x16 = MSB
0x19 = LSB
Units = pH

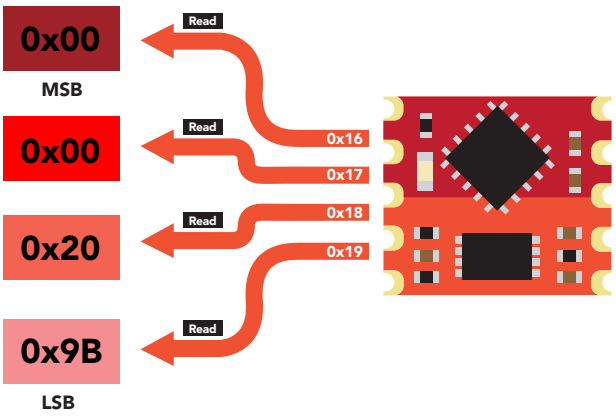
The last pH reading taken is stored in these four registers. To read the value in this register, read the bytes MSB to LSB and assign them to an unsigned long, cast to a float and divide that number by 1000.

0x00
0x01
0x02
0x03
0x04
0x05
0x06
0x07
0x08
0x09
0x0A
0x0B
0x0C
0x0D
0x0E
0x0F
0x10
0x11
0x12
0x13
0x14
0x15
0x16
0x17
0x18
0x19

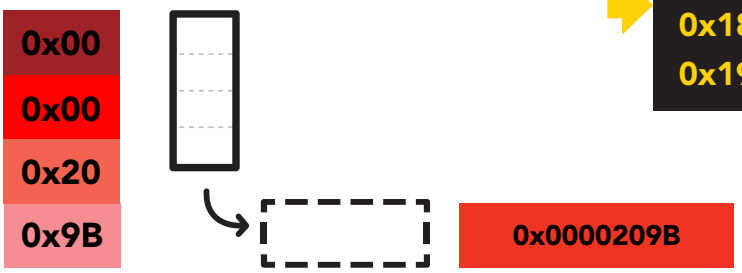
### Example

**Reading an pH of 8.347**

**Step 1** read 4 bytes




**Step 2** read unsigned long




**Step 3** cast unsigned long to a float


**Step 4** divide by 1,000

# Calibration theory

**The first calibration point must be the Mid point (pH 7.00)**

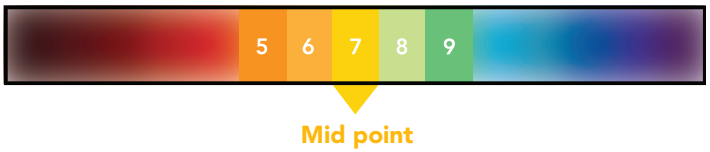
If this is your first time calibrating the pH OEM™ circuit, we recommend that you follow this calibration order.


1 **Mid point** 2 **Low point** 3 **High point**

## Single, Two point, or Three point calibration

### No calibration

### Single point calibration




### Two point calibration

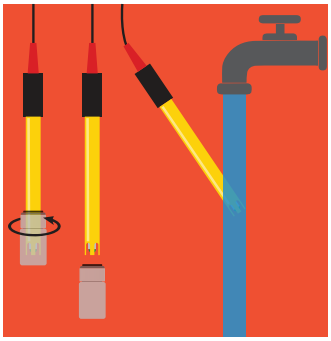

### Three point calibration

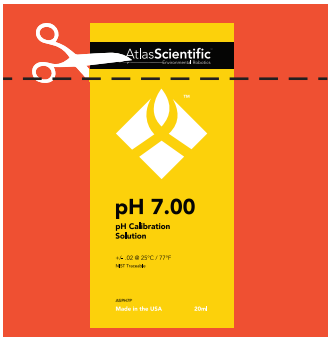

Two point calibration will provide high accuracy between **7.00** and the second point calibrated against, such as a **4.00**.

Three point calibration will provide high accuracy over the full pH range. Three point calibration at **4.00**, **7.00** and **10.00** should be considered the standard.

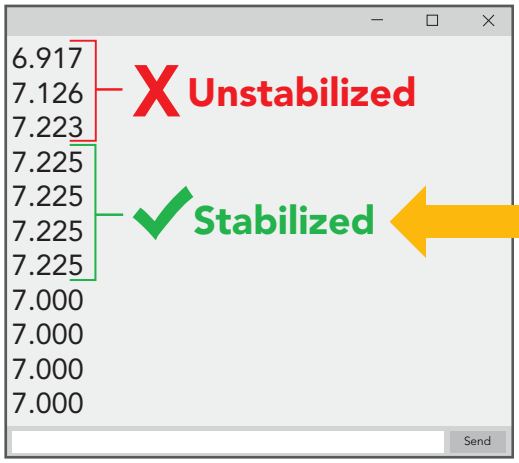
# Mid point calibration

Remove the soaker bottle and rinse off the pH probe. Remove the top of the pH **7.00** calibration solution pouch. Place the pH probe inside the pouch and let the probe sit in the calibration solution until the readings stabilize *(small movement from one reading to the next is normal)*.







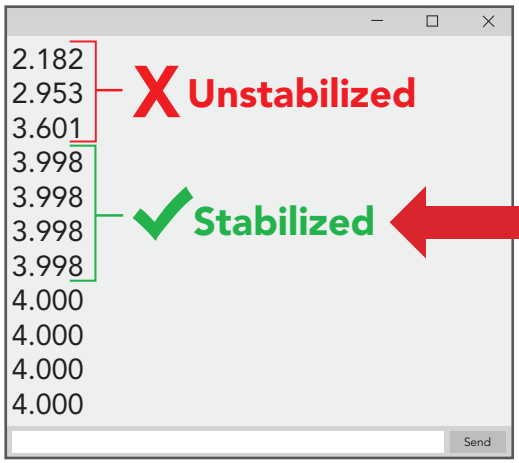

Once the readings have stabilized, calibrate to pH 7.

*After 20 mins, the calibration solution inside an open pouch is no longer considered accurate.*

*Dispose of the unused solution, after calibration.*

# Low point calibration

- Rinse off the probe before calibrating to the low point.
- Open the pouch of pH **4.00** calibration solution, and place probe inside the pouch.
- Wait for readings to stabilize *(1 – 2 minutes)*.




Once the readings have stabilized, calibrate to pH 4.

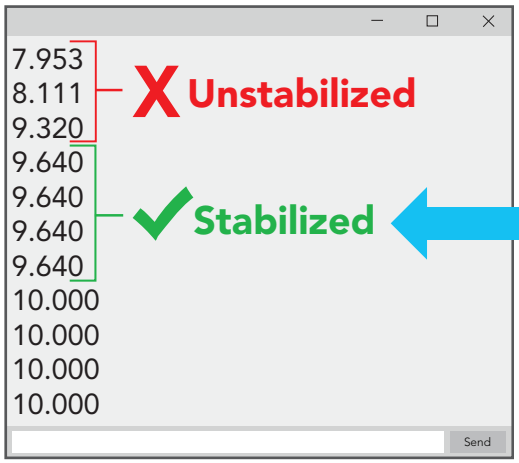
*After 20 mins, the calibration solution inside an open pouch is no longer considered accurate.*

*Dispose of the unused solution, after calibration.*

# High point calibration

- Rinse off the probe before calibrating to the high point.
- Open the pouch of pH **10.00** calibration solution, and place probe inside the pouch.
- Wait for readings to stabilize *(1 – 2 minutes)*.




Once the readings have stabilized, calibrate to pH 10.

