I²C Communication Protocol

For PFLOW products with the I²C interface User Reference

Document No. PFLOW2001-AN-I2C
Issue date: 22.06.2022
Revision: VA. 1.1

Table of Contents

1 Overview 3
2 Pin Definitions 3
3 Bus Properties 3
3.1 Bus Idle (A) 4
3.2 Start Condition (B) 4
3.3 Stop Condition (C) 4
3.4 Data (D) 4
3.5 Acknowledge Bit 4
4 Device Address 5
5 Commands 6
5.1 Write to the sensor commands 6
5.2 Read from the sensor commands 6
5.3 I2C interface read/write sequences 7
6 Commands description 8
6.1 Set I2C Address (Command Code: 0x00A4) 8
6.1.1 Procedure 8
6.1.2 Note 8
6.1.3 Sample Code 9
6.2 Calibrate the offset of flow rate (Command Code: 0x00F0) 10
6.2.1 Procedure 10
6.2.2 Note 10
6.2.3 Sample code 10
6.3 Read the sensor SN (Command Code: 0x0030) 11
6.3.1 Procedure 11
6.3.2 Note 11
6.3.3 Sample code 12
6.4 Read flow rate (Command Code: 0x003A) 13
6.4.1 Procedure 13
6.4.2 Note 13
6.4.3 Sample code 14
7 References: 15

1 Overview

This document is a description of the I²C data communication protocols for PFLOW2001 products that are configured with this option. This document will also include some I²C bus basics, but it is not intended to have the detailed descriptions of I²C bus specifications for which please refer to the standard document such as that listed in the reference of this file.

2 Pin Definitions

Each product may have a different Pinout assignment, please refer to the corresponding datasheet.

3 Bus Properties

The I²C bus has two primary properties:

The data transfer starts only when the bus is idle.

During data transfer, the serial data line (SDA) must be kept steady whenever the serial clock line (SCL) is *high*. When the serial clock line is high, a change of the serial data line from *high* to *low* or from *low* to *high* corresponds to a start condition or a stop condition, respectively.

Fig. 3-1 The bus timing diagram:

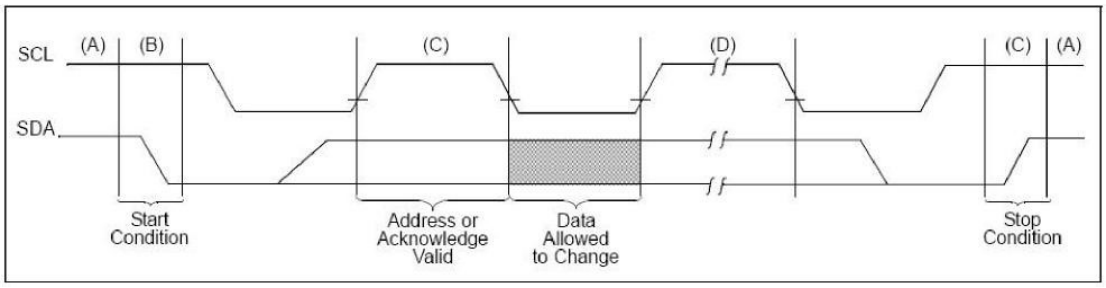


Fig. 3-1: Timing diagram of the I²C bus

3.1 Bus Idle (A)

At this status, both the serial data line and the serial clock line are *high*.

3.2 Start Condition (B)

When SCL is *high*, a change of SDA from high to low means a start condition that initiates the data transfer. A start condition must be in effect before any data transfer commands can be executed.

3.3 Stop Condition (C)

When SCL is high, a change of SDA from *low* to high means a stop condition that ends the data transfer. All the data transfer commands will not function after a stop condition is in place.

3.4 Data (D)

After the start condition, the serial data line must be kept steady when the serial clock line is *high*.

The serial data line can only be changed during the period when the serial clock line is *low*, and each data bit must associate a clock pulse.

