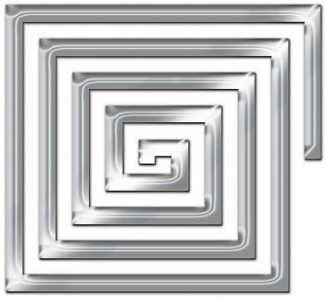


MEMSCAP

The Power of a Small World™

TP1200 User Manual

Revision History

Revision	Reference	Released Date	Approved By	Description
00		2014-12-02		First release.
01		2018-08-08		Appendix C EEPROM Layoyt Table deleted. Last part of section 7.2.3 deleted. Section 7.2.4 deleted. Min_P, Max_P, Min_T and Max_P updated, pressure and temperature limits added. Look up table examples in section 8.1 and 8.2 updated using Float-32 values. The "Float32 x 2 x Points" updated to "Float32 x 2 x Points_lin_T". Added "CRC" to section 4. Changed from "ADC1" to "ADC" in figure section 6.
02		2018-11-014		Added a comment in section 8.2 regarding observed charging effect
03	2093-19	2019-04-04		Added “pressure” to text in section 6.2.1 Charging effect comment moved to section 7.2 Appendix B communication example updated
04	3023-19	2019-10-14	RG	Section 2 Documents updated for TP1200

Table of contents

1 Introduction

This document contains information on how to use the MEMSCAP TP1200 pressure transducer.

2 Documents

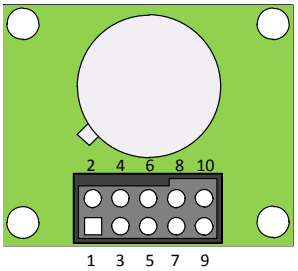
25LC256. *Datasheet for EEPROM 25LC256, downloadable from www.microchip.com.*
AD7793. *Datasheet for 24-bit ADC AD7793, downloadable from www.analog.com.*
SWD11587. *TP1200 EEPROM Layout.*
TS1115. *TP1200 Technical Specification.*

3 Acronyms and Abbreviations

ADC	Analogue to Digital Converter
EEPROM	Electrically Erasable Programmable Read Only Memory
FS	Full Scale
FSO	Full Scale Output
LUT	Look-Up Table.
MRO	Maintenance, Repair and Overhaul
SPI	Serial Peripheral Interface
CRC	Cyclic Redundancy Check

4 Electrical Interface

The pinout of the TP1200 is shown below.



Pinout
1. Power in
2. CS_EE (chip select EEPROM)
3. WP_EE (write protect EEROM)
4. SCLK (common clock for SPI interface)
5. MISO (data from the TP1200)
6. VCC_ERROR
7. CS_ADC (chip select ADC)
8. Not in use – leave open
9. GND
10. MOSI (data into the TP1200)

4.1 Pinout

4.1.1 Power

Supply voltage to the TP1200 is applied to the Power in and GND pins. See the TP1200 Data Sheet for supply voltage range. Inside the TP1200 the applied voltage is regulated to 5V using a linear regulator (LP2951 from National Semiconductor). Memscap recommends a supply voltage of 10-12 volts.

4.1.2 CS_EE

CS_EE is the chip-select for the EEPROM. The CS_EE has a weak pull-up inside the TP1200, but it is recommended that this signal is connected to the external logic-high level with a 22kΩ resistor.

4.1.3 WP_EE

WP_EE is for write-protecting the contents of the EEPROM. This signal should be connected to GND.

4.1.4 SCLK

SCLK is the clock for the SPI bus. Data will be clocked into the TP1200 on the positive flank. Data to be read from the TP1200 will be set up at the negative flank, and is stable for reading on the positive flank of the SCLK.

4.1.5 MISO

MISO is data from the TP1200 (Master In Slave Out – the TP1200 is regarded as the slave in the system). Inside the TP1200 the data outputs from the EEPROM and the ADC are wire-ORed through 470Ω resistors and connected to the MISO output.

4.1.6 VCC_ERROR

This signal indicates that internal voltage regulator in the TP1200 operates as it should. This signal should be +5V. If the signal drops below 4.5V it indicates an error. See the data sheet for the LP2951 from National Semiconductor for more details of its operation. The ERROR output from the LP2951 is connected to VCC_ERROR through a 470Ω resistor.

4.1.7 CS_ADC

CS_ADC is the chip-select for the A/D converter interfacing the pressure sensor. The CS_ADC has a weak pull-up inside the TP1200, but it is recommended that this signal is connected to the external logic-high level with a 22kΩ resistor.

4.1.8 GND

GND is the common reference for the input voltage and the logic signals. GND is not connected to the mounting holes.

4.1.9 MOSI

MOSI is data into the TP1200 (Master Out Slave In). Inside the TP1200 the ADC and the EEPROM share this line.

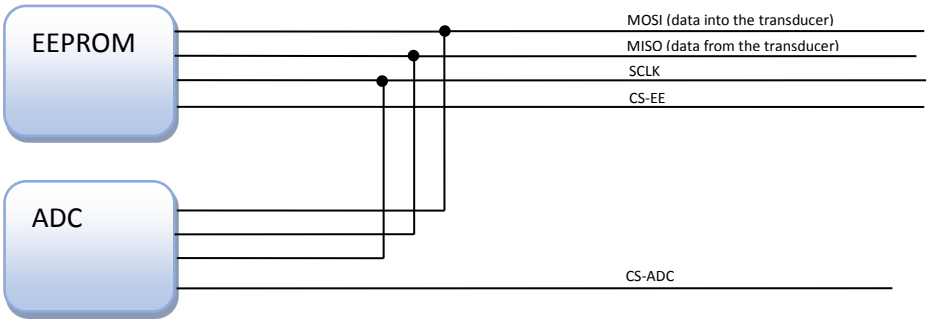
4.2 Logic Levels

All digital inputs and outputs are TTL compatible.

