Expertise
and insight
for the future

Oleg Languev

Simulator of the resistive temperature sensor

Metropolia University of Applied Sciences

Bachelor of Engineering

Electronics

Bachelor's Thesis

2 December 2019

Author Title Number of Pages Date	Oleg Languev Simulator of the resistive temperature sensor 38 pages + 8 appendices 2 December 2019
Degree	Bachelor of Engineering
Degree Programme	Electronics
Professional Major	Electronics
Instructors	Matti Fischer, Principal Lecturer Iiro Koskinen, Software Engineer, M.Sc. Tero Kapanen, Hardware and Platform Engineering Manager, M.Sc. Altti Helläkoski, VP Engineering, M.Sc.

The given work represents the development of the simulator of resistance temperature sensors of two types: PT100 and PT1000. The purpose is the simulation of changing temperature for EKE Electronics. Ltd. temperature monitoring modules and reducing the number of manual manipulations.
The PT simulator uses digital potentiometers in order to produce the resistance in ten degrees wider range than modules may measure. Potentiometers receive the commands by I2C bus from Python scripts.
The developed board works with Raspberry Pi as a hat. It is controllable via SSH or can be used with the monitor and keyboard. Temperature can be programmable set with error in one or two degrees, the whole tested range is covered with the produced temperature. Set commands of PT simulator can be sent from Raspberry Pi or from connected PC.
The achieved accuracy is enough to test the temperature monitoring software. Improving accuracy and stability of the output is the goal of the future board development.

Keywords	Temperature simulation, temperature module, RTD

Contents

List of Abbreviations

PCB	Printed Circuit Board
PTI	PTI sensor input module
RTD	Resistance temperature detector
PT100	Platinum temperature sensor with positive resistance coefficient, resistance $R_{0°C}$ = 100 Ω
PT1000	Platinum temperature sensor with positive resistance coefficient, resistance $R_{0°C}$ = 1000 Ω
CPU	Central processing unit
SUT	System under test
RPI	Raspberry Pi
I^2C	“Inter-Integrated Circuit”, synchronous serial computer bus
GPIO	A general-purpose input/output
DMM	A digital multimeter

1 Introduction

Temperature is one of the core parameters in the train safety monitoring. The number of temperature sensors can be several hundred in complicated train systems. EKE-Electronics Ltd. provides technical solutions for train automation, onboard safety and remote condition monitoring, including temperature. Overheat can activate the safety functions.

Integration testing of the software tends to increase automation and decrease manual testing. Manual testing is slower, more error-prone and cannot be done as often as automated testing.

EKE-Electronics Ltd. develops and uses PTI modules (PT input modules) to measure resistance from the PT sensors (platinum resistance thermometers) and converts resistance to temperature. There are two models: one for PT100 and another for PT1000. PT100 and PT1000 are resistance temperature detectors (RTD) that can bear the temperature range (-200, 660) °C. For PT100 resistivity is 100 Ω at 0 °C, for PT1000 resistivity is 1000 Ω.

Both modules have the same output characteristics: temperature range [-110, 325] °C, 0.01°C resolution, ±1°C uncertainty. Each module measures resistance from six channels. One channel measures one PT sensor.

Actual testing system includes two types of test equipment: one block consists of six manually controlled potentiometers with knobs. Another block consists of six precise resistors, providing stable temperature.

Temperature input is implemented with setting up of the resistance with a multimeter. Then block is connected to the PTI module. Temperature changes are provided by manual adjustment of the potentiometer values. Test user tunes up one or two channels at time by turning the knob, although control of more channels is challenging. Providing a complex adjustment for multiple PTI modules at the same time is almost impossible, because user should use both hands to turn two knobs.

Manual testing itself has significant limitations. High probability of human error, slow input, issues with time-based input or complicated patterns, troubles with longstanding testing, poor scalability. In addition to generic manual testing issues, actual test equipment cannot show the input temperature, because knobs have no marking, and there is no possibility to read the resistance after the test box has been connected to PTI module.

It is highly desirable to develop a new test equipment. Simulating the resistance over some predefined range can minimize the manual input. The PT simulator should provide the opportunity to develop an automated input with clear temperature reading during the testing process.

The project scope consists of:

- Developing of the two temperature simulators: PT100 and PT1000,
- Output temperature range should be [-120, 335] °C. It is slightly wider than PTI module temperature range [-110, 325] °C. The reason is coverage of the boundary values,
- Required resolution is 0.35°C,
- Required accuracy is ±1°C,
- Possibility to input the temperature value without translation to resistance,
- Changing temperature in user interface.

Benefits of using PT simulator are following:

- Possibility to use in stand-alone mode or as the part of automated testing environment,
- Reducing testing costs and involving personnel,
- Reducing probability of erroneous input,
- Availability of regressive testing after every change in software,
- Possibility to create more complex test cases.

2 Theoretical background

Train system includes several different input/output modules. One of them is PTI module, measuring resistance and translating it to a temperature. Thus, for testing purposes, input should be simulated. Current solution is using digital potentiometers. There are two main advantages of that way:

- Resistive output is passive and closely imitates PT sensor,
- Circuit is relatively compact and simple.

2.1 PTI module

PTI module measures temperature from resistance thermometers (PT sensors) to six independent channels. Data are transmitted from the input of PTI module to the CPU module (central processing unit). Figure 1 describes the data flow between CPU and PTI module. One CPU can control several PTI modules. IO bus connects all the modules inside rack.

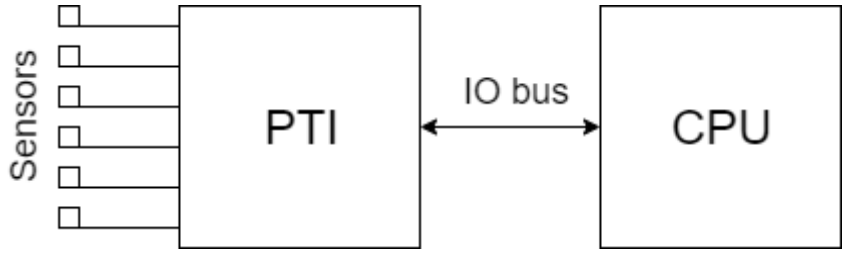


Figure 1. PTI-CPU connection from technical manual of PTI3593A [1, p. 11]

PTI module is the part of IO of the train system. Data processing in the module include resistance to temperature translation and checking the channel error: “measurement out of range: < -110 °C or > 325 °C”.

PTI modules measure temperature only in range [-110, 325] °C. If temperature is higher or lower the limits, channel error will be sent. There are two different PTI modules:

- PTI2037A measures PT100 sensors with 1 mA excitation current,
- PTI3593A measures PT1000 sensors with 250 µA excitation current.

Figure 2 shows the functional diagram of PTI2037A. Functional diagram for PTI3593A is the same, with PT1000 sensor on the input. The differences are about which sensor type and excitation current are used.

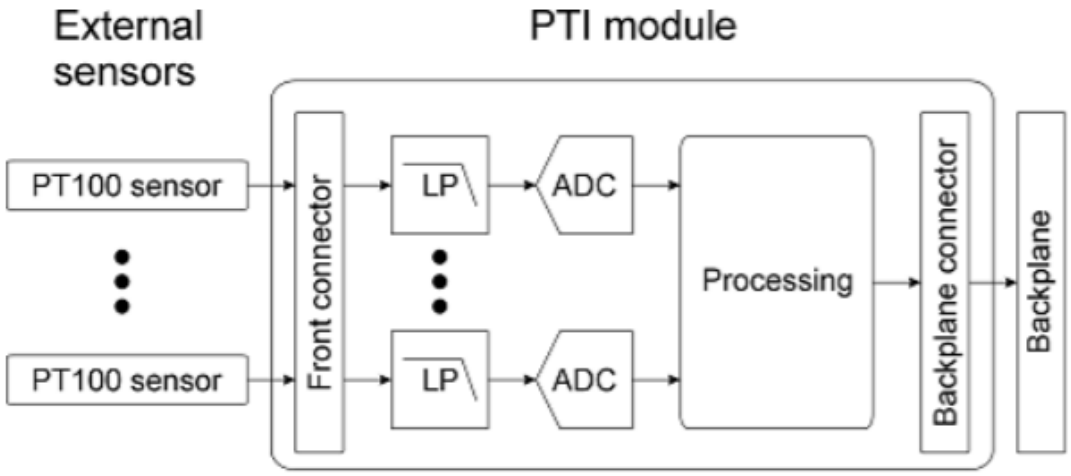


Figure 2. Functionality diagram of PTI module from technical manual PTI2037A [2, p. 8]

Table 1 shows the message content PTI module sends to CPU. The PT simulator can impact on 12 bytes of channel temperature and cause two types of channel errors (next six bytes).

Table 1. Data message field [2]

Bytes	Field	Range	Notes
12	6 channel temperatures	[-110, 325] °C	Resolution is 0.01 °C (16-bit signed value)
6	6 channel diagnostic bytes	[0x00, 0xFF]	Bits explained in table below
1	Heartbeat	[0x00, 0xFF]	Increases for every data reply, wraps to 0x00 after 0xFF
1	Module temperature in °C	[-55 °C, 125] °C	Resolution is 1 °C (8-bit signed value)

1	Module status	0 or 1	0 = OK, 1 = module internal failure
1	Reset status	0 or 1	1 for first data query after reset (boot-up), 0 for subsequent queries
16	MD5	Message digest	Calculated from all bytes above

Measurement accuracy of PTI modules is ±1 °C with resolution of 0.01 °C. Connections of the sensors are shown in Figure 3. Sensor simulator can provide only two-wire type of connection, where resistance is read between inputs SENSE_P and SENSE_N. Each channel of the PTI simulator should be permanently associated with channel number in PTI module.

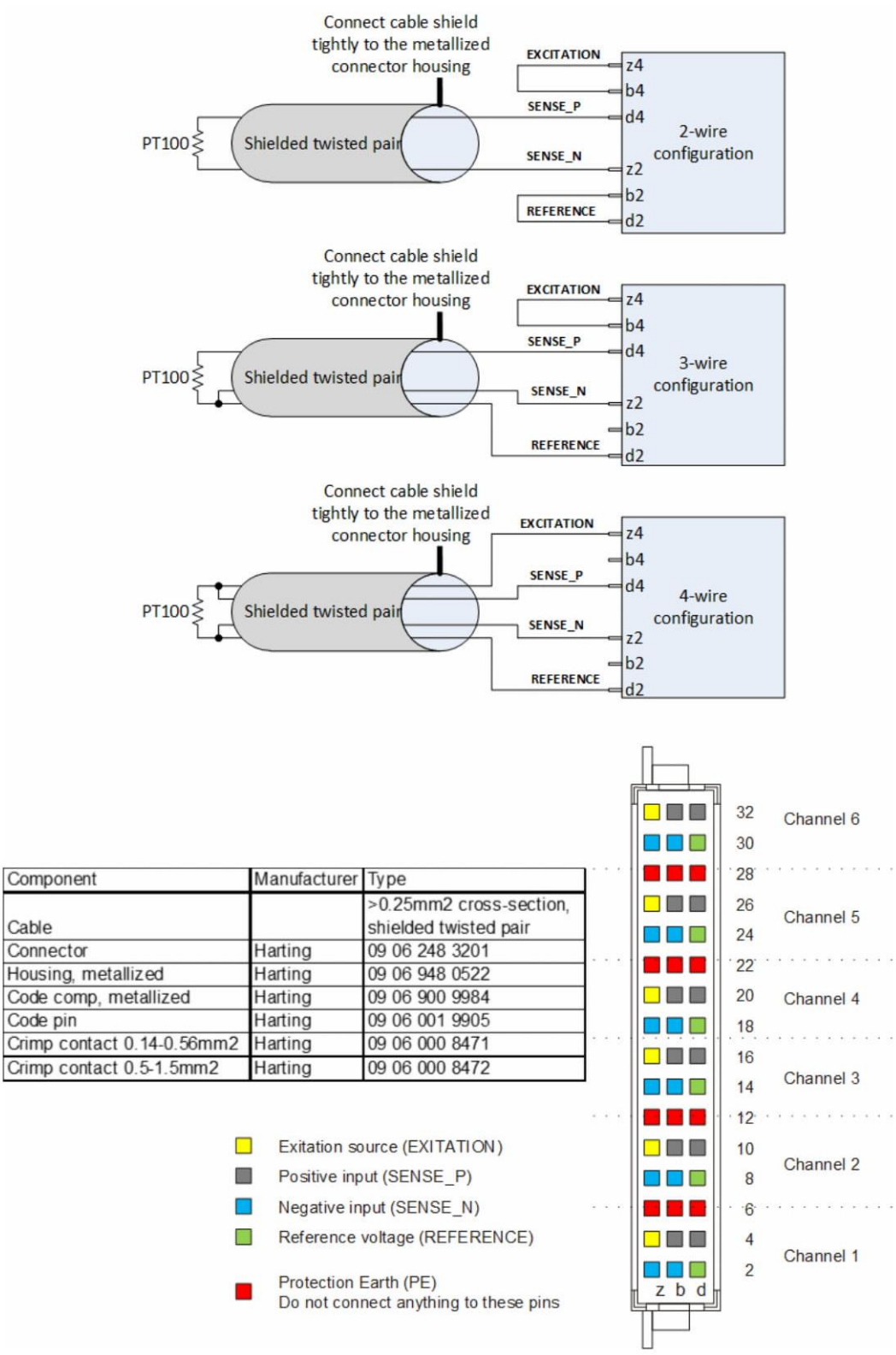


Figure 3. Connecting I/O cable to input pins from PTI2037A technical manual [2, p. 14]

2.2 Theory of operation of the PT100/PT1000 sensors

PT sensors belong to resistance thermometers with positive resistance coefficient. Increasing temperature leads to increasing resistivity of thermometer. The number in the name of sensor means the resistivity at 0 °C:

- PT100 (0 °C) = 100 Ω
- PT1000 (0 °C) = 1000 Ω

PT sensor is resistance thermometer. The construction is shown in Figure 4.

