Atlas Scientific™
Environmental Robotics

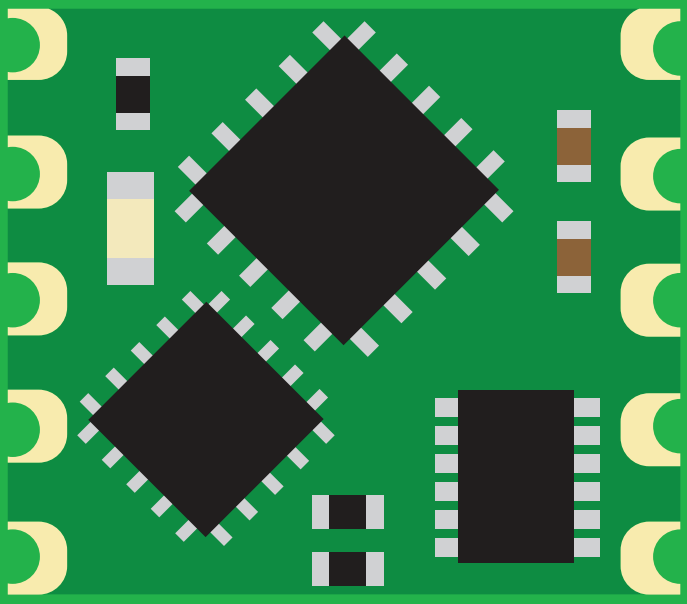
V 2.9

Revised 3/21

OEM-EC™

Embedded Conductivity Circuit

Reads	**Conductivity** **Total dissolved solids *(ppm)*** **Salinity = PSU (ppt) 0.00 – 42.00**
Range	**0.07 – 500,000+ µS/cm**
Accuracy	**+/ – 2%**
Response time	**1 reading every 640ms**
Supported probes	**K 0.01 – K 600** **any brand**
Calibration	**1 or 2 point**
Temp compensation	**Yes**
Data protocol	**SMBus/I²C**
Default I²C address	**0x64**
Operating voltage	**3.3V – 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.

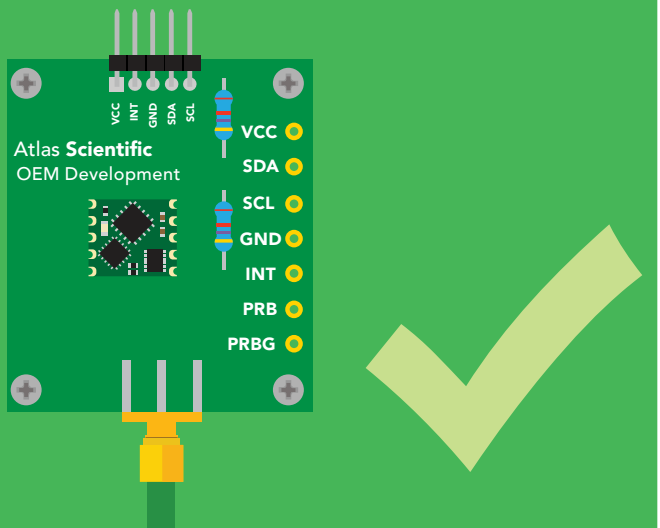
STOP

SOLDERING THIS DEVICE VOIDS YOUR WARRANTY.

Before purchasing the Conductivity 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.

Get this device working in our OEM Development board first!



Do not solder wires to this device.

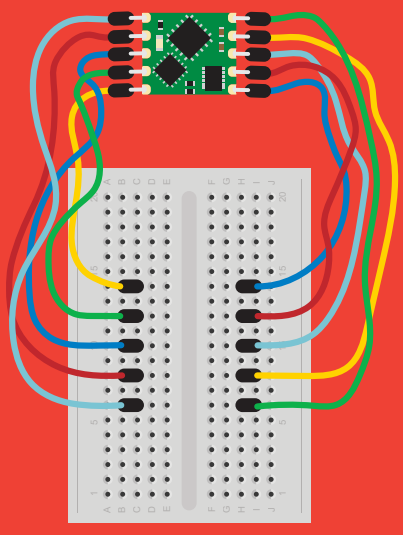
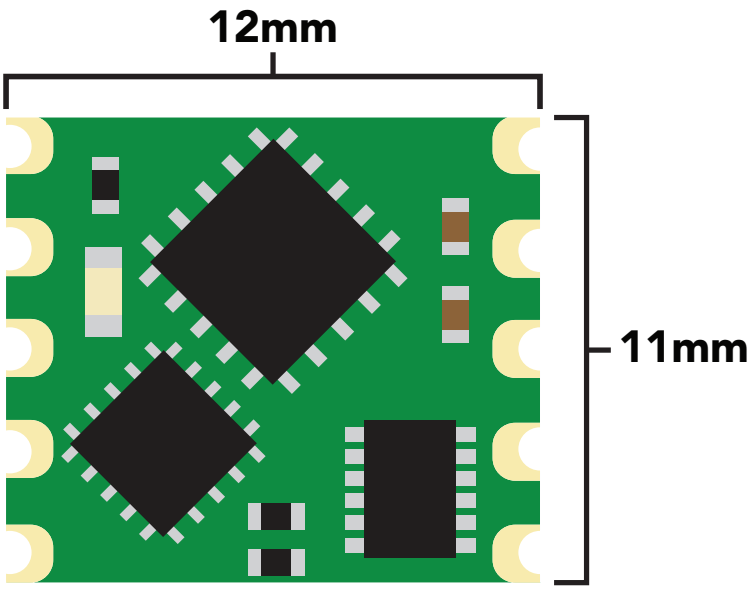


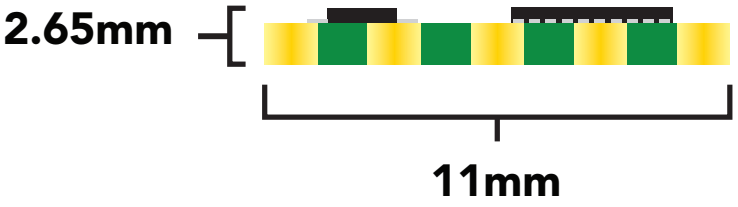
Table of contents

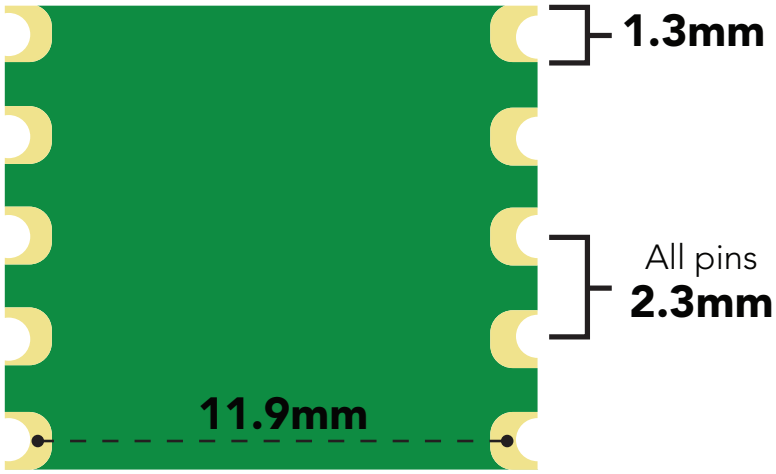
REGISTERS

Atlas Scientific™ Environmental Robotics

OEM circuit dimensions







Center to center

Absolute max ratings

Parameter	MIN	TYP	MAX
Storage temperature	-60 °C		150 °C
Operational temperature	-40 °C	25 °C	125 °C
VCC	3.3V	5V	5.5V

Power consumption

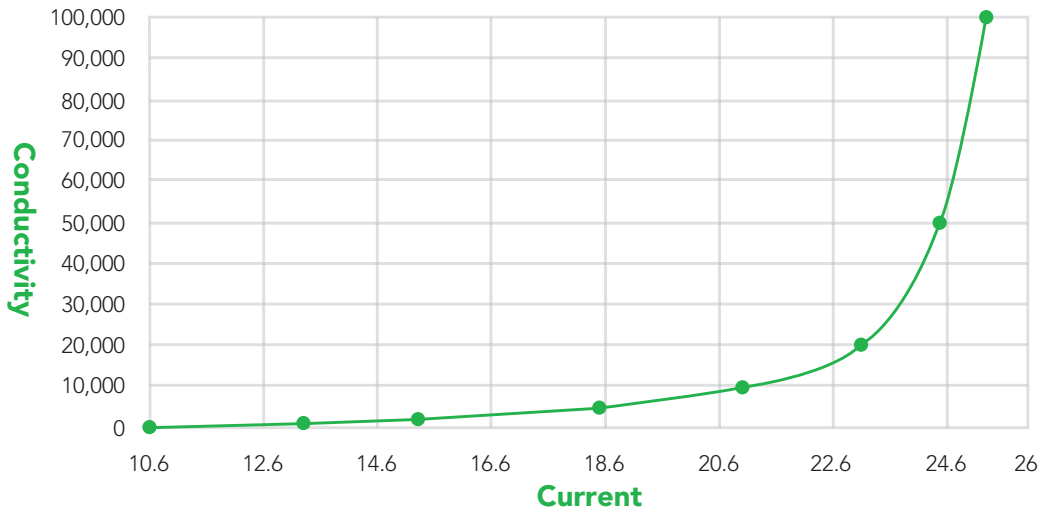
The current used by the EC OEM™ is not constant. It changes as the conductivity of the water changes. For example, if the water has a conductivity of 0, then no electricity is flowing through the probe, and the current draw will only be what it takes to power the CPU.