5 Block Diagram and SPI Bus

The TP1200 contains an EEPROM, 25LC256 from Microchip, and an analogue to digital converter, AD7793BCPZ from Analog Devices. Please download the data sheets from the manufacturers for detailed information on operation.

Two-way communication with the TP1200 is done over the SPI bus. Inside the TP1200 the EEPROM and the ADC share the signals SCLK, MOSI and MISO. Each of the two devices has its own chip-select signal (CS). SCLK is the clock, MOSI is for clocking data into the transducer and MISO is for reading data from the transducer. Both chip-selects are active-low. Make sure that only one chip-select is asserted at a time! Data is set up on negative transition of SCLK, and is read on the positive flank. See the referenced data sheets for the EEPROM and AD converters for detailed timing information.



6 Communication

Communication with the TP1200 consists of reading the EEPROM once (initialization), and then repeated measurements on the sensor by reading the appropriate A/D converter. The sensor can measure both temperature and pressure by addressing the ADC in the appropriate way.

6.1 Initialization

The TP1200 requires no initialization, but after power-up the device controlling the TP1200 will typically start by reading the contents of the EEPROM. The EEPROM contains general information about the TP1200, ADC settings for both the pressure and temperature reading as well as look-up tables for calculating temperature and pressure from the values that can be read from the A/D converter.

6.1.1 EEPROM Reading

Use the following sequence to read the EEPROM contents:

- Assert the EEPROM by pulling CS_EE low.
- Write 0x03 to SPI (read instruction).
- Write 0x00 to SPI (most significant address byte).
- Write 0x00 to SPI (least significant address byte).
- Read all bytes of data from SPI. The address counter in the EEPROM will automatically increase for every eight bits read as long as the CS_EE is kept low.
- Set CS_EE high when all bytes have been read.

6.1.2 Verify EEPROM Data Integrity

The EEPROM contains several checksums (CRCs). See the document *TP1200 EEPROM Layout* for details. These CRCs should be used to ensure that the data read from the EEPROM is correct. Checking the CRC can be done in two ways, using the methods given in the Appendix.

1. Calculate the CRC of all the contents in the EEPROM, including the stored CRC. If the contents is correct, the new CRC shall always be DEBB20E3(hex).
2. Calculate the CRC of the EEPROM contents, not including the stored CRC. Do a bit-inversion of the calculated CRC. The result shall then be the same as the CRC value stored in the EEPROM.

6.1.3 Decoding EEPROM Contents

If the content of the EEPROM is verified to be correct, continue by extracting information according to the EEPROM Layout description. Essential data to be extracted are:

- Information about calibration ranges.
- ADC settings – different settings for temperature and pressure readings.
- A look-up table for temperature linearization.
- A two-dimensional look-up table for pressure linearization.

In addition the EEPROM contains various fields of product specific information.

6.2 ADC Communication

The TP1200 contains an AD7793 from Analog Devices. Please download the data sheet from http://www.analog.com for a complete understanding of how the A/D converter work. The ADC can read both temperature and pressure from the sensor. Assert CS_ADC to communicate with the ADC.

The parameters for setting the ADC are read from the transducer EEPROM. Refer to the document *TP1200 EEPROM Layout* for information on where in the EEPROM the data for the various ADC registers can be found.

6.2.1 Initialisation

Follow this procedure to obtain an ADC value representing the pressure and temperature from the sensor:

- Set ADC_CS low.
- Write 0xFFFFFFFF to SPI.
- Wait 2 ms.
- Write 0x08 and ADC_mode (16 bits) to SPI.
- Write 0x28 and ADC_InOut (8 bits) to SPI.
- Write 0x10 and ADC_config_P (16 bits) to SPI.
- Write 0x30 and ADC_offset_P (24 bits) to SPI.
- Write 0x38 and ADC_gain_P (24 bits) to SPI.
- Write 0x10 and ADC_config_T (16 bits) to SPI.
- Write 0x30 and ADC_offset_T (24 bits) to SPI.
- Write 0x38 and ADC_gain_T (24 bits) to SPI.
- Set ADC_CS high.

6.2.2 Measure

- Set ADC_CS low.
- Write 0x10 and ADC_config_P (16 bits) to SPI.
- Write 0x08 and ADC_mode,
  with the 3 most significant bits equal to 001 in binary, (16 bits) to SPI.
- Wait for low MISO (optionally with timeout equal to ADC_timeout ms).
- Write 0x58 to and read ADC_data_P (24 bits) from SPI.
- Write 0x10 and ADC_config_T (16 bits) to SPI.
- Write 0x08 and ADC_mode,
  with the 3 most significant bits equal to 001 in binary, (16 bits) to SPI.
- Wait for low MISO (optionally with timeout equal to ADC_timeout ms).
- Write 0x58 to and read ADC_data_T (24 bits) from SPI.
- Set ADC_CS high.

6.2.3 Safety and Built In Test

- Set ADC_CS low.
- Write 0x68 and verify ADC_InOut (8 bits) from SPI.
- Write 0x10 and ADC_config_P (16 bits) to SPI.
- Write 0x70 and verify ADC_offset_P (24 bits) from SPI.
- Write 0x78 and verify ADC_gain_P (24 bits) from SPI.
- Write 0x10 and ADC_config_T (16 bits) to SPI.
- Write 0x70 and verify ADC_offset_T (24 bits) from SPI.
- Write 0x78 and verify ADC_gain_T (24 bits) from SPI.
- Set ADC_CS high.

