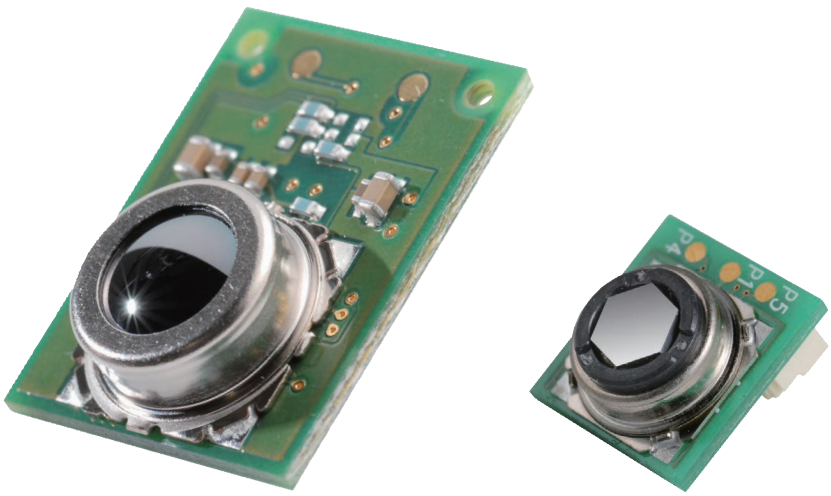
OMRON

## MEMS Thermal Sensors

# D6T

## User's Manual

MEMS Thermal Sensors



A284-E1-05

**Table of Contents**

# 1 Overview

This user manual describes the usage procedures, precautions, and other information regarding D6T-series MEMS Thermal Sensors. This document also serves as a supplement to the product catalog. Reference this document together with the product catalog when using this device.

# 2 Principles of Operation

The following list describes an overview of the measuring operation of the MEMS Thermal Sensors.

- The silicon lens focuses radiant heat (far-infrared rays) emitted from objects onto the thermopile sensor in the module. (*1)
- The thermopile sensor generates electromotive force in accordance with the radiant energy (far-infrared rays) focused on it.
- The values of this electromotive force and the internal thermal sensor are measured. Then, the device calculates the measured value (temperature of the object) via an interpolation calculation that compares the measured values with an internally stored lookup table. (*2)
- The measured value is output via the $I^2C$ bus, and read using a host system.

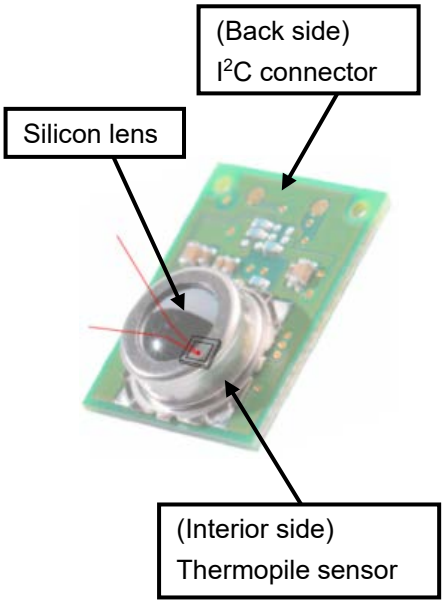
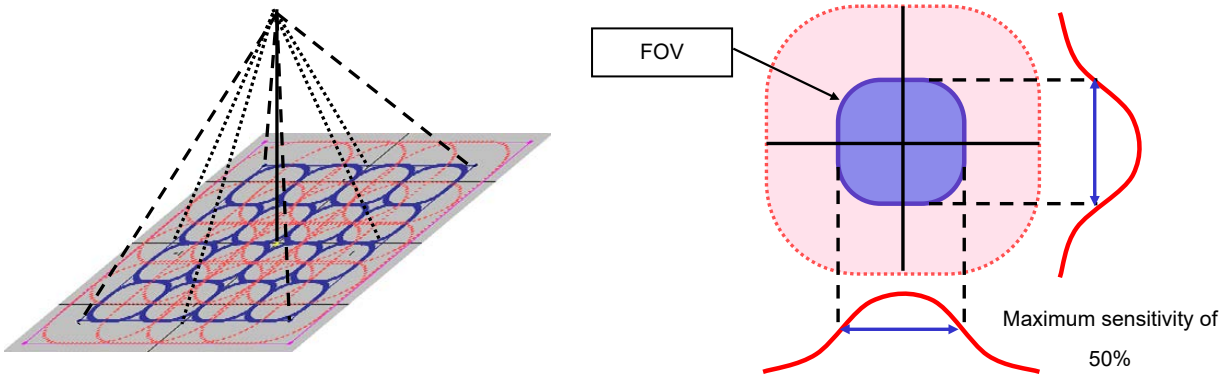


Fig. 1. Module Configuration

*1. The D6T-1A-01/02 models use a silicon filter.
*2. D6T-1A-01/02、D6T-8L-09/09H use a temperature conversion circuit in the ASIC to calculate measured values (temperatures of objects).

## 3 Product Features

MEMS Thermal Sensors feature a silicon lens optically designed to have specific sensitivity characteristics. Our Thermal Sensors feature the same field of view (FOV) at a maximum sensitivity of 50% as general sensors.




a) Conceptual illustration of D6T-44L-06/06H FOV (16-channel)

b) Conceptual illustration of single-element FOV and XY-axis sensitivity characteristics

Fig. 2. Field of View (FOV) and Sensitivity Characteristics Illustrations

The sensitive areas of elements are wider than the FOV-specification width. If the size of the measured object is smaller than the sensitive area of an element, the background temperature of objects other than the intended object will become a factor.

Our Thermal Sensors use a reference heat source (a blackbody furnace) to correct temperature values. However, note that differences in emissivity due to composition of measured objects, surface shape, and the occupancy ratio of objects within sensitive areas all affect temperature values.

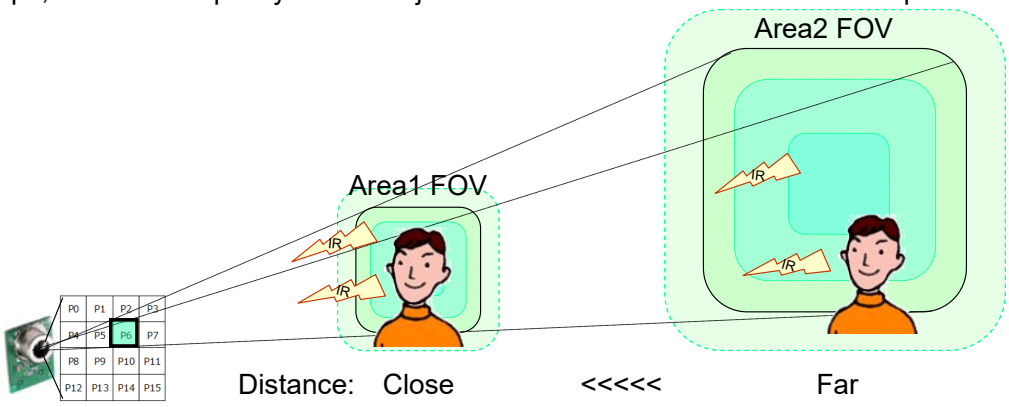



Fig. 3. Distance as Factor of Fluctuations in Temperature Values

The measurable area (FOV) enlarges as the distance between the measured object increases. The occupancy ratio of objects (people) in the FOV reduces as the distance increases. For this reason, as the distance increases, the temperature values become more a representation (level of influence) of the background temperature than the temperature of the intended object (people). In other words, to correctly measure temperature of the intended objects, the measured object must be sufficiently larger than the FOV area.

Using a MEMS Thermal Sensor as a human sensor is limited to close-distance applications for simple determination of temperature value only. To increase the detection distance, determination accuracy must be improved through software processing that factors temporal changes, position of heat sources, human behavior information, and so on.

# 4 Usage Procedure

## 4.1 Connectors

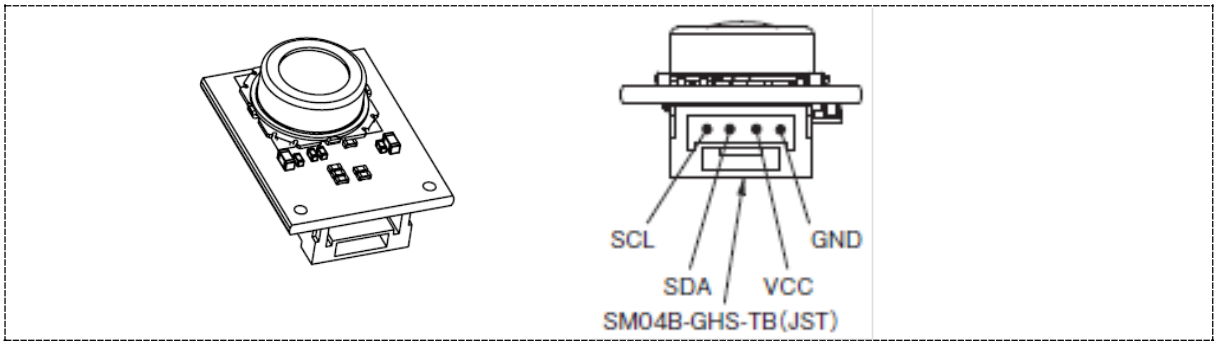



Fig. 4. Product Exterior (Reference)

Connector Pins

Table 1. Connector Pin Table

| | | |
|---|---|---|
| 1 | GND | GND power supply pin |
| 2 | VCC | VCC power supply pin (5 V ±10%) |
| 3 | SDA | $I^2C$ (5 V) data |
| 4 | SCL | $I^2C$ (5 V) clock |

Connector Parts Materials

Connector part model: SM04B-GHS-TB (JST)
Contact: SSHL-002T-P0.2 (JST)
Housing: GHR-04V-S (JST)

The lens height and circuit board size varies by model. Refer to the product catalog for more information on dimensions. Use a 4-pin connector as described above to connect this module to systems.

## 4.2 Example Electrical Connections

Scenario 1: 5 V MCU Direct Connection (Same voltage as the microcontroller power supply)

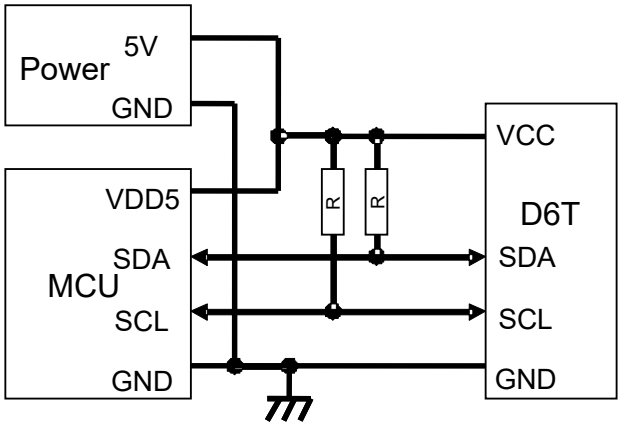



Fig. 5. Connecting to 5 V Microcontroller

Scenario 2: 3 V MCU ($I^2C$ port is 5 V fault tolerant)

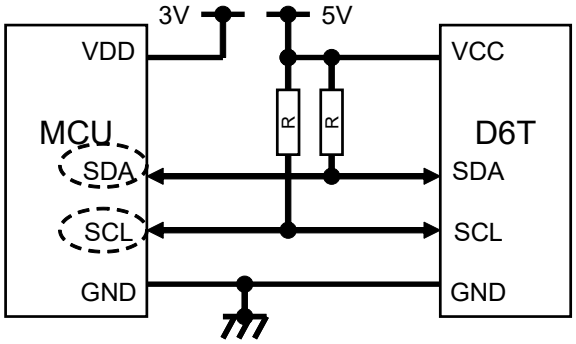



Fig. 6. 5 V Fault Tolerant Specification

Scenario 3: Using an $I^2C$ Level Converter
(Not a 5 V fault tolerant specification, or other devices are also connected to the 3 V $I^2C$ bus)

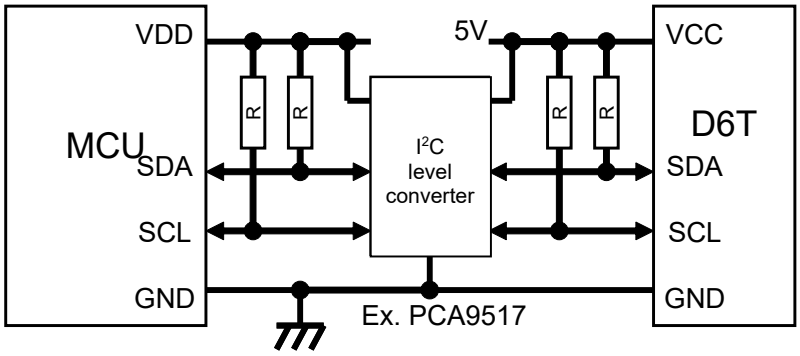



Fig. 7. Using a Level Converter

Scenario 4: Using a Bidirectional Open-Drain GPIO Terminal and Performing $I^2C$ Communication Processing in Software
(MCU does not have built-in $I^2C$ functionality)
*1. For D6T-44L-06 / D6T-44L-06H / D6T-32L-01A, clock stretch needs to be supported. Refer to the chapter on clock stretch of the D6T communication specification below.

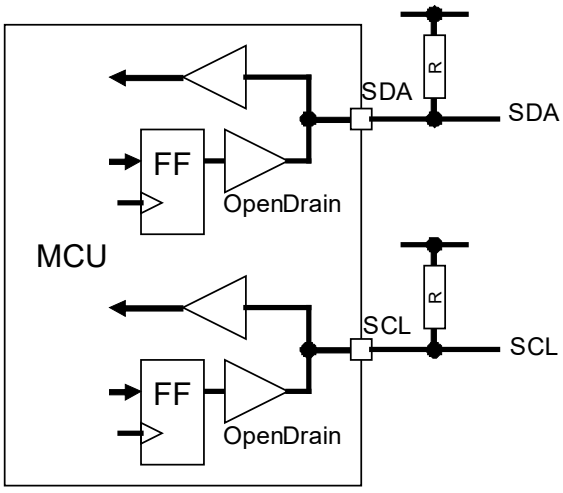



Fig. 8. Using a GPIO Terminal

Scenario 5: Using an I²C Bus-Switching IC (Connecting multiple D6T sensors)
(This sensor cannot change slave addresses)
*1. Most bus-switching ICs also have power voltage conversion functionality.