However, if the water has a very high conductivity, then the current draw is higher. After 100,000µs, the current consumption no longer increases *(this does not mean the readings are limited to 100,000µs).*

Current usage at 3.3V

Min = 10.6 mA
Max = 26 mA

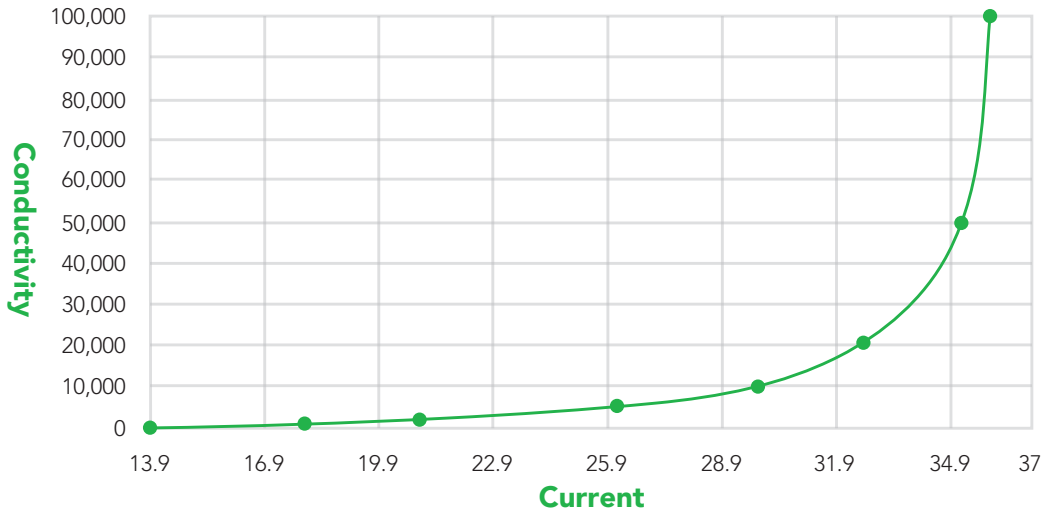
Current	Conductivity
10.6 mA	0µs
13.3 mA	1,000µs
15.3 mA	2,000µs
18.5 mA	5,000µs
21.0 mA	10,000µs
23.1 mA	20,000µs
24.5 mA	50,000µs
25.3 mA	100,000µs



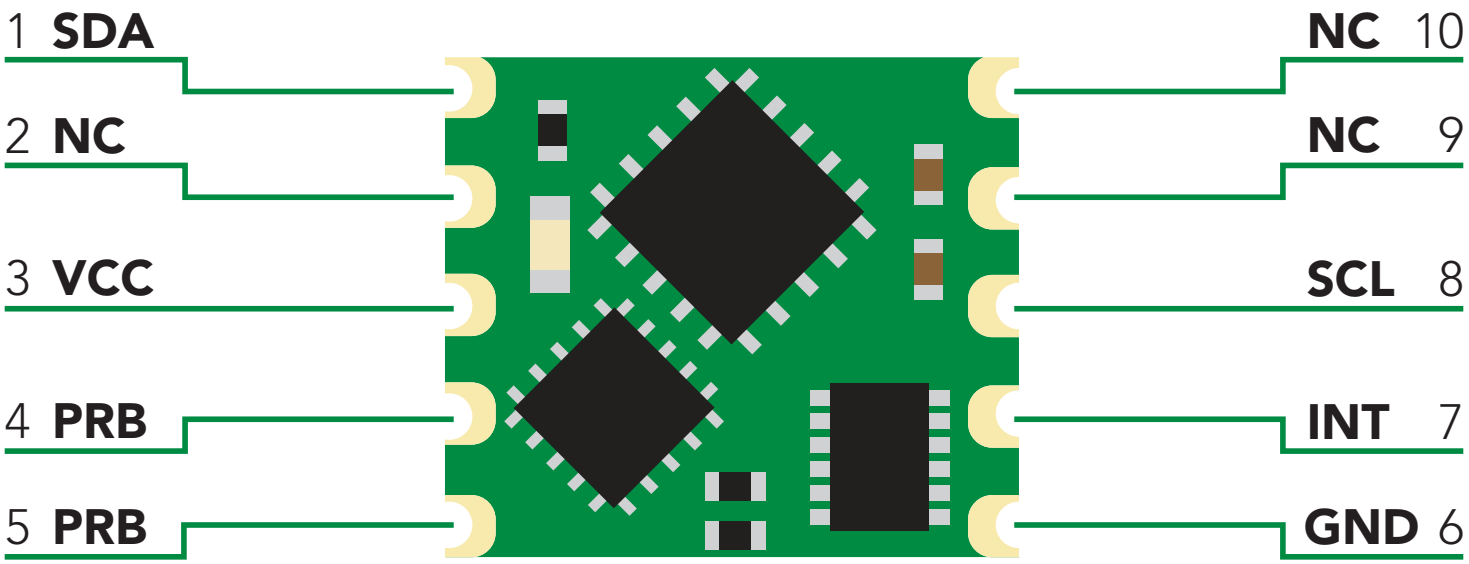
Current usage at 5V

Min = 13.9 mA
Max = 37 mA

Current	Conductivity
13.9 mA	0µs
18.0 mA	1,000µs
21 mA	2,000µs
26 mA	5,000µs
30 mA	10,000µs
33 mA	20,000µs
35.2 mA	50,000µs
36.2 mA	100,000µs



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™ EC OEM™ will always produce a reading with a resolution of two decimal places.

Example
0.07µS
150,234.78µS

Power on/start up

Once the Atlas Scientific™ EC 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 EC OEM™ Class Embedded Conductivity Circuit is the core electronics needed to read the electrical conductivity of water from a wide range of conductivity probe types (K 0.01 to K 600). The EC OEM™ Embedded Conductivity Circuit will meet, or exceed the capabilities and accuracy found in all models of bench top laboratory grade conductivity meters.

The EC OEM™ is an SMBus / I²C slave device that communicates to a master device at a speed of 10 – 100 kHz. Read and write operations are done by accessing **36** different 8 bit registers. A user controllable LED and output pin can be used for debugging and interrupt signaling.

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

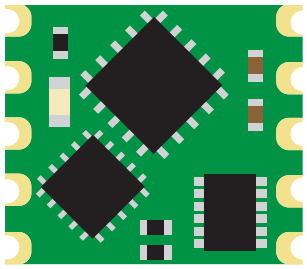
0x04: Interrupt control	R/W
0x05: LED control	R/W
0x06: Active/hibernate	R/W
0x07: New reading available	R/W

Probe type

0x08: Set probe type MSB	R/W
0x09: Set probe type LSB	R/W

Calibration

0x0A: Calibration value MSB	R/W
0x0B: Calibration value high byte	R/W
0x0C: Calibration value low byte	R/W
0x0D: Calibration value LSB	R/W
0x0E: Calibrate request	R/W
0x0F: Calibration confirm	R



Read only R
Read and write R/W

Compensation

0x10: Temperature compensation MSB	R
0x11: Temperature compensation high byte	R
0x12: Temperature compensation low byte	R
0x13: Temperature compensation LSB	R

Confirmation

0x14: Temperature confirm MSB	R
0x15: Temperature confirm high byte	R
0x16: Temperature confirm low byte	R
0x17: Temperature confirm LSB	R

Sensor Data

0x18: EC reading MSB	R
0x19: EC reading high byte	R
0x1A: EC reading low byte	R
0x1B: EC reading LSB	R
0x1C: TDS reading MSB	R
0x1D: TDS reading high byte	R
0x1E: TDS reading low byte	R
0x1F: TDS reading LSB	R
0x20: Salinity reading MSB	R
0x21: Salinity reading high byte	R
0x22: Salinity reading low byte	R
0x23: Salinity reading LSB	R

The default device address is **0x64**
This address can be changed.