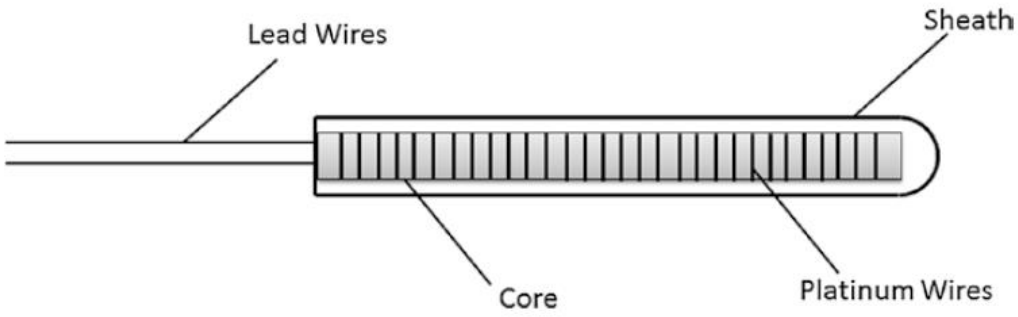


Figure 4. An example of resistance thermometer construction from Thermal Sensors [3, p. 7]

The typical temperature range for PT sensors is [-250, 600] °C. Relationships between temperature and resistance above 0 °C is described by quadratic equation (1), and below 0 °C they are described as quartic equation (2), known as Callendar-Van Dusen equations [4, p. 598]:

$$R_T = R_0[1 + AT + BT^2], (0\,°C \leq T < 850\,°C) \quad (1)$$

$$R_T = R_0[1 + AT + BT^2 + CT^3(T - 100)], (-200\,°C < T < 0\,°C) \quad (2)$$

Where T is the temperature in °C
R_0 is the resistance with 0 °C
$A = 3.9083 \times 10^{-3}\,°C^{-1}$
$B = -5.775 \times 10^{-7}\,°C^{-2}$
$C = -4 \times 10^{-12}\,°C^{-4}$

Because the temperature range of PTI module started from -110 °C, conversion tables with relationships were calculated (TempVSResPT100.txt, TempVSResPT1000.txt). The calculated results are rounded to third digit after decimal separator for temperature. Resistance values are non-linear and rounded to fifth digit. The tables are used in calibration process. Appendix 1 has the listing script, creating of conversion tables.

Conversion tables have the wider range [-120, 335] °C for testing also boundary values of PTI module. Table 2 represents the calculated values for maximum and minimum resistance and temperature used in the project.

Table 2. Minimum and maximum values for PT sensors

	Min values		**Max values**	
	°C	Ω	°C	Ω
PT100	-120	52.102	335	224.467
PT1000	-120	521.025	335	2244.671

The resistance values of PT1000 sensor are 10 times larger than values of PT100 sensor for the whole scale. The sensitivity of PT100 sensor is approximately 0.358 Ω/°C, and for PT1000 it is 3.58 Ω/°C.

2.3 Digitally controlled potentiometer

Digital potentiometers are digitally controlled resistors or RDAC (resistive digital-to-analog converters). They can be used as replacement of mechanical potentiometers. Range of resistance values is from 1 kΩ to 1 MΩ. Figure 5 shows the logic of integrated circuit inside a digital potentiometer: RDAC register sets the steps R_S, shown on the output. The system is equal to mechanical potentiometer: the full value is AB = AW + BW. Therefore, if wiper position is changed to (N+1), the AW will decrease to one R_S and BW will increase to one R_S.

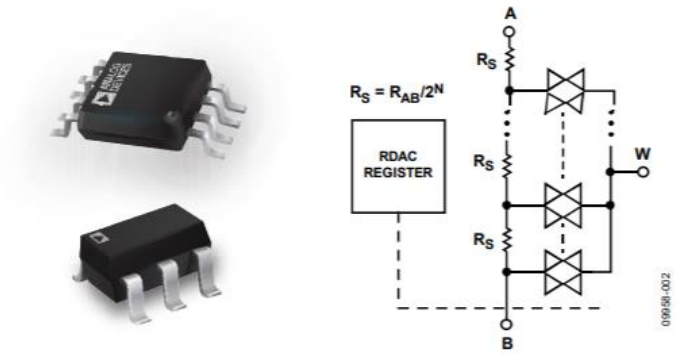


Figure 5. Digital potentiometer from Usach [5, p. 1]

For the following example digital potentiometer with 8-bit RDAC register is used. Therefore, it is 2^8 = 256 wiper positions.

When RDAC has the minimum value (0x00), wiper W is standing on the B-point, and output resistances BW = R_W, AW = AB + R_W. If the supply voltage is 5V, typical wiper resistance is 75 Ω. When code has the maximum value (0xFF for 256-position digital potentiometer), the output resistance BW = (255/256)*R_{AB} + R_W, wiper is on the position A: AW = R_W.

Wiper resistance is the smallest possible value of the output. As shown in table 2, R_W should be lower or equal 52.102 Ω for PT100, and lower than or equal to 521.025 Ω for PT1000 to cover the whole temperature range. Digital potentiometer with three output pins: A, B, W – can be used in rheostat mode (Figure 6 shows the rheostat mode configuration), when only two of three pins are used. Pins B and W are connected to the output, and pin A is not connected.

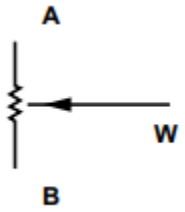


Figure 6. Rheostat mode configuration

The used potentiometer is AD5254. With 5V power supply the wiper resistance of the digital potentiometer in rheostat mode is in range [75, 130] Ω [6, p. 3]. In one chip there are four integrated circuits, each can be presented as a variable resistor with individual control via I²C bus. Connected in parallel, they have wiper resistance approximately four times smaller.

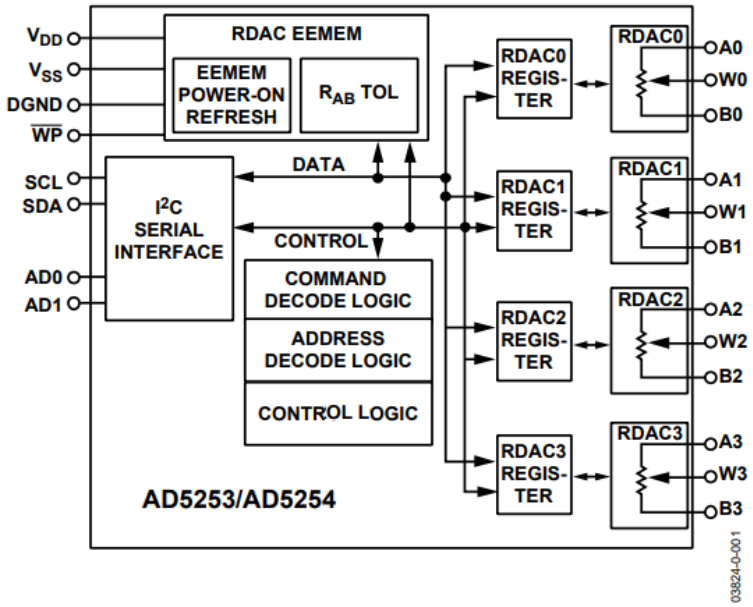


Figure 7. Functional block diagram of AD5254 from AD5254 datasheet [6, p. 1]

Connections of the AD5254 are shown in Figure 7. SCL and SDA are connection pins of I²C bus. $\overline{WP}$ is inverted input for the writing protection, to allow the writing $\overline{WP}$ should be connected to V_{DD}. V_{SS} is negative power supply. If single-supply power range is used, V_{DD} should be in range [2.7, 5.5] V and V_{SS} = 0 V. DGND is digital ground. Ax, Bx and Wx are outputs of the potentiometers. AD0 and AD1 are the address pins for slave device address (not RDACs). Normal I²C addresses have seven bits. The most significant five bits of the digital potentiometer are static: 01011, and the rest two are values of AD1 and AD0. There are four possible addresses of digital potentiometers, as shown in Table 3.

Table 3. Slave addresses

Fixed part	AD1	AD0	Explanation
01011	0	0	AD1 is grounded, AD0 is grounded
01011	0	1	AD1 is grounded, AD0 is connected to V_{DD}
01011	1	0	AD1 is connected to V_{DD}, AD0 is grounded
01011	1	1	AD1 is connected to V_{DD}, AD0 is connected to V_{DD}

Figure 8 shows the data structure of the digital potentiometer writing command.

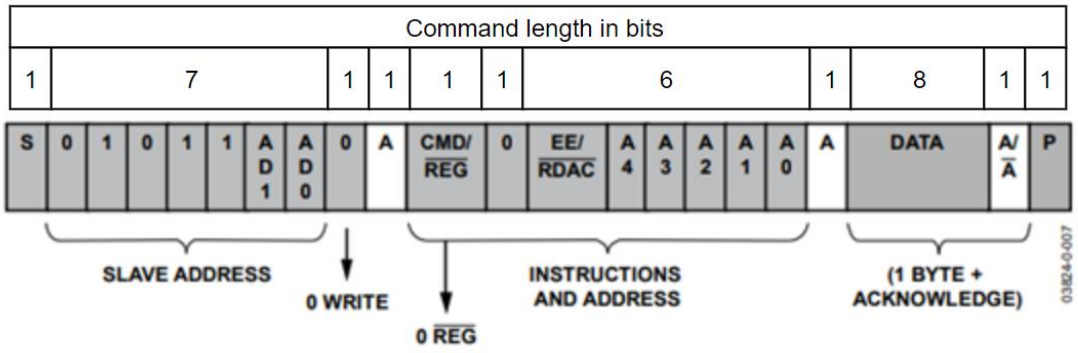


Figure 8. Single write command from datasheet AD5253_5254 [6, p. 15]

- S = start condition
- P = stop condition
- A = acknowledge (SDA low)
- $\overline{A}$ = not acknowledge (SDA high)
- AD1, AD0 = address pins for device selecting, hardware connected
- R/W= read enable bit at logic high; write enable bit at logic low
- CMD/REG = command enable bit at logic high; register access bit at logic low
- EE/RDAC = EEMEM register at logic high; RDAC register at logic low
- A4, A3, A2, A1, A0 = RDAC/EEMEM register addresses

EEMEM registers have predefined value 128 (the midscale value), and there is no need to change it. Only RDAC registers are used in this project.

Table 4. Addresses for RDAC writing ($\mathrm{R}/\overline{\mathrm{W}} = 0, \mathrm{CMD}/\overline{\mathrm{REG}} = 0, \mathrm{EE}/\overline{\mathrm{RDAC}} = 0$) from datasheet [6, p. 15]

A4	A3	A2	A1	A0	RDAC
0	0	0	0	0	RDAC0
0	0	0	0	1	RDAC1
0	0	0	1	0	RDAC2
0	0	0	1	1	RDAC3

Table 4 shows the addresses of RDAC registers. The values are used in writing values of digital potentiometers.

3 Simulation method and used materials

Functional block diagram of the PT simulator is shown in Figure 9. The testing system consists of:

- PC, controlling the PT simulator,
- RPI – Raspberry Pi 3B+ (in headless mode) and PT simulator board,
- System under test (SUT): PTI module.

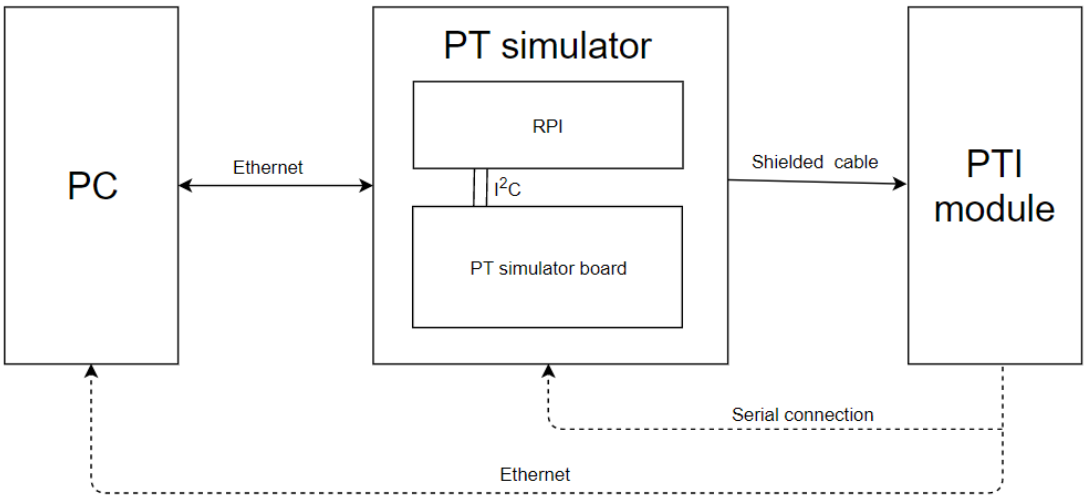


Figure 9. Structure of the PT simulator and its connections

PT simulator should cover all possible resistance values of PT sensors in the chosen range with acceptable resolution of 0.35 °C. The difference between PT100 and PT1000 simulators is the only value of digital potentiometer. It will be shown in Section 3.1.

There is no additional powered connection between PTI module and simulator. Only passive resistive output of the simulator and connective wires. Power supplies should be independent. Simulator board is controlled and powered by Raspberry Pi 3B+ with Raspbian Buster (kernel version 4.19). Control software is written by Python 3.

Reading from PTI module is done via serial port with serial-to-USB connector, because RPI has no serial port.