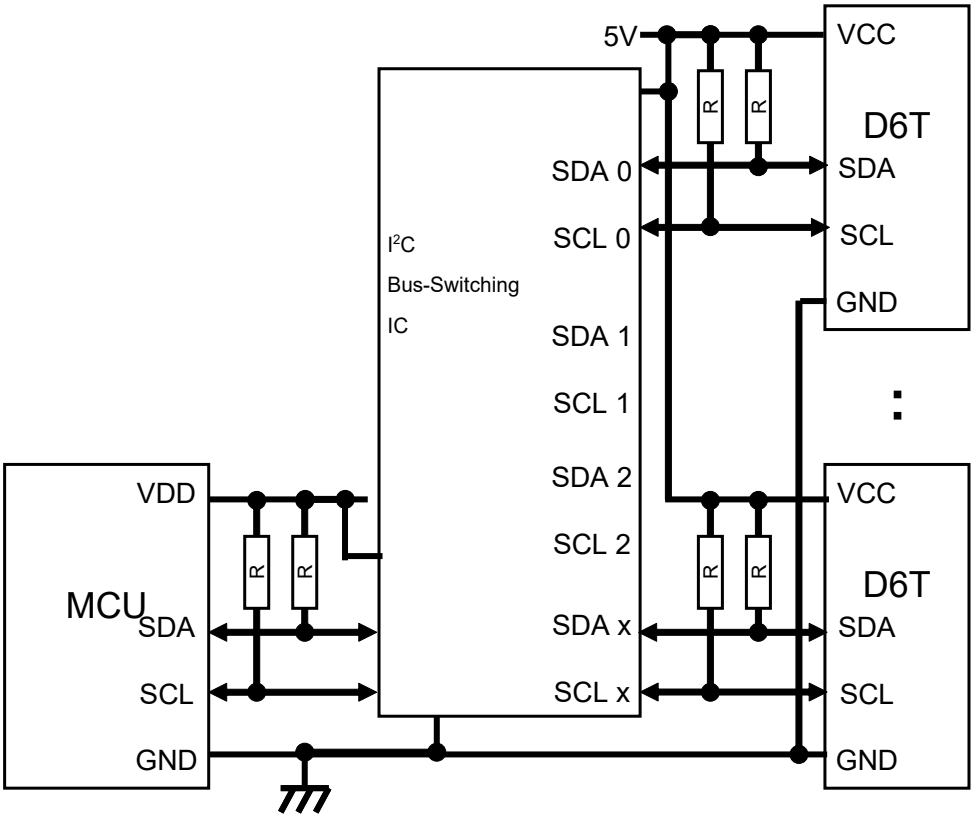



Fig. 9. When using the I²C Bus-Switching IC

Pull-up Resistance Values

The appropriate value of the pull-up resistor depends on the usage conditions of the user such as the capacity components of the wiring. Be sure to evaluate these before making a decision.

(Check the I²C specifications. In most cases, the range is set to approximately 3 k to 10 kΩ.)

### 4.3 Surface Cover Material

Make sure that cover material used when installing the MEMS Thermal Sensor as part of an assembly has sufficient radiant heat (far-infrared) transmissivity. A far-infrared transmissive grade of high-density polyethylene (HDPE) is often used as a cover material due to being relatively inexpensive and easy to mold. The rate of decay varies depending on cover thickness, and so make the cover as thin as possible to reduce negative impact on detection performance. However, if the cover is too thin, the internal sensor will be visible as illustrated in the following photos.

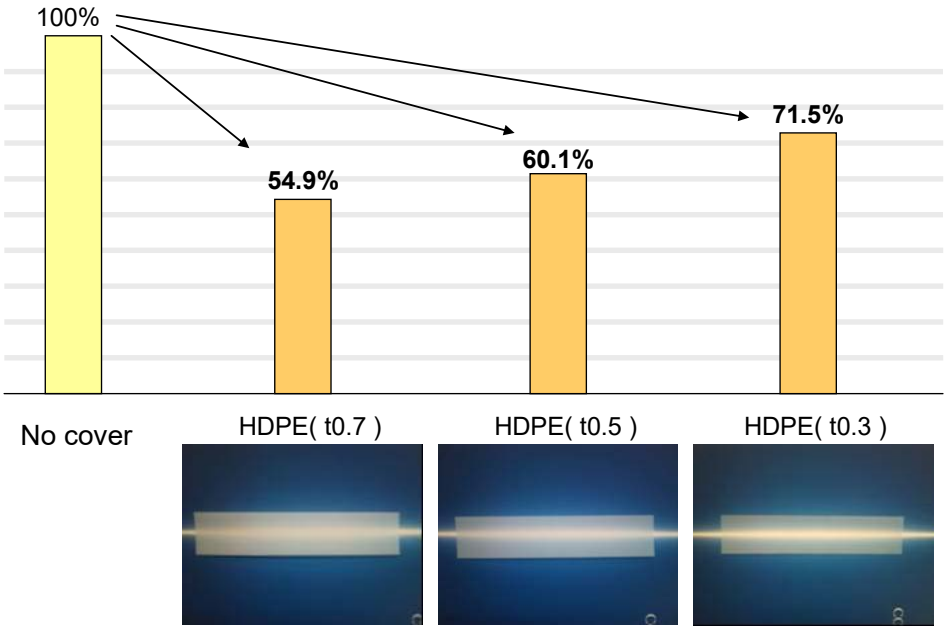
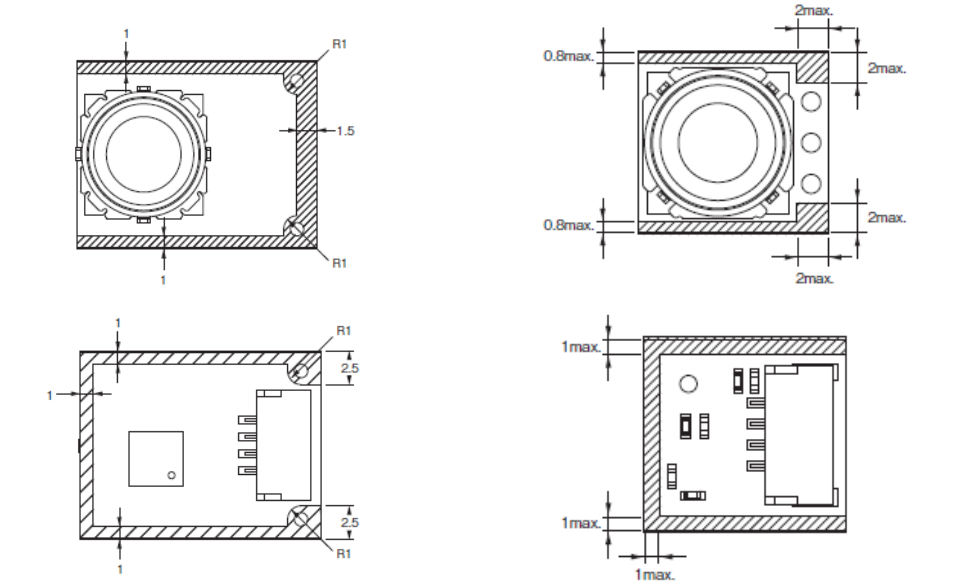


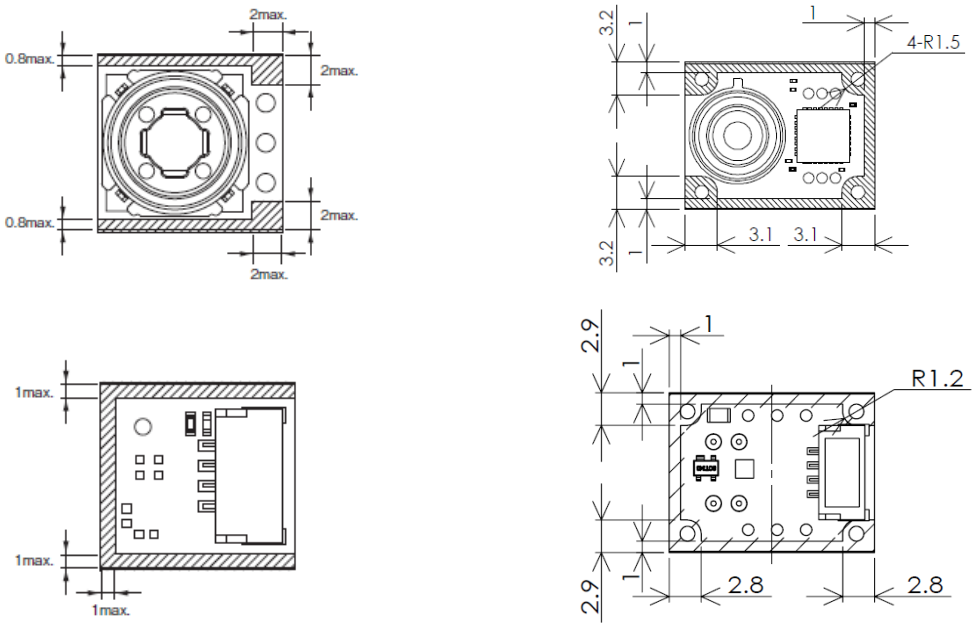

Fig. 10. Relationship Between HDPE Thickness and Transparency (Reference)

## 4.4 Sensor Securement

Install the MEMS Thermal Sensor so that it is enclosed by casing and secured at mountable areas.




D6T-44L-06 / D6T-44L-06H　　D6T-8L-09 / D6T-8L-09H




D6T-1A-01 / D6T-1A-02　　D6T-32L-01A

Fig. 11. Mountable Areas (Shaded Areas)

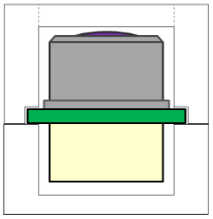


Fig. 12. Sensor Securement (Reference)

## 5 Development/Evaluation Tool

Note:1. 2JCIE-EV01-RP1, 2JCIE-EV01-AR1, 2JCIE-EV01-FT1, and 2JCIE-HARNESS-01 in this catalog are discontinued at the end of January 2025.

Two types of sample code are available as software development support tools.

1. Sample code for Raspberry Pi
2. Sample code for Arduino

The sample codes for Raspberry Pi and Arduino can be used together with the OMRON evaluation board to allow hardware to be easily connected.
The OMRON evaluation board supports the following 3 types of platform. Evaluation can be performed easily by connecting non-contact thermal sensor D6T, evaluation board, and harness to the platform.
Evaluation Board URL: (http://www.omron.co.jp/ecb/sensor/evaluation-board/2jcie)

Table 2. Evaluation Board Map

| Platform | Evaluation Board | Harness for Connection (Between evaluation board and D6T) | Sample Code |
|---|---|---|---|
| Raspberry Pi *1 For connection | 2JCIE-EV01-RP1  | 2JCIE-HARNESS-01 | https://github.com/omron-devhub/d6t-2jcieev01-raspberrypi |
| Arduino *2 For connection | 2JCIE-EV01-AR1  | 2JCIE-HARNESS-01 | https://github.com/omron-devhub/d6t-2jcieev01-arduino |
| ESP32 Feather *3 For connection | 2JCIE-EV01-FT1  | 2JCIE-HARNESS-01 | https://github.com/omron-devhub/d6t-2jcieev01-arduino |

*1. Raspberry Pi is a registered trademark of the Raspberry Pi Foundation.
*2. Arduino is a registered trademark of Arduino LLC and Arduino SRL.
*3. Feather is a registered trademark of Adafruit Industries LLC.

The sample code can be used without the evaluation board. However, the customer needs to provide wiring to the sensor.

Sample codes are written in C language. The $I^2C$ control function can be changed according to the MCU to be used in order to develop software using the sample codes.

The sample code is for evaluation purposes only, and OMRON does not guarantee its function.

## 1. Sample code for Raspberry Pi

(1) Connect D6T, harness and OMRON evaluation board (2JCIE-EV01-RP1) to Raspberry Pi

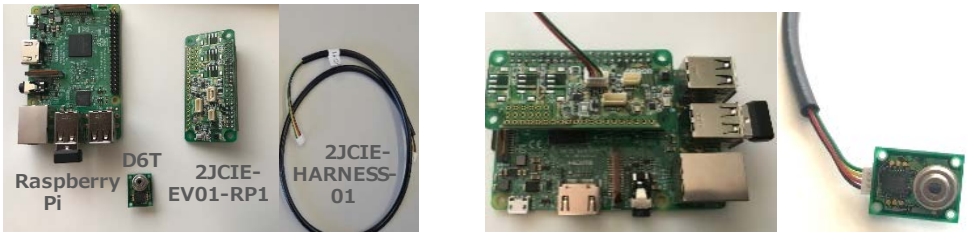



Fig.13 Set-up

(2) Enable the I²C

Launch Raspberry Pi, open “Preferences” > “Raspberry Pi Configuration” from the Start menu, “Enable” the I²C, then restart.

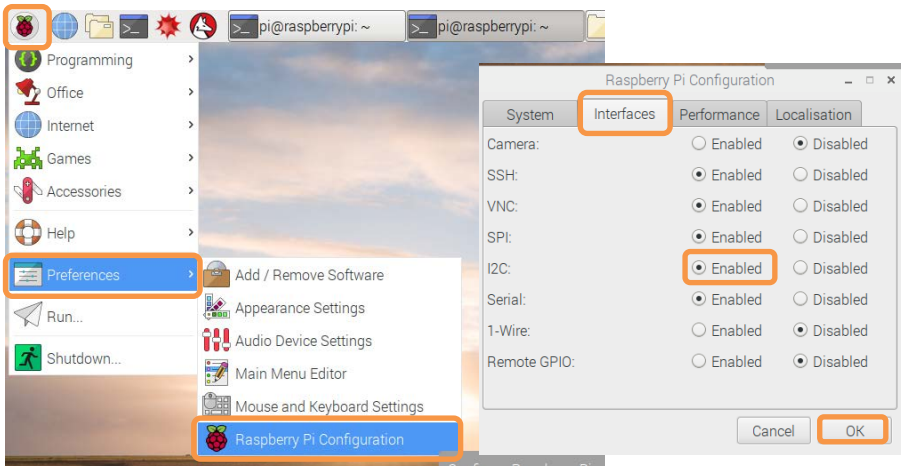



Fig.14 Enable I²C

(3) Download sample code

Open GitHub from the URL below and download Zip file.

GitHub URL: https://github.com/omron-devhub/d6t-2jcieev01-raspberrypi

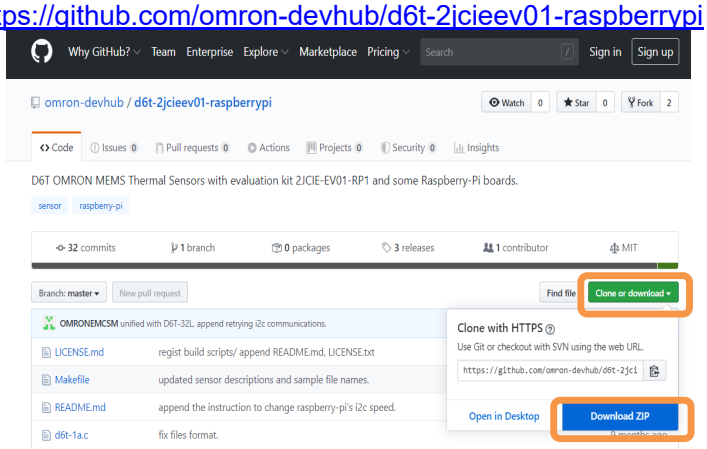



Fig.15 Download sample code

Alternatively, open the Terminal and execute the following command.

```
~$ git clone https://github.com/omron-devhub/d6t-2jcieev01-raspberrypi
```

Alternatively, after downloading the Zip file on another PC with the Internet connection, move the zip file to the Raspberry Pi using a USB memory, etc.