7 Linearization and Compensation

Compensated values for temperature and pressure are calculated using the digital values read from the A/D converter and the contents of the look-up tables stored in the EEPROM. Compensated temperature is calculated using a linear approximation, and compensated pressure is calculated using a bi-linear approximation. The same methods are used for both absolute and the differential sensors.

7.1 Calculating Compensated Temperature

From the EEPROM it is possible to extract a table called Lin_data. This table contains a number of ADC values and the corresponding temperature. By doing a linear interpolation new values can be calculated.

This algorithm may be used for calculating the temperature:

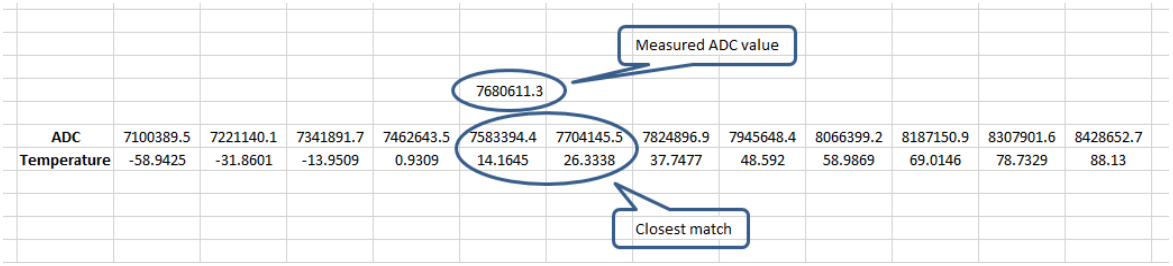
1. From Lin_data_T, let x1 be the ADC value just smaller than the measured ADC value.
2. From Lin_data_T, let x2 bet the ADC value just greater than the measured ADC value.
3. Let y1 and y2 be the temperature values that correspond to x1 and x2.
4. Calculate the temperature:

$$a = \frac{y2 - y1}{x2 - x1}$$

$$b = y1 - a * x1$$

$$temperature = a * ADCvalue + b$$

A typical look-up table (Lin_data_T) is shown below. If the measured ADC value is 7680611, it is clear that the corresponding temperature will lie somewhere between 14.1645°C and 26.3338°C. By using the algorithm described the temperature is calculated to be 23.96°C.



ADC	7100389.5	7221140.1	7341891.7	7462643.5	7583394.4	7704145.5	7824896.9	7945648.4	8066399.2	8187150.9	8307901.6	8428652.7
Temperature	-58.9425	-31.8601	-13.9509	0.9309	14.1645	26.3338	37.7477	48.592	58.9869	69.0146	78.7329	88.13

Make sure to handle values outside the calibration range separately, as this may lead to erroneous calculations!

7.2 Calculating Compensated Pressure

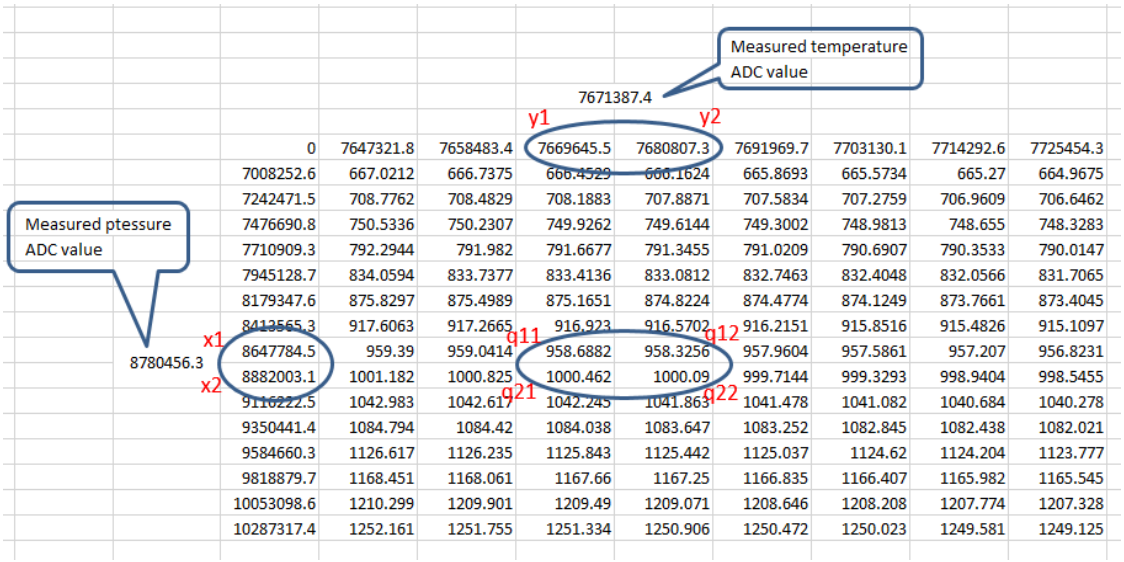
From the EEPROM it is possible to extract a two-dimensional table called LUT_P. This table contains a number of ADC values along one axis and ADC values for pressure along the other axis. For each pair of ADC-values the table contains a pressure value. By doing a bilinear interpolation new values can be calculated.

