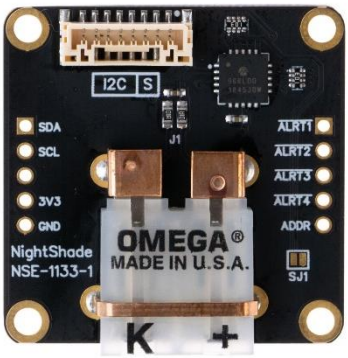
Thermocouple Converter - MCP96L00 - Trēo™ Module

Module Features

- Microchip MCP96L00
- RoHS Compliant
- Software Library
- NightShade Trēo™ Compatible
- Breakout Headers



MCP96L00 Features

(from Microchip)

- Supports K, J, T, N, S, E, B, & R Type Thermocouples
- 0.0625°C Resolution
- Internal Cold-Junction Temperature Compensation
- Programable Rising or Falling Temperature Alerts for Hot and Cold Junctions
- Low Power Consumption

Applications

- Hand-held Temperature Measurement
- Process Control
- Oven Temperature Control

Description

The MCP96L00 Trēo™ Module is a Thermocouple Converter module that that features Microchip's MCP96L00 Thermocouple Converter. It converts the electromotive force (EMF) generated by a thermocouple to a temperature in degrees Celsius. Alerts can be set for the temperature rising or falling past the set limit on the hot or cold temperature junctions. The alert can be used to generate a hardware interrupt at the host. This module is a part of the NightShade Treo system, patent pending.

Trēo™ Compatibility

Electrical

Communication	I2C
Max Current, 3.3V	3mA
Max Current, 5V	0mA

Mechanical

- 35mm x 35mm Outline
- 30mm x 30mm Hole Pattern
- M2.5 Mounting Holes

Table of Contents

1 Summary

The MCP96L00 device can be operated in three different modes: continuous read, burst read, and shutdown modes. Continuous read mode continually samples the temperature sensor, which allows for heavy filtering of the result. The drawback to this mode is that the device suffers self-heating which adds error to the temperature measurement. This can be combated by using the burst read mode. A burst read of X samples is measured and then filtered, providing a filtered data point. Once the burst is complete, the device enters shutdown mode to save power and prevent self-heating between samples. The thermocouple temperature is read with the readTemperature() method and a burst is collected using the setBurstRead() method. The result of a burst read is retrieved with the readTemperature() method after the burst is complete. The user can determine if a single read or burst read is complete using the dataRead() and burstReady() methods, respectively. Measurement details such as thermocouple type, resolution, and alert limits can be set using the other methods in this library.

2 What is Trēo™?

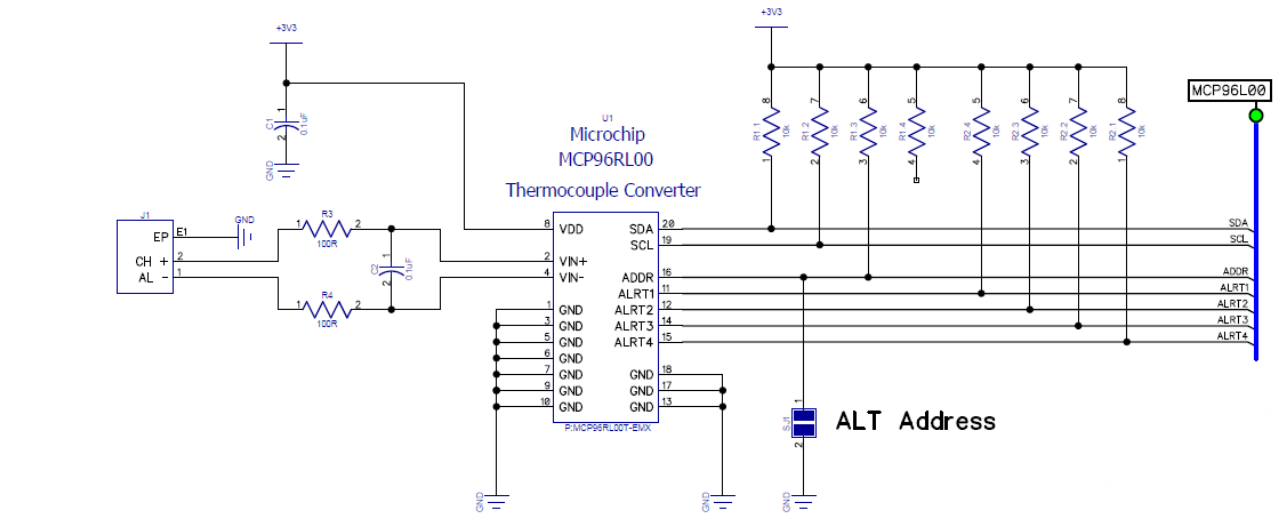
NightShade Trēo is a system of electronic modules that have standardized mechanical, electrical, and software interfaces. It provides you with a way to quickly develop electronic systems around microprocessor development boards. The grid attachment system, common connector/cabling, and extensive cross-platform software library allow you more time to focus on your application. Trēo is supported with detailed documentation and CAD models for each device.