(4) Make file
Open the Terminal and execute the following command.

```
pi@raspberrypi:~ $ cd Downloads/
pi@raspberrypi:~/Downloads $ unzip d6t-2jcieev01-raspberrypi-master.zip
pi@raspberrypi:~/Downloads $ cd d6t-2jcieev01-raspberrypi-master/
pi@raspberrypi:~/Downloads/d6t-2jcieev01-raspberrypi-master $ make all
```

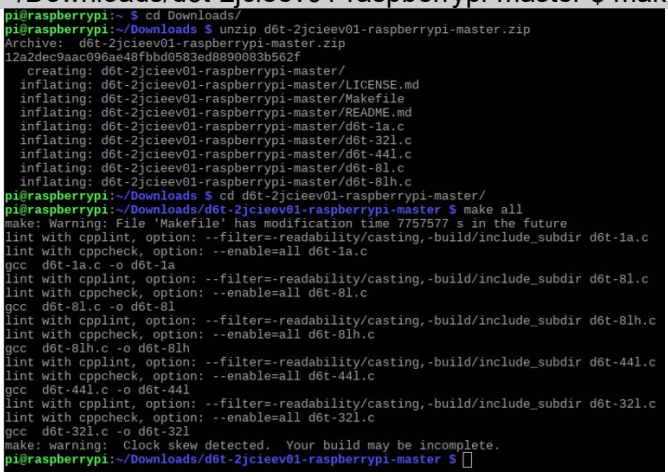



Fig.16 Make file

(5) Run the file
Execute the following command to obtain the data.

-- For D6T-1A-01 / D6T-1A-02

```
pi@raspberrypi:~/Downloads/d6t-2jcieev01-raspberrypi-master $ ./d6t-1a
```

-- For D6T-8L-09

```
pi@raspberrypi:~/Downloads/d6t-2jcieev01-raspberrypi-master $ ./d6t-8l
```

-- For D6T-8L-09H

```
pi@raspberrypi:~/Downloads/d6t-2jcieev01-raspberrypi-master $ ./d6t-8lh
```

-- For D6T-44L-06 / D6T-44L-06H

```
pi@raspberrypi:~/Downloads/d6t-2jcieev01-raspberrypi-master $ ./d6t-44l
```

-- For D6T-32L-01A

```
pi@raspberrypi:~/Downloads/d6t-2jcieev01-raspberrypi-master $ ./d6t-32l
```

(To stop obtaining data, press the “Ctrl” and “C” keys at the same time.)

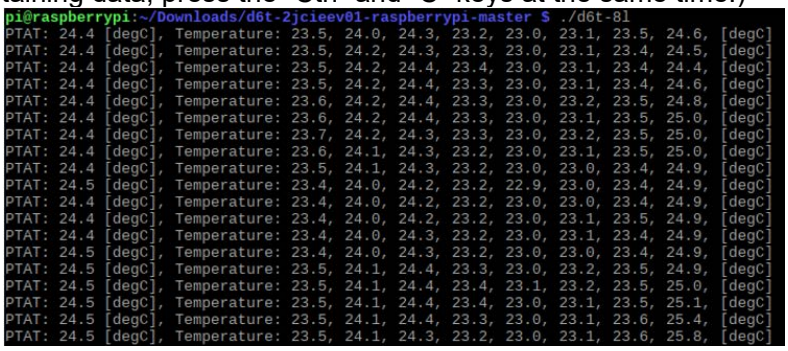



Fig.17 Run the file

## 2. Sample code operation procedure for Arduino

(1) Connect D6T, harness and OMRON evaluation board (2JCIE-EV01-AR1) to Arduino

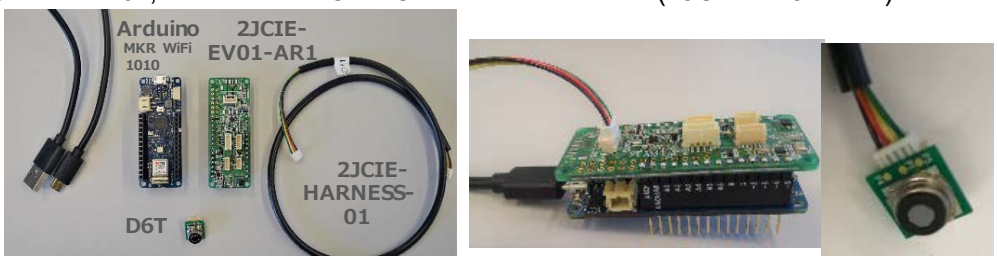



Fig.18 Set-up

(2) Download Arduino IDE

Download Arduino IDE from the URL below.

https://www.arduino.cc/en/Main/Software

(3) Recognize Arduino on Arduino IDE

Open Arduino IDE and connect Arduino to the PC via USB.

If the following message appears, install the package for Arduino MKR.

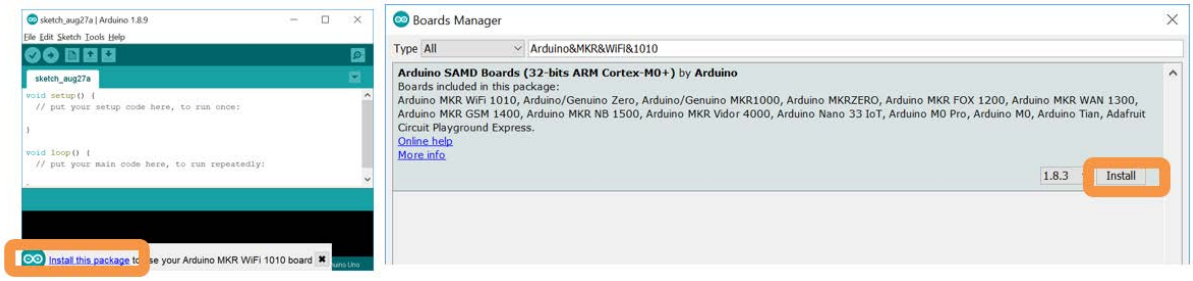



Fig.19 Arduino IDE

Install the driver.

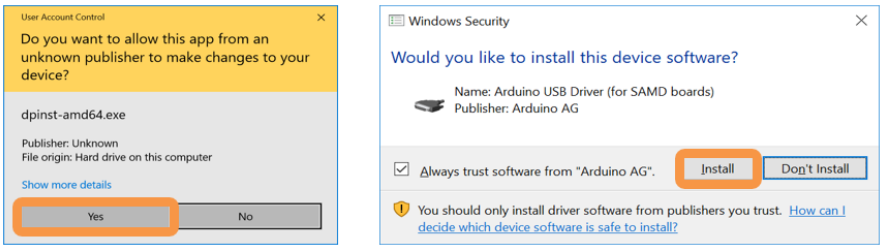



Fig.20 Driver

Look for “Device Manager” in the Windows Start menu.

Check Arduino's COM port number recognized by the PC.

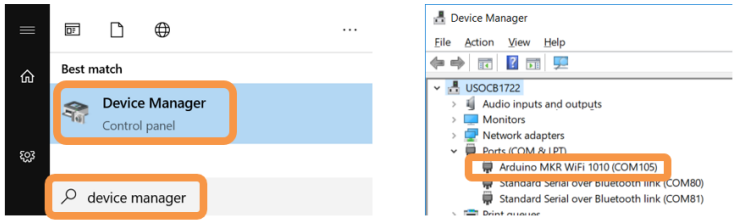



Fig.21 Device manager

“Tools” -> “Board Arduino” -> “Arduino MKR WiFi1010”.

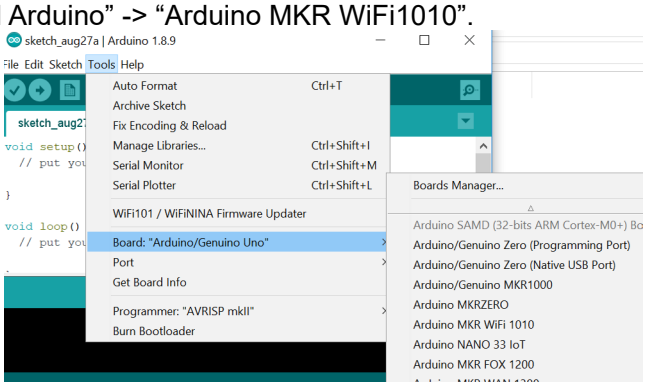



Fig.22 Select Arduino

“Tools” -> “PORT” -> Choose Arduino's COM port.

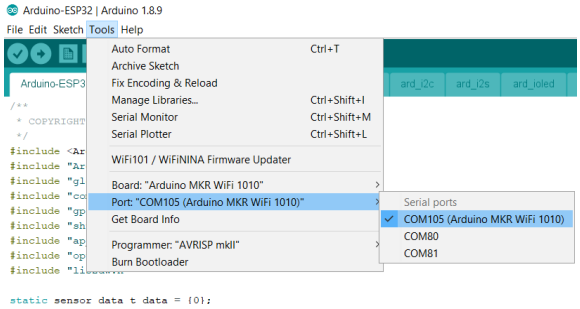



Fig.23 Select COM port

(4) Download sample code

Download the zip file by connecting to the GitHub's URL below.

GitHub URL: https://github.com/omron-devhub/d6t-2jcieev01-arduino

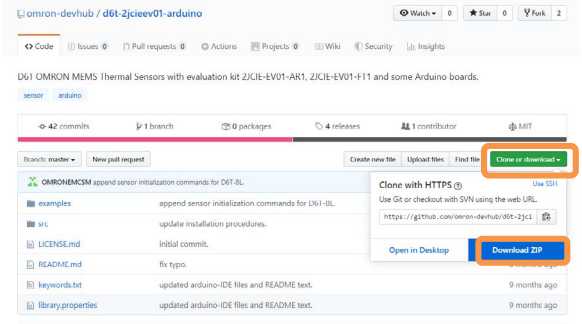



Fig.24 Download sample code

(5) Upload the sample code to Arduino.

“Sketch” -> “Include Library” -> “Add .ZIP Library”

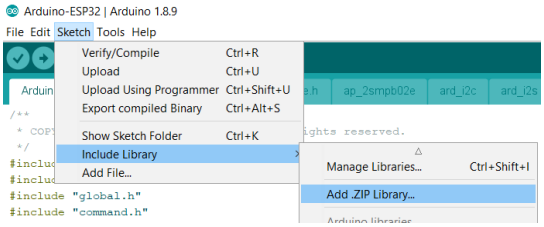



Fig.25 Add zip

Select the “Downloads” folder. Select “d6t-2jcieev01-arduino-master.zip”.

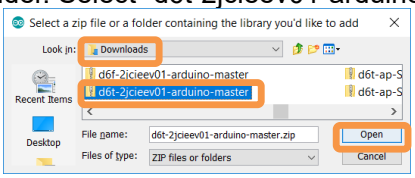



Fig.26 Select zip file

Select the file according to the model of the sensor.
“File” -> “Examples” -> “D6T-2JCIE-EV01”
-> “d6t-1a” or “d6t-8l” or “d6t-8lh or “d6t-44l” or “d6t-32l”

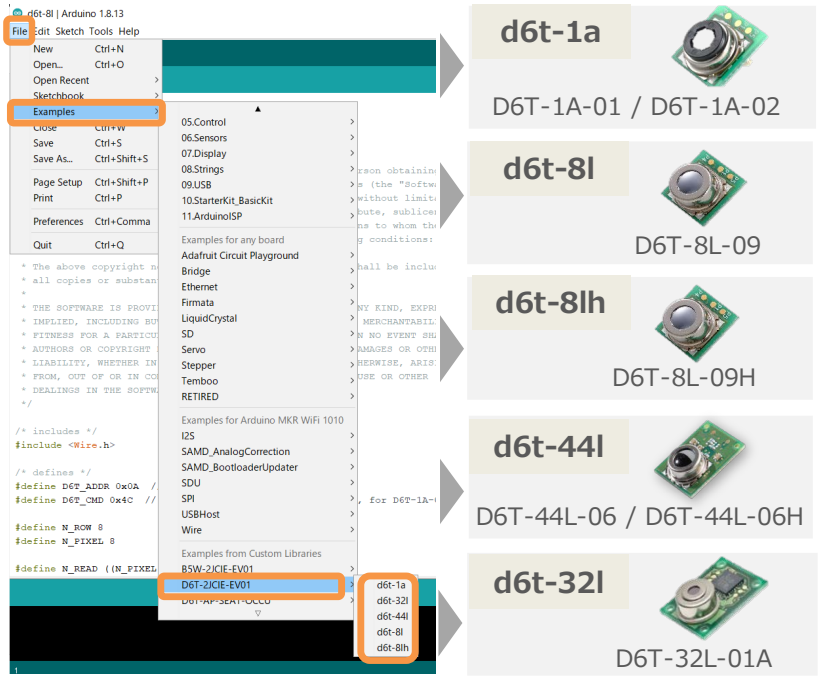



Fig.27 Select file

Click “Verify”. Check for any errors.

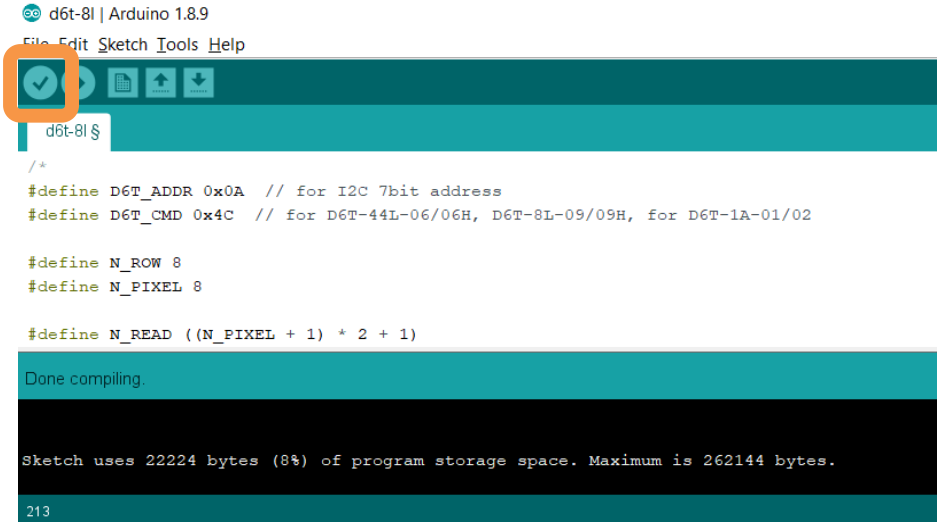



Fig.28 Verify

Click “Upload”. Check to see if “CPU reset” is shown.


