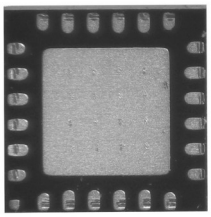
ScioSense



PCAP04

Capacitance-to-Digital Converter

PCAP04 datasheet

Revision: 6

Release Date: 2023-09-25

Document Status: Preliminary

PCap04 is a capacitance-to-digital converter (CDC) with integrated digital signal processor (DSP) for on-chip data post-processing. Its front end is based on ScioSense PICOCAP principle. This conversion principle offers outstanding flexibility with respect to power consumption, high resolution and speed. Further, PCap04 covers a wide capacitance input range from a few femtofarads up to several hundreds of nanofarads.

It is easy to configure the PCap04 for different capacitance measurement tasks, i.e. single as well as differential sensors in both, grounded or floating connection. The on-chip DSP allows to implement sensor algorithms like linearization and temperature compensation, with data output in a digital (SPI or IIC) or analog (PDM/PWM) way.

This revision applies to silicon version V2 in which the I2C bug is corrected.

Key Features & Benefits

- Up to 6 capacitors grounded, 3 capacitors floating
- Capacitance range 1pF to 100nF
- Internal reference 1pF to 31pF
- Integrated guard driver
- Up to 8aF at 2.5Hz and 10pF base capacitance
- Up to 50kHz sample rate
- Up to 20-bit resolution
- 32-bit DSP
- 3k ROM code, 1k NVRAM
- 96 x 32 bit RAM
- SPI / IIC interface
- PDM / PWM outputs, GPIO
- Supply voltage 2.1/3.0V to 3.6V
- Operating current down to 3µA
- PCap04-Bxxx -40°C to 85°C
- PCap04-Axxx -40°C to 125°C
- QFN24 or die (1.588mm x 1.46mm)VDD: 1.71 to 1.98V

Applications

- Building
- Position sensors
- Pressure sensors
- Force sensors
- Proximity sensor
- Acceleration sensors
- Inclination sensors
- Humidity sensors
- Dewpoint sensors
- Tilt sensors
- Angle sensors
- Wireless applications
- Level sensors
- IoT devices