This algorithm may be used for calculating the pressure:

1. From LUT_P, let x1 be the pressure ADC value just smaller than the measured pressure ADC value.
2. From LUT_P, let x2 be the pressure ADC value just greater than the measured pressure ADC value.
3. From LUT_P, let y1 be the temperature ADC value just smaller than the measured temperature ADC value.
4. From LUT_P, let y2 be the temperature ADC value just greater than the measured temperature ADC value.
5. Let q11 be the value in LUT_P that corresponds to x1, y2. q11=LUT_P[x1, y1].
6. Let q21=LUT_P[x2, y1].
7. Let q12=LUT_P[x1, y2].
8. Let q22=LUT_P[x2, y2].
9. The compensated pressure can now be calculated using bilinear interpolation:

$$pressure \approx \frac{q11}{(x2-x1)(y2-y1)}(x2-x)(y2-y) + \frac{q21}{(x2-x1)(y2-y1)}(x-x1)(y2-y) + \frac{q12}{(x2-x1)(y2-y1)}(x2-x)(y-y1) + \frac{q22}{(x2-x1)(y2-y1)}(x-x1)(y-y1)$$

The figure below shows an extract of a LUT_P table and measured ADC values for temperature and pressure. Using the algorithm above, the calculated pressure is 982.29 (mbar).



0	7647321.8	7658483.4	7669645.5	7680807.3	7691969.7	7703130.1	7714292.6	7725454.3
7008252.6	667.0212	666.7375	666.4529	666.1624	665.8693	665.5734	665.27	664.9675
7242471.5	708.7762	708.4829	708.1883	707.8871	707.5834	707.2759	706.9609	706.6462
7476690.8	750.5336	750.2307	749.9262	749.6144	749.3002	748.9813	748.655	748.3283
7710909.3	792.2944	791.982	791.6677	791.3455	791.0209	790.6907	790.3533	790.0147
7945128.7	834.0594	833.7377	833.4136	833.0812	832.7463	832.4048	832.0566	831.7065
8179347.6	875.8297	875.4989	875.1651	874.8224	874.4774	874.1249	873.7661	873.4045
8413565.3	917.6063	917.2665	916.923	916.5702	916.2151	915.8516	915.4826	915.1097
8647784.5	959.39	959.0414	958.6882	958.3256	957.9604	957.5861	957.207	956.8231
8882003.1	1001.182	1000.825	1000.462	1000.09	999.7144	999.3293	998.9404	998.5455
9116222.5	1042.983	1042.617	1042.245	1041.863	1041.478	1041.082	1040.684	1040.278
9350441.4	1084.794	1084.42	1084.038	1083.647	1083.252	1082.845	1082.438	1082.021
9584660.3	1126.617	1126.235	1125.843	1125.442	1125.037	1124.62	1124.204	1123.777
9818879.7	1168.451	1168.061	1167.66	1167.25	1166.835	1166.407	1165.982	1165.545
10053098.6	1210.299	1209.901	1209.49	1209.071	1208.646	1208.208	1207.774	1207.328
10287317.4	1252.161	1251.755	1251.334	1250.906	1250.472	1250.023	1249.581	1249.125

Make sure to handle values outside the calibration range separately, as this may lead to erroneous calculations!

The following is valid for part no 32074, 32073, 32072, 32127 and 32147:

Due to a charging effect in the HW of the TP1200, the first pressure reading after having read the temperature will have an offset in the range of +0.01% of FS. It is therefore recommended to either make more that one reading, omit the first reading and use the average of the remainder, or alternatively, if only one reading is to be used, read twices, discard the first and use the second reading for pressure value. CRC Calculation

A. CRC Calculation

Below are four C# methods; CalcTable, CalculateCrcWithoutInversion, Calculate and IsOk.

CalcTable calculates the crcTable_32 found below. This method is included for reference only – it is not needed since the table can be copied and used as it is.

CalculateCrcWithoutInversion calculates a 32-bit checksum and return the result without a bit-inversion.

Calculate calls the CalculateCrcWithoutInversion method and returns the bit-inverted result.

IsOk check that content of a memory block – including CRC – is correct. (If the contents is correct, calculating a new CRC of both data and the old CRC will always give the same result called crcMagicWord in the code below.)