Fig.29 Upload

(6) Obtaining data

“Tools” -> “Serial Monitor”

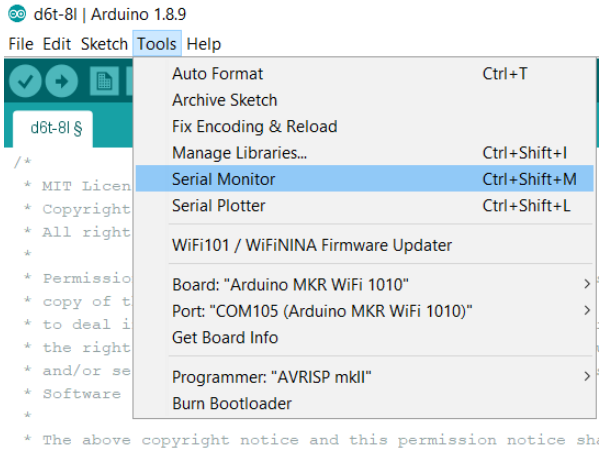



Fig.30 Serial monitor

The data is shown.

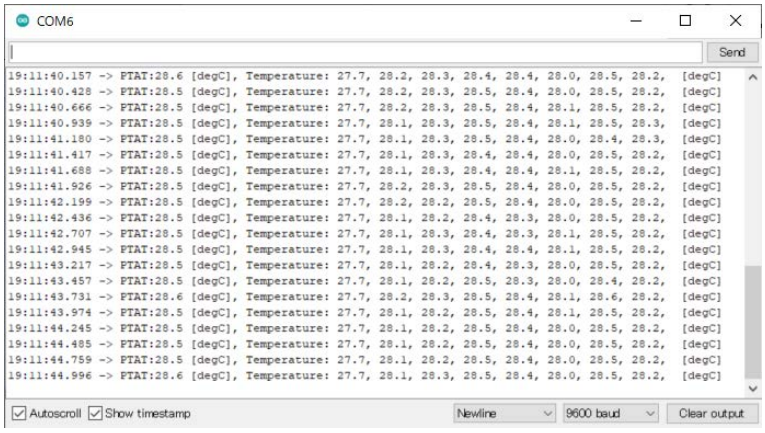



Fig.31 data

## 6 Frequently Asked Questions

| | |
|---|---|
| Question | Time from power ON until measurement is possible. |
| Answer | The output temperature reaches the temperature accuracy range within a few seconds after the power is turned on, but it takes about 15 minutes for the temperature to stabilize completely. (Reference Value) |
| Question | Communication error with the sensor occurs. Or an abnormal value is output. What are the possible factors? |
| Answer | Check the following points.<br>● Common for D6T<br>・ Is the voltage of 5V supplied correctly?<br>・ Does the pull-up resistor have an appropriate value?<br>・ Is the conversion of the sensor data performed properly?<br>(Example) When P0(Low):0x20, P0 (High):0x01,<br>Data = (int16_t) ( ((uint16_t)P0(Low)) + (((uint16_t)P0(High))<< 8) ) = 0x0120 = 288<br>●D6T-1A-01 / D6T-1A-02<br>・ Are the data measured at intervals of 100 ms or longer?<br>●D6T-8L-09<br>・ Are the data measured at intervals of 250 ms or longer?<br>・ Is the initialization command sent after the power is turned on?<br>●D6T-8L-09H<br>・ Are the data measured at intervals of 250 ms or longer?<br>・ Is the initialization command sent after the power is turned on?<br>・ Is the sensor data converted correctly? (D6T-8L-09H outputs a value 5 times the temperature.)<br>Example) Temperature output value: 560 -> 560/5 = 112.0 (°C)<br>●D6T-44L-06 / D6T-44L-06H<br>・ Is the $I^2C$ clock stretch of the MCU enabled?<br>・ Are the data measured at intervals of 300 ms or longer?<br>●D6T-32L-01A<br>・ Is the $I^2C$ clock stretch of the MCU enabled?<br>・ Does the $I^2C$'s read function have a sufficient data buffer (2051 bytes)?<br>・ Are the data measured at intervals of 200 ms or longer? |
| Question | It is in communication with the sensor, but the temperature is high (or low). Or the temperature is not stable. What are the possible factors? |
| Answer | The following factors may be in play.<br>・ The temperature includes the background temperature as the object is smaller relative to the FOV (field of view).<br>・ Something other than the object is present in the field of view.<br>・ The object being measured is unstable in temperature.<br>・ The temperature is changing.<br>・ The customer's frame is affecting the temperature. (The frame is in the field of view. The interior of the frame is generating heat. ) |
| Question | Is it possible to further reduce the power consumption? |
| Answer | The D6T Series does not have an “operation mode for power-saving sleep” setting, making it necessary to turn off the power in order to reduce the power consumption. |
| Question | Is driving at the power supply voltage of 3 [V] or changing the slave address possible? |
| Answer | The D6T series does not have these features. |
| Question | Is it possible to further expand the viewing angle? |
| Answer | The FOV is set in consideration of the restrictions due to the thickness and refractive index of the silicon lens.<br>As the FOV per element increases, the measurement and detection capability decreases. This is one of the reasons why the viewing angle cannot simply be widened. In order to measure a wide range, it is necessary to move the sensor or set up multiple sensors. |

| | |
|---|---|
| Question | Is it possible to distinguish between humans, animals, electrical appliances, etc.? |
| Answer | The non-contact temperature sensor is only capable of acquiring measured temperature data. The objects need to be distinguished by the user using software based on the behavior of the measurement data. Judgment software can be developed in consideration of usage conditions to improve the judgment accuracy. |
| Question | What is the distance that it can detect as a motion sensor? |
| Answer | The distance is greatly affected by the installation conditions and the performance of the judgment algorithm.<br>Although it depends on the FOV area per element of the thermopile sensor and the size of the measured object, a rough guideline for the distance is about 5 to 6 [m]. |
| Question | Can malfunctions be caused by the infrared remote controller? |
| Answer | Since the silicon lens hardly transmits visible light to near infrared rays with a wavelength of 1.2 [μm] or less, there is no risk of malfunctions due to infrared rays of the remote controller. Far infrared rays emitted as radiation heat are about 4 to 14 [μm]. |

## 7 Definition of Terms

- Thermopile

  A device cascaded to a thermocouple to increase voltage.

  Thermocouples are arranged so that hot junctions are adjacent.

- NETD (used in catalogs)

  Acronym for Noise Equivalent Temperature Difference. This represents the conversion of noise into a temperature value.

  This term is often used to represent temperature resolution as the estimated minimum value by which changes in temperature can be determined.

- FOV

  Acronym for Field of View. This term is used to represent the viewing angle index. This value is often defined using a sensitivity peak of 50%.

# D6T Communication specifications

D6T-1A-01
D6T-1A-02

D6T-8L-09

D6T-8L-09H

D6T-44L-06
D6T-44L-06H

D6T-32L-01A

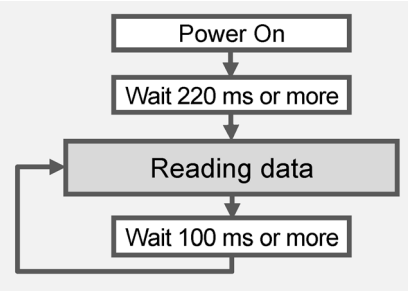
# Communication specifications
# D6T-1A-01
# D6T-1A-02

(1) Outline of I²C Interface

| | |
|---|---|
| Slave Address | 7bit (0001_010b)<br>8bit (with R/W bit) expression: Read : 15h, Write : 14h |
| Data bit width | 8bit (MSB-first) |
| Clock speed | Max. 100kHz |
| Clock stretch supported | Not supported |

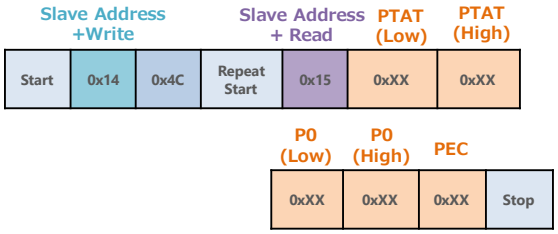
(2) Communication procedure

The procedure for communicating with the sensor is as shown on the right. The sensor with standard specifications updates the measurement data within every 100 ms. This is repeated regardless of the communication, and the measurement timing cannot be controlled externally.




Reading data

Send the following commands to acquire the data.
The received data has 5 bytes.




[Received data item]

・PTAT:

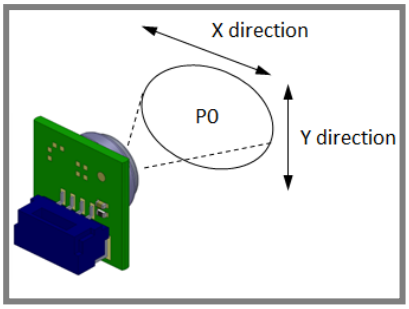
Reference temperature data inside the sensor. 10-fold temperature value [°C] with a 16-bit signed integer (two's complement).

(Calculation example)
When PTAT (Low): 0x20, PTAT (High): 0x01,
0x0120 -> 288 -> 288/10 = 28.8 (°C) at Int16

・ P0:

Temperature data for each pixel. See the figure for the layout. 10-fold temperature value [°C] with a 16-bit signed integer (two's complement)

(Calculation example)
When P0 (Low): 0x87, P0 (High): 0xFE,
0xFE87 -> -121 -> -121/10 = -12.1 (°C) at Int16



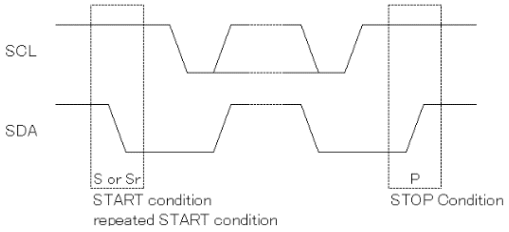

・PEC:

Packet error check code. PEC is data for checking for errors using the CRC-8 method, and is added at the end of the communication output. The target range of PEC is from Slave Address + Write to P0 (High) data. PEC values are created using each 1-byte value. This PEC value can be used by users to detect communication failures and improve data reliability.
(See SMBus specifications for details)

(3) I²C access protocol

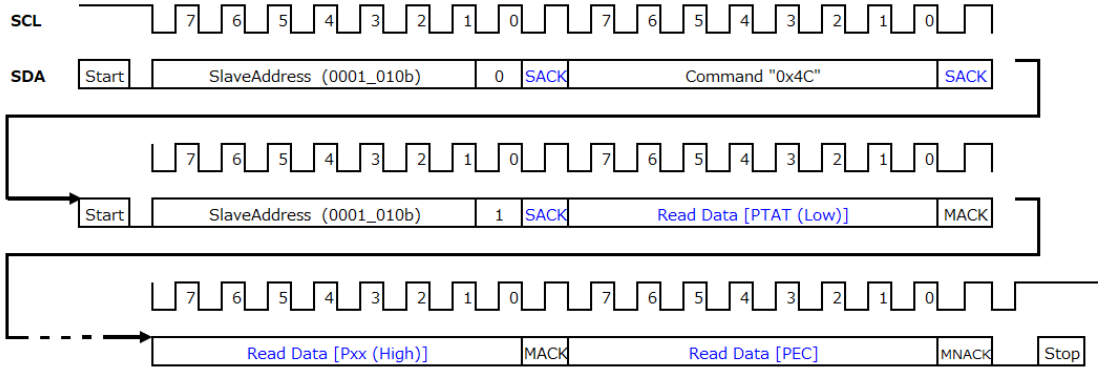
The terms of the I²C access protocol are as follows.

| | |
|---|---|
| “S” | : Start Condition |
| “Sr” | : Repeat Start Condition |
| “P” | : Stop Condition |
| “W/R” | : Write (Lo) / Read (Hi) |
| “SACK” | : Acknowledge by Slave |
| “MACK” | : Acknowledge by Master |
| “MNACK” | : No-acknowledge by Master |




I²C read access protocol

First, send a command in write mode. Next, set to Repeat Start Condition and then switch to read mode to read the data.




***1. Black characters : Master → Slave**, **Blue characters : Slave → Master**

If the SDA or SCL continues low input for the following amount of time, the communication will time out and the sensor will stop.

・D6T-1A-01 / D6T-1A-02: 70msec

If the sensor determines that the communication has timed out, NACK is returned at the Write access, but the read value will be FFFFh at the Read access. PEC is recommended for checking data as it is capable of judging that the read value is abnormal.

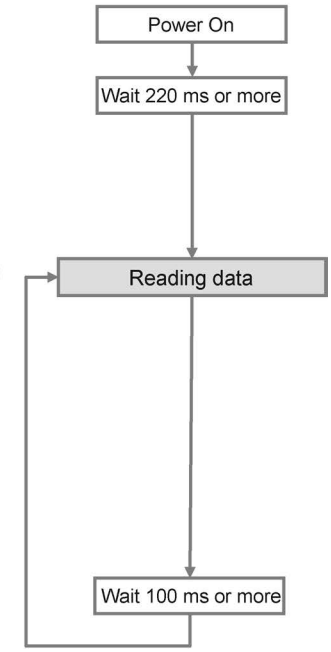
(4) Sample Code
Sample code for Raspberry Pi can be found in the URL below.
https://github.com/omron-devhub/d6t-2jcieev01-raspberrypi/blob/master/d6t-1a.c

Sample code for Arduino can be found in the URL below.
https://github.com/omron-devhub/d6t-2jcieev01-arduino/blob/master/examples/d6t-1a/d6t-1a.ino

The structure of the sample code is as follows.

```
/** <!-- main - Thermal sensor {{{1 -->
 * Read data
 */
int main() {
    int i;
        int16_t itemp;

        delay(220);

        while(1){
                // Read data via I2C
                memset(rbuf, 0, N_READ);
                uint32_t ret = i2c_read_reg8(D6T_ADDR, D6T_CMD, rbuf, N_READ);
                D6T_checkPEC(rbuf, N_READ - 1);

        //Convert to temperature data (degC)
                ptat = (double)conv8us_s16_le(rbuf, 0) / 10.0;
                for (i = 0; i < N_PIXEL; i++) {
                        itemp = conv8us_s16_le(rbuf, 2 + 2*i);
                        pix_data[i] = (double)itemp / 10.0;
                }

        //Output results
                printf("PTAT: %4.1f [degC], Temperature: ", ptat);
                for (i = 0; i < N_PIXEL; i++) {
                    printf("%4.1f, ", pix_data[i]);
                }
                printf("[degC]\n");

                delay(100);
        }
}
```

I²C Read
PEC confirmation
Convert to PTAT
Convert to temperature value
Display on screen



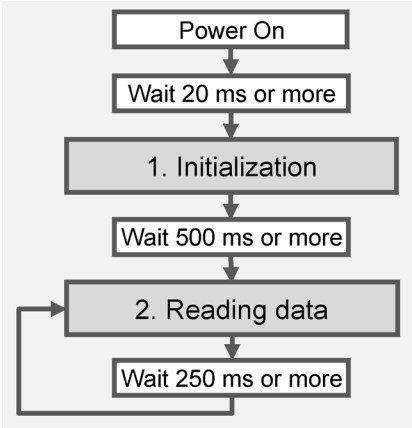
# Communication specifications D6T-8L-09

