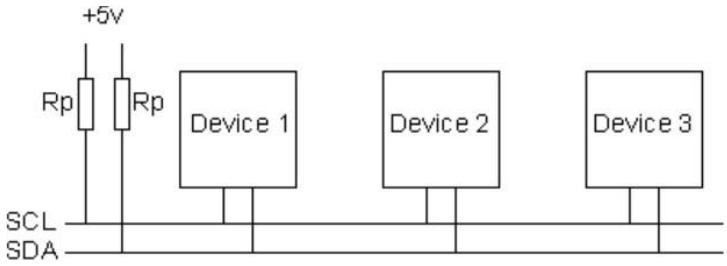
Manual for I2C communication with a KKD18/34

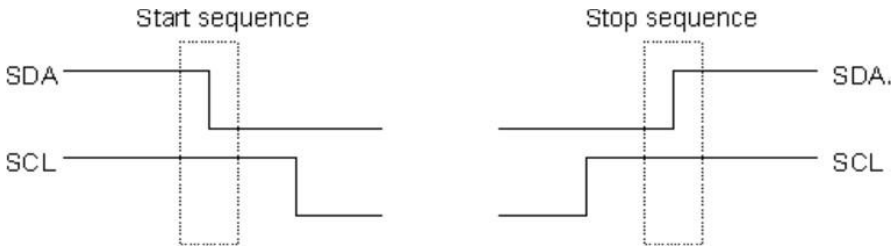
The I2C-Bus (general information):

An I2C-Bus needs mainly two wires, called SCL and SDA. SCL is the clock line. SDA is the data line. SCL is used to synchronize the data transfers, while SDA is used to transfer the data. The I2C-Bus has to be supplied with 5V. For this a +Supply and a GND wire are necessary. Both SCL and SDA lines are "open drain" drivers. What this means is that the chip can drive its output low, but it cannot drive it high. For the line to be able to go high it is necessary to provide pull-up resistors to the supply. There should be a resistor from the SCL line to the Supply line and another from the SDA line to the Supply line. Only one set of pull-up resistors, is needed for the whole I2C bus, not for each device, as illustrated below:



The devices on the I2C bus are either masters or slaves. The master is always the device that drives the SCL clock line. The slaves are the devices that respond to the master. A slave cannot initiate a transfer over the I2C bus, only a master can do that. There can be, and usually are, multiple slaves on the I2C bus, however there is normally only one master. Slaves will never initiate a transfer. Both master and slave can transfer data over the I2C bus, but that transfer is always controlled by the master.

When the master (the controller) wishes to talk to a slave (a KKD18 for example) it begins by issuing a start sequence on the I2C bus. A start sequence is one of two special sequences defined for the I2C bus, the other being the stop sequence. The start sequence and stop sequence are special in that these are the only places where the SDA (data line) is allowed to change while the SCL (clock line) is high. When data is being transferred, SDA must remain stable and not change while SCL is high. The start and stop sequences mark the beginning and end of a transaction with the slave device.



Data is transferred in sequences of 8 bits. The bits are placed on the SDA line starting with the MSB (Most Significant Bit). The SCL line is pulsed high, then low. Remember that the chip cannot really drive the line high, it simply "let´s go" of it and the resistor actually pulls it high. For every 8 bits transferred, the device receiving the

data sends back an acknowledge bit, so there are actually 9 SCL clock pulses to transfer each 8 bit byte of data. If the receiving device sends back a low ACK bit, then it has received the data and is ready to accept another byte. If it sends back a high then it is indicating it cannot accept any further data and the master should terminate the transfer by sending a stop sequence. The maximal clock (SCL) speed for I2C of KKD Sensors is 100 kHz.

I

I2C Device Addressing (general function):

All I2C addresses are either 7 bits or 10 bits. The use of 10 bit addresses is rare and is not covered here. The KKD-modules and the common chips that are used will have 7 bit addresses. This means up to 128 devices on the I2C bus are possible, since a 7bit number can be from 0 to 127. When sending out the 7 bit address, the Master still always send 8 bits. The extra bit is used to inform the slave if the master is writing to it or reading from it. The 7 bit address is placed in the upper 7 bits of the byte and the Read/Write (R/W) bit is in the LSB (Least Significant Bit).