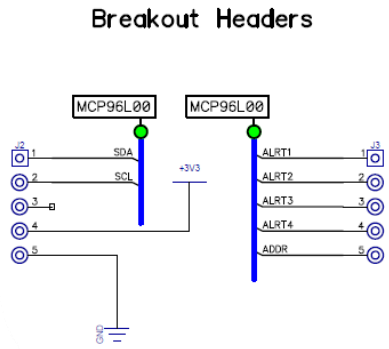
Learn more about Trēo here.

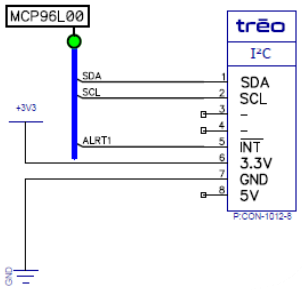
3 Electrical Characteristics

	Minimum	Nominal	Maximum
Voltages			
$V_{i/o}$ (SDA, SCL, INT)	-0.3V	-	3.6V
$V_{3.3V}$	3.1V	3.3V	3.5V
Measurement			
Sampling Rate	3.125Hz	-	200Hz
Measurement Range	-2047.9375°C	-	+2047.9375°C
Precision (Max Res.)	0.0625°C	-	-
Error	-	2.0°C	4.0°C
I2C Slave Address			
SJ1 Open (Default)		0x67	
SJ1 Closed (Soldered)		0x60	
Operating Temperature	-25°C	-	+85°C