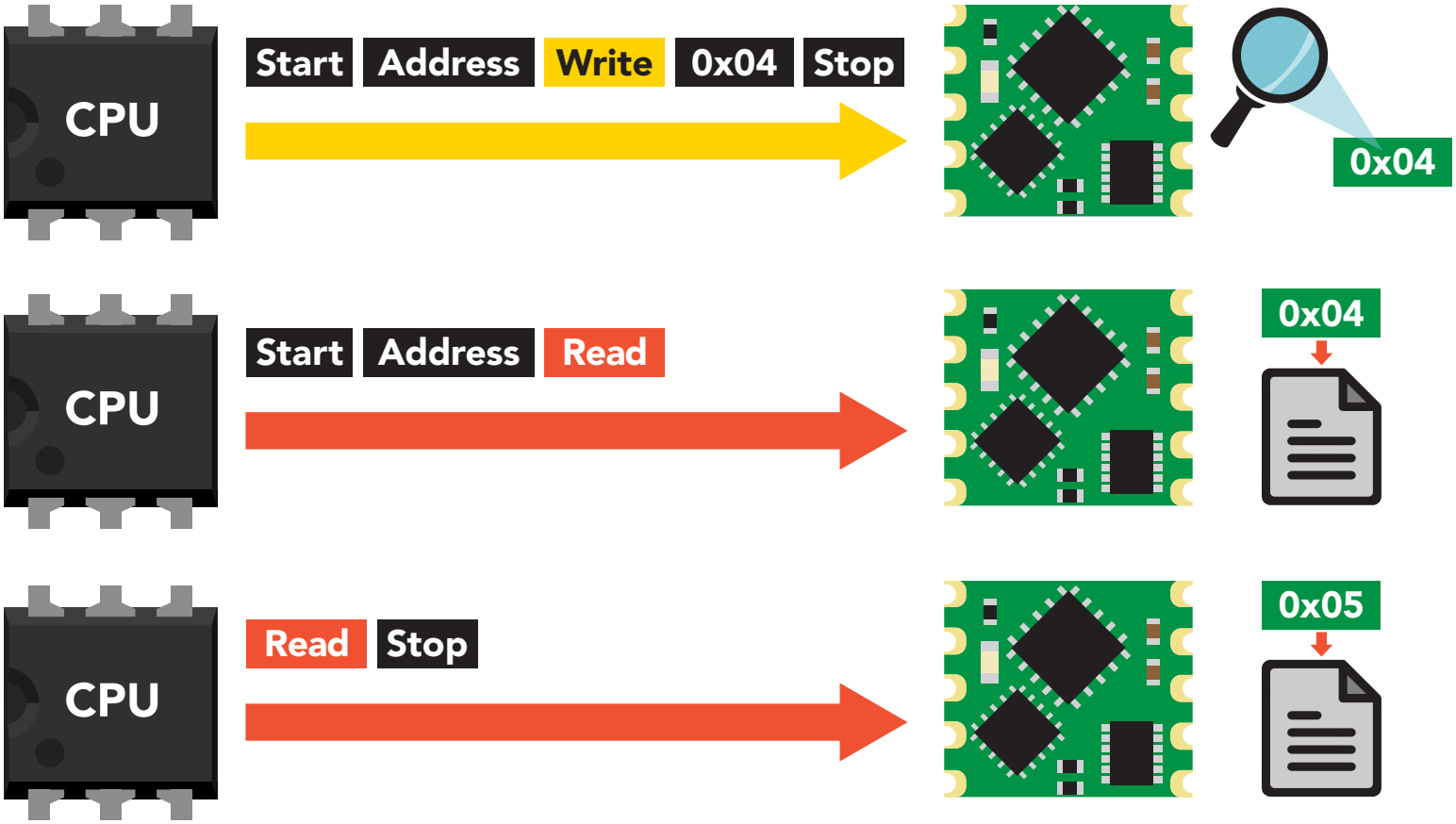
Each Conductivity reading takes 640ms

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 **0x64**
This address can be changed.

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



Example code
reading two registers

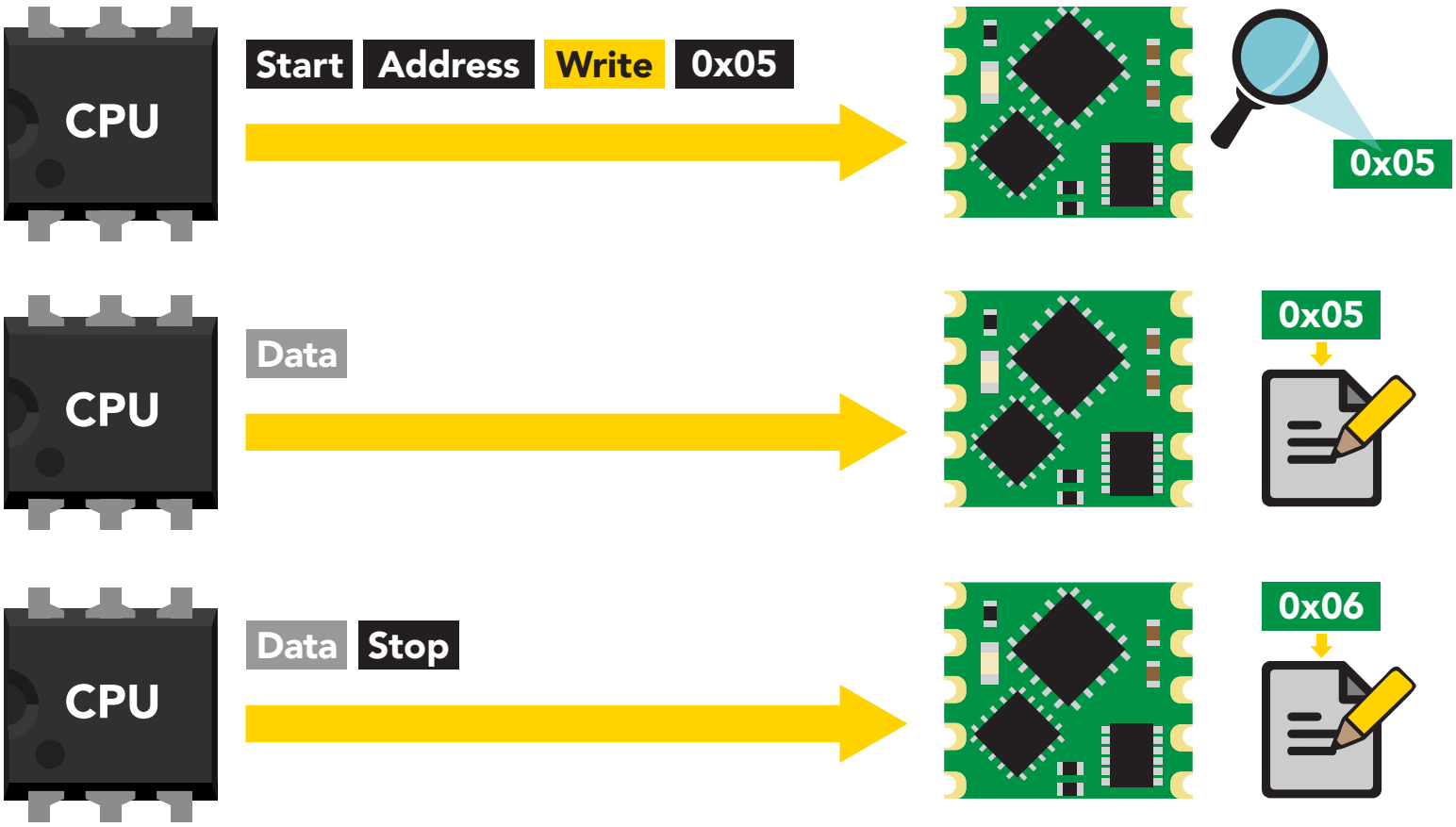
```
byte i2c_device_address=0x64;
byte reg_4, reg_5;

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=0x64;
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.

When transmitting a floating point number to any of these 3 register blocks, the number must first be multiplied by 100. This would have the effect of removing the floating point. Internally the EC OEM™ will divide the number by 100, converting it back into a floating point number.

Example

Setting a conductivity probe type of K = 4.56
4.56 X **100** = 456
Transmit the number 456 to the Set Probe Type Registers

Setting a calibration value of 14.56µs
14.56 X **100** =1456
Transmit the number 1456 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 3 register blocks the value must be divided by 100 to return it to its originally intended value.

Receiving floating point numbers

After receiving a value from any of the reading registers, the number must be divided by 100 to convert it back into a floating point number.

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

Reading an EC value of 14.56µs
Value received = 1456
1456 / **100** =14.56µs

Reading a TDS value of 7.86
Value received = 786
786 / **100** = 7.86 TDS

Reading a Salinity value of 15.84
Value received =1584
1584 / **100** =15.84 PSS

Registers

Device information

0x00: Device type (R)
0x01: Firmware version (R)

0x00 – Device type register

1 unsigned byte
Read only value = 4
4 = EC

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=0x64;
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=0x64;
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 = **0x64**
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=0x64;
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();
```

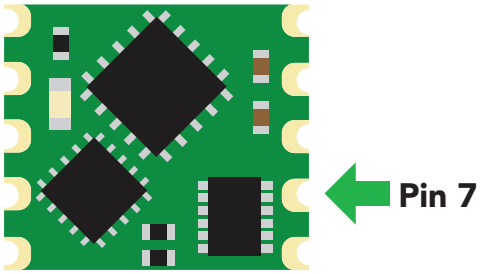

Control registers

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

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)



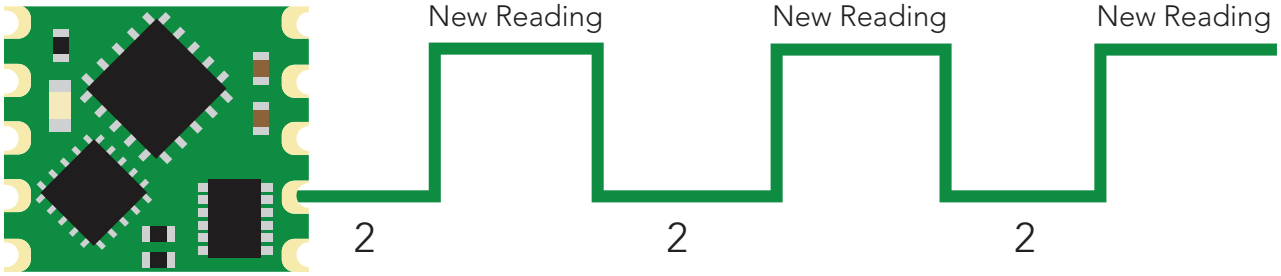
The Interrupt control register adjusts the function of pin 7 (the interrupt output pin).

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.