Finding the right address is a bit confusing, if you have not worked with I2C before. It works like this:

To write to the Device with the address 0x51 (hexadecimal), the master has to actually send out 0xA2 (hexadecimal) which is 0x51 moved over by 1 bit. It is probably easier to think of the I2C bus addresses as 8 bit addresses. The last bit of the address defines if the master want to write or read from the Device. If the bit is zero the master is writing to the slave. If the bit is 1 the master is reading from the slave.

Before reading data from the slave device, the master has to tell it which of its internal addresses it want to read. So a read of the slave actually starts off by writing to it. This is the same as when the master wants to write to it: It send the start sequence, the I2C address of the slave with the R/W bit low (even address) and the internal register number you want to write to. Now it sends another start sequence (sometimes called a restart) and the I2C address again - this time with the read bit set. Then the master can read as many data bytes as you wish and terminate the transaction with a stop sequence.

This is how it works:
The first thing that will happen is that the master will send out a start sequence. This will alert all the slave devices on the bus that a transaction is starting and they should listen incase it is for them. Next the master will send out the device address. The slave that matches this address will continue with the transaction, any others will ignore the rest of this transaction and wait for the next. Having addressed the slave device the master must now send out the internal location or register number inside the slave that it wishes to write to or read from. This number depends on what the slave actually is and how many internal registers it has. The KKD-Modules have 15 locations, numbered 0-14.

Address	Name	Content	Format	Description
0	Res0	Temperature	24 bit signed (two's complement), fixed-point, 12 bit fractional	Temperature (-40 … 125 °C)
1	Res1	Pressure	24 bit signed (two's complement), fixed-point, 12 bit fractional	Pressure (0,1 … 0,99)
2	Res2	not used		
3	Res3	not used		
4	Res4	not used		
5	DAC	not used		
6	Res6	not used		
7	Res7	not used		
8	Res8	not used		
11	Res8	not used		
12	Res9	not used		
13	Res10	ID	24 bit integer	ID Number
14	Res11	Version	24 bit integer	Ver. Number

Operating Conditions and clock stretching:

That's almost it for simple I2C communications, but there is one more complication. When the master is reading from the slave, it's the slave that places the data on the SDA line, but it's the master that controls the clock. What if the slave is not ready to send the data? With devices such as EEPROMs this is not a problem, but when the slave device is actually a microprocessor with other things to do, it can be a problem. The microprocessor on the slave device will need to go to an interrupt routine, save its working registers, find out what address the master wants to read from, get the data and place it in its transmission register. This can take many µS to happen, meanwhile the master is blissfully sending out clock pulses on the SCL line that the slave cannot respond to. The I2C protocol provides a solution to this: the slave is allowed to hold the SCL line low! This is called **clock stretching**. When the slave gets the read command from the master it holds the clock line low. The microprocessor then gets the requested data, places it in the transmission register and releases the clock line allowing the pull-up resistor to finally pull it high. From the master's point of view, it will issue the first clock pulse of the read by making SCL high and then check to see if it really has gone high. If it's still low then it's the slave that holding it low and the master should wait until it goes high before continuing. Luckily the hardware I2C ports on most microprocessors will handle this automatically.
Sometimes however, the master I2C is just a collection of subroutines and there are a few implementations out there that completely ignore clock stretching. They work

with things like EEPROM's but not with microprocessor slaves that use clock stretching. The result is that erroneous data is read from the slave**.**
For Communication with a KKD-module, the master has to use clock stretching!
Operating Conditions:

PARAMETER	Symbol	MIN	TYP	MAX	UNITS
Supply Voltage	Vdd	2,6		5,5	V
Digital port voltage	Vio	-0,5		5,5	V
I2C bus frequency		0		100	kHz
Internal pull-up resistor			10		kΩ
Default address			0x51		hexadecimal

