By D. Hofer, Keller AG für Druckmesstechnik

Communication Protocol 4 LD ... 9 LD

Date of Release 11. Feb. 2012

Filename Communication Protocol 4LD..9LD Feb12.doc

Abstract

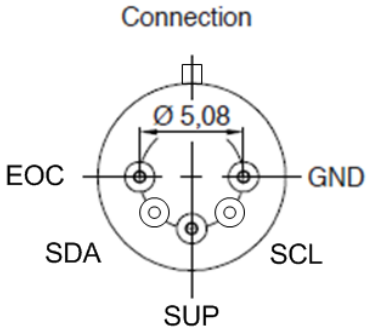
Visually the Series 4 LD … 9 LD are like standard KELLER pressure transdurcers with a 5 pinout to connect the half-open Wheatstone Bridge. But these I²C versions contain beside the pressure sensor a very tiny signal conditioner. This results in an OEM pressure transmitter with a digital interface. The “D” stands for “digital” and for “dual”; the LD-Line provides pressure and temperature information.

The most important topics regarding the communication with the Series 4 LD … 9 LD and KELLER’s unique embedded DSP core, are listed in this protocol description - especially the interpretation of the readout values.

This file is still under construction.

1 Electrical interface and Pinout

Lable	Description	Wire
SUP	1,8...3,6 V	BK
GND	GND	WH
SCL	I²C Clock	YE
SDA	I²C Data	BU
EOC	End of Conv.	RD



Pull-up resistors are needed at SDA and SCL. 1..10kOhm are recommended. In order to optimise the data rate or low power consumption, other resistance values are possible.

The EOC-Pin supplies an active high level in idle state and an active low level during conversion.

Notes

Be careful with the glazed pins, cracks in the glass pills causes leakage => damage

Do not touch the steal diaphragm!

2 Get Measurement Data

2.1 Get the digital Values

ADDR default = 0x00

First byte is: (ADDR << 1) + 1 for Read
(ADDR << 1) + 0 for Write

1. Request Measurement (2 bytes from Master)

ADDR	| W	0xAC

2. Wait 10ms or wait for EOC=1 (goes up to VDD) or check the “Busy?” flag [5] in the status byte (only one byte reading needed).

3. Read Measurement (1 byte from Master, 5 bytes from Slave)

ADDR	| R	STATUS	P MSB	P LSB	T MSB	T LSB

Getting only the pressure information; it is possible to read out only 3 bytes from the Slave.

2.2 Interpretation of the digital Values

P [u16]

16384	P_{16384} resp. P_min, e.g. –1 bar PR
49152	P_{49152} resp. P_max, e.g. 30 bar PR

The pressure range of the transmitter is stored in its memory and/or written on the associating papers.

$P\,[bar] = (P\,[u16] - 16384) \times (P_{49152} - P_{16384}) / 32768 + P_{16384}$

The output range is ¼ to ¾ of the 16 bit output word. This way a little over- and under-pressure is measurable and the exceeding resolution of more then 30'000 point guarantee a very high resolution of 10'000 points even for the next lower standard pressure range.

T [u16]

384	-50°C
64384	150°C

T[°C] = (floor(T[u16] / 16) – 24) x 0.05°C – 50°C

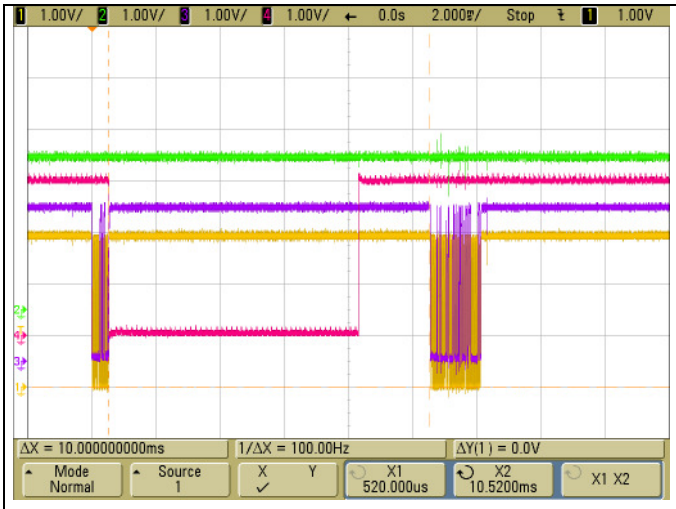
= (T[u16] >> 4) – 24) x 0.05°C – 50°C

Reduce the 16 bits of the temperature information first to 12bit; the last 4 bits are anyway noise floor. This way a resolution of 1/20°C is still given.

