AtlasScientific™
Environmental Robotics

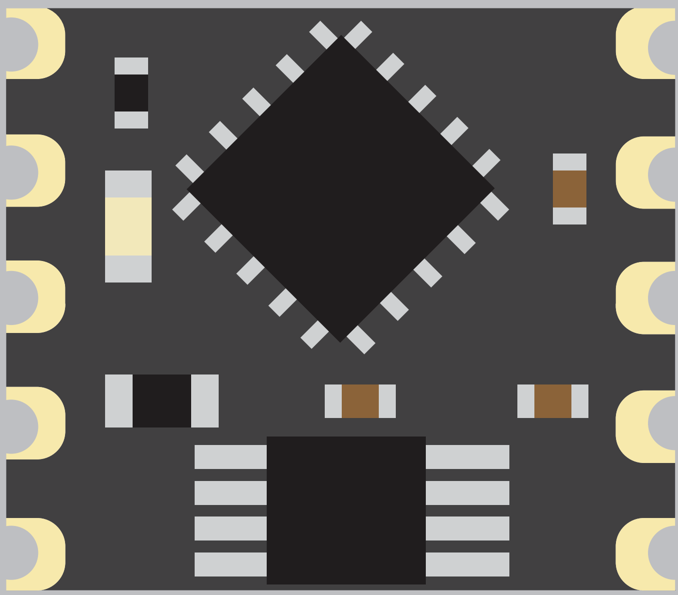
V 1.2

Revised 3/22

OEM-RTD™

2 Wire – Embedded Temperature Circuit

Reads	**Temperature in °C**
Range	**-126.000 °C – 1254 °C**
Resolution	**0.001**
Accuracy	**+/– (0.1 + 0.0017 x ˚C)**
Response time	**1 reading every 420ms**
Supported probes	**Any PT-100 or PT-1000 RTD**
Supported configuration	**2 wire RTD**
Calibration	**Single point**
Data protocol	**SMBus/I²C**
Default I²C address	**0x68**
Operating voltage	**3.0V – 5.5V**
Data format	**ASCII**



Written by Jordan Press
Designed by Noah Press

This is an evolving document, check back for updates.

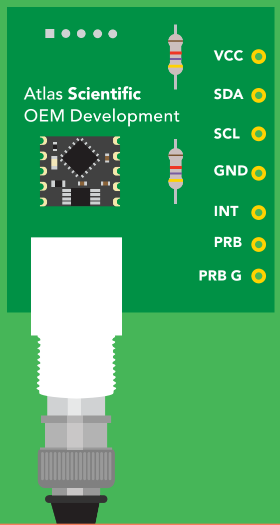
STOP

SOLDERING THIS DEVICE VOIDS YOUR WARRANTY.

Before purchasing the RTD 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.

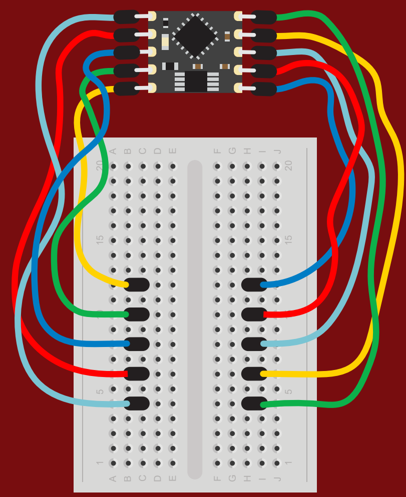
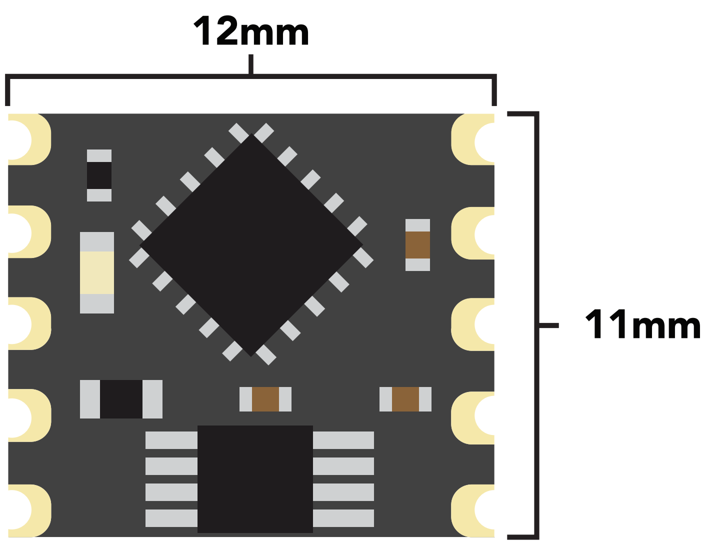
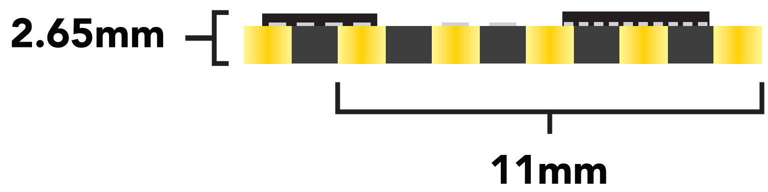


