1 Introduction

This document describes the calibration process for the E703.x1 pressure sensor signal conditioner (PSSC) IC-family.

2 Calibration Sequence

The calibration sequence can be divided into five steps:

1. Setup environment
2. Collect calibration raw data
3. Calculate coefficients
4. Program coefficients in non-volatile memory (NVM)
5. Verify calibration

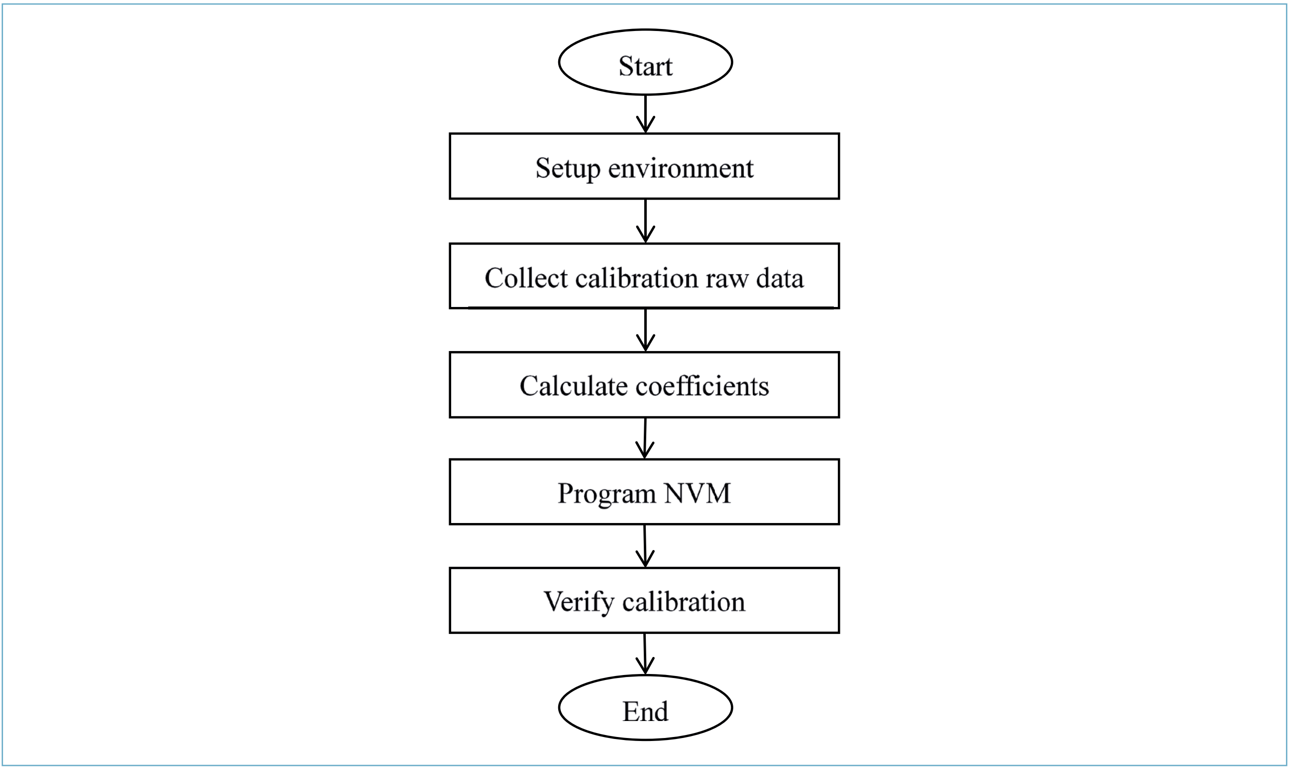


Figure 1. Calibration Flowchart

There are special cases, where it is necessary to add steps to the calibration sequence or extend a single step. For example, if the analog output should be re-calibrated over temperature, additional raw data must be collected for each temperature point. These special cases are explained in this document.

2.1 Setup environment

Before collecting raw data for calibration, the analog front end (AFE) of the chip must be initialized within working registers (RAM). These registers are listed in Table 1. Refer to the chip's data sheet for bit level descriptions of these registers.

The following function within "pssc_com_lib.dll" may be used to initialize each of these registers individually in RAM;
`pssc_com_write_reg (com_t handle, uint8_t address, uint16_t data);`

Alternatively, since these registers are located in consecutive word addresses, the following function may also be used.
`pssc_com_write_regs (com_t handle, uint8_t address, uint32_t words_to_write, uint16_t data[]);`

Table 1. These registers must be configured in RAM before collecting raw data for calibration

RAM address	NVM address	Name	Description
0x14	0x30	CFG_ADC	analog to digital converter settings (see data sheet register description for more details)
0x16	0x32	CFG_LP0	low pass filter settings (see data sheet register description for more details)
0x18	0x34	CFG_LP1	low pass filter settings (see data sheet register description for more details)
0x1A	0x36	CFG_AFE0	analog front-end settings (see data sheet register description for more details)
0x1C	0x38	CFG_AFE1	analog front-end settings (see data sheet register description for more details)
0x1E	0x3A	CFG_AFE2	analog front-end settings (see data sheet register description for more details)

2.1.1 Configuration Array

The calculation routines of the calibration DLL will use chip's configuration information that are stored in chip's non-volatile memory (NVM). Therefore, a configuration array consisting of chip's NVM must be generated and its array pointer is passed to the calibration DLL. The configuration array will be used to transfer data to and from calibration DLL.

One option is to configure the AFE registers first and collect raw data and then generate configuration array before calling calibration DLL. It is also possible to generate a complete configuration array, with desired AFE configurations in its fields, and use it throughout calibration process including, collecting raw data, coefficients calculations, verification, and finally write its contents to the chip's NVM after a successful calibration. This method is used in this document and this section describes how to generate a complete and valid configuration array.

Calibration DLL and all PSSC libraries require that the configuration array be declared as "cfg_t" type, defined in "pssc_cfg_data.h" file.

The configuration array will consist of 64 words of chip's NVM with one additional word for chip unique version information.

Table 2 shows the complete structure of the configuration array.