(1) Outline of I²C Interface

| | |
|---|---|
| Slave Address | 7bit (0001_010b)<br>8bit (with R/W bit) expression: Read : 15h, Write : 14h |
| Data bit width | 8bit (MSB-first) |
| Clock speed | Max. 100kHz |
| Clock stretch supported | Not supported |

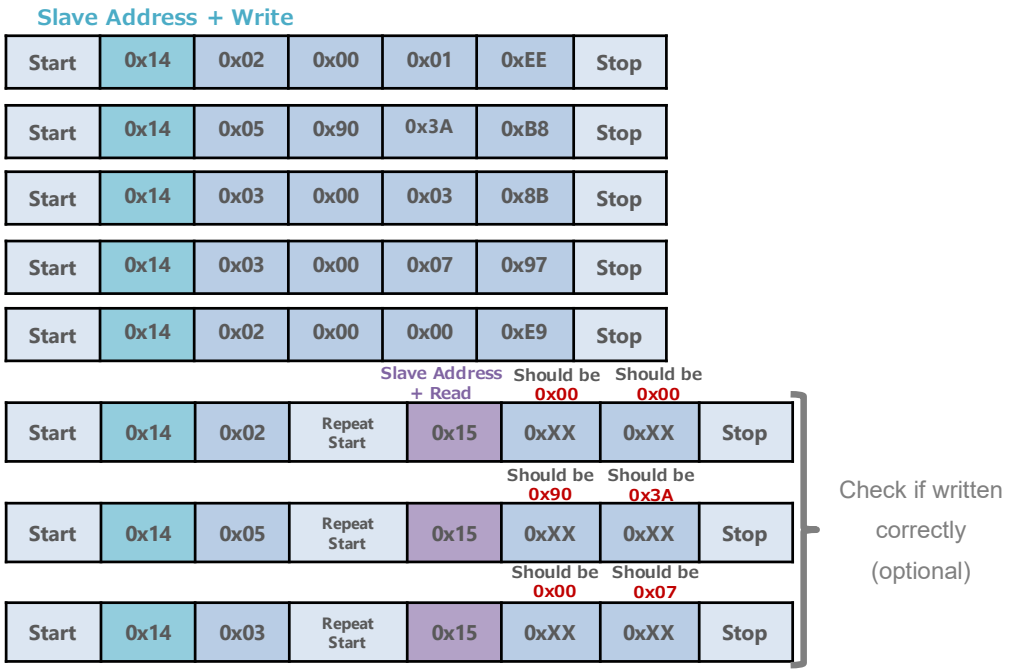
(2) Communication procedure

The procedure for communicating with the sensor is as shown below.
The sensor with standard specifications updates the measurement data within every 250 ms. This is repeated regardless of the communication, and the measurement timing cannot be controlled externally.




1. Initialization

Send and receive commands as shown below.



2. Reading data

Send the following commands to acquire the data. The received data has 19 bytes.

| | Slave Address +Write | | | Slave Address + Read | PTAT(Low) | PTAT(High) |
|---|---|---|---|---|---|---|
| Start | 0x14 | 0x4C | Repeat Start | 0x15 | 0xXX | 0xXX |

| P0(Low) | P0(High) | P1(Low) | P1(High) | P2(Low) | P2(High) | P3(Low) | P3(High) |
|---|---|---|---|---|---|---|---|
| 0xXX | 0xXX | 0xXX | 0xXX | 0xXX | 0xXX | 0xXX | 0xXX |

| P4(Low) | P4(High) | P5(Low) | P5(High) | P6(Low) | P6(High) | P7(Low) | P7(High) | PEC | |
|---|---|---|---|---|---|---|---|---|---|
| 0xXX | 0xXX | 0xXX | 0xXX | 0xXX | 0xXX | 0xXX | 0xXX | 0xXX | Stop |

[Received data item]

・PTAT:

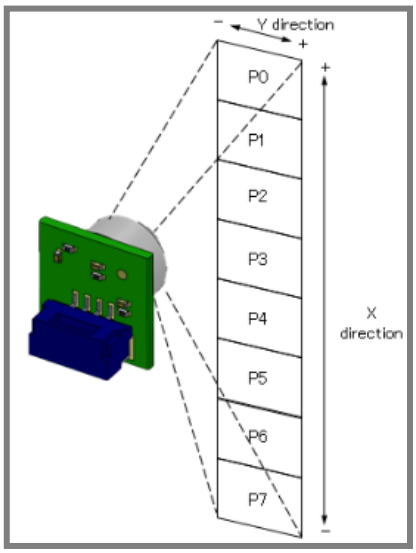
Reference temperature data inside the sensor. 10-fold temperature value [°C] with a 16-bit signed integer (two's complement).

(Calculation example)
When PTAT (Low): 0x20, PTAT (High): 0x01,
0x0120 -> 288 -> 288/10 = 28.8 (°C) at Int16

・P0 to P7:

Temperature data for each pixel. See the figure for the layout. 10-fold temperature value [°C] with a 16-bit signed integer (two's complement)

(Calculation example)
When P0 (Low): 0x87, P0 (High): 0xFE,
0xFE87 -> -121 -> -121/10 = -12.1 (°C) at Int16



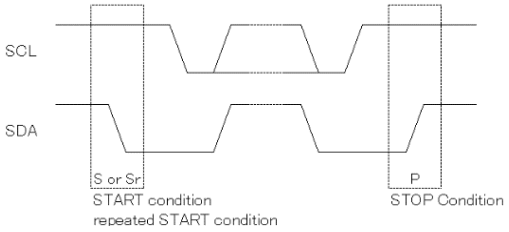

・PEC:

Packet error check code. PEC is data for checking for errors using the CRC-8 method, and is added at the end of the communication output. The target range of PEC is from Slave Address + Write to P7 (High) data. PEC values are created using each 1-byte value. This PEC value can be used by users to detect communication failures and improve data reliability.
(See SMBus specifications for details)

(3) I²C access protocol

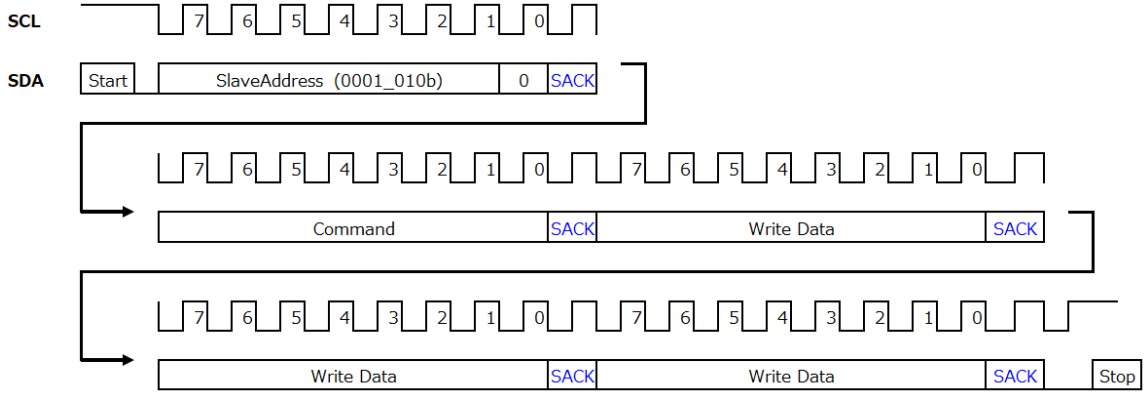
The terms of the I²C access protocol are as follows.

| | |
|---|---|
| “S” | : Start Condition |
| “Sr” | : Repeat Start Condition |
| “P” | : Stop Condition |
| “W/R” | : Write (Lo) / Read (Hi) |
| “SACK” | : Acknowledge by Slave |
| “MACK” | : Acknowledge by Master |
| “MNACK” | : No-acknowledge by Master |




I²C write access protocol

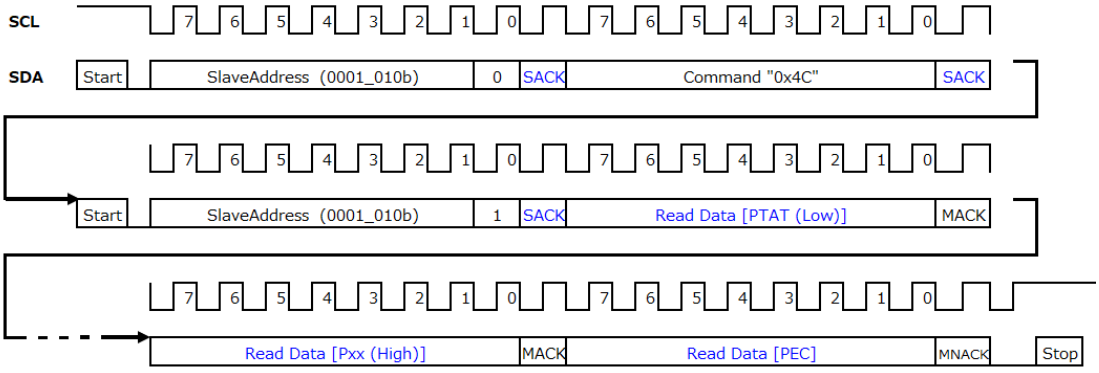
After the Start Condition, send the data with the write signal (=”0” at bit0) added to the slave address (bit7 to 1) and set it to write mode. Then, continue to send write data and set it to Stop Condition.




***1. Black characters : Master → Slave**, **Blue characters : Slave → Master**

I²C read access protocol

First, send a command in write mode. Next, set to Repeat Start Condition and then switch to read mode to read the data.




***1. Black characters : Master → Slave**, **Blue characters : Slave → Master**

If the SDA or SCL continues low input for the following amount of time, the communication will time out and the sensor will stop.

・D6T-8L-09: 70msec

If the sensor determines that the communication has timed out, NACK is returned at the Write access, but the read value will be FFFFh at the Read access. PEC is recommended for checking data as it is capable of judging that the read value is abnormal.

(4) Sample Code
Sample code for Raspberry Pi can be found in the URL below.
https://github.com/omron-devhub/d6t-2jcieev01-raspberrypi/blob/master/d6t-8l.c

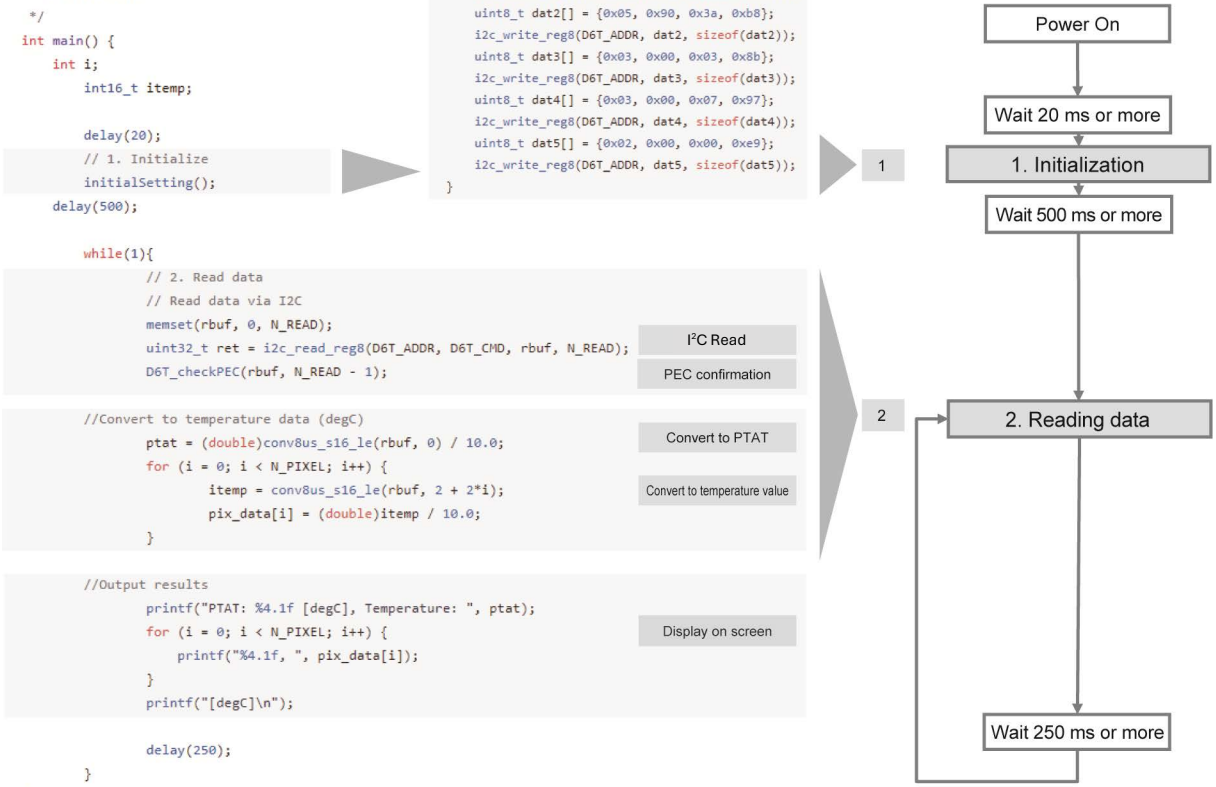
Sample code for Arduino can be found in the URL below.
https://github.com/omron-devhub/d6t-2jcieev01-arduino/blob/master/examples/d6t-8l/d6t-8l.ino

The structure of the sample code is as follows.

```
/** <!-- main - Thermal sensor {{{1 -->
 * 1. Initialize.
 * 2. Read data
 */
int main() {
    int i;
        int16_t itemp;

        delay(20);
        // 1. Initialize
        initialSetting();
    delay(500);

        while(1){
                // 2. Read data
                // Read data via I2C
                memset(rbuf, 0, N_READ);
                uint32_t ret = i2c_read_reg8(D6T_ADDR, D6T_CMD, rbuf, N_READ);
                D6T_checkPEC(rbuf, N_READ - 1);

        //Convert to temperature data (degC)
                ptat = (double)conv8us_s16_le(rbuf, 0) / 10.0;
                for (i = 0; i < N_PIXEL; i++) {
                        itemp = conv8us_s16_le(rbuf, 2 + 2*i);
                        pix_data[i] = (double)itemp / 10.0;
                }

        //Output results
                printf("PTAT: %4.1f [degC], Temperature: ", ptat);
                for (i = 0; i < N_PIXEL; i++) {
                    printf("%4.1f, ", pix_data[i]);
                }
                printf("[degC]\n");

                delay(250);
        }
}
```

```
void initialSetting(void) {
    uint8_t dat1[] = {0x02, 0x00, 0x01, 0xee};
    i2c_write_reg8(D6T_ADDR, dat1, sizeof(dat1));
    uint8_t dat2[] = {0x05, 0x90, 0x3a, 0xb8};
    i2c_write_reg8(D6T_ADDR, dat2, sizeof(dat2));
    uint8_t dat3[] = {0x03, 0x00, 0x03, 0x8b};
    i2c_write_reg8(D6T_ADDR, dat3, sizeof(dat3));
    uint8_t dat4[] = {0x03, 0x00, 0x07, 0x97};
    i2c_write_reg8(D6T_ADDR, dat4, sizeof(dat4));
    uint8_t dat5[] = {0x02, 0x00, 0x00, 0xe9};
    i2c_write_reg8(D6T_ADDR, dat5, sizeof(dat5));
}
```