Setup of a I2C-Adapter for communicating with a KKD18/34

This Chapter will go in depth with building up a system, which is able to communicate with a KKD-Module from Pewatron. First point is the Hardware. You need an Adapter, which is able to send out and receive I2C commands and answers. Second point is the software. The Software has to be able to send out I2C commands in the hexadecimal system. In this specific example an USB-I2C Adapter from the company ELV (available here: www.elv.de ; item number: 68-08 41 23) and the free Software “HTerm” (available here: http://www.der-hammer.info/terminal/) were used. It is not needed to use exactly this Hardware and Software combination. Most Hardware and Software, which is able to work with the I2C-Bus, should be working with the KKDmodules.

Note: All commands in this Chapter are written in the hexadecimal number system.

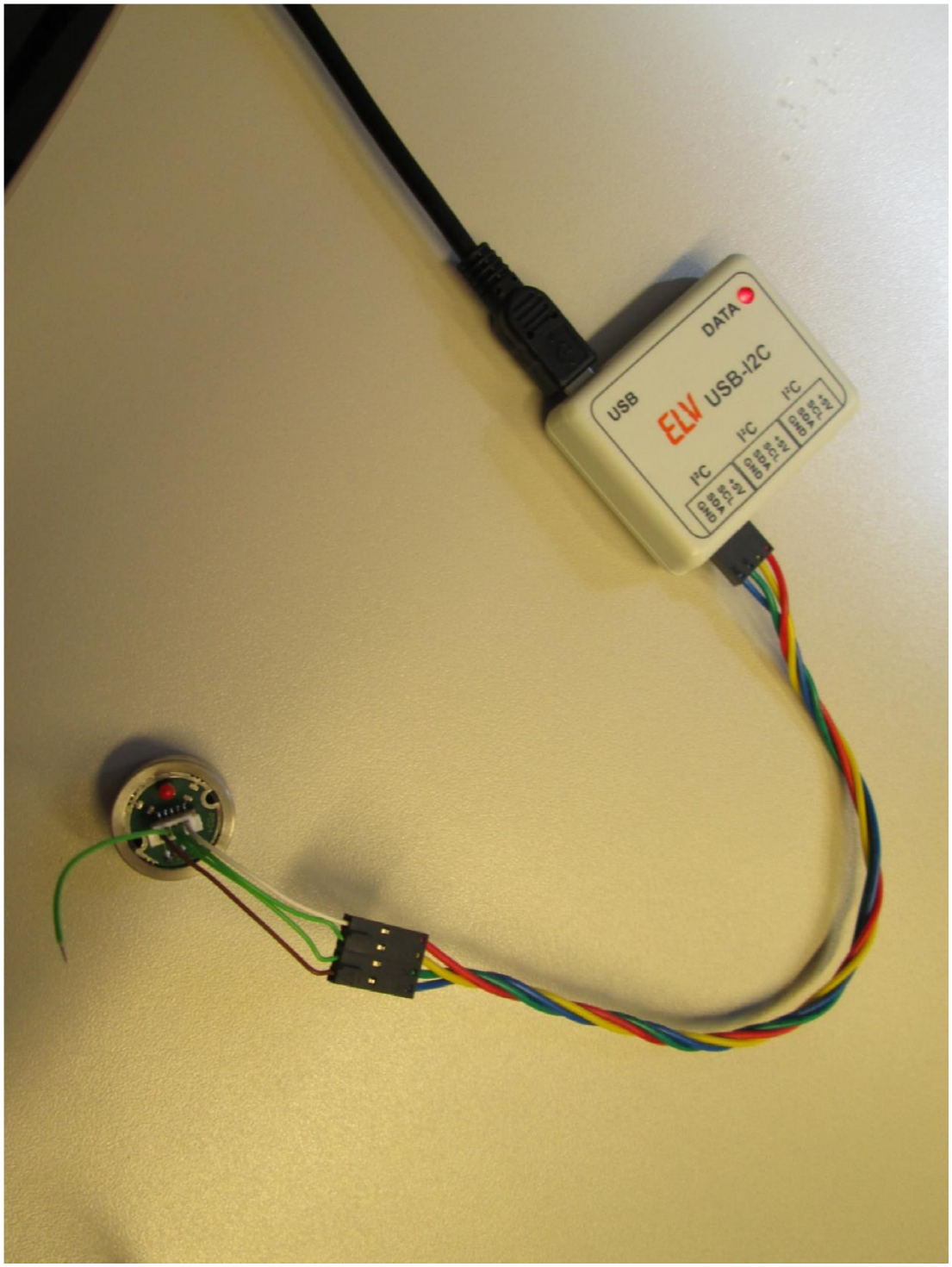
At first the adapter has to be connected to the KKD-module. Like mentioned in the previous Chapter, I2C needs 4 wires. SCL and SDA which are the data lines and a +Supply wire and a ground wire for the Supply. The pin assignment for the KKDModules are shown in their specific datasheets.