Content Guide

1 Block diagram

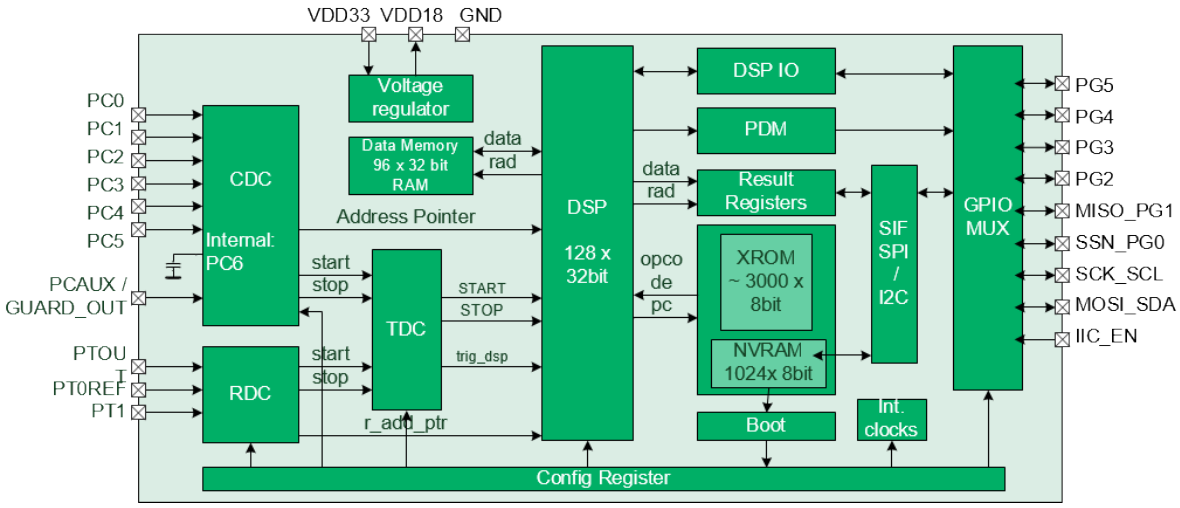


Figure 1: Functional Blocks

2 Pin assignment

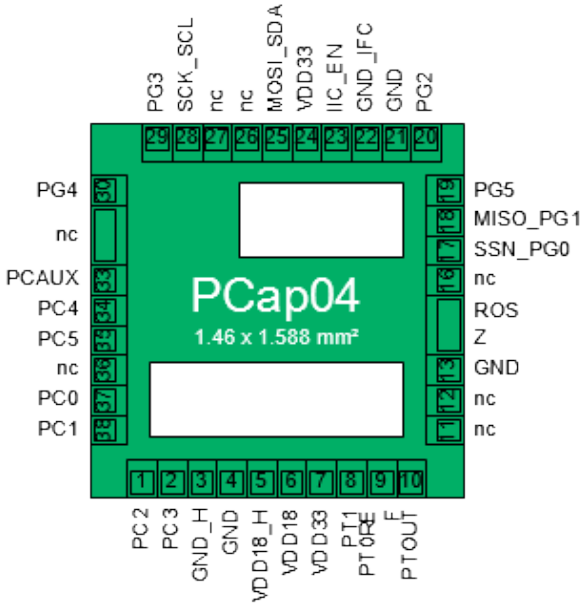


Figure 2: Pin Diagram of PCAP04 Die

Die dimensions: 1.588 mm x 1.46 mm (w/ seal) with pad pitch 120 µm, pad opening is 85 µm x 85 µm.

Thickness dice in waffle pack 300 µm ± 10 µm

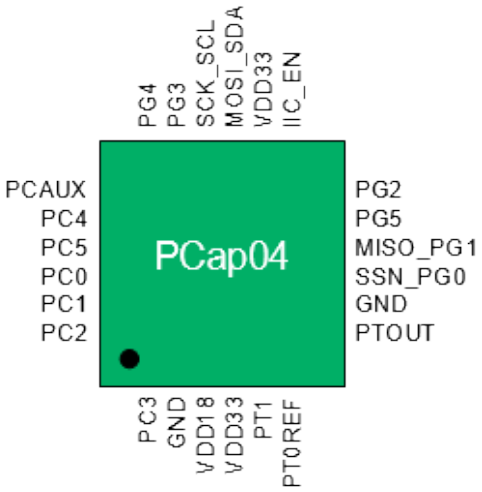
Thickness wafer 735 µm.

Table 1: Pad coordinates

Pin Name	If not used	Description	
1	PC2	260.0	59.5
2	PC3	380.0	59.5
3	GND_H	488.0	59.5
4	GND	608.0	59.5
5	VDD18_H	728.0	59.5
6	VDD18	848.0	59.5
7	VDD33	968.0	59.5
8	PT1	1088.0	59.5
9	PT0REF	1208.0	59.5
10	PTOUT	1328.0	59.5
11	n.c.	No pad	No pad
12	n.c.	No pad	No pad
13	GND	1528.5	501.0
14/15	n.c.	No pad	No pad
16	n.c.	No pad	No pad
17	SSN_PG0	1528.5	965.0
18	MISO_PG1	1528.5	1085.0
19	PG5	1528.5	1205.0
20	PG2	1333.0	1400.5
21	GND	1213.0	1400.5
22	GND_ifc	1093.0	1400.5
23	IIC_EN	973.0	1400.5
24	VDD33	853.0	1400.5
25	MOSI_SDA	733.0	1400.5
26	n.c.	No pad	No pad
27	n.c.	No pad	No pad
28	SCK_SCL	374.7	1400.5
29	PG3	254.7	1400.5
30	PG4	59.5	1205.0
31/32	n.c.	No pad	No pad

33	PCAUX	59.5	859.9
34	PC4	59.5	739.8
35	PC5	59.5	619.8
36	n.c.	No pad	No pad
37	PC0	59.5	379.8
38	PC1	59.5	259.8

Pad coordinates are center/center (x/y-direction), relative to die origin die dimensions: 1.588 mm x 1.46 mm (with seal, 15 µm each side) with pad pitch 120 µm, pad opening is 85 µm x 85 µm.



Package dimension: 4 x 4 mm²

Suitable socket: e.g. Plastronics 32QN50S15050D.

