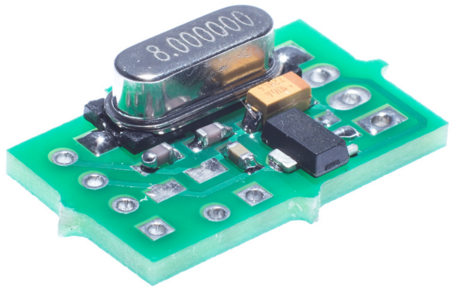
DATASHEET
SMT172 to I^{2}C INTERFACE BOARD

September 28, 2016

reference
smt172toiic

Highlights

- Accurate measurement in full SMT172 temperature range
- Measures the output of the SMT172 in 0.001 °C steps
- Both I^{2}C and PWM (SMT16030 like) output
- Board temperature range 0 – 75 °C
- Programmable secondary address, allowing for multiple sensors with interfaces on the same I^{2}C bus
- 3.3 V. regulator for powering the SMT172
- Powered by the I^{2}C bus itself
- Can be used for all housings of the SMT172



Introduction

Regularly Smartec is asked for a temperature sensor with an I^{2}C output. Because of the low power consumption of the SMT172 sensor itself, the self heating of the sensor is negligible, which contributes to its precision. An on chip integrated I^{2}C interface would produce too much heat. The only way to preserve the sensors precision and still have an I^{2}C interface, is to combine them on a tiny pcb. This board also does the calculations on the sampled duty-cycle and converts the result in ASCII data with an accuracy in the range of mK.

The board can be used in PWM mode (instead of I^{2}C mode), to meet compatibility requirements with some existing (obsolete) SMT160-30 applications.

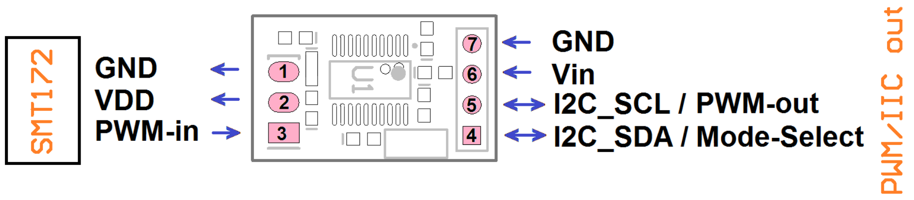
The output signal

The output signal is a standard compatible I^{2}C signal (slave) and the standard address is 111 (0x6F). A secondary address (7 bits) with the default value of 115 (0x73) is available. The value of this secondary address can be changed by the user. This process will be explained below.

Additional information

Board size	: 12 x 20 mm
Voltage	: 4 to 6 V
Primary I^{2}C address (7bit)	: 111
Secondary I^{2}C address (7bit)	: programmable, 0x11 to 0xEF

Pinout:



Mode Selection:

The SMT172 to interface can work both in Sensor-to-PWM mode and Sensor-to-I2C mode,

PIN 4	state	Working Mode
H	Internal pull-up	STM172 to I^{2}C
L	Internal pull-up	STM172 to PWM

In an I^{2}C bus, the SDA lines of all masters and slaves are connected to each other and so are all SCL lines. And each line is pulled up separately.

The interface has an internal 10k pull-up resistor for SDA and one for SCL. If you connect the board to a I^{2}C bus (or let the mode select pin float), the board will operate in I^{2}C mode. If you need the board to operate in PWM mode, pin 4 must be connected to GND.

Reading data from the I^{2}C interface:

Reading from the I^{2}C interface has to be done by a device capable of handling the connecting wires (GND, Vin, SCL and SDA). Microcontroller boards like Arduino and the Raspberry Pi are very capable of doing this with a minimum of effort for the user. Below you find three little Python programs to control the I^{2}C interface, that we tested on a Pi board. The first program reads data from a single interface. The second program shows you how to change the secondary address on that interface. The third program shows you how to read two interfaces connected to the same bus, one of which has been given an alternative secondary address, as shown in the second program.