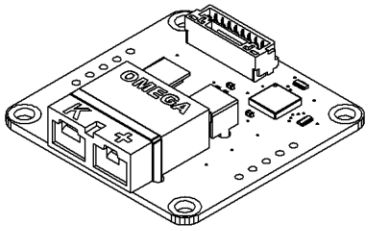
4 Electrical Schematic

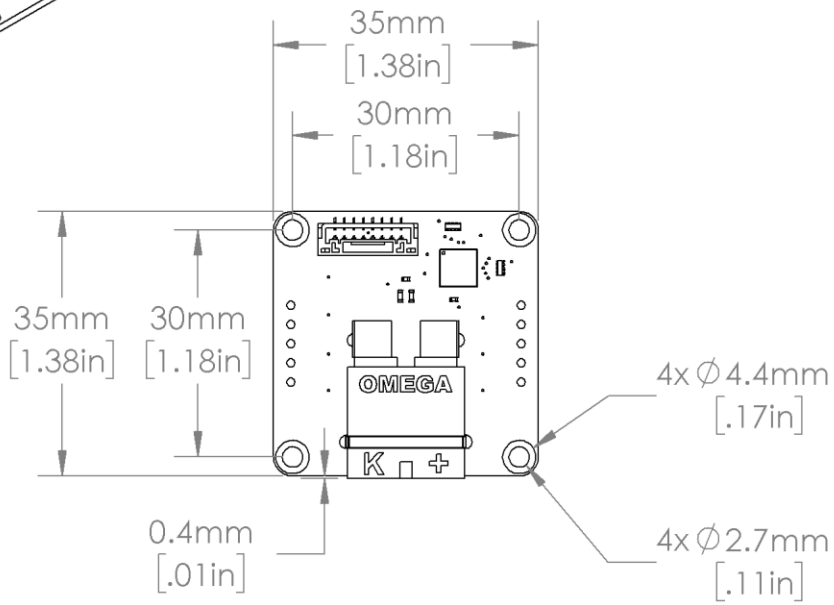


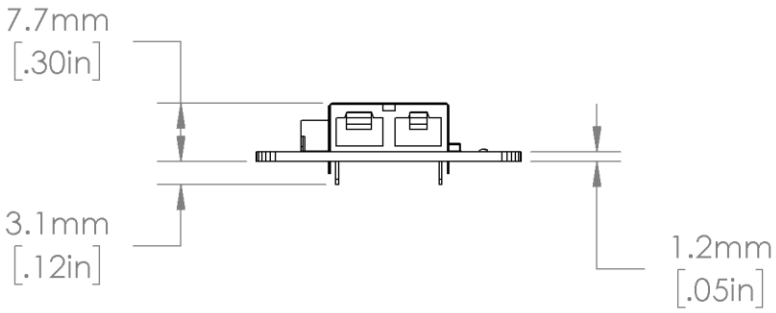




5 Mechanical Outline







6 Example Arduino Program

```
/*******************************************************
  MCP9600_ThermocoupleConverter - NightShade_Treo by NightShade Electronics

  This sketch demonstrates the functionality of the
  NightShade Trēo MCP9600 thermocouple converter module.
  (NSE-1133-1) It prints the measured temperature to
  Serial at 115200 baudrate.

  Created by Aaron D. Liebold
  on February 15, 2021

  Links:
  NightShade Trēo System: https://nightshade.net/treo
  Product Page: https://nightshade.net/product/treo-thermocouple-converter-mcp96l00/

  Distributed under the MIT license
  Copyright (C) 2021 NightShade Electronics
  https://opensource.org/licenses/MIT
*******************************************************/

// Include NightShade Treo Library
#include <NightShade_Treo.h>

// Declare Objects
NightShade_Treo_MCP9600 temp(1);

void setup() {
  temp.begin();
  Serial.begin(115200);
}

void loop() {
  if (temp.dataReady()) {
    Serial.print("Temp: ");
    Serial.print((float) temp.readTemperature()*0.0625);
    Serial.println("°C");
  }
  delay(1000);
}
```

7 Library Overview (C++ & Python)

C++ Class

NightShade_Treo_MCP9600 <classObject>();

Python Module

<classObject> = NightShade_Treo.MCP9600()

7.1 Constructors

NightShade_Treo_MCP9600(int port, uint8_t slaveAddress, uint32_t clockSpeed)

Creates a MCP96L00 object.

Arguments:

port	Integer of the I2C port used (e.g. 0 = "/dev/i2c_0")
slaveAddress	7-bit slave address
clockSpeed	Desired clock speed for the bus

Returns:

Nothing

NightShade_Treo_MCP9600(int port)

Creates a MCP96L00 object assuming the default slave address and clock speed.

Arguments:

port	Integer of the I2C port used. (e.g. 0 = "/dev/i2c_0")

Returns:

Nothing

7.2 Methods

begin()

Initializes the module.

Arguments:

None

Returns:

Error	0 = Success

setThermocoupleType(int setting)

Sets the type of thermocouple being used.

Arguments:

setting	0: Type K 1: Type J 2: Type T 3: Type N 4: Type S 5: Type E 6: Type B 7: Type R

Returns:

Error	0 = Success

setFilterCoefficient(int setting)

Sets the coefficient for the low-pass data filter.

Arguments:

setting	0 (No Filtering) 1 (Minimum Filtering) 2 3 4 (Medium Filtering) 5 6 7 (Maximum Filtering)

Returns:

Error	0 = Success

setColdJunctionRes(int setting)

Sets the resolution of the cold-junction temperature measurement.