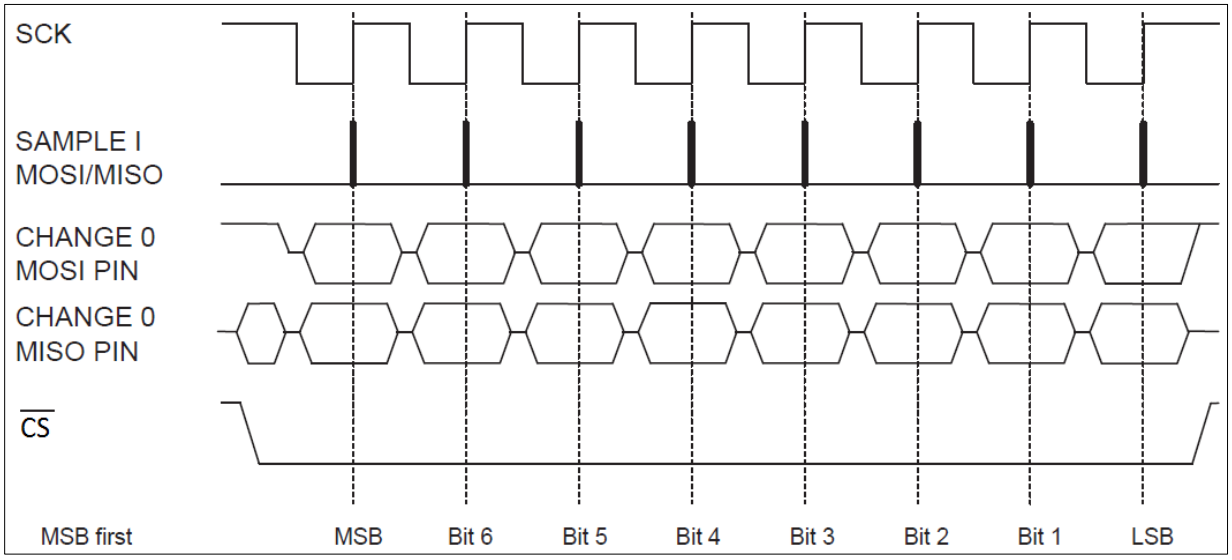
```
public static class Crc32
{
    // This class contains various methods for calculating CRC.

    // When re-calculating CRC, including the CRC itself, this will always be the result if the CRC is correct.
    private const UInt32 crcMagicWord = 0xDEBB20E3;

    private static void CalcTable()
    {
        //
        // This method will generate the look-up table for CRC calculation.
        // This method is here for reference only. The table is already generated and shown on the next page.
        UInt16 i, j;
        UInt32 Crc;
        UInt32[] CrcTable32 = new UInt32[256];

        UInt32 CRC_POLY = 0xEDB88320;

        for (i = 0; i < 256; i++)
        {
            Crc = i;
            for (j = 8; j > 0; j--)
            {
                if ((Crc & 1) != 0)
                    Crc = (Crc >> 1) ^ CRC_POLY;
                else
                    Crc >>= 1;
            }
            CrcTable32[i] = Crc;
        }
    }
```

```
        static UInt32[] crcTable_32 = new UInt32[256]
        {
            0x00000000, 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419, 0x706af48f, 0xe963a535, 0x9e6495a3,
            0x0edb8832, 0x79dcb8a4, 0xe0d5e91e, 0x97d2d988, 0x09b64c2b, 0x7eb17cbd, 0xe7b82d07, 0x90bf1d91,
            0x1db71064, 0x6ab020f2, 0xf3b97148, 0x84be41de, 0x1adad47d, 0x6ddde4eb, 0xf4d4b551, 0x83d385c7,
            0x136c9856, 0x646ba8c0, 0xfd62f97a, 0x8a65c9ec, 0x14015c4f, 0x63066cd9, 0xfa0f3d63, 0x8d080df5,
            0x3b6e20c8, 0x4c69105e, 0xd56041e4, 0xa2677172, 0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b,
            0x35b5a8fa, 0x42b2986c, 0xdbbbc9d6, 0xacbcf940, 0x32d86ce3, 0x45df5c75, 0xdcd60dcf, 0xabd13d59,
            0x26d930ac, 0x51de003a, 0xc8d75180, 0xbfd06116, 0x21b4f4b5, 0x56b3c423, 0xcfba9599, 0xb8bda50f,
            0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924, 0x2f6f7c87, 0x58684c11, 0xc1611dab, 0xb6662d3d,
            0x76dc4190, 0x01db7106, 0x98d220bc, 0xefd5102a, 0x71b18589, 0x06b6b51f, 0x9fbfe4a5, 0xe8b8d433,
            0x7807c9a2, 0x0f00f934, 0x9609a88e, 0xe10e9818, 0x7f6a0dbb, 0x086d3d2d, 0x91646c97, 0xe6635c01,
            0x6b6b51f4, 0x1c6c6162, 0x856530d8, 0xf262004e, 0x6c0695ed, 0x1b01a57b, 0x8208f4c1, 0xf50fc457,
            0x65b0d9c6, 0x12b7e950, 0x8bbeb8ea, 0xfcb9887c, 0x62dd1ddf, 0x15da2d49, 0x8cd37cf3, 0xfbd44c65,
            0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2, 0x4adfa541, 0x3dd895d7, 0xa4d1c46d, 0xd3d6f4fb,
            0x4369e96a, 0x346ed9fc, 0xad678846, 0xda60b8d0, 0x44042d73, 0x33031de5, 0xaa0a4c5f, 0xdd0d7cc9,
            0x5005713c, 0x270241aa, 0xbe0b1010, 0xc90c2086, 0x5768b525, 0x206f85b3, 0xb966d409, 0xce61e49f,
            0x5edef90e, 0x29d9c998, 0xb0d09822, 0xc7d7a8b4, 0x59b33d17, 0x2eb40d81, 0xb7bd5c3b, 0xc0ba6cad,
            0xedb88320, 0x9abfb3b6, 0x03b6e20c, 0x74b1d29a, 0xead54739, 0x9dd277af, 0x04db2615, 0x73dc1683,
            0xe3630b12, 0x94643b84, 0x0d6d6a3e, 0x7a6a5aa8, 0xe40ecf0b, 0x9309ff9d, 0x0a00ae27, 0x7d079eb1,
            0xf00f9344, 0x8708a3d2, 0x1e01f268, 0x6906c2fe, 0xf762575d, 0x806567cb, 0x196c3671, 0x6e6b06e7,
            0xfed41b76, 0x89d32be0, 0x10da7a5a, 0x67dd4acc, 0xf9b9df6f, 0x8ebeeff9, 0x17b7be43, 0x60b08ed5,
            0xd6d6a3e8, 0xa1d1937e, 0x38d8c2c4, 0x4fdff252, 0xd1bb67f1, 0xa6bc5767, 0x3fb506dd, 0x48b2364b,
            0xd80d2bda, 0xaf0a1b4c, 0x36034af6, 0x41047a60, 0xdf60efc3, 0xa867df55, 0x316e8eef, 0x4669be79,
            0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236, 0xcc0c7795, 0xbb0b4703, 0x220216b9, 0x5505262f,
            0xc5ba3bbe, 0xb2bd0b28, 0x2bb45a92, 0x5cb36a04, 0xc2d7ffa7, 0xb5d0cf31, 0x2cd99e8b, 0x5bdeae1d,
            0x9b64c2b0, 0xec63f226, 0x756aa39c, 0x026d930a, 0x9c0906a9, 0xeb0e363f, 0x72076785, 0x05005713,
            0x95bf4a82, 0xe2b87a14, 0x7bb12bae, 0x0cb61b38, 0x92d28e9b, 0xe5d5be0d, 0x7cdcefb7, 0x0bdbdf21,
            0x86d3d2d4, 0xf1d4e242, 0x68ddb3f8, 0x1fda836e, 0x81be16cd, 0xf6b9265b, 0x6fb077e1, 0x18b74777,
            0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c, 0x8f659eff, 0xf862ae69, 0x616bffd3, 0x166ccf45,
            0xa00ae278, 0xd70dd2ee, 0x4e048354, 0x3903b3c2, 0xa7672661, 0xd06016f7, 0x4969474d, 0x3e6e77db,
            0xaed16a4a, 0xd9d65adc, 0x40df0b66, 0x37d83bf0, 0xa9bcae53, 0xdebb9ec5, 0x47b2cf7f, 0x30b5ffe9,
            0xbdbdf21c, 0xcabac28a, 0x53b39330, 0x24b4a3a6, 0xbad03605, 0xcdd70693, 0x54de5729, 0x23d967bf,
            0xb3667a2e, 0xc4614ab8, 0x5d681b02, 0x2a6f2b94, 0xb40bbe37, 0xc30c8ea1, 0x5a05df1b, 0x2d02ef8d
        };


    private static UInt32 CalculateCrcWithoutInversion(List<byte> array, UInt16 startAddr, UInt16 endAddr)
        {
            // Calculate the CRC of a section of "array", including the data at "startAddr" and "endAddr".
            // Note that this method does not invert the bits. This is because the same method is used both
            // for calculating the CRC (bits should be inverted) and checking that the data including the CRC
            // is corrct (no inversion).
            UInt32 crc = 0xFFFFFFFF;
            int currentByte = startAddr;
            byte idx;

            while (currentByte <= endAddr)
            {
                idx = (byte)(crc ^ array[currentByte]);
                crc = (crc >> 8) ^ crcTable_32[idx];
                currentByte++;
            }

            return crc;
        }

        public static UInt32 Calculate(List<byte> array, UInt16 startAddr, UInt16 endAddr)
        {
            return ~CalculateCrcWithoutInversion(array, startAddr, endAddr);
        }

        public static bool IsOk(List<byte> array, UInt16 startAddr, UInt16 endAddr)
        {
            return (CalculateCrcWithoutInversion(array, startAddr, endAddr) == crcMagicWord);
        }

    }
```