The scaling goes from –50 to 150°C but the working temperature range of the transmitter is at maximum -40..85°C (depending on the order; 0..50°C and –10..80°C are the standard temperature ranges).

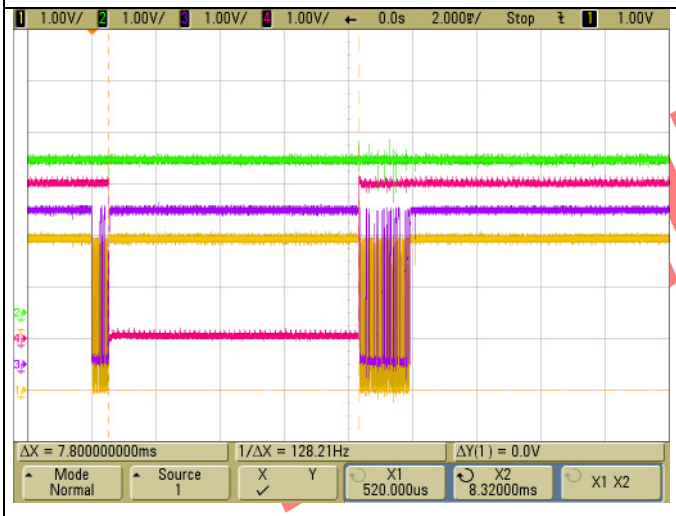
2.3 Variants to detect the end of conversion

Yellow: SCL, Blue: SDA, Red: EOC, Green: SUP



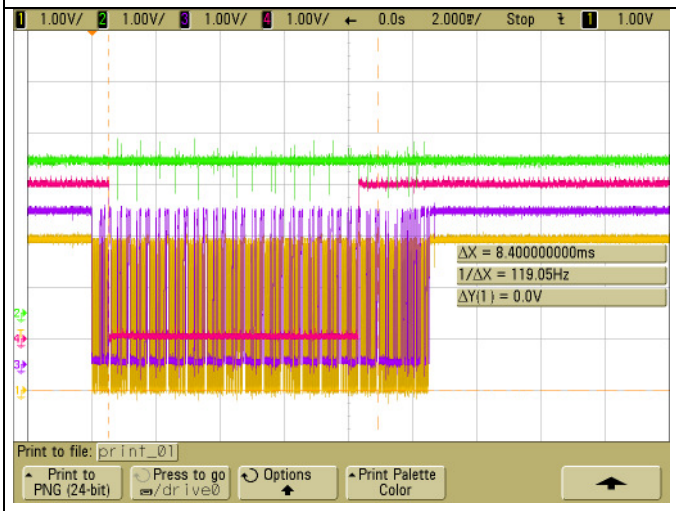
The simplest way to detect the end of a conversion (EOC) is to wait until the new data is definitely ready to read out. Being on the safe side; the conversion and the conditioning of the pressure and temperature value is completed after ~10ms.

While the 10ms of waiting, the Master controller can be in sleep mode or doing some other tasks like requesting other pressure transmitters on the bus to make a new conversion.



The handshake-solution, done by the additional EOC wire, is very elegant and is suitable to save time and power. The Master controller can be in sleep mode and will be awoken by an external interrupt on the positive slope of the EOC pin. Polling the level of the EOC wire is also possible.

For this solution an additional wire per transmitter is needed. It is not possible to connect all the EOC wires commonly like SCL and SDA of the bus system.



To save time without the additional EOC wire is possible by reading out the status of the pressure transmitter. There is no request needed, just a simple readout command for the first byte that contains the “Busy?” flag.

Table 3.3 General Status Byte.

Bit	7	6	5	4	3	2	1	0
Meaning	0	Powered?	Busy?	Mode		Memory error?	Data transfer	Special

Bit 6 (“Powered?”) and bit 5 (“Busy?”) will be „1“ during the conversion. At the end of the conversion bit 5 changes to “0”. Then the new data is ready to read out by additional Clocks for the pressure and temperature bytes or a new readout command to shift out the whole 5 byte data frame.