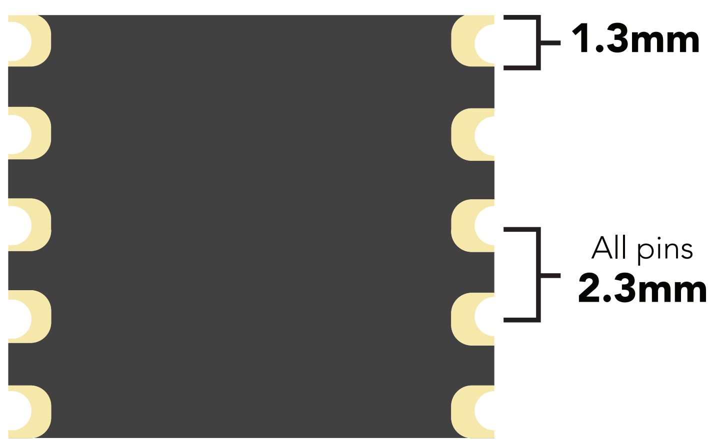
Table of contents

REGISTERS

OEM circuit dimensions







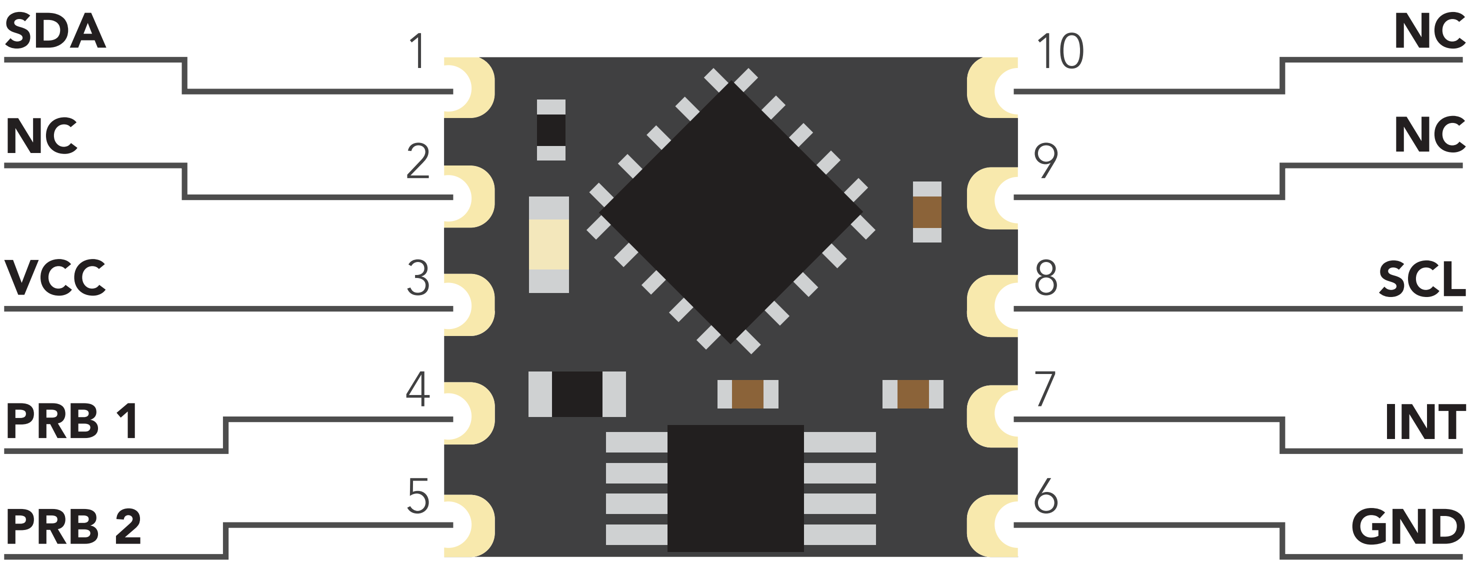
Power consumption

	LED	OPERATIONAL	HIBERNATION
3.3V	ON	4.2 mA	3.7 mA
	OFF	3.7 mA	2.3 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™ RTD OEM™ will always produce a reading with a resolution of three decimal places.

Example

100.123 ˚C

-76.000 ˚C

Power on/start up

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

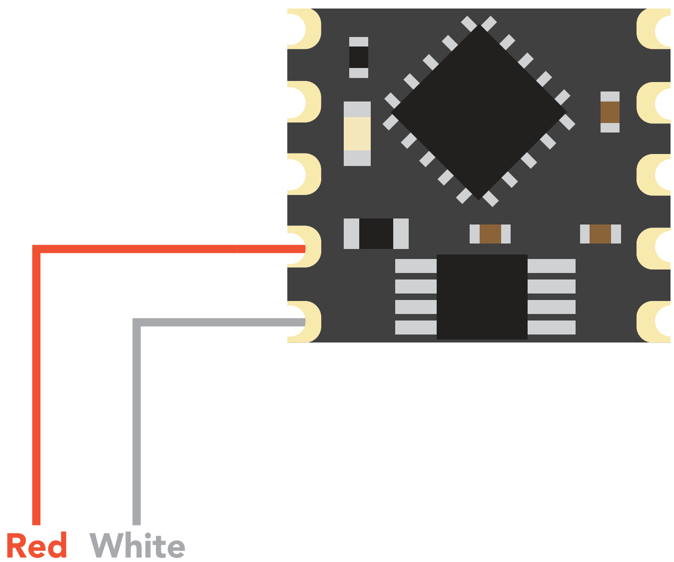
Settings that are retained if power is cut	Settings that are *NOT* retained if power is cut
Calibration	Active/Hibernation mode
I²C address	LED control
	Interrupt control

RTD connection

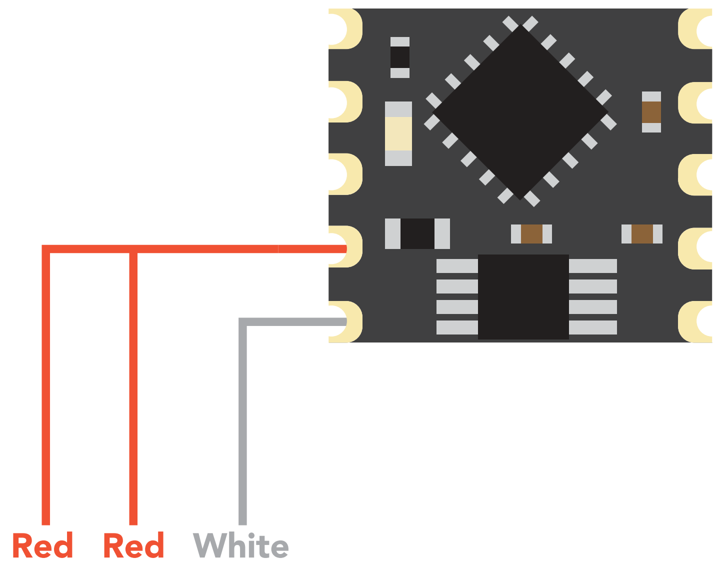
The Atlas Scientific™ RTD OEM™ will automatically detect if the attached probe is a ***PT-100*** or ***PT-1000 probe***.

Keep in mind that PT-100 / PT-1000 probes have no polarity.
It's not possible to connect the leads to the probe in reverse.

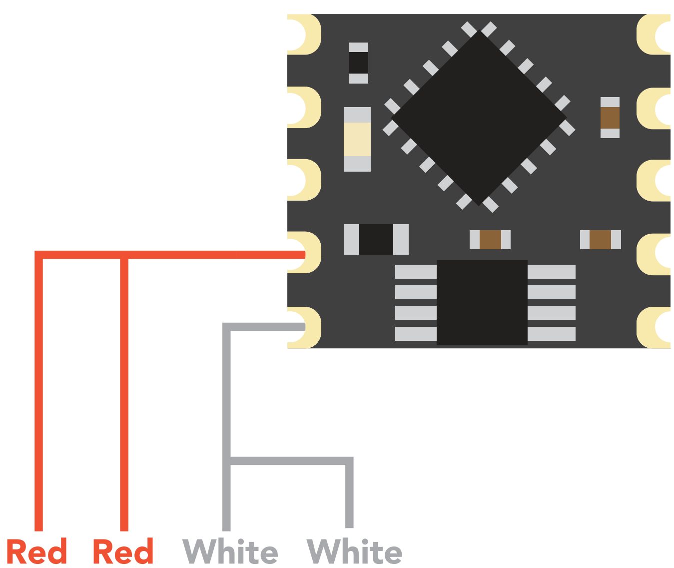
2 wire



3 wire



4 wire



AtlasScientific™
Environmental Robotics

System overview

The Atlas Scientific™ RTD OEM™ Class Embedded Circuit is the core electronics needed to read temperature from any brand of PT-100 or PT-1000 RTD temperature probe. The RTD 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 **18** different 8 bit registers.

Accessible registers

Device information

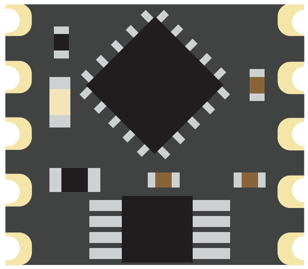
Register	Access
0x00: Device type	R
0x01: Firmware version	R

Device address

Register	Access
0x02: SMBus/I²C address lock/unlock	R/W
0x03: SMBus/I²C address	R/W

Control

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



Read only R
Read and write R/W

Calibration

Register	Access
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

Register	Access
0x0C: Calibrate request	R/W
0x0D: Calibration confirm	R

Sensor Data

Register	Access
0x0E: Temperature reading MSB	R
0x0F: Temperature reading high byte	R
0x10: Temperature reading low byte	R
0x11: Temperature reading LSB	R

The default device address is **0x68**
This address can be changed.