3.1 Choosing the values of digitally controlled potentiometers

Digital potentiometer is expected to produce stable resistance. The output is passive and has no influence on PTI module. As shown in the table 2, the desired range is [52.1, 224.5] Ω for PT100 and [521, 2245] Ω for PT1000. Chosen digital potentiometer AD5254 has several different resistances: 1 kΩ and 10 kΩ, and it has four circuits in one chip that can be connected together. There are 256 possible values for each circuit that can be set independently. Wiper resistance (minimum possible resistance), when V_{DD} = 5V, R_W = 75-130 Ω for both 1 kΩ and 10 kΩ versions.

The range of temperatures are from -120 °C to 335 °C. Because inner circuits in the chip can be set independently, the maximum possible number of taps is shown in equation 3:

$$T_a = (2^8)^4 = 2^{32} \quad (3)$$

Many of these steps can be reduced, because the difference between some of them is approximately 0 Ω.

The following connection scheme has been chosen: four resistors are connected in parallel, and each pair has the same value of tap, as shown in Figure 10. All terminals B and all terminals W are connected together. Terminals A are not connected, as shown in Figure 6, rheostat mode configuration.

Connected in parallel, output R_{WB} is in a range between the minimum wiper resistance, divided by four, and maximum value, divided by four. Tolerance of AD5254 is 30% [6].

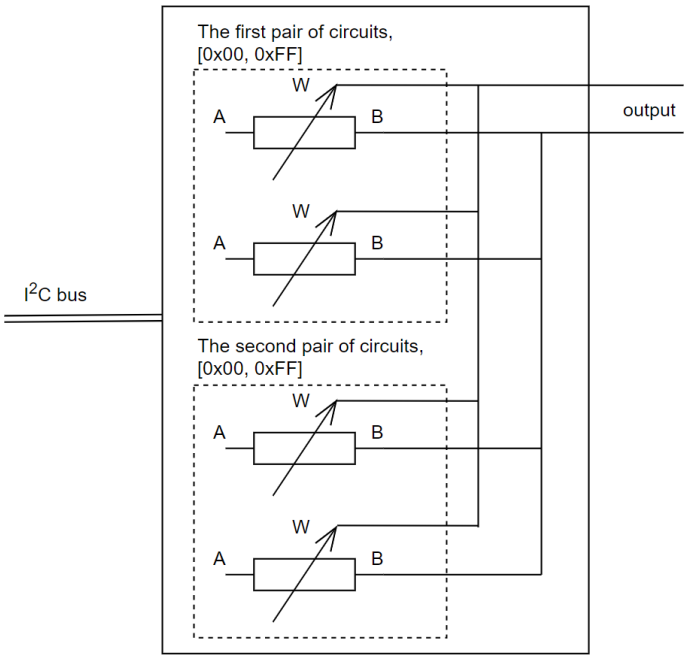


Figure 10. Connection scheme of AD5254

Table 5 represents the possible range of maximum and minimum values of the digital potentiometers, when all the channels get corresponding code, calculated by equations 4 – 6. Equation 4 is the same for 1 kΩ and 10 kΩ versions of AD5254, equation 5 is for 1 kΩ version, equation 6 is for 10 kΩ version.

$$R_{WB(0x00)} = \frac{1}{\frac{1}{R1(0x0)}+\frac{1}{R2(0x0)}+\frac{1}{R3(0x0)}+\frac{1}{R3(0x0)}}, \quad (4)$$

where $R_{(0x0)}$ in range $(75, 130)\ \Omega$

$$R_{WB(0xFF)} = \frac{1}{\frac{1}{R1(0xF)}+\frac{1}{R2(0xF)}+\frac{1}{R3(0xF)}+\frac{1}{R3(0xF)}}, \quad (5)$$

where $R_{(0xF)}$ in range $(700, 1300)\ \Omega$

$$R_{WB(0xFF)} = \frac{1}{\frac{1}{R1(0xF)}+\frac{1}{R2(0xF)}+\frac{1}{R3(0xF)}+\frac{1}{R3(0xF)}}, \quad (6)$$

where $R_{(0xF)}$ in range $(7000, 13000)\ \Omega$

Table 5. Calculated values with the lowest and highest codes

	AD5254 1 kΩ			AD5254 10 kΩ		
Code	**-30%**	**Nominal**	**+ 30%**	**-30%**	**Nominal**	**+ 30%**
0x00, Ω	13.125	18.75	32.5	13.125	18.75	32.5
0xFF, Ω	170	250	325	1750	2500	3250
Step size, Ω	Average: [0.109, 0.115] Max: [0.356, 0.654]			Average: [1,092, 1.148] Max: [3.557, 5.52]		

Measured values from the PT simulator board:

- Channel 1: R_{0x00} = 16.23 Ω, R_{0xFF} = 268.10 Ω
- Channel 2: R_{0x00} = 16.20 Ω, R_{0xFF} = 268.54 Ω
- Channel 3: R_{0x00} = 16.34 Ω, R_{0xFF} = 267.51 Ω
- Channel 4: R_{0x00} = 16.59 Ω, R_{0xFF} = 265.08 Ω
- Channel 5: R_{0x00} = 16.27 Ω, R_{0xFF} = 268.25 Ω
- Channel 6: R_{0x00} = 16.01 Ω, R_{0xFF} = 266.82 Ω

If the digital potentiometers have values 0x00, the corresponding range is lower than minimum possible value, readable by PTI module (as shown in table 2, it is 56.186 Ω for PTI2037A and 561.864 Ω for PTI3593A).

Equation 7 from the AD5254 datasheet [6, p. 3] is used in calculation of nominal step values:

$$R_{WB(D)} = \frac{D}{256} \times R_{AB} + 75\Omega \quad (7)$$

Where:

- D is the tap value of the RDAC, in range [0, 255],
- 75 Ω is the nominal wiper resistance from the datasheet,
- RAB = 1 kΩ or 10 kΩ for PT100/PT1000 simulation respectively.

Noise characteristics of the digital potentiometers are also important. Resistor noise voltage with R_{WB} (midscale value) = 500 Ω, f = 1 kHz (thermal noise only) is 3 nV/√Hz (from datasheet, 1 kΩ version [6, p. 4]). R_{WB} (midscale value) = 5 kΩ, f = 1 kHz (thermal noise only) is 9 nV/√Hz (from datasheet, 10 kΩ version [6, p. 6]).

Values of digital potentiometers are corresponding to thermal noise of ordinary resistor: 2.869 nV/√Hz for 500 Ω and 9.072 nV/√Hz for 5000 Ω (from table of resistance noise, [7, p. 240]).

Thermal noise for PT100 is approximately $R = \frac{3\ nV/\sqrt{Hz}}{1\ mA} = 3\mu\Omega/\sqrt{Hz}$, and in degrees $T = \frac{3\ \mu\Omega/\sqrt{Hz}}{0.358\ \Omega/°C} = 8.38 \times 10^{-6}\ °C/\sqrt{Hz}$.

Thermal noise for PT1000 is $R = \frac{9\ nV/\sqrt{Hz}}{1\ mA} = 9\mu\Omega/\sqrt{Hz}$, and in degrees $T = \frac{9\ \mu\Omega/\sqrt{Hz}}{0.358\ \Omega/°C} = 25.14 \times 10^{-6}\ °C/\sqrt{Hz}$.

Expected noise from digital potentiometer should be close to noise of the ordinary potentiometers used in the actual testing setup.

3.2 Simulator board

3.2.1 Schematic

The actual version of the simulator board consists of six digital potentiometers AD5254 (1 kΩ version) and two address translators LTC4316. This is PT100 simulator that will be used with PTI2037A module. Appendix 2 shows the full schematics.

Address translators have two different XOR sequences: 000 0100 and 000 0000. The first potentiometer translates the voltage from 3.3 V to 5 V and changes one bit in a fixed part of digital potentiometer's address. The second translator changes the voltage level.

Table 6 shows the addresses on the PT simulator board and corresponding channels. Output terminal has the connection in ascending order, from 0x28 to 0x2f.

Table 6. Table of connection between addresses and channel numbers

Channel number	**Channel address**	
	hex	bin
1	0x28	0101000
2	0x2b	010 1011

3	0x2c	0101100
4	0x2d	0101101
5	0x2e	0101110
6	0x2f	0101111

Figure 11 shows the connection of the address translator. Input I^2C bus from RPI (SCL and SDA, 3.3 V) connects to the corresponding input puns SCLIN and SDAIN. After the translation with XOR sequence 0000100, the output I^2C bus from the address translator (SCL0 and SDA0, 5V, [8, p. 1]) from the output pins SCLOUT and SDAOUT goes to digital potentiometers. After the translation their addresses are in range 01010XX.

There is a possibility to disable address translator: corresponding connectors (X3 and X4) can be closed by a jumper wire. Normally they should be open. When this connector is closed, address translator and all the following digital potentiometers are disabled.

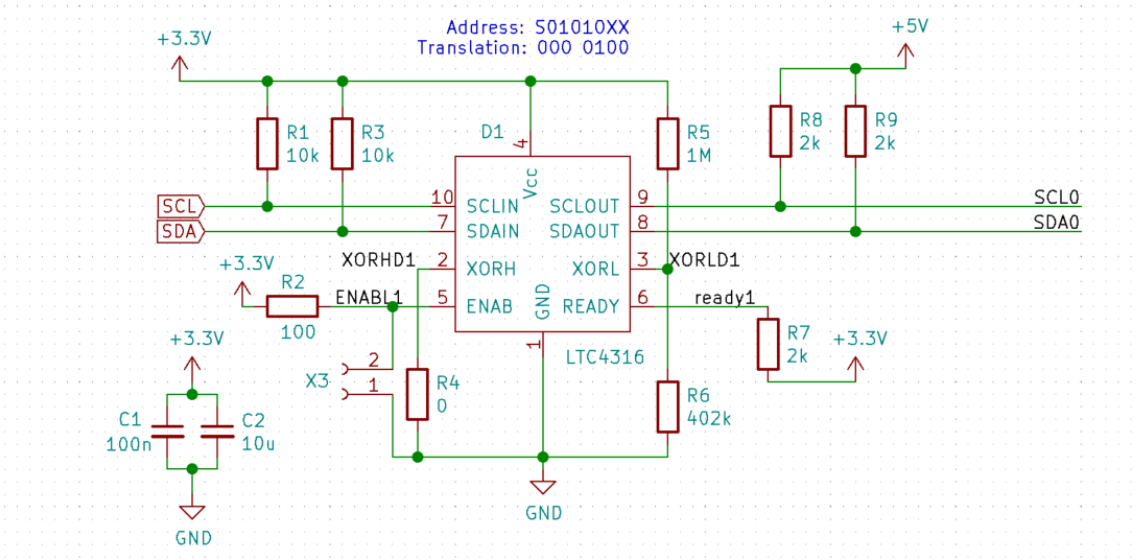


Figure 11. Address translator

Figure 12 shows the connection of digital potentiometer. Bus input is connected with address translator output. All resistors are connected in parallel, and two 0 Ω resistors (R18 and R19) are added for debugging purposes. When they are removed from the board, digital potentiometer will be physically disconnected from the output.

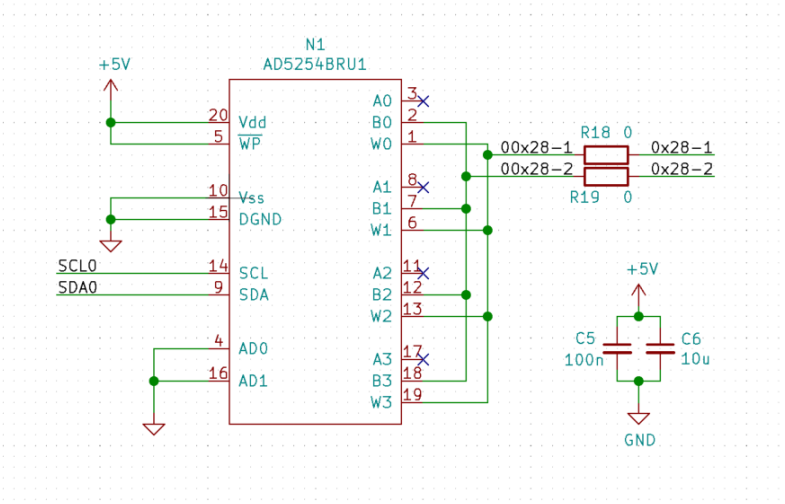


Figure 12. Digital potentiometer

All digital potentiometers and address translators have bypass capacitors (C1, C2 or C5, C6) to prevent unwanted power supply noise. Bill of materials, used in PT simulator, is shown in Appendix 3.

3.2.2 PCB design

The board is constructed as Raspberry hat. Non-plated through holes in the corners of the outline are placed for mounting PT simulator to RPI. Board to board connector X1 provides the shortest connection from PRI to PT simulator for reducing the noise impact.

Figure 13 shows the actual design (without filled copper areas). Each layer with filled copper areas and the larger version of Figure 13 can be found in Appendix 4.

All elements except connector X1 are mounted on top layer. Channel numeration started from the first pin of the output connector X5.