Each data transfer will begin with a start condition and end after a stop condition. Every byte for the serial data line must be 8 bits long. The number of bytes that can be transmitted per transfer is unrestricted. Each byte has to be followed by an Acknowledge bit. The number of data bytes between a start condition and a stop condition will be decided by the bus master.

3.5 Acknowledge Bit

The transmit mode of the master can be initiated by sending a start bit and then followed by the slave address. Finally, it is followed by a single bit if the user likes to write(0) to or read(1) from the slave.

If the slave exists on the bus, then it will respond with acknowledge (ACK) bit (active *low* for acknowledgment) for that address. The master must provide an extra SCL pulse for each ACK bit.

The master then continues to be in a transmit or receive mode (according to the read/write bit being sent), and the slave continues to be in its complementary mode (receive or transmit, respectively).

For the address and the data bytes, the most significant bit will be sent first. The start bit is indicated by a *high-to-low* transition of SDA with SCL *high*; the stop bit is indicated by a *low-to-high* transition of SDA with SCL *high*.

If the master is to write into the slave, then it shall repeatedly send a byte while the slave will return an ACK bit (Fig. 3-2). (In this case, the master is at the master transmit mode and the slave is at the receive mode.)

If the master is to read from the slave, then it shall repeatedly receive a byte from the slave, and the master will send an ACK bit after each byte received except for the last one (Fig. 3-2). (In this case, the master is at the master receive mode and the slave is at the slave transmit mode).

The master can end the transmission with a stop bit, or it may send another start bit if it is to retain the control of the bus for another transfer (a "combined message").

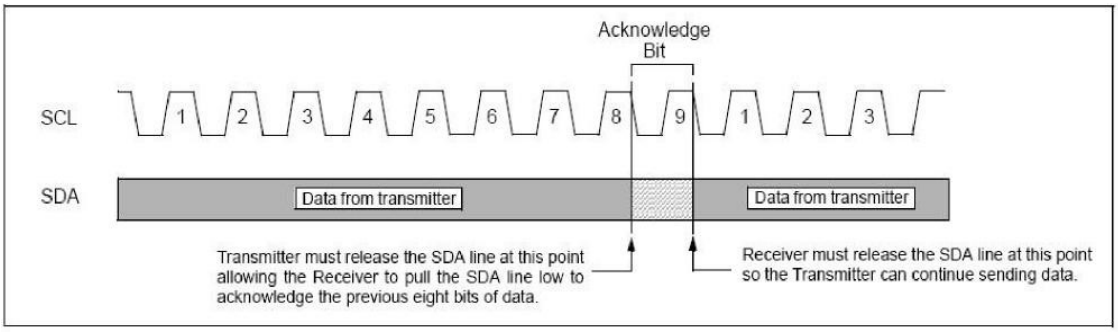


Fig. 3-2: Acknowledge Bit

4 Device Address

The I^2C interface adopts the slave-master mode with a transfer rate of 10kbit/s ~ 100kbit/s. The slave address is always an even number between 02h and FEh, i.e., a total of 127 available slave addresses are available. Multiple products can be configured as a multi-master multi-slave bus for multi-device communication, on the condition that each product (i.e., the slave device) has a unique address. The default address is 01h.

The address 00h is a reserved address for broadcasting. The user can use this address to perform a “write” operation on all the slave devices. However, it is prohibited to use 00h as any slave address. The user can only use the default address of a slave or reset its address through the access provided by the broadcasting address.

5 Commands

The communication mode is based on the command Interpretation mechanism. The data are accessed via predefined commands. A command consists of a command code and none, one, or multiple command values. If a command has only a command code without any command value, it is called a “pure” command; otherwise, it is called a “compound” command. An entire command must be continuously transmitted into the product by inquiry within a frame (between a start bit and a stop bit); otherwise, an error may occur.

The following are 4 predefined commands.

