Application Note 001

PHPS I2C communication

Introduction:

This document describes the basic I2C communication process with APSP pressure sensors PHPS-3500, PHPS-4500, PHPS-5500, PHPS-5600 and PHPS-8500.

1. I2C protocol

1.1. General description

In I2C communication a serial data line (SDA) and a serial clock line (SCL) are required for communication between connected devices to the I2C bus. Both connected lines SDA and SCL are bidirectional lines, which are connected to supply voltage with pull-up resistors (see application circuit in Figure 6). As seen from figure 1 there can be more slave devices (up to 127) connected to the I2C bus, which is limited with the number of the 7-bit slave address. The I2C bus is free when both connection lines are HIGH and can be put LOW by devices connected to the I2C bus.

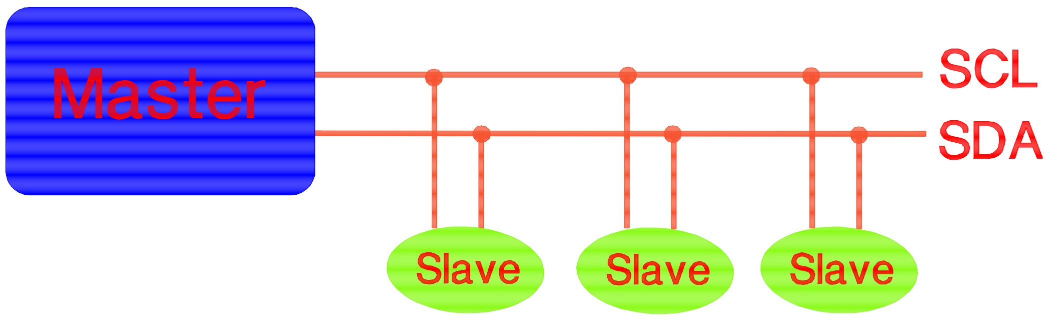


Figure 1: I2C communication example

I2C communication behaves as a Master-Slave principle (see figure 2), where the master leads the communication and asks for certain information from the slave. The master device generates the clock (SCL) and generates START & STOP command for data transition.

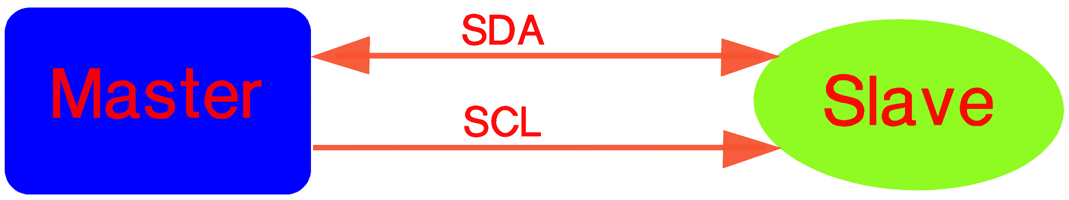


Figure 2: Master-Slave principle

Masters and slaves can act as a transmitter or as a receiver depending on the information which needs to be sent or read. The transmitter is a device that sends data to the I2C bus ("master transmitter" normally sends requests to the slave, when "slave transmitter" normally replies to the master by sending requested information). The receiver is a device that receives data from the I2C bus.

I2C standard protocol is presented in figure 3.