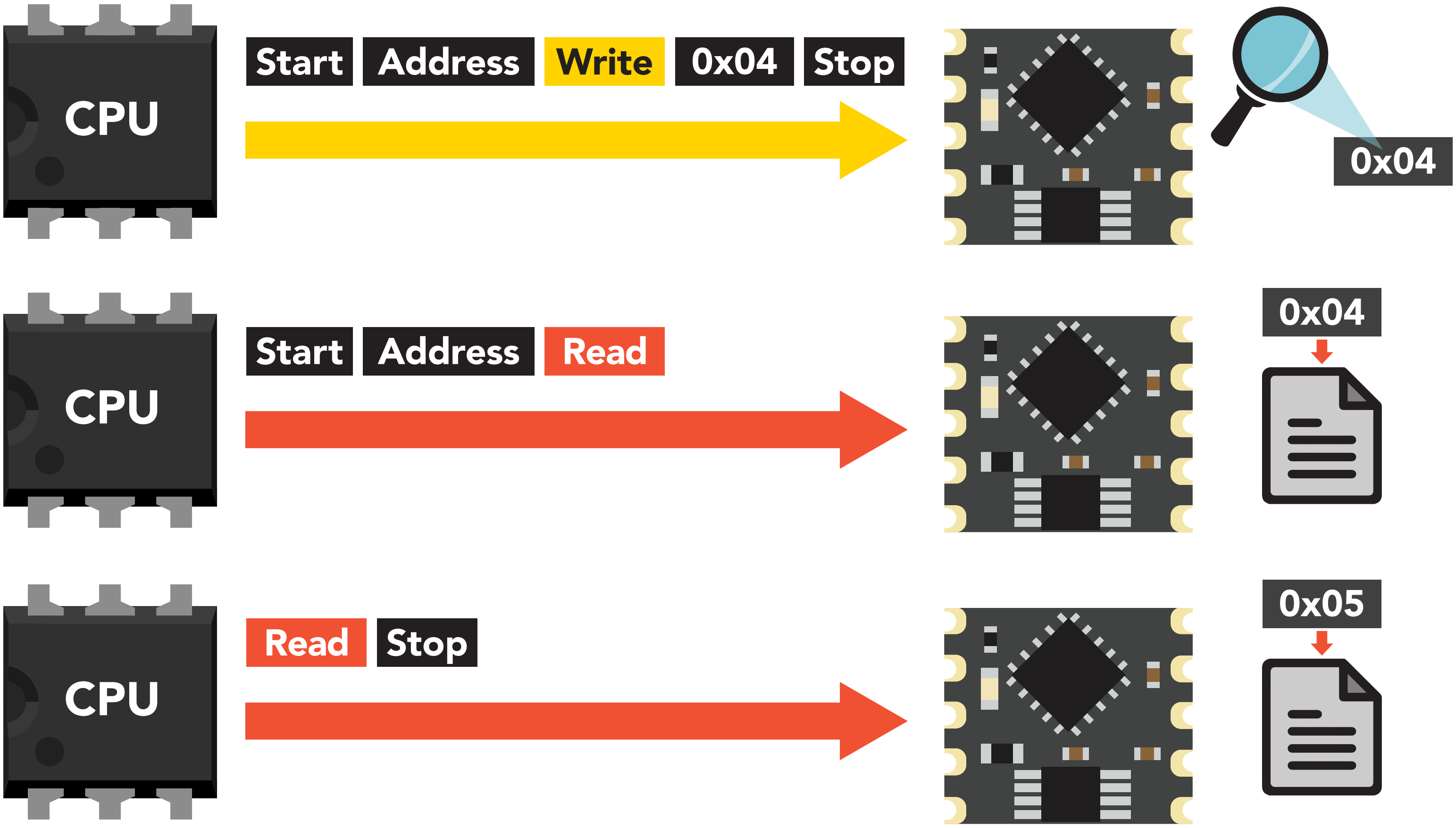
Each RTD reading takes 420ms

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 **0x68**
This address can be changed.

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



Example code
reading two registers

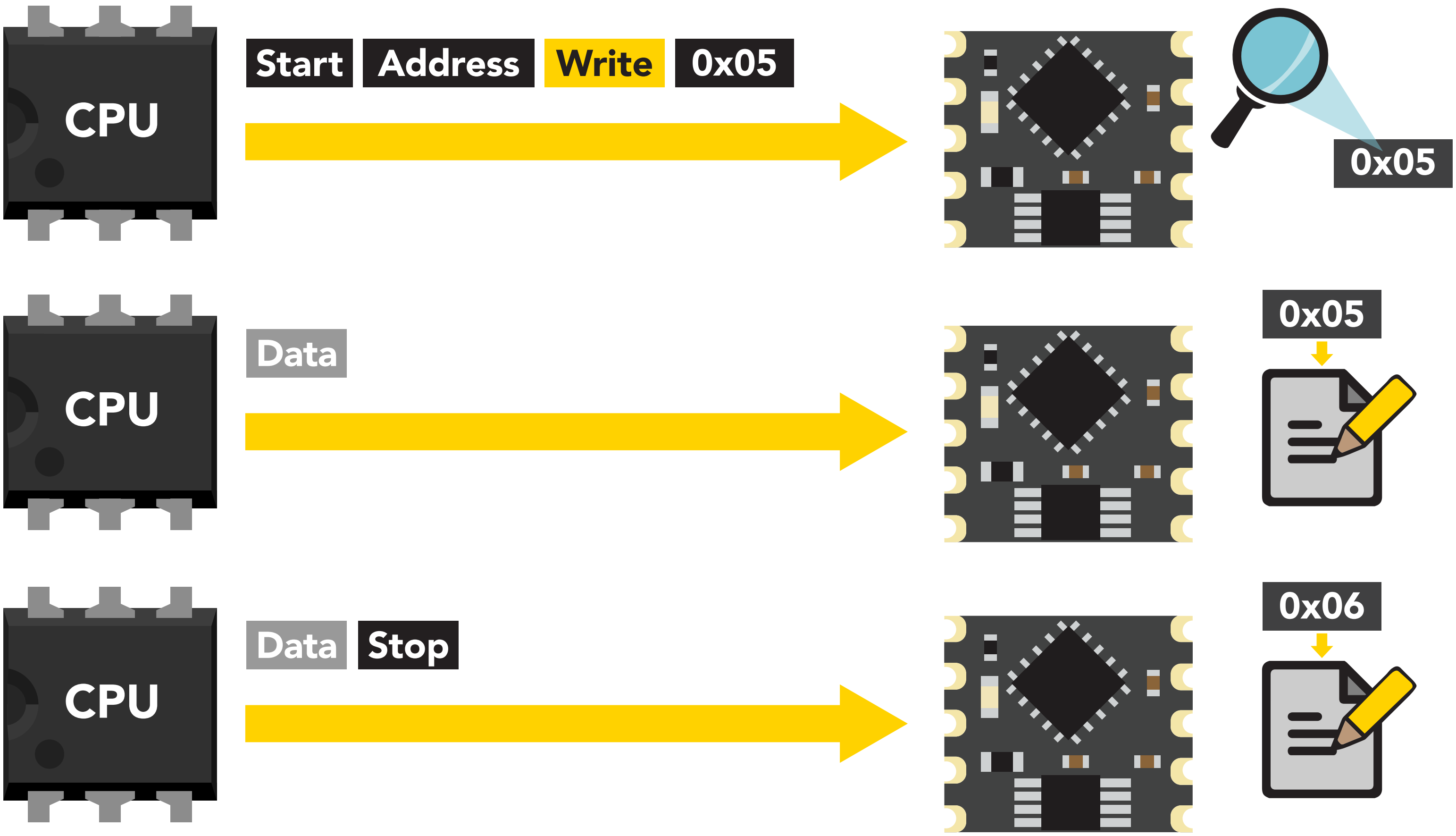
```
byte i2c_device_address=0x68;
byte reg_4, reg_5;

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=0x68;
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.

When transmitting a floating point number to the calibration value registers, the number must first be multiplied by 1,000. This would have the effect of removing the floating point. Internally the RTD OEM™ will divide the number by 1,000; converting it back into a floating point number.

Example
Setting an RTD calibration value of: 100.123 ˚C
100.123 X 1,000 = 100,123
Transmit the number 100,123 to the Calibration value registers.

Setting an RTD calibration value of: -76 ˚C
-76 X 1,000 = -76,000
Transmit the number -76,000 to the Calibration value registers.

When reading back a value stored in the calibration value registers, the value must be divided by 1,000 to return it to its originally intended value.

Receiving floating point numbers

After receiving a value from the temperature reading registers, the number must be divided by 1,000 to convert it back into a floating point number.

Example
Reading a temperature value of 34.786
Value received = 34,786
34,786 / **1,000** = 34.786

Reading an temperature value of -98.335
Value received = -98,335
-98,335 / **1,000** = -98.335

Registers

Device information

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

0x00 – Device type register

1 unsigned byte
Read only value = 5
5 = RTD

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=0x68;
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();
```


AtlasScientific Environmental Robotics

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.

