

PICOCAP®

Data Sheet

PCapØ2 Linearize

Description of Linearization Firmware
Version 03.02.xx

November 4th 2013

Document-No: DB_PCapØ2_Linearize_en VO.2

Published by acam-messelectronic gmbh

©acam-messelectronic gmbh 2013

Legal note

The present manual (data sheet and guide) is still under development, which may result in corrections, modifications or additions. acam cannot be held liable for any of its contents, neither for accuracy, nor for completeness. The compiled information is believed correct, though some errors and omissions are likely. We welcome any notification, which will be integrated in succeeding releases.

The acam recommendations are believed useful, the firmware proposals and the schematics operable, nevertheless it is of the customer's sole responsibility to modify, test and validate them before setting up any production process.

acam products are not designed for use in medical, nuclear, military, aircraft, spacecraft or life support devices. Nor are they suitable for applications where failure may provoke injury to people or heavy material damage. acam declines any liability with respect to such non-intended use, which remains under the customer's sole responsibility and risk. Military, spatial and nuclear use subject to German export regulations.

acam do not warrant, and it is not implied that the information and/or practice presented here is free from patent, copyright or similar protection. All registered names and trademarks are mentioned for reference only and remain the property of their respective owners. The acam logo and the PICOCAP logo are registered trademarks of acam-messelectronic gmbh, Germany.

Support / Contact

For a complete listing of Direct Sales, Distributor and Sales Representative contacts, visit the acam web site at:

<http://www.acam.de/sales/distributors/>

For technical support you can contact the acam support team in the headquarters in Germany or the Distributor in your country. The contact details of acam in Germany are:

support@acam.de

or by phone

+49-7244-74190.

Content

1	Introduction	1-1
1.1	Non-Linearity and Temperature Dependence	1-1
1.2	Polynomial Approximation	1-2
1.3	Determination of coefficients	1-4
1.4	1/2-point calibration	1-5
2	PICCAP Linearize Firmware Version 03.02.xx	2-6
2.1	Functionality	2-6
2.2	Implemented Functions	2-7
2.3	Result Registers	2-10
2.4	EEPROM	2-10
2.5	Parameter Registers	2-13
3	Linearization by Means of Evaluation Software	3-16
3.1	Sample Project (purposes of illustration)	3-16
3.2	Sensor Characterization	3-16
3.3	Temperature Sensor Characterization	3-17
3.4	One/Two Point Calibration	3-19
3.5	Expert	3-20
3.6	Load & Save	3-20
3.7	Write to EEPROM	3-21
4	DLL Functions	4-1
4.1	Capacitance	4-1
4.2	Temperature	4-5
5	Linearize Example Source Code	5-7
5.1	Example C-Code	5-7
5.2	Output via Terminal:	5-10
7	Miscellaneous	7-11

1 Introduction

Most types of capacitive sensors show a non-linear behavior. This means that the physical unit Z and the sensor's capacitance is not simply linearly proportional. Furthermore, the relation will include a temperature dependent term. The physical unit itself may be pressure, humidity, position or anything else.

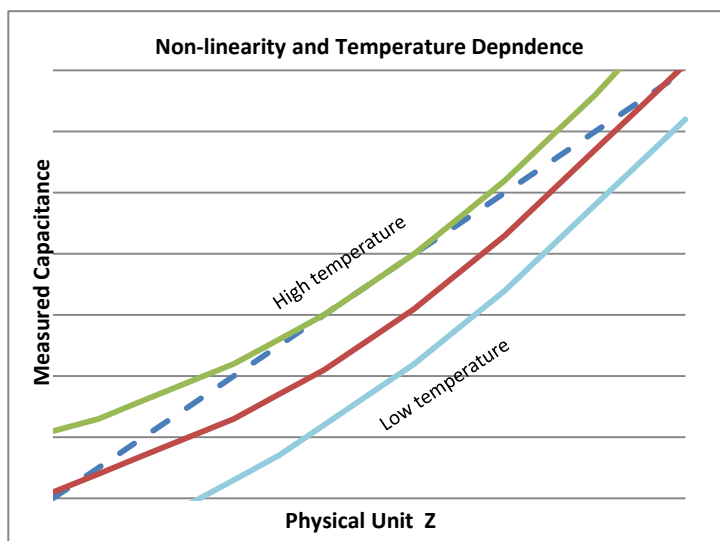
This datasheet describes the linearization firmware PCap02_Linearize, version 03.02.xx. This firmware is provided by acam for free and can be used to linearize sensors and to compensate them over temperature inside on the chip. Running with this firmware, the PCap02 provides not only the basic capacitance (sensor) and resistance ratios (temperature). The 48-bit DSP takes the resistance ratio to calculate the temperature and, based on this, takes the capacitance ratio to do all the further calculations. The final results Z for the sensor and the temperature ϑ are provided in read registers 0 and 1.

The linearization coefficients can be determined individually and stored in the EEPROM. For a simplified process the firmware offers the option of a simple 2-point calibration where the same linearization coefficients are used for a complete batch of sensors. Individual sensors are then calibrated only at two points.

1.1 Non-Linearity and Temperature Dependence

The characteristics of the non-linearity as well as the behavior over temperature are defined by the mechanical, electrical and chemical properties of the sensor itself.

Figure 1-1 Non-Linearity and Temperature Influence



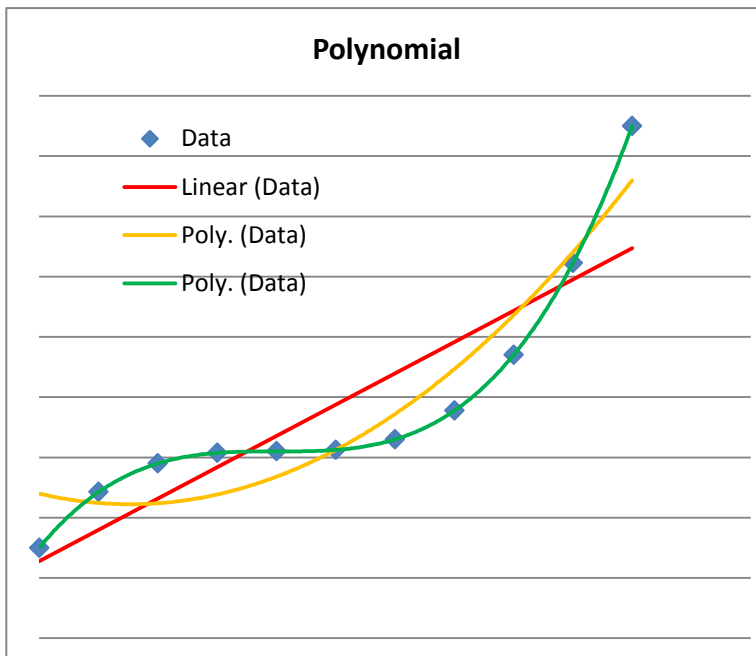
So in any case it is necessary to characterize a sensor by collecting data at different temperatures for various reference values of the unit of interest. On the basis of those data, a processor can correct the measured value by means of a correction table or a complex mathematical calculation. In many cases mathematical operations for linearization are very effective and

provide higher precision than correction tables.

1.2 Polynomial Approximation

An elegant way to approximate a non-linear function is the polynomial approach. The higher the order of the polynomial the better will be the approach. Of course, the mathematical effort will also increase with that. The following graph illustrates the “best-fit” – curves ranging from a straight line to a 3rd order polynomial:

Figure 1-2 Polynomial Approximation



The blue dots indicate a non-linear response of a sensor. The red line shows the linear approximation, the yellow line an approximation by a polynomial of 2nd order and the green line an approximation by a polynomial of 3rd order. Obviously the quality of approximation gets better the higher the order of the polynomial – especially if the non-linear curve bends several times and/or has a turning point.

For the PCapØ2 linearization firmware we decided to implement a 3rd-order polynomial approach for the linearization of the capacitance as well as for the resistance-to-temperature conversion.

$$Z = k_3 C^3 + k_2 C^2 + k_1 C + k_0 \quad (1)$$

$$\vartheta = tc_3 R^3 + tc_2 R^2 + tc_1 R + tc_0 \quad (2)$$

k_x	Coefficients of the capacitance polynomial
tc_x	Coefficients of the temperature polynomial
C	Capacitance ratio
R	Resistance ratio
Z	Output quantity

Additionally, the temperature information is used to correct the capacitance information. This correction is done by replacing the linearization coefficients by polynomials of second degree with temperature.

$$k_3 = cc_{32}\vartheta^2 + cc_{31}\vartheta + cc_{30} \quad (3a)$$

$$k_2 = cc_{22}\vartheta^2 + cc_{21}\vartheta + cc_{20} \quad (3b)$$

$$k_1 = cc_{12}\vartheta^2 + cc_{11}\vartheta + cc_{10} \quad (3c)$$

$$k_0 = cc_{02}\vartheta^2 + cc_{01}\vartheta + cc_{00} \quad (3d)$$

k_x	Coefficients of linearization polynomial
cc_{yy}	Coefficients of temperature compensated polynomial
ϑ	Temperature

The 12 coefficients cc_x fully describe the characteristics of the sensor. The key point is to determine the coefficients cc_{yy} accurately to describe the non-linear characteristic of the sensor best possible. The choice of the right calibrations points is therefore important.

Note: In the firmware the function is expressed as a function of the temperature and capacitance. Replacing k_x in (1) with the substitution from (3a), (3b), (3c) and (3d) and regrouping the formula gives:

$$Z = a_2\vartheta^2 + a_1\vartheta + a_0 \quad (4)$$

1.3 Determination of coefficients

The cc_x coefficients for linearization are determined by means of least squares method. A set of measurement data needs to be collected to characterize the sensor. The physical parameter should be measured at minimum four values and three temperatures, in total minimum 12 points, to have enough data for a polynomial of third degree. Having more points will give better approximation. Critical points might be weighted by adding them twice.