Arguments:

setting	0: 0.0625°C 1: 0.25°C

Returns:

Error	0 = Success

setMeasurementRes(int setting)

Sets the resolution of the hot-junction temperatures measurement.

Arguments:

setting	0: 18-Bit Measurement 1: 16-bit Measurement 2: 14-bit Measurement 3: 12-bit Measurement

Returns:

Error	0 = Success

setPowerMode(int setting)

Sets the operational mode.

Arguments:

setting	0: Normal Operation 1: Shutdown Mode 2: Burst Mode

Returns:

Error	0 = Success

setAlertLimit(int alertNumber, int value)

Sets the alert limit for alert 1, 2, 3, or 4. The Alert 1 pin is connected to the hardware interrupt pin.

Arguments:

alertNumber	1: Alert 1 (HW Interrupt) 2: Alert 2 3: Alert 3 4: Alert 4
value	Alert limit value

Returns:

Error	0 = Success

setAlertHyst(int alertNumber, uint8_t value)

Sets the alert limit hysteresis for alert 1, 2, 3, or 4. The Alert 1 pin is connected to the hardware interrupt pin.

Arguments:

alertNumber	1: Alert 1 (HW Interrupt) 2: Alert 2 3: Alert 3 4: Alert 4
value	Alert limit hysteresis value

Returns:

Error	0 = Success

setupAlert(int alertNumber, int selectJunction, int triggerDirection, int enableAlertOutput)

Sets the conditions for alert 1, 2, 3, or 4. The alert output pins are active-low and the Alert 1 pin is connected to the hardware interrupt pin.

Arguments:

alertNumber	1: Alert 1 (HW Interrupt) 2: Alert 2 3: Alert 3 4: Alert 4
selectJunction	0: Alert monitors cold-junction temperature 1: Alert monitors hot-junction temperature
triggerDirection	0: Alert on rising (heating) temperatures 1: Alert on falling (cooling) temperatures
enableAlertOutput	0: Alert output is disabled 1: Alert output is enabled

Returns:

Error	0 = Success

clearAlertInterrupt(int alertNumber)

Clears an active alert.

Arguments:

alertNumber	1: Alert 1 (HW Interrupt) 2: Alert 2 3: Alert 3 4: Alert 4

Returns:

Error	0 = Success

readAlertLimit(int alertNumber)

Returns the limit value of Alert 1, 2, 3, or 4.

Arguments:

alertNumber	1: Alert 1 (HW Interrupt) 2: Alert 2 3: Alert 3 4: Alert 4

Returns:

Alert limit value (int)

readStatusReg()

Returns the value of the Status register.

Arguments:
None

Returns:
register value (uint8_t)

readTemperature()

Reads the thermocouple hot-junction temperature.

Arguments:
None

Returns:
Temperature (int, 0.0625°C/LSB)

readColdJuncTemperature()

Reads the cold-junction temperature.

Arguments:
None

Returns:
Temperature (int, 0.0625°C/LSB)

readDeltaTemperature()

Reads the delta between the hot-junction and cold-junction temperatures.

Arguments:
None

Returns:
Temperature (int, 0.0625°C/LSB)

dataReady()

Returns a 1 when a new temperature conversion is completed. This flag is cleared when the readTemperature() method is called.

Arguments:
None

Returns:
status (int) 0: Temperature read in progress
1: Temperature read has completed

setBurstRead(int setting)

Measures a set number of samples and then enters shutdown mode. This can be used instead of continuous mode to limit the amount of self-heating in the device. A set of samples is taken, rather than a single sample, so that the samples can be filtered. The result is read with the readTemperature() method.

Arguments:

setting	0: 1 sample 1: 2 samples 2: 4 samples 3: 8 samples 4: 16 samples 5: 32 samples 6: 64 samples 7: 128 samples

Returns:

Error	0 = Success

burstReady()

Returns 1 when the burst read has completed. This flag is cleared when the setBurstRead() method is called.

Arguments:

None

Returns:

status (int)	0: Burst read in progress 1: Burst read has completed