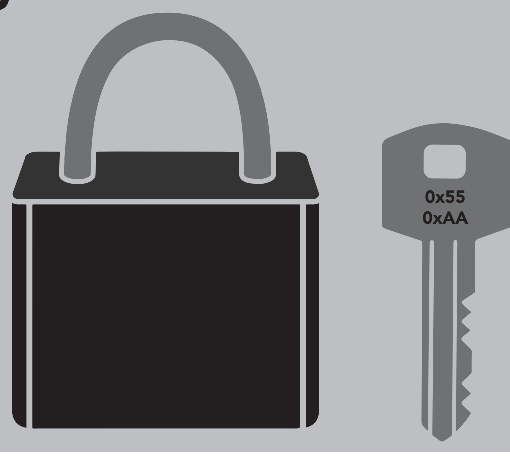
Step 1
Issue unlock command

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=0x68;
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();
```

0x00
0x01
0x02
0x03
0x04
0x05
0x06
0x07
0x08
0x09
0x0A
0x0B
0x0C
0x0D
0x0E
0x0F
0x10
0x11

Step 2
Change address

0x03 – I²C address register

1 unsigned byte
Default value = **0x68**
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=0x68;
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

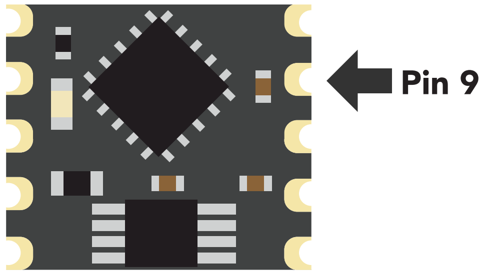
0x04: Interrupt control	R/W
0x05: LED control	R/W
0x06: Active/hibernate	R/W
0x07: New reading available	R/W

0x00
0x01
0x02
0x03
0x04
0x05
0x06
0x07
0x08
0x09
0x0A
0x0B
0x0C
0x0D
0x0E
0x0F
0x10
0x11

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)



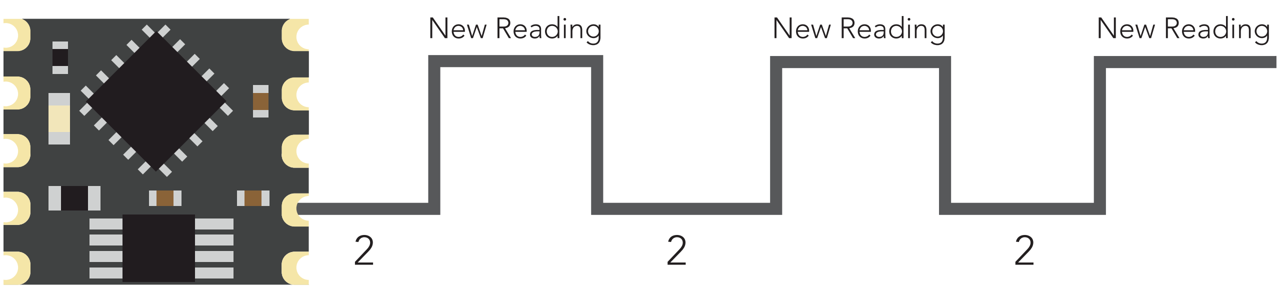
The Interrupt control register adjusts the function of pin 9 (the interrupt output pin).

⚠ 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 9) 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.

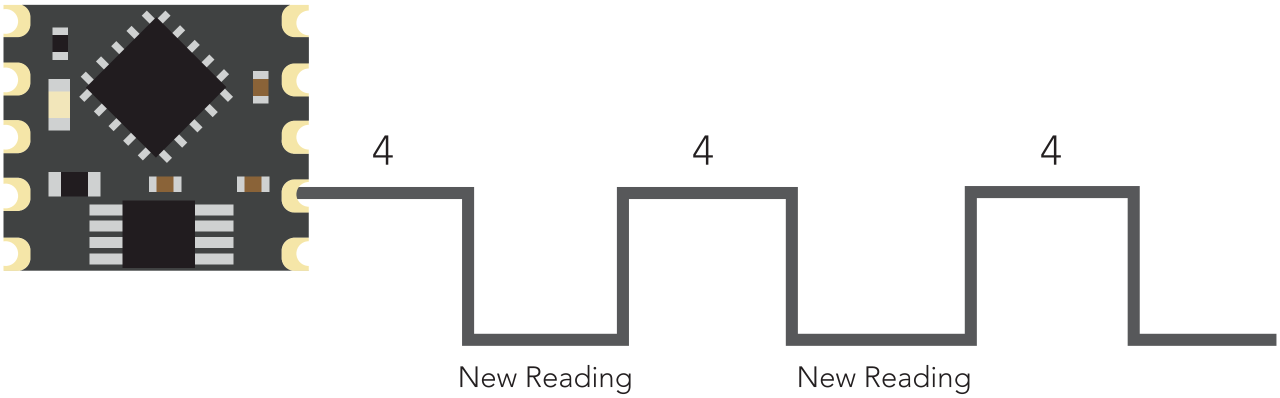
Example code
Setting pin high on new reading

```
byte i2c_device_address=0x68;
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 9) 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.

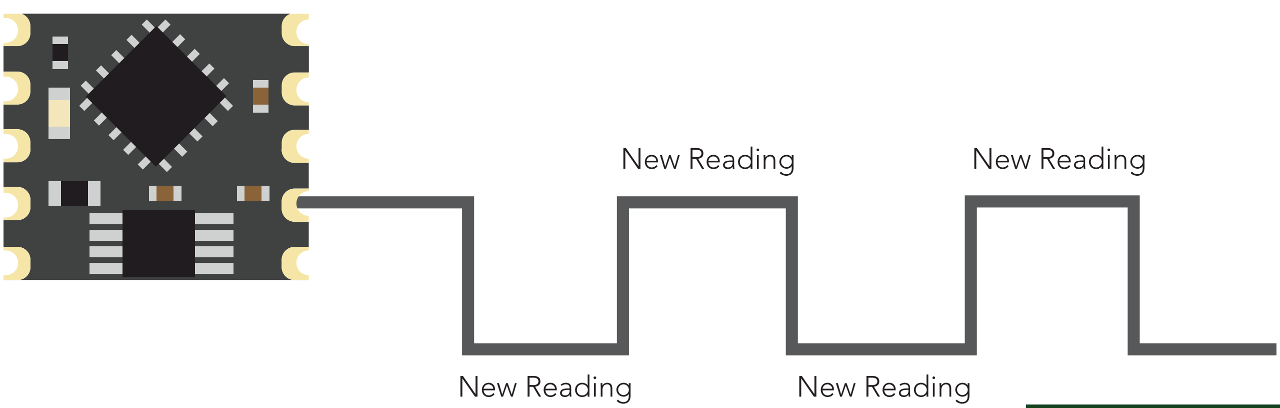
Example code
Setting pin low on new reading

```
byte I2C_device_address=0x68;
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 9) 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=0x68;
byte int_control=0x04;

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

0x00
0x01
0x02
0x03
0x04
0x05
0x06
0x07
0x08
0x09
0x0A
0x0B
0x0C
0x0D
0x0E
0x0F
0x10
0x11

0x00
0x01
0x02
0x03
0x04
0x05
0x06
0x07
0x08
0x09
0x0A
0x0B
0x0C
0x0D
0x0E
0x0F
0x10
0x11

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=0x68;
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.

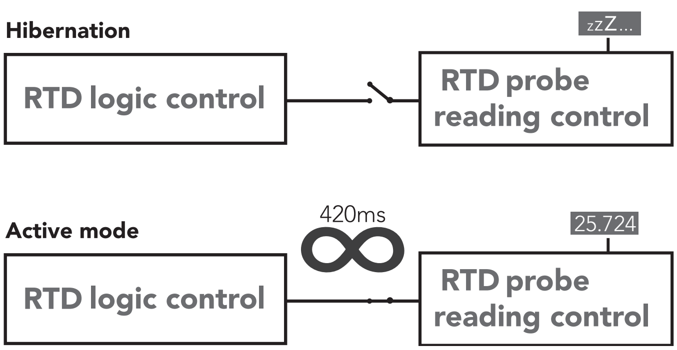
0x06 – Active/hibernate register

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 RTD readings