Figure 3: Pin Diagram of PCAP04 QFN24

Table 2: Pin description

Pins Die	Pins QFN24	Pin Name	If not used	Description
1	24	PC2	open	CDC port
2	1	PC3	open	CDC port
3		GND_H	GND	Ground
4	2	GND	GND	Ground
5		VDD18_H	VDD18	Core supply voltage
6	3	VDD18[1]	VDD18	Core supply voltage
7	4	VDD33[2]	VDD33	I/O supply voltage
8	5	PT1	open	RDC port (temperature sensor)

1 Connect buffer capacitor ≥ 4.7µF
2 Connect buffer capacitor ≥ 10µF

9	6	PT0REF	open	RDC port (temperature sensor or external reference)
10	7	PTOUT[3]	open	Discharge capacitor for RDC
11		n.c.		No pad
12		n.c.		No pad
13	8	GND[4]		Ground
14/15		n.c.		No pad
16		n.c.		Always open
17	9	SSN_PG0	open	Serial select line. Otherwise, general purpose I/O port
18	10	MISO_PG1	open	Master in/Slave out when SPI is used. Otherwise, general purpose I/O port
19	11	PG5	open	General purpose I/O port
20	12	PG2	open	General purpose I/O port
21		GND	GND	Ground
22		GND_ifc	GND	Ground
23	13	IIC_EN		Serial interface select, 0 = SPI enable 1 = IIC enable
24	14	VDD33	VDD33	I/O supply voltage
25	15	MOSI_SDA		Master out/Slave in when SPI is used. Otherwise, serial data for IIC
26		n.c.		Always open
27		n.c.		Always open
28	16	SCK_SCL		Serial clock for SPI/IIC
29	17	PG3	open	General purpose I/O port
30	18	PG4	open	General purpose I/O port
31/32		n.c.		No pad
33	19	PCAUX	open	Auxiliary port. For external compensation capacitance or external discharge resistor. Guarding output
34	20	PC4	open	CDC port
35	21	PC5	open	CDC port
36		n.c.	open	Always open
37	22	PC0	open	CDC port
38	23	PC1	open	CDC port

[3] Connect 10nF C0G

[4] Center pad is internally connected to GND. No wires other than GND are allowed underneath. It is recommended to not use the center pad. Too much solder paste could reduce solder quality.

3 Absolute maximum ratings

Table 3: Absolute Maximum Ratings

Symbol	Parameter	Min	Max	Units	Comments
Electrical Parameters					
V_{DD} / V_{GND}	Supply Voltage to Ground	-0.3	4.0	V	
V_{IN}	Input Pin Voltage to Ground	-0.3	4.0	V	
I_{SCR}	Input Current (latch-up immunity) @125°C	± 100		mA	JEDEC JESD78D Nov 2011
Continuous Power Dissipation (TA = 70°C)					
P_T	Continuous Power Dissipation		1.44	mW	
Electrostatic Discharge					
ESD_{HBM}	Electrostatic Discharge HBM	± 1000		V	JS-001-2014
Operating and Storage Conditions					
T_A	Operating Ambient Temperature PCap04-Bxxx PCap04-Axxx	 -40 -40	 85 125	 °C °C	
R_{THJA}	Junction to Ambient Thermal Resistance		28	°C/W	
T_J	Operating Junction Temperature PCap04-Bxxx PCap04-Axxx		 85 125	 °C °C	
T_{STRG}	Storage Temperature Range	-55	150	°C	
T_{BODY}	Maximum Package Body Temperature during Reflow		260	°C	IPC/JEDEC J-STD-020 The reflow peak soldering temperature (body temperature) is specified according to IPC/JEDEC J-STD-020 "Moisture/Reflow Sensitivity Classification for Non-hermetic Solid State Surface Mount Devices." The lead finish for Pb-free leaded packages is "Matte Tin" (100% Sn)
R_{HNC}	Relative Humidity (non-condensing)	5	85	%	
MSL	Moisture Sensitivity Level	3			Maximum floor life time of 168 h
t_{STRG_DOF}	Storage Time for DOF/die or wafers on foil		3	months	Refers to indicated date of packing
T_{STRG_DOF}	Storage Temperature for DOF/die or wafers on foil	17	28	°C	