Table 2. Structure of configuration array

	Array cfg[]	
	Description	This array contains the complete IC configuration of NVM. It will be used to transfer data to and from the calibration DLL routines.
	Declaration	unsigned short 65x16 bit values
	Entries of cfg[x] array	
0	SER0	serial number low word
1	SER1	serial number high word
2	RATIO_DAC01	analog output temperature compensation coefficients
3	RATIO_DAC23	analog output temperature compensation coefficients
4	RATIO_DAC45	analog output temperature compensation coefficients
5	ABS_V_DAC01	analog output temperature compensation coefficients
6	ABS_V_DAC23	analog output temperature compensation coefficients
7	ABS_V_DAC45	analog output temperature compensation coefficients
8	ABS_I_DAC01	analog output temperature compensation coefficients
9	ABS_I_DAC23	analog output temperature compensation coefficients
10	ABS_I_DAC45	analog output temperature compensation coefficients
11	G01	sensor current compensation coefficients
12	G2OFF	sensor current compensation coefficients
13	CFG_CAL0	internal calibration value
14	CFG_CAL1	internal calibration value
15	CRC8	8 bit CRC over all words above
16	CFG_EN	operation mode

17	CFG_SPI_I2C	SPI and I2C settings
18	CFG_PADS0	further output settings
19	CFG_PADS1	further output settings
20	CFG_PADS2	further output settings
21	CFG_PERIOD	sample period settings
22	CFG_AODO	analog digital output settings
23	CFG_SBC_INTF	sensor bridge check and internal failure detection settings
24	CFG_ADC	ADC settings
25	CFG_LP0	low pass filter settings
26	CFG_LP1	low pass filter settings
27	CFG_AFE0	analog front-end settings
28	CFG_AFE1	analog front-end settings
29	CFG_AFE2	analog front-end settings
30	USER	user defined information
31	T_O	temperature pre-scaling offset
32	T_F	temperature pre-scaling gain
33	S_O	pressure pre-scaling offset
34	S_F	pressure pre-scaling gain
35	S0	pressure compensation coefficient
36	S1	pressure compensation coefficient
37	S2	pressure compensation coefficient
38	S3	pressure compensation coefficient
39	S4	pressure compensation coefficient
40	S5	pressure compensation coefficient
41	S6	pressure compensation coefficient
42	S7	pressure compensation coefficient
43	S8	pressure compensation coefficient
44	S9	pressure compensation coefficient
45	S10	pressure compensation coefficient
46	S11	pressure compensation coefficient
47	S12	pressure compensation coefficient
48	S13	pressure compensation coefficient
49	S14	pressure compensation coefficient
50	S15	pressure compensation coefficient
51	T0	absolute temperature compensation coefficient
52	T1	absolute temperature compensation coefficient
53	T2	absolute temperature compensation coefficient
54	T3	absolute temperature compensation coefficient
55	ALARM_LO	alarm output low limit
56	ALARM_HI	alarm output high limit
57	AODO_O	analog/digital output scaling offset
58	AODO_F	analog/digital output scaling gain
59	LIMIT_LO	analog/digital output limitation low value
60	LIMIT_HI	analog/digital output limitation high value
61	ERR_MASK	error mask for analog/digital output
62	ERR_VAL_HI_LO	error value for analog/digital output level setting
63	CRC16	16 bit CRC over all words above
64	VERSION	chip version value

2.1.2 Generating initial configuration array

This section describe several ways to generate a valid configuration array. Figure 2 shows recommended steps.

Direct configuration array creation
User may declare the configuration array within calibration software and initialize it with all necessary configurations manually. Alternatively, a configuration text file may be created and then imported to the calibration software. A configuration text file generated by PSSC GUI can be reviewed for format.

PSSC GUI
The simplest way to generate a configuration array for a chip, is the use of the PSSC GUI software. The GUI will guide the user to configure all necessary registers. All the entries of Table 1 can be configured on the analog front-end tab (AFE) of the GUI software. After initialization of all necessary fields in the GUI, the contents can be written in the chip's NVM or saved in a text file, or both.
Not all register bits can be set by the GUI, it is limited to the most common functions of the chip. Additional configuration may be done within user's calibration software.

PSSC configuration library
The "pssc_cfg_lib.dll" library was created to abstract user from the register bit-level. It contains functions which will set a combination of register bits to achieve the user defined behavior. The PSSC GUI uses this library to update and initialize configuration registers. A sample source codes utilizing functions with-in this library, "generate_configuration_example.c" is provided in the PSSC GUI software package.

After a valid configuration array is generated, its contents may be written to working registers (RAM) or NVM. Proceeding with the remaining calibration process, collecting raw data, depends on whether configuration array is stored in NVM or RAM.

1. Write the configuration to the non-volatile memory (NVM) of the chip. The "pssc_com_cfg_nvm_write (handle, &cfg)" library function writes contents of configuration array to the chip's NVM. Using this option makes it possible to switch off power supply to the device without losing the configurations.

   One of the followings reactivates the chip with the new configurations;
   a) A power cycle will start an automatic transfer of NVM contents to RAM.
   b) The "pssc_com_cfg_nvm_transfer (com_t handle)" library function transfers NVM contents to RAM during run time.

2. Write the configuration to the working register (RAM).
   The "pssc_com_cfg_write (handle, &cfg)" library function transfers contents of configuration array to working registers (RAM) of the chip. In this case the chip must be supplied all the time or the setup process must be repeated every time after each power up sequence.