5.1 Write to the sensor commands

Command name	Command type	Command Code [2-Bytes]	Command value [2-Bytes] +[1Byte CRC]
Set I²C address	Write	00A4	New i2c address of the slave
Calibrate the offset, flow rate	Write	00F0	dummy data set , will be ignored

All command values written to the sensor shall be in 2-Byte format followed by 1 Byte CRC.

i.e.: reset flow rate offset:
Send to the sensor the command to reset offset: 0x00, 0xF0
Write to the sensor any 2-Byte fixed value i.e. 0xAA, 0x55
Follow it by CRC-8: 0x36

5.2 Read from the sensor commands

Command name	Command type	Command Code [2-Bytes]	Returned data sequence: [2-Bytes] +[1Byte CRC]
Read the sensor serial number	Read	0x0030	18 Bytes, HEX
Read flow rate	Read	0x003A	6 Bytes, HEX

The sensor returns the most significant Byte at first

All data from the sensor are transmitted in following sequence: 2-Bytes Data, 1 Byte CRC, 2-Bytes

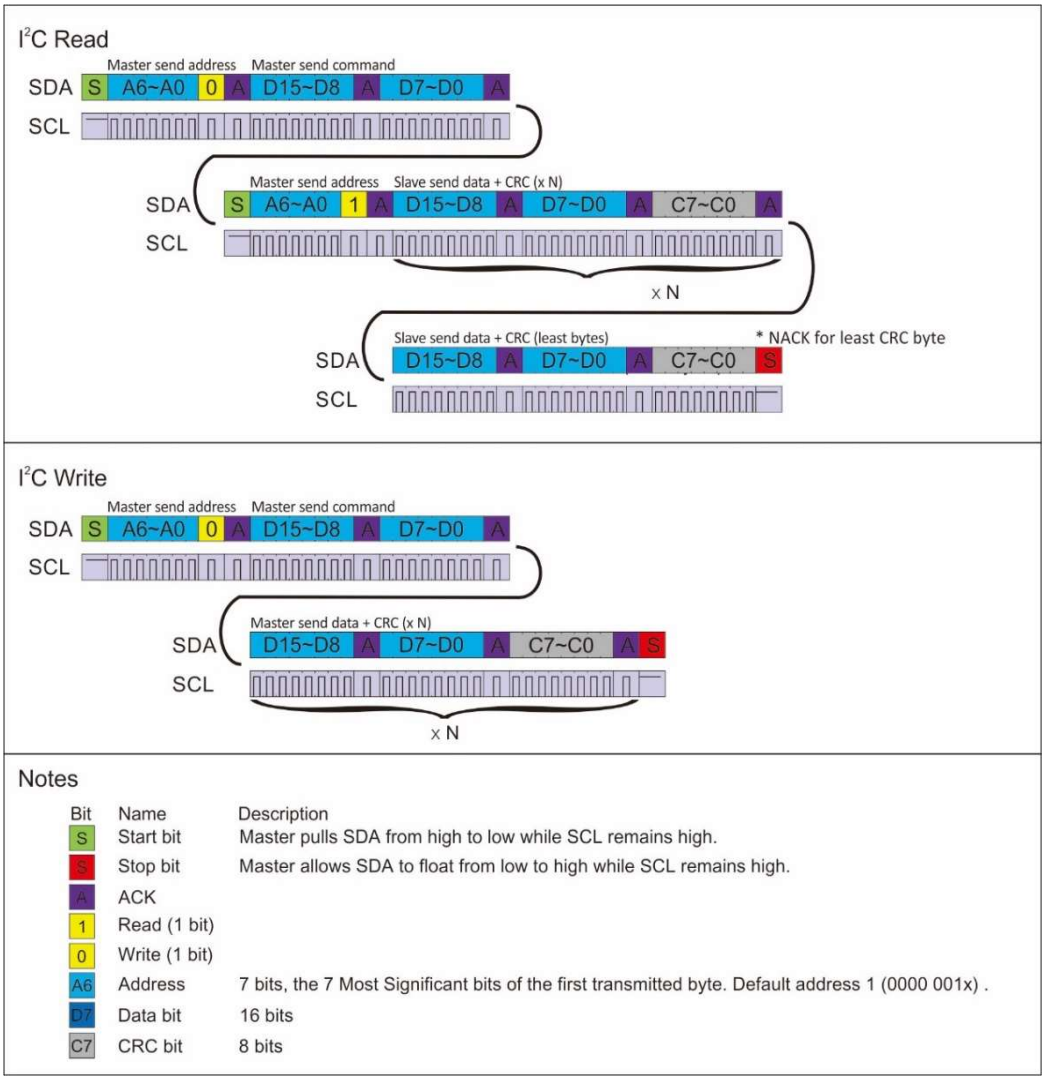
When reading from the sensor, after the transmission from master to slave is finished, the bus shall be kept active, by sending only a restart condition.