R_{HOPEN_DOF}	Relative Humidity for DOF/die or wafers on foil in open package		15	%	Opened package
$R_{HUNOPEN\_DOF}$	Relative Humidity for DOF/die or wafers on foil in sealed package	40	60	%	Sealed bag
t_{STRG_WP}	Storage Time for WP/wafers or die in waffle pack		6	Months	17-28°C 40-60% relative humidity storage in original Ultrapack boxes
t_{STRG_WP}	Storage Time for WP/wafers or die in waffle pack		2	Years	19-25°C <15% relative humidity storage in closed cabinet with dry air
t_{STRG_WP}	Storage Time for WP/wafers or die in waffle pack		5	Years	19-25°C <5% relative humidity storage in closed cabinet with dry air
t_{STRG_WP}	Storage Time for WP/wafers or die in waffle pack		10	years	19-25°C <5% relative humidity storage in closed cabinet and closed Ultrapak box with safeguarded Nitrogen atmosphere

Stresses beyond those listed under Absolute Maximum Ratings may cause permanent damage to the device. These are stress ratings only. Functional operation of the device at these or any other conditions beyond those indicated under Electrical Characteristics is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

4 Electrical characteristics

Characteristics indicate conditions for which the device is guaranteed to be functional. For details on the test conditions see the notes at the table footer.

Table 4: Electrical characteristics of PCAP04

Symbol	Parameter	Conditions[1]		Min	Typ	Max	Unit
V_{DD}	Supply Voltage	Relative to GND Power-down: VDD has to be <0.05V before power-up	4	2.1		3.6	V
V_{DD}	Supply Voltage	NVRAM recall -Bx: -40°C to 85°C -Ax: -40°C to 125°C		2.5		3.6	V
V_{DD}	Supply Voltage	NVRAM store -25°C to 60°C		3.0		3.6	V
$V_{IO\_DIGITAL}$	Digital Ports Input Voltage	Relative to GND	4	- 0.6	3.3	V_{DD} + 0.6 ≤ 3.6	V
$V_{IO\_DIGITAL}$	Digital Ports Input Switching Levels	HIGH to LOW LOW to HIGH	2	0.7 * V_{DD}		0.3 * V_{DD}	V
V_{OH}	Digital Ports Output Voltage	HIGH	2	V_{DD} - 0.4		V_{DD} + 0.1	V
V_{OL}	Digital Ports Output Voltage	LOW	2	-0.1		0.4	V
I_{leakH}	Digital Ports Leakage	IIC_EN=V_{DD} HIGH	2	-0.1		1.0	µA
I_{leakPU}	Digital Ports Leakage	Internal pull-ups	2	-2.8		-5.2	µA
I_{leakL}	Digital Ports Leakage	IIC_EN=V_{DD} LOW	2	-1.0		0.1	µA

[1] Test conditions are:
1. 100% production tested
2. 100% production tested at 85°C (-Bxxx)/125°C (-Axxx) wafer sort and guaranteed by design and characterization at specified temperatures.
3. Sample tested only
4. Parameter is guaranteed by design and characterization testing.
5. Parameter is a typical value only
6. 100% production tested at 25°C and guaranteed by design and characterization for industrial temperature range
7. Important: We guarantee the data for data retention and endurance only under the assumption, that the customer does not change the registers 62 and 63 and NVRAM adr 654 to 959 (Unique ID). In addition, it is mandatory to follow the given procedure for ERASE NVRAM as described in section NVRAM and ROM precisely. Otherwise, we do no longer guarantee the data retention time and endurance cycles.