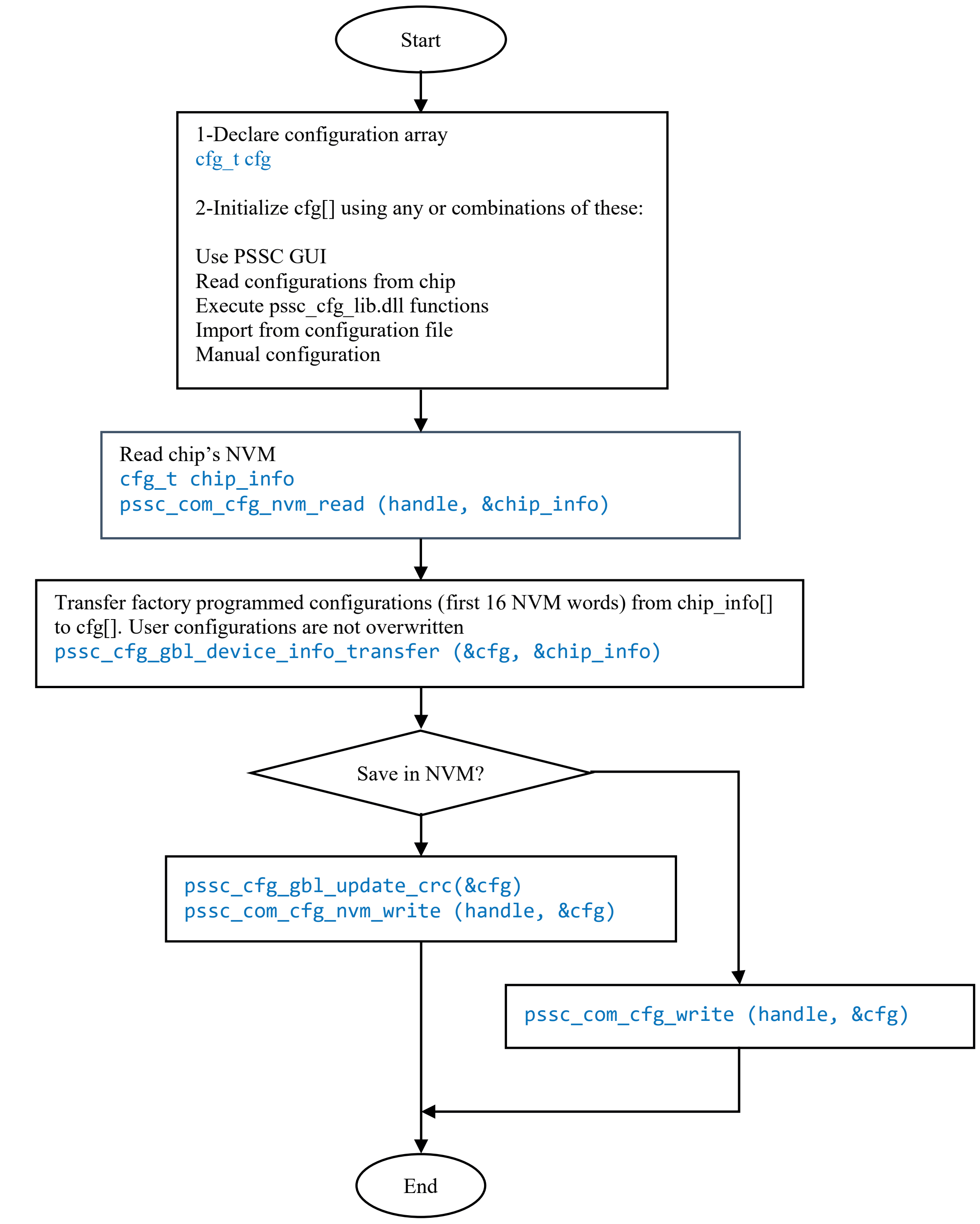

Start
1-Declare configuration array
cfg_t cfg
2-Initialize cfg[] using any or combinations of these:
Use PSSC GUI
Read configurations from chip
Execute pssc_cfg_lib.dll functions
Import from configuration file
Manual configuration
Read chip's NVM
cfg_t chip_info
pssc_com_cfg_nvm_read (handle, &chip_info)
Transfer factory programmed configurations (first 16 NVM words) from chip_info[] to cfg[]. User configurations are not overwritten
pssc_cfg_gbl_device_info_transfer (&cfg, &chip_info)
Save in NVM?
pssc_cfg_gbl_update_crc(&cfg)
pssc_com_cfg_nvm_write (handle, &cfg)
pssc_com_cfg_write (handle, &cfg)
End

Figure 2. Generating configuration array

2.2 Collect raw data for calibration

Raw data are read from the chip ADC after application of temperature and pressure for each selected calibration point. For accurate calibration it is necessary to wait and make sure both temperature and pressure have reached their target values, are stabilized, and sensor-module and all its components have reached temperature equilibrium.

In order to select the best values for AFE registers (listed in Table 1), it may become necessary to repeat collection of raw data at each calibration point with different AFE settings in the same calibration run. In this case, separate set of raw data must be collected for each of these configuration settings.

It is recommended to read multiple ADC values for each applied temperature and pressure for each calibration point and use average value to cancel measurement noise.

Figure 3 illustrate steps for collecting raw data for digital output.
Figure 4 illustrate steps for collecting raw data for voltage output or 4…20mA current loop.

Collecting raw data for selected Temperature source

1. Using the chip's internal temperature diode or an external temperature source.
   In this case only the values of ADC_T (raw temperature) and ADC_S (raw pressure) values should be read from the chip ADC.

2. Using the bridge resistance.
   In addition to the ADC_T and ADC_S the internal chip temperature diode ADC_TC also must be read from the chip. This is necessary to compensate temperature coefficient (TC) of the bridge resistance measurement shunt.

Number of calibration points

The required number of calibration points is determined by sensor characterization data and the order of polynomial selected for compensation. The minimum number of calibration points is 2 for 1st order pressure and 0th order temperature compensation. Two different pressure points is required. The maximum number of points is 10 for cubic, 3rd order pressure and 3rd order temperature compensation.

Pressure and temperature for calibration points should be selected to cover the end application's entire operating range with sufficiently large delta-pressure and delta-temperature between all points. Calibration points may be measured in any order. Figure 5…16 show reduced calibration points arrangements.

