



SmartElex Thermocouple Amplifier - MCP9600

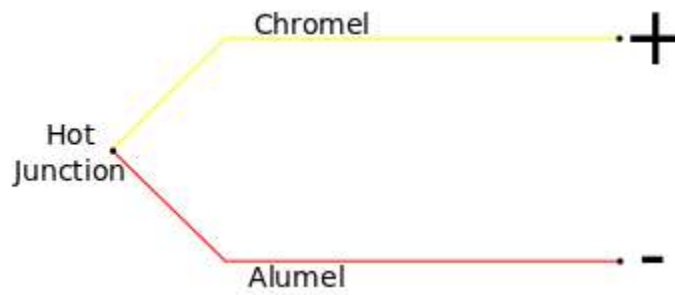
The MCP9600 has a resolution of 0.0625°C , and an accuracy of $\pm 1.5^{\circ}\text{C}$ (worst-case). The board come ready to accept a K-type thermocouple, which gives a temperature range of -200°C to 1350°C . Additionally, the MCP9600 has four onboard temperature alerts for those interrupt triggers as well as the ability to enter alternate operation modes in order to save power.



A Brief Theory of Operation

Thermocouples are fairly ubiquitous, being used in everything from industrial kilns and diesel engines to pilot light sensors and thermostats in your home or office. Let's take a brief moment to go over how they work.

Roughly a couple hundred years ago, a man named Thomas Seebeck discovered the principal that thermocouples use. He noticed that if you take two wires made of dissimilar metals, connect them at the two ends, and make a temperature gradient between one end and the other, a voltage potential formed and current flowed. One junction is held in the environment where the temperature of interest exists. This is known as the hot junction. The other junction is referred to as the cold junction.



K-type thermocouple with cold junction spread for voltage measurement

There are many types of thermocouples, which mainly differ by the types of metals used in the two wires. The most common general purpose thermocouple is **type K**. They are made out of chromel and alumel. These two alloys produce a potential of approximately $41.276 \mu\text{V}/^\circ\text{C}$, and voltage out can be calculated using the equation below.

$$V_{OUT} = \frac{41.276 \mu\text{V}}{^\circ\text{C}} \times (T_{HOT} - T_{COLD})$$

K-type thermocouples can read between from -200°C to $+1350^\circ\text{C}$ (-330°F to $+2460^\circ\text{F}$) and are fairly stable.

Hardware Overview

At the heart of this board is the MicroChip Technology Inc MCP9600 Thermocouple EMF to Temperature Converter. Inside this chip are two temperature sensors, one for the thermocouple itself (the hot junction) and one for the chip itself (the cold junction). In addition, the MCP9600 has four onboard temperature alerts that allow you to set a temperature limit to trigger an interrupt when the temperature reaches a certain value. This frees up your microcontroller and your I²C bus. The MCP9600 can also be put into alternate operation modes in order to save power. The sensor supports a burst mode, where it will take a specifiable number of samples, return the results, and then go to sleep. More information can be found in the MCP9600 Datasheet (PDF).

Power

Ideally, power will be supplied via the connectors on either side of the board. Alternatively, power can be supplied through the header along the bottom side of the board labeled 3V3 and GND. The input voltage range should be between **2.7-5.5V**.

⚡ **Note:** There is no onboard voltage regulation on either of these boards. If you choose to provide power via the plated through holes, ensure that your voltage does not exceed **5.5V**.

I²C address of the board is **0x60 by default**, but has 7 other addresses the board can be configured to use.

I²C Pins

The I²C pins break out the functionality of the connectors. Depending on your application, you can connect to these pins via the plated through holes for SDA and SCL.

Alert Pins

The Thermocouple Amplifier has four configurable alert pins. Each alert can be configured for:

- Temperature
- Hysteresis
- Junction Alert - hot (thermocouple) or cold (internal MCP9600 sensor)
- Edge - from cold to hot, or from hot to cold
- Logic Level - active high or active low
- Alert Mode - interrupt or comparator

Check out the Temperature Alerts section of the Arduino library function descriptions to learn more about how to use the alert pins.