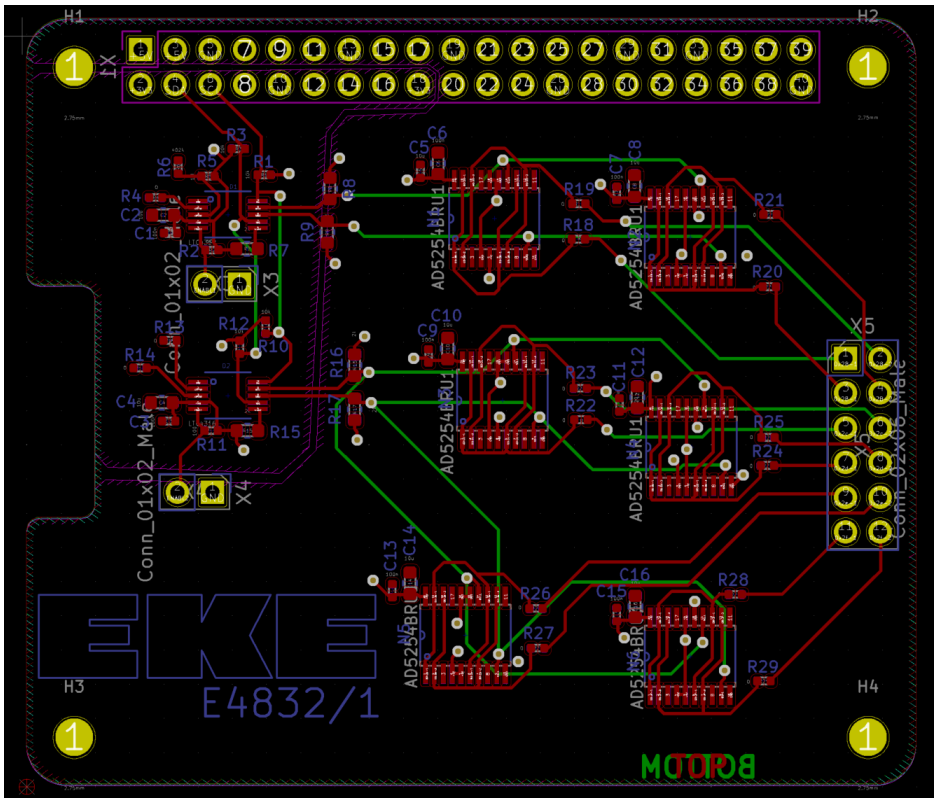


Figure 13. Top view of PT simulator's PCB design

Two inner layers are GroundPlane and PowerPlane. There are no blind or burried vias. PT simulator is connected to RPI, then RPI should be powered. Connection between PC and RPI is done by Ethernet cable.

4 Simulator software

PT simulator software consists of two main part: temperature input and calibration. Digital potentiometers have a wide tolerance, up to 30%, therefore in calibration process the exact output of the potentiometers and corresponding values of RDACs (taps) are written in calibration tables. After calibration PT simulator is ready to use. System configuration is explained in Appendix 5.

Available calibration parameters and the calibration process are described in section 4.1. Section 4.2 describes the actual input module and two additional testing modules used for the board evaluation.

4.1 Calibration of the PT simulator

Calibration process creates calibration tables with actual taps and corresponding output resistance. The calibration table is individual to every board and channel. The process includes digital multimeter (DMM) connected via Ethernet.

Test leads should be connected to the output pins. PT simulator should be disconnected from PTI module before start. Figure 14 shows the process of calibration. Appendix 6 has the full listing of the calibration script.

The channel name is the number from 1 to 6. For example, channel 1 has address “0x28” and calibration table “0x28.txt”.

After the input of channel name, the corresponding address of calibrated channel is detected by sending minimum and maximum codes to the channels, one by one. If DMM detects the resistance changes, the connected channel is detected. If DMM cannot detect any changes, the channel is not detected, and calibration process is aborted.

Then the calibration script asks for input steps in rows and columns. Digital potentiometer taps can be described as table, where the values of the first pair of the circuits are the rows, and the values of the second pair are the columns: [0, FF] x [0, FF] values.

Setting 1/1 (all values in columns, all values in rows) gives the all possible values: 65 535. Setting 1/5 means that all values in columns are used, but values in the rows are done with step 5. Therefore, only 13 107 possible values are checked, in five times less that the whole range.

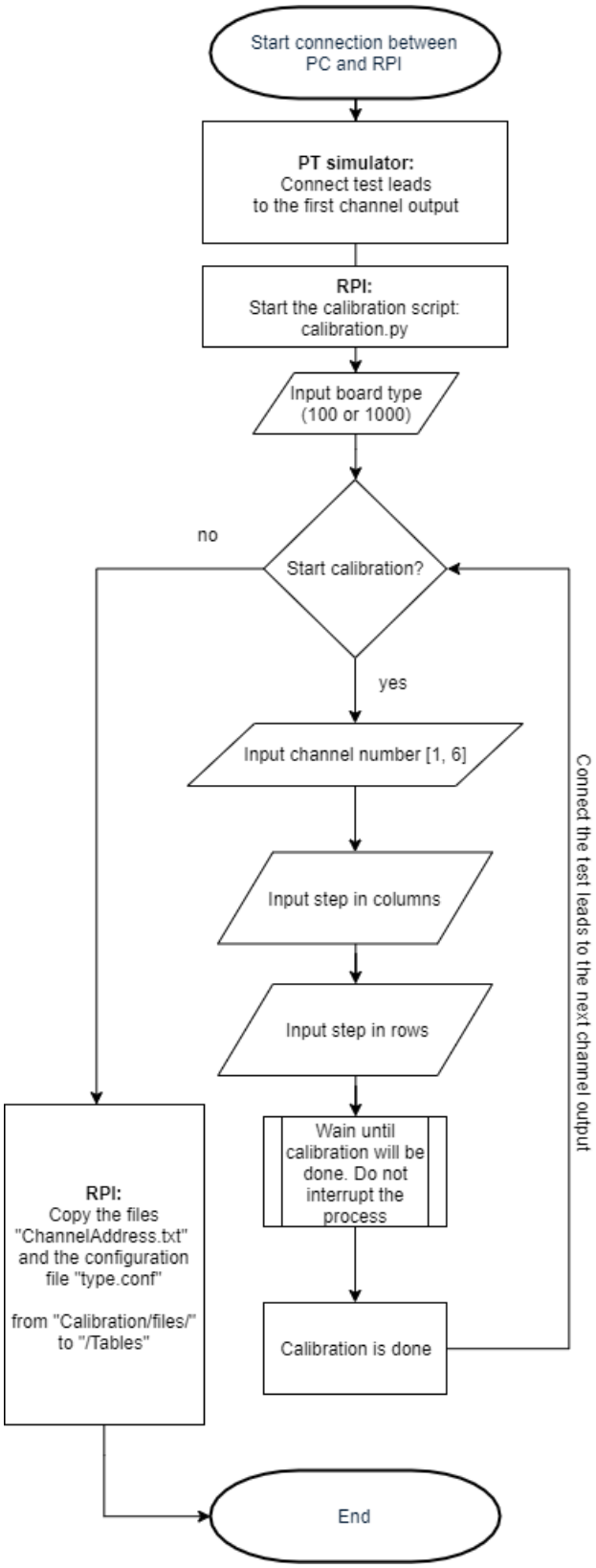


Figure 14. Calibration workflow

Setting 5/1 means that every fifth value in column is used, but the all values in rows. The result can slightly change between 5/1 and 1/5, because the potentiometers inside one AD5254 chip has different values due to the tolerances. Other settings (5/5, 10/1, 1/10, 5/10, 10/5, 10/10) have even shorter covered range.

After setting the steps and channel name, the calibration process has begun. Script is sending taps for the first and second pairs of potentiometers with chosen step and then read back the resistance from DMM. Then resistance and steps are written to the file "ChannelAddress_raw" on RPI. Next, the data is sorted. Too short steps are removed, if they are shorter than 0.0001 Ω or approximately 0.003 degree. Existing steps are translated from resistance to temperature. Then data are written to the calibration table with name "ChannelAddress.txt".

After all the channels will be calibrated, the calibration tables should be copied from the folder "pt-simulator/Calibration/files" to the folder "pt-simulator/Tables". Then PT simulator is ready to use.

Table 7. Values with different calibration input, values in Ohm

Channel 2, settings	Tap values covered	Time, minutes	Max step, °C	Avg step, °C
1/1	65 535 (full range)	82	0.191	0.019
1/5	13 107	18.0	0.469	0.053
5/1	13 107	16.8	0.607	0.053
1/10	6 553	9.6	0.799	0.096
10/1	6 553	8.5	0.697	0.096
5/10	1 310	2.1	3.658	0.436
10/5	1 310	2.0	3.326	0.433
5/5	2 621	4.3	2.318	0.222
10/10	655	1.2	6.429	0.868

Table 7 shows the time consumption, maximum step over range and average step of different calibration settings. Resolution is not equal the average step but can be described as approximately the same value. Step start to grow to the end of the temperature scale, after 200 °C.

The difference between sent value and the closest possible value in calibration table is the source of error. Small step and wide range of the calibration table makes the difference between sent and closest possible values shorter.

4.2 Temperature input with PT simulator software

The main purpose of the PT simulator is the temperature input. Device is ready to use after the calibration has been done to all channels.

After the connection the outputs of PT simulator to PTI module temperature can be set. The name of the script is “ManualInput.py”. Steps of usage flow is shown on the Figure 15. All channels will be changed at the same time, but it can be changed with command “c”. Then only chosen channels will be set, and other will keep the last value.

Temperature can be set by input float number. If the input is incorrect, script will show prompt again without changing channel values. The actual temperature measured by PTI is shown after every successful attempt and written into the file with the name “date_session.txt”. The quit command “q” finished the script.

The full listing of “ManualInput.py” can be found in Appendix 7. Table 8 consists of used functions.

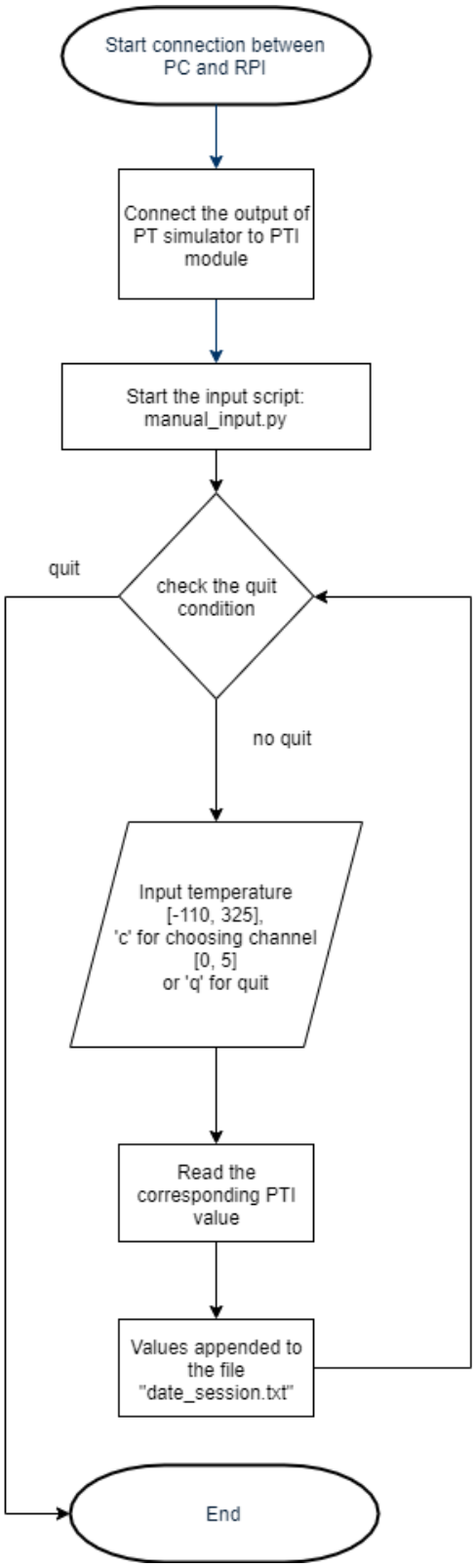


Figure 15. Usage workflow

Table 8. Modules and functions

Function	Input	Output	Description
Module DataInput.py (full listing in Appendix 6)			
TemperatureWriting	temp = 0 temperature in degrees, float channels = None list of the channels [1-6]	List of the values in format "Channel N, input temperature, closest calibration value, time or "Channel N is not available! time	The module has the only writing function, combining all additional modules. It receives the temperature and the list of the channels and return fill information about written temperature or information about unavailability of the channel
Module I2CWriting.py			
I2CWriting	address is digital potentiometer address in hex or int tap1, tap2 are numbers in range (0, 255)	OK: if writing was successful NOK: if writing was unsuccessful or there is no input	The function is used for the writing in available channels.
Module TempVSRes.py			
TemperatureToResistance	Temperature, °C, float	Resistance, Ω, float	The function receives temperature and returns corresponding resistance, following the conversion table
ResistanceToTemperature	Resistance, Ω, float	Temperature, °C, float	The function receives resistance and returns corresponding temperature, following the conversion table
Module FileReading.py			
ReadFile1	filename	Two lists: integers and strings. NOK, if file has not found	All functions are used for reading the conversion and calibration tables
ReadFile2	filename	Two lists of float numbers. NOK, if file has not found	
ReadFile3	filename	Three lists: two in integer, one in float. NOK, if file has not found	
Module SearchNearest.py			
SearchNearest	value_list, value	Index, int. NOK, if input was invalid	Function searches nearest value in the list and return its index

5 Results

PT simulator for PT100 sensor has been chosen for evaluation. The reason for that is the sensitivity of PT100 sensor is 10 times higher than PT1000. Another reason is the PTI2037A modules are in use in the actual project and should be tested as soon as possible, and PT simulator will be helpful.

Measurement setup consists of:

- PT100 simulator board as a hat of RPI and Raspberry power supply,
- PTI2097A with CPU module in rack,
- Digital multimeter DM3068,
- Connection cable between PT simulator and PTI module,
- Ethernet cable between laptop and PT simulator.

PT100 simulator is calibrated for all channels with 5/1 accuracy (Table 7).