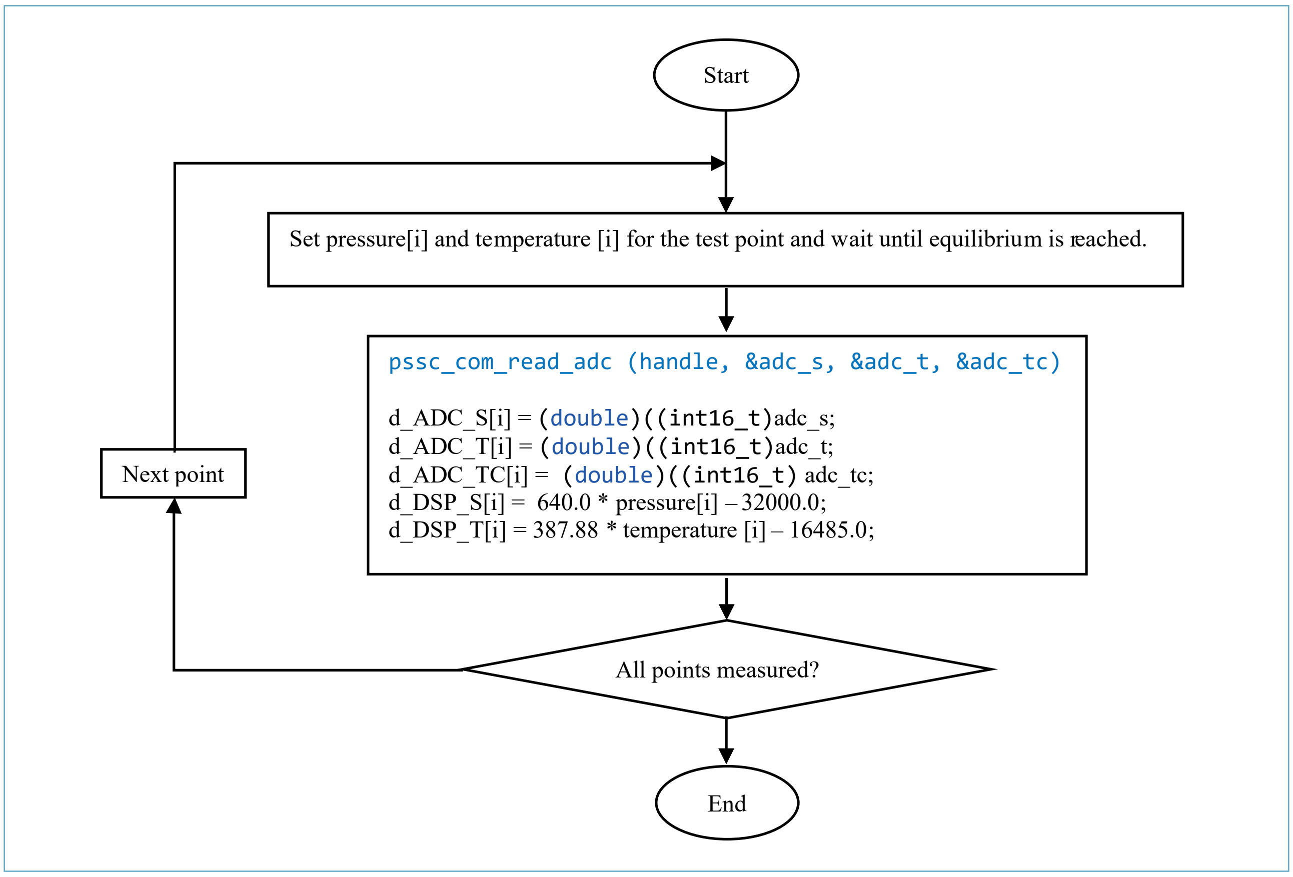
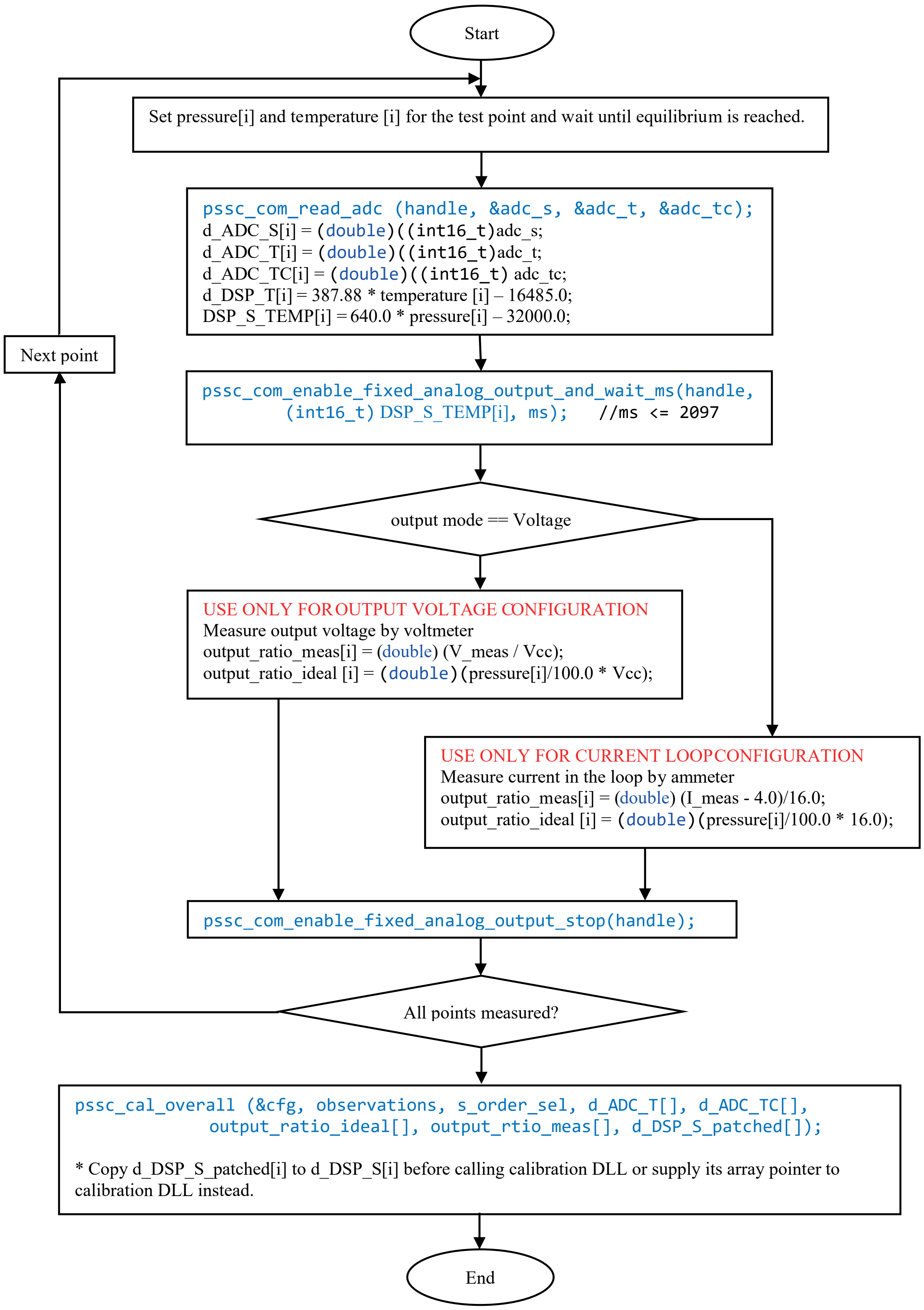

Start
Set pressure[i] and temperature [i] for the test point and wait until equilibrium is reached.
pssc_com_read_adc (handle, &adc_s, &adc_t, &adc_tc)
d_ADC_S[i] = (double)((int16_t)adc_s;
d_ADC_T[i] = (double)((int16_t)adc_t;
d_ADC_TC[i] = (double)((int16_t) adc_tc;
d_DSP_S[i] = 640.0 * pressure[i] – 32000.0;
d_DSP_T[i] = 387.88 * temperature [i] – 16485.0;
All points measured?
Next point
End

Figure 3. Collecting raw data for digital output

In this document:
(double) pressure[i] is in % of full scale input expressed in range of 0.0 to 100.0 for each test point.

(double) temperature [i] is in degree Celsius in the rage of -40.0 to +125.0 for each test point.