Example:

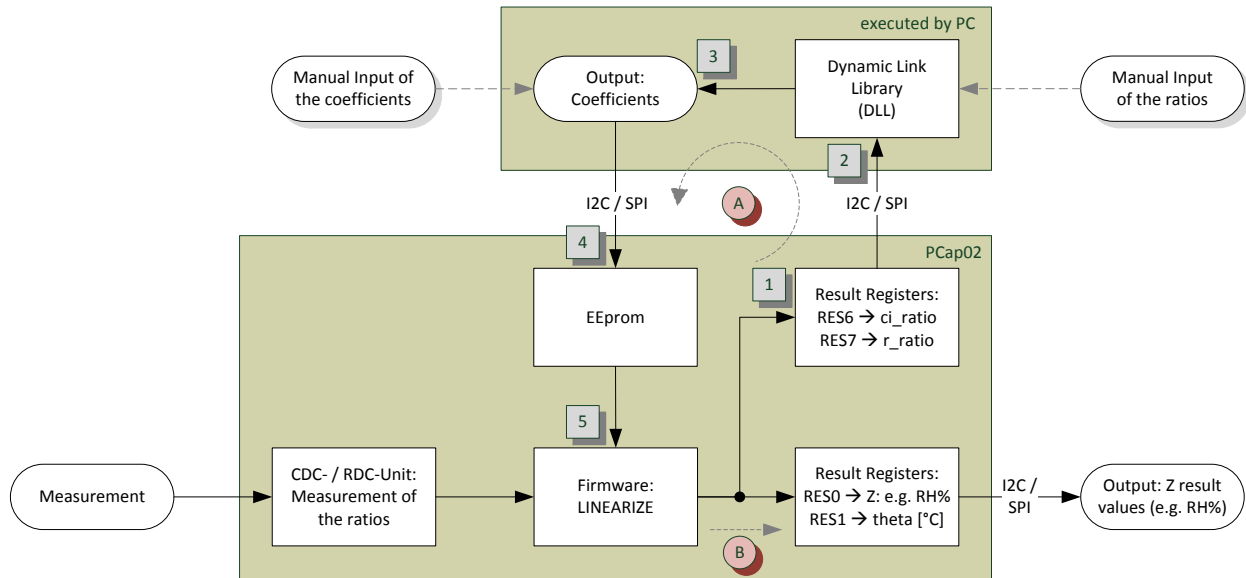
θ	Z	Cr/Cs
20°C	10%	0,85123
20°C	20%	0,86443
20°C	30%	0,87743
40°C	10%	0,8411
...

Having such a data set, it is possible to determine the linearization coefficients by means of e.g. the "least squares" method, LINEST or RGP function in Excel. acam provides a DLL to use this function in any kind of software. The DLL is fed with the collected data and gives back the coefficients as output. Further details are described in chapter 4.

Figure 1-3 shows again the major action items during a calibration run:

- A.1 Collect raw data of capacitance ratio (ci_ratio) at various measure points by means of the linearization firmware.
- A.2 Transfer these data to the DLL.
- A.3 In the DLL the coefficients will be calculated.
- A.4 Get back the coefficients cc_x and write the coefficients into the EEPROM.
- B Make again measurements at various points and read back the Z output for verification.

Figure 1-3 Sensor Characterization



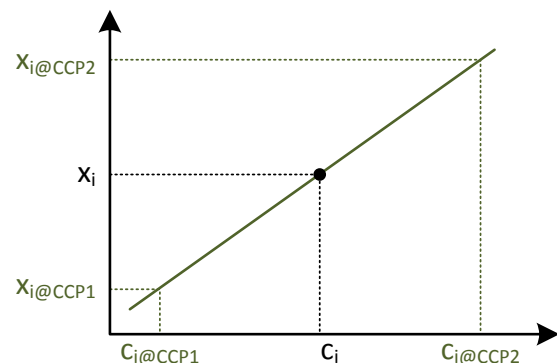
This procedure can be used for a full calibration of each single transducer and will offer the best precision.

1.4 1/2-point calibration

In many applications a full calibration of every single sensor will be too expensive. In case the sensors show more or less the same characteristics over a production lot, there is a chance that 2-point calibration or even 1-point calibration is sufficient to achieve a good level of precision. In such case, the full measurement data set is collected only for a small number of samples. The coefficients from this sample lot are then used for all other sensors of the lot. The individual sensor itself is calibrated only at two points, ideally taken at two different temperatures. From those two calibration points the offset and slope are calculated and used to correct the initial capacitance ratio. For convenience, the firmware is programmed in a way that the user enters the theoretical ratios at the two calibration points and then the really measured ratios.

The 2-point calibrated intermediate result x_i is calculated as:

$$x_i = \frac{x_{i@CCP2} - x_{i@CCP1}}{c_{i@CCP2} - c_{i@CCP1}} \cdot (c_i - c_{i@CCP1}) + x_{i@CCP1}$$



$x_{i@CCP}$ Theoretical capacitance ratios at calibration points

$c_{i@CCP}$ Measured capacitance ratios at calibration points

c_i Actual capacitance ratio

x_i 2-point corrected capacitance ratio

This intermediate x_i result is then fed into the linearization polynomial for calculation of the final Z result.

In case of a 1-point calibration the coefficients $x_{i@ccp1}$ and $c_{i@ccp1}$ simply need to be set to 0.

2 PICOCAP Linearize Firmware Version 03.02.xx

2.1 Functionality

The Firmware is capable of performing these tasks:

- Capacitance
 - Output of original inverted capacitance ratio c_{i_ratio} (c_{ref}/c_{sense})
 - Output of x_i : 2-point corrected capacitance ratio
 - Output of Z: linearized and temperature corrected result
 - Selectable sensor port (C1..C7, 1x 4 bit to select in PARA8)
 - Single result, no support for combo sensors at the moment
 - Polynomial of 3rd order for the capacitance linearization
 - Coefficients are temperature corrected by a polynomial of 2nd order
 - => Total 12 coefficients
 - 2-point calibration => another 4 calibration values
 - Programmable limits for minimum/maximum of Z
- Temperature
 - Output of original inverted resistance ratio r_{i_ratio} (r_{ref}/r_{sense})
 - Output of 2-point corrected resistance ratio y_i
 - Output of final temperature ϑ in °C
 - Selectable temperature sensor input (R0..R2 in PARA8)
 - Single result
 - Linearization by polynomial 3rd order
 - => 4 coefficients + 4 calibration values from 2-point calibration
 - Programmable limits for minimum/maximum of ϑ
- Alarm Outputs

- 2 Alarm Outputs
- Selectable alarm source (Z-result/theta, 1 bit in PARA8)
- On/off threshold each
- Selectable polarity
- PDM
 - Pulse0 := capacitance is fixed
 - Pulse1 := temperature is fixed
 - Each output is scalable via "scale" and "offset"
 - Limits can be set in LSB
- Filter
 - Selectable median 5 filter for capacitance
 - Selectable median 5 filter for temperature
 - Both to be activated in PARA8

2.2 Implemented Functions

2.2.1 Variables and Coefficients

$c_i = \frac{c_{ref}}{c_{sense}}$	Inverse capacitance ratio	x_i	Inverse 2-point corrected capacitance ratio
$r_i = \frac{r_{ref}}{r_{measure}}$	Inverse resistance ratio	y_i	Inverse 2-point corrected resistance ratio
Z	Linearized and temperature compensated final result	ϑ	Linearized temperature result
$cc_{k,l}$	Coefficients for the capacitance polynomial, non-inverse	$cci_{k,l}$	Coefficients for the inverse capacitance polynomial
tc_k	Coefficients for the temperature polynomial, non-inverse	tci_k	Coefficients for the inverse temperature polynomial
$x_i@CCPn$	Expected values for capacitance ratios with 2-point calibration	$y_i@TCPn$	Expected values for resistance ratios with 2-point calibration

2.2.2 Temperature Linearization

$$y_i = \frac{y_{i@TCP2} - y_{i@TCP1}}{r_{i@TCP2} - r_{i@TCP1}} \cdot (r_i - r_{i@TCP1}) + y_{i@TCP1}$$

Inverse linearized resistance ratio
(2-Point Calibration)

$$y_i = TQ(\vartheta) = \sum_{k=0}^3 tci_k \cdot \vartheta^k$$

Inverse Temperature Polynomial

$$\vartheta = TP(y) = TQ^{-1} = \sum_{k=0}^3 tc_k \cdot \frac{1}{y_i^k}$$

Temperature Polynomial

2.2.3 Sensor Linearization

$$x_i = \frac{x_{i@CCP2} - x_{i@CCP1}}{c_{i@CCP2} - c_{i@CCP1}} \cdot (c_i - c_{i@CCP1}) + x_{i@CCP1}$$

Inverse linearized capacitance ratio
(2-Point Calibration)

$$x_i = CQ(z, \vartheta) = \sum_{k=0}^3 \sum_{l=0}^2 cci_{k,l} \cdot z^k \cdot \vartheta^l$$

Inverse Capacitance Polynomial

$$Z = CP(x_i, \vartheta) = CQ^{-1} = \sum_{k=0}^3 \sum_{l=0}^2 cc_{k,l} \cdot \frac{1}{x_i^k} \cdot \vartheta^l$$

Capacitance Polynomial

2.2.4 Calculation Values

The firmware takes care of the wide range of possible parameters. It therefore controls the number of division steps and also makes the correct shift operations to achieve the full resolution with the necessary number of relevant bits during all calculations. Those data, too, are all calculated automatically by the DLL.

cn_div32..cn_div00 Number of division steps
for capacitance linearization

cn_shift2..cn_shift0 Number of shift operations
for capacitance linearization

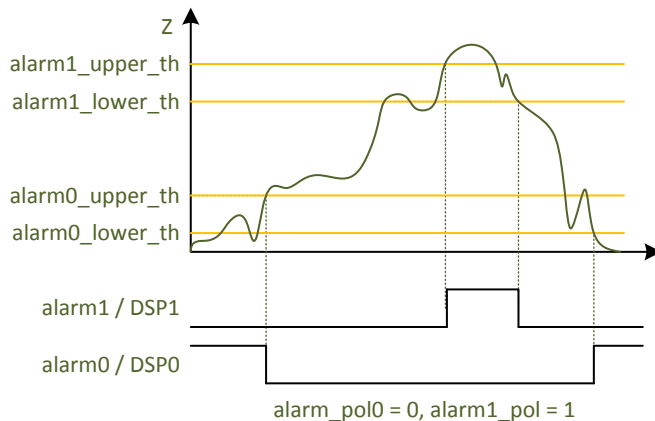
tn_div2..tn_div0 Number of division steps
for temperature calculation

2.2.5 Alarm Levels

The firmware provides two alarm outputs:

- Alarm0 sets output DSP0 which can be passed to general purpose I/Os PGO or PG2.
- Alarm1 sets output DSP1 which can be passed to general purpose I/Os PG1 or PG3.