Note: Don´t use a supply Voltage higher than 5V! Devices could be damaged!

After the connection of the I2C adapter and the KKD-module, the software needs to send commands and receive the right Information. Every communication starts with a start sequence and ends with a stop sequence. These sequences are predefined in the hardware you use. In the USB-I2C adapter, which is used for this example, the command for the start sequence is “S” and for the stop sequence it is “P”. If you use a different hardware and you don’t know this commands look for them in the manual of your hardware.

Start your communication with a start sequence on the I2C-Bus. After that you have to address the KKD-module. Standard address of the KKD-module is 0x51. It is not

possible to instantly address the module with this, a read or write Bit at the eight bit for your address has to be set. The correct addresses for the KKD-module are 0x51 moved over by 1 bit. So 0xA2 is the address for writing and 0xA3 for reading. After the address you need to define which internal register of the KKD-module you want to read/write to. In the KKD-modules there are two which are important: 0x40 is the internal register for the Temperature, 0x41 is for the pressure.



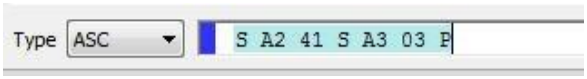
Picture 1 - Connection of KKD Module with I2C Adapter

Specific examples for pressure:

Reading pressure form a KKD-Module (written in the hexadecimal system):

1. Send a start sequence
2. Send 0xA2 (I2C address of the KKD-Module with write bit)
3. Send 0X41 (Address of the internal register for pressure)
4. Send a start Sequence again
5. Send 0xA3 (I2C address of the KKD-Module with read bit)
6. Send 0x03 (Master need to read 3 bytes from the KKD-Module)
7. Send a stop Sequence

Command should look like this:



Note: Command for the Start/Stop sequence depends on your I2C Adapter. In this case it is S/P but different Adapters may use other commands.

You will get an answer like this:

```
01 A4 2C
```

To convert the answer to an usable value you need to convert it into the decimal system first. It works like shown here:

Every number/letter stands for a multiple of a 16er power, starting with 16^0 on the right side. The numbers over 9 are represented as letters from A to F. In this example calculation would look like this:

C: $12 \times 16^0 = 12$
2: $2 \times 16^1 = 32$

4: $4 \times 16^2 = 1024$

A: $10 \times 16^3 = 40960$

1: $1 \times 16^4 = 65536$
Sum: $12 + 32 + 1024 + 40960 + 65536 = 107564$

Now, you have to convert this decimal value to a pressure value.

NOTE: The max. conversion range is 2^{17} digits (=131072 digits). Zero point is defined for 10% and endpoint is defined for 90% of max. conversion range (2^{17}).

Formula looks like this:

$p = (x – ZP) / (EP-ZP) \times (p_{EP} – p_{ZP})$ Definition:

p = pressure x = actual value from i2C
ZP = Zeropoint of conversion (=13107 d)
EP = Endpoint of conversion (=117965 d)
p_{EP} = Endpoint pressure p_{ZP} = Zeropoint pressure

Example:

p = ? x =
107564 ZP =
13107 EP =
117965 p_{EP} =
10 bar p_{ZP} =
0 bar

Result: p = 9,008 bar

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