Start
Set pressure[i] and temperature [i] for the test point and wait until equilibrium is reached.
Next point
output mode == Voltage
USE ONLY FOR OUTPUT VOLTAGE CONFIGURATION
Measure output voltage by voltmeter
USE ONLY FOR CURRENT LOOP CONFIGURATION
Measure current in the loop by ammeter
All points measured?
* Copy d_DSP_S_patched[i] to d_DSP_S[i] before calling calibration DLL or supply its array pointer to calibration DLL instead.
End

```
pssc_com_read_adc (handle, &adc_s, &adc_t, &adc_tc);
d_ADC_S[i] = (double)((int16_t)adc_s;
d_ADC_T[i] = (double)((int16_t)adc_t;
d_ADC_TC[i] = (double)((int16_t) adc_tc;
d_DSP_T[i] = 387.88 * temperature [i] – 16485.0;
DSP_S_TEMP[i] = 640.0 * pressure[i] – 32000.0;
```

```
pssc_com_enable_fixed_analog_output_and_wait_ms(handle,
        (int16_t) DSP_S_TEMP[i], ms);   //ms <= 2097
```

```
output_ratio_meas[i] = (double) (V_meas / Vcc);
output_ratio_ideal [i] = (double)(pressure[i]/100.0 * Vcc);
```

```
output_ratio_meas[i] = (double) (I_meas - 4.0)/16.0;
output_ratio_ideal [i] = (double)(pressure[i]/100.0 * 16.0);
```

```
pssc_com_enable_fixed_analog_output_stop(handle);
```

```
pssc_cal_overall (&cfg, observations, s_order_sel, d_ADC_T[], d_ADC_TC[],
        output_ratio_ideal[], output_rtio_meas[], d_DSP_S_patched[]);
```