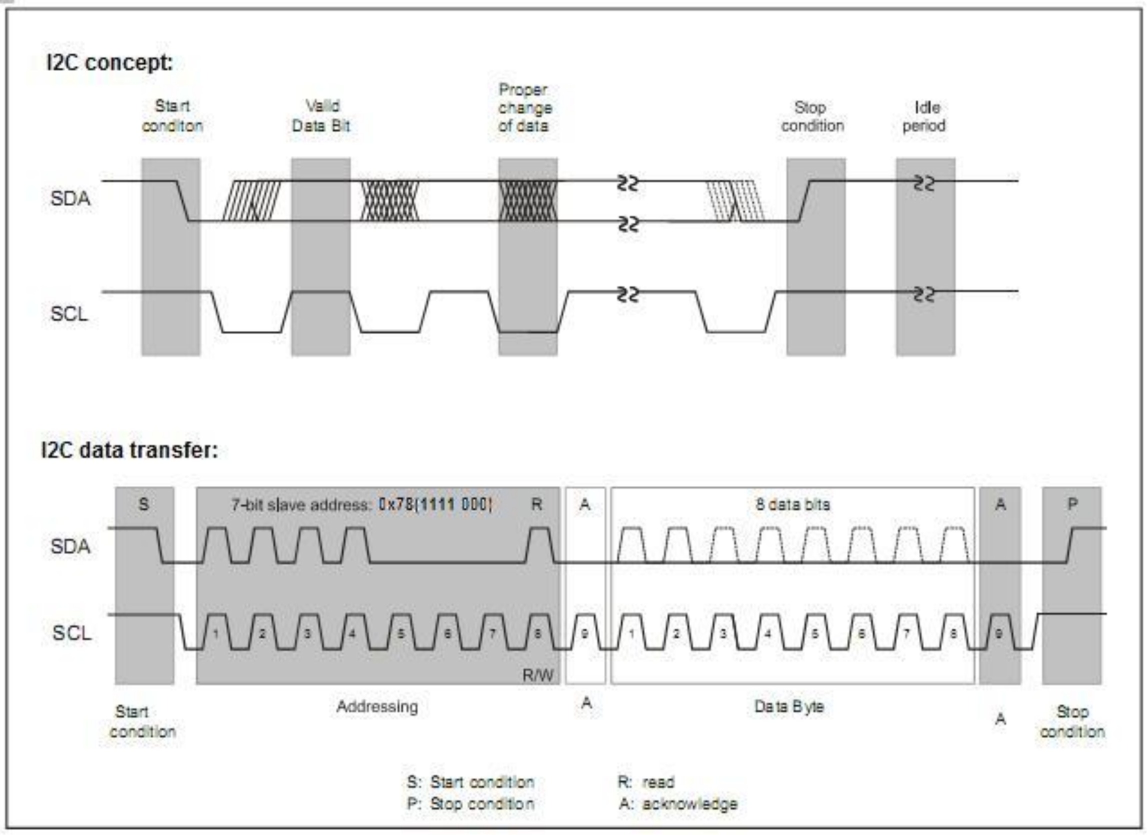


Figure 3: I2C standard communication protocol

1.2. Communication phases

- **Buss free – Idle state:** When a bus is free both lines SCL and SDA are pulled up – HIGH.
- **START condition (S):** Each data transfer starts with the start condition, which is always sent by the master. This start condition acts as a signal to all I2C connected devices giving information that there will be something transmitted. A start condition is defined as the transition of HIGH to LOW on the SDA line when the SCL line is HIGH (see figure 3).

- **STOP condition (P):** Each data transfer stops with a stop condition, which is also generated by a master when a data transfer has finished. The stop condition is defined as the transition of LOW to HIGH on the SDA line when the SCL line is HIGH (see figure 3).
- **Valid data:** Data is always transmitted in bytes (8 bits) starting with the MSB (most significant bit). One data bit is transferred with each clock pulse. Transmitted data are valid (after generating start condition) only during the HIGH period of the clock and data can changes can be done during the LOW period of the clock (see figure 3).
- **Acknowledge (A):** Each sent byte needs to be followed with the acknowledge bit generated from the receiver that corrects data has been received. Acknowledge means also that the device can continue with further data transfer. For that purpose, the master must generate an extra clock pulse. Transmitter releases clock HIGH during acknowledge clock pulse, if not then further byte will not be sent.
- **Slave address:** After the start condition master sends addressing byte - slave address to define with which slave device he wants to communicate. This addressing byte includes a 7-bit slave address (up to 128 devices) + 1 R/W bit (data direction bit). If the R/W bit is set to “0” (W) then the master wishes to transmit data to the selected slave. If the R/W bit is set to “1” (R) then the master request data from the slave. The addressed slave answers with an acknowledge, all other slaves connected with the I2C-bus ignore this communication.

  APSP PHPS pressure sensors have a default programmed slave address to 0x78 (1111 000b). For connecting more slave devices to the I2C bus each connected device should have its own slave address (up to 128 devices).

1.3. I2C communication overview

In figure 3 is presented complete data transfer. After generating the start condition master also sends the slave address with data direction bit (R/W), which gives information about read or write transfer. Addressed slave replies to this always with acknowledge (A) bit first. Now the unlimited numbers of data (bytes) can be transferred (every transferred byte needs to be confirmed with acknowledge bit). This transfer can be stopped by the master with generating the stop condition. If the master wishes to communicate also with other slave address, it can generate also a second start condition without stopping the first one.

2. DIGITAL DATA TRANSFER ON I2C BUS

2.1. PRESSURE & TEMPERATURE DATA TRANSFER ON I2C BUS

Digital data transfer is presented in figure 4.