*After 20 mins, the calibration solution inside an open pouch is no longer considered accurate.*

*Dispose of the unused solution, after calibration.*

The pH OEM™ circuits default temperature compensation is set to 25° C. If the temperature of the calibration solution is +/- 2° C from 25° C, consider setting the temperature compensation first. ***Temperature changes of < 2° C are insignificant.***

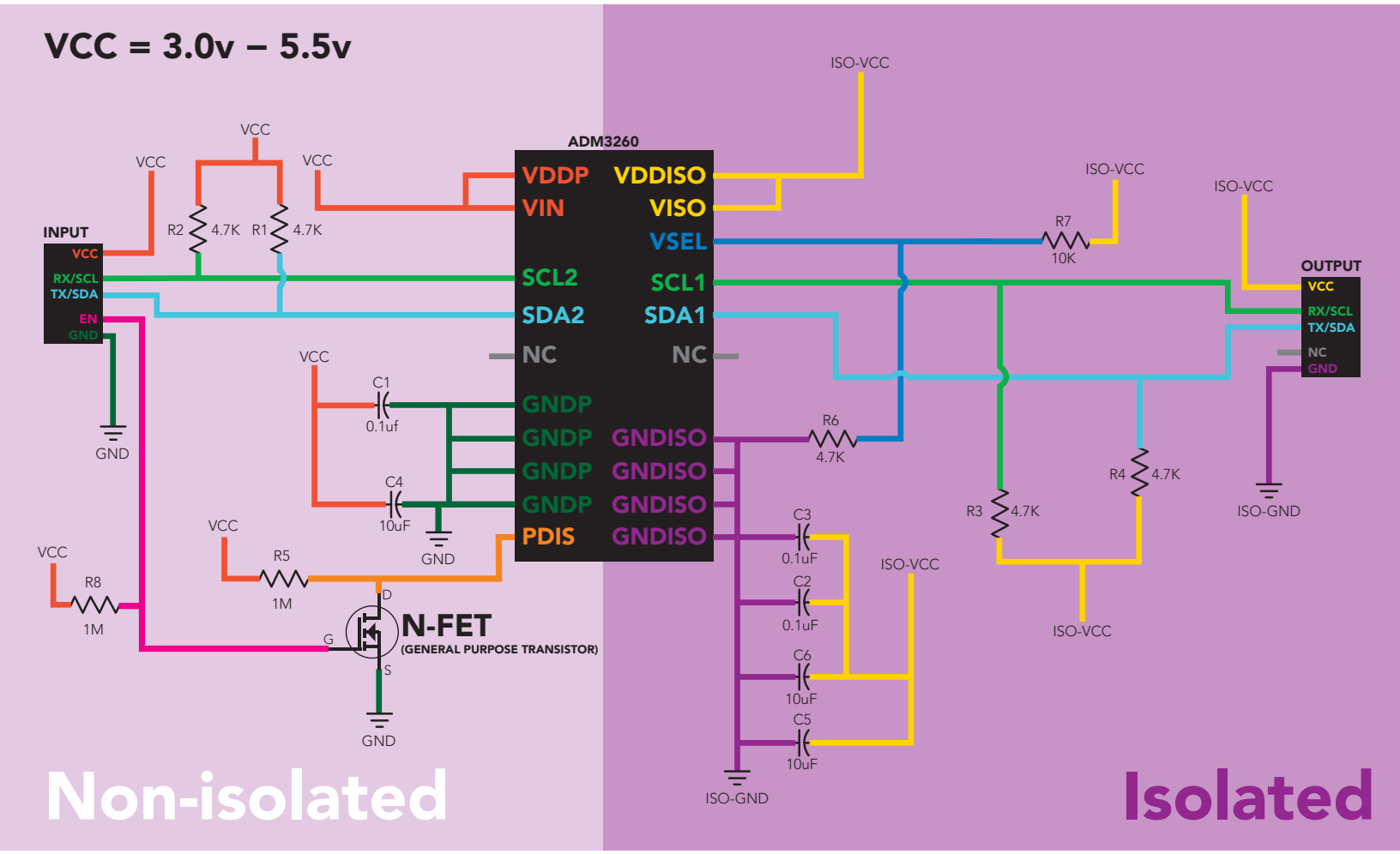
# OEM electrical isolation

If the pH OEM™ Class Embedded Circuit is going to be used in consumer, industrial, or scientific/medical applications electrical isolation is strongly recommended. Electrically isolating the device will insure that the readings are accurate, the pH probe does not interfere with other sensors and that outside electrical noise does not affect the device.

The goal of electrically isolating the pH OEM™ device is to insure that the device no longer shares a common ground with the master CPU, other sensors and other devices that are can be traced back to a common ground. It is important to keep in mind that simply isolating the power and ground is not enough. Both data lines (SDA, SCL) and the INT pin must also be isolated.

This technology works by using tiny transformers to induce the voltage across an air gap. PCB layout requires special attention for EMI/EMC and RF Control, having proper ground planes and keeping the capacitors as close to the chip as possible are crucial for proper performance. The two data channels have a 4.7kΩ pull up resistor on both the isolated and non-isolated lines (R1, R2, R3, and R4) The output voltage is set using a voltage divider (R6 and R7) this produces a voltage of 3.9V regardless of your input voltage.

**Isolated ground is different from non-isolated ground, these two lines should not be connected together.**

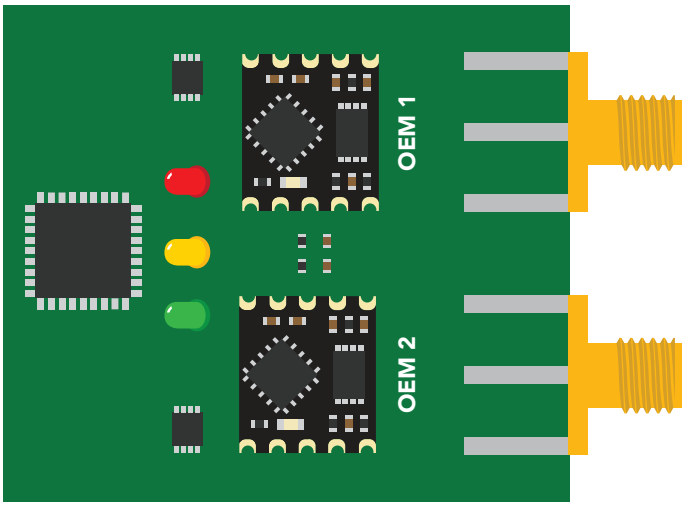


# Designing your product

The pH OEM™ circuit is a sensitive device. Special care ***MUST*** be taken to ensure your pH readings are accurate.

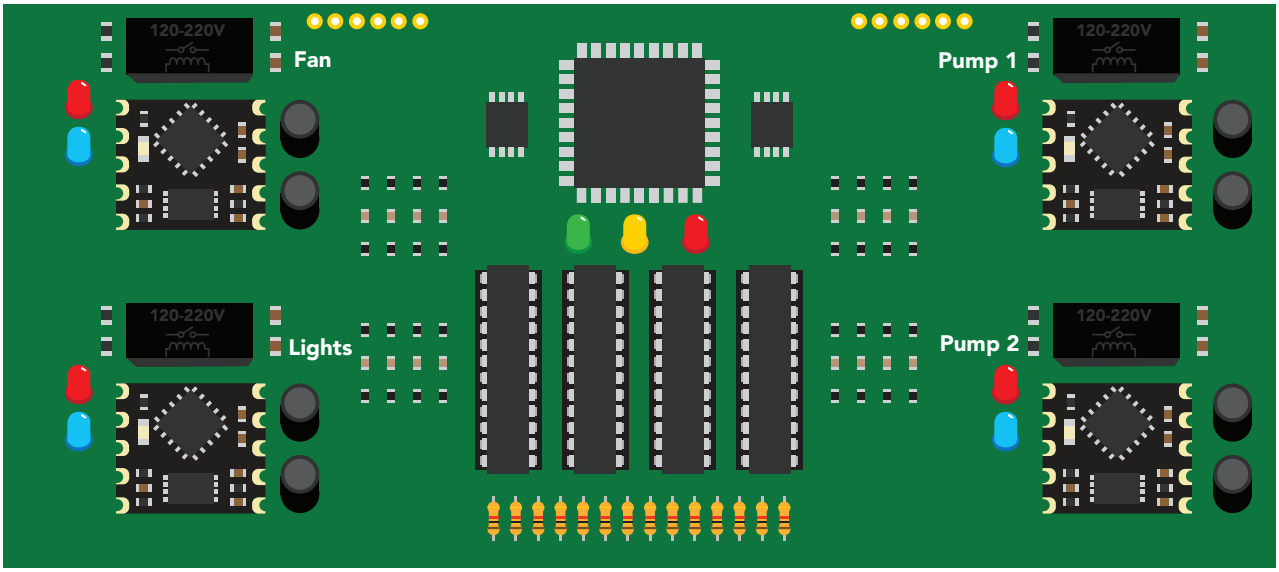
## Simple design

Simple low voltage computer systems experience little to no problems during development and have no reported issues from the target customer.