Figure 4. Collecting raw data for analog output

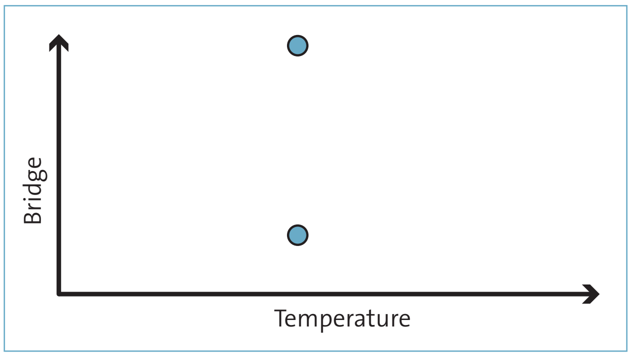


Figure 5. linear bridge, 0 order temperature

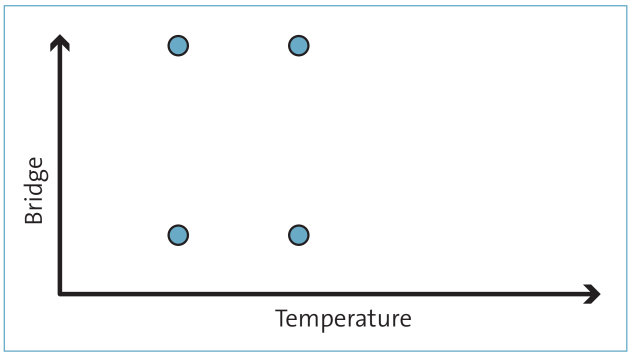


Figure 6. linear bridge, 1st order temperature

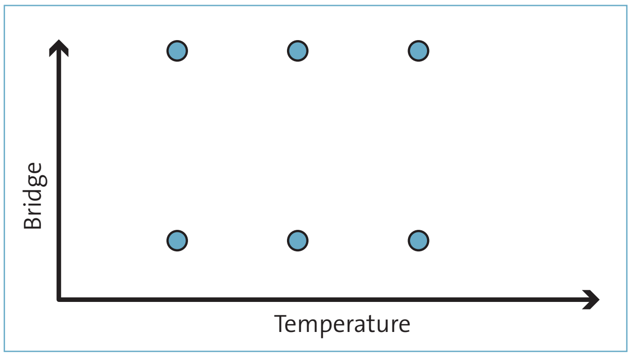


Figure 7. linear bridge, 2nd order temperature

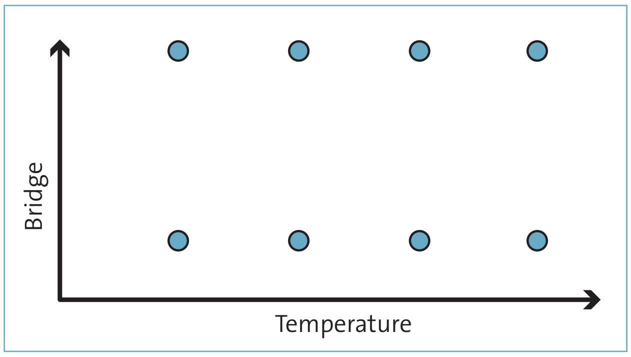


Figure 8. linear bridge, 3rd order temperature

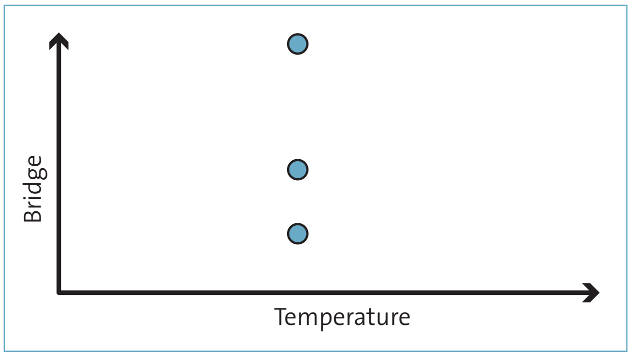


Figure 9. 2nd order bridge, 0 order temperature

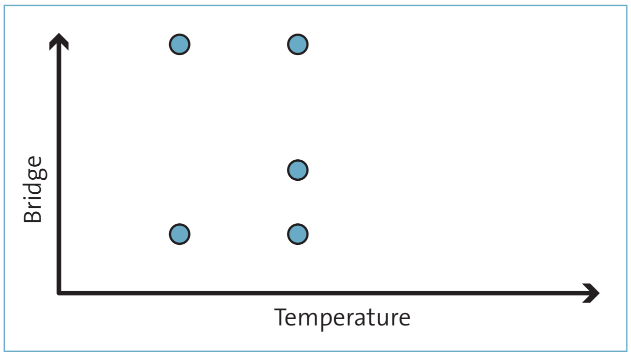


Figure 10. 2nd order bridge, 1st order temperature

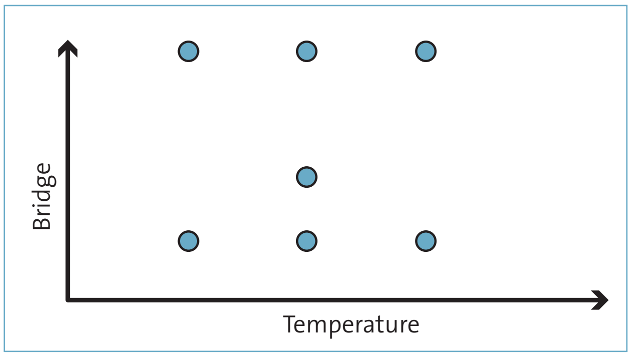


Figure 11. 2nd order bridge, 2nd order temperature

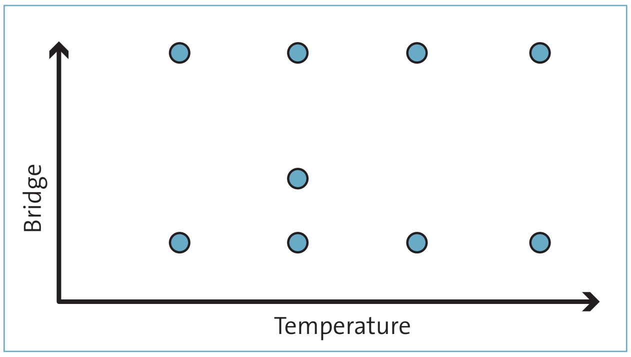


Figure 12. 2nd order bridge, 3rd order temperature

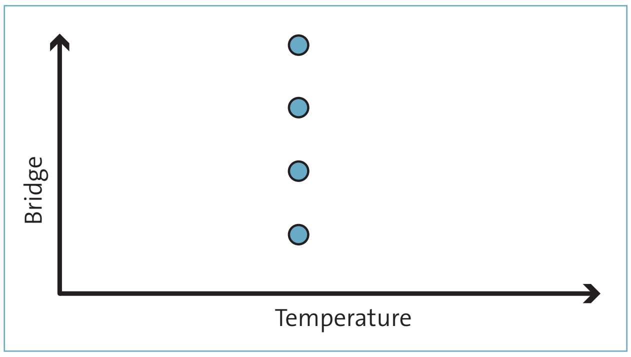


Figure 13. 3rd order bridge, 0 order temperature

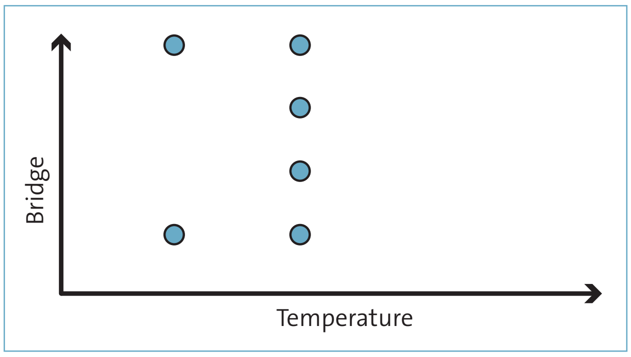


Figure 14. 3rd order bridge, 1st order temperature

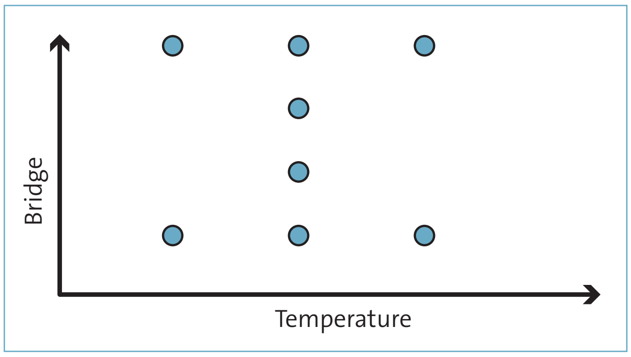


Figure 15. 3rd order bridge, 2nd order temperature

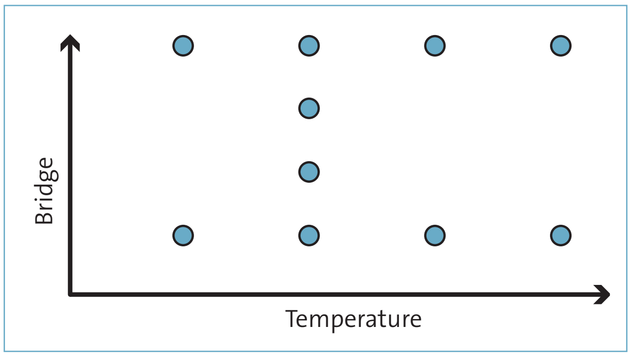


Figure 16. 3rd order bridge, 3rd order temperature

2.3 Calculating polynomial coefficients

To calculate the coefficients for the correction polynomial, the raw data and other required information, as described in Table 3, must be supplied to the calibration DLL. The calibration DLL for Windows is located in “pssc_cal_lib.dll” and in “pssc_cal_lib.so” for Linux.

The calibration function calculate the calibration coefficients and store them in the configuration array. Additional accuracy information will also be provided by the calibration DLL.

In the case the calibration DLL function returns with an error, the configuration array can contain invalid information and should not be written to the chip’s NVM otherwise, defined behavior of the chip is not guaranteed. It is recommended to evaluate the fitting information provided by calibration DLL.

Function pssc_cal_calibration	
Description	Default routine to calculate absolute temperature and pressure compensation coefficients from raw ADC data.
Declaration	see below

Declaration:

```
cal_err_code  pssc_cal_calibration(
                      cfg_t        *cfg,
                      int           t_order_sel,
                      int           s_order_sel,
                      int           observations,
                      double        d_ADC_T[],
                      double        d_ADC_TC[],
                      double        d_ADC_S[],
                      double        d_DSP_T[],
                      double        d_DSP_S[],
                      fit_info     *t_fit_info,
                      double        t_err_at_point[],
                      fit_info     *s_fit_info,
                      double        s_err_at_point[]
                      );
```

Return value pssc_cal_calibration

cal_err_code: The function returns an integer, as error code, with the following interpretation:

Name	Value	Description
NO_ERROR	0	No error occurred
UNDEF	1	General error occurred
INSUFFICIENT_POINTS	2	The fit function has not sufficient sampling points for the (user defined) coefficient selection.
OUT_OF_SCALE_T	3	One of the temperature coefficients is out of scale, too small ADC values
OUT_OF_SCALE_S	4	One of the sensor coefficients is out of scale, too small ADC values