Figure 2-1



The source can be selected for both alarm outputs independently between Z and ϑ (Parameter 8: alarmx_sel). Also the polarity of the alarm outputs can be selected independently (Parameter 8: alarmx_pol). For each alarm output an upper and lower threshold can be configured. The difference defines the hysteresis.

2.2.6 Pulse Outputs

The use of the two pulse density modulated outputs is fixed.

PULSE0: Z PULSE1: ϑ

Slope and offset are set independently in parameter registers 3 to 6. The limits for the PULSE outputs are defined by Pulse_Z_min, Pulse_Z_max, Pulse_theta_min and Pulse_theta_max (EEPROM values 32 to 35).

The slope is set by a fractional number. In case the Z result has 15 fractional digits, the slope needs to have 5 fractional digits. In general the sum of fractional digits of Z and slope has to be 20. The slope itself depends on the configuration of the PULSE output resolution. According to that the maximum value of PULSEx can be 1023, 4095, 16383 or 65535.

The offset is set with 1 fractional bit, although the final pulse output needs to be an integer. This allows a correct mathematical rounding in case the last bit is 1.

For details please see datasheet PCap02 Vol. 1 section 4.6.2.

2.3 Result Registers

RES#	name	fpp*	description
0	Z:	15	Z-result, final result, signed 24 bit, thereof 15 bit fractional
1	theta [°C]	8	Temperature, signed 24 bit, thereof 8 bit fractional
2	reserved		
3	reserved		
4	reserved		
5	reserved		
6	ci_ratio	22	(Median of) selected capacitance ratio, 24 bit, thereof 22 bit fractional
7	ri_ratio	22	(Median of) selected resistance ratio, 24 bit, thereof 22 bit fractional
8	xi_ratio	22	2-point corrected ci_ratio, , 24 bit, thereof 22 bit fractional
9	yi_ratio	22	2-point corrected ri_ratio, , 24 bit, thereof 22 bit fractional
10	Pulse_Z:	0	Output value for pulse interface PDM0, Z
11	Pulse_theta	0	Signed output value for pulse interface PDM1, 9

*fpp = Fixed point position

2.4 EEPROM

2.4.1 Calibration Values

#	Name	Type	fpp	EEPROM Addr. [23:16].[15:8].[7:0]	Description
0	ci_at_ccp1	24u	22	0, 1, 2	Measured value of capacitance ratio at capacitance calibration point 1
1	ci_at_ccp2	24u	22	3, 4, 5	Measured value of capacitance ratio at capacitance calibration point 2
2	xi_at_ccp1	24u	22	6, 7, 8	Nominal value for capacitance ratio at capacitance calibration point 1
3	xi_at_ccp2	24u	22	9, 10, 11	Nominal value for capacitance ratio at capacitance calibration

#	Name	Type	fpp	EEPROM Addr. [23;16],[15;8],[7;0]	Description
					point 2
4	Z_min	24s	var* (typ 15)	12, 13, 14	Minimum output value for Z, e.g. 0 for output in %
5	Z_max	24s	var* (typ 15)	15, 16, 17	Maximum output value for Z, e.g. 100 for output in %
6	ri_at_tcp1	24u	22	18, 19, 20	Measured resistance ratio at temperature calibration point 1
7	ri_at_tcp2	24u	22	21, 22, 23	Measured resistance ratio at temperature calibration point 2
8	yi_at_tcp1	24u	22	24, 25, 26	Nominal value of resistance ratio at temperature calibration point 1
9	yi_at_tcp2	24u	22	27, 28, 29	Nominal value of resistance ratio at temperature calibration point 2
10	theta_min	24s	8	30, 31, 32	Minimum output value for ϑ , e.g. -40 for output in °C
11	theta_max	24s	8	33, 34, 35	Maximum output value for ϑ , e.g. -125 for output in °C
12	cc32	24s	var*	36, 37, 38	Coefficients for the capacitance polynomial. In case no temperature compensation is necessary use only cc30, cc20, cc10 and cc00 and set all other coefficients to "0"
13	cc22	24s	var*	39, 40, 41	
14	cc12	24s	var*	42, 43, 44	
15	cc02	24s	var*	45, 46, 47	
16	cc31	24s	var*	48, 49, 50	
17	cc21	24s	var*	51, 52, 53	
18	cc11	24s	var*	54, 55, 56	
19	cc01	24s	var*	57, 58, 59	
20	cc30	24s	var*	60, 61, 62	
21	cc20	24s	var*	63, 64, 65	
22	cc10	24s	var*	66, 67, 68	
23	cc00	24s	var*	69, 70, 71	
24	tc3	24s	var*	72, 73, 74	Coefficients for the temperature polynomial.
25	tc2	24s	var*	75, 76, 77	
26	tc1	24s	var*	78, 79, 80	

#	Name	Type	fpp	EEPROM Addr. [23;16],[15;8],[7;0]	Description
27	tc0	24s	var*	81, 82, 83	
28	alarm0_upper_th	24s	var*	84, 85, 86	Upper threshold for alarm0 source selectable by PARA8 fpp same as alarm source
29	alarm0_lower_th	24s	var*	87, 88, 89	Lower threshold for alarm0 source selectable by PARA8 fpp same as alarm source
30	alarm1_upper_th	24s	var*	90, 91, 92	Upper threshold for alarm1 source selectable by PARA8 fpp same as alarm source
31	alarm1_lower_th	24s	var*	93, 94, 95	Lower threshold for alarm1 source selectable by PARA8 fpp same as alarm source
32	Pulse_Z_min	24s	0	96, 97, 98	Lower limit for Z pulse_out PDM0 (in LSB, Normally 0)
33	Pulse_Z_max	24s	0	99, 100, 101	Upper limit for Z pulse_out PDM0 (in LSB, normally resolution of pulse)
34	Pulse_theta_min	24s	0	102, 103, 104	Lower limit for theta pulse_out PDM1 (in LSB, Normally 0)
35	Pulse_theta_max	24s	0	105, 106, 107	Upper limit for theta pulse_out PDM1 (in LSB, normally resolution of pulse)

*var := variable value for fpp

2.4.2 Calculation Values

#	Name	Type	EEPROM Address	Description
0	cn_div32	8u	108	Division steps for the cc32 term
1	cn_div22	8u	109	Division steps for the cc22 term
2	cn_div12	8u	110	Division steps for the cc12 term
3	cn_div31	8u	111	Division steps for the cc31 term
4	cn_div21	8u	112	Division steps for the cc21 term
5	cn_div11	8u	113	Division steps for the cc11 term
6	cn_div30	8u	114	Division steps for the cc30 term
7	cn_div20	8u	115	Division steps for the cc20 term
8	cn_div10	8u	116	Division steps for the cc10 term
9	cn_shift2	8s	117	Shift operations capacitance

#	Name	Type	EEPROM Address	Description
10	cn_shift1	8s	118	Shift operations capacitance
11	cn_shift0	8s	119	Shift operations capacitance
12	tn_div2	8u	120	Division steps for temperature
13	tn_div1	8u	121	Division steps for temperature
14	tn_div0	8u	122	Division steps for temperature
15			123 TO 127	Free EEPROM space. Could be used for e.g. serial numbers.

2.5 Parameter Registers

#	Name	Type	Name	Description
0			PARA0	Not used
1			PARA1	Not used
2			PARA2	Not used
3	pulse_slope_Z	24s	PARA3	Setting slope for the PDM0 output (Z): $fpp_Z_result + fpp_z_slope = 20$ $fpp_z_slope = 20 - fpp_z_result$
4	pulse_offset_Z	24s	PARA4	Setting offset for the PDM0 output (Z): $fpp\ 1$
5	pulse_theta_slope	24s	PARA5	Setting slope for the PDM1 output (9): $fpp_theta + fpp_theta_slope = 20$ bzw. $fpp_theta_slope = 20 - fpp_theta$
6	pulse_theta_offset	24s	PARA6	Setting offset for the PDM1 output (v): $fpp\ 1$
7	Gain_Corr	24s	PARA7	Correction factor for the gain drift. $fpp\ 21$
8	exFlags		PARA8	See below

2.5.1 Flags in PARA8

#	Name	Type		Description
0	CDC_TRIG_BG		PARA8<0>	The CDC end triggers the bandgap refresh Setting this = 1 is recommended for