B. Communication example

SPI:

Leading edge: Setup (Falling).
Trailing edge: Sample (Rising).
SCK Rising flange. Idle high.
MSB First.
/CS Activ Low.



EEPROM:

Example read the EEPROM:

CS = 0	// Enable EEPROM.
Write 0x03	// Read Command.
Write 0x00	// Address High.
Write 0x00	// Address Low.
Read 0xhh	// Data 0.
Read 0xhh	// Data 1.
Read 0xhh	// Data 2.
. . .	
. . .	// Data 3 – 32765.
. . .	
Read 0xhh	// Data 32766.
Read 0xhh	// Data 32767.
CS = 1	// Disable EEPROM.

To start reading from another address, use address bytes in the beginning.

ADC:

RESET: Send four bytes of 0xFF to the ADC.

CS = 0			// Enable ADC.
Write 0xFF			
Write 0xFF			
Write 0xFF			
Write 0xFF			
CS = 1			// Disable ADC.

After a reset command you have to wait for 500us before writing to any register.

SET CONTENTS: Example for initialize of temperature.

EEprom address:	Dec	Hex	
CS = 0			// Enable ADC.
Write 0x08			// MODE_REGISTER.
Write 0x40	25	0x19	// mode byte 2
Write 0x01	24	0x18	// mode byte 1
Write 0x28			// IO register.
Write 0x02	28	0x1C	// IO Byte 1
Write 0x10			// CONFIG_REGISTER.
Write 0x00	69	0x45	// config byte 2
Write 0x01	68	0x44	// config byte 1
Write 0x30			// OFFSET_REGISTER.
Write 0xC6	74	0x50	// offset byte 3
Write 0x5D	73	0x49	// offset byte 2
Write 0x40	72	0x48	// offset byte 1
Write 0x38			// GAIN_REGISTER.
Write 0x54	78	0x4E	// gain byte 3
Write 0xAD	77	0x4D	// gain byte 2
Write 0x00	76	0x4C	// gain byte 1
CS = 1			// Disable ADC.

Send this one more time with CONFIG, OFFSET & GAIN for pressure. (Same register address).