## Complex design

Complex computer systems with multiple voltages and switching, can lead to extended and unnecessary debugging time. Target customers can experience frequent accuracy issues.

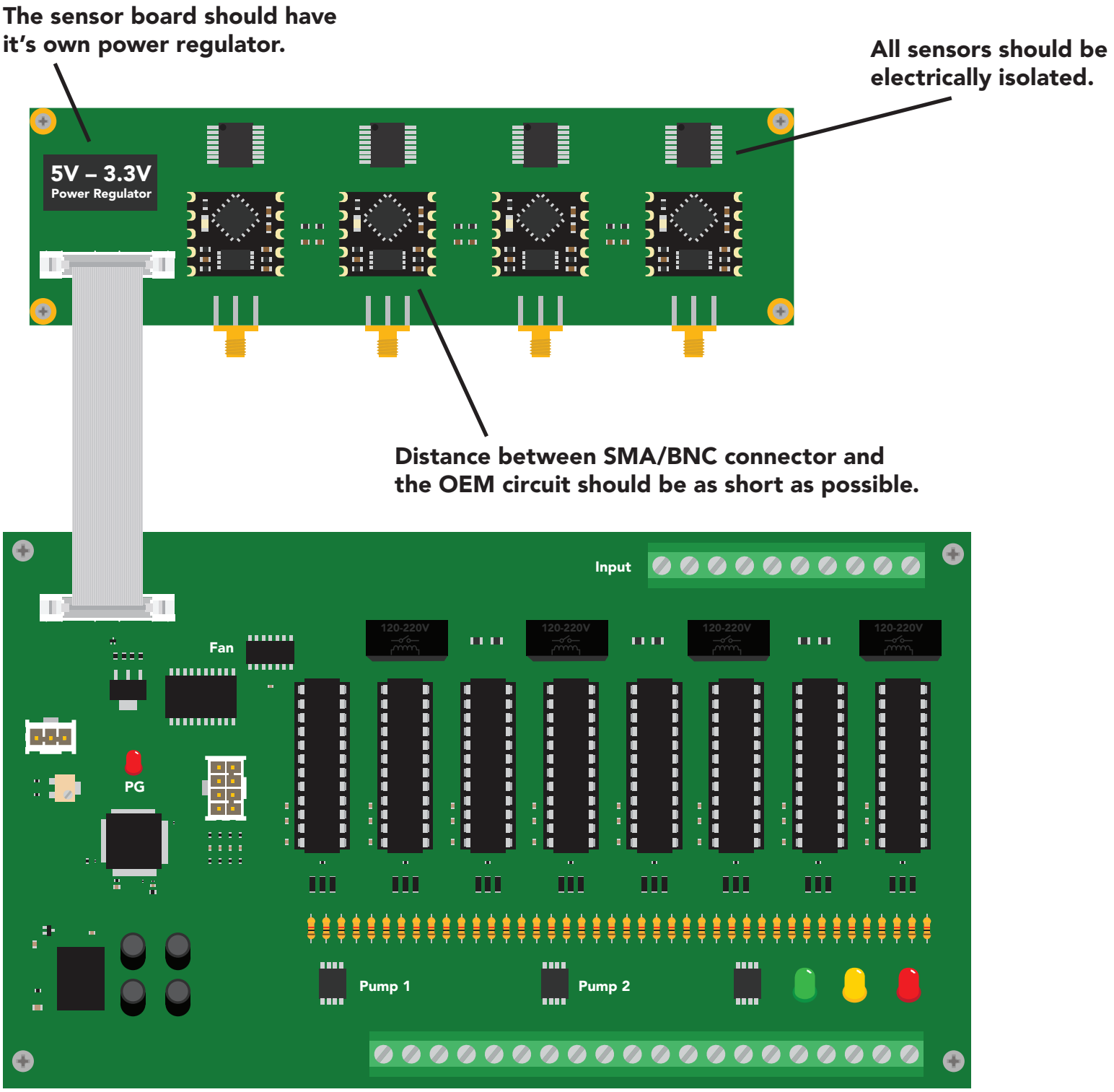


# How to add chemical sensing to a complex computer system

Placing the OEM™ circuits onto their own board is ***strongly recommended***; Not only does this help keep the design layout simple and easy to follow, it also significantly reduces debugging and development time.

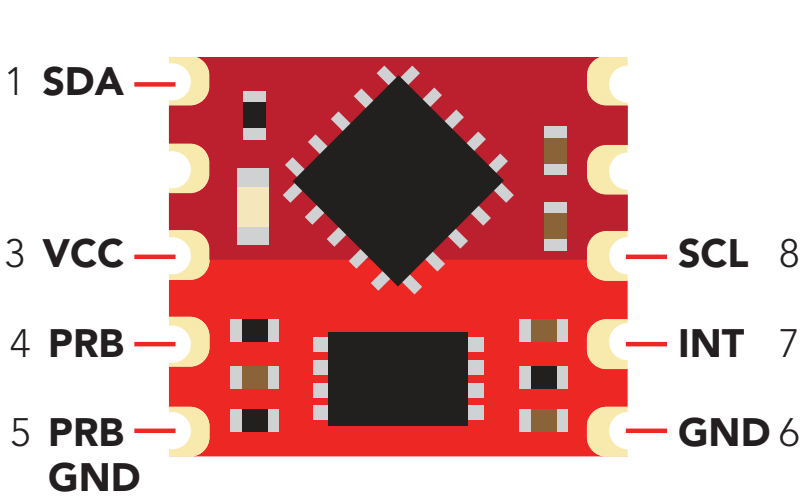
Target customers will experience accurate, stable and repeatable readings for the life of your product.