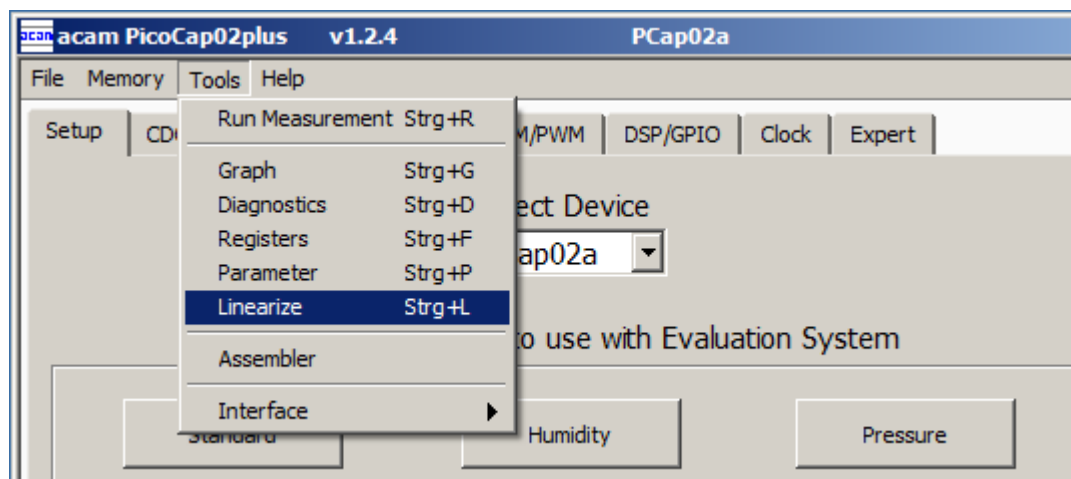
#	Name	Type		Description
				typical applications
1	DSP_TRIG_CDC		PARA8<1>	End of DSP-calculations triggers the next CDC (for quasi continuous mode, C_TRIG_SEL=2, CONV_TIME=0, only with firmware version 3 and later)
2	INTN_TRIG_BG		PARA8<2>	End of serial interface (SIF) read command triggers the bandgap refresh
3	INTN_TRIG_CDC		PARA8<3>	End of serial interface (SIF) read command triggers the next CDC (only reasonable with C_TRIG_SEL = 2 CONV_TIME = 0 INTN_TRIG_EN = 1, only with firmware version 3 and later)
4	Z_median_en		PARA8<4>	Enable median 5 filter for Capacitance & Z-result
5	theta_median_en		PARA8<5>	Enable median 3 filter for temperature
6	alarm0_select		PARA8<6>	alarm0 source: (DSP_OUT_0) 0 := Z-result 1 := temperature
7	alarm0_pol		PARA8<7>	0 := low_active (alarm_source > upper_threshold output = low; alarm_source < lower_threshold => output = high) 1 := high_active (alarm_source > upper_threshold output = high, alarm_source < lower_threshold output = low)
8	alarm1_select		PARA8<8>	alarm1 source (DSP_OUT_1) 0 := Z-result 1 := temperature
9	alarm1_pol		PARA8<9>	see alarm0_pol
10 11 12	c_sel0 c_sel1 c_sel2	3u	PARA8<12..10>	select Capacitance for polynomial determination: 0 := C1/CO 1 := C2/CO

#	Name	Type		Description
				2 := C3/CO 3 := C4/CO 4 := C5/CO 5 := C6/CO 6 := C7/CO
13 14	r_sel0 r_sel1	2u	PARA8<14..13>	select resistor for temperature polynomial: 0 := R0/Rref 1 := R1/Rref 2 := R2/Rref

3 Linearization by Means of Evaluation Software

Starting with evaluation software PCap02plus version V1.2.4 the linearization feature is implemented. Under menu item "Tools" there is a selection "Linearize" which opens a separate Window.

Figure 3-1 Linerize Menu



This new windows includes all elements to collect data for the sensor characterization, to calculate the coefficients for linearization, and to perform a 2-point calibration. All calculations are based on the linearize_r01.dll library file.

3.1 Sample Project (purposes of illustration)

Preparing:

non-linearized sensor, which is to be linearized

Definition of the linearized measuring range 0 to 8 inH2O, with 2 inH2O steps

Definition of the temperature range 20 to 50 °C, with 10 °C steps

3.2 Sensor Characterization

The first step is the characterization of the sensor. Therefore, it is necessary to collect data at several measurement points and at several temperatures.

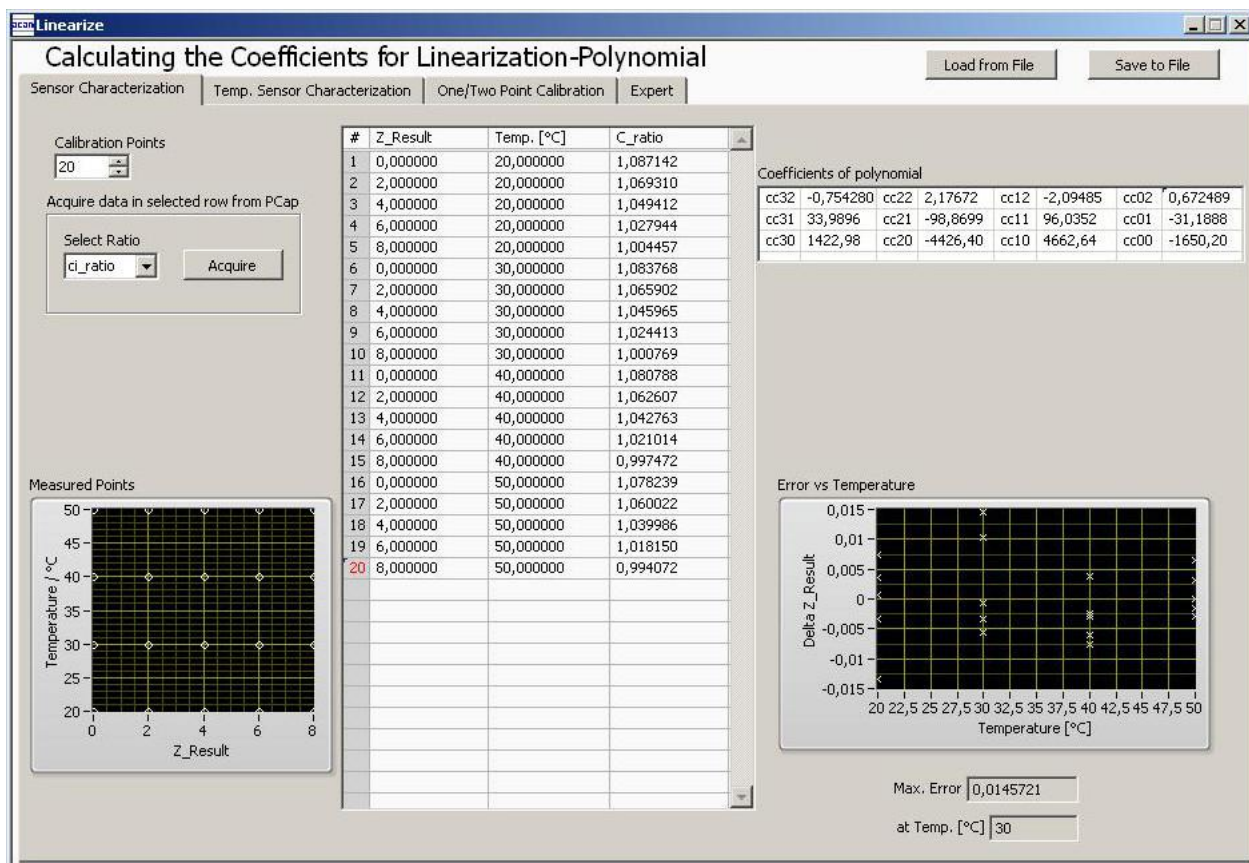
As mentioned earlier, the data collection should be made of minimum 12 measurements, taken at least at 3 different temperatures. The temperatures should cover the operating temperature range of interest of the final device. The number of calibration points is set at

the top left. This is the first thing to be done. Then calibration can begin. Line by line the user can enter the reference values for Z and ϑ at the various calibration points. Having the cursor in this line it is sufficient to press the acquire button to get the actual ci_ratio result. But of course the value can be entered manually, too.

The graph on the bottom left shows the Z, ϑ distribution of the calibration points. Ideally it should have dots on three different lines covering the operating range of the sensor.

The table on the left shows the calculated calibration coefficients and the graph below shows the deviation due to the mathematical approximation.