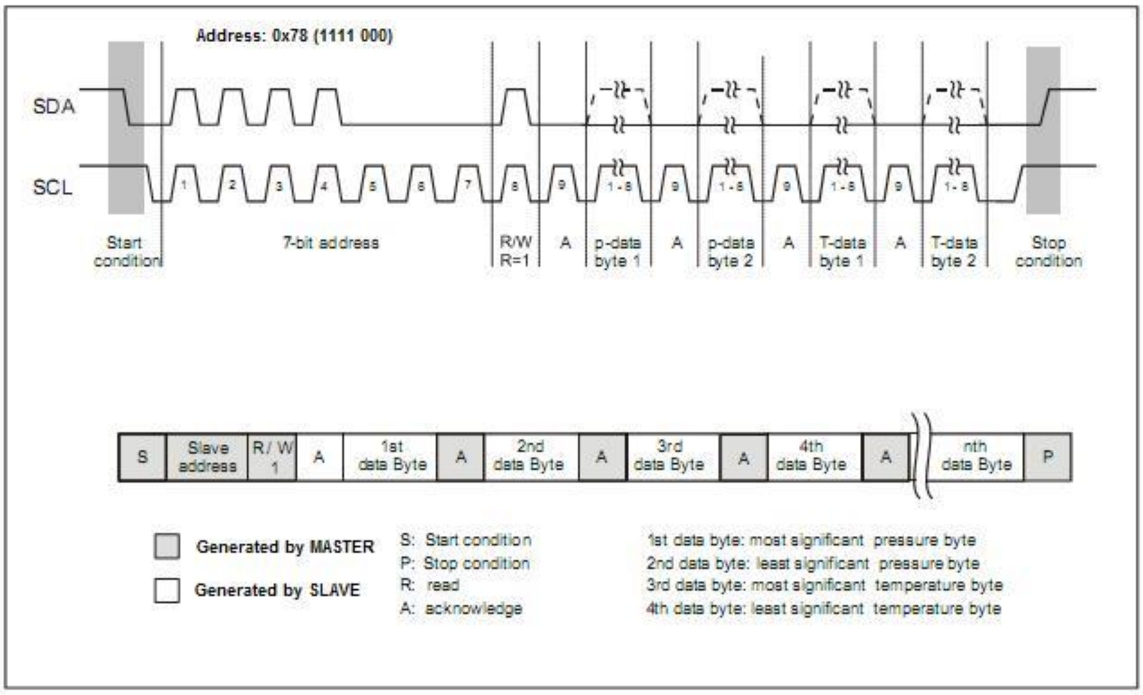


Figure 4: Digital pressure & temperature data transfer from APSP PHPS sensors

Pressure and temperature output signals from PHPS pressure sensors come as 15-bit values to the output register. Master, which would like to read this data, starts communication with the start condition. After that master sends a 7-bit slave address (factory default is 0x78) and data direction bit R/W (for read data R/W=”1”). Slave confirms this address with acknowledge (A) bit first and afterward sends desired data with bytes (8 bits): the first byte is a most significant byte for pressure value, the second byte is a least significant byte for pressure value, the third byte is a most significant byte for temperature value, the fourth byte is a least significant byte for temperature value. Master must confirm each received byte with acknowledge bit (see figure 4). Master can stop the data transfer by sending the stop condition or it can generate additional acknowledge bit after 4 receiving bytes of data (pressure and temperature) for continuous data receiving from slave (PHPS sensor).

2.1. Calculation pressure formula

Master receives pressure data as 15-bit values which can be converted to actual pressure data with pressure units in mbar using the simple formula below.

Definitions:

P= pressure (mbar)
Pmin= min pressure (mbar)
Pmax= max pressure (mbar)
D = digital pressure (counts)
Dmax = max digital pressure (counts)
Dmin = min digital pressure (counts)
S= sensitivity (count/mbar)

$$S = \frac{\mathrm{Dmax} - \mathrm{Dmin}}{\mathrm{Pmax} - \mathrm{Pmin}} \qquad P = \frac{\mathrm{D} - \mathrm{Dmin}}{\mathrm{S}} + \mathrm{Pmin}$$