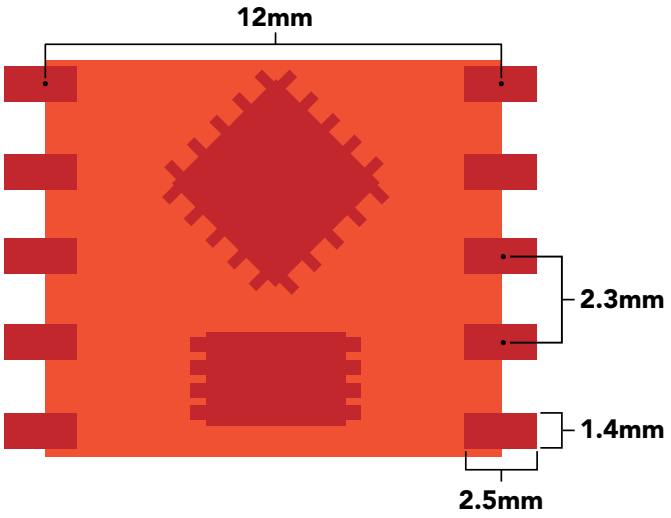


AtlasScientific™ Environmental Robotics

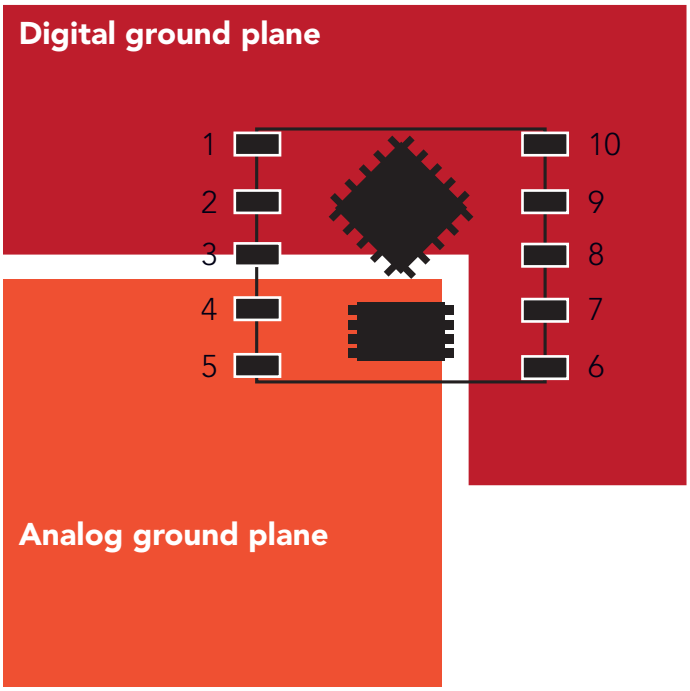
# Designing your PCB



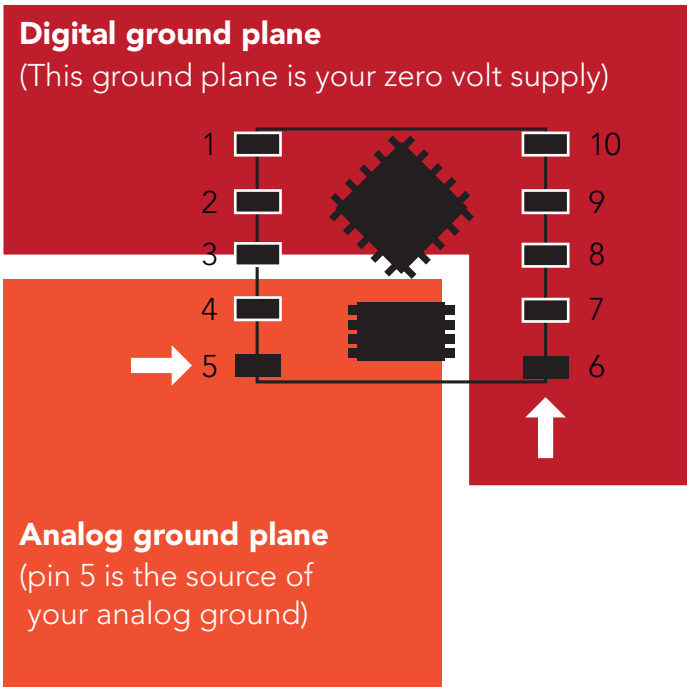




The pH OEM™ circuit requires two separate ground planes to operate properly.
One ground plane is for the digital section of the device, the other is for the analog section.

**1** Create two double-sided ground planes, just like the image below.




**2** Connect pin 5 to the analog ground plane, and pin 6 to the digital ground plane.

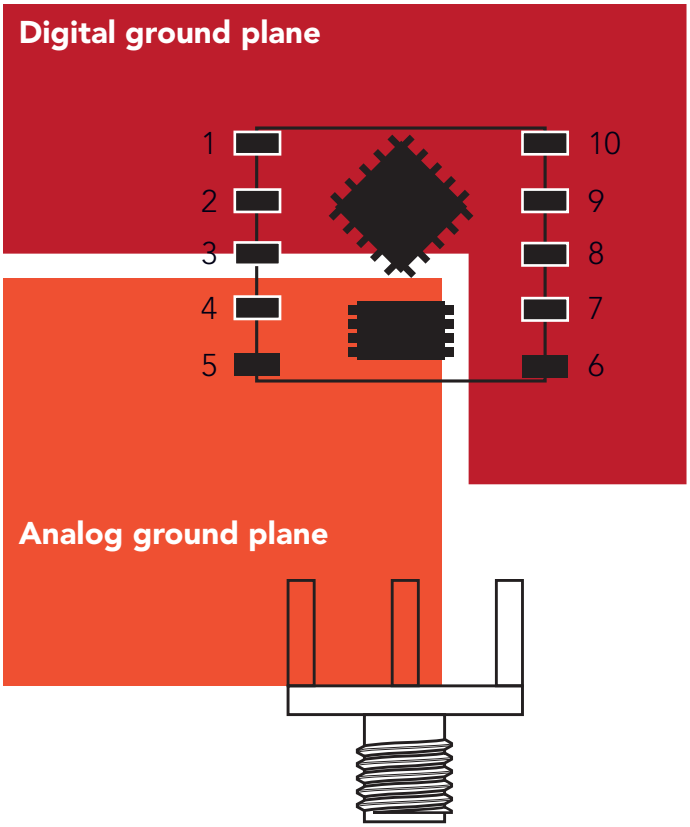


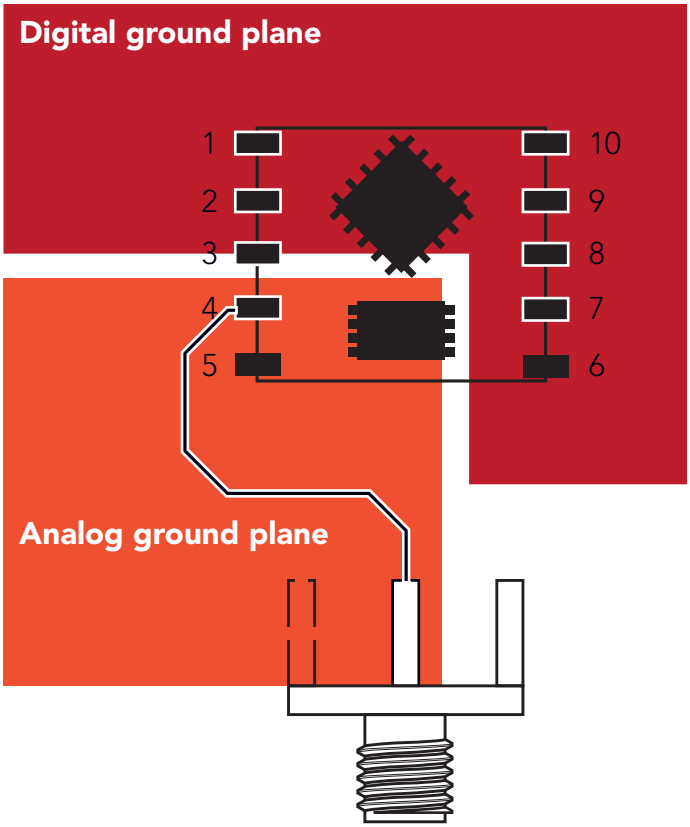

 

AtlasScientific™ Environmental Robotics

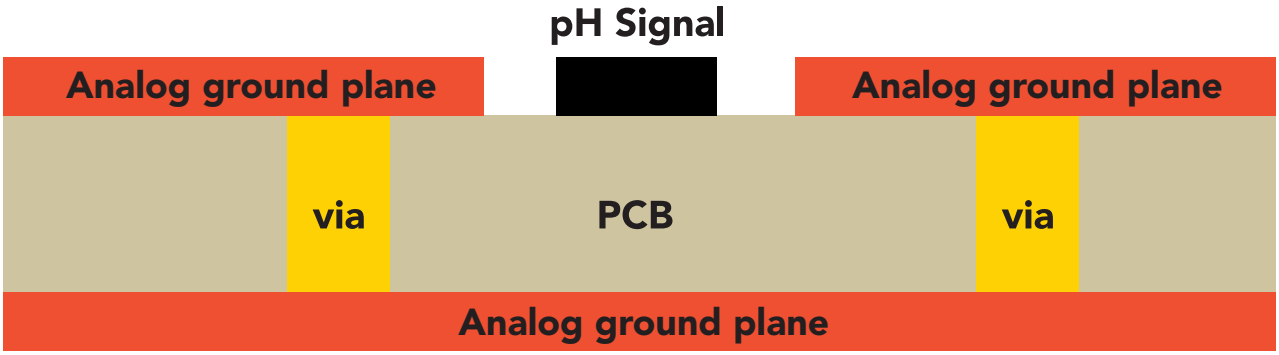
**3** Place the probe connector (BNC/SMA) close to the pH OEM™ circuit.