Parameter of the pssc_cal_calibration

Parameter	Description
*cfg	Pointer to a configuration array of the chip which has been uploaded before measuring calibration data at the selected points. The function will use the configurations to adapt the calculations. All calculated coefficients will be stored within this array.
t_order_sel	Polynomial order selection for the absolute temperature compensation (DSP_T). The following values are possible:

Name	Value	Description
T_AUTO_ORDER	-1	Automatic detection of compensation order based on the number of different temperature points taken.
T_CONST	0	Constant result
T_1ST_ORDER	1	1st order compensation
T_2ND_ORDER	2	2nd order compensation
T_3RD_ORDER	3	3rd order compensation

s_order_sel	Polynomial order selection for the pressure compensation (DSP_S). The following values are possible:

Name	Value	Description
S_AUTO_ORDER	-1	Automatic detection of compensation order based on the number of different temperature points taken.
bit field	0x0000 to 0xFFFF	A 16-bit field, where each bit corresponds to one of the 16 possible coefficients. Example: 0xFFFF results in the usage of all coefficients and 0x0011 will only select coefficient S0 and S4 (pressure value offset and gain).

observations	Integer number of observation points.
d_ADC_T[]	Array of raw temperature for observation points. These values represent the temperature information for compensating the bridge TC or/and as absolute temperature information.
d_ADC_TC[]	Array of raw chip temperature for observation points. This values must be given when the sensor temperature setting is chosen. They will be used to compensate the TC of the sensor bridge current measurement.
d_ADC_S[]	Array of raw pressure for observation points.
d_DSP_T[]	Array of expected compensated absolute temperature output values corresponding to the d_ADC_T and d_ADC_TC raw data. It must be calculated for each test point. d_DSP_T[]= 387.88 * T – 16485.0 (T is applied temperature for each observation point entered in degree Celsius and it must be in the range of -40 to +125).
d_DSP_S[]	Array of expected compensated pressure output values corresponding to the observation raw data. It must be calculated for each test point. d_DSP_S[]= 640.0 * P – 32000.0 (P is applied pressure for each observation point entered in % of full scale and it must be in the range of 0 to 100).

*t_fit_info	Pointer to structure of absolute temperature compensation fitting information declared as;
	`Fit_info t_fit_info;`
	
	`typedef struct {`
	`  double err_max;`
	`  double err_rms;`
	`  double used_adc_t_range;`
	`  double used_adc_s_range;`
	`  double used_adc_ts_area;`
	`  double sample_density;`
	`} fit_info;`

t_err_at_point[]	Array of differences between compensated curve and each given temperature observation point.

*s_fit_info	Pointer to structure of pressure compensation fitting information declared as:
	`Fit_info s_fit_info;`
	
	`typedef struct {`
	`  double err_max;`
	`  double err_rms;`
	`  double used_adc_t_range;`
	`  double used_adc_s_range;`
	`  double used_adc_ts_area;`
	`  double sample_density;`
	`} fit_info;`

s_err_at_point[]	Array of differences between compensated curve and each given pressure observation point:

2.4 Programming configuration array to non-volatile memory

After a successful calibration, the configuration array contains all of required information and may be stored permanently in the chip's NVM. Any manual change of configuration array after calibration requires recalculation of CRC.

The communication library provides all necessary functions to read and write from/to all chip's NVM as well as reading a single word. Writing a single word alone is not supported because it will result in mismatch of CRC. However, it can be accomplished by reading all NVM into an array (cfg_t type), change target register(s), calculate new CRC and write array back to the chip's NVM.

To program the NVM without using the communication library, please refer to the following technical note describing this procedure in detail:
703x1_write_nvm_xxxxxxxx.pdf