Figure 3-2 Tab Sensor Characterization



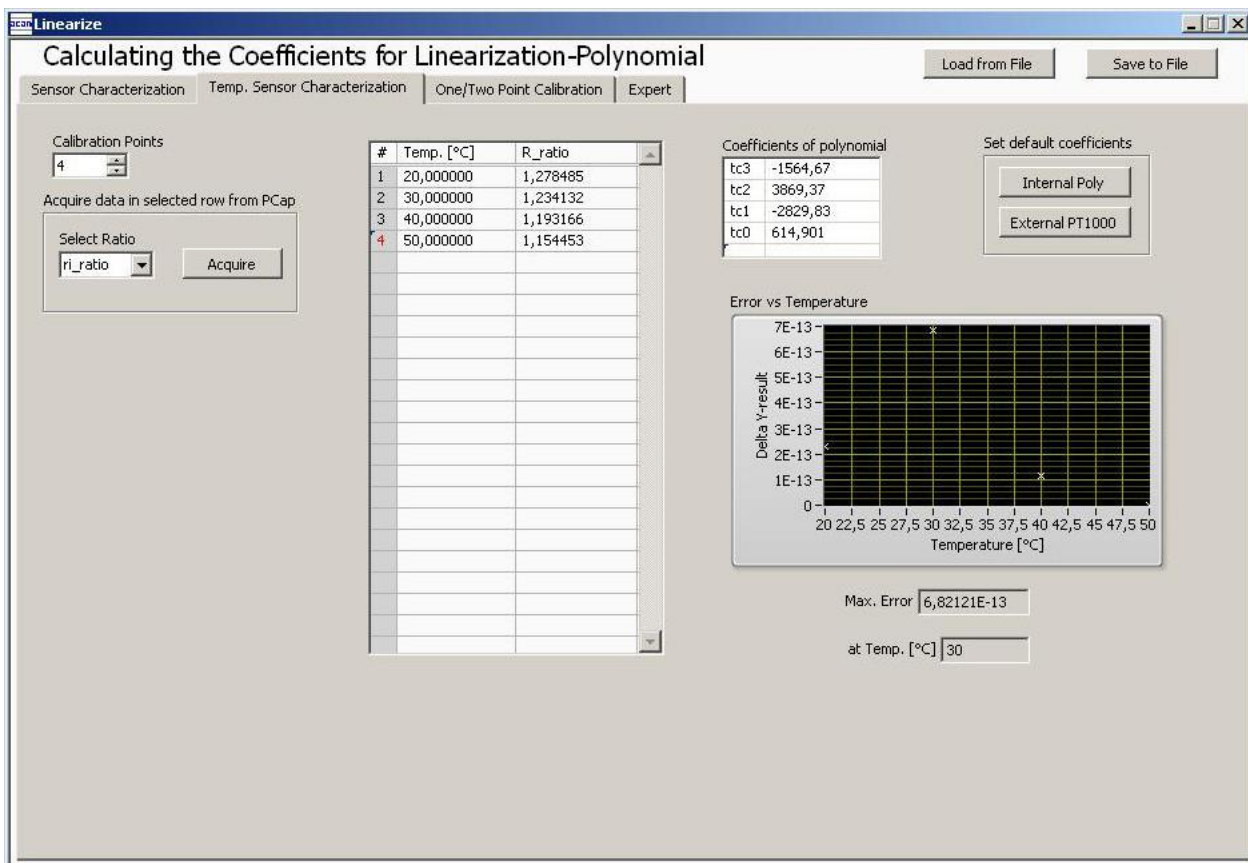
3.3 Temperature Sensor Characterization

Together with the calibration of the capacitance sensor it is mandatory to calibrate the temperature, too. Whether the internal aluminum sensor is used or an external platinum

sensor or any other sensor: they need to be calibrated to get the correct temperature information which is then used as input for the polynomial correction of the capacitance measurement.

The tab „Temperature Sensor Characterization” (Figure 3-3) offers a tool very similar to the capacitive sensor characterization. The resistance ratio has to be collected at several temperature points. For best approximation 4 calibration points are needed. In case of 2 or 3 calibration points a 2nd respectively a 3rd order polynomial is calculated.

Figure 3-3 Temperature Sensor Characterization



On the right side of the tab “Temperature Sensor Characterization” there are two buttons to select default characteristic data for the internal aluminum sensor and a platinum sensor. The aluminum is assumed to be linear in a range of 10 °C to 70°C so only two coefficients are used.

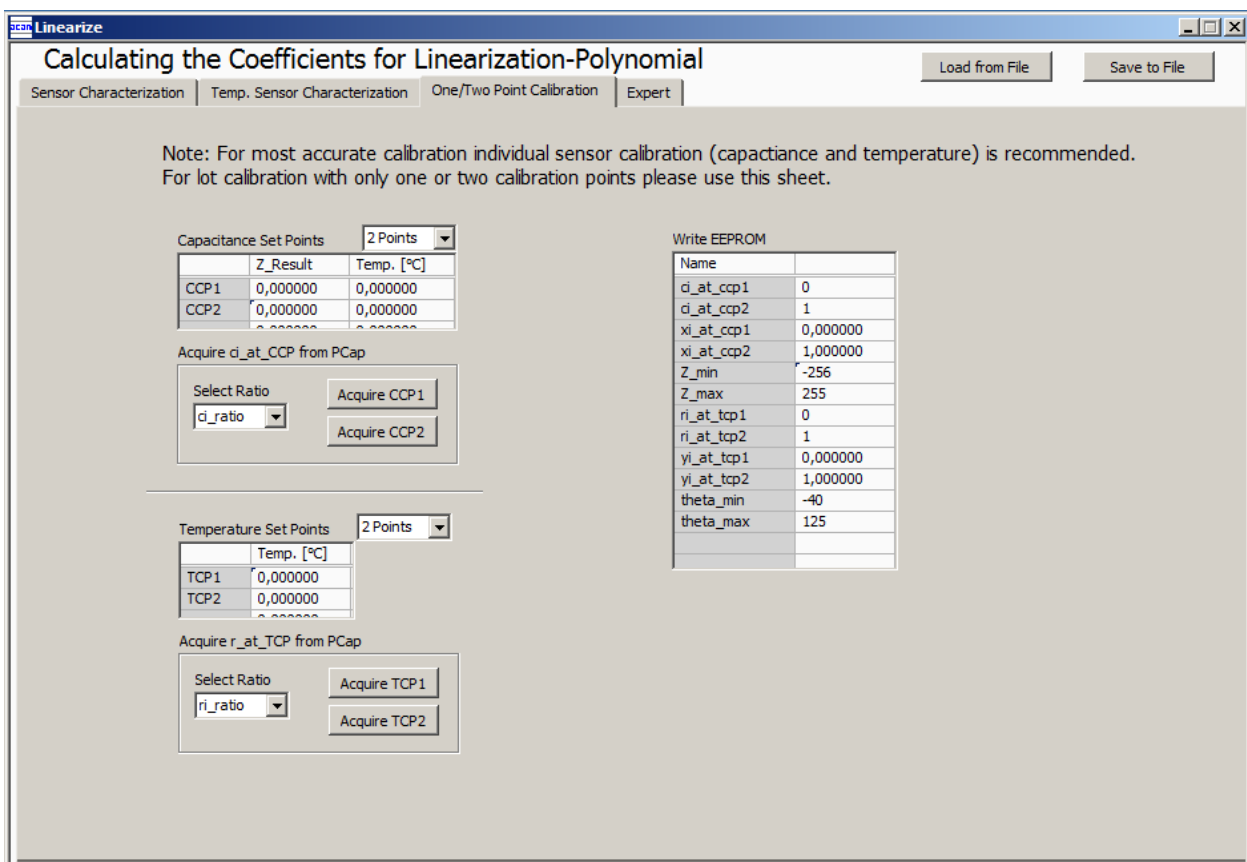
In case the default values are used it is necessary to have at least a two point calibration of the temperature (see next section).

3.4 One/Two Point Calibration

Once a batch is characterized with respect to the capacitive sensor and the resistive temperature sensor it might be sufficient to perform two-point or even one-point calibration for the rest of the sensors in the batch.

The tab “One/Two Point Calibration” offers a simple GUI to do that. On this page the user enters the reference values for Z and θ . CCP1 stands for capacitance calibration point 1 etc.. When the calibration conditions are reached pressing the acquire buttons will read the actual ratios while the theoretical ones are calculated on basis of the linearization coefficients. Together with programmable limits for minimum and maximum this gives an additional set of 12 parameters to be written into the EEPROM.

Figure 3-4 One/Two Point Calibration



Linearize

Calculating the Coefficients for Linearization-Polynomial

Sensor Characterization | Temp. Sensor Characterization | **One/Two Point Calibration** | Expert

Load from File | Save to File

Note: For most accurate calibration individual sensor calibration (capactiance and temperature) is recommended. For lot calibration with only one or two calibration points please use this sheet.

Capacitance Set Points: 2 Points

	Z_Result	Temp. [°C]
CCP1	0,000000	0,000000
CCP2	0,000000	0,000000

Acquire ci_at_CCP from PCap

Select Ratio: ci_ratio

Acquire CCP1 | Acquire CCP2

Temperature Set Points: 2 Points

	Temp. [°C]
TCP1	0,000000
TCP2	0,000000

Acquire r_at_TCP from PCap

Select Ratio: ri_ratio

Acquire TCP1 | Acquire TCP2

Write EEPROM

Name	
ci_at_ccp1	0
ci_at_ccp2	1
xi_at_ccp1	0,000000
xi_at_ccp2	1,000000
Z_min	-256
Z_max	255
ri_at_tcp1	0
ri_at_tcp2	1
yi_at_tcp1	0,000000
yi_at_tcp2	1,000000
theta_min	-40
theta_max	125

3.5 Expert

As indicated by the name this tab is for experts only. It displays the numbers of division steps respectively shift operation to achieve the maximum resolution over all calculations.

Those are stored in the EEPROM, too. But they are calculated by the DLL and for information purpose only.

3.6 Load & Save

The linearization data as well as the One/Two Point Calibration can be exported into an external file with extension .dat. Additionally those data can be loaded from external files into the software.