I²C Read
PEC confirmation
Convert to PTAT
Convert to temperature value
Display on screen



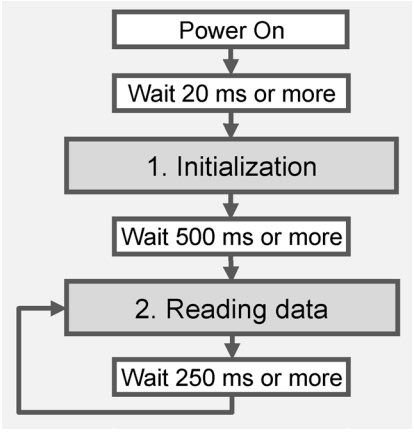
# Communication specifications D6T-8L-09H

(1) Outline of I²C Interface

| | |
|---|---|
| Slave Address | 7bit (0001_010b)<br>8bit (with R/W bit) expression: Read : 15h, Write : 14h |
| Data bit width | 8bit (MSB-first) |
| Clock speed | Max. 100kHz |
| Clock stretch supported | Not supported |

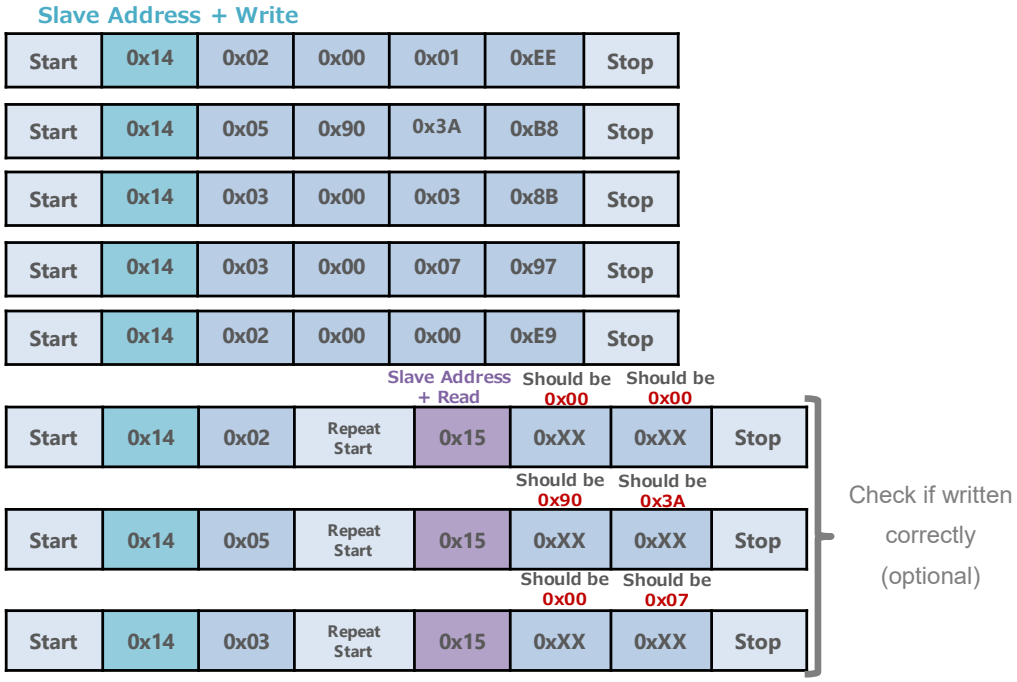
(2) Communication procedure

The procedure for communicating with the sensor is as shown below.
The sensor with standard specifications updates the measurement data within every 250 ms. This is repeated regardless of the communication, and the measurement timing cannot be controlled externally.




1. Initialization

Send and receive commands as shown below.



2. Reading data

Send the following commands to acquire the data. The received data has 19 bytes.

| | Slave Address +Write | | | Slave Address + Read | PTAT(Low) | PTAT(High) |
|---|---|---|---|---|---|---|
| Start | 0x14 | 0x4C | Repeat Start | 0x15 | 0xXX | 0xXX |

| P0(Low) | P0(High) | P1(Low) | P1(High) | P2(Low) | P2(High) | P3(Low) | P3(High) |
|---|---|---|---|---|---|---|---|
| 0xXX | 0xXX | 0xXX | 0xXX | 0xXX | 0xXX | 0xXX | 0xXX |

| P4(Low) | P4(High) | P5(Low) | P5(High) | P6(Low) | P6(High) | P7(Low) | P7(High) | PEC | |
|---|---|---|---|---|---|---|---|---|---|
| 0xXX | 0xXX | 0xXX | 0xXX | 0xXX | 0xXX | 0xXX | 0xXX | 0xXX | Stop |

[Received data item]

・PTAT:

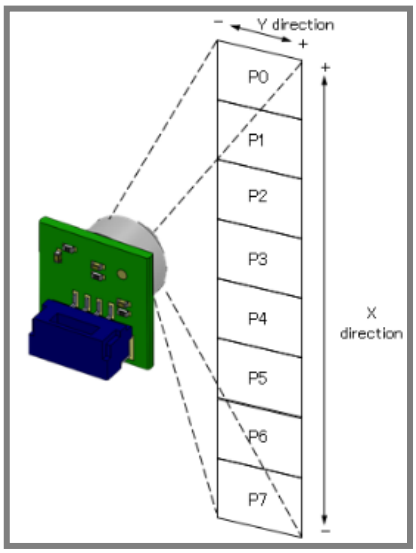
Reference temperature data inside the sensor. 10-fold temperature value [°C] with a 16-bit signed integer (two's complement).

(Calculation example)
When PTAT (Low): 0x20, PTAT (High): 0x01,
0x0120 -> 288 -> 288/10 = 28.8 (°C) at Int16

・P0 to P7:

Temperature data for each pixel. See the figure for the layout. 5-fold temperature value [°C] with a 16-bit signed integer (two's complement)

(Calculation example)
When P0 (Low): 0x30, P0 (High): 0x02,
0x0230 -> 560 -> 560/5 = 112.0 (°C) at Int16



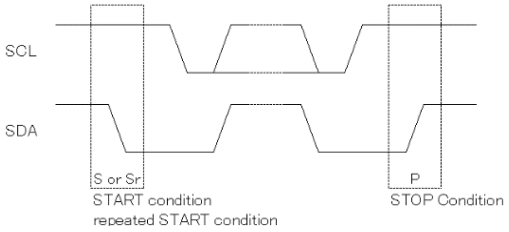

・PEC:

Packet error check code. PEC is data for checking for errors using the CRC-8 method, and is added at the end of the communication output. The target range of PEC is from Slave Address + Write to P7 (High) data. PEC values are created using each 1-byte value. This PEC value can be used by users to detect communication failures and improve data reliability.
(See SMBus specifications for details)

(3) I²C access protocol

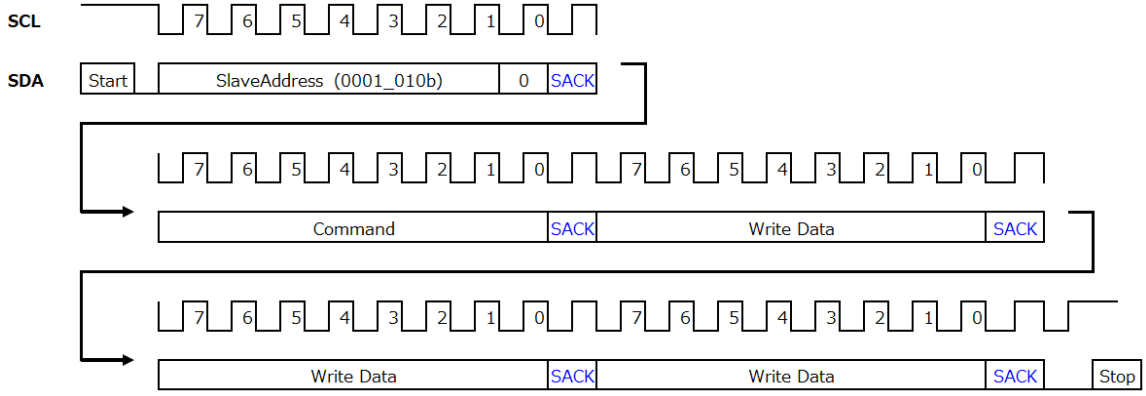
The terms of the I²C access protocol are as follows.

| | |
|---|---|
| “S” | : Start Condition |
| “Sr” | : Repeat Start Condition |
| “P” | : Stop Condition |
| “W/R” | : Write (Lo) / Read (Hi) |
| “SACK” | : Acknowledge by Slave |
| “MACK” | : Acknowledge by Master |
| “MNACK” | : No-acknowledge by Master |




I²C write access protocol

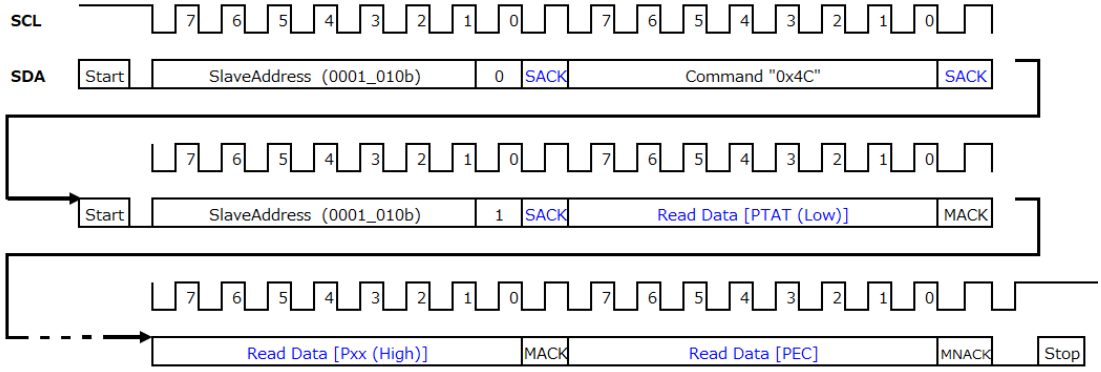
After the Start Condition, send the data with the write signal (=”0” at bit0) added to the slave address (bit7 to 1) and set it to write mode. Then, continue to send write data and set it to Stop Condition.




***1. Black characters : Master → Slave**, **Blue characters : Slave → Master**

I²C read access protocol

First, send a command in write mode. Next, set to Repeat Start Condition and then switch to read mode to read the data.




***1. Black characters : Master → Slave**, **Blue characters : Slave → Master**

If the SDA or SCL continues low input for the following amount of time, the communication will time out and the sensor will stop.

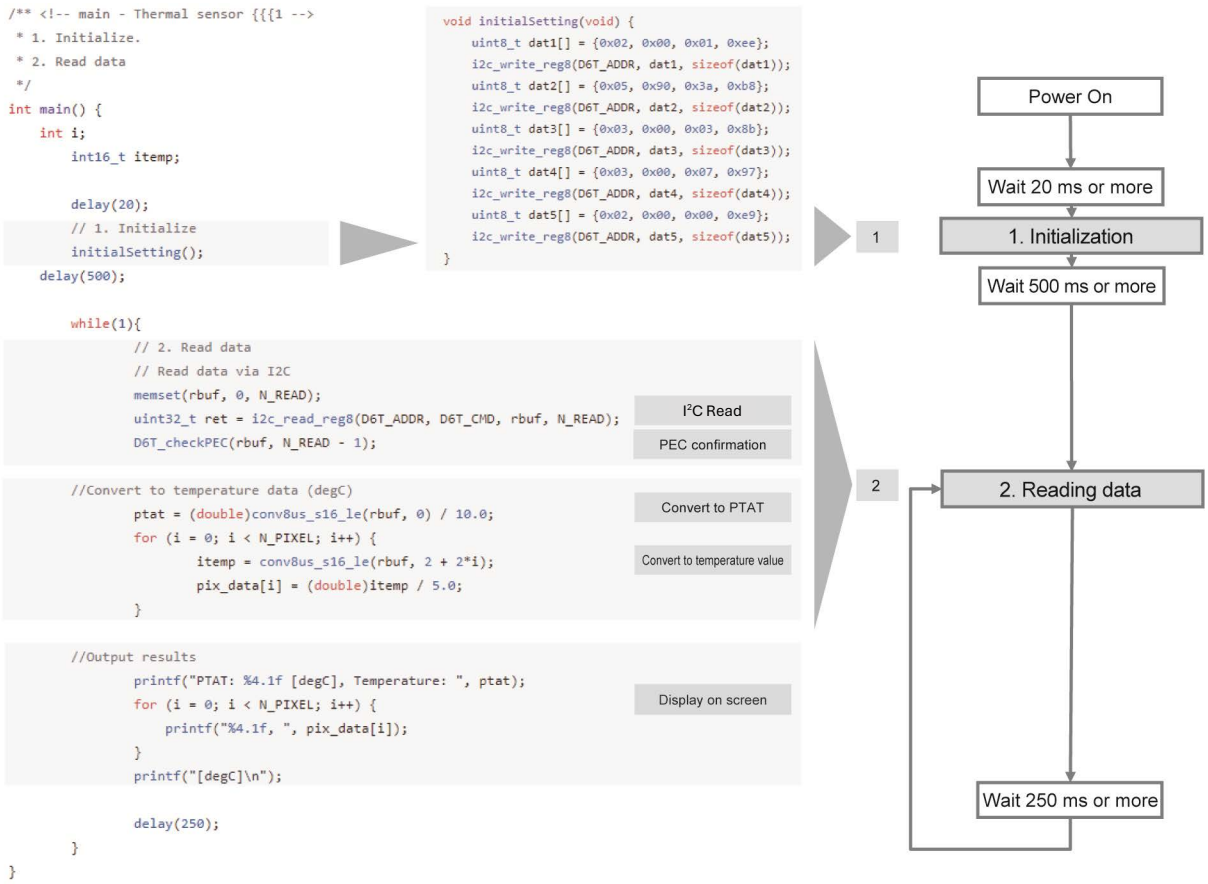
・D6T-8L-09H: 70msec

If the sensor determines that the communication has timed out, NACK is returned at the Write access, but the read value will be FFFFh at the Read access. PEC is recommended for checking data as it is capable of judging that the read value is abnormal.

(4) Sample Code
Sample code for Raspberry Pi can be found in the URL below.
https://github.com/omron-devhub/d6t-2jcieev01-raspberrypi/blob/master/d6t-8lh.c

Sample code for Arduino can be found in the URL below.
https://github.com/omron-devhub/d6t-2jcieev01-arduino/blob/master/examples/d6t-8lh/d6t-8lh.ino

The structure of the sample code is as follows.