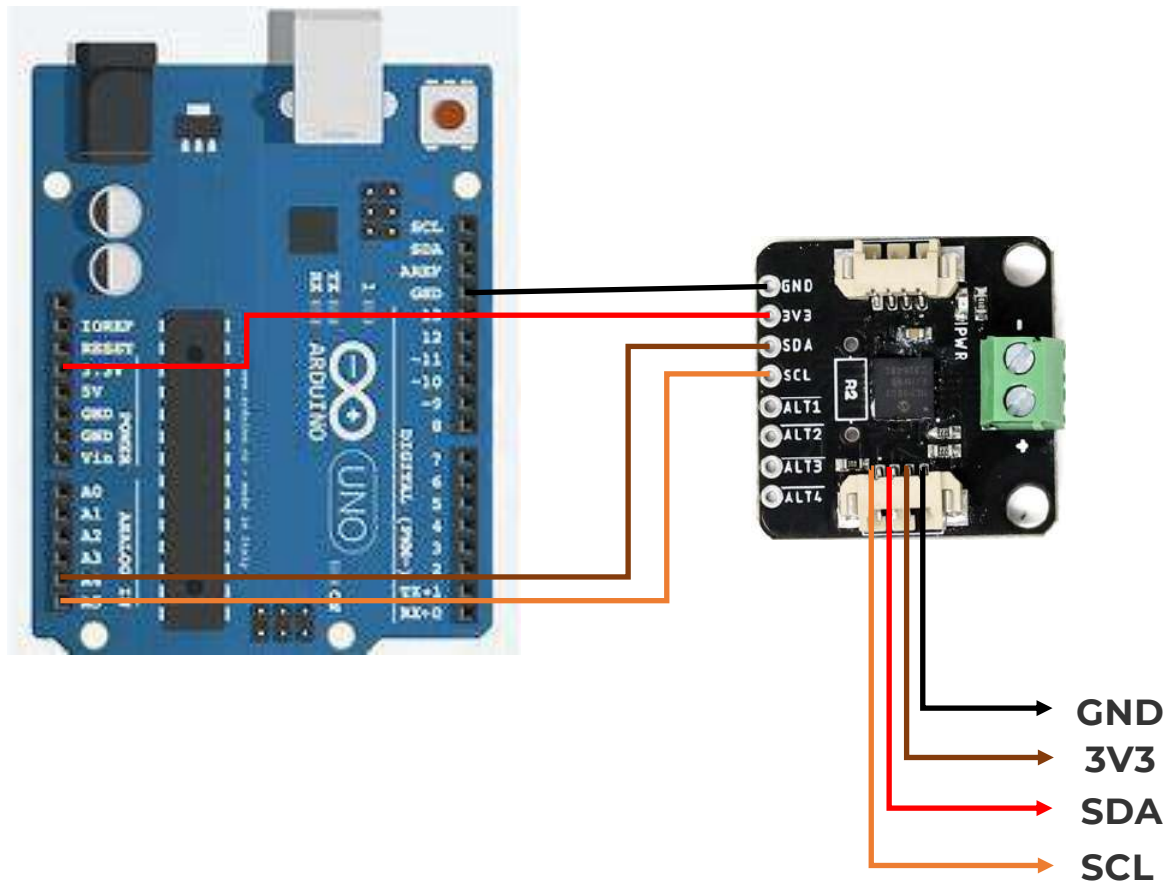
ADDR Jumper

The MCP9600 uses an analog voltage to set the I²C address. By default, the ADDR jumper pulls the address pin low, which gives the board a **default address of 0x60**. Cutting the jumper pulls the address pin high through a 10K resistor, which will change the address to 0x67.

For addresses other than 0x67, a resistor will need to be soldered. Refer to the table below for the resistor value and corresponding address. Resistor location on each board:

Address	Resistor (kΩ)
0x60	0
0x61	2.2
0x62	4.3
0x63	7.5
0x64	13
0x65	22
0x66	43
0x67	N/A

Wiring



Arduino	MCP9600
SCL(A5)	SCL
SDA(A4)	SDA
5v OR 3.3v	3V3
GND	GND

Software Setup and Programming

SparkFun has written a library to work with the Thermocouple. You can obtain this library through the Arduino Library Manager by searching for **MCP9600**. Find the one written by **SparkFun Electronics** and install the latest version.