First attempt wasn't successful because of huge instability of the output: up to 10 degrees. Connection cable was changed from unshielded twisted pair to shielded twisted pair and shield was grounded for the both ends. After several improvements, which will be explained further in section "Discussion", the result has become more stable and reliable, as shown in figures 17-21.

Evaluation method:

- send temperature with 5 °C step,
- measure the output by DMM,
- measure the output by PTI module.

Input temperature [-120, 325] °C has been read first on digital multimeter, 10 samples with 1 second delay. Noise peak-to-peak amplitude is the difference between the lowest and the highest sample values.

Temperature difference is the difference between sent value (10 °C) and the average value, read on the DMM or PTI (9.78 °C and 19.96 °C). All the values are shown in Table 9.

Table 9. Temperature difference and noise amplitude over the channels

Maximum values, °C	**Channel 1**	**Channel 2**	**Channel 3**	**Channel 4**	**Channel 5**	**Channel 6**
Noise p/p amplitude, DMM	0.03	0.011	0.018	0.012	0.017	0.023
Temperature difference, DMM	0.12	0.249	0.262	0.418	0.753	0.038
Noise p/p amplitude, PTI	0.17	0.06	0.08	0.06	0.06	0.08
Temperature difference, PTI	1.005	1.24	1.145	1.91	2.17	1.53

The following Figures 17-20 show the distribution of noise and difference over temperature values. Noise and difference for DMM look relatively random (Figures 17-18) the same as noise for PTO module (Figure 19). Temperature difference for PTI module clearly increases on by the end of temperature scale, up to 2.17 °C difference. The possible reasons for that behavior are the influence of the power supply or self-heating of the board.

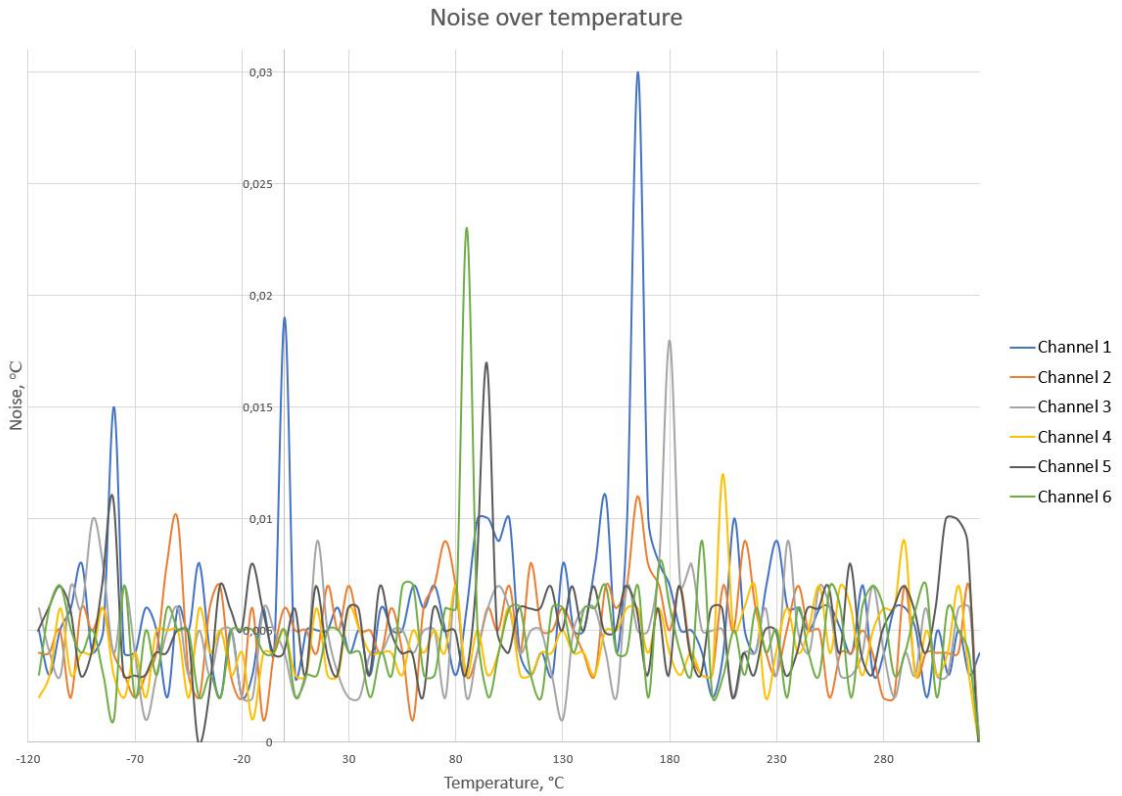


Figure 16. Noise over temperature values (DMM)

Figure 16 shows the noise distribution for all channels, reading on digital multimeter. The average noise amplitude is in range [0, 0.01] °C. Every channel has the peak value. The smallest one is 0.011 °C on the channel 2. The highest one is 0.03 °C on the channel 1.

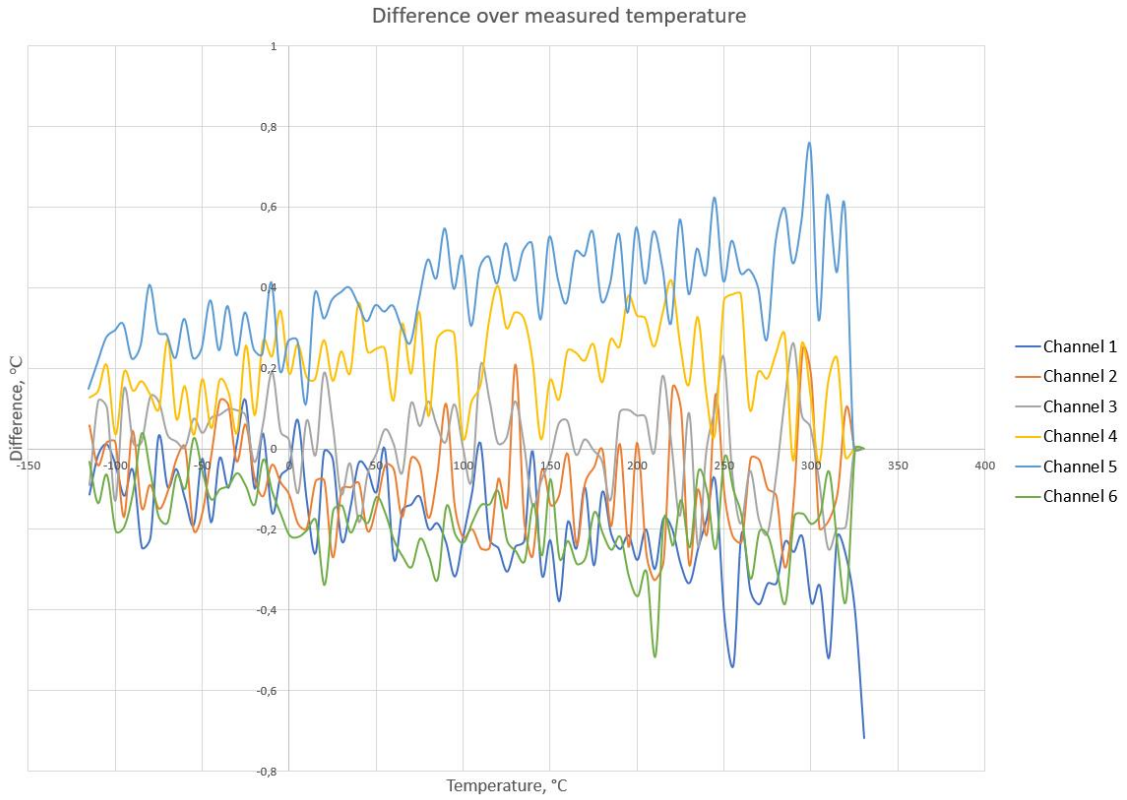


Figure 17. Temperature difference over temperature values (DMM)

Figure 17 shows the temperature difference between sent and received values on DMM. Most of the difference values lays in range [-0.4, 0.6] °C. Closer to the end of the scale, after 200 °C, all channels tend to show the difference peaks: [-0.718, 0.753] °C.

The values from calibration table can differ from the actual values, if the ambient temperature is changing. For AD5254, 1 kΩ version, resistance temperature coefficient is 650 ppm/°C, or 0.65 Ω/°C. It is approximately 1.7 °C (measured) / °C (ambient).

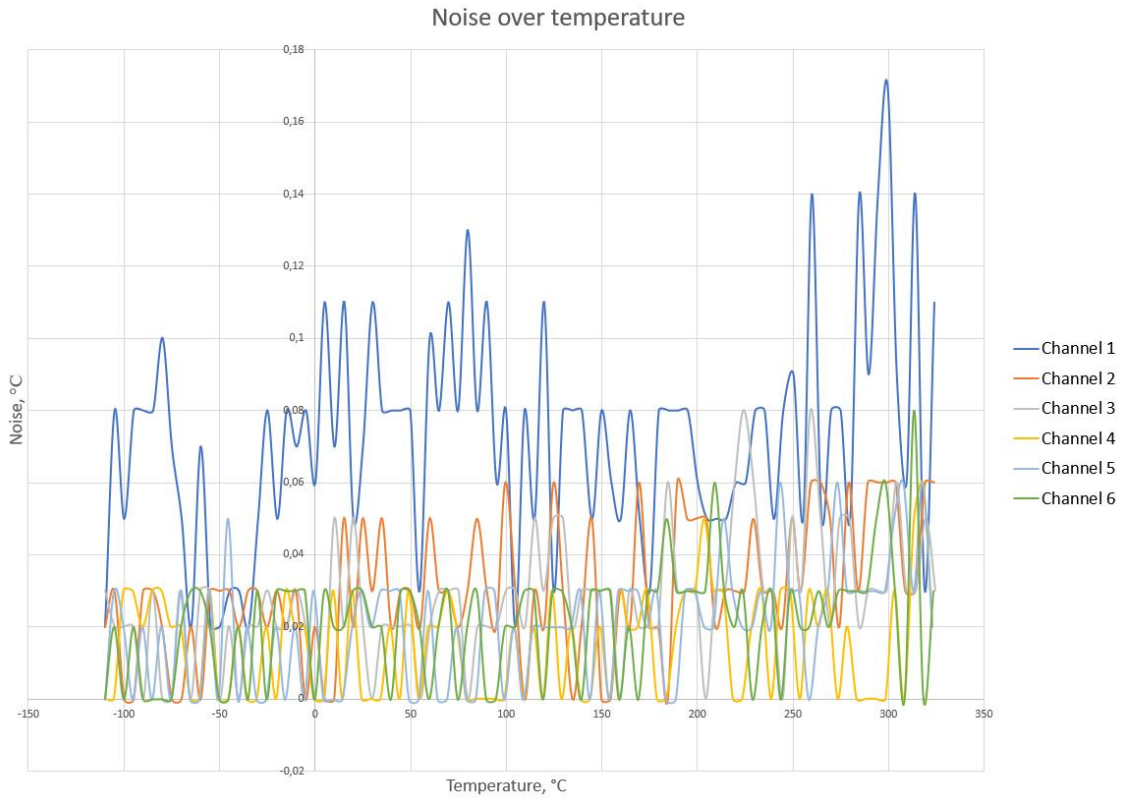


Figure 18. Noise over temperature values (PTI)

For the PTI modules the noise amplitude is higher, as shown in Figure 18. Values for all channels except channel 1 are in range [0, 0.08] °C. For channel 1 the peak value is 0.17 °C with range [0.02, 0.14] °C. The possible reason is the position of channel's digital potentiometer is close to the noise sources, such as USB ports and GPIO pins (Figure 16).

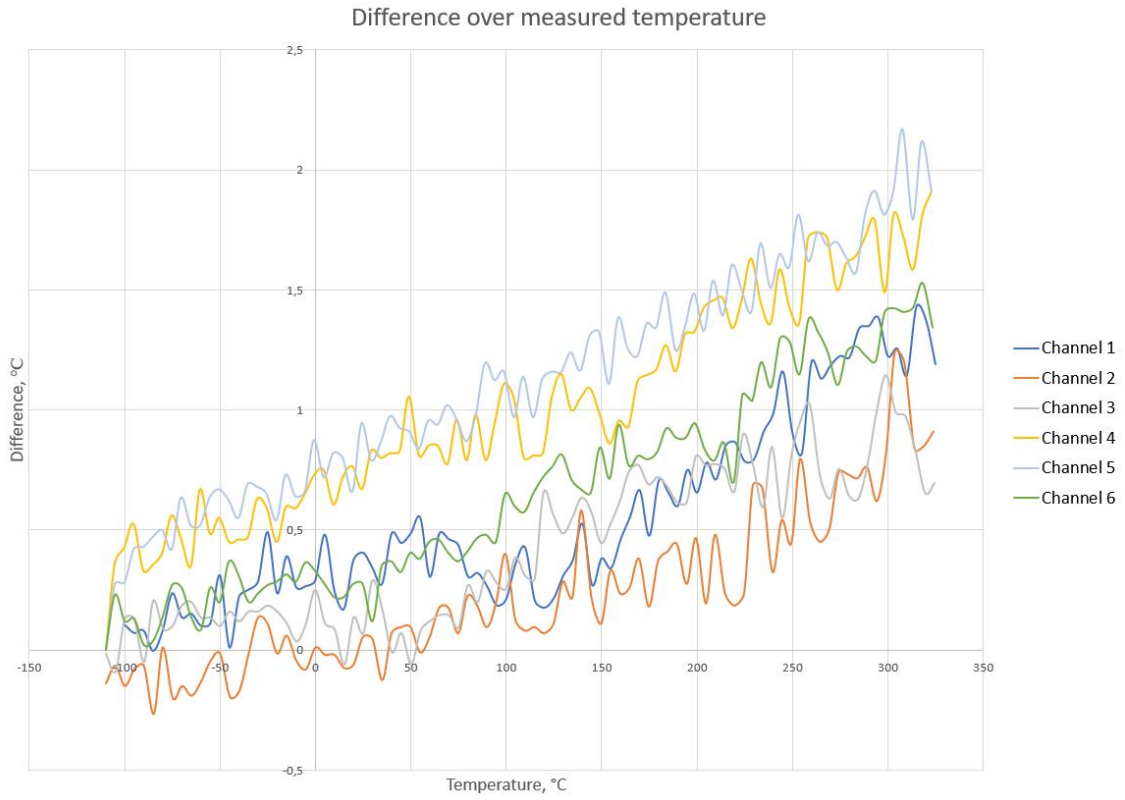


Figure 19. Temperature difference over temperature values (PTI)

Difference over temperature, shown in Figure 19, looks very different from DMM difference measurements (Figure 18). All the channels tend to increase the difference notably over the temperature range.