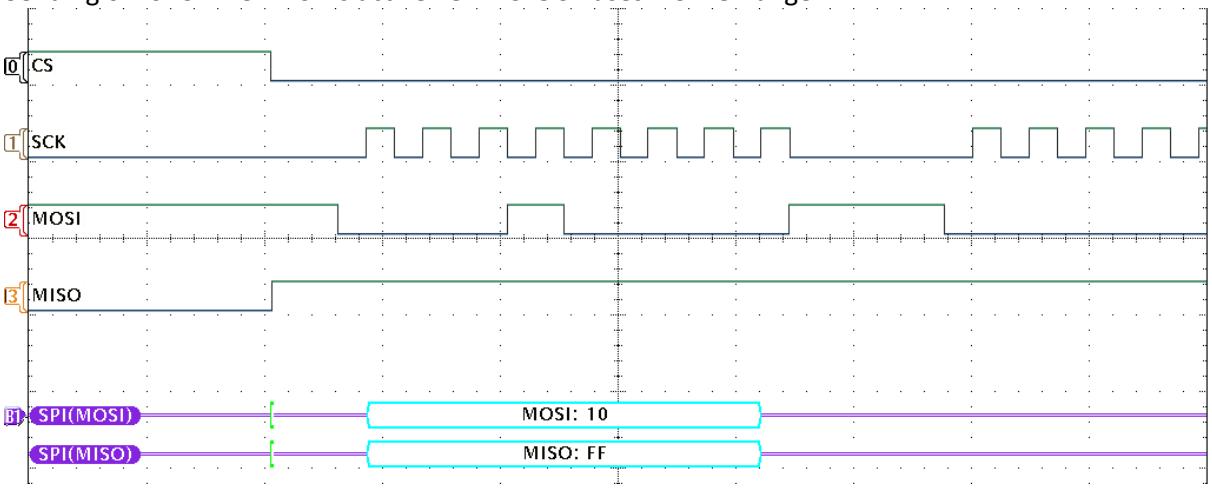
READ: Example for reading temperature:

CS = 0			// Enable ADC.
Write 0x10			// Config register
Write 0x00	33	0x21	// conf byte 1
Write 0x01	32	0x20	// conf byte 2
Write 0x08			// Mode register
Write 0x20			// mode byte 1 (Single mode! Eeprom is idle.)
Write 0x01	24	0x18	// mode Byte 2
Wait for MISO to go low.			// ADC Conversion time.
Write 0x58			// Read data command.
Read 0xhh			// Adc data 1
Read 0xhh			// Adc data 2
Read 0xhh			// Adc data 3
CS = 1			// Disable ADC.

SPI Mode

Example on SPI transmission:

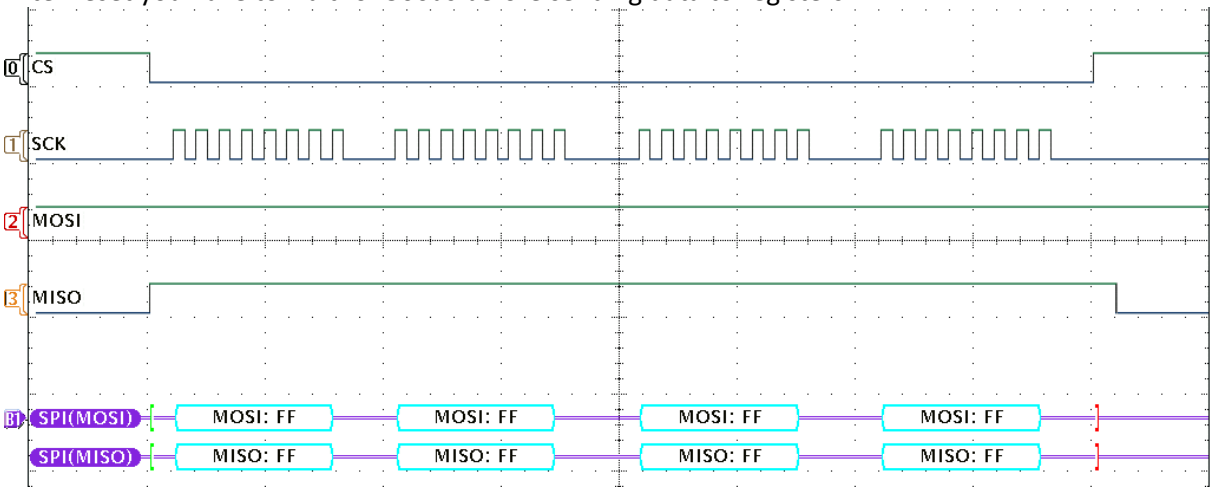
Sending 0x10. CHIP SELECT is active LOW. CLOCK uses RISING flange.



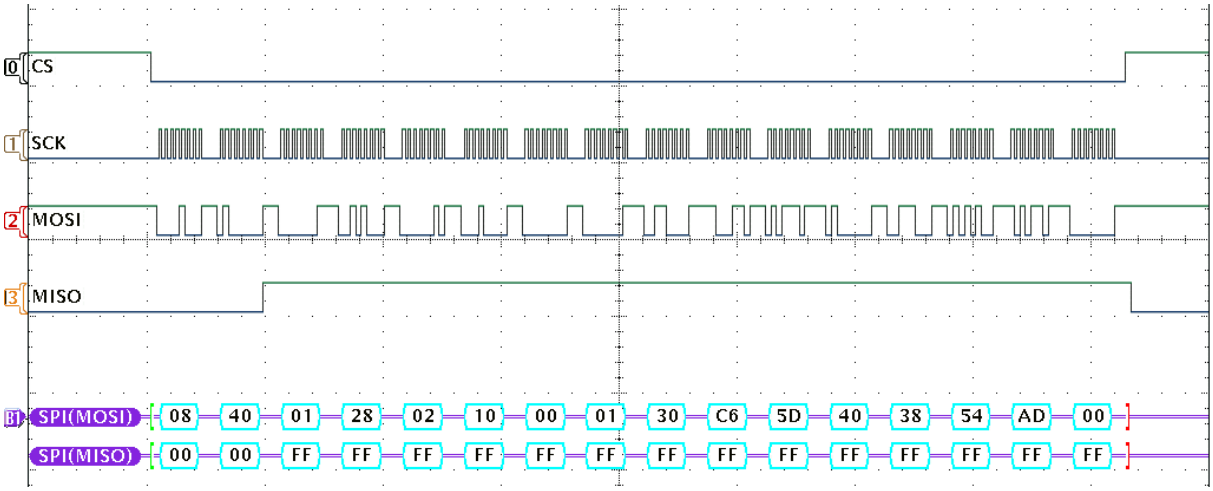
RESET COMMAND

Send four byte of 0xFF to the ADC to reset.