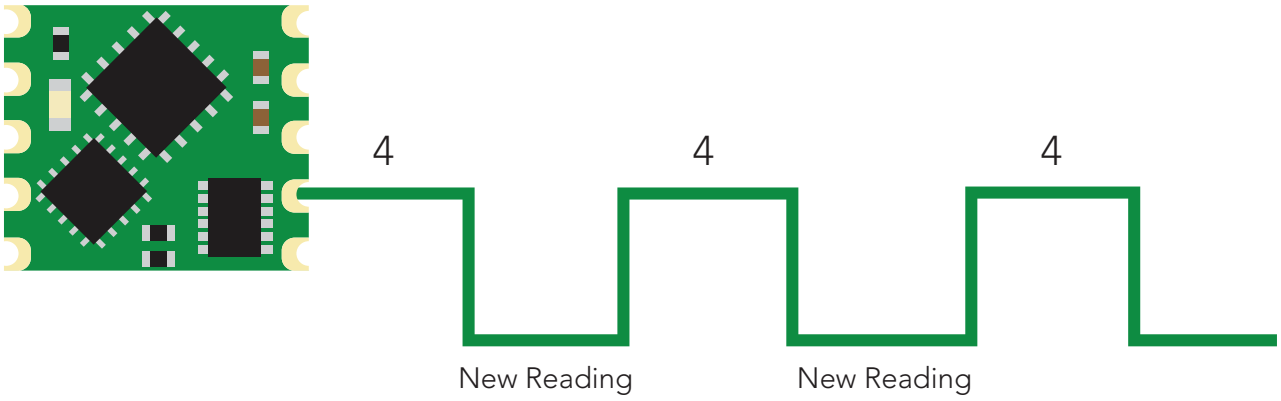
Example code
Setting pin high on new reading

```
byte i2c_device_address=0x64;
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.

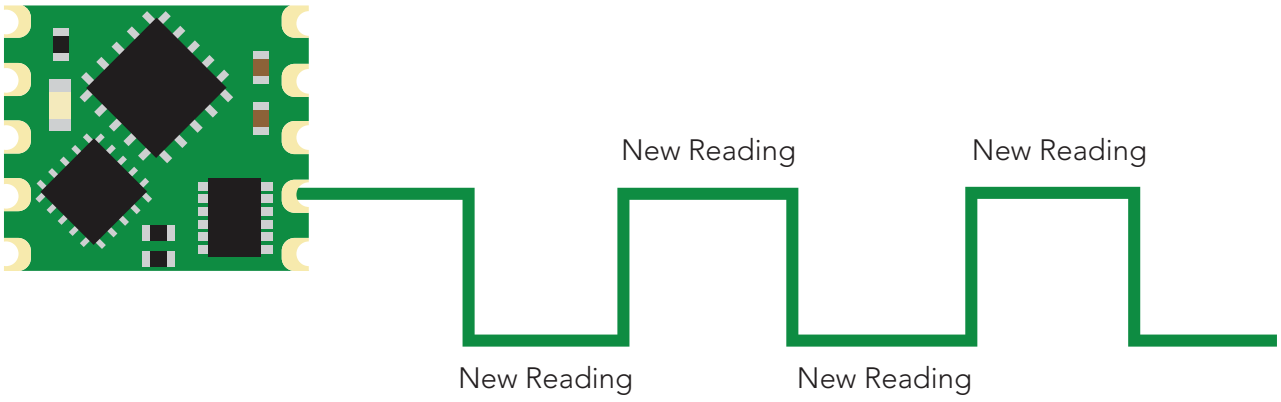
Example code
Setting pin low on new reading

```
byte I2C_device_address=0x64;
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=0x64;
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=0x64;
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.

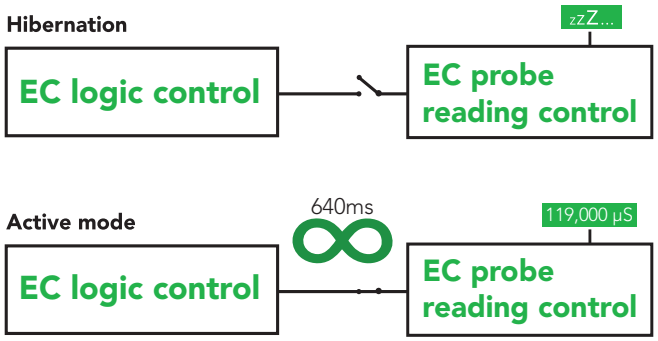
0x06 – Active/hibernate register

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 EC readings

```
byte i2c_device_address=0x64;
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 640ms. **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=0x64;
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_EC();
Wire.beginTransmission(i2c_device_address);
Wire.write(nra);
Wire.write(0x00);
Wire.endTransmission();
}
```


Probe type

0x08: Set probe type MSB	R/W
0x09: Set probe type LSB	R/W

0x08 – 0x09 Set probe type

unsigned word
Default value = K 1.0

The EC OEM™ Embedded Electrical Conductivity Circuit needs to be connected to a conductivity probe. Conductivity probes are differentiated by the probe's cell constant. Any cell constant from K 0.01 – K 600 can be used. **By default the probe type is set to K 1.0**

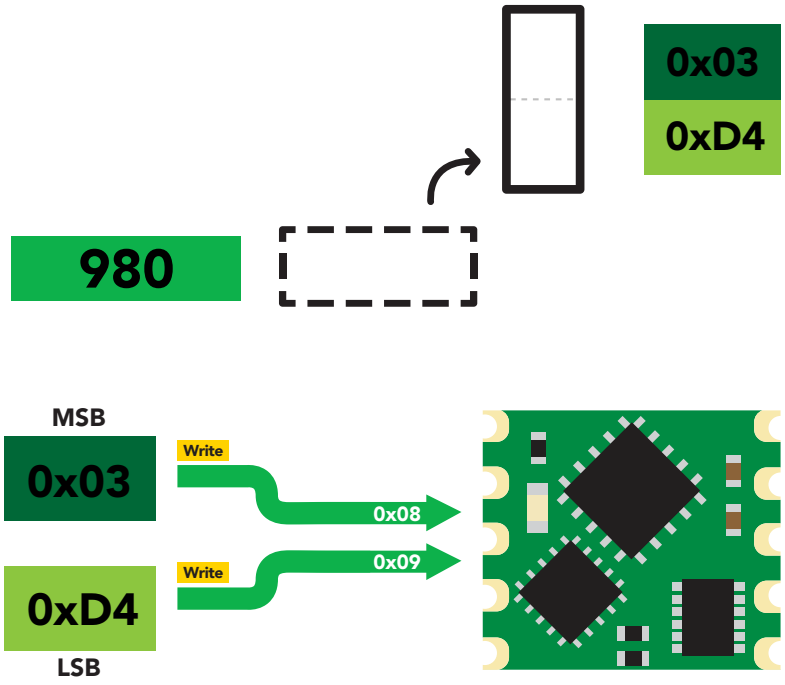
To send a new K constant to the EC OEM™, the value of the K constant must be multiplied by 100 and then transmitted to the EC OEM™. The K constant will be divided by 100 internally.

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

Example

To set the probe type to a K constant of K = 9.8, the master device will transmit the number 980. Obviously the number 980 cannot be stored in a single 8 bit register. Move the value from a float to an unsigned word. Break up the unsigned word into its MSB and LSB. Send the MSB to register 0x08 and the LSB to register 0x09

K = 9.8
9.8 x 100 = 980
980 to HEX = 0x03D4
Probe Type MSB Register = 0x03
Probe Type LSB Register = 0xD4



Calibration

0x0A: Calibration value MSB	R/W
0x0B: Calibration value high byte	R/W
0x0C: Calibration value low byte	R/W
0x0D: Calibration value LSB	R/W

0x0A – 0x0D Calibration registers

Signed long
0x0A = MSB
0x0D = LSB
Units = µS

Calibration values can be whole number, or floating point.
***Calibration values are in microsiemens only.**

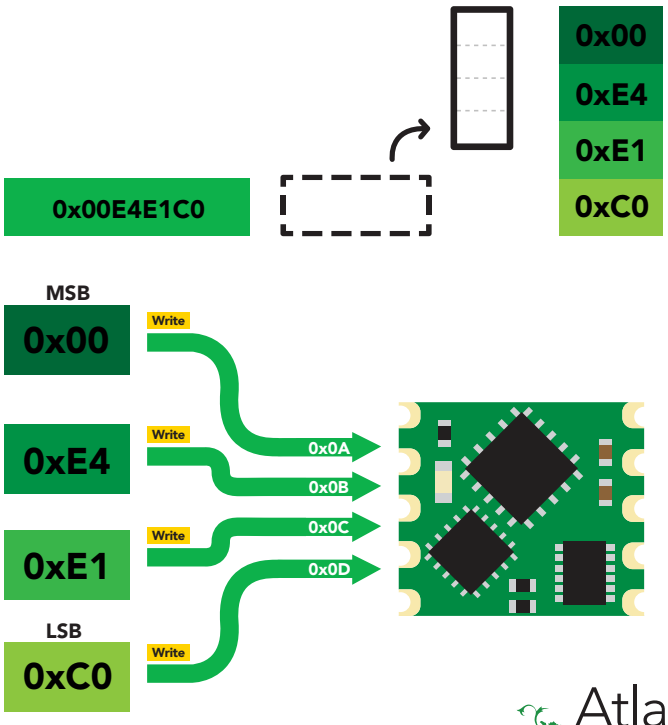
After sending a value to this register block, calibration is **not** complete. The calibration request register must be set after loading a calibration value into this register block.