```
byte i2c_device_address=0x68;
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=0x68;
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_RTD();
Wire.beginTransmission(i2c_device_address);
Wire.write(nra);
Wire.write(0x00);
Wire.endTransmission();
}
```


AtlasScientific
Environmental Robotics

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 = °C

A calibration point can be a single whole number, or single floating point number up to three decimal place.

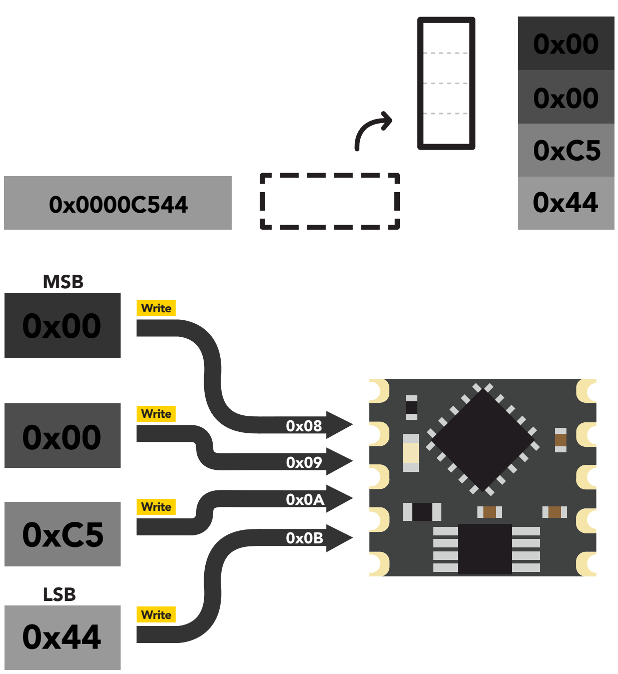
Example
100
-21.4
49.613

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.

When sending a calibration temperature to the RTD OEM™ the value of the calibration temperature must be multiplied by 1,000 and then transmitted to the RTD OEM™.

Example
Calibrating to a temperature of 50.5
calibration value = 50.5
50.5 x 1,000 = 50,500
50500 to HEX = 0x0000C544

calibration MSB Register = 0x00
calibration high byte Register = 0x00
calibration low byte Register = 0xC5
calibration LSB Register = 0x44



0x00
0x01
0x02
0x03
0x04
0x05
0x06
0x07
0x08
0x09
0x0A
0x0B