The linearization file includes the calibration data, the polynomial coefficients and the multiplication steps/shifts. In following, such a file is shown as an example:

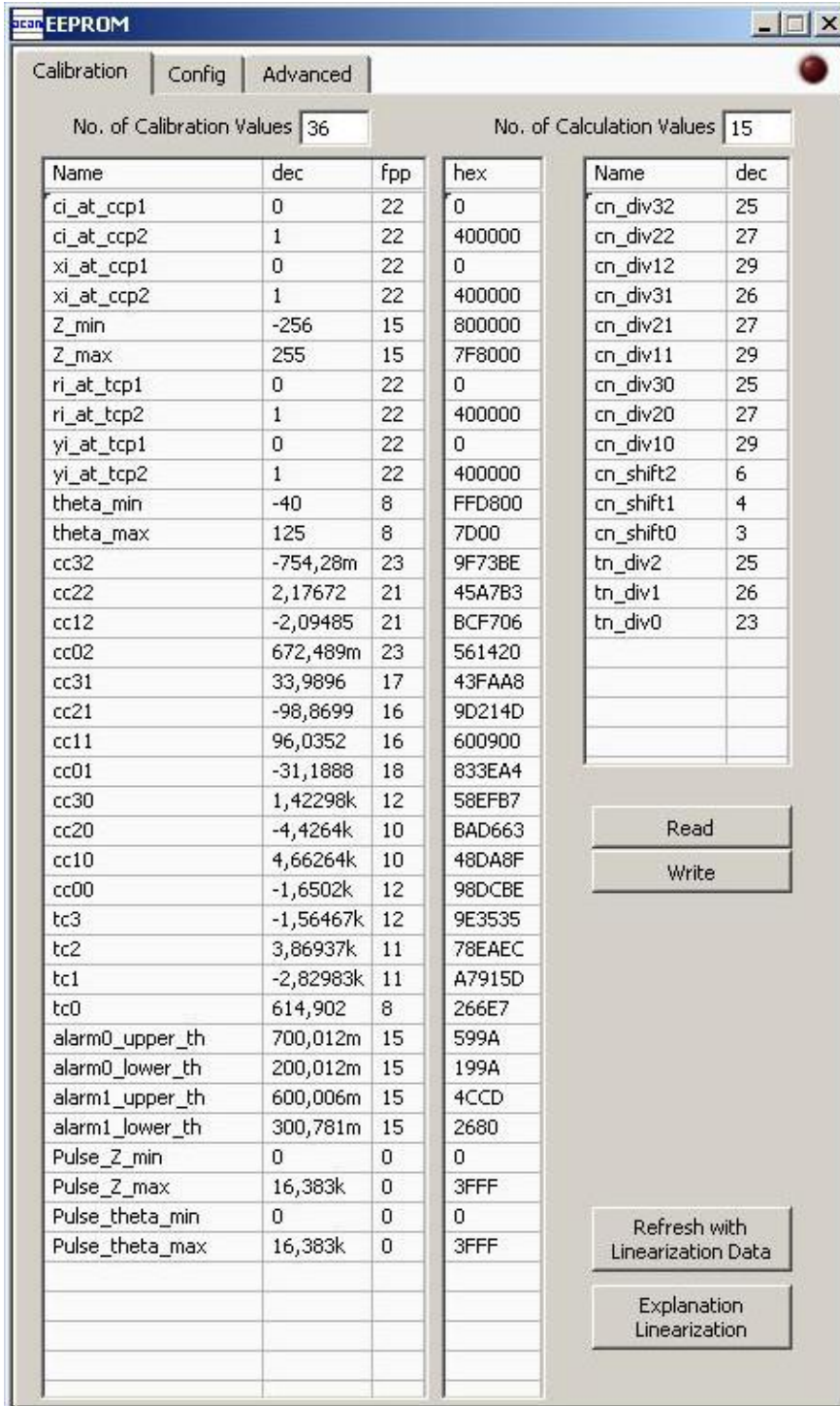
% Linearization Data File: LinData_Export.dat Saved on 29.08.2013 14:49

```
% c_lin_coeff_dut: C_in, theta, Z_Result
0.990373    0.968582    0.949290    0.931916    0.916191    0.901869    0.888779
    0.876746    0.865660    0.855405    0.845899    0.990578    0.968738
    0.949436    0.932071    0.916365    0.902076    0.889012    0.877027
    0.865987    0.855779    0.846315    0.990343    0.968419    0.948988
    0.931478    0.915617    0.901155    0.887920    0.875752    0.864536
    0.854150    0.844512
25.0000 25.0000 25.0000 25.0000 25.0000 25.0000 25.0000 25.0000 25.0000 25.0000 80.0000 80.0000
80.0000 80.0000 80.0000 80.0000 80.0000 80.0000 80.0000 80.0000 80.0000 80.0000 -10.0000 -10.0000
-10.0000 -10.0000 -10.0000 -10.0000 -10.0000 -10.0000 -
10.0000 -10.0000 -10.0000
0.00000 0.100000 0.200000 0.300000 0.400000 0.500000 0.600000
0.700000 0.800000 0.900000 1.00000 0.00000 0.100000 0.200000
0.300000 0.400000 0.500000 0.600000 0.700000 0.800000
0.900000 1.00000 0.00000 0.100000 0.200000 0.300000 0.400000
0.500000 0.600000 0.700000 0.800000 0.900000 1.00000
% t_lin_coeff_dut: R_in, theta
1.25859 1.21503
10.0000 20.0000
% c_2p_nominal: Z_Result, theta t_2p_nominal: theta
0.200000 0.800000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000
25.0000 80.0000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
25.0000 60.0000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
% c_fpp, c_fpp_int, theta_fpp, z_fpp, r_fpp, r_int_fpp
22 26 8 15 22 25
% General Settings
0 7 6 7
% cc: coefficients for the capacitance polynomial
-12,6380 35,1708 -36,7076 14,1334 -0,0384877 0,116354 -0,118114
0,0402407 0,000297351 -0,000886266 0,000890576 -0,000301428
% tc: coefficients for the temperature polynomial
-268,932 351,061 0,00000 0,00000
```


3.7 Write to EEPROM

In the last step, the linearization coefficients need to be transferred to the EEPROM.

Therefore, open the EEPROM window, press “Refresh with Linearization Data” and then press “Write”.



The screenshot shows the EEPROM window with three tabs: Calibration, Config, and Advanced. The Calibration tab is active. It displays two tables: one for calibration values (36 values) and one for calculation values (15 values). The calibration table has columns for Name, dec, fpp, and hex. The calculation table has columns for Name and dec. Below the tables are buttons for Read, Write, Refresh with Linearization Data, and Explanation Linearization.

Name	dec	fpp	hex
ci_at_ccp1	0	22	0
ci_at_ccp2	1	22	400000
xi_at_ccp1	0	22	0
xi_at_ccp2	1	22	400000
Z_min	-256	15	800000
Z_max	255	15	7F8000
ri_at_tcp1	0	22	0
ri_at_tcp2	1	22	400000
yi_at_tcp1	0	22	0
yi_at_tcp2	1	22	400000
theta_min	-40	8	FFD800
theta_max	125	8	7D00
cc32	-754,28m	23	9F73BE
cc22	2,17672	21	45A7B3
cc12	-2,09485	21	BCF706
cc02	672,489m	23	561420
cc31	33,9896	17	43FAA8
cc21	-98,8699	16	9D214D
cc11	96,0352	16	600900
cc01	-31,1888	18	833EA4
cc30	1,42298k	12	58EFB7
cc20	-4,4264k	10	BAD663
cc10	4,66264k	10	48DA8F
cc00	-1,6502k	12	98DCBE
tc3	-1,56467k	12	9E3535
tc2	3,86937k	11	78EAEC
tc1	-2,82983k	11	A7915D
tc0	614,902	8	266E7
alarm0_upper_th	700,012m	15	599A
alarm0_lower_th	200,012m	15	199A
alarm1_upper_th	600,006m	15	4CCD
alarm1_lower_th	300,781m	15	2680
Pulse_Z_min	0	0	0
Pulse_Z_max	16,383k	0	3FFF
Pulse_theta_min	0	0	0
Pulse_theta_max	16,383k	0	3FFF

Name	dec
cn_div32	25
cn_div22	27
cn_div12	29
cn_div31	26
cn_div21	27
cn_div11	29
cn_div30	25
cn_div20	27
cn_div10	29
cn_shift2	6
cn_shift1	4
cn_shift0	3
tn_div2	25
tn_div1	26
tn_div0	23

Buttons: Read, Write, Refresh with Linearization Data, Explanation Linearization

4 DLL Functions

4.1 Capacitance

4.1.1 Coefficients for single devices (c_lin_coeff_dut)

Name: c_lin_coeff_dut

Function: Determines the polynomial coefficients which fit best with least square method for 3rd order to Capacitance and 2nd order to temperature. The coefficients were determined as double values and 3byte-Integer to write them to the EEPROM of PCap02.

Declaration: void __cdecl c_lin_coeff_dut(int32_t C_in[], double z[], double theta[], int32_t C_fpp, int32_t C_int_fpp, int32_t z_fpp, bool inverse, double cc[], uint32_t cc_fpp[], int32_t ccx[], double cci[], double error_vs_z[], uint32_t cn_div[], int32_t cn_shift[], int32_t n_samples, int32_t n_cc, int32_t n_cn_div, int32_t n_cn_shift);

Name	Type	Description
int32_t C_in[]	in	Array of capacitance ratios, data for all measured points
double z[]	in	Array of the result samples (reference values, e.g. Pressure, Humidity)
double theta[]	in	Array of reference temperature samples (reference values)
int32_t C_fpp	in	Fix point position for C_in, 8-bit number typ: C_fpp = 22 => 22 fractional bits (default: 22)
int32_t C_int_fpp	in	Fix point position of c ratio to determine division steps. 8-bit number (default: 26)
int32_t z_fpp	in	Fix point position for the result. 8-bit number (typically: 15)
bool inverse	in	Flag to sign C_r is inverse false := $C_{in} = C_r = \frac{C_{measure}}{C_{ref}}$ true := $C_{in} = C_i = \frac{C_{ref}}{C_{measure}}$ (default: true)
double cc[]	out	Array of 12 coefficients for the result polynomial
uint32_t cc_fpp []	out	Array of 12 integer values with the fix point positions to ccx

Name	Type	Description
int32_t ccx[]	Out → EEPROM	Array 12 (4*3) unsigned integer coefficients for DSP
double cci[]	out	Array of 12 coefficients for the inverse capacitance polynomial
double error_vs_z[]	out	Mathematical error vs z input
uint32_t cn_div[]	Out → EEPROM	Array of 9 integer values contains the number of division steps for the cc determination
int32_t cn_shift[]	Out → EEPROM	Array of 3 signed integer value for the shift steps after last sum. n_shift < 0 := right shift n_shift ≥ 0 := left shift
int32_t n_samples	in	Number of samples / length of the arrays C_in[], z[], theta[] and error_vs_z[]
int32_t n_cc	in	Length of array cc (strongly recommended: 12)
int32_t n_cn_div	in	Length of array n_cn_div (strongly recommended: 9)
int32_t n_cn_shift	in	Length of array n_cn_shift (strongly recommended: 3)