I_{ddq}	Static Supply Current	-40°C 35°C 85°C 125°C	2		1.28 1.6 4.26 22.5	- 5 18 50	µA
R_P	Precharge Resistance	R_C_PCHG0 R_C_PCHG1	2	7 126	10 180	13 234	kOhm
R_D	Discharge Resistance	R_DCHG0 R_DCHG1 R_DCHG2 R_DCHG3	2	7 21 63 126	10 30 90 180	13 39 117 234	kOhm
C_{ref0} C_{ref1} C_{ref2} C_{ref3} C_{ref4}	Internal Reference		5		0.4 1.4 3.4 7.4 15.4		pF
C_R	Internal RDC Discharge Cap	PTOUT n.c.	2	65	94	122	pF
I_{C_G7} I_{C_GLow}	Driver 7 Driver Low	PC0 = V_{DD} PC0 = V_{DD} * 0.05 PCAUX = V_{DD}	3		-3.5 -9		mA
V_{GUARD0} V_{GUARD1} V_{GUARD2} V_{GUARD3}	Gain Guarding Opamp	PC0 = 1.0V	3		1.00 1.01 1.02 1.03		V
$R_{R\_DIS\_SEN}$	RDC Temperature Sensor	-40 °C 35°C 85 °C 125 °C	2,6	- 1120 1200 1360	1047 1330 1572 1688	- 1680 1800 2040	Ohm
$R_{R\_DIS\_REF}$	RDC Reference		2	1100	1476	1690	Ohm
C_R	RDC Internal Capacitance		2	65	94	122	pF
f_{OLF0} f_{OLF1} f_{OLF2} f_{OLF3}	OLF Frequency	Low temperature minimum values, high temperature maximum values	2	2.8 18.5 35.5 69.0	10 60 100 200	16.5 96.4 171.4 305	kHz
f_{OHF}	OHF Frequency	High temperature minimum values, low temperature maximum values 85°C 125°C	2	 1.36 1.17	 2	 2.6	MHz

	NVRAM Data Retention PCap04-Bxxx PCap04-Axxx	3.0V to 3.6V 85°C 125°C	4,7	 20 20			Years
	NVRAM Endurance PCap04-Bxxx PCap04-Axxx	 25°C 85°C 25 °C 125 °C	4,7	 10^4 10^3 10^5 10^4			Cycles

4.1 Measurement Current[2]

Table 5: Total Current I [µA] as a Function of Conversion Rate (CONV_TIME) and Resolution (C_AVRG) in Triggered Mode

LP Oscillator Freq. [kHz]	CONV_ TIME	Measure Rate [Hz]	I [µA]					
			C_AVRG (RMS Resolution [Bits])					
			1	4	16	64	256	1024
			(13.6)	(14.6)	(15.6)	(16.6)	(17.8)	(18.6)
50	10000	2.5	2.3	2.5	2.7	4.6	11.8	44.0
50	2500	10	2.8	3.3	4.6	12.2	43.8	
50	1250	20	3.3	4.2	7.9	23.9		
50	625	40	4.6	6.0	14.0			
50	250	100	8.3	12.0	32.2			
50	125	200	14.3	22.3				
50	50	500	32.6	53.8				
50	25	1000	63.9					
50	12	2080	90					
200	24	4160	156					
200	12	9320	305					

[2] Temperature measurement in addition to capacitive measurement will add between 2µA and 10µA approximately, depending on speed. Total consumption values below 30 µA may be obtained only when driving the on-chip 1.8 volts core supply generator in an energy-saving mode; ultimate microampere savings also demand to slow down the DSP. Typical data.

4.2 CDC Characteristics[3]

Table 6: CDC Resolution

	Floating Fully Compensated			Grounded Internally Compensated		
Rate	RMS Noise	Eff. Resolution [Bits]		RMS Noise	Eff. Resolution [Bits]	
[Hz]	[aF]	10pF Base	1pF Span	[aF]	10pF Base	1pF Span
2.5	8	20.2	16.9	9	20.1	16.8
5	12	19.7	16.4	13	19.6	16.3
10	19	19.0	15.7	19	19.0	15.7
25	28	18.4	15.1	26	18.5	15.2
100	56	17.4	14.1	52	17.5	14.2
250	91	16.7	13.4	78	17.0	13.7
1000	156	16.0	12.7	148	16.0	12.7
2000	218	15.5	12.2	192	15.6	12.3
4000	328	14.9	11.6	272	15.1	11.8
8000				385	14.6	11.3

Table 7: Voltage-Dependent Offset Error (PSRR)

Base /Gain	Mode	2.4V	2.7V	3.0V	3.3V	3.6V
10 pF	Single-ended, internal compensation	< 1 fF	< 1 fF	0 fF	< 1 fF	< 1 fF
150 pF		< 1 fF	< 1 fF	0 fF	< 1 fF	< 1 fF
10 pF	Floating, full compensation	< 1 fF	< 1 fF	0 fF	< 1 fF	< 1 fF
150 pF		< 1 fF	< 1 fF	0 fF	< 1 fF	< 1 fF