**4** Using a 0.4mm trace width connect pin 4 (PRB) to pin 1 on the BNC/SMA. Keep this trace as short as possible. This trace is the pH signal path.



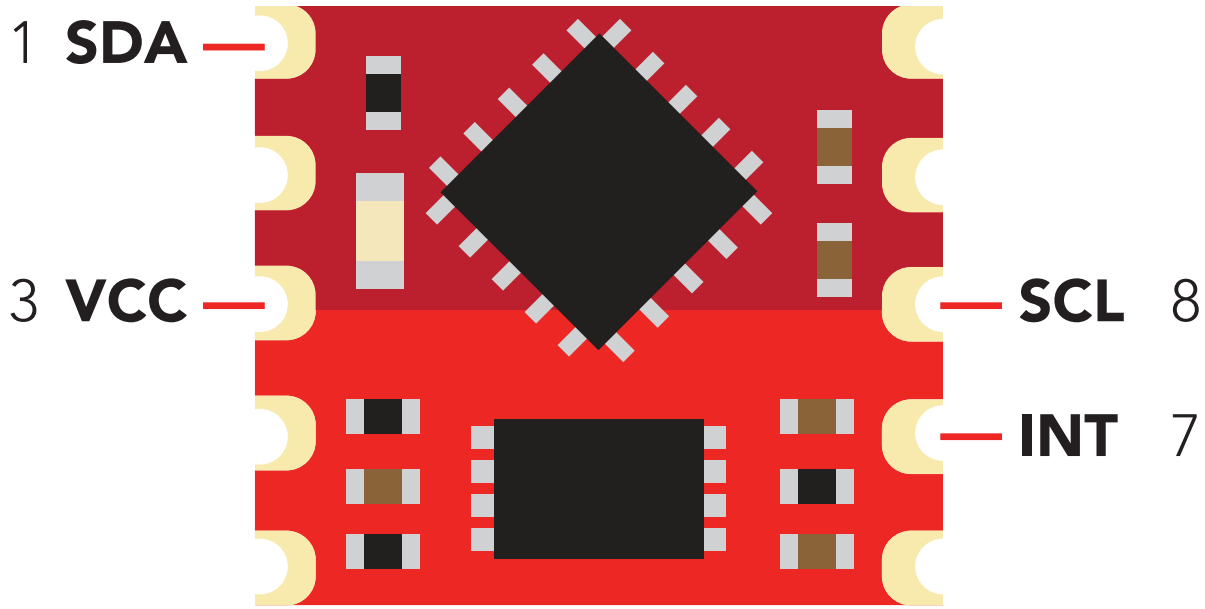




## Cross section of the pH signal path

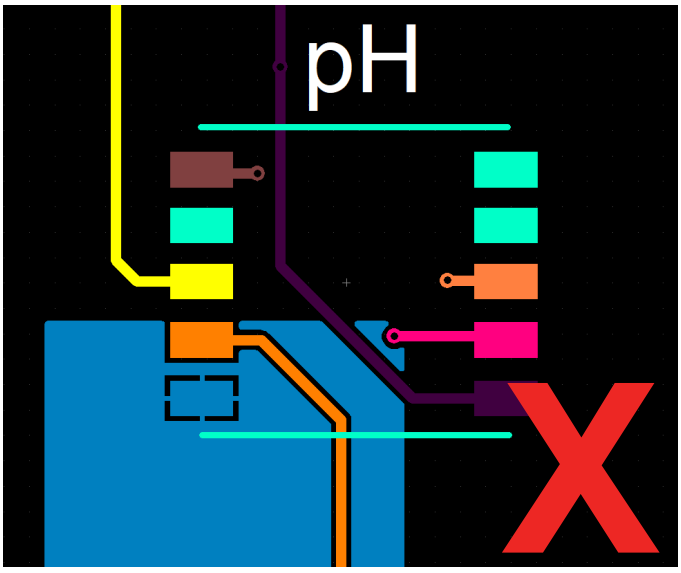


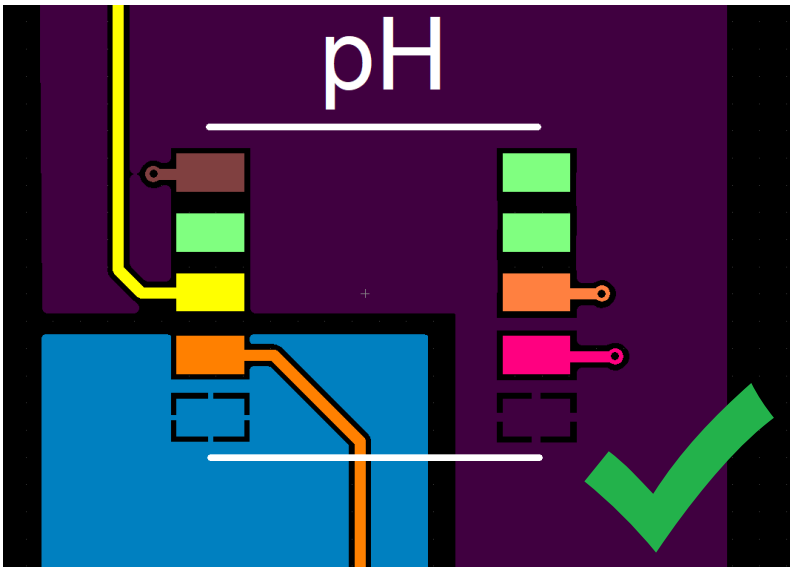

This cross section is an example of how the analog ground plane protects the pH signal. The analog ground should surround the pH signal, on both the top and bottom layers.