4.1.2 Coefficients for characteristically (c_lin_coeff_batch)

Name: c_lin_coeff_batch

Declaration: void __cdecl C_lin_coeff_batch(double z[], int32_t z_fpp, double theta[], int32_t C_in[], bool inverse, int32_t C_fpp, int32_t C_int_fpp, uint16_t method, int32_t ci_mean_sel, double cc[], uint32_t cc_fpp[], int32_t ccx[], double cci[], double best_fit[], uint32_t cn_div[], uint32_t cn_shift[], uint32_t n_samples, int32_t n_sample_x_devices, int32_t n_cc, int32_t n_cn_div, int32_t n_cn_shift);

name	type	description
double z[]	in	array of the result samples (reference values, e.g. Pressure, Humidity)
int32_t z_fpp	in	fix point position for the result (typically 15)
double theta[]	in	array of temperature samples (reference values)
int32_t C_in[]	in	array of capacitance ratios for example 4 calibration points per device under test: DUT1[1], DUT1[2], DUT1[3], DUT1[4], DUT2[1], DUT2[2], DUT2[3], DUT2[4],

name	type	description
		DUT3[1]...
bool inverse	in	flag to sign C_r is inverse $\text{false} := C_{in} = C_r = \frac{C_{measure}}{C_{ref}}$ $\text{true} := C_{in} = C_i = \frac{C_{ref}}{C_{measure}}$
int32_t C_fpp	in	fix point position for C_in typ: C_fpp = 22 => 22 fractional bits (for Linearize firmware strongly recommended: 22)
int32_t C_int_fpp	in	fix point position of c ratio to determine division steps. (for Linearize firmware strongly recommended: 26)
uint16_t method	in	select a method to determine characteristic function: 0:= median for each measurement point median over all devices 1:= mean for each measurement point mean value over all devices 2:= all use all measurement points
int32_t ci_mean_sel	in	select element for mean determination typically: n_samples/2
double cc[]	out	array of 12 coefficients for the result polynomial
uint32_t cc_fpp[]	out	array of 12 integer values with the fix point positions to ccx
int32_t ccx[]	Out → EEPROM	array 12*3 unsigned integer coefficients for DSP
double cci[]	out	array of 12 coefficients for the inverse capacitance polynomial
double best_fit[]	out	mathematical error vs z input (best fit)
uint32_t cn_div[]	Out → EEPROM	array of 9 integer values contains the number of division steps for the cc determination
uint32_t cn_shift[]	Out → EEPROM	Array of 3 signed integer value for the shift steps after last sum. n_shift < 0 := right shift n_shift >= 0 := left shift

name	type	description
uint32_t n_samples	in	number of samples / length of the arrays theta[], z[] and error_vs_z[]
int32_t n_sample_x_devices	in	number of samples * devices / length of the array C_in[]
int32_t n_cc	in	length of array cc (strongly recommended: 12)
int32_t n_cn_div	in	length of array n_cn_div (strongly recommended: 9)
int32_t n_cn_shift	in	length of array n_cn_shift (strongly recommended: 3)

4.1.3 Nominal value for 2-point calibration (c_2p_nominal)

Name: c_2p_nominal

Function: Determines nominal values from reference values and coefficients

Declaration: void __cdecl C_2p_nominal(double z[], double theta[], double cc[], int32_t C_fpp, int32_t C_out_hex[], double C_out[], int32_t n_samples, int32_t n_cc);

name	type	description
double z[]	in	Array of result samples (reference values, e.g. Pressure, Humidity)
double theta[]	in	Array of temperature samples (reference values)
double cc[]	in	Array of 12 polynomial coefficients, cci values from c_lin_coeff_batch
int32_t C_fpp	in	Fix point position for C_out e.g. C_fpp = 22 => 22 fractional bits (for Linearize firmware strongly recommended: 22)
int32_t C_out_hex[]	Out → EEPROM	Array of the nominal capacitance ratios (integer with C_fpp fractional bits), x _i
int32_t C_out[]	out	Array of the nominal capacitance ratios.
int32_t n_samples	in	Number of calibration points (1 or 2) / length of the array z[], theta[], C_out[] and C_out_hex[]
int32_t n_cc	in	Length of array cc (strongly recommended: 12)

4.2 Temperature

4.2.1 Coefficients for single devices (t_lin_coeff_dut)

Name: t_lin_coeff_dut

Function: Determines polynomial coefficients which fit best with least square method for 3rd order to Capacitance and 2nd order to temperature. The coefficients were determined as double values and 3byte-Integer to write to EEPROM to **PICCAP**.

Declaration: void __cdecl T_lin_coeff_dut(int32_t R_in[], double theta[], uint32_t r_fpp, uint32_t r_int_fpp, uint32_t theta_fpp, bool r_inverse, double tc[], uint32_t tc_fpp[], int32_t tcx[], double tci[], uint32_t tn_div[], double error_vs_t[], int32_t n_samples, int32_t n_tn_div, int32_t n_tc);

name	type	description
double R_in[]	in	Array of resistor ratios
double theta[]	in	Array of the result samples
uint32_t r_fpp	in	Fix point position for r_in e.g. C_fpp = 22 => 22 fractional bits (for Linearize firmware strongly recommended: 22)
uint32_t r_int_fpp	in	Fix point position of r ratio to determine division steps. (for Linearize firmware strongly recommended: 25)
uint32_t theta_fpp	in	Fix point position for the result (default: 8)
bool r_inverse	in	Flag to sign R_r is inverse false := $r_{in} = r_r = \frac{r_{measure}}{r_{ref}}$ true := $r_{in} = r_i = \frac{r_{ref}}{r_{measure}}$ default := true
double tc[]	out	Array of 4 coefficients for the result polynomial round to 5 decimal fractional digits
uint32_t tc_fpp[]	out	Array of 4 integer values with the fix point positions to tcx
uint32_t tcx[]	out → EEPROM	Array of 4 unsigned integer coefficients for DSP

name	type	description
double tci[]	out	Array of 4 coefficients for the inverse capacitance polynomial
uint32_t tn_div[]	Out → EEPROM	Array of 3 integer values contains the number of division steps for the tc determination
double error_vs_t[]	out	Mathematic error vs temperature input
int32_t n_samples	in	Number of samples / length of the arrays R_in[] , theta[] and error_vs_t
int32_t n_tn_div	in	Length of array tn_div (strongly recommended: 3)
int32_t n_tc	in	Length of array tc (strongly recommended: 4)

4.2.2 Nominal value for 2-point calibration

Name: t_2p_nominal

Function: Determines nominal values from reference values and coefficients

Declaration: void __cdecl t_2p_nominal(double theta[], double tc[], int32_t R_fpp, int32_t T_out_hex[], double T_out[], int32_t n_points, int32_t n_tc);

name	type	description
double theta[]	in	array of temperature samples (reference values)
double tc[]	in	array of 4 polynomial coefficients
int32_t R_fpp	in	fix point position for t_out e.g. C_fpp = 22 => 22 fractional bits (for Linearize firmware strongly recommended: 22)
int32_t T_out_hex[]	out	array of the nominal temperature ratios (integer with r_fpp fractional bits)
double T_out[]	out	array of the nominal capacitance ratios.
int32_t n_points	in	number of calibration points (1 or 2) / length of array theta[], t_out and t_out_hex
int32_t n_tc	in	length of array tc (Strongly recommended: 4)