The changes are significant, up to 2.17 °C to the end of scale: sending value is 310 °C, but measuring value on PTI module is 312,17 °C.

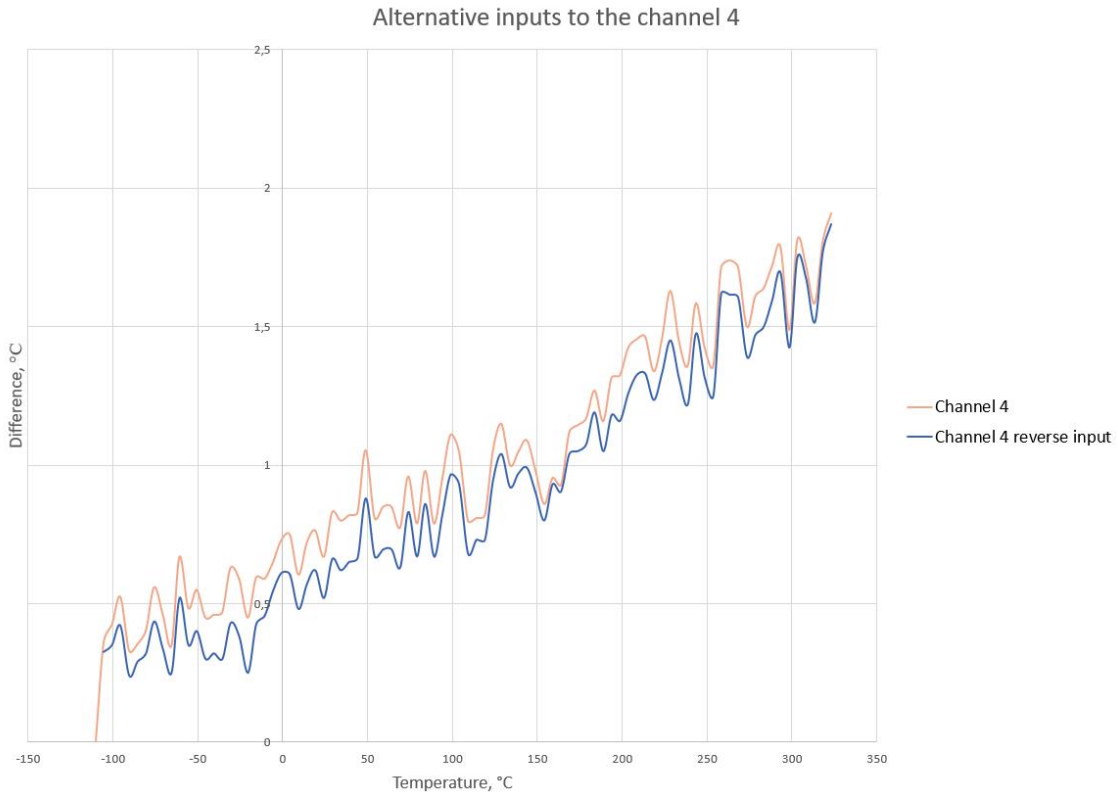


Figure 20. Alternative inputs in channel 4 (PTI)

To check the repeatability of growing difference, the measurements in the opposite direction has been done:

- First graph shows values in range [-120, 335] °C with step 5 °C
- Second graph shows values in range [335, -120] °C with step -5 °C

If the difference tends to increase to the end of measurement, error rate should grow in the opposite directions: the bigger errors should appear closer to 300 °C on the first graph and closer to -100 °C on the second graph.

Result is shown in Figure 20. Visible peaks and bottoms are on the same temperature values, not in the opposite. Therefore, the error can be considered as stable.

6 Discussion

The results have not completely fulfilled the scope. The noise performance is worse than expected and the error is higher. One of the reasons had been the bad grounding. The shield of the cable should be grounded in two points: on PT simulator board and on the rack, in that case an error will be reduced for the values, shown in Result section.

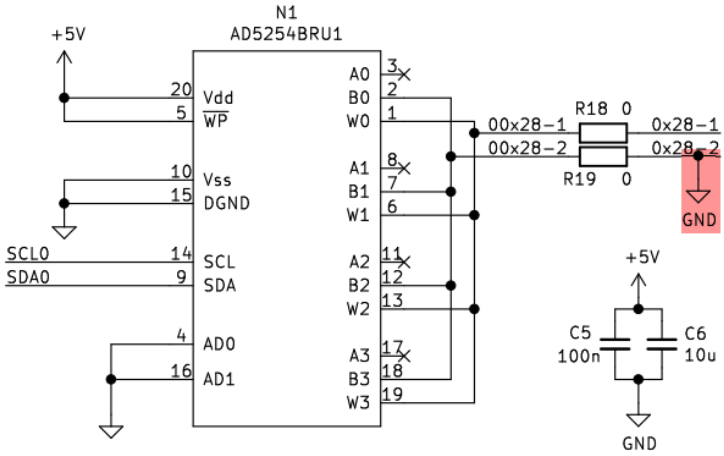


Figure 21. Ground of the channel output

All the negative output pins have been grounded as shown in the Figure 21. Now it is done with soldering the secondary pins to the ground plane. Grounding makes the output more stable: error is decreasing significantly, from 0.7 °C – 1.5 °C to 0.08 °C – 0.14 °C, as shown in Figures 17-19 (DMM and PTI noise are measured after grounding).

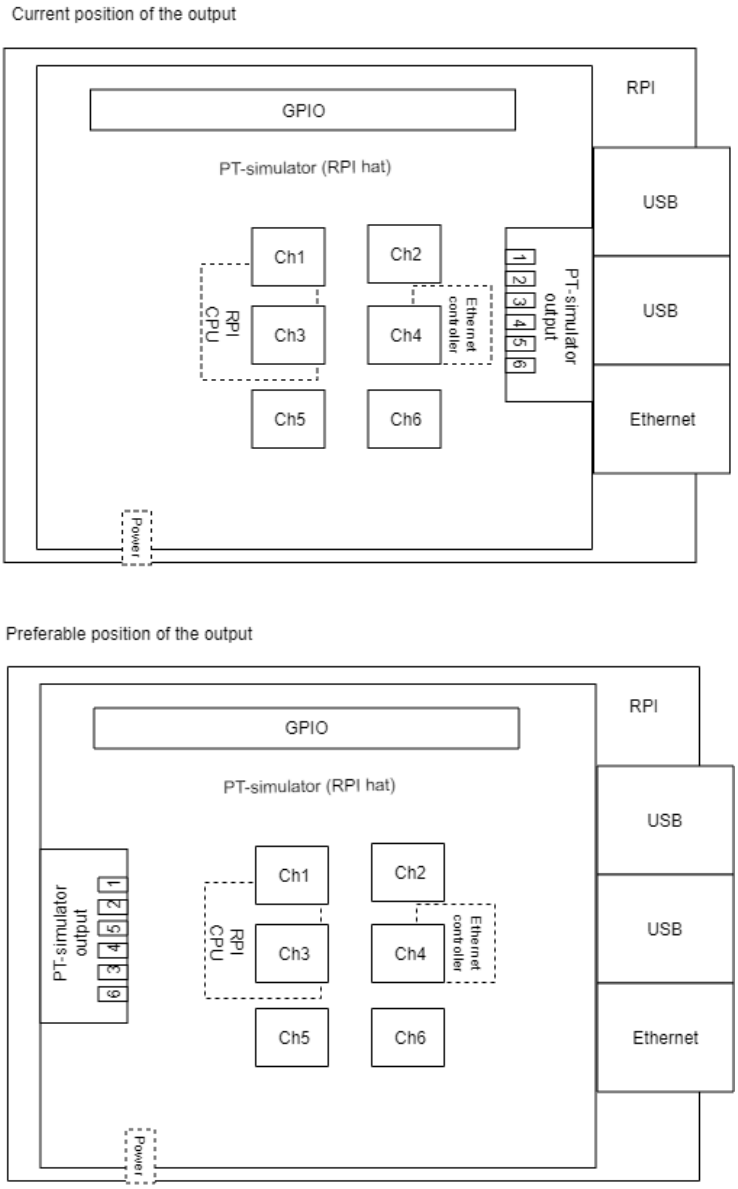


Figure 22. Positions of the output pins

Schematics of the next revision of the board should be changed. PT simulator output will be moved on the opposite direction, as shown in Figure 22. Therefore, noise from USB ports, Ethernet and Ethernet controller will not affect so much as it actually does.

Other changes will include the additional space between digital and analog lines (Baker's recommendation, [7]).

Possible reason for the high noise is digital potentiometer's degradation because of static shocks, caused by touching without grounded bracelets or another electromagnetic protection. To avoid this risk and to protect PT simulator from accidental damage, it should be used in protective case.

Development of the PT simulator software will include graphical user interface and opportunity to control many PT simulator from one PC for future integration in the existing automated framework.

7 Conclusions

Goal of the project was in the creation of the digitally controlled PT simulators for emulation PT100 and PT1000, based on digital potentiometers. Both simulators were developed and tested, their parameters are close to the expected. Fulfilled points of the scope:

- Developing of the two temperature simulators: PT100 and PT1000,
- Output temperature range should be [-120, 335] °C,
- Required resolution is 0.35 °C, reached resolution in average is 0.019 °C and the maximum step is 0.19 °C (Table 7),
- Possibility to input the temperature value without translation to resistance,
- Changing temperature in user interface.

Not fully covered points of scope:

- Required accuracy is ±1 °C achieved for DMM, but reached accuracy error for PTI module is up to 2.17 °C (Figure 20),

Future development will include the reducing of the noise, further development of the software, integration in the test automation system.

PT simulator can be used in software testing after improvement of the existing weaknesses, especially in projects with shorter temperature range, for example [-60, 200] °C.

References

[1] EKE-Electronics Ltd, EKE-Trainnet® PTI3593A SIL Pt1000 Temperature Sensor Input Module Technical Manual, Espoo: EKE-Electronics Ltd, 2018.

[2] EKE-Electronics Ltd, EKE-Trainnet® PTI2037A PT100 Temperature Sensor Input Module Technical Manual, Espoo: EKE-Electronics Ltd, 2018.

[3] C. M. Jha, Thermal sensors. Principles and Applications for Semiconductor Industries, New York: Springer, 2015.

[4] J. Fraden, Handbook of Modern Sensors. Fifth Edition, San Diego: Springer International Publishing, 2016.

[5] M. Usach, "Replacing Mechanical Potentiometers with Digital Potentiometers. Application note," Norwood.

[6] Analog Devices, Inc., "AD5254," 2003-2012. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/AD5253_5254.pdf.

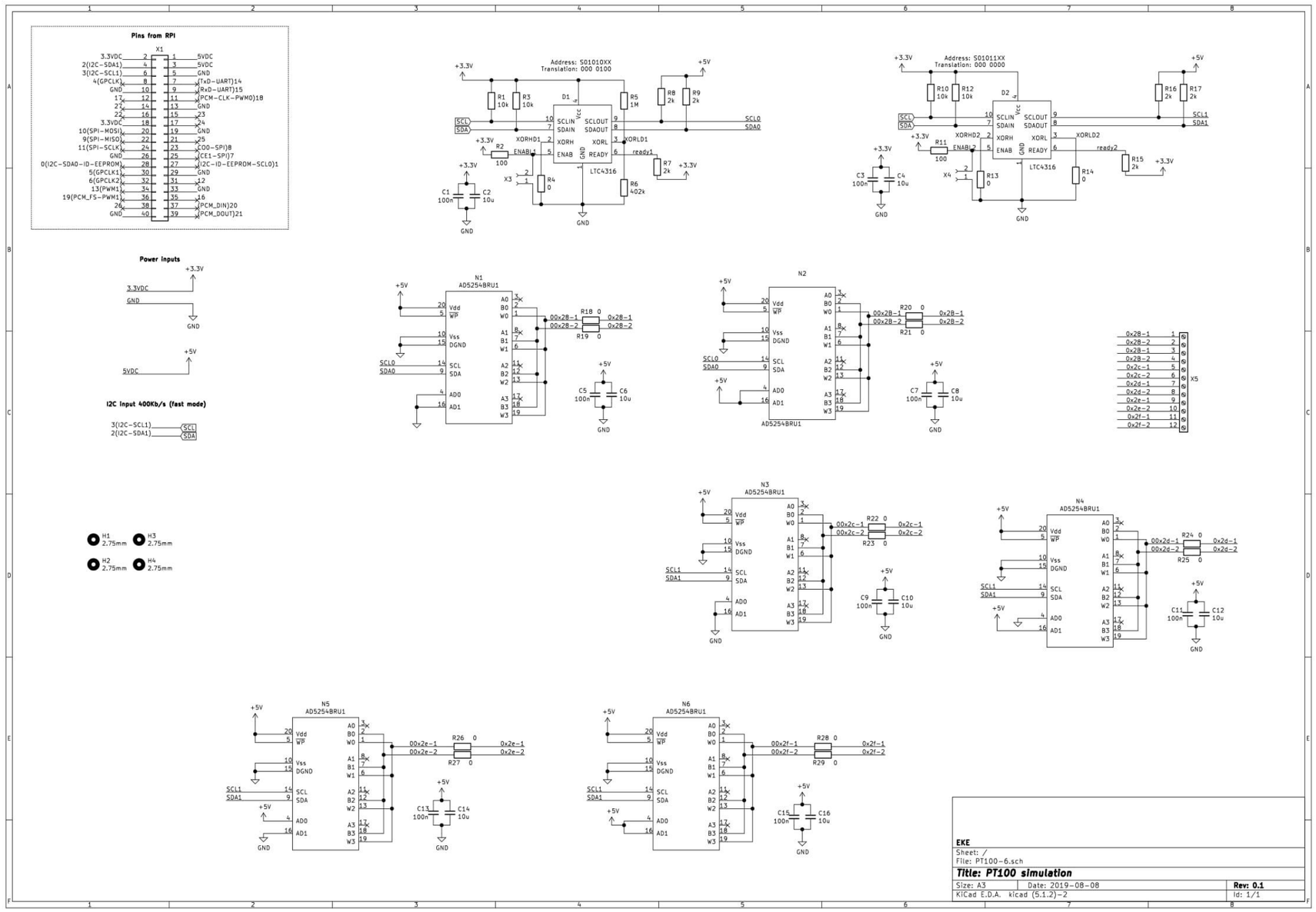
[7] B. C. Baker, Real Analog Solutions for Digital Designers, Oxford: Newnes, 2005.