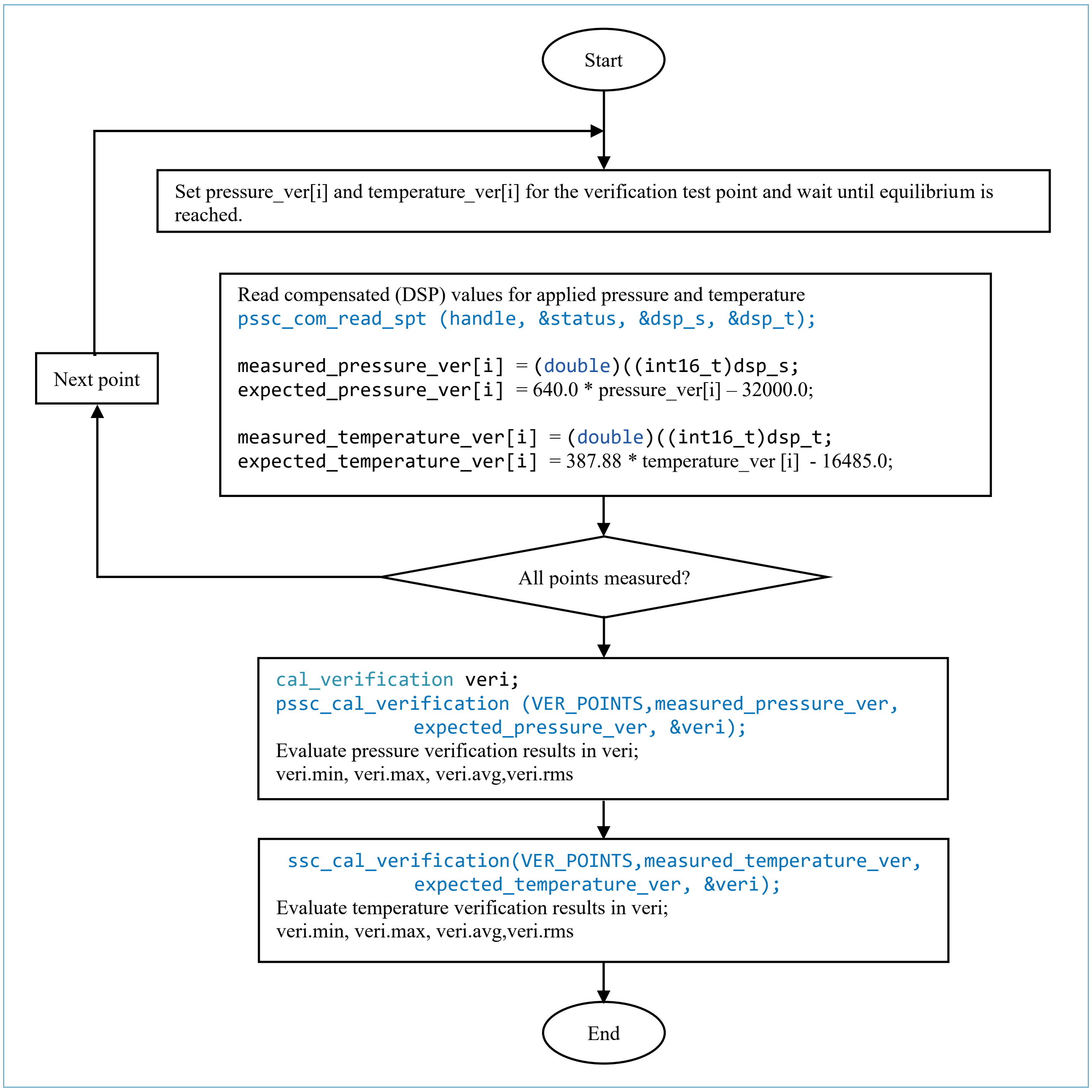

Start
Set pressure_ver[i] and temperature_ver[i] for the verification test point and wait until equilibrium is reached.
Read compensated (DSP) values for applied pressure and temperature
pssc_com_read_spt (handle, &status, &dsp_s, &dsp_t);
measured_pressure_ver[i] = (double)((int16_t)dsp_s;
expected_pressure_ver[i] = 640.0 * pressure_ver[i] – 32000.0;
measured_temperature_ver[i] = (double)((int16_t)dsp_t;
expected_temperature_ver[i] = 387.88 * temperature_ver [i] - 16485.0;
Next point
All points measured?
cal_verification veri;
pssc_cal_verification (VER_POINTS,measured_pressure_ver, expected_pressure_ver, &veri);
Evaluate pressure verification results in veri;
veri.min, veri.max, veri.avg,veri.rms
ssc_cal_verification(VER_POINTS,measured_temperature_ver, expected_temperature_ver, &veri);
Evaluate temperature verification results in veri;
veri.min, veri.max, veri.avg,veri.rms
End

Figure 17. Calibration verification

2.5 Verify calibration

In order to guarantee a calibrated device will measure pressure and temperature correctly in the end application, it is recommended to verify its output with one or more different pressure(s) and temperature(s) than those used for the calibration. Verification step may be included as integrated part of calibration process or it also may be done after calibration. In the first case, the configuration array which contains newly calculated calibration coefficients by the calibration DLL must be written to the working registers (RAM) before start of verification process as illustrated in Figure 17.
The application output values will be measured at each verification point and compared to the expected value. The calibration library contain a verification functions which can be used to generate accuracy information (minimum, maximum, average and RMS error).

3 Appendix

3.1 Examples

The PSSC GUI software can be used together with MAZ calibration hardware kits to perform the complete calibration sequence. To perform a calibration with PSSC GUI software, please refer to: pssc_eval_kit_xxxxxxxx.pdf

Complete source codes for single chip calibration and mass calibration are included in the PSSC GUI software package.

3.2 CRC calculation source codes

The library function "pssc_cfg_gbl_update_crc(&cfg)" calculates CRC8 and CRC16 of contents of configuration array (cfg[]) and stores them in CRC8 and CRC16 fields. Calculation of CRC8 is not necessary unless factory programmed values are modified. The source codes are provided here for additional information.

```
#define NVM_CRC_8_POLY        0x07
#define NVM_CRC_8_INIT        0xFF

#define NVM_CRC_16_POLY       0x8005
#define NVM_CRC_16_INIT       0xFFFF

#define CRC_8                         8       // bit
#define CRC_16                        16      // bit
#define SIZE_OF_CFG_ARRAY     65      // words

//===========================================================================
uint16_t NVM_CRC (uint16_t crc_length, uint16_t crc_polynomial, uint16_t crc_value,
                        uint16_t* data, uint32_t number_of_bits)
{
   uint16_t crc_bit;
   uint16_t data_bit;
   uint32_t    i;

   for (i = 0; i < number_of_bits; i++)
   {
      crc_bit = (crc_value >> (crc_length - 1)) & 0x0001;
      data_bit = (data[i >> 4] >> (((15 - (i % 16)) + 16 - crc_length) % 16)) & 0x0001;
      if (crc_bit != data_bit)
         crc_value = (crc_value << 1) ^ crc_polynomial;
      else
         crc_value = crc_value << 1;
   }

   return crc_value & (~(0xFFFF << crc_length));
}
//===============================================================================
// Example:
void calculate_crc()
{
   uint8_t i;
   uint16_t    data[SIZE_OF_CFG_ARRAY];
   uint16_t    crc16;
uint8_t        crc8;

   for (i=0; i<SIZE_OF_CFG_ARRAY; i++)
      data[i]= cfg.mem[i];
   // calculate crc8 for the first 15 words plus low-byte of CRC8 filed. The high byte of
// CRC8 field is not included.
//
crc8 = (uint8_t) NVM_CRC (CRC_8, NVM_CRC_8_POLY, NVM_CRC_8_INIT, data, (15*16+ 8));
//
// write crc8 to high byte of CRC8 field.
   cfg.mem[15]= (crc8 << 8) | ((uint8_t) cfg.mem[15]);

   // calculate crc16 of the first 63 (0…62) words of configuration array. Contents of
// CRC16 and VERSION fields are not included.
   crc16 = NVM_CRC (CRC_16, NVM_CRC_16_POLY, NVM_CRC_16_INIT, data, (63*16));
   cfg.mem[63]= crc16;
}
```

Usage Restrictions

Elmos Semiconductor AG provide the E703.x1 Demonstration Board simply and solely for IC evaluation purposes in laboratory. The Kit or any part of the Kit must not be used for other purposes or within non laboratory environments. Especially the use or the integration in production systems, appliances or other installations is prohibited.

The pcb´s are delivered to customer are for the temporary purpose of testing, evaluation and development of the Elmos IC´s only. Elmos will not assume any liability for additional applications of the pcb.

Disclaimer

Elmos Semiconductor AG shall not be liable for any damages arising out of defects resulting from (1) delivered hardware or software, (2) non observance of instructions contained in this document, or (3) misuse, abuse, use under abnormal conditions or alteration by anyone other than Elmos Semiconductor AG. To the extend permitted by law Elmos Semiconductor AG hereby expressively disclaims and user expressively waives any and all warranties of merchantability and of fitness for a particular purpose, statutory warranty of non-infringement and any other warranty or product liability that may arise by reason of usage of trade, custom or course of dealing.

Elmos Semiconductor AG – Headquarters
Heinrich-Hertz-Str. 1 | 44227 Dortmund | Germany
Phone +49 (0) 231-75 49-100 | Fax +49 (0) 231-75 49-159
sales-germany@elmos.com | www.elmos.com