TECHNICAL REPORT: I&C CHALLENGE- RTD TEMEPERATURE MEASUREMENT: PT100/PT1000

Nindo Emmanuel
May 5, 2025

INTRODUCTION

Goal:

- Measure the resistance of a Pt100 or Pt1000 RTD using the ADS1220 24-bit ADC.
- Send it over SPI to a microcontroller (ESP32 Node MCU-32s).
- Convert it to temperature on

CIRCUIT OVERVIEW

This is an implementation of a 2/3/4-wire RTD measurement circuit configuration, which removes the effect of wire resistance, electrical noise, and converts the readings to relative temperature measurements.

Key Components

- RTD (Pt100 or Pt1000)
- ADS1220 (24-bit precision ADC with internal reference and excitation current source)
- ESP32 (controller for data processing and SCADA output)

Pt100/Pt1000 support

This implementation supports the two types of RTDs by measuring the effective resistance and comparing it to a rage relative to its normal working resistance and thus outputs the correct referencing resistance.

Noise Mitigation

Low-pass filters are implemented at the voltage measurement references AIN0 and AIN1 and at the reference voltage pins circuit. This ensures electrical noise is filtered from the system.

Self-Heating Mitigation

The use of a stabilized 3.3V source ensured low voltage fluctuations, thus reducing the self-heating characteristic of RTD. Though the self-heating phenomenon cannot be totally

reduced this is one of the methods that may limit the extent of error in the readings from the RTDs.

SCADA/HMI Analog / Serial Output

SCADA systems typically accept a voltage range such as 0–3.3 V or 0–5 V or serial communication, UART. Scaling the temperature (0–100°C) linearly to DAC output and giving an analog output of this using `dacWrite(DAC_PIN, value);` on ESP32, where `value ∈ [0, 255]` for 8-bit DAC or through Serial monitor.

Firmware calculation

ADS1220 is a 24-bit ADC, giving a signed range of:

$$\pm 223 = \pm 8{,}388{,}608\ counts$$

Internal Reference voltage may be computed as :

$$Vref = Vin \times R21/(R20 + R21)$$
$$Vref = 3.3V \times 4.7/(10k\Omega + 4.7k\Omega) = 3.3V \times 14.74.7 \approx 1.65V$$

With an internal reference of 1.65V and a gain of 1, the input voltage is:

$$VRTD = (raw\_ADC \times 2.04)/88{,}388{,}608$$

Then:

$$RRTD = \frac{VRTD}{IEXC} = \frac{raw \times 1.65}{88{,}388{,}608 \times 0.00025}$$

Converting Resistance to Temperature

The Callendar-Van Dusen equation for a platinum RTD:

$$T = \frac{RRTD - R0}{\alpha R_0}$$

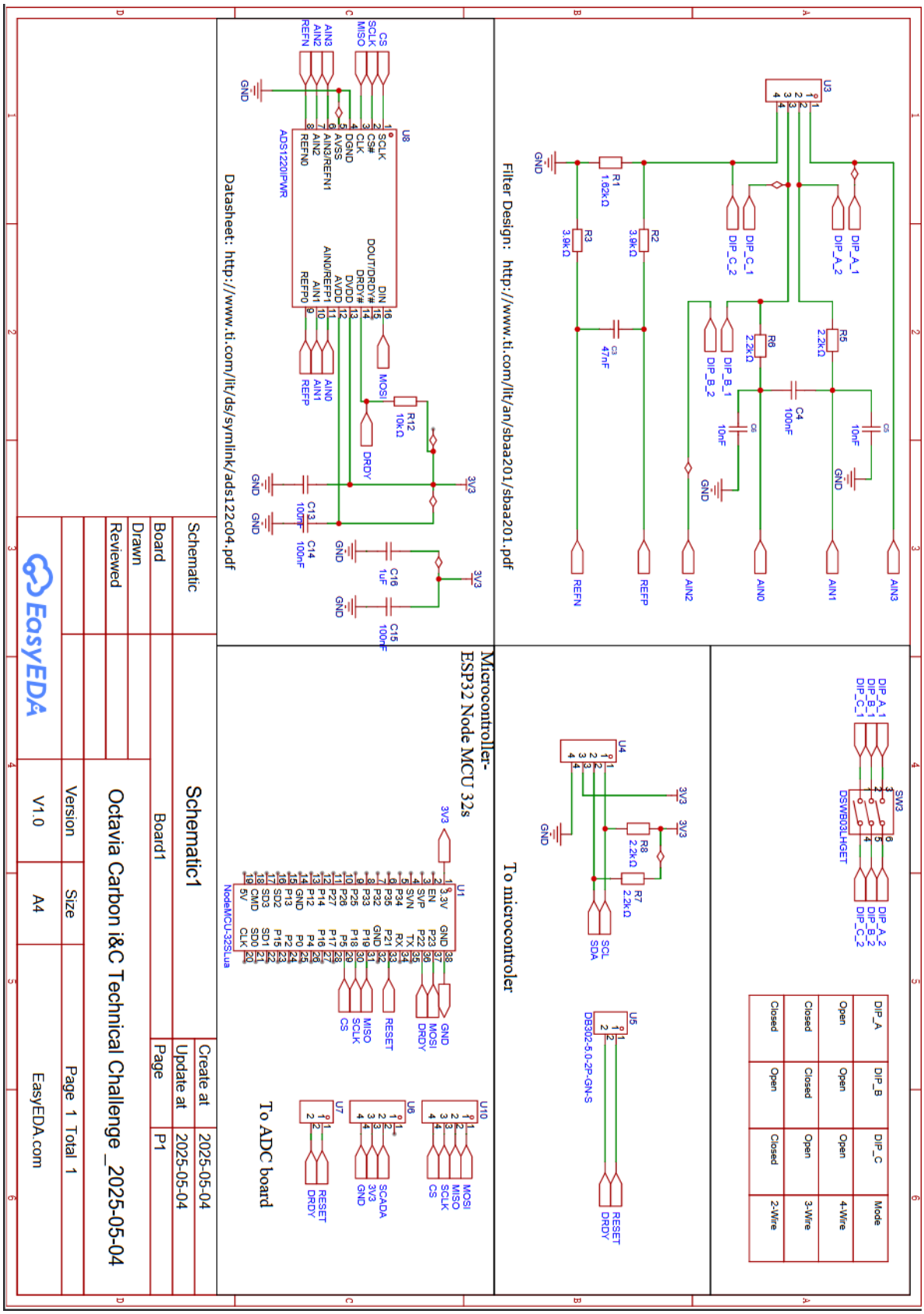
Where:

R_0=100Ω (Pt100) or 1000 Ω (Pt1000)

α=0.00385 for standard RTDs

This approximation works accurately between 0°C and 200°C

Appendices


Filter Design: http://www.ti.com/lit/an/sbaa201/sbaa201.pdf
Datasheet: http://www.ti.com/lit/ds/symlink/ads122c04.pdf
To microcontroler
Microcontroller-
ESP32 Node MCU 32s
To ADC board
Schematic1
Board1
Octavia Carbon i&C Technical Challenge _2025-05-04
EasyEDA
V1.0
A4
Page 1 Total 1
EasyEDA.com
2025-05-04
P1