To send a new calibration value to the EC OEM™ the value of the calibration solution must be multiplied by 100 and then transmitted to the EC OEM™. The calibration value will be divided by 100 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 0x0A, 0x0B, 0x0C and 0x0D.

Example

Calibrating to a 150,000µs solution.
calibration value = 150,000µS
150,000.00 x 100 = 15,000,000
15,000,000 to HEX = 0x00E4E1C0

calibration MSB Register = 0x00
calibration high byte Register = 0xE4
calibration low byte Register = 0xE1
calibration LSB Register = 0xC0



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
0x1A
0x1B
0x1C
0x1D
0x1E
0x1F
0x20
0x21
0x22
0x23

0x0E – Calibration request register

1 unsigned byte

Command values
1 Clear calibration = (delete all calibration data)
2 Dry calibration
3 Single point calibration
4 Dual point calibration low
5 Dual point calibration high

Once a calibration value has been transmitted to the previous registers (0x0A – 0x0D) 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 EC 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 five possible values, calibration will commence once an I^2C stop bit has been transmitted.

0x0F – Calibration confirmation register

1 unsigned byte

Command values
0 = dry calibration
1 = single point calibration
2 = low point calibration
3 = 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 0–3.

Bit 3 (High)	Bit 2 (Low)	Bit 1 (Single)	Bit 0 (Dry)	Decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

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

Temperature compensation

0x10: Temperature compensation MSB	R
0x11: Temperature compensation high byte	R
0x12: Temperature compensation low byte	R
0x13: Temperature compensation LSB	R

0x10 – 0x13 Temperature compensation registers

Unsigned long
0x10 = MSB
0x13 = LSB
Default value = 25 °C
Units = °C

The EC OEM™ Embedded Electrical Conductivity Circuit can take temperature compensated conductivity 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 EC OEM™ the value of the temperature must be multiplied by 100 and then transmitted to the EC 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)

0x10	0x11	0x12	0x13

Temperature confirmation

0x14: Temperature confirm MSB	R
0x15: Temperature confirm high byte	R
0x16: Temperature confirm low byte	R
0x17: Temperature confirm LSB	R

0x14 – 0x17 Temperature confirmation registers

Unsigned long
0x14 = MSB
0x17 = 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 conductivity readings. This register can be used to be sure that the conductivity readings that are being taken are at the correct temperature.

If the temperature compensation register has changed from 25˚C to 30˚C, reading this register will show what temperature the conductivity 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. Divide that number by 100.

Sensor data

0x18: EC reading MSB	R
0x19: EC reading high byte	R
0x1A: EC reading low byte	R
0x1B: EC reading LSB	R

0x18 – 0x1B EC reading registers

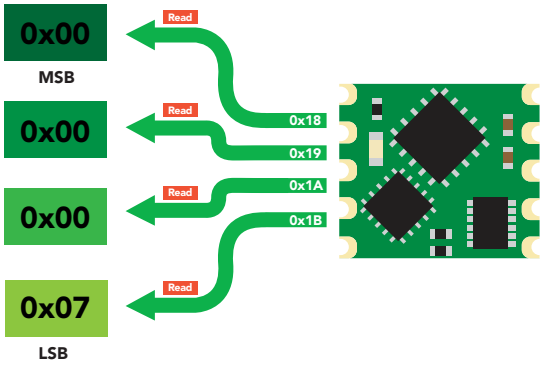
Signed long
0x18 = MSB
0x1B = LSB
Units = µS

The last EC reading taken is stored in these four registers. **EC is always expressed in microsiemens.** 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.

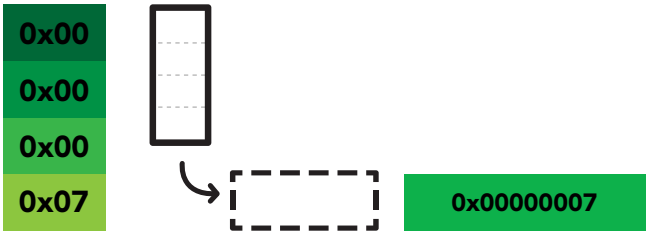
Example

Reading an EC of 0.07µs

Step 1 read 4 bytes

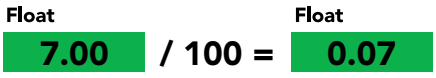


Step 2 read unsigned long



Step 3 cast unsigned long to a float

Step 4 divide by 100



0x1C: TDS reading MSB	R
0x1D: TDS reading high byte	R
0x1E: TDS reading low byte	R
0x1F: TDS reading LSB	R

0x1C – 0x1F TDS reading registers

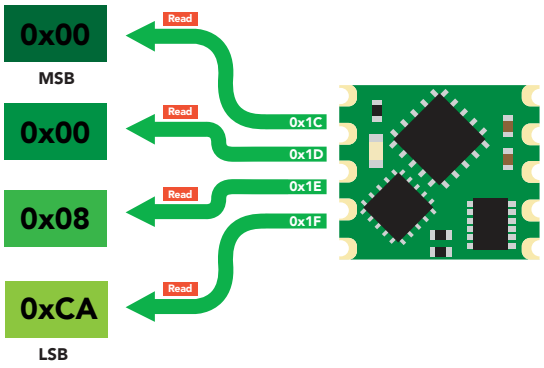
Signed long
0x1C = MSB
0x1F = LSB
Units = TDS

The last TDS 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. Divide that number by 100.

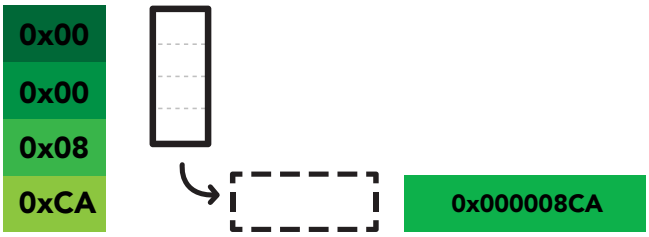
Example

Reading a TDS of 22.5

Step 1 read 4 bytes



Step 2 read unsigned long



Step 3 cast unsigned long to a float

Step 4 divide by 100

0x20: Salinity reading MSB	R
0x21: Salinity reading high byte	R
0x22: Salinity reading low byte	R
0x23: Salinity reading LSB	R

0x20 – 0x23 Salinity reading registers

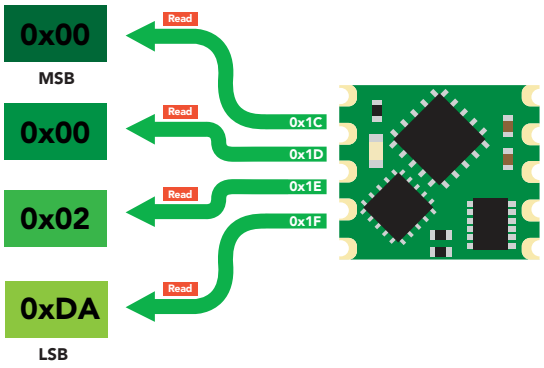
Signed long
0x20 = MSB
0x23 = LSB
Units = Salinity

The last Salinity 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. Divide that number by 100.