This variant effects the highest power consumption because the Master controller is nonstop busy and also the pull-up resistors are energized more often.

3 Optional further Commands

3.1 User-Words

16bit memory cells

MTP Address	Description	Customer-Definition	Remarks
0x00	Cust_ID0	Equipment# [0..63] MSB | Place# [0…1023] LSB	for DB access
0x01	Cust_ID1	File# LSB	for DB access
0x11	Not assigned	File# MSB	presently AUX
0x12	Not assigned	Year [0..31]+2012 | Month [0..15] | Day [0..31] | P-Mode[0..3] LSB	Y:5bit|M:4bit|D:5bit|P:2bit
0x13	Not assigned	P_16384 [f32 (IEEE 754, single) MSB]	Pmin [bar] als 32bit float
0x14	Not assigned	P_16384 [f32 (IEEE 754, single) LSB]	
0x15	Not assigned	P_49152 [f32 (IEEE 754, single) MSB]	Pmax [bar] als 32bit float
0x16	Not assigned	P_49152 [f32 (IEEE 754, single) LSB]	

IEEE 754: single bzw. float von “single-precision binary floaing-piont format”

P-Mode[0..3]: 0=PR, 1=PA, 2=PAA, 3=PR

Read Memory Content:

ADDR default = 0x00

First byte is: (ADDR << 1) + 1 for Read
(ADDR << 1) + 0 for Write

1. Request Measurement (2 bytes from Master)

ADDR | W	MTP Address (0x13..0x16)

2. Wait for 0.5ms or check the “Busy?” flag

3. Read Measurement (1 byte from Master, 3 bytes from Slave)

ADDR | R	STATUS	Mem MSB	Mem LSB

4. Interpretation

In the two LSBs of cell 0x12 is the pressure mode (sealed or vented gauge and zero definition) stored.

The content of cell 0x13 and 0x14 is a floating-point value that indicates the pressure in [bar] for the lower output value, 16384.

The content of cell 0x15 and 0x16 is a floating-point value that indicates the pressure in [bar] for the lower output value, 49152.

3.2 Recommended Slave Addresses

If you want to combine more than one pressure transmitter on the same I2C bus, the slave addresses have to be unique. For this purpose the memory content of -for example- a second transmitter have to be overwritten. It is not possible to erase the content to make any possible change because the memory bases "on a one time programmable technology", so it is only possible to add some "1"s by burning additional bit-cells. After adding 6 (or 7) "1"s to the 7 bit slave address register, there is a further possibility to make changes: cleaning the whole memory content by incrementation of the page counter. That gives you in minimum a second chance to choose a slave address absolutely independent from the tries before.

The conclusion is that it is not possible to change the slave address uncountable times. So it is recommended to plan the whole bus system and program the bus addresses once or in case of something unpredictable a second time.

To have more than one possibility per memory page to change the slave address, we recommend the following set off addresses.

Shot	Description	Slave-ADDR
0	1st Transmitter, default	0x00
1	2nd Transmitter	0x20
2	3rd Transmitter	0x21
3	4th Transmitter	0x23
4	5th Transmitter	0x27
5	6th Transmitter	0x2F
6	7th Transmitter	0x3F
(7)	(8th Transmitter)	(0x7F)

With the mentioned addresses it is possible to make for example a 3rd transmitter on the bus to a 4th.

The „I²C committee" does not recommend to use addresses between 0x78 and 0x07F, so the 7th try is possible but not favored.

The addresses 0x00 to 0x07 are also reserved but 0x00 (our default) is the „General call address".

If you change the slave address and don't use a new memory page, the checksum can not be updated. The STATUS byte is then no longer 0x40 (only "Powered?" bit is set), it becomes 0x44 ("Memory error?" appears) but that does not matter.