After reset you have to wait for 500us before sending data to registers.

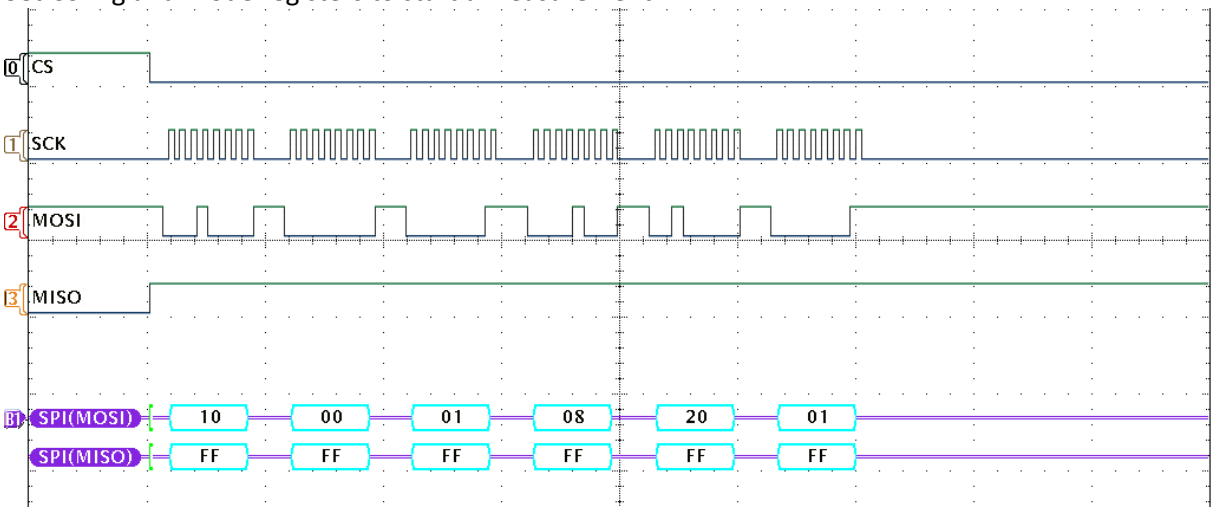


SET CONTENT COMMAND



MEASURE COMMAND

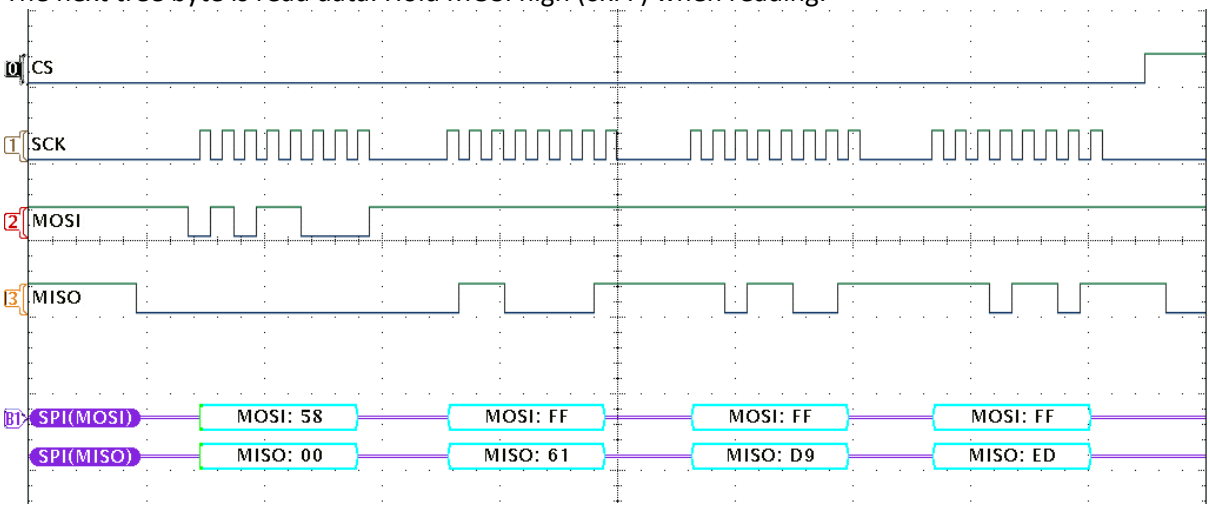
Set Config and Mode registers to start a measurement.



The ADC hold the MISO line HIGH to the measurement is done. Takes about 5ms.
When the MISO goes LOW you can read the measured data.

READ COMMAND

When the MISO is LOW you can read the data by sending 0x58.
The next tree byte is read data. Hold MOSI high (0xFF) when reading.



In this example we read 0x61 D9 ED. This is 6.412.781 in decimal value.
As long as CS is LOW you can repeat the read command for reading the same data.
For new measure, send the MEASURE COMMAND.

TROUBLESHOOT:

Don’t receive data on MISO:
Is MISO an input in your system?
Is ADC the only device on the MISO line that is active? (Often a programmer use this line).