Program 1

```
import smbus
from time import sleep
bus = smbus.SMBus(1)
pda = 0x6f # 111d primary device address
while True:
    a = bus.read_byte_data(pda, 0)
    b = bus.read_byte_data(pda, 1)
    c = bus.read_byte_data(pda, 2)
    temp = a*256*256+b*256+c-273150
    print temp
    sleep(1)
```

The actual temperature is calculated from the three consecutive byte values, which are contained in a number of registers (registers 0 till 2) at address 111d. Because the board gives the so called absolute temperature in milliKelvin, the number 273150 has to be subtracted to get the temperature in °C (times 1000)

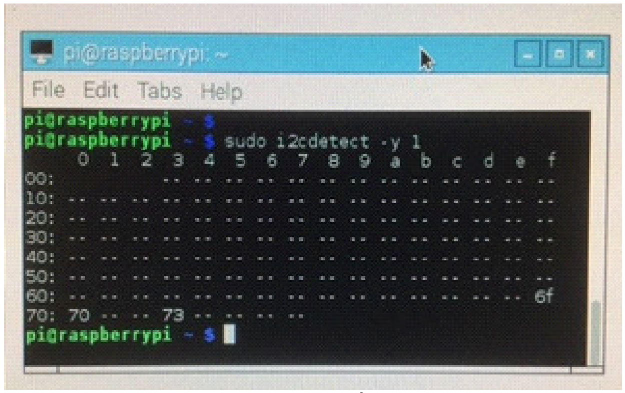
Program 2

```
import smbus
from time import sleep
bus = smbus.SMBus(1)
pda = 0x6F   # 111d =  primary device address
sda = 0x70   # 112d =  secondary device address
sda_write_value = sda * 2 # lsb is skipped!
bus.write_byte_data(pda, 10, sda_write_value)   #create secondary address
bus.write_byte_data(pda, 11, sda_write_value)   #write copy
print 'sda written'
```

The secondary address is changed from the original factory default value, which is 0x73, to the chosen value, in this case 0x70. Since the I^{2}C chip only uses the 7 most significant bits, the value of 0x70 has to be multiplied by 2 in order to shift one bit to the left, before it can be written to register 10 at the pda. This has to be done a second time to register 11 (Please don't ask us why!)

Now you can connect this interface with the new secondary address together with one that has not been changed (with factory default secondary address 0x73) both to the same I^{2}C bus and run program 3. Before you do so, you may want to run the following command from the Raspberry Pi command prompt:

sudo i2cdetect -y 1, which will give you an output like this:



```
pi@raspberrypi ~ $
pi@raspberrypi ~ $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- 6f
70: 70 -- -- 73 -- -- -- --
pi@raspberrypi ~ $
```

This command lists the available I^{2}C addresses on the bus. Here you see 0x6f, which is the primary address of all I^{2}C interfaces and 0x70 and 0x73, which now are the secondary addresses of the two interfaces. Now you can read both interfaces independently, as program 3 does.

Program 3

```
import smbus
from time import sleep
bus = smbus.SMBus(1)
pda  = 0x6f # 111d primary device address
sda1 = 0x70 # 112d secondary device address from the first  I2C interface
sda2 = 0x73 # 115d secondary device address from the second I2C interface
while True:
    a = bus.read_byte_data(sda1, 0)
    b = bus.read_byte_data(sda1, 1)
    c = bus.read_byte_data(sda1, 2)
    temp1 = a*256*256+b*256+c-273150
    a = bus.read_byte_data(sda2, 0)
    b = bus.read_byte_data(sda2, 1)
    c = bus.read_byte_data(sda2, 2)
    temp2 = a*256*256+b*256+c-273150
    print temp1, temp2
    sleep(1)
```

Note:

You can reprogram an I^{2}C interface being connected to the bus one at the time only, as long as they still have the factory settings, because they cannot be distinguished from each other. Once the interfaces have different secondary addresses, these secondary addresses can be used to change them as you like, because you can now communicate with each interface independently. As strange as it may sound, you can change the secondary address by writing to it.

ORDER CODE:

SMT172TOIIC