0x0C
0x0D
0x0E
0x0F
0x10
0x11

0x0C – Calibration request register

1 unsigned byte

Command values

1 = Clear calibration (delete all calibration data)
2 = Single point calibration

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

0x0D – Calibration confirmation register

1 unsigned byte

Command values

0 = no calibration
1 = calibration

After a calibration event has been successfully carried out, the calibration confirmation register will reflect what that calibration has been done.

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

Sensor data

0x0E: Temperature reading MSB	R
0x0F: Temperature reading high byte	R
0x10: Temperature reading low byte	R
0x11: Temperature reading LSB	R

0x0E – 0x11 RTD reading registers

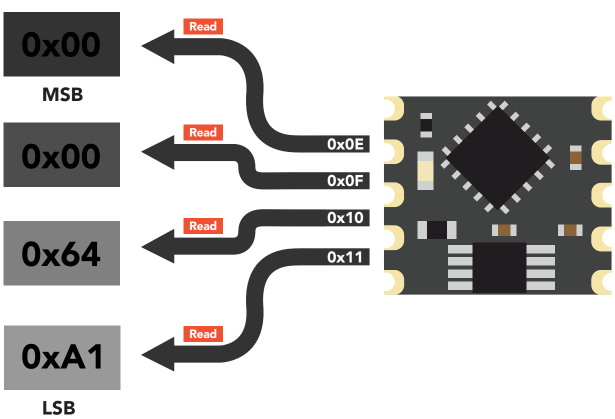
Signed long
0x0E = MSB
0x11 = LSB
Units = °C

The last temperature 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 a signed long, cast to a float. Divide that number by 1,000.

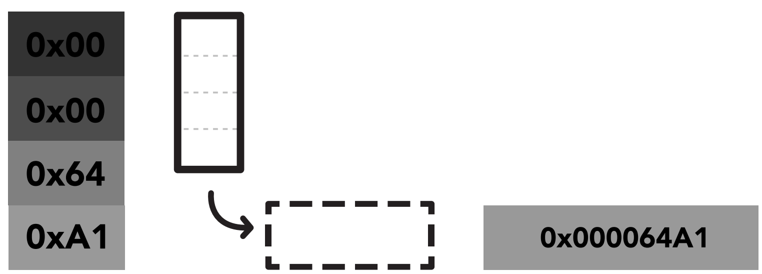
Example

Reading an temperature of 25.761 ˚C

Step 1 read 4 bytes