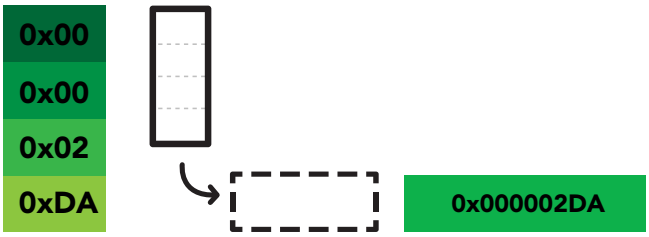
Example

Reading a Salinity of 7.3

Step 1 read 4 bytes

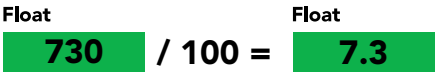


Step 2 read unsigned long



Step 3 cast unsigned long to a float

Step 4 divide by 100



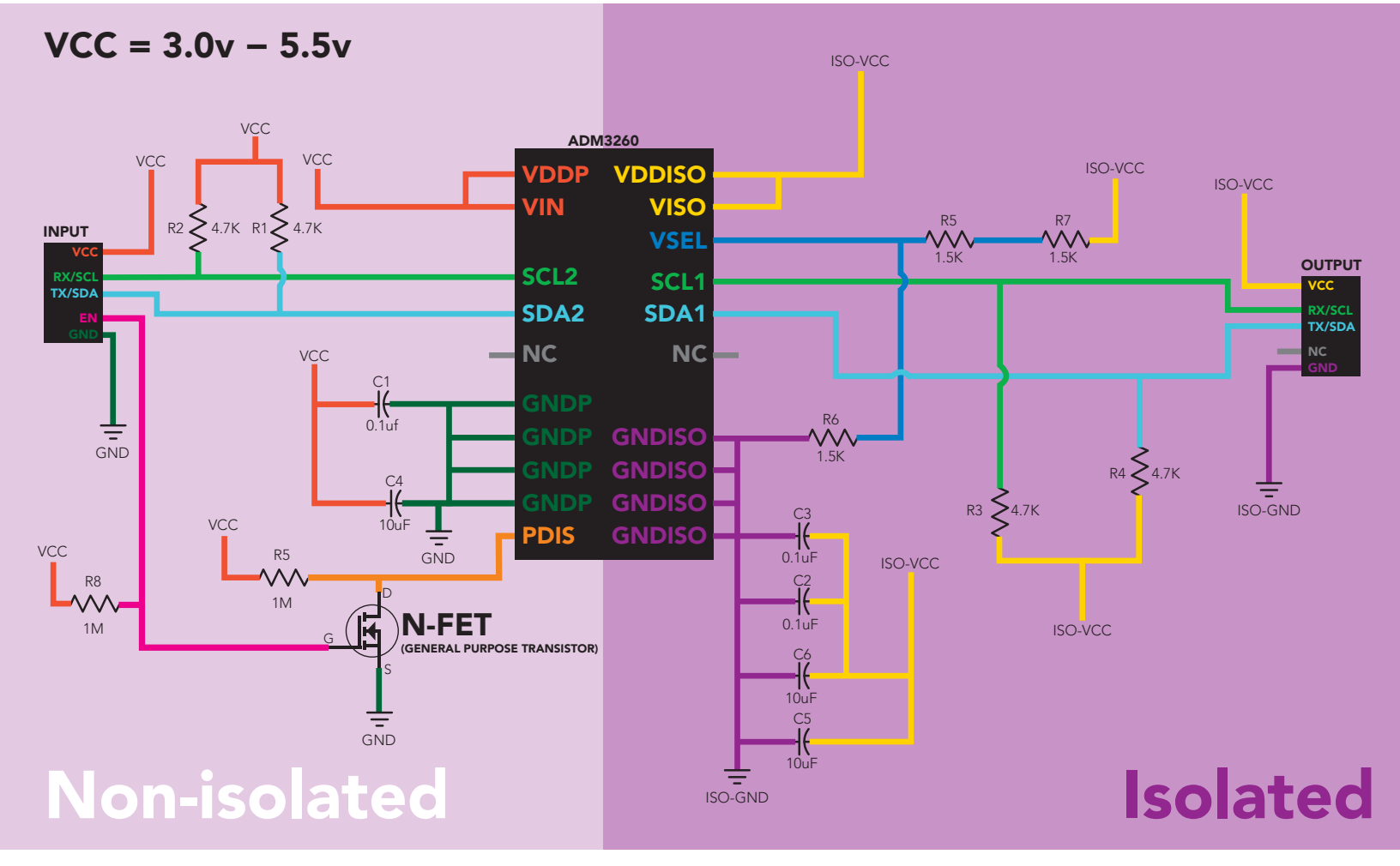
OEM electrical isolation

If the EC OEM™ Class Embedded Conductivity 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 EC probe does not interfere with other sensors and that outside electrical noise does not affect the device.

The goal of electrically isolating the EC OEM™ circuit 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 (R5, R6, and R,7) 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.

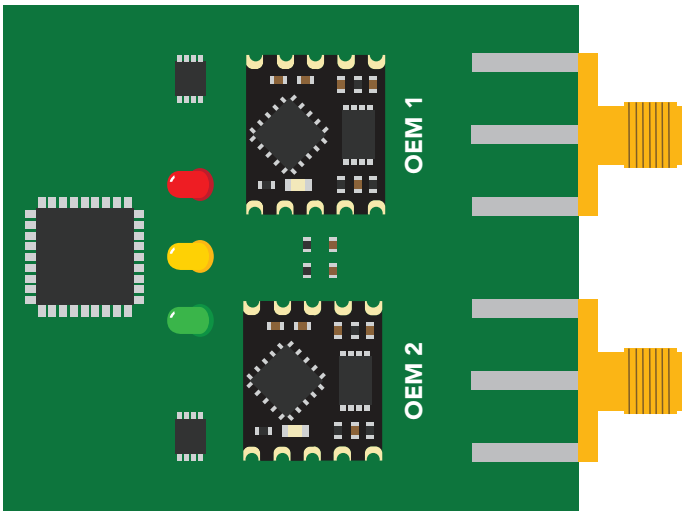


Designing your product

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

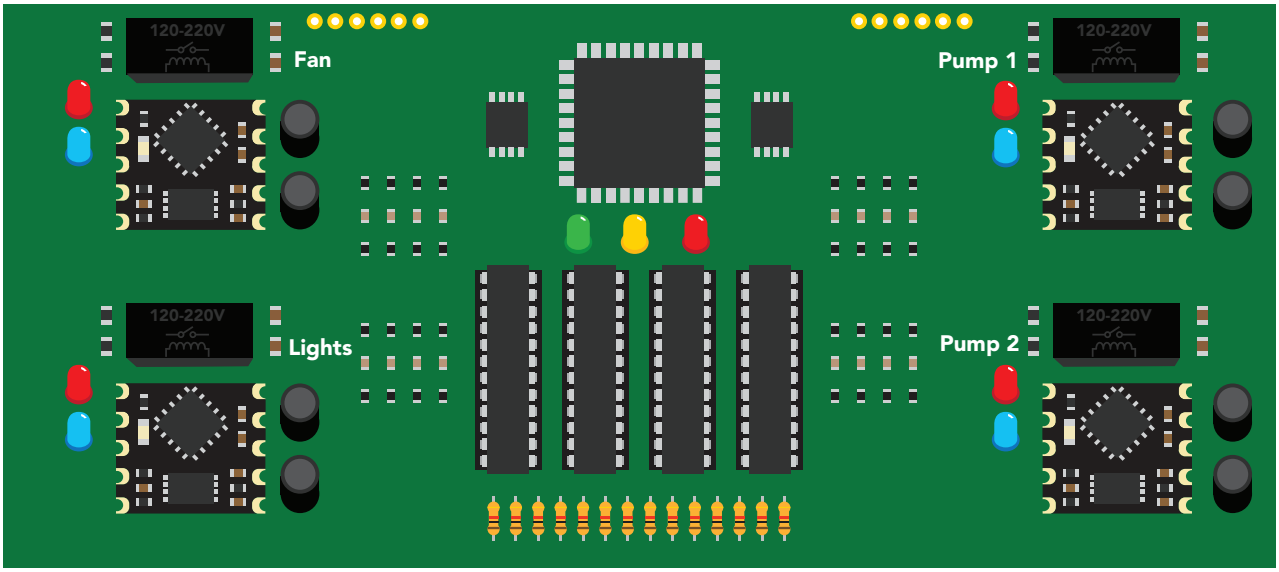
Simple design

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



Complex design

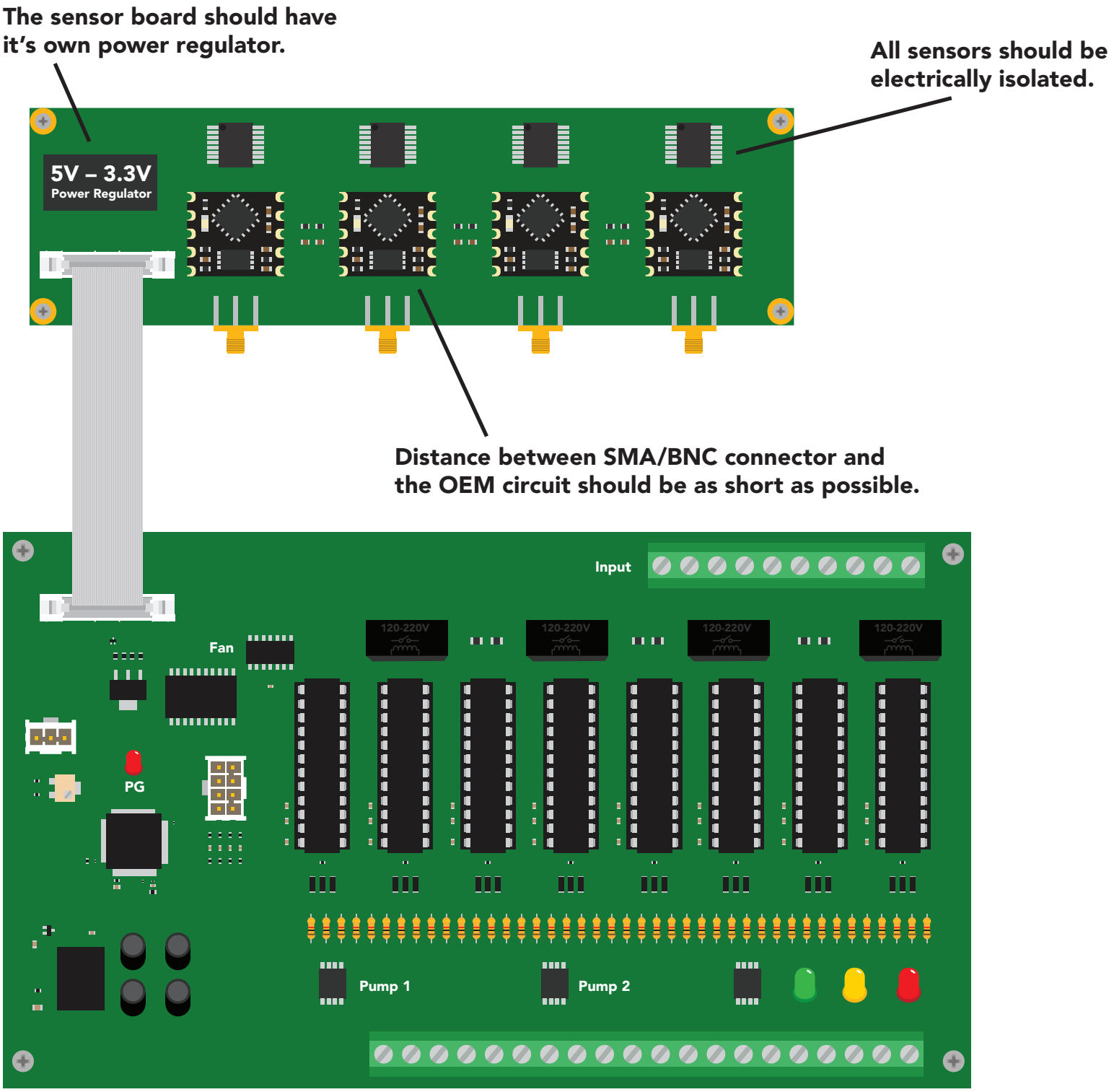
Complex computer systems with multiple voltages and switching, can lead to extended and unnecessary debugging time. Target customers can experience frequent accuracy issues.