[8] Linear Technology Corporation, "LTC4316," 10 2015. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/4316fa.pdf.

Appendix 1. Listing of the script “ConversionTableCreation.py”

```
'''Data creation module is creating conversion table for PT100 or PT1000 tem-
perature sensor.'''

print ("Conversion table is created...")

# Reading board type
while True:
    checker = input("Input the board type (100 or 1000): ")
    if checker=='100' or checker=='1000':
        break

f = open("tempVSresPT{}.txt".format(checker), "w+")
A = 3.9089e-3
B = -5.775e-7
C = -4.183e-12
R0 = int(float(checker))

# Resolution 0.001
decades = 1000


res_list = []
resVStemp = ()

# Creating table
for t in range (-120*decades, (335*decades+1)):
    t1 = t/decades
    if t1<0:
        res = R0 * (1 + A*t1 + B*(t1 ** 2) + C*(t1 ** 3)*(t1-100))
    else:
        res = R0 * (1 + A*t1 + B*(t1 ** 2))
    resVStemp = (t1, round(res, 5))
    res_list.append(resVStemp)

for x in res_list:
    str1 = "\t".join(str(e) for e in x)
    f.write(str1)
    f.write("\n")


f.close()
```

Appendix 2. Schematics of the PT simulator board

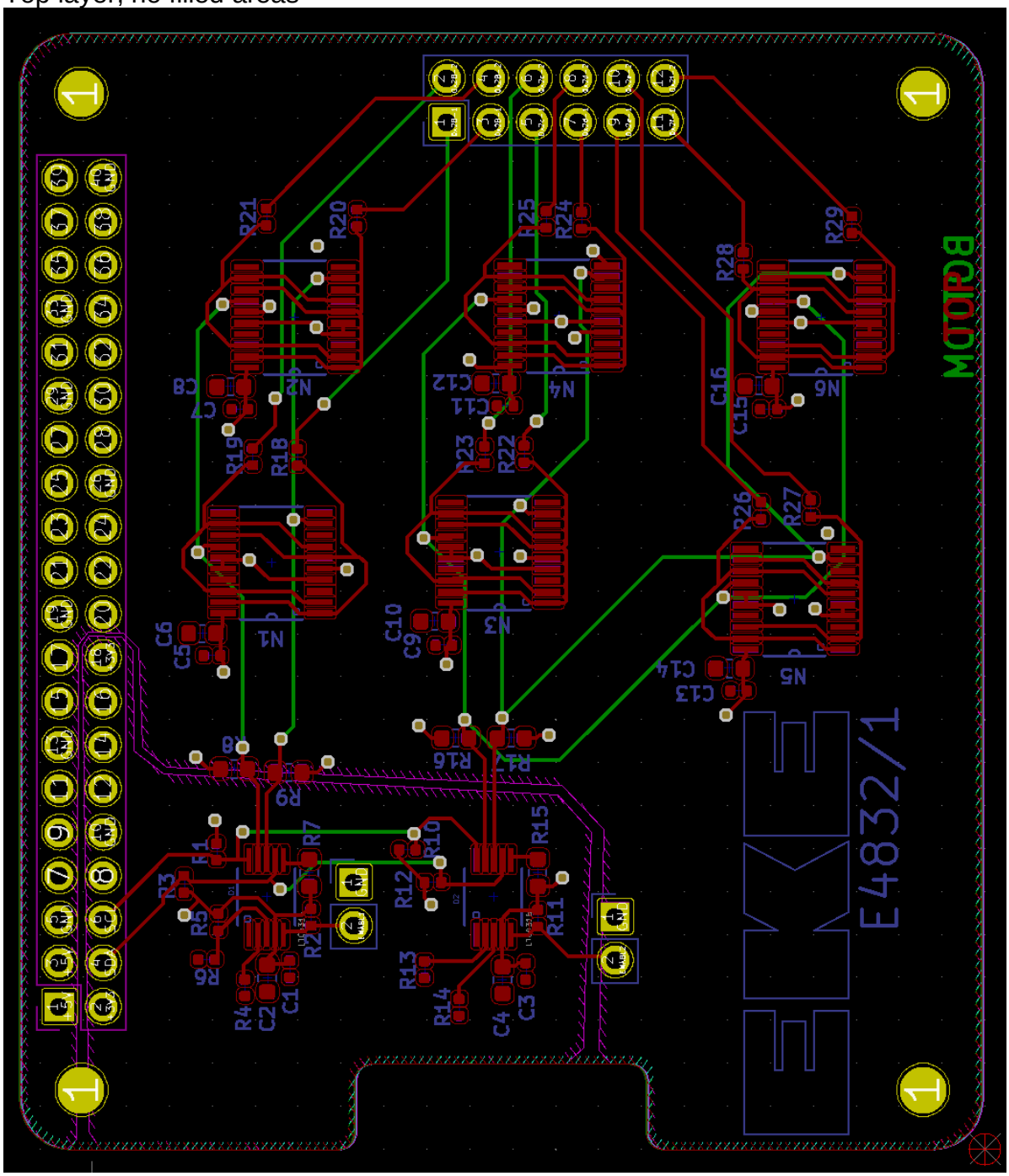


Appendix 3. A bill of materials

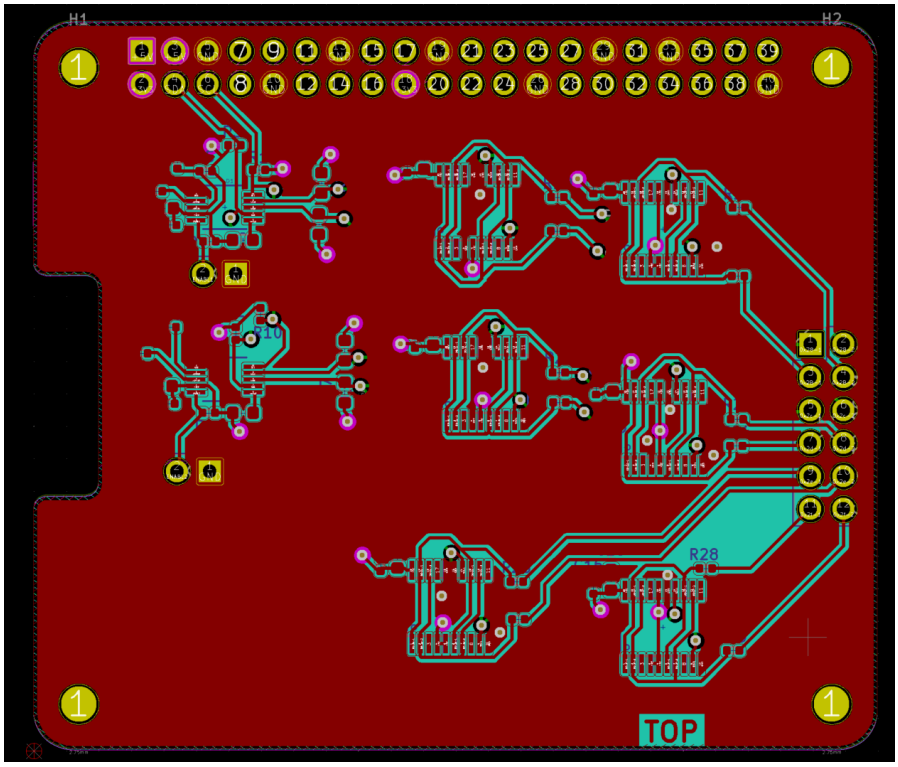
Reference	Value	Amount	Name
C1, C3, C5, C7, C9, C11, C13, C15	100n	8	Capacitor
C2, C4, C6, C8, C10, C12, C14, C16	10u	8	Capacitor
D1, D2	LTC4316	2	LTC4316
H1-H4		4	Mounting Hole
N1-N6	AD5254BRU1	6	Digital potentiometer
R5	1M	1	Resistor
R6	402k	1	Resistor
R1, R3, R10, R12	10k	4	Resistor
R2, R11	100	2	Resistor
R4, R13, R14, R18-R29	0	15	Resistor
R7-R9, R15-R17	2k	6	Resistor
X1	02x20 Female	1	Connector
X3, X4	01x02 Male	2	Connector
X5	02x06 Male	1	Connector