Step 2 read signed long



Step 3 cast signed long to a float

Unsigned Long 0x000064A1 → **Float** 25,761

Step 4 divide by 1,000

Float 25,761 / 1,000 = **Float** 25.761

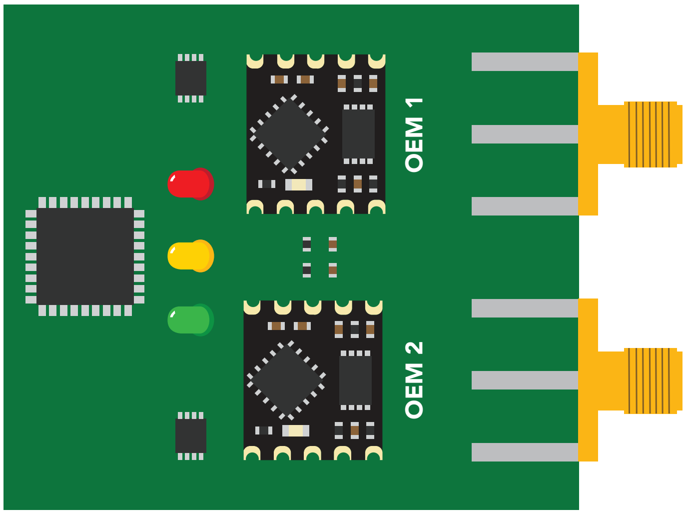
AtlasScientific™ Environmental Robotics

Designing your product

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

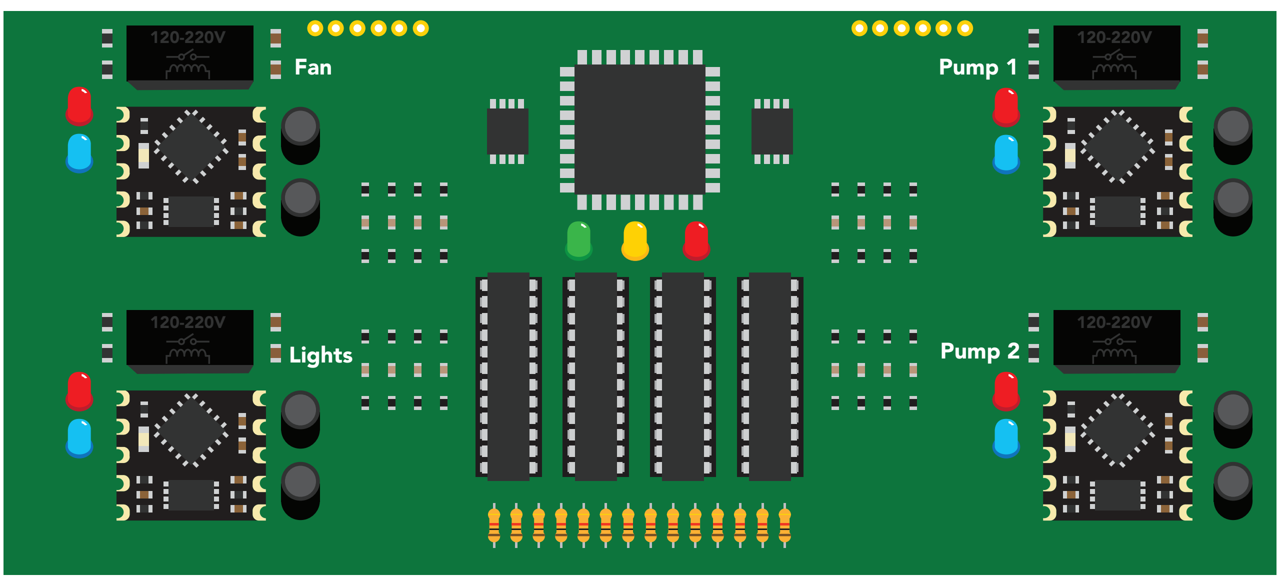
Simple design

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.

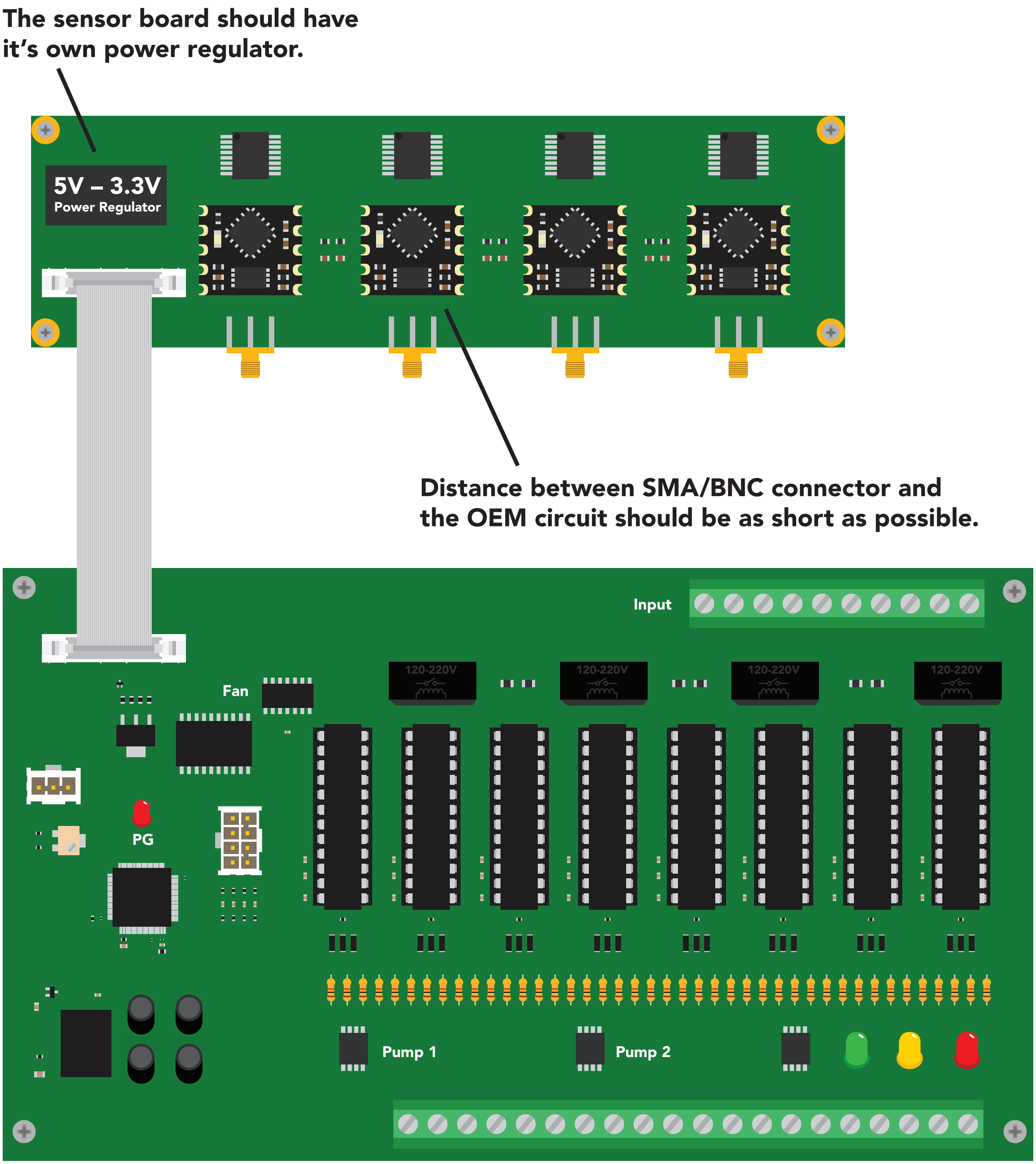


AtlasScientific™ Environmental Robotics

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.

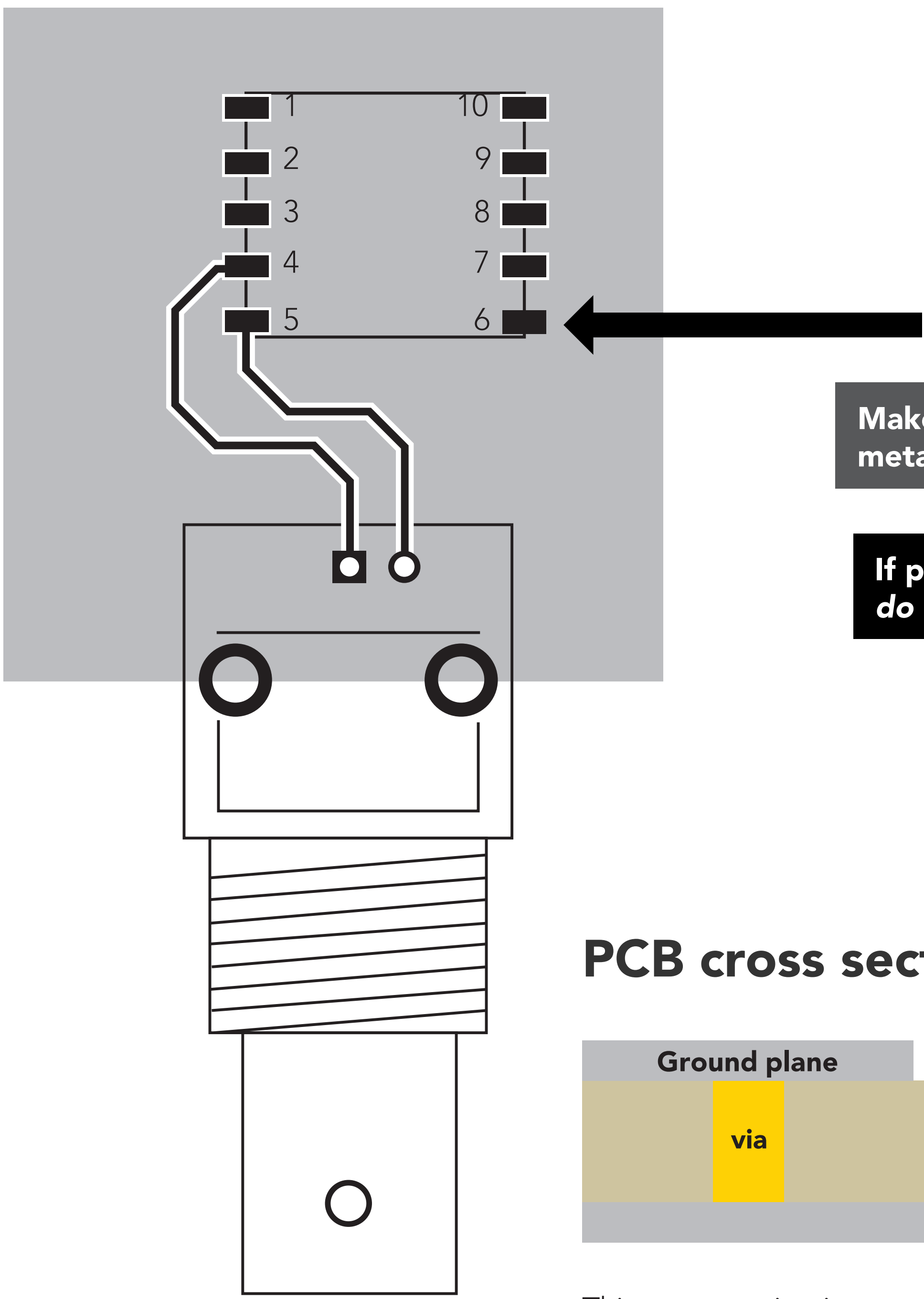
Target customers will experience accurate, stable and repeatable readings for the life of your product.



Designing your PCB

Create the traces as short as possible from the RTD OEM™ 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.

Ground Plane



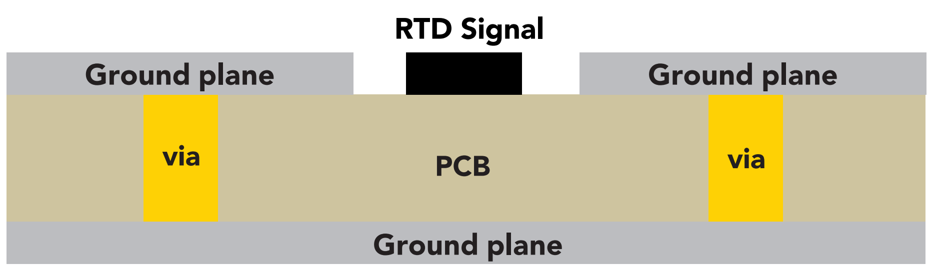
Connect pin 6 to the ground plane.

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

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