MCP9600 Library Overview

Device Status

- `bool begin( uint8_t address = DEV_ADDR, TwoWire &wirePort = Wire )` - Sets device I2C address to a user-specified address, over whatever port the user specifies. If left blank, the default address 0x60 is used on the Wire bus.

- **bool available(void)** - Also referred to as the data ready bit, returns true if the thermocouple (hot) junction temperature has been updated since last checked.
- **bool isConnected(void)** - Returns true if the thermocouple will acknowledge over I2C, and false otherwise.
- **uint16_t deviceID(void)** - Returns the contents of the device ID register. The upper 8 bits are constant, but the lower contain revision data.
- **bool checkDeviceID(void)** - Returns true if the constant upper 8 bits in the device ID register are what they should be according to the datasheet.
- **bool resetToDefaults(void)** - Resets all device parameters to their default values. Returns 1 if there was an error, 0 otherwise.

Sensor measurements

- **float getThermocoupleTemp(bool units = true)** - Returns the thermocouple temperature, and clears the data ready bit. Set units to true for Celcius (default), or false for freedom units (Fahrenheit).
- **float getAmbientTemp(bool units = true)** - Returns the ambient (IC die) temperature. Set units to true for Celcius (default), or false for freedom units (Fahrenheit).
- **float getTempDelta(bool units = true)** - Returns the difference in temperature between the thermocouple and ambient junctions. Set units to true for Celcius (default), or false for freedom units (Fahrenheit).
- **signed long getRawADC(void)** - Returns the raw contents of the ADC register.
- **bool isInputRangeExceeded(void)** - Returns true if the MCP9600's EMF range has been exceeded, and false otherwise.

Measurement configuration

- **bool setAmbientResolution(Ambient_Resolution res)** - Changes the resolution on the cold (ambient) junction, for either 0.0625 or 0.25 degree C resolution. Lower resolution reduces conversion time.
- **Ambient_Resolution getAmbientResolution(void)** - Returns the resolution on the cold (ambient) junction, for either 0.0625 or 0.25 degree C resolution. Lower resolution reduces conversion time.
- **bool setThermocoupleResolution(Thermocouple_Resolution res)** - Changes the resolution on the hot (thermocouple) junction, for either 12, 14, 16, or 18-bit resolution. Lower resolution reduces conversion time.

- **Thermocouple_Resolution getThermocoupleResolution(void)** - Returns the resolution on the hot (thermocouple) junction, for either 12, 14, 16, or 18-bit resolution. Lower resolution reduces conversion time.
- **uint8_t setThermocoupleType(Thermocouple_Type type)** - Changes the type of thermocouple connected to the MCP9600. Supported types are K, J, T, N, S, E, B, R.
- **Thermocouple_Type getThermocoupleType(void)** - Returns the type of thermocouple connected to the MCP9600 as found in its configuration register. Supported types are K, J, T, N, S, E, B, R.
- **uint8_t setFilterCoefficient(uint8_t coefficient)** - Changes the weight of the on-chip exponential moving average filter. Set this to 0 for no filter, 1 for minimum filter, and 7 for maximum filter.
- **uint8_t getFilterCoefficient(void)** - Returns the weight of the on-chip exponential moving average filter.
- **bool setBurstSamples(Burst_Sample samples)** - Changes the amount of samples to take in burst mode. Returns 0 if set successfully, 1 otherwise.
- **Burst_Sample getBurstSamples(void)** - Returns the amount of samples to take in burst mode, according to the device's configuration register.
- **bool burstAvailable(void)** - Returns true if all the burst samples have been taken and the results are ready. Returns false otherwise.
- **bool startBurst(void)** - Initiates a burst on the MCP9600.
- **bool setShutdownMode(Shutdown_Mode mode)** - Changes the shutdown "operating" mode of the MCP9600. Configurable to Normal, Shutdown, and Burst. Returns 0 if properly set, 1 otherwise.
- **Shutdown_Mode getShutdownMode(void)** - Returns the shutdown "operating" mode of the MCP9600. Configurable to Normal, Shutdown, and Burst.

