131 E. Brokaw Road | San Jose CA 95112 | +1 (408) 392-0989 | info@posifatech.com | www.posifatech.com

Posifa Pressure Sensor I²C Specification

The PVC4001-C, PVC40101-C, and PVC5816 all have two different sensors, each covering a specific measurement range:

- Pressure sensor: 900 Torr to 10 Torr
- Pirani sensor: 10 Torr to 1 mTorr

Both sensors are accessed via the same I²C interface. The pressure sensor has an I²C address of 0x6D, while the Pirani sensor has an address of 0x50.

This application note outlines the I²C communication protocol for the Posifa pressure sensor.

1 I²C COMMANDS

TYPE	DESCRIPTION	SUPPORT
Measurement Request (MR)	Wakes up the sensor, performs a sensor measurement, stores the sensor measurement data in internal registers, and returns to sleep	I²C
Get Data (GD)	Retrieves the sensor measurement data from the internal pressure sensor registers*	I²C

*Note: GD does not initiate a new measurement. Repeated GD commands will return the same (or stale) sensor measurement data. An MR is required to perform a full sensor measurement cycle to refresh the sensor register data.

The GD command is used to read out data from the pressure sensor. With the start of communication, the entire sensor measurement output packet will be loaded in a serial output register. The register will be updated after the communication is finished. The output is always scaled to 24 bits. The ordering of the bits is “big-endian.”

131 E. Brokaw Road | San Jose CA 95112 | +1 (408) 392-0989 | info@posifatech.com | www.posifatech.com

2 I²C COMMAND FORMAT

2.1 PC GET DATA (GD)

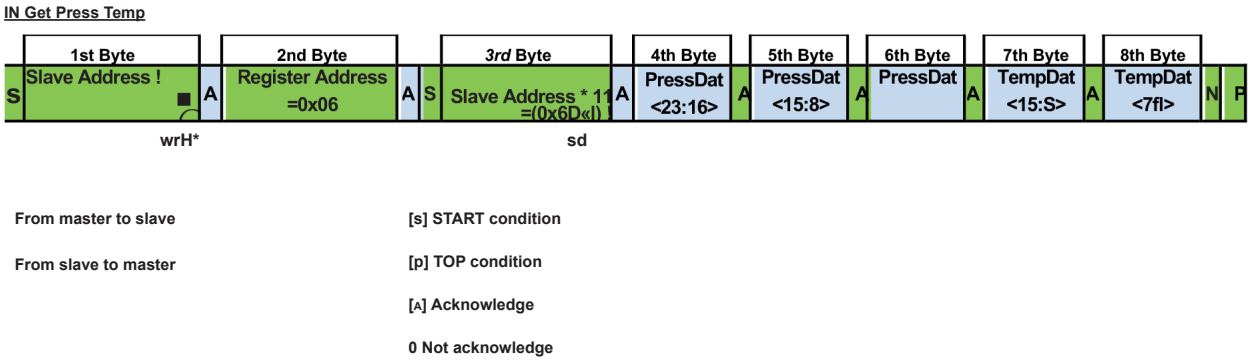
2.1.1 An I²C GD command starts with the 7-bit slave address and the 8th bit = 1 (READ). The device then sends acknowledge (ACK), indicating I²C communication success. The number of data bytes returned by the device is determined by the master, which controls NACK and stop conditions.

2.1.2 Figure 1 displays an example for sending three bytes followed by reading five bytes. The first byte contains the I²C address, followed by the internal register address (〇x〇6). Then the I²C address is repeated, followed by the slave sending out three pressure bytes and two temperature bytes.

2.1.3 The GD command is used to retrieve the pressure and temperature sensor data after an MR command has been executed.

2.1.4 N ote that the two temperature byte codes are formatted in 2's complement.

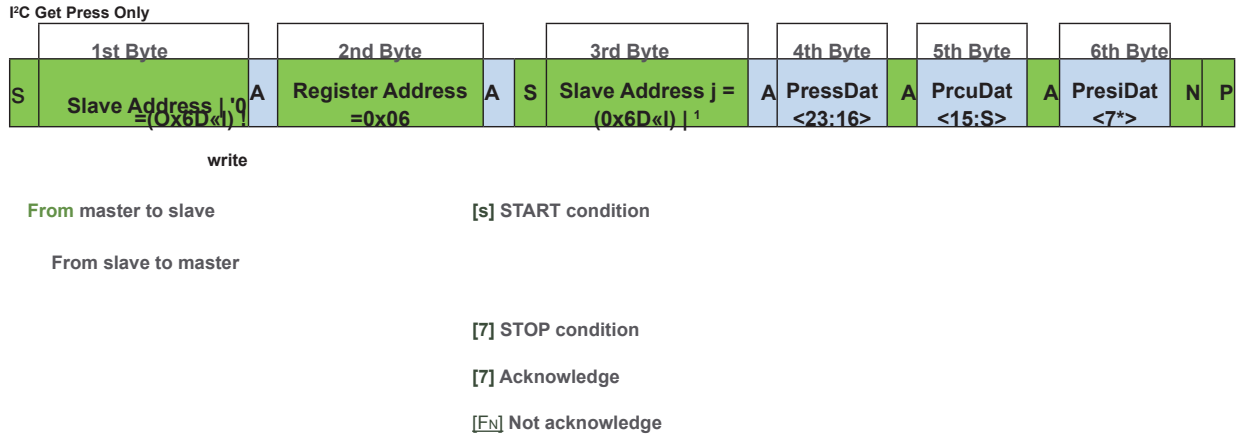
FIGURE 1: SLAVE ADDRESS FOLLOWED BY THREE PRESSURE AND TWO TEMPERATURE BYTES



For pressure data only, the data stream can be terminated after the sixth pressure byte. See Figure 2 below.