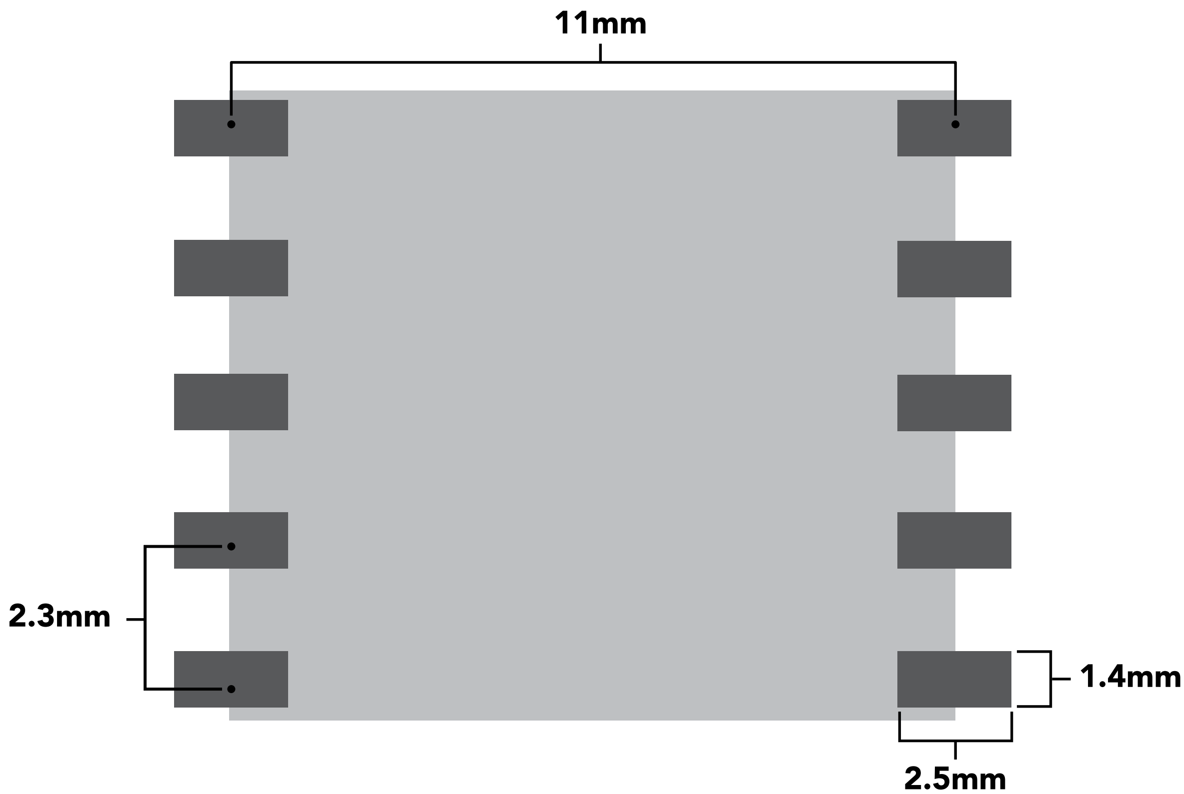
Keep the traces for both probe and probe ground as short as possible.

PCB cross section of the signal path



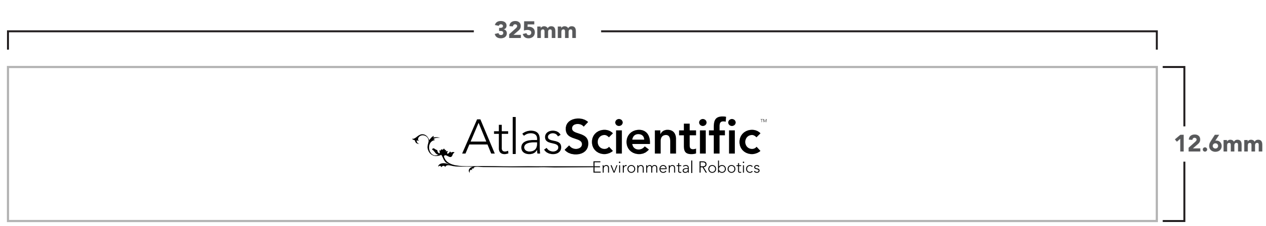
This cross section is an example of how the ground plane protects the RTD signal. The ground plane should surround the RTD signal, on the top layer as well as the bottom layer.

Recommended pad layout

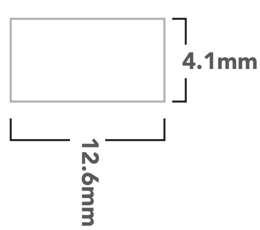


IC tube measurements

Top View



Side View

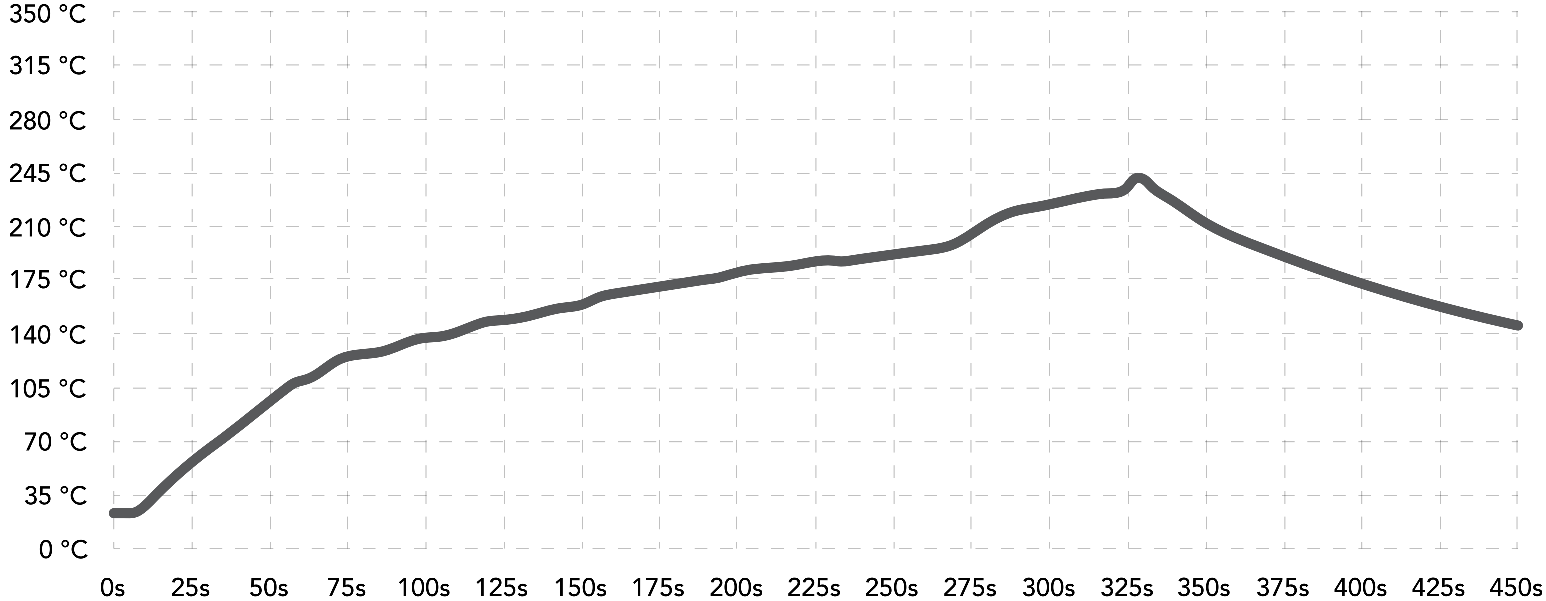


Fits 25 RTD OEM™ circuits

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

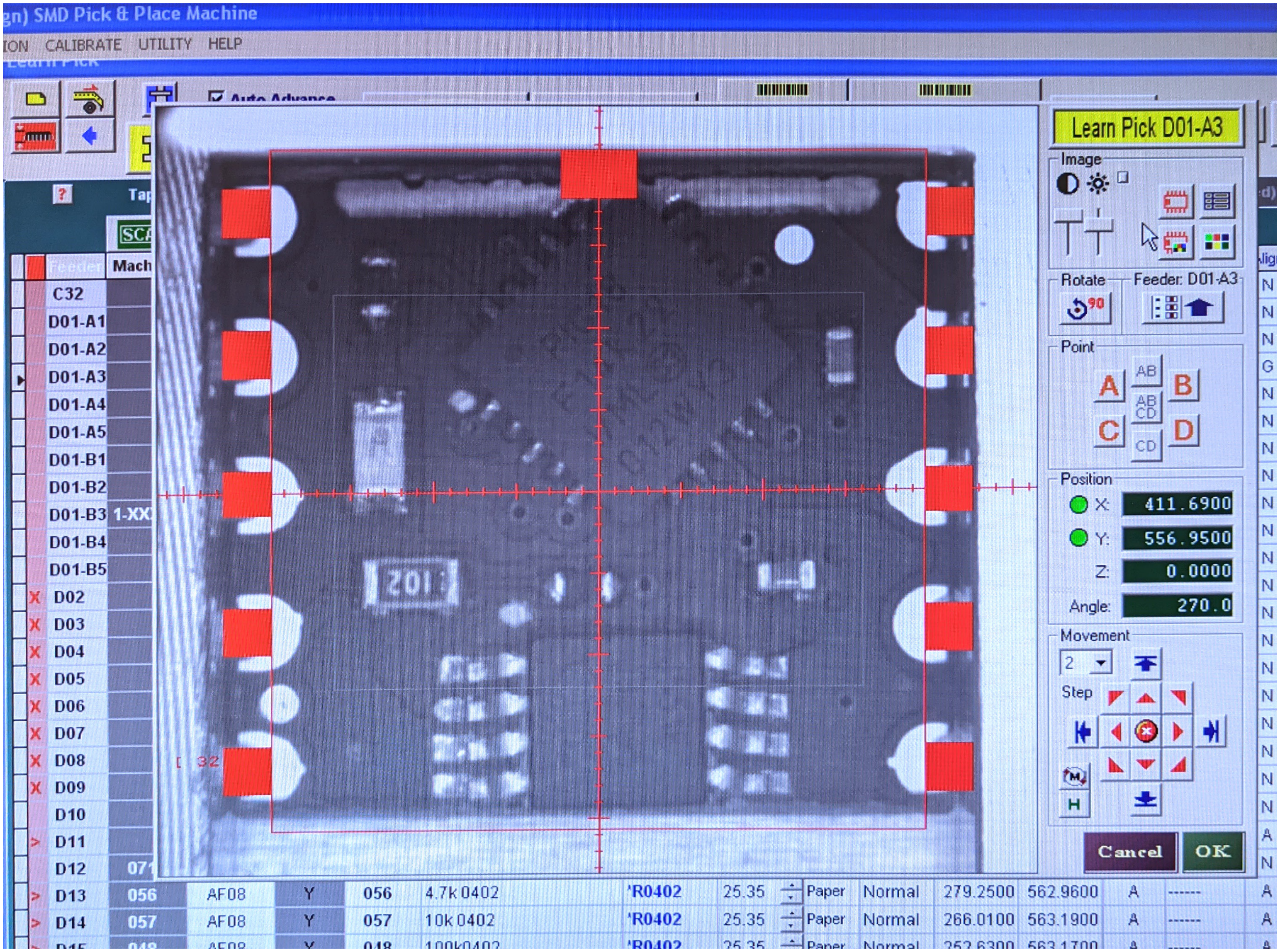
plastic thickness 0.5mm

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



Atlas Scientific Environmental Robotics

Datasheet change log

Datasheet V 1.2

Revised wiring diagram on pg 6.

Datasheet V 1.1

Revised operating voltages on pages 1 & 4.

Datasheet V 1.0

New datasheet

Firmware updates

V4.0 – Initial release (June, 28 2017)

V5.0 – (November 27, 2018)

- Fixed a bug where the calibration status didn't load correctly on power up.