Temperature Alerts

- **bool configAlertTemp(uint8_t number, float temp)** - Configures the temperature at which to trigger the alert for a given alert number.
- **bool configAlertJunction(uint8_t number, bool junction)** - Configures the junction to monitor the temperature of to trigger the alert. Set to zero for the thermocouple (hot) junction, or one for the ambient (cold) junction.
- **bool configAlertHysteresis(uint8_t number, uint8_t hysteresis)** - Configures the hysteresis to use around the temperature set point, in degrees Celcius.

- **bool configAlertEdge(uint8_t number, bool edge)** - Configures whether to trigger the alert on the rising (cold -> hot) or falling (hot -> cold) edge of the temperature change. Set to 1 for rising, 0 for falling.
- **bool configAlertLogicLevel(uint8_t number, bool level)** - Configures whether the hardware alert pin is active-high or active-low. Set to 1 for active-high, 0 for active-low.
- **bool configAlertMode(uint8_t number, bool mode)** - Configures whether the MCP9600 treats the alert like a comparator or an interrupt. Set to 1 for interrupt, 0 for comparator. More information is on pg. 34 of the datasheet.
- **bool configAlertEnable(uint8_t number, bool enable)** - Configures whether or not the interrupt is enabled or not. Set to 1 to enable, or 0 to disable.
- **bool clearAlertPin(uint8_t number)** - Clears the interrupt on the specified alert channel, resetting the value of the pin.
- **bool isTempGreaterThanLimit(uint8_t number)** - Returns true if the interrupt has been triggered, false otherwise

Example Code

Once you've installed the library, you should be able to find the examples in Arduino under **File > Examples > SparkFun MCP9600 Thermocouple Library**.

Example 1: Basic Readings

While all of the examples are well commented, let's quickly run through example 1 just to get your feet wet. The best part? It's pretty much plug and play. So go ahead and open up Example 1, or alternatively, copy and paste the code below into an Arduino window:

```
#include <SparkFun_MCP9600.h>
```

```
MCP9600 tempSensor;
```

```
void setup(){
```

```
  Serial.begin(115200);
```

```
  Wire.begin();
```

```
  Wire.setClock(100000);
```

```
tempSensor.begin();    // Uses the default address (0x60) for SparkFun  
Thermocouple Amplifier
```

```
//tempSensor.begin(0x66); // Default address (0x66) for SparkX Thermocouple  
Amplifier
```

```
//check if the sensor is connected
```

```
if(tempSensor.isConnected()){
```

```
    Serial.println("Device will acknowledge!");
```

```
}
```

```
else {
```

```
    Serial.println("Device did not acknowledge! Freezing.");
```

```
    while(1); //hang forever
```

```
}
```

```
//check if the Device ID is correct
```

```
if(tempSensor.checkDeviceID()){
```

```
    Serial.println("Device ID is correct!");
```

```
}
```

```
else {
```

```
    Serial.println("Device ID is not correct! Freezing.");
```

```
    while(1);
```

```
}
```

```
}
```

```
void loop(){ //print the thermocouple, ambient and delta temperatures every 200ms if  
available
```

```
  if(tempSensor.available()){
```

```
    Serial.print("Thermocouple: ");
```

```
    Serial.print(tempSensor.getThermocoupleTemp());
```

```
    Serial.print(" °C  Ambient: ");
```

```
    Serial.print(tempSensor.getAmbientTemp());
```

```
    Serial.print(" °C  Temperature Delta: ");
```

```
    Serial.print(tempSensor.getTempDelta());
```

```
    Serial.print(" °C");
```

```
    Serial.println();
```

```
    delay(20); //don't hammer too hard on the I2C bus
```

```
  }
```

```
}
```

What You Should See

Open up the Serial Monitor and make sure your baud rate is 115200. You should see something like the image below. If you move the hot junction (ie, the thermocouple wand) closer to a heat source, you should see the Thermocouple temperature go up.