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.

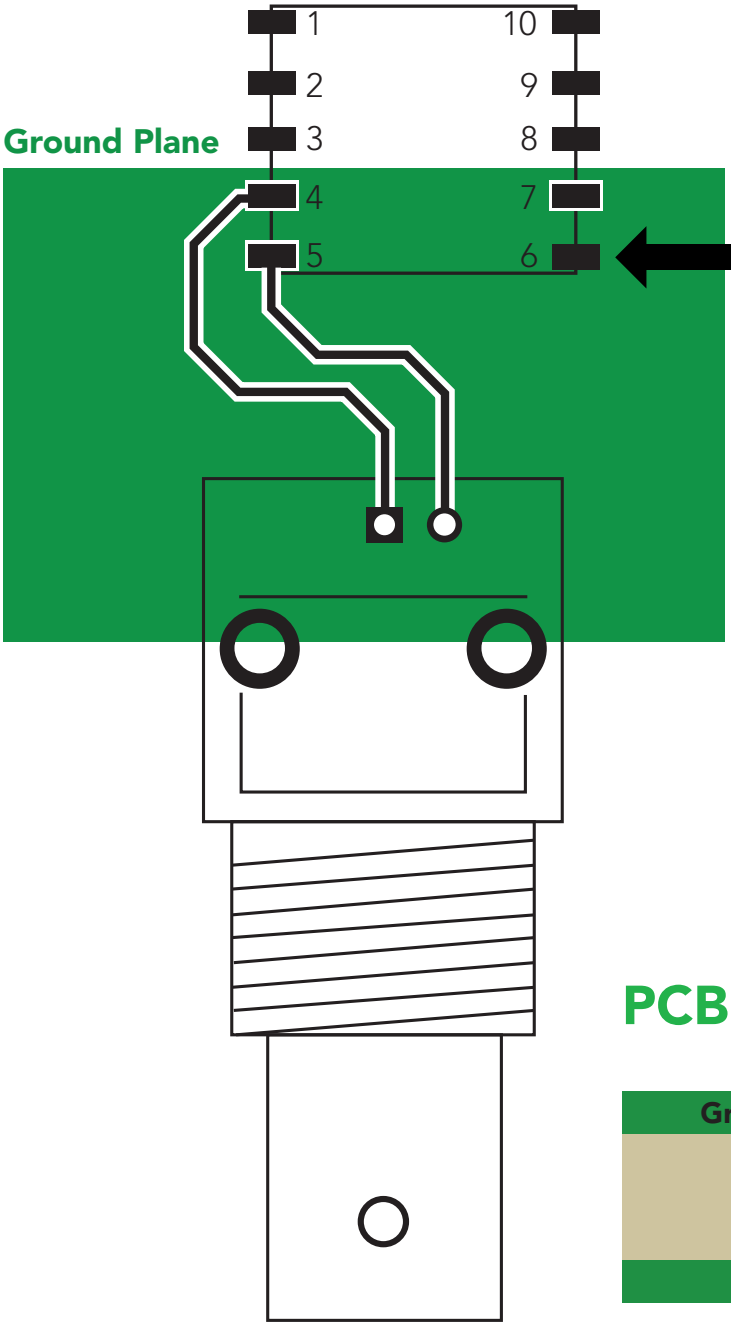
Target customers will experience accurate, stable and repeatable readings for the life of your product.



AtlasScientific™ Environmental Robotics

Designing your PCB

Create the traces as short as possible from the EC OEM™ circuit to your probe connection. Keep the traces on your top layer, keep a distance of 1mm for any other trace. use 0.4mm trace width. Use a ground plane underneath the traces and probe connection.



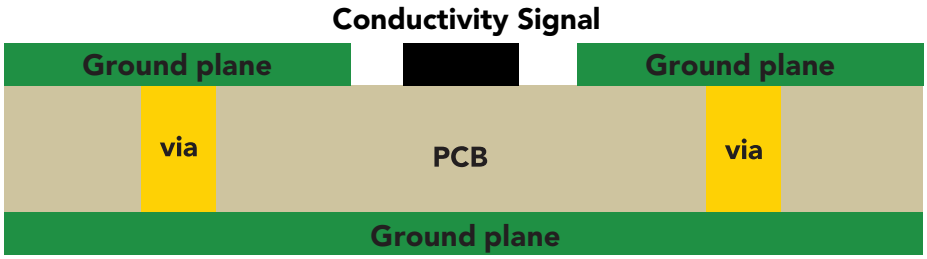
Connect pin 6 to the ground plane.

Make sure there are no vias or exposed metal underneath the EC OEM™ circuit.

If pin 7(INT) is unused leave it floating, *do not connect pin 7 to VCC or ground.*

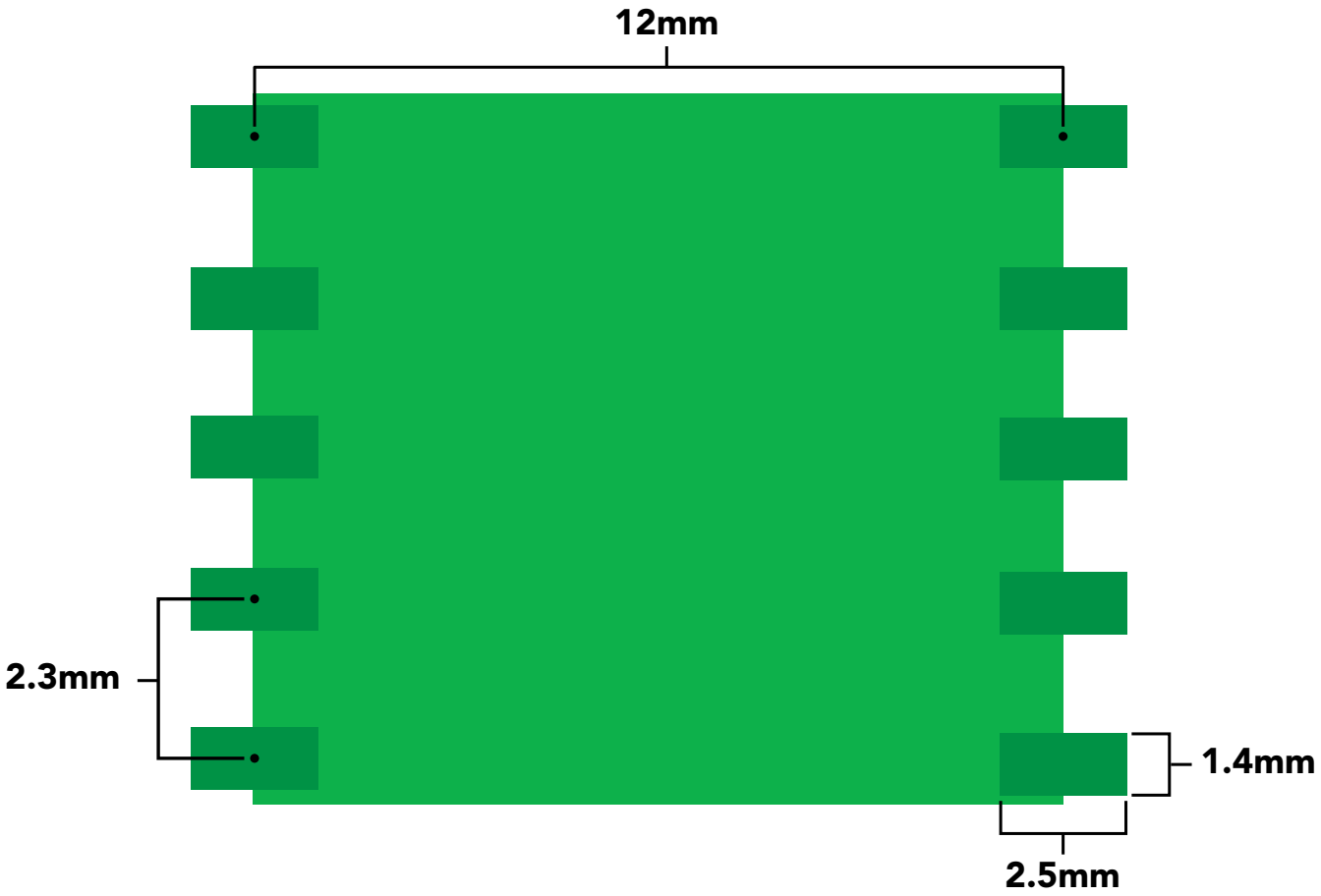
Keep the traces for both probe and probe ground as short as possible.