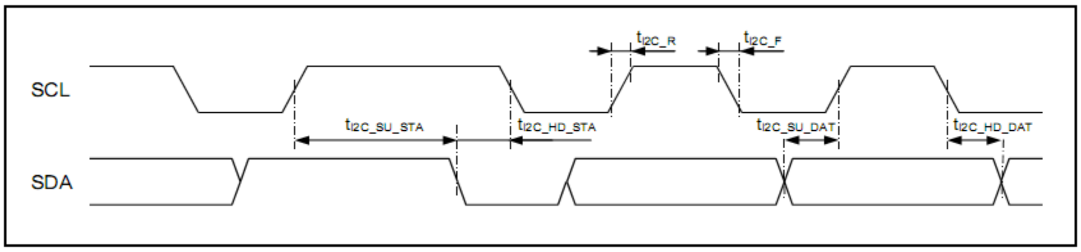
Example: For our PHPS-4500-350M pressure sensor with pressure range 0 to 350 mbar with analog output 0.5 to 4.5 V (equivalent digital output 3277 to 29491 counts) we measure a digital value of 7850 counts. Let’s calculate this value in pressure with pressure units in mbar:

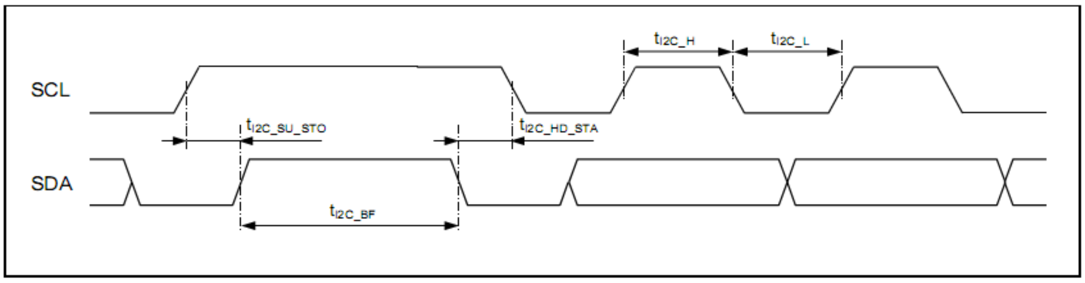
$$S = \frac{29491-3277}{350\mathrm{mbar}-0\mathrm{mbar}} = 74{,}90 \text{ counts/mbar}$$

$$P = \frac{7850-3277}{74{,}90} + 0\text{mbar} = 61{,}05 \text{ mbar}$$

Temperature values from digital temperature values are calculated in the same manner.

2.2. I2C Timings parameters





Nr.	Parameter	Symbol	min	typ	max	Unit	Conditions
1	SCL Clock frequency1	f_{SCL}			400	kHz	$f_{CLK} \geq 2MHz$
2	Bus free time betw. start and stop condition	t_{I2C_BF}	1.3			µs	
3	Hold time start condition	$t_{I2C\_HD\_STA}$	0.6			µs	
4	Setup time repeated start condition	$t_{I2C\_SU\_STA}$	0.6			µs	
5	Low period SCL/SDA	t_{I2C_L}	1.3			µs	
6	High period SCL/SDA	t_{I2C_H}	0.6			µs	
7	Data hold time	$t_{I2C\_HD\_DAT}$	0			µs	
8	Data setup time	$t_{I2C\_SU\_DAT}$	0.1			µs	
9	Rise time SCL/SDA	t_{I2C_R}			0.3	µs	
10	Fall time SCL/SDA	t_{I2C_F}			0.3	µs	
11	Setup time stop condition	$t_{I2C\_SU\_STO}$	0.6			µs	
12	Noise interception SDA	t_{I2C_NI}			50	ns	Spikes are suppressed