[3] Typical capacitive noise and resolution vs. output data rate, 10pF base + 1pF span, fast settle, MR1, V = 3.0V. Span means the maximum variation of the sensor capacitance in the application. The table gives the root mean-square (RMS) noise in aF as a function of output data rate in Hz, measured at 3.0V supply voltage using the maximum possible sample size for in-chip averaging at the minimum possible cycle time. Bit values are calculated as a binary logarithm of noise over the span (BITs = ln(span/noise)/ln(2)). The measurements have been done with the PCap04 evaluation board, with fixed C0G ceramic capacitors, configuration for maximum resolution.
Both, sensor and reference are connected “floating” or “grounded”, as indicated. In floating mode compensation mechanisms for both internal and external stray capacitances are activated, in grounded mode only the internal compensation is active.

Table 8: Voltage-Dependent Gain Error (PSRR)

Base /Gain	Mode	2.4V	2.7V	3.0V	3.3V	3.6V
10pF / 4.7pF	Single-ended, internal compensation	1.9 fF	0.9 fF	0 fF	- 0.7 fF	- 1.2 fF
150pF / 47pF		- 22 fF	- 10 fF	0 fF	0.7 fF	20 fF
10pF / 4.7pF	Floating, full compensation	0.6 fF	6 fF	0 fF	- 0.7 fF	- 1.5 fF
150pF / 47pF		- 11 fF	- 11 fF	0 fF	8 fF	19 fF

Table 9: Temperature-Dependent Offset and Gain Error

Error Type	Capacitance [pF]	Mode	Temperature Range	Typ. Drift
Offset drift	10	Single-ended, internal compensation	-10°C to 85°C	2.5 fF
	150			9 fF
	10	Floating, full compensation		1.5 fF
	150			12 fF
Gain drift	10 + 4.7	Single-ended, internal compensation		42 fF = 94 ppm/K
	150 + 47			69 fF = 15 ppm/K
	10 + 4.7	Floating, full compensation		8 fF = 18 ppm/K
	150 + 47			96 fF = 22 ppm/K

4.3 RDC Characteristics

Table 10: Resolution RDC Unit

Measurement Conditions	R2/Rref Typ.	RMS Noise R2/Rref	Typical RMS Noise Temperature [4]
No averaging, 2 fake measurements	0.899	31.7 ppm	12.2 mK
16-fold averaging, 8 fake measurements	0.897	20.2 ppm	7.8 mK
Measurement conditions	10 nF @ PTOUT, 25°C		

Typical linearity error with internal Al-thermometer after linearization and conversion into temperature, assuming a linear relation between temperature and resistivity:

[4] After linearization in post-processing software

- -20°C < Temp. < 0°C: 290 mK
- 0°C < Temp. < 80°C: 110 mK

4.4 Timing Characteristics

Table 11: Timing Characterisitcs

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
t_{START}	Start-Up Time[5]		3.9		4.0	ms
t_C	CDC Discharge Time	Measure range 1	0		20	µs
t_R	RDC Discharge Time	Measure range 1	0		20	µs
f_{SPI}	SPI Bus Frequency	Clock frequency	0		20	MHz
f_{I2C}	I²C Bus Frequency	Data rate	0	100		kHz

[5] This timing refers to a hardware power-on. For software POR see 59

5 Detailed Description

PCap04 is an integrated solution for digitizing capacitive and resistive sensors, including a DSP for data processing like linearization and temperature correction. A 6-channel CDC allows to handle grounded and floating sensors in single and differential mode. The capacitance range applicable is from a few pF to hundreds of nanofarads. The RDC unit is mainly intended for measuring temperature, by means of an internal sensor and reference or by means of external resistors like PT1000.

A 32-bit digital signal processor (DSP) in Harvard architecture is integrated to the PCap04. It is responsible for taking the information from the CDC and RDC measuring units, for processing the data and making them available to the user interface. Both, the CDC/RDC raw data as well as the data processed by the DSP are stored in the RAM. The program for the DSP is stored in the NVRAM. The DSP can collect various status information from a set of 64 I/O Bits and write back 16 of those. This way, the DSP can react on and also control the GPIO pins of PCap04. The DSP is internally clocked at approximately 60MHz. The internal clock is stopped through a firmware command, to save power. The DSP starts again upon a GPIO signal or an “end of measurement” condition.