131 E. Brokaw Road | San Jose CA 95112 | +1 (408) 392-0989 | info@posifatech.com | www.posifatech.com

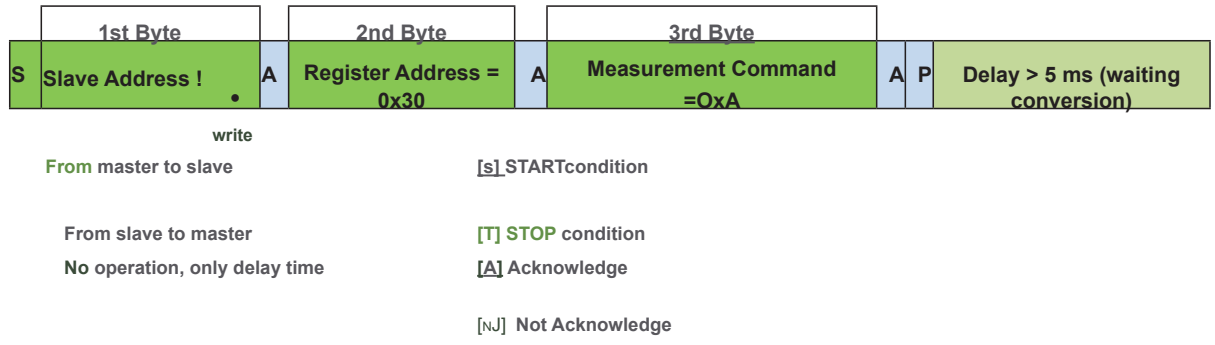
FIGURE 2: 7-BIT SLAVE ADDRESS FOLLOWED BY THREE PRESSURE BYTES



2.2 I²C MEASUREMENT REQUEST (MR)

2.2.1 The I²C MR is used to wake up the device from Sleep Mode and start a complete sensor measurement cycle, before the device returns to Sleep Mode again. The measurement cycles start with a temperature measurement, followed by a pressure measurement. The sensor measurements are digitized and run through an on-board compensation algorithm before the final measurement values are written to the digital output register. As shown in Figure 3, the communication requires the slave address (0x6D) and a WRITE bit (0) to initiate the MR. This is followed by two bytes: register address (0x30) and measurement (0xA). After the pressure responds with the slave ACK, the master terminates the communication with a stop condition.
Sensor measurement conversion time takes approximately 5 ms, so MRs should not be sent faster than every 5 ms.

FIGURE 3: I²C MEASUREMENT REQUEST COMMAND



131 E. Brokaw Road | San Jose CA 95112 | +1 (408) 392-0989 | info@posifatech.com | www.posifatech.com

3 CALCULATING OUTPUT

After retrieving the data, the compensated output can be scaled to real-world values by following the equations below.

3.1 PRESSURE OUTPUT

An example of the 24-bit compensated pressure with a full-scale range of 30 kPa to 120 kPa can be calculated as follows:

Pressure [kPa] = (Pressure 3rd Byte [23:16] x 65536 + Pressure 2nd Byte [15:8] x 256 + Pressure 1st Byte [7:0]) / 2^6/1000

3.2 TEMPERATURE OUTPUT

The 16-bit compensated temperature can be calculated as follows:

Positive Temperature [°C] = (Temperature High Byte [15:8] x 256 + Temperature Low Byte [7:0]) / 2A8

Negative Temperature [°C] = (Temperature High Byte [15:8] x 256 + Temperature Low Byte [7:0]-65536) / 2A8

131 E. Brokaw Road | San Jose CA 95112 | +1 (408) 392-0989 | info@posifatech.com | www.posifatech.com

4 SAMPLE CODE FOR PRESSURE SENSOR:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows;
using Dln;

static void Main()
{
        byte[] sendBytes = new byte[2];
        sendBytes[0] = 0x0a;
        maf.i2c_Write((byte)maf.i2c_ch_1, 0x6d, 0x30, sendBytes, 1);
        Thread.Sleep(100);

        byte[] recdata2 = maf.i2c_Read((byte)maf.i2c_ch_1, 0x6d, 0x06, 5);

        if (recdata2[0] != 0 || recdata2[1] != 0 || recdata2[2] != 0)
        {

                int Posfia_Pressure = (int)((recdata2[0] * 65536 + recdata2[1] * 256 + recdata2[2]) / 64);

        }
}


public void i2c_Write(ushort i2c_no, ushort address, byte memAddr, byte[] buffer, int numBytes)
{
   /    / int memAddr = 0;
        int memLen = 1;
        byte[] buf = new byte[numBytes];
         if (i2c == null)
         {
                return;
        }
        for (int i = 0; i < numBytes; i++)
         {
        buf[i] = Convert.ToByte(buffer[i]);
        }
         try
        {
                i2c.Write(address, memLen, memAddr, buf, numBytes);
        }
        catch
         {
                //i2c.Write(address, memLen, memAddr, buf, numBytes);
        }
}
```

131 E. Brokaw Road | San Jose CA 95112 | +1 (408) 392-0989 | info@posifatech.com | www.posifatech.com

```
public byte[] i2c_Read(ushort i2c_no, byte address, byte memAddr, int numBytes)
{
        // int  = 0;
        int memLen = 1;

        byte[] buffer = new byte[Rx_bytes];

        if (i2c == null)
        {
                return buffer;
        }
        try
        {
                i2c.Read(address, memLen, memAddr, buffer);
        }
        catch
        {
                Console.WriteLine("I2C Read Error.");
                buffer[0] = 255;
        }

        return buffer;
}
```