Link to GitHub repository: https://github.com/sirNindo/RTD_Pt100-Pt100-.git

```
#include <SPI.h>

#define CS_PIN     5     // Chip Select
#define DRDY_PIN   4     // Data Ready from ADS1220
#define DAC_PIN    25    // DAC1 output (GPIO25) to SCADA (analog voltage output)
// ADS1220 SPI Commands
#define CMD_RESET   0x06
#define CMD_START   0x08
#define CMD_RDATA   0x10
#define CMD_WREG    0x40
bool isPt1000 = false;  // Default is Pt100
void writeRegister(uint8_t reg, uint8_t value) {
  digitalWrite(CS_PIN, LOW);
  SPI.transfer(CMD_WREG | (reg & 0x03)); // Write 1 register
  SPI.transfer(0x00); // Number of registers to write - 1
  SPI.transfer(value);
  digitalWrite(CS_PIN, HIGH);
}
void sendCommand(uint8_t cmd) {
  digitalWrite(CS_PIN, LOW);
  SPI.transfer(cmd);
  digitalWrite(CS_PIN, HIGH);
}
void setupADS1220() {
  sendCommand(CMD_RESET);
  delay(100);
```

```
  // Config:
  // MUX = AIN0-AIN1, Gain = 1
  writeRegister(0x00, 0x08);
  // Data rate = 20SPS
  writeRegister(0x01, 0x04);
  // Internal reference, no burnout
  writeRegister(0x02, 0x40);
  writeRegister(0x03, 0x00); // Default settings

  sendCommand(CMD_START);
}
void setup() {
  Serial.begin(115200);
  pinMode(CS_PIN, OUTPUT);
  pinMode(DRDY_PIN, INPUT);
  digitalWrite(CS_PIN, HIGH);
  SPI.begin();
  delay(100);
  setupADS1220();
  // Initial auto-detection of RTD type
  long raw = readRawData();
  float resistance = convertToResistance(raw);
  isPt1000 = resistance > 200.0; // Threshold to differentiate Pt100/Pt1000
  Serial.print("RTD Type Detected: ");
  Serial.println(isPt1000 ? "Pt1000" : "Pt100");
  analogWriteResolution(8);  // Set resolution for DAC output
}
long readRawData() {
  while (digitalRead(DRDY_PIN) == HIGH); // Wait for DRDY = LOW
```

```
  digitalWrite(CS_PIN, LOW);
  SPI.transfer(CMD_RDATA);
  uint8_t b1 = SPI.transfer(0x00);
  uint8_t b2 = SPI.transfer(0x00);
  uint8_t b3 = SPI.transfer(0x00);
  digitalWrite(CS_PIN, HIGH);
  long raw = ((long)b1 << 16) | ((long)b2 << 8) | b3;
  if (raw & 0x800000) raw |= 0xFF000000; // Sign extend
  return raw;
}
float convertToResistance(long raw, float vref = 1.65, float gain = 1.0, float i_exc = 0.00025) {
  float voltage = (raw * vref) / (gain * 8388608.0); // 2^23
  return voltage / i_exc;
}
float resistanceToTemperature(float R) {
  float R0 = isPt1000 ? 1000.0 : 100.0;
  float alpha = 0.00385;
  return (R - R0) / (R0 * alpha);  // Approximate linear temperature value for 0–200°C
}
void outputToSCADA(float temperature) {
  // Scale temp to voltage: -200°C = 0V, 800°C = 3.3V pt100 operation range
  float clampedTemp = constrain(temperature, 200.0, 800.0);
  float voltage = clampedTemp * (3.3 / 100.0);
  int dacValue = map(voltage * 1000, 0, 3300, 0, 255); // For 8-bit DAC
  Serial.print("Temperature:");
  Serial.print(dacValue)  ;
  dacWrite(DAC_PIN, dacValue);
}
void loop() {
```

```
  long raw = readRawData();
  float resistance = convertToResistance(raw);
  float temperature = resistanceToTemperature(resistance);
  // Print to Serial
  Serial.print("Raw ADC: "); Serial.print(raw);
  Serial.print(" | Resistance: "); Serial.print(resistance, 2); Serial.print(" Ω");
  Serial.print(" | Temp: "); Serial.print(temperature, 2); Serial.println(" °C");
  // Output to SCADA via DAC
  outputToSCADA(temperature);
  delay(1000);
}
```