**5** Rout the other traces as you see fit. If pin 7(INT) is unused leave it floating, *do not connect pin 7 to VCC or ground.*



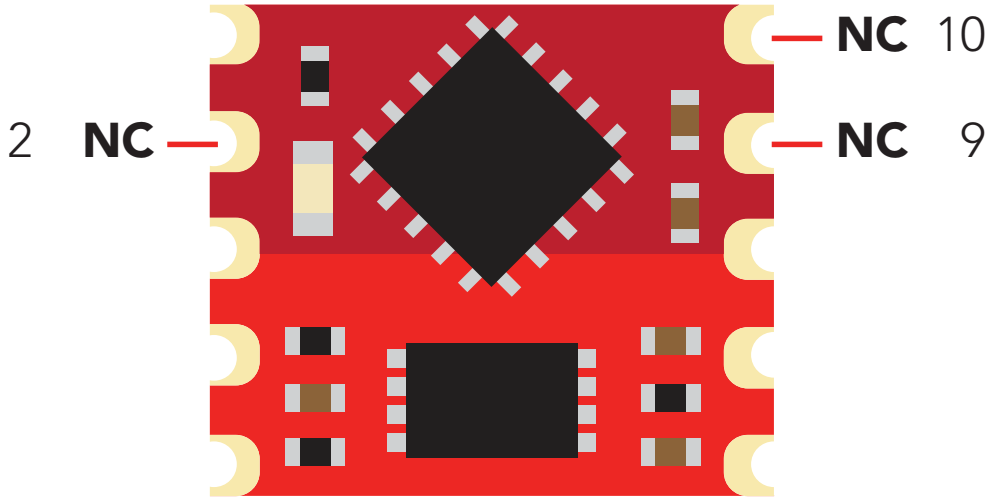

## *NEVER* place vias under the OEM footprint.



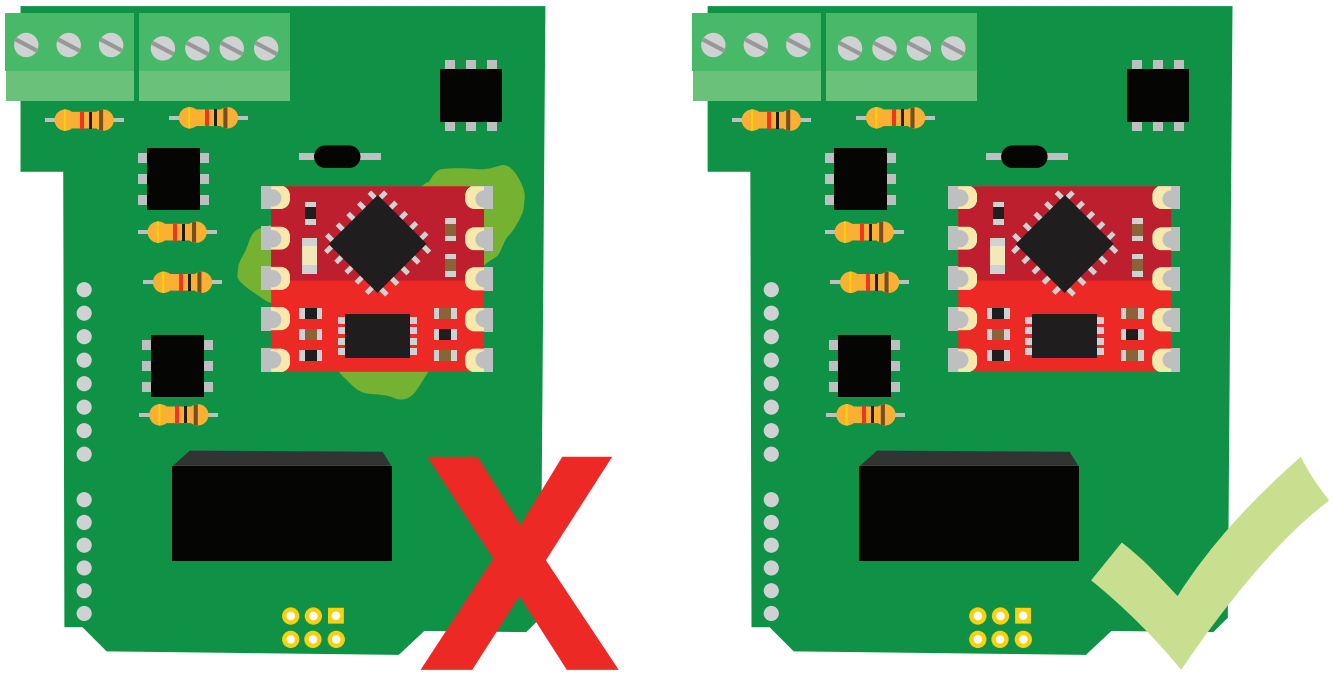



**6** Pins marked NC *(No Connect)* must be left floating. ***NEVER*** *connect pins marked NC to VCC or ground.*



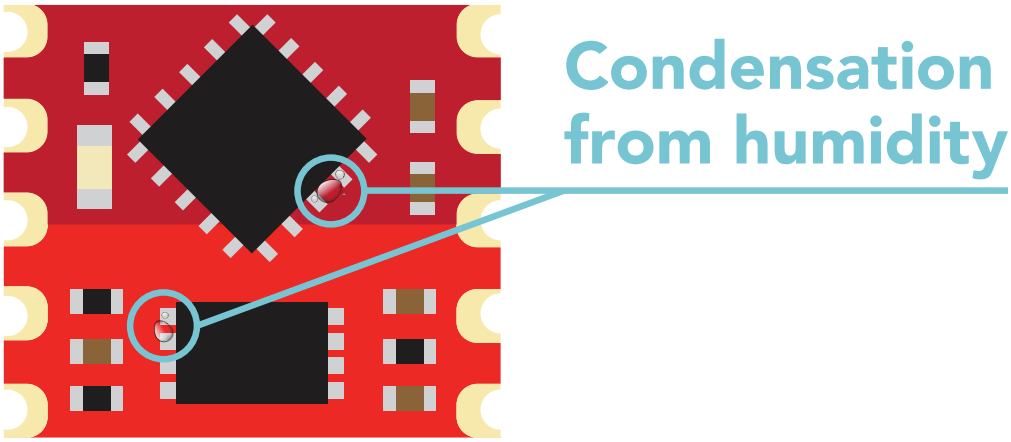

**7** If the pH OEM™ circuit is going to be hand soldered, avoid using rosin core solder. Use as little flux as possible. Do not let liquid flux seep under the pH OEM™ circuit. After the pH OEM™ circuit has been soldered to the PCB all flux residue *MUST* be removed. Failure to do so will result in poor quality readings.



***DO NOT SKIP THIS STEP***
The PCB must be washed with an ultrasonic cleaner, OR cleaned with a commercial flux removing chemical, OR soaked in alcohol for ~20 minutes.

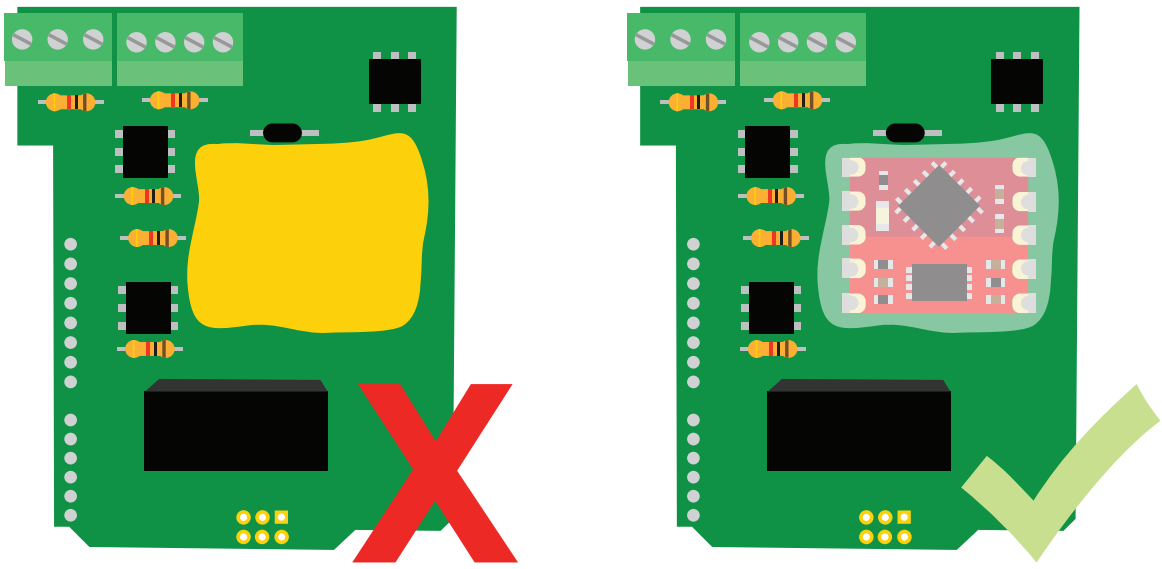
# Humidity shielding

Humidity levels ***above 65%*** can affect the pH readings. If you believe the target customer will operate the equipment in a humid environment, act accordingly.