Sending a stop bit and releasing the bus will cause a communication error for the next communication and invalid response from the sensor (see Fig. 4 for details).

Please be aware that the most I2C communication libraries for common master drivers, per default send the stop bit after the communication, which is incorrect for the sensor.

Invalid response is easy to recognize as it has following sequence of 12 Bytes:
0-0-0-0-1-7-[6 Bytes of random data]
Receiving such a data package means, that the master send stop bit and released the bus after the read command.
Please refer to the sample code in chapter 7.3 Read the sensor serial number

5.3 I2C interface read/write sequences



6 Commands description

6.1 Set I2C Address (Command Code: 0x00A4)

This command is a “compound” command with one 2-Byte command value, that is an even number between 0002h and 00FEh, representing the new I^2C address of the sensor.

Command value to send to the sensor are 2 Bytes of Data (new i2c address) followed by 1 Byte CRC:

MSB		LSB
Byte-1	Byte-2	Byte-3
Data-1	Data-2	CRC

Data-1 will be always 0x00

6.1.1 Procedure

The I2C The procedure is as follows:

1. The master sends a start bit.
2. The master sends “slave address + write bit”, and the slave acknowledges it.
3. The master sends the command code in 2 Bytes “0x00A4”, and the slave acknowledges it.
4. The master sends the command value , the new address in 2 Bytes (e.g., I2C address 00 02), and the slave acknowledges it.
5. The master send 1 Byte CRC8 of the transmitted new I2C address and the slave acknowledges it.
6. The master sends a stop bit.

6.1.2 Note

This command is used for setting the I^2C address. For detailed information on the I^2C address, please refer to Section 4.

The 8-bit new address will be converted internally in the sensor microcontroller to the 7-bit one. It means, that if the new transmitted address is defined in the command as i.e. 0x0A, it will be converted to 7-bit and set as 0x05.

The best way to manage this, is to shift the desired address in the code 1-bit left before setting it as new address:

For example

The new desired address: 0x05

Shift the new address 1 bit left: 0x05 <<1 = 0x0A

And follow the standard procedure steps 4 & 5:

4. Send the command value: 0x00 0xA0
5. Send the CRC8 of 0x000A: 0x36

The new set I2C address will be 0x05

6.1.3 Sample Code

This is the code example for Wire library for Microchip Atmega microcontrollers:

```
#include <Wire.h>
int16_t old_address_PFLOW =0x01;            // defined old i2c address
int16_t new_address_PFLOW =0x05;            // new i2c address

new_address_PFLOW = new_address_PFLOW << 1; // shift the new address 1 bit
left


Wire.beginTransmission(old_address_PFLOW);    // transmition initialization

Wire.write(0x00);                           // writing to the sensor
Wire.write(0xA4);                           // commando to change i2c
address

Wire.write(0x00);                           // writing to the sensor
Wire.write(new_address_PFLOW);              // new shifted i2c address

Wire.write(0x36);                           // writing tot he sensor
                                            // CRC8 of the new address

Wire.endTransmission();                     // end of communication
                                            // if the function returned 0,
                                            // , the communication was
successfull
```

6.2 Calibrate the offset of flow rate (Command Code: 0x00F0)

This command is a “compound” command. Its command value can be any number since it acts as a dummy.

Command value to send to the sensor are 2 Bytes of Data (new i2c address) followed by 1 Byte CRC:

MSB		LSB
Byte-1	Byte-2	Byte-3
Data-1	Data-2	CRC

6.2.1 Procedure

1. The master sends a start bit;
2. The master sends “slave address + write bit”, and the slave acknowledges it;
3. The master sends the command code “0x00F0”, and the slave acknowledges it;
4. The master sends the command value (e.g., any number), and the slave acknowledges it;
5. The master sends 1-Byte of CRC8 of the command value
6. The master sends a stop bit.