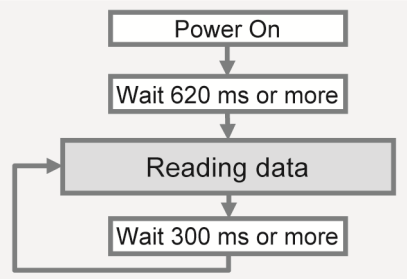
# Communication specifications
# D6T-44L-06
# D6T-44L-06H

(1) Outline of I²C Interface

| | |
|---|---|
| Slave Address | 7bit (0001_010b)<br>8bit (with R/W bit) expression: Read : 15h, Write : 14h |
| Data bit width | 8bit (MSB-first) |
| Clock speed | Max. 100kHz |
| Clock stretch supported | Supported |

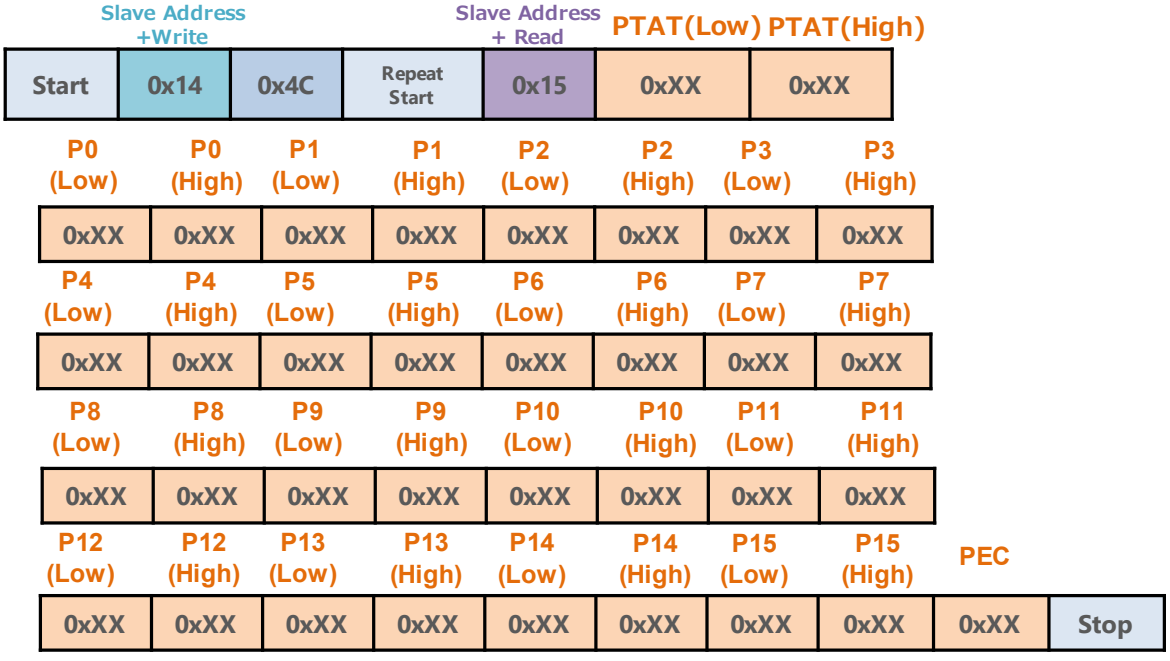
(2) Communication procedure

The procedure for communicating with the sensor is as shown below.
The sensor with standard specifications updates the measurement data within every 300 ms. This is repeated regardless of the communication, and the measurement timing cannot be controlled externally.




Reading data

Send the following commands to acquire the data. The received data has 35 bytes.

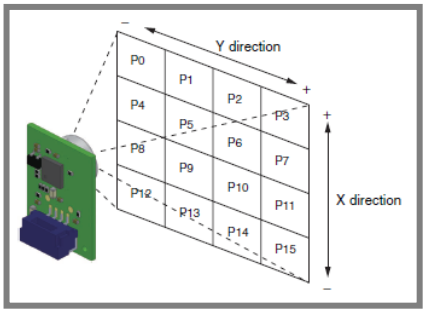


[Received data item]

・PTAT:

Reference temperature data inside the sensor. 10-fold temperature value [°C] with a 16-bit signed integer (two's complement).

(Calculation example)
When PTAT (Low): 0x20, PTAT (High): 0x01,
0x0120 -> 288 -> 288/10 = 28.8 (°C) at Int16




・P0 to P15:

Temperature data for each pixel. See the figure for the layout. 10-fold temperature value [°C] with a 16-bit signed integer (two's complement)

(Calculation example)
When P0 (Low): 0x87, P0 (High): 0xFE,
0xFE87 -> -121 -> -121/10 = -12.1 (°C) at Int16

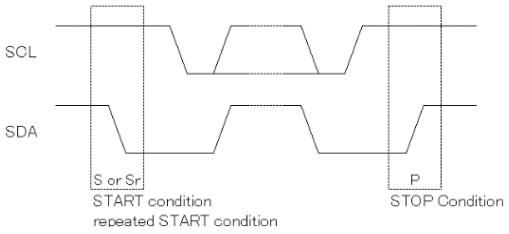
・PEC:

Packet error check code. PEC is data for checking for errors using the CRC-8 method, and is added at the end of the communication output. The target range of PEC is from Slave Address + Write to P15 (High) data. PEC values are created using each 1-byte value. This PEC value can be used by users to detect communication failures and improve data reliability.
(See SMBus specifications for details)

(3) I²C access protocol

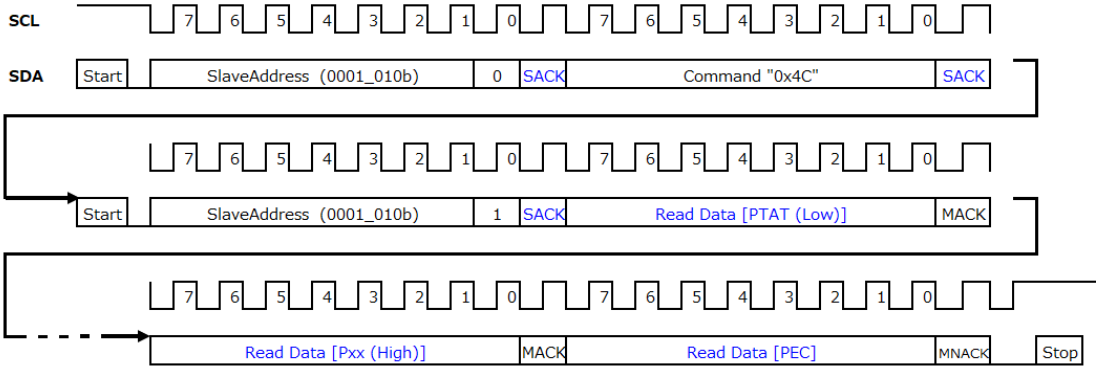
The terms of the I²C access protocol are as follows.

| | |
|---|---|
| “S” | : Start Condition |
| “Sr” | : Repeat Start Condition |
| “P” | : Stop Condition |
| “W/R” | : Write (Lo) / Read (Hi) |
| “SACK” | : Acknowledge by Slave |
| “MACK” | : Acknowledge by Master |
| “MNACK” | : No-acknowledge by Master |




I²C read access protocol

First, send a command in write mode. Next, set to Repeat Start Condition and then switch to read mode to read the data.




***1. Black characters : Master → Slave**, **Blue characters : Slave → Master**

If the SDA or SCL continues low input for the following amount of time, the communication will time out and the sensor will stop.

・D6T-44L-06 / D6T-44L-06H: 1sec

If the sensor determines that the communication has timed out, NACK is returned at the Write access, but the read value will be FFFFh at the Read access. PEC is recommended for checking data as it is capable of judging that the read value is abnormal.

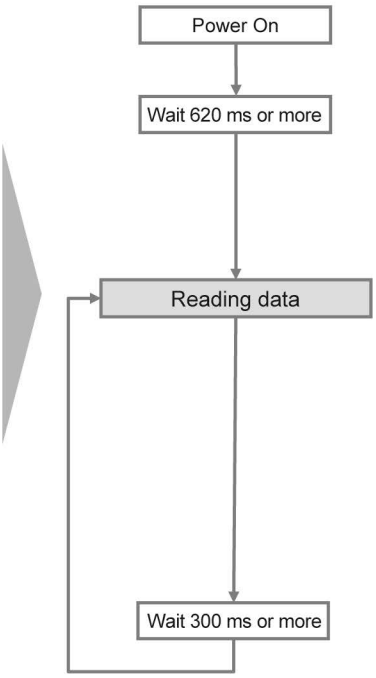
(4) Sample Code
Sample code for Raspberry Pi can be found in the URL below.
https://github.com/omron-devhub/d6t-2jcieev01-raspberrypi/blob/master/d6t-44l.c

Sample code for Arduino can be found in the URL below.
https://github.com/omron-devhub/d6t-2jcieev01-arduino/blob/master/examples/d6t-44l/d6t-44l.ino

The structure of the sample code is as follows.

```
/** <!-- main - Thermal sensor {{{1 -->
 * Read data
 */
int main() {
    int i;
        int16_t itemp;

        delay(620);

        while(1){
                // Read data via I2C
                memset(rbuf, 0, N_READ);
                uint32_t ret = i2c_read_reg8(D6T_ADDR, D6T_CMD, rbuf, N_READ);
                D6T_checkPEC(rbuf, N_READ - 1);

        //Convert to temperature data (degC)
                ptat = (double)conv8us_s16_le(rbuf, 0) / 10.0;
                for (i = 0; i < N_PIXEL; i++) {
                        itemp = conv8us_s16_le(rbuf, 2 + 2*i);
                        pix_data[i] = (double)itemp / 10.0;
                }

        //Output results
                printf("PTAT: %4.1f [degC], Temperature: ", ptat);
                for (i = 0; i < N_PIXEL; i++) {
                    printf("%4.1f, ", pix_data[i]);
                }
                printf("[degC]\n");

                delay(300);
        }
}
```

I²C Read
PEC confirmation
Convert to PTAT
Convert to temperature value
Display on screen

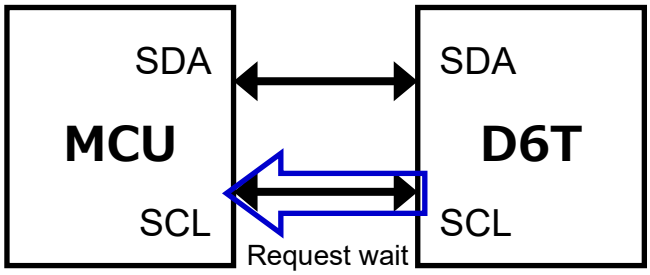


(5) Clock stretch

Clock stretch is a function that allows the slave to extend the SCL LOW period and make the master wait when the I²C slave cannot be processed in time. As the slave (D6T) has this function, the master (MCU) must also support this function.

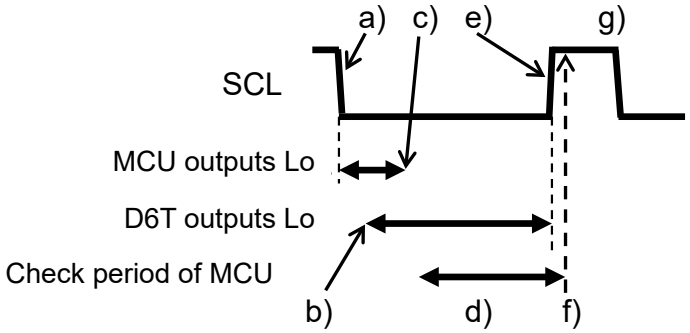
The I²C modules built into many MCUs have the automatic response function. Enable the clock stretch according to the specifications of the MCU used.

When using a software-based I²C library such as an MCU with no built-in I²C module, check whether there is a wait-compatible function beforehand. If this function doesn't exist, it is necessary to add a wait detection routine as shown below to the SCL output.




Wait detection routine

| I²C master | I²C slave (sensor) |
|---|---|
| a) Output Lo to SCL (per Ack) | Lo detection check for the SCL terminal |
| **(Fixed wait)**<br>**c) Change SCL output to Hi-Z**<br>**Change SCL terminal to input mode**<br>**d) Check if the SCL terminal changes to Hi**<br>**Waiting for checking (LOOP)** | b) Output Lo to SCL (wait request)<br>Waiting...<br>:<br>: |
| | Wait completed<br>e) Change SCL output to Hi-Z |
| **f) Check completed (Hi detection)**<br>**Change SCL terminal to output mode**<br>g) Move on to the next process | |




If setting the wait detection routine is difficult, add a wait time of 160 µsec at each Ack to the program.

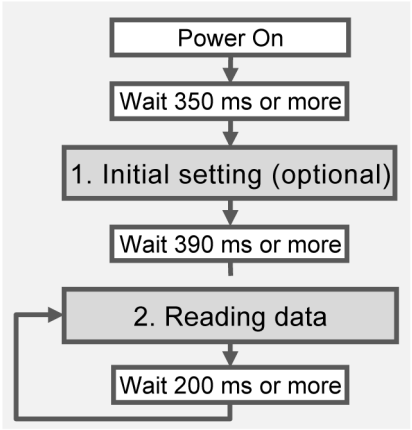
# Communication specifications D6T-32L-01A

(1) Outline of I²C Interface

| | |
|---|---|
| Slave Address | 7bit (0001_010b)<br>8bit (with R/W bit) expression: Read : 15h, Write : 14h |
| Data bit width | 8bit (MSB-first) |
| Clock speed | Max. 1000kHz (Fast-Mode Plus) |
| Clock stretch supported | Supported |

(2) Communication procedure

The procedure for communicating with the sensor is as shown below. The sensor with standard specifications updates the measurement data within every 200 ms. This is repeated regardless of the communication, and the measurement timing cannot be controlled externally.




1. Initial setting (optional)

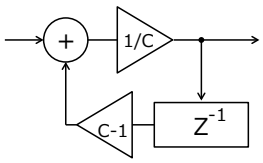
Send commands as shown below. If the settings are not changed, there is no need to send the initial settings. If the settings are not changed, the unit will operate with the default settings. Perform the initial settings 350 ms after the power is turned on. If the power is turned off, the changed settings will be deleted and the default settings will be used.

| | Slave Address + Write | Register Address | Write data | |
|---|---|---|---|---|
| Start | 0x14 | 0x01 | 0xXX | Stop |

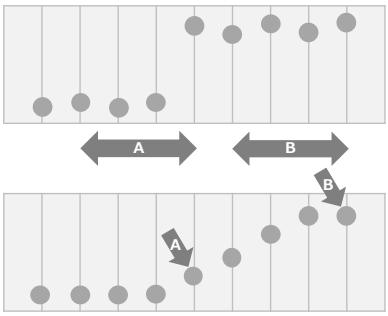
Below are the details of the registers.

| Address | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| 0x01 | IIR[3] | IIR[2] | IIR[1] | IIR[0] | Avg[3] | Avg[2] | Avg[1] | Avg[0] |
| Default | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

IIR[3:0]: IIR filter settings.
The coefficient setting of the IIR filter can be set to 0 (through) or 1 to 15.
If C is 4 times the set value,
then Y = ((C-1) x Yold + X) / C.
The higher the coefficient of the IIR filter, the smaller the noise, but the response time will be slower.