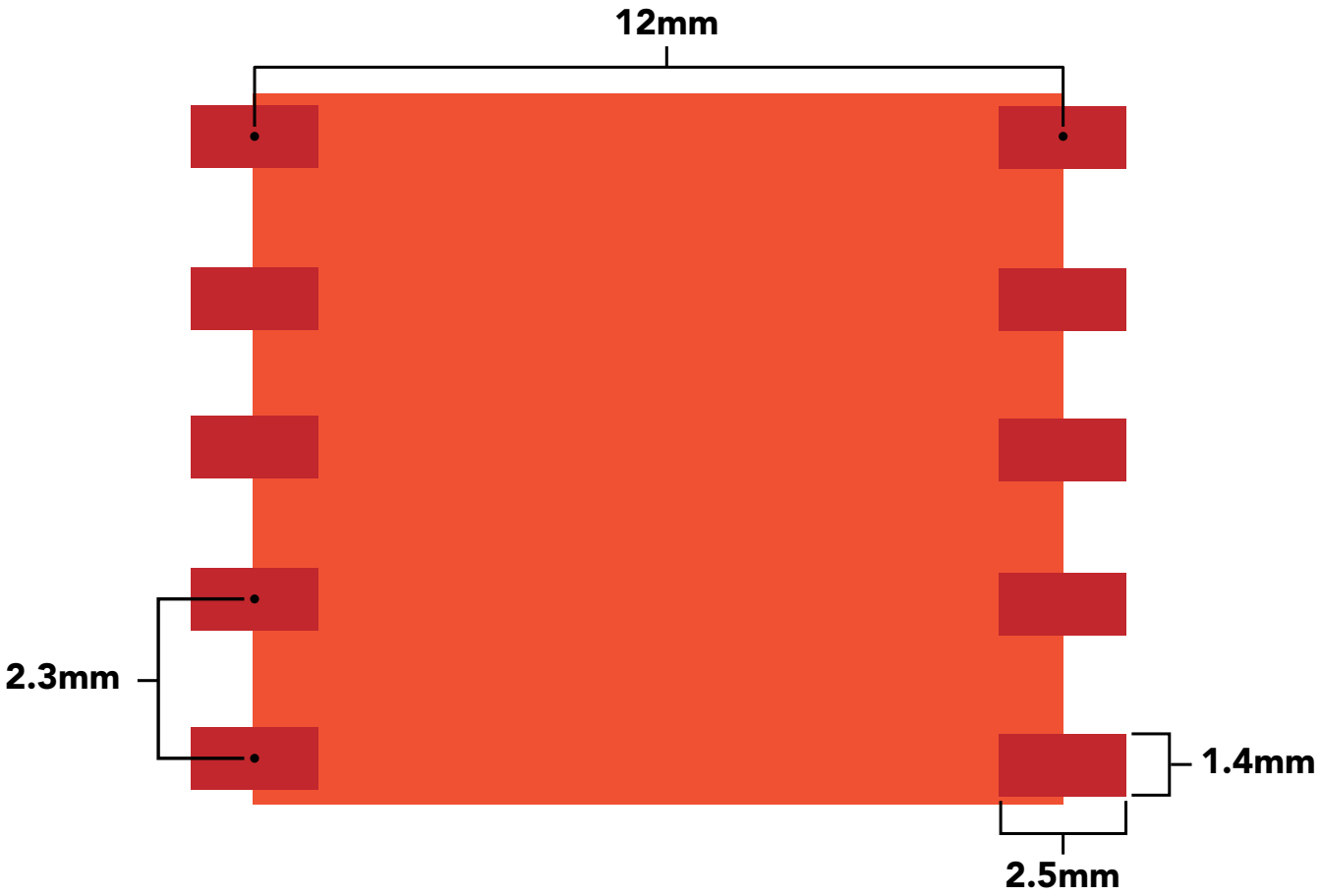

## Shielding process:

**1.** Solder the OEM-pH circuit to your PCB.
**2.** Wash the board to ensure that all flux has been removed.
**3.** Test to ensure the OEM-pH circuit is delivering accurate pH readings.
**4.** Encase the OEM-pH circuit in ***clear epoxy***.



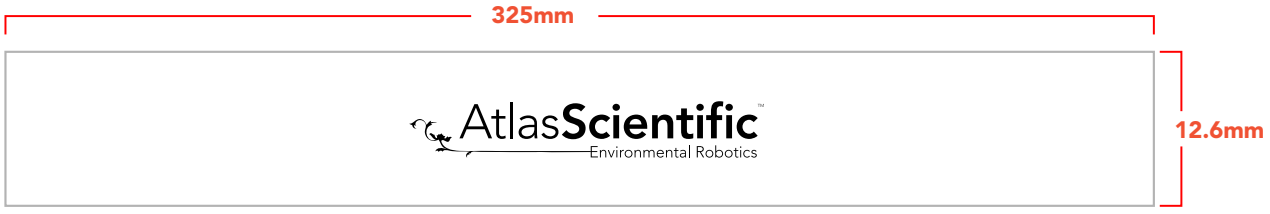
***Only Use Clear Epoxy!***
Do not use colored epoxy. Do not use black potting compound designed for electronics. Using colored epoxy or black potting compound will severely damage the OEM-pH circuit.

# Recommended pad layout

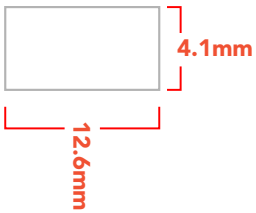



# IC tube measurements

## Top View




## Side View




**Fits 25 pH OEM™ circuits**

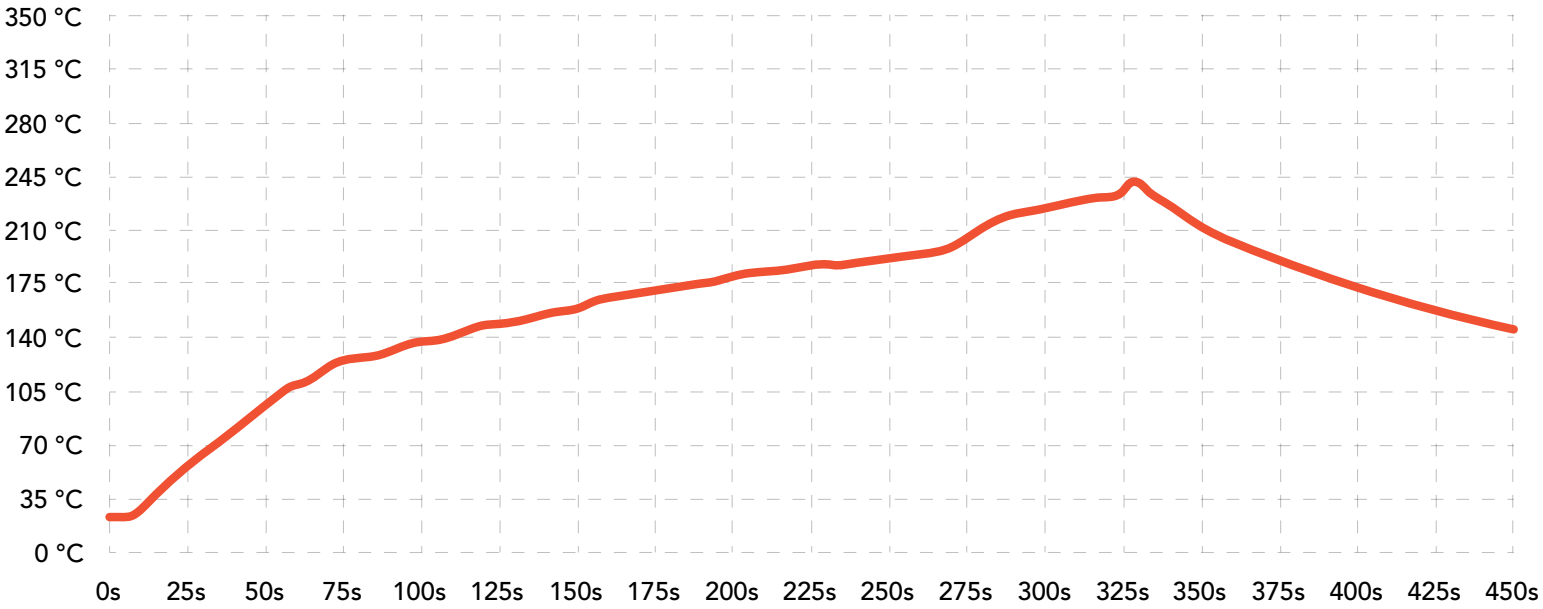
| | inside dimensions | outside dimensions |
|---|---|---|
| L | 325mm | 325mm |
| W | 11.6mm | 12.6mm |
| H | 3.1mm | 4.1mm |