6.2.2 Note

This command is used to calibrate the offset of the flow sensor and then use it for the subsequent measurement. Please ensure that there is NO flow in the flow channel before executing this command.
The command value in the step 4. will be ignored, it can be any 2-Byte hex value followed by 1-byte CRC8.

6.2.3 Sample code

This is the code example for Wire library for Microchip Atmega microcontrollers:

```
#include <Wire.h>
uint8_t address_PFLOW = 0x50;       //sensor i2c address definition

Wire.beginTransmission(address_PFLOW); // transmition initialization at the
                                       // sensor address

Wire.write(0x00);                    // write to the sensor commando
Wire.write(0xF0);                    // to flow rate offset reset

Wire.write(0xAA);                    // write to the sensor command value
Wire.write(0x55);                    // any random 2-Byte HEX value
Wire.write(0x36);                    // CRC8 of 0xAA55

Wire.endTransmission();              //end of communication with stop bit
                                     //if the function returned 0,
                                     //the communication was successfull
```

6.3 Read the sensor SN (Command Code: 0x0030)

This command is used to read the serial number of the product. The serial number is unique for each product and is the manufacturer ID number for tracking its factory default parameter settings.

This command is a “pure” command (which has no command values). The slave returns 18 Bytes of data in sequence 2 Bytes serial number data followed by 1 Byte CRC:

MSB									LSB
Byte-1	Byte-2	Byte-3	Byte-4	Byte-5	Byte-6	…	Byte-16	Byte-17	Byte-18
Data-1	Data-2	CRC	Data-3	Data-4	CRC	…	Data-11	Data-12	CRC

6.3.1 Procedure

1. The procedure is as follows:
2. The master sends a start bit;
3. The master sends “slave address + write bit”, and the slave acknowledges it;
4. The master sends the command code “0x0030”, and the slave acknowledges it;
5. The master sends a re-start bit (master keeps bus active). There is no stop-bit!
6. The master sends “slave address + read bit”, and the slave acknowledges it;
7. The slave returns the 1st byte of the series number, and the master acknowledges it;
8. The slave returns the 2nd byte of the series number, and the master acknowledges it;
9. The slave returns CRC8 of the 1st and 2nd Byte
10. …
11. The slave returns the 17th byte of the series number, and the master acknowledges it;
12. The slave returns the 18th byte of the series number, and the master doesn’t acknowledge it;
13. The master sends a stop bit.

6.3.2 Note

Do not release the bus after the read commando, no stop condition!
See chapter 5.2 Read from the sensor for details!

The serial number data package is 18-Bytes long sequence, where every 2 Bytes of data are followed by 1 Byte of CRC8.
The serial number itself consists of 12 Bytes HEX, which are representing ASCII code.
The serial number is predeceased and followed by “* *” (two stars), which are coded in ASCII as 2A 2A.
E.g.:
18-Bytes received from the sensor: 2A 2A FA 42 31 E6 52 33 BF 31 33 75 34 33 34 2A 2A FA
12 Bytes of data (without CRC): 2A 2A 42 31 52 33 31 33 34 33 2A 2A
Decoded ASCII: * * B 1 R 3 1 3 4 3 * *
Serial number: B1R31343

6.3.3 Sample code

This is the code example for Wire library for Microchip Atmega microcontrollers:

```
#include <Wire.h>

uint8_t address_PFLOW = 0x50;           // define sensor i2C address
uint8_t Readbyte[18];                   // define size of the buffer for
data
                                        //read out from the sensor


Wire.beginTransmission(address_PFLOW); // initialization of the transmition
Wire.write(0x00);                      //sending commando to the sensor
Wire.write(0x30);                      // to read serial number
Wire.endTransmission(false);           // communication finalizing without
stop bit


Wire.requestFrom(address_PFLOW, 18);    // requesting 18 bytes from the
sensor
delay(2);                               // 2 msec delay; sensor response
time
while Wire.available();


for (int i = 0; i < 18; i++) {      // reading out 18 byte of the data from
the
      Readbyte[i] = Wire.read();    // sensor to the buffer
    }
```