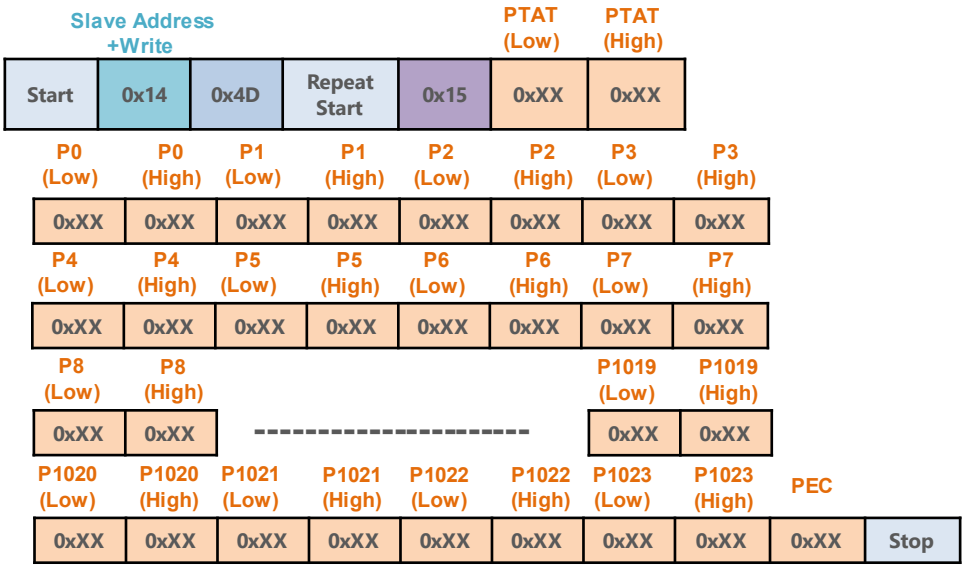

Avg[3:0]: Moving average setting.
The moving average can be set from 0 to 10.
0 and 1 are the same.
The default value is 4.
The higher the number of moving averages, the smaller the noise, but the response time will be slower.




The figure on the right is an example of the moving average. When the moving average is set to 4, the value will be the average of the past 4 data.

2. Reading data

Send the following commands to acquire the data. The received data has 2051 bytes.

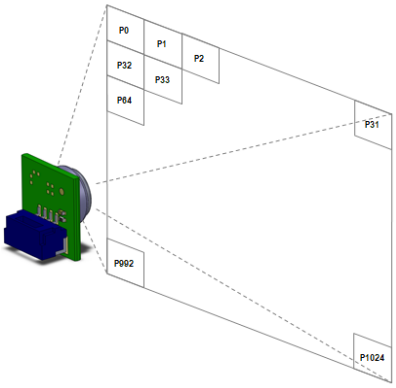



[Received data item]

・PTAT:

Reference temperature data inside the sensor. 10-fold temperature value [°C] with a 16-bit signed integer (two's complement).

(Calculation example)
When PTAT (Low): 0x20, PTAT (High): 0x01,
0x0120 -> 288 -> 288/10 = 28.8 (°C) at Int16




・P0 to P1023:

Temperature data for each pixel. See the figure for the layout. 10-fold temperature value [°C] with a 16-bit signed integer (two's complement)

(Calculation example)
When P0 (Low): 0x87, P0 (High): 0xFE,
0xFE87 -> -121 -> -121/10 = -12.1 (°C) at Int16

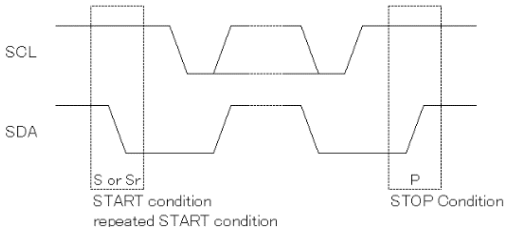
・PEC:

Packet error check code. PEC is data for checking for errors using the CRC-8 method, and is added at the end of the communication output. The target range of PEC is from PTAT (Low) to P1023 (High) data. PEC values are created using each 1-byte value. This PEC value can be used by users to detect communication failures and improve data reliability.
(See SMBus specifications for details)

(3) I²C access protocol

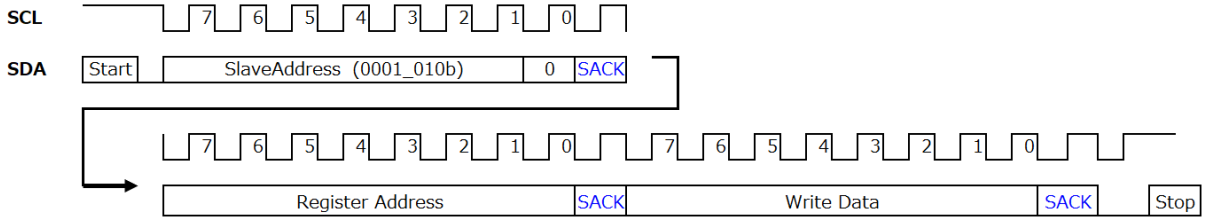
The terms of the I²C access protocol are as follows.

| | |
|---|---|
| “S” | : Start Condition |
| “Sr” | : Repeat Start Condition |
| “P” | : Stop Condition |
| “W/R” | : Write (Lo) / Read (Hi) |
| “SACK” | : Acknowledge by Slave |
| “MACK” | : Acknowledge by Master |
| “MNACK” | : No-acknowledge by Master |




I²C write access protocol

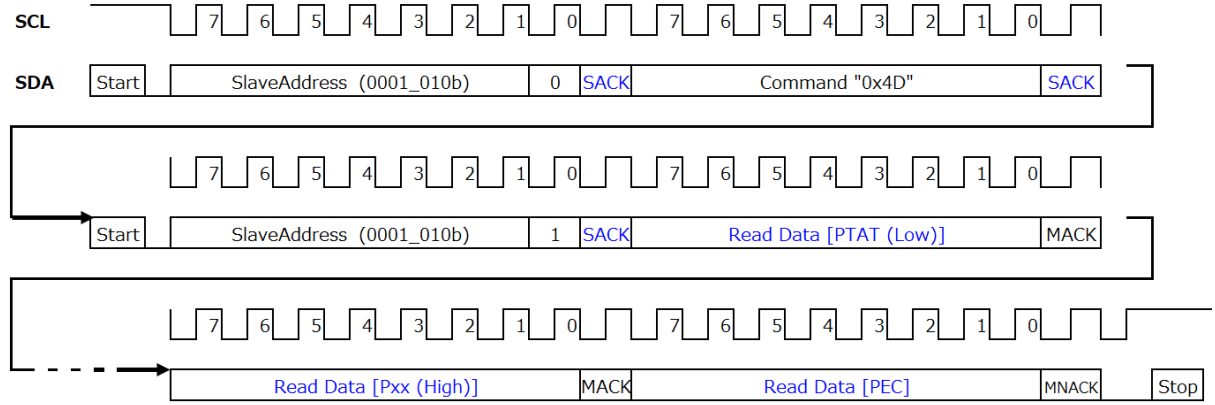
After the Start Condition, send the data with the write signal (=”0” at bit0) added to the slave address (bit7 to 1) and set it to write mode. Then, send write data and set to Stop Condition.




***1. Black characters : Master → Slave**, **Blue characters : Slave → Master**

I²C read access protocol

First, send a command in write mode. Next, set to Repeat Start Condition and then switch to read mode to read the data.




***1. Black characters : Master → Slave**, **Blue characters : Slave → Master**

If the SDA or SCL continues low input for the following amount of time, the communication will time out and the sensor will stop.

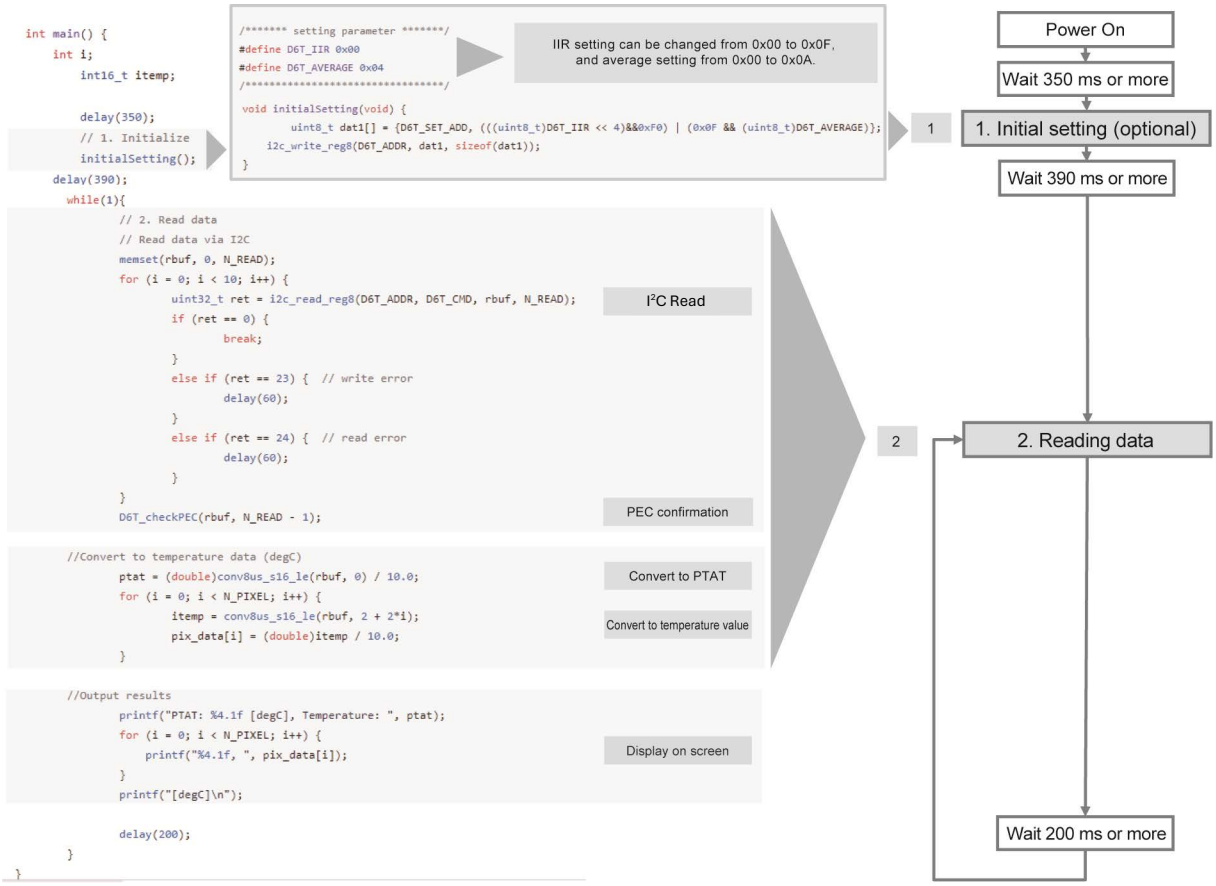
- D6T-32L-01A: 1 sec

If the sensor determines that the communication has timed out, NACK is returned at the Write access, but the read value will be FFFFh at the Read access. PEC is recommended for checking data as it is capable of judging that the read value is abnormal.

(4) Sample Code
Sample code for Raspberry Pi can be found in the URL below.
https://github.com/omron-devhub/d6t-2jcieev01-raspberrypi/blob/master/d6t-32l.c

Sample code for Arduino can be found in the URL below.
https://github.com/omron-devhub/d6t-2jcieev01-arduino/blob/master/examples/d6t-32l/d6t-32l.ino

The structure of the sample code is as follows.

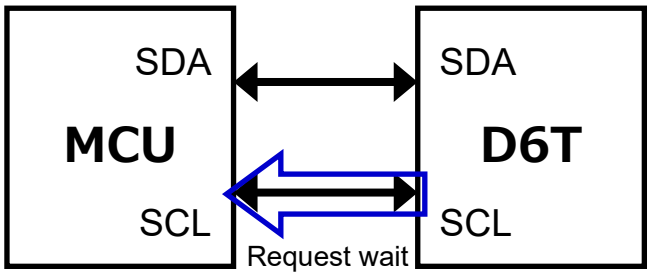


(5) Clock stretch

Clock stretch is a function that allows the slave to extend the SCL LOW period and make the master wait when the I²C slave cannot be processed in time. As the slave (D6T) has this function, the master (MCU) must also support this function.

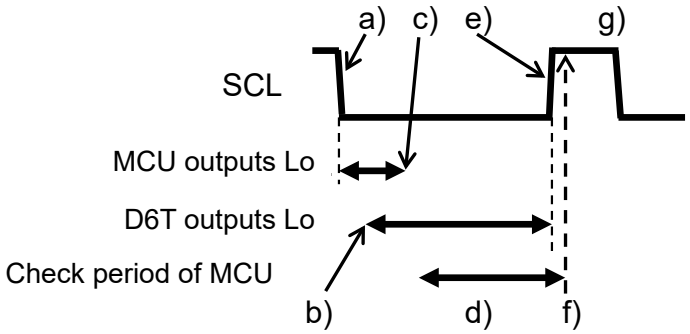
The I²C modules built into many MCUs have the automatic response function. Enable the clock stretch according to the specifications of the MCU used.

When using a software-based I²C library such as an MCU with no built-in I²C module, check whether there is a wait-compatible function beforehand. If this function doesn't exist, it is necessary to add a wait detection routine as shown below to the SCL output.




Wait detection routine

| I²C master | I²C slave (sensor) |
|---|---|
| a) Output Lo to SCL (per Ack) | Lo detection check for the SCL terminal |
| **(Fixed wait)**<br>**c) Change SCL output to Hi-Z**<br>**Change SCL terminal to input mode**<br>**d) Check if the SCL terminal changes to Hi**<br>**Waiting for checking (LOOP)** | b) Output Lo to SCL (wait request)<br>Waiting...<br>:<br>: |
| | Wait completed<br>e) Change SCL output to Hi-Z |
| **f) Check completed (Hi detection)**<br>**Change SCL terminal to output mode**<br>g) Move on to the next process | |




If setting the wait detection routine is difficult, add a wait time of 160 µsec at each Ack to the program.

Please check each region's Terms & Conditions by region website.


## OMRON Corporation
**Device & Module Solutions Company**

### Regional Contact

**Americas**
https://components.omron.com/us

**Europe**
https://components.omron.com/eu

**Asia-Pacific**
https://components.omron.com/ap

**China**
https://components.omron.com.cn

**Korea**
https://components.omron.com/kr

**Japan**
https://components.omron.com/jp





**Cat. No. A284-E1-05**
0424 (0718)