PCB cross section of the signal path



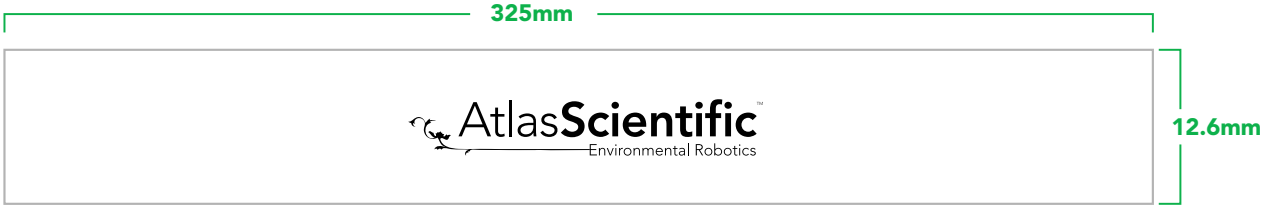
This cross section is an example of how the ground plane protects the Conductivity signal. The ground plane should surround the Conductivity signal, on the top layer as well as the bottom layer.

Recommended pad layout

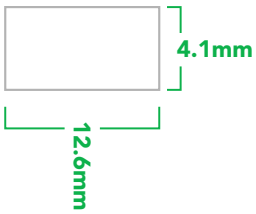


IC tube measurements

Top View



Side View

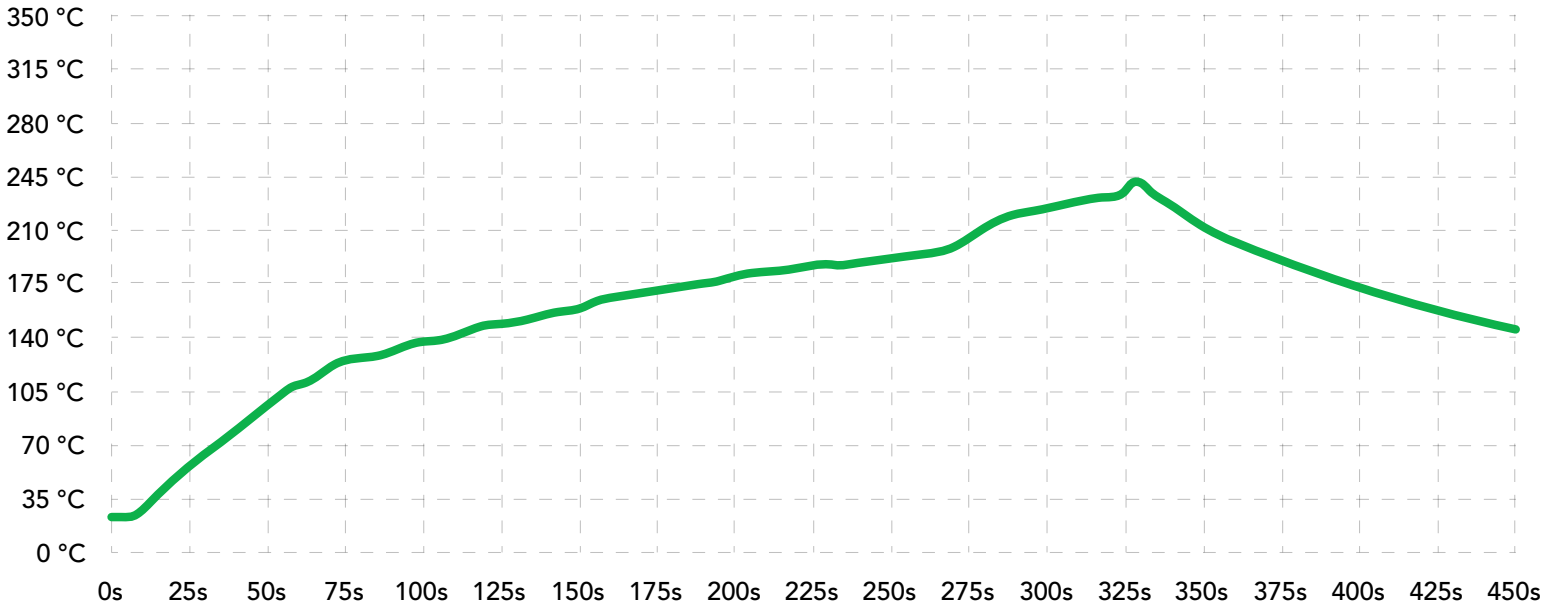


Fits 25 EC OEM™ circuits

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

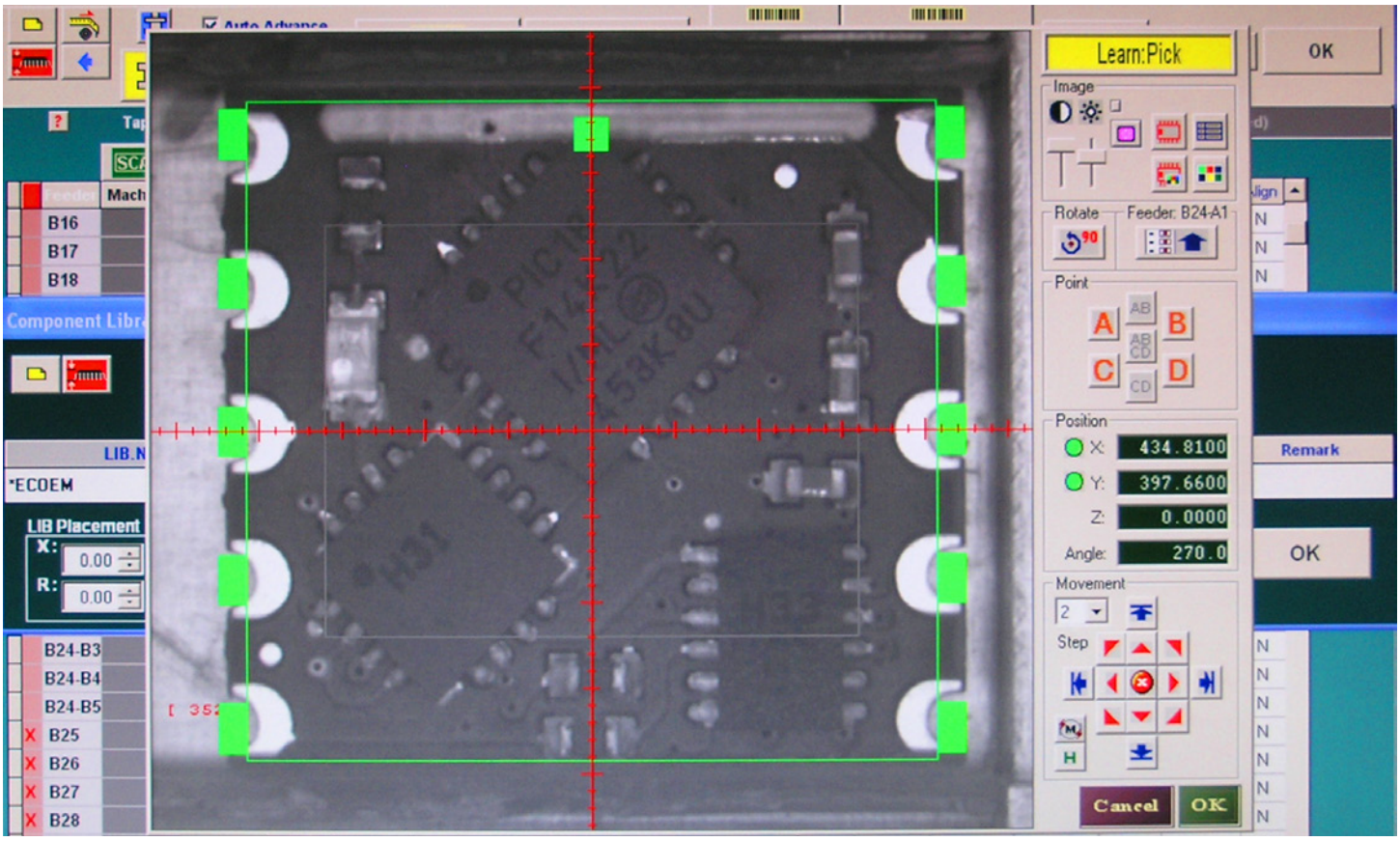
plastic thickness 0.5mm

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



AtlasScientific™ Environmental Robotics

Datasheet change log

Datasheet V 2.9

Revised artwork on pg 8.

Datasheet V 2.8

Added more info for "Power consumption" on pg 5.

Datasheet V 2.7

Added "Designing you product" on pg 29.

Datasheet V 2.6

Revised information about salinity throughout datasheet.

Datasheet V 2.5

Revised information about *desiging your own EC board* on pg. 29

Datasheet V 2.4

Revised isolation schematic on pg. 28

Datasheet V 2.3

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

Datasheet V 2.2

Corrected typos on pg 21 & 24.

Datasheet V 2.1

Revised range of suppoted probe types.

Datasheet V 2.0

Revised entire datasheet.

Firmware updates

V1.0 – Initial release (Oct 10, 2015)

V2.0 – (June 2, 2015)
- Improved default calibration values.

V3.0 – (Aug 28, 2015)
- Fixed glitch in cal clear command.

V4.0 – (June 3, 2016)
- Simplified LED functionality.

V5.0 – (July 6, 2017)
- Fixed glitch in confirming single point calibration.