Figure 5: Timing I2C protocol

3. APPLICATION SHEME

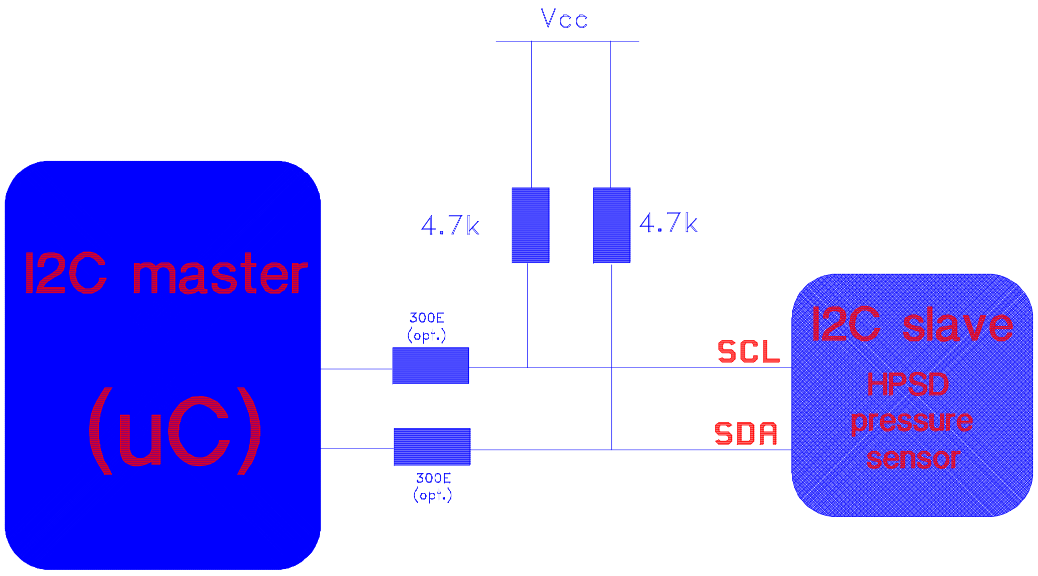


Figure 6: Application circuit

SCL and SDA lines need to be connected to a power supply via pull-up resistors as shown in figure 6. APSP recommends using 4.7k ohms resistors as pull-up resistors and 300 ohm resistors as serial resistors.

4. EXAMPLE PROGRAM CODE

Simple example code for pressure readings from APSP PHPS pressure sensor using Arduino is presented below.

```
#include <Wire.h>             // Include Arduino Wire library for I2C, URL: https://github.com/PaulStoffregen/Wire
#include <SoftwareSerial.h>  // Include softwareSerial library, URL: https://github.com/PaulStoffregen/SoftwareSerial

#define RxPin 0   // Define Serial communication pin Rx
#define TxPin 1   // Define Serial communication pin Tx
SoftwareSerial portOne(RxPin, TxPin);

#define SLAVE_ADDR 120  // Define Slave I2C Address , Slave address: 0x78

// Define pressure range
const float Pmin = 0;                                      // Max. pressure (mbar)
const float Pmax = 200.0;                                  // Min. pressure (mbar)
const uint32_t DPmin = 2482;                               // Max. digital pressure
const uint32_t DPmax = 22342;                              // Min. digital pressure
const float S_P = (DPmax - DPmin) / (Pmax - Pmin);        // Pressure sensitivity
// Define temperature range
const float Tmin = 0;                                      // Max. temperature (°C)
const float Tmax = 50.0;                                    // Min. temperature (°C)
const uint32_t DTmin = 8192;                               // Max. digital temperature
const uint32_t DTmax = 24576;                              // Min. digital temperature
const float S_T = (DTmax - DTmin) / (Tmax - Tmin);        // Temperature sensitivity

void setup() {
     Wire.begin();               // Initialize I2C communications as Master
     portOne.begin(9600);        // Setup serial monitor
     delay(100);                 // Wait 100ms
}

void loop() {
     byte rawdata[5];                        // Define array for raw data

     Wire.requestFrom(SLAVE_ADDR, 4);        // Request 4 bytes from the sensor (Slave)
     for (uint8_t i = 0; i < 4; i++) {
            if (Wire.available()) {
            rawdata[i] = Wire.read();        // Store 1 byte at a time to data array
            } else {
            rawdata[i] = 0;                  // Store value 0 if data no avaliable
            }
     }

  uint16_t x = (uint16_t)rawdata[0] << 8 | (uint16_t)rawdata[1];    // Convert pressure data from 2 bytes to unsigned integer
  uint16_t DPvalue = *(uint16_t*)&x;
  x = (uint16_t)rawdata[2] << 8 | (uint16_t)rawdata[3];             // Convert temperature data from 2 bytes to unsigned integer
  uint16_t DTvalue = *(uint16_t*)&x;

  float P = (DPvalue - DPmin) / S_P + Pmin;  // Calculate pressure value
  float T = (DTvalue - DTmin) / S_T + Tmin;  // Calculate temperature value

  portOne.println(P);           // Serial Monitor Pressure
  portOne.println(T);           // Serial Monitor Temperature
  delay(500);                   // Wait 0.5s
}
```

Revision History

Revision Letter	Date	Description / Changes
A	05.11.2010	Initial Creation
B	09.12.2021	Added family PHPS-8500

We are here for you. Addresses and Contacts.

Headquarter Switzerland:

Angst+Pfister Sensors and Power AG
Thurgauerstrasse 66
CH-8050 Zurich
Phone +41 44 877 35 00
sensorsandpower@angst-pfister.com

Office Germany:

Angst+Pfister Sensors and Power Deutschland GmbH
Edisonstraße 16
D-85716 Unterschleißheim
Phone +49 89 374 288 87 00
sensorsandpower.de@angst-pfister.com

Scan here and get an overview of personal contacts!

sensorsandpower.angst-pfister.com