Bit	7	6	5	4	3	2	1	0
Meaning	0	Powered?	Busy?	Mode		Memory error?	Data transfer	Special

4 Source Code

Still under construction

4.1 Header-File

```
////////////////////////////////////////////////////////////////////////////
// constants
#define SDA_OUT      TRISDbits.TRISD5     // RD5 is SDA (without MSSP)
#define SDA_OD       LATDbits.LATD5
#define SDA_IN       PORTDbits.RD5
#define SCL_OUT      TRISDbits.TRISD6     // RD6 is SCL (without MSSP)
#define SCL_OD       LATDbits.LATD6


#define cZI_Pmin     0                    // fix coded or read out from the userMEM
#define cZI_Pmax     30                   // fix coded or read out from the userMEM

////////////////////////////////////////////////////////////////////////////
// global variables
#ifndef __C_ZI_ZSSC_I2C__
extern
#endif
_F32 ZI_pressure;    // Variable for pressure value in [bar] as single (IEEE 754)

#ifndef __C_ZI_ZSSC_I2C__
extern
#endif
_F32 ZI_temperature;// Variable for temperature value in [°C] as single (IEEE 754)

#ifndef __C_ZI_ZSSC_I2C__
extern
#endif
_U8 ZI_status;       // Variable for 8 bit status

#ifndef __C_ZI_ZSSC_I2C__
extern
#endif
_U8 ZSSCget[5];      // Array to receive data frame


////////////////////////////////////////////////////////////////////////////
// prototypes global functions
_U8 get_PnT_GPIO(_U8);
_U8 get_PnT_MSSP(_U8);
```

4.2 C-File

```
////////////////////////////////////////////////////////////////////////////
// global functions

_U8 get_PnT_GPIO(_U8 ADDR){
    _U8 ZSSCerror=0;
    _F32 Pmin, Pmax;
    union {
        _F32 floatingpoint;
        _U32 twotimesU16;
    } cast;

    ////// read the scasling //////
    I2C_write_1Byte(ADDR,0x13);
    Delay10TCYx(125);                   // 0.4us x 10 x 125 = 0.5ms
    I2C_read_xByte(ADDR,3);
    ZMDget[1]=0x80;
    ZMDget[2]=0;
    cast.twotimesU16 = (((_U32)(ZMDget[1]))<<24) + (((_U32)(ZMDget[2]))<<16);
    I2C_write_1Byte(ADDR,0x14);
    Delay10TCYx(125);                   // 0.4us x 10 x 125 = 0.5ms
    I2C_read_xByte(ADDR,3);
    cast.twotimesU16 += (((_U32)(ZMDget[1]))<<8) +  ((_U32)(ZMDget[2]));
    Pmin= cast.floatingpoint;

    I2C_write_1Byte(ADDR,0x15);
    Delay10TCYx(125);                   // 0.4us x 10 x 125 = 0.5ms
    I2C_read_xByte(ADDR,3);
    cast.twotimesU16 = (((_U32)(ZSSCget[1]))<<24) + (((_U32)(ZSSCget[2]))<<16);
    I2C_write_1Byte(ADDR,0x16);
    Delay10TCYx(125);                   // 0.4us x 10 x 125 = 0.5ms
    I2C_read_xByte(ADDR,3);
    cast.twotimesU16 += (((_U32)(ZSSCget[1]))<<8) + ((_U32)(ZSSCget[2]));
    Pmax= cast.floatingpoint;

    ////// request new conversion //////
    if(I2C_write_1Byte(ADDR,0xAC)){return 0x91;}

    ////// wait for new conversion result //////
    Delay100TCYx(250);                  // 0.4us x 100 x 250 = 10ms

    ////// read the results out //////
    if(I2C_read_xByte(ADDR,5)){return 0x91;}

    ////// interpret integer values //////
    ZI_status       =                                       ZSSCget[0];     //  [U8]
    ZI_pressure     = (_F32)( (((_U16)(ZSSCget[1]))<<8) + (_U16)(ZSSCget[2]) ); // p[U16]
    ZI_temperature  = (_F32)( (((_U16)(ZSSCget[3]))<<8) + (_U16)(ZSSCget[4]) ); // T[U16]

    ZI_pressure     = (ZI_pressure-16384)*(Pmax-Pmin)/32768+Pmin;                // p[bar]
    ZI_temperature  = (((((_U16)ZI_temperature)>>4)-24)*0.05)-50;               // T[°C]

    return ZSSCerror;
} // end of _U8 get_PnT_GPIO(_U8 ADDR)
```