Appendix 4. PCB layout

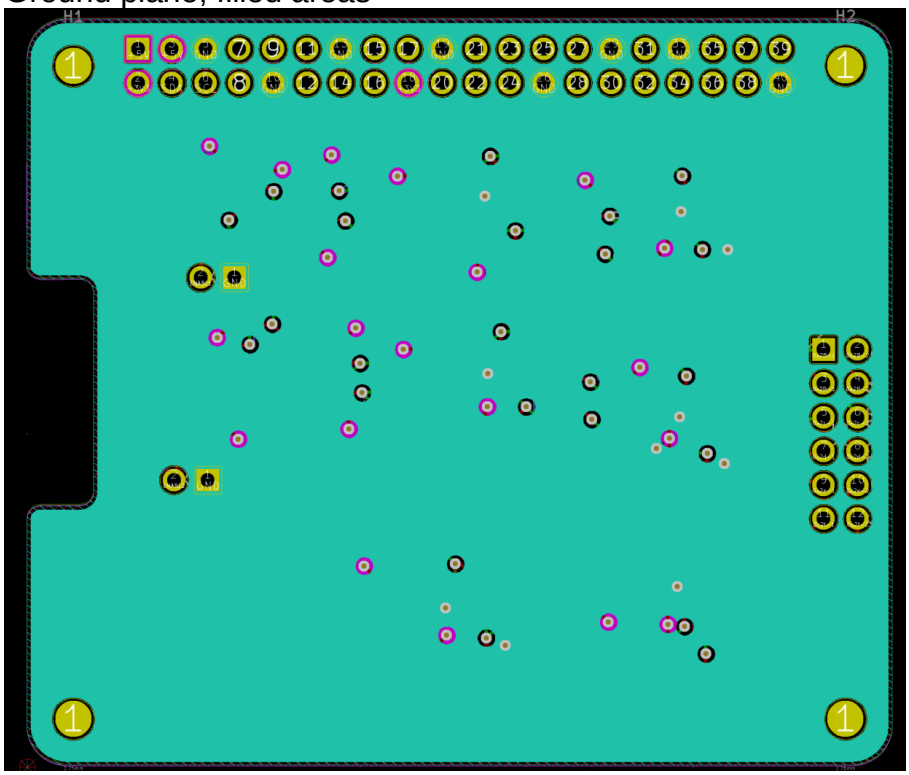
Top layer, no filled areas



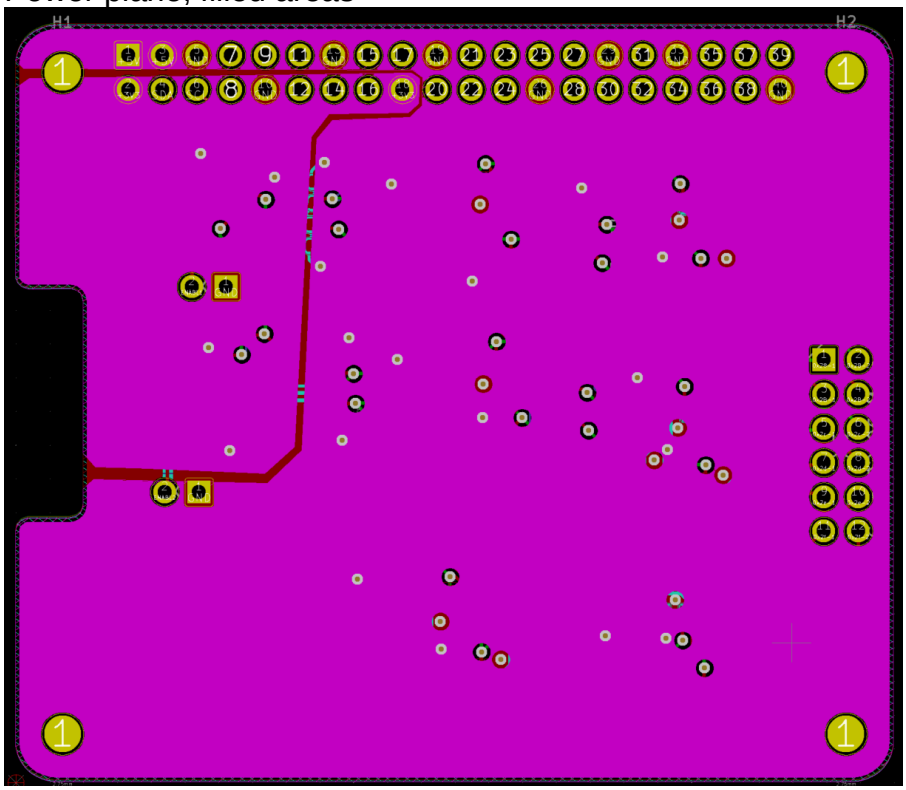
Top layer, filled areas



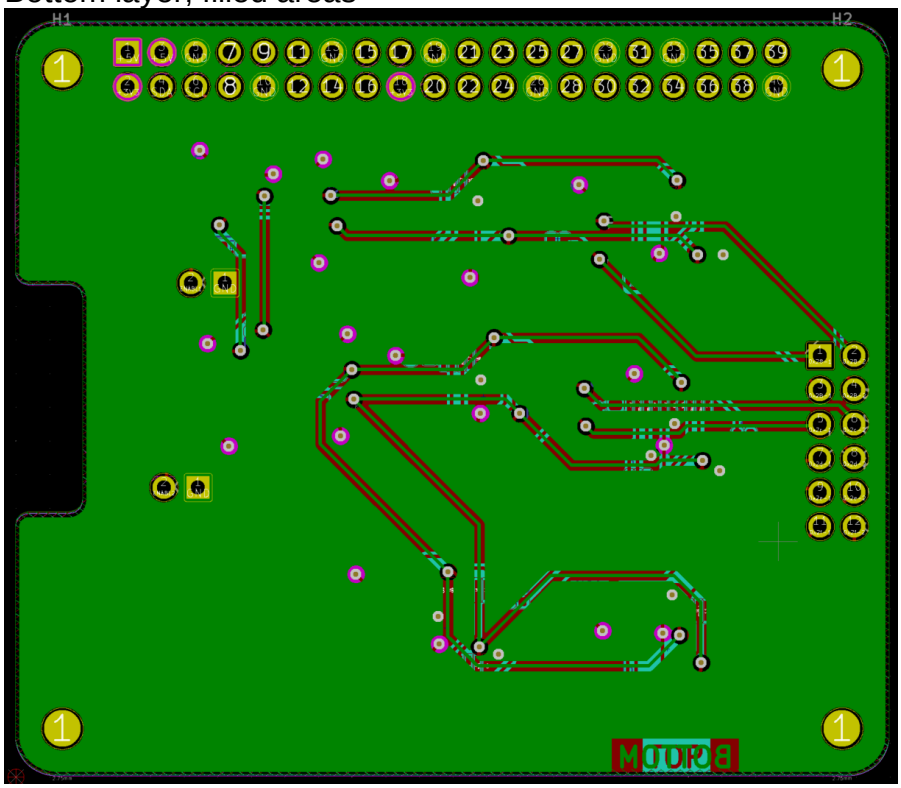
Ground plane, filled areas



Power plane, filled areas



Bottom layer, filled areas



Appendix 5. System configuration

RPI should have the static IP address, available SSH and additional settings. It can be set using the following commands:

```
$ sudo raspi-config
```

- Interfacing options/SSH/Yes
- Interfacing options/I²C/Yes
- Interfacing options/Serial/Yes (if the serial port will be used)

Changing IP address and netmask:

```
$ ifconfig eth0 192.168.1.3 netmask 255.255.0.0 up
```

Disabling sleep and hibernation to prevent calibration freeze:

```
$ sudo systemctl mask sleep.target suspend.target hibernate.target hybrid-
sleep.target
```

Changing I²C bus speed from default 100 kbps to 400 kbps:

```
$ sudo nano /boot/config.txt
```

Add the following text:

```
dtparam=i2c_arm=on,i2c_arm_baudrate=400000
```

And then reboot the device.

Using PuTTY or MobaXterm, PC should be connected to RPI using IP address that was set before. The protocol is SSH, the port is 22

Code should be copied or cloned to the RPI (for example, ~/home/pi/pt-simulator)

All scripts can be started by the following command of RPI:

```
$ python3 ScriptName.py
```

Appendix 6. Listing of the module “Calibration.py”

```
'''Module is performing the calibration of the channels for PT simulator.
Type of the board is depending on input:
100 means PT100, corresponding PTI module PTI2037A;
1000 means PT1000, corresponding module PTI3593A.
'''
import I2CWriting as wr
import vxi11
import time
import SearchNearest
import FileReading

host = '192.168.137.10'
OK = True
NOK = False

checker = ''

def dm3068_init(host):
    '''Function dm3068_init initializes the connection with digital multime-
ter'''
    instr = vxi11.Instrument(host)
    time.sleep(0.1)
    instr.timeout = 10*1000
    instr.write(":MEAS MANU")
    instr.write("DISP OFF")
    instr.write(":TRIG:DELAY 0")
    instr.write(":MEAS:RES 2")
    instr.write(":FUNC:RES")
    instr.write("RES:NPLC 1")
    instr.write("TRIG:COUNT 1")
    instr.write("SAMP:COUN 5")
    time.sleep(0.1)

    return instr


def dm3068_meas_res(instr):
    '''Function dm3068_meas_res reads the resistance measurement from digital
multimeter'''
    return float(instr.ask(":MEAS:RES?"))


def Calibration(step1 = 5, step2 = 5):
    '''Function Calibration initializes the calibration process by defining
the connected channel'''
    instr = dm3068_init(host)


    channels = wr.full_checking()
    #print(channels)
    comm = 0

    for addr in channels:
        wr.writing_i2c(int(addr, 16), 0, 0)
        time.sleep(0.1)
        val1 = dm3068_meas_res(instr)
```

```
            wr.writing_i2c(int(addr, 16), 255, 255)
            time.sleep(0.1)
            val2 = dm3068_meas_res(instr)

            if (val2-val1>100):
                test_ch = addr


        try:
            WritingValues(test_ch, instr, step1, step2)
            instr.write("DISP ON")
        except:
            print("Channel cannot be defined, connect the leads or try again")
            return

        return test_ch


def WritingValues(addr, instr, step1, step2):
    '''Function WritingValues writes values in range [0, 255]x[0, 255] with
given steps'''
    print("Calibration for channel {} in progress. It can take a while. Do not
stop!".format(addr))
    if addr == None:
        return NOK


    calib = open("files/{}_raw".format(addr), "w+")

    values = []
    for i in range(0, 256, step1):
        start_t = time.time()
        #draw_progress(i)
        for j in range(0, 256, step2):
            wr.writing_i2c(int(addr, 16), i, j)
            time.sleep(0.01)
            val_r = dm3068_meas_res(instr)


            values.append("{}\t{}\t{}\n".format(i, j, val_r))



            if j == 0:

                print ("{0}\t{1:.3f} Ohms\t{2}".format(i, val_r,
time.strftime("%H:%M:%S")))



    calib.writelines(values)
    calib.close()
    return OK



def Sorting (filename):
    '''Function Sorting receives the raw unsorted values in Ohms, sorts it
with specific step and writes the calibration table'''
    result = FileReading.ReadFile2("Tables/tempVSres{}.txt".format(checker))
```

```
    if (result!=NOK):
        temp_list = result[0]
        res_list = result[1]
    else:
        return NOK

    D = []
    if filename == None:
        return NOK
    with open("files/{}_raw".format(filename)) as calib:
        list_values = calib.readlines()
        for line in list_values:
            list_v = [float(i) for i in line.split('\t')]
            D.append([int(list_v[0]), int(list_v[1]), list_v[2]])

    D.sort(key = lambda x: x[2])

    ind = SearchNearest.SearchNearest(res_list, D[0][2])
    D[0][2] = temp_list[ind]

    D1 = []

    for x in D:
        ind = SearchNearest.SearchNearest(res_list, x[2])
        x[2] = temp_list[ind]
        D1.append (x)

    D2 = []
    lastvalue=D1[0]
    D2.append(lastvalue)

    if checker == 'PT100':
        min_step = 0.001
    elif checker == 'PT1000':
        min_step = 0.01
    else:
        min_step = 0

    for x in D1:
        if (x[2]-lastvalue[2]) >= min_step:
            D2.append(x)
            lastvalue = x

    f = open("files/{}.txt".format(filename), "w+")
    for x in D2:
        str1 = "\t".join(str(e) for e in x)
        f.write(str1)
        f.write("\n")
    f.close()


def name_check (chan_name):
    '''The function checks input. Return value is channel number or NOK'''
    if chan_name == '1' or chan_name == '2' or chan_name == '3' or chan_name
== '4' or chan_name == '5' or chan_name == '6':
        return int(float(chan_name))
    else:
        return NOK
```

```
if __name__ == "__main__":
    comm = ''
    chan_number = NOK

    while True:
        b_type = input("Input board type (100/1000): ")
        if b_type == '100' or b_type == '1000':
            checker = 'PT{}'.format(int(float(b_type)))
            with open("type.conf", 'w+') as f:
                f.write(checker+'\n')
            break

    while comm != 'n':
        comm = input("Start calibration? y/n: ")

        if comm == 'y':


            while (chan_number==NOK):
                chan_name = input("Input channel name [1-6]: ")
                chan_number = name_check (chan_name)


            val1=time.time()

            step1 = int(float(input("Input step in columns [1, 5, 10] (recom-
mended 5): ")))
            step2 = int(float(input("Input step in rows [1, 5, 10] (recom-
mended 1): ")))
            channel = Calibration(step1, step2)

            Sorting(channel)

            comm = ''
            val2= time.time()

            with open("type.conf", 'a') as f:
                f.write('{}\t{}\n'.format(str(chan_number), channel))
                chan_name = ''
                chan_number = NOK


            print("Calibration of channel {}, time spent is {} seconds".for-
mat(channel, val2-val1))

        elif comm == 'n':
            break
```

Appendix 7. Listing of the script “ManualInput.py”

```
'''The module's function is reading user input and write the temperature val-
ues to the channels: all or chosen.
Results of the input session can be found in the file "channel_data/Date_ses-
sion.txt"
'''
from DataInput import TemperatureWriting
import SerialReading
import time
comm = ''
ch = ''
channels = []
temp = 0
ser = SerialReading.CreateSerial()

# Input cycle
while (comm != 'q'):
    outp = []
    outp.clear()
    comm = input("Input temperature [-110, 325], 'c' for choosing channel 'c'
or 'q' for quit: ")
    if comm == 'q':
        break
    # Choosing channels
    elif comm == 'c':
        ch = input("Write channel numbers with ', ', channels should be in
range [0, 5]: ")
        try:
            channels = [int(float(i)) for i in ch.split(', ')]
        except:
            pass
        continue

    #Input temperature. Only one float value
    else:
        try:
            temp = float(comm)
        except:
            continue
    outp = TemperatureWriting(temp, channels)

    SerialReading.SerialReading(ser)
    pti_values = SerialReading.SerialReading(ser)
    pti_values_str = ''

    for line in pti_values[0:5]:
        pti_values_str+=str(line)+'\t'
    pti_values_str+=str(pti_values[5])+'\n'

    outp.append(pti_values_str)
    for o in outp:
        print(o)

    #writinf output to the file
    with open("channel_data/{}_session.txt".format(time.strftime("%d.%m.%Y")),
'a+') as f:
        f.writelines(outp)
```

Appendix 8. Listing of the module “DataInput.py”

```
'''The module "Data Input" receives the temperature and optionally the list of
channels, then checks every channel for availability and write the closest
temperature from calibration table.
The output is a list of channel numbers and written values'''
import I2CWriting as wr
import FileReading
import SearchNearest
import sys
import time

OK = True
NOK = False

try:
    with open('type.conf') as f:
        checker = f.readline()
        channel_dict = FileReading.ReadFile1("type.conf")
        #print(checker)
except:
    print("board type is undefined!")
    sys.exit()

def ChannelCheck (channels):
    '''The function "ChannelCheck":
    input: list of the channels or individual channel
    output: dictionary of channel number and channel address
    '''

    outp = {}

    if type(channels) == int:
        if channel_dict.get(channels) == None:
            return channel_dict
        else:
            outp.update({channels: channel_dict.get(channels)})
    elif type(channels) == list or type(channels) == tuple:
        for ch in channels:
            if channel_dict.get(ch) == None:
                pass
            else:
                outp.update({ch: channel_dict.get(ch)})
    else:
        return channel_dict

    if len(outp) == 0:
        return channel_dict
    else:
        return outp



def TemperatureWriting (temp = 0, channels = None):
    '''The function "TemperatureWriting":
    input: float temperature in degrees, list of the channel numbers [1-6] or
individual channel
    Output: list of strings with "{}\t{}\t{}\t{}\n".format(addr, temp, val-
ues_list[ind], time.strftime("%H:%M:%S")
```

```
    '''
    channels_dict = ChannelCheck(channels)
    outp = []

    for addr in channels_dict.values():
        calib = "Tables/{}.txt".format(addr)
        result = FileReading.ReadFile3(calib)
        tap1_list = result[0]
        tap2_list = result[1]
        values_list = result[2]

        for it in channels_dict.items():
            if it[1]==addr:
                ch_n = it[0]

        ind = SearchNearest.SearchNearest(values_list, temp)

        if wr.CheckingByCh(addr):
            wr.I2CWriting(addr, tap1_list[ind], tap2_list[ind])

            str_n = "{}\t{}\t{}\t{}\n".format(ch_n, temp, values_list[ind],
time.strftime("%H:%M:%S"))
        else:
            str_n = "{} is not available!\t{}\n".format(ch_n,
time.strftime("%H:%M:%S"))

        outp.append(str_n)


    return outp
```