**plastic thickness 0.5mm**



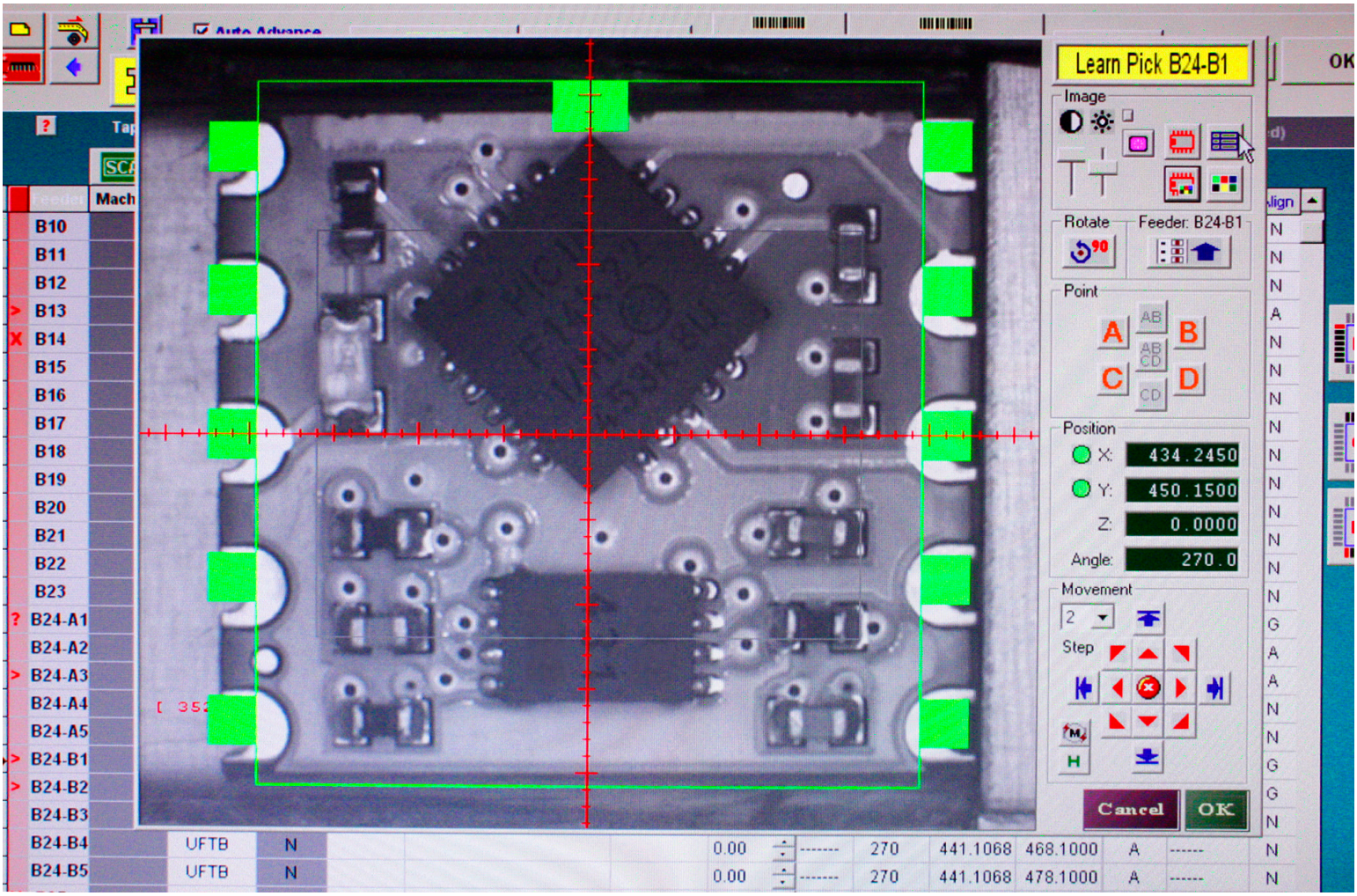

AtlasScientific™ Environmental Robotics

# Recommended reflow soldering profile




| # | Temp | Sec | # | Temp | Sec | # | Temp | Sec | # | Temp | Sec |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 30 | 15 | **11** | 163 | 10 | **21** | 182 | 10 | **31** | 100 | 25 |
| **2** | 90 | 20 | **12** | 165 | 10 | **22** | 183 | 10 | **32** | 80 | 30 |
| **3** | 110 | 8 | **13** | 167 | 10 | **23** | 185 | 10 | **33** | 30 | 30 |
| **4** | 130 | 5 | **14** | 170 | 10 | **24** | 187 | 10 | **34** | 0 | 15 |
| **5** | 135 | 5 | **15** | 172 | 10 | **25** | 220 | 30 | | | |
| **6** | 140 | 5 | **16** | 174 | 10 | **26** | 225 | 20 | | | |
| **7** | 155 | 8 | **17** | 176 | 10 | **27** | 230 | 20 | | | |
| **8** | 156 | 10 | **18** | 178 | 10 | **28** | 235 | 8 | | | |
| **9** | 158 | 10 | **19** | 180 | 10 | **29** | 170 | 20 | | | |
| **10** | 160 | 10 | **20** | 181 | 10 | **30** | 130 | 20 | | | |

# Pick and place usage




# Datasheet change log

### Datasheet V 4.5

Added information on humidity and the OEM circuit on pg 36.

### Datasheet V 4.3

Added new graphic on pg 3.

### Datasheet V 4.2

Revised operating voltages on pages 1 & 5.

### Datasheet V 4.1

Added a page about flux removal on pg 3.

**Datasheet V 4.0**

Revised artwork on pg 7.

**Datasheet V 3.9**

Added "Calibration theory" on pg 24.

**Datasheet V 3.8**

Added "Designing you product" on pg 25.

**Datasheet V 3.7**

Revised calibration information on pg. 19

**Datasheet V 3.6**

Changed calibration confrim register 0x0D from R/W to R.

**Datasheet V 3.5**

Expanded upon the "Designing your PCB" section of datasheet, pg. 25

**Datasheet V 3.4**

Revised isolation schematic on pg. 24

**Datasheet V 3.3**

Changed "Max rate" to "Response time" on cover page.

**Datasheet V 3.2**

Revised temperature compensation register information.

**Datasheet V 3.1**

Corrected max rate reading on cover page.

**Datasheet V 3.0**

Revised entire datasheet

# Firmware updates

V4.0 – Initial release (July 7, 2015)