6.4 Read flow rate (Command Code: 0x003A)

This command is used to read the current instant flow rate. The flow rate consists of 4-Bytes value, which are combined with CRC to give a 6 Byte data package:

MSB LSB

Byte-1	Byte-2	Byte-3	Byte-4	Byte-5	Byte-6
Flow1	Flow2	CRC	Flow3	Flow4	CRC

6.4.1 Procedure

This command is a “pure” command (which has no command values). The procedure is as follows:

1. The master sends a start bit;
2. The master sends “slave address + write bit”, and the slave acknowledges it;
3. The master sends the command code “0x003A” (2 Bytes), and the slave acknowledges it;
4. The master sends a re-start bit (master keeps bus active). There is no stop-bit!
5. The master sends “slave address + read bit”, and the slave acknowledges it;
6. The slave returns the most significant Byte of the flow rate index (Byte1), and the master acknowledges it;
7. The slave returns the 2nd Byte of the flow rate index, and the master acknowledges it;
8. The slave returns the 3rd Byte of data, which is CRC8 of Byte1 and Byte2, and the master acknowledges it;
9. The slave returns the 4th Byte of the flow rate index (Byte4), and the master acknowledges it;
10. The slave returns the 5th Byte of the data (flow rate index), and the master acknowledges it;
11. The slave returns the 6th Byte of data, which is CRC8 of Byte4 and Byte5, and the master acknowledges it;
12. The slave returns the next least significant 8 bits of the flow rate index (Byte3), and the master acknowledges it;
13. The slave returns the least significant 8 bits of the flow rate index (Byte4), and the master acknowledge it;
14. The master sends a stop bit.

6.4.2 Note

The actual flow rate is then calculated by:
Measured flow rate [sccm] = Data (4-Byte, DEC) / 1000

E.g.:

Received flow data from the sensor: 0x00 12 D6 87
The flow rate: 1234567 (DEC) / 1000 = 1234.567 sccm

6.4.3 Sample code

This is the code example for Wire library for Arduino Atmega microcontrollers.
The flow rate is requested every 1sec from the sensor. Received data is processed to flow rate in sccm and displayed in serial monitor.

```
#include <Wire.h>
uint8_t address_PFLOW = 0x50;
uint8_t Readflow [6];

/// setting up the interface:
void setup()
{
Wire.begin();
Wire.setClock(100000);      // i2c clock set to 100kHz
Serial.begin(9600);   // initialize Serial Protocol. Change baudrate if
necessary
while (!Serial);
}

/// reading out the flow rate from the sensor
Void loop()
{
Wire.beginTransmission(address_PFLOW); // initialization of the transmition
Wire.write(0x00);                      //sending commando to the sensor
Wire.write(0x3A);                          // to read flow rate
Wire.endTransmission(false);             // communication finalizing
without stop bit
delay(2);                            // 2 msec delay; sensor response time

Wire.requestFrom(address_PFLOW, 6);       // receiving 6 bytes from the
sensor
    for (int i = 0; i < 6; i++) {
      Readflow[i] = Wire.read();
    }
/// processing of received data (packing flow data in one float variable)

long Flow = (long)Readflow[0] << 24;
Flow += (long)Readflow[1] << 16;
Flow += (long)Readflow[3] << 8;
Flow += (long)Readflow[4]; // bytes [2] and [3] are neglected as they
consist CRC

float FlowDec = (float)Flow/1000;       // calculation to sccm-value

Serial.print(FlowDec, DEC);       // displaying flow rate in Arduino serial
monitor
Serial.print("; ");

Delay (1000);
}
```

7 References:

[1] I²C-Bus Specification and User Manual, Philips Semiconductors (now NXP Semiconductors).

Rev. 03, 19 June 2

Angst+Pfister
Sensors and Power

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