5 Linearize Example Source Code

5.1 Example C-Code

Example code for Microsoft VisualC++ 2010:

```
#include "StdAfx.h"
#include <stdio.h>
#include <conio.h>
#include <stdint.h>
#include "linearize_r01.h"

#define printPrghHeader      printf("\n-----
-----"); \
                                printf("\n----- PCap02plus
Linearization -----"); \
                                printf("\n----- (c) acam messelektronic
gmbh, 2013 -----"); \
                                printf("\n-----
-----\n\n");

/* --- main() --- */
int main ( int argc, char *argv[] )
{
    //printf("Hello World");
    printPrghHeader
    int i;

    /* ----- Parameter for c_lin_coeff_dut ----- */
    int C_in[] = { 0x5353F7, 0x523D70, 0x51374B, 0x504189, 0x4F5C28,
0x4E76C8, 0x4D9168, 0x4CBC6A, 0x4C5A1C, 0x53126E, 0x51EB85, 0x50E560, 0x4FDF3B, 0x4EF9DB,
0x4E147A, 0x4D1EB8, 0x4C49BA, 0x4BE76C, 0x529FBE, 0x5178D4, 0x50624D, 0x4F4BC6, 0x4E5604,
0x4D6041, 0x4C6A7E, 0x4B851E, 0x4B126E };
    double theta [] = { 15.6, 15.6, 15.6, 15.6, 15.6, 15.6, 15.6, 15.6,
25.1, 25.1, 25.1, 25.1, 25.1, 25.1, 25.1, 25.1, 25.1, 39.7, 39.7, 39.7, 39.7, 39.7,
39.7, 39.7, 39.7 };
    double z [] = { 0.208, 0.309, 0.408, 0.507, 0.606, 0.704, 0.803, 0.901,
0.945, 0.208, 0.309, 0.408, 0.507, 0.606, 0.704, 0.803, 0.901, 0.945, 0.208, 0.309,
0.408, 0.507, 0.606, 0.704, 0.803, 0.901, 0.945 };
    const int n_samples = sizeof(C_in) / sizeof(C_in[0]);
    //const int n_samples = C_in.length();
    const int n_cc = 12;
    const int n_cn_div = 9 ;
    const int n_cn_shift = 3 ;
    int C_fpp = 22;
    int C_int_fpp = 26;
    int z_fpp = 8 ;
    int inverse = 0 ;

    /* Outputs */
    double cc[n_cc];
    uint32_t cc_fpp[n_cc];
    int32_t ccx[n_cc];
    double cci[n_cc];
    uint32_t cn_div[n_cn_div];
    int cn_shift[n_cn_shift];
```

```

double error_vs_z[n_samples];
/* -----

/* ----- Parameter for c_2p_nominal ----- */
double z_nominal[]      = { 22.8, 73.6 };
double theta_nominal[]  = { 15.4, 39.7 };
int n_samples_2P        = sizeof(z_nominal) / sizeof(z_nominal[0]);

/* Outputs */
int   C_out_hex[2] ;
double C_out[2]      ;
/* -----

/* ----- Start -----*/
    c_lin_coeff_dut (   C_in, z, theta, C_fpp, C_int_fpp, z_fpp, inverse,
// input
                        cc, cc_fpp, ccx, cci, error_vs_z, cn_div, cn_shift,
// output
                        n_samples, n_cc, n_cn_div, n_cn_shift);
// size of arrays (input)

/* -----*/
printf("Coefficients for single devices -- c_lin_coeff_dut\n");
printf("Calibration values cc (ascending / [0..11])\n");
for ( i=0; i<=n_cc/4-1; i++ )
{
    printf("k0%d %f\tk1%d %f\tk2%d %f\tk3%d %f\n", i, cc[i*4+0], i, cc[i*4+1],
i, cc[i*4+2], i, cc[i*4+3]);
}

printf("\ncc_fpp (ascending / [0..11])\n");
for ( i=0; i<=n_cc/4-1; i++ )
{
    printf("#d %d\t#d %d\t#d %d\t#d %d\n", i*4+0, cc_fpp[i*4+0], i*4+1,
cc_fpp[i*4+1], i*4+2, cc_fpp[i*4+2], i*4+3, cc_fpp[i*4+3]);
}

printf("\ncn_div (ascending / [0..8])\n");
for ( i=0; i<=n_cn_div/3-1; i++ )
{
    printf("#d %d\t#d %d\t#d %d\n", i*3+0, cn_div[i*3+0], i*3+1,
cn_div[i*3+1], i*3+2, cn_div[i*3+2]);
}
/*-----*/
C_2p_nominal (      z_nominal, theta_nominal, cci, C_fpp,      // input
                  C_out_hex, C_out,                  // output
                  n_samples_2P, n_cc);                // size of arrays (input)

printf("\nNominal value for 2-point calibration -- c_2p_nominal\n");
for ( i=0; i<=n_samples_2P-1; i++ )
{
    printf("#d: Z: %f, theta: %f, C_out: %f, C_out_hex %x\n", i, z_nominal[i],
theta_nominal[i], C_out[i], C_out_hex[i]);
}
/*-----*/
printf("\nPress any key to continue");
getch();

return 0;      /* return value main() */

```

```
} // EOF
```

Header file:

```
#include "stdint.h"
#pragma pack(push)
#pragma pack(1)

#ifdef __cplusplus
extern "C" {
#endif

/*!
 * Function: determines polynomial coefficients which fit best with least
 * square method for 3rd order to capacity and 2nd order to temperature. The
 * coefficients were determined as double values and 3byte-Integer to write to
 * EEPROM to PCap.
 */
void __cdecl c_lin_coeff_dut(int32_t C_in[], double z[], double theta[],
    int32_t C_fpp, int32_t C_int_fpp, int32_t z_fpp, int inverse,
    double cc[], uint32_t cc_fpp[], int32_t ccx[], double cci[],
    double error_vs_z[], uint32_t cn_div[], int32_t cn_shift[],
    int32_t n_samples, int32_t n_cc, int32_t n_cn_div, int32_t n_cn_shift);

/*!
 * Function: Determines nominal values from reference values and coefficients
 */
void __cdecl C_2p_nominal(double z[], double theta[], double cc[],
    int32_t C_fpp, int32_t C_out_hex[], double C_out[], int32_t n_samples,
    int32_t n_cc);

/*!
 * C_lin_coeff_batch
 */
void __cdecl C_lin_coeff_batch(double z[], int32_t z_fpp, double theta[],
    int32_t C_in[], int inverse, int32_t C_fpp, int32_t C_int_fpp,
    uint16_t method, int32_t ci_mean_sel, double cc[], uint32_t cc_fpp[],
    int32_t ccx[], double cci[], double best_fit[], uint32_t cn_div[],
    int32_t cn_shift[], uint32_t n_samples, int32_t n_samples_x_devices,
    int32_t n_cc, int32_t n_cn_div, int32_t n_cn_shift);

/*!
 * Function: determines polynomial coefficients which fit best with least
 * square method for 3rd order to capacity and 2nd order to temperature. The
 * coefficients were determined as double values and 3byte-Integer to write to
 * EEPROM to PCap.
 */
void __cdecl T_lin_coeff_dut(int32_t R_in[], double theta[], uint32_t r_fpp,
    uint32_t r_int_fpp, uint32_t theta_fpp, int r_inverse, double tc[],
    uint32_t tc_fpp[], int32_t tcx[], double tci[], uint32_t tn_div[],
    double error_vs_t[], int32_t n_samples, int32_t n_tn_div, int32_t n_tc);

/*!
 * Function: Determines nominal values from reference values and coefficients
 */
void __cdecl T_2p_nominal(double theta[], double tc[], int32_t R_fpp,
    int32_t T_out_hex[], double T_out[], int32_t n_points, int32_t n_tc);

/*!
 * Rgp
```

```

*/
void __cdecl Rgp(double X[], double Y[], double *slope, double *offset,
    int32_t n_input);

long __cdecl LVDLLStatus(char *errStr, int errStrLen, void *module);

#ifdef __cplusplus
} // extern "C"
#endif

#pragma pack(pop)

```

5.2 Output via Terminal:

```

----- PCap02plus Linearization -----
----- <c> acam messeletronic gmbh, 2013 -----
-----
Coefficients for single devices -- c_lin_coeff_dut
Calibration values cc <ascending / [0..11]>
k00 6.366533 k10 5.682388 k20 -14.418227 k30 4.947223
k01 -5.296444 k11 13.150975 k21 -10.896175 k31 3.009743
k02 0.114421 k12 -0.284301 k22 0.235036 k32 -0.064652

cc_fpp <ascending / [0..11]>
#0 8 #1 11 #2 14 #3 17
#4 15 #5 18 #6 19 #7 21
#8 22 #9 24 #10 25 #11 26

cn_div <ascending / [0..8]>
#0 24 #1 24 #2 24
#3 24 #4 26 #5 25
#6 25 #7 26 #8 26

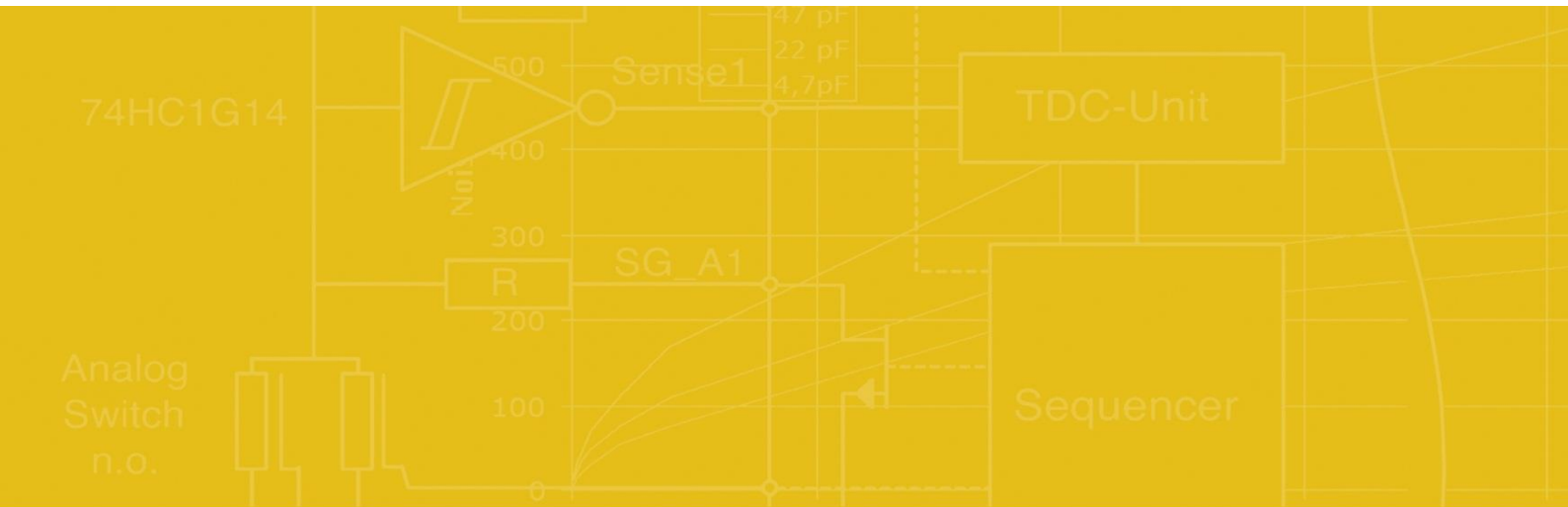
Nominal value for 2-point calibration -- c_2p_nominal
#0: Z: 22.800000, theta: 15.400000, C_out: 1.298861, C_out_hex 53208b
#1: Z: 73.600000, theta: 39.700000, C_out: 1.208541, C_out_hex 4d58bc

Press any key to continue

```

Figure 6-1: Output via Terminal

7 Miscellaneous



acam-messelectronic gmbh
Friedrich-List-Straße 4
76297 Stutensee-Blankenloch
Germany
Phone +49 7244 7419 - 0
Fax +49 7244 7419 - 29
E-Mail support@acam.de
www.acam.de