In its simplest form, the DSP transfers the pure time measurement information from the CDC/RDC to the read registers without any further processing. The next higher step is to calculate the capacitance ratios including the information from the compensation measurements, as it is provided in ScioSense standard firmware version PCap04_standard_v01.hex. Finally, ScioSense provides a ready-made linearize firmware that performs a linearization via polynomial of third degree and temperature compensation via polynomial of second degree. Many functional blocks for the linearization firmware are implemented as ROM code. This way, the main firmware can be very compact and can fit into the 1k NVRAM.

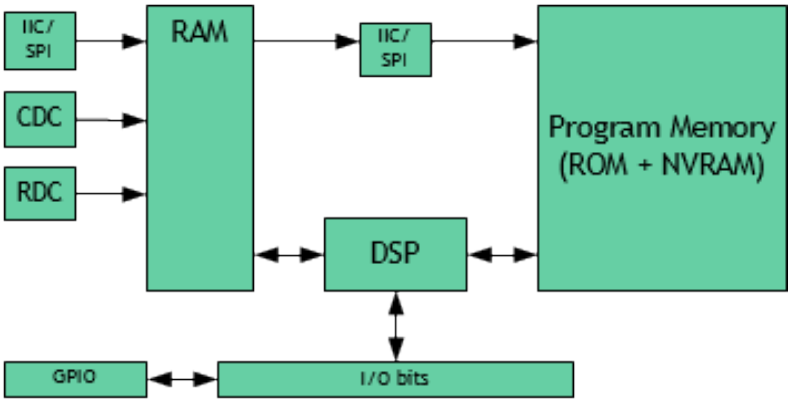


Figure 4: CDC, RDC and DSP Embedding

The content of the read registers will always depend on the firmware in use. With the standard firmware it will be the pure capacitance and resistance ratios. With the linearization firmware it might the linearized and calibrated result, e.g. a pressure given in Pascal or humidity given in percent. The DSP is ScioSense proprietary to cover low-power tasks as well as very high data rates. It is programmed in assembler. A user-friendly

assembler software with a graphical interface, help text pop-ups as well as sample code sustain programming efforts.

6 Register Description

6.1 Configuration Registers

The PCap04 offers 48 registers for configuring the hardware (CDC, RDC, clocks, PDM/PWM, DSP). All these 48 registers are of one byte size. Additional four registers are used as special function registers. The 48th register contains nothing but one single bit, the RUNBIT, which enables/disables the front-end and the DSP. All configurations are written simultaneously to registers and to the RAM Part of NVRAM and can be read back.

Table 12: Register Overview

Addr	Name	<D7>	<D6>	<D5>	<D4>	<D3>	<D2>	<D1>	<D0>
0	CFG0	I2C_A		OLF_FTUNE				OLF_CTUNE	
1	CFG1	OX_DIS		OX_DIV4	OX_ AUTOST OP_DIS	OX_STO P	OX_RUN		
2	CFG2	RDCHG_INT_SEL1		RDCHG_INT_SEL0		RDCHG_ INT_EN		RDCHG_ EXT_EN	
3	CFG3		AUX_ PD_DIS	AUX _ CINT	RDCHG _OPEN		RDCHG _ PERM _ EN	RDCHG_ EXT _ PERM	RCHG _ SEL
4	CFG4	C_REF _ INT		C_ COMP_ EXT	C_ COMP_ INT			C_ DIFFERE NTIAL	C_ FLOATIN G
5	CFG5	CY_PRE_ MR1_ SHORT		C_PORT_ PAT		CY_ HFCLK_ SEL	CY_ DIV4_DIS	CY_PRE_ LONG	C_DC_ BALANCE
6	CFG6			C_PORT_EN					
7	CFG7	C_AVRG<7:0>							
8	CFG8				C_AVRG<12:8>				
9	CFG9	CONV_TIME<7:0>							
10	CFG10	CONV_TIME<15:8>							
11	CFG11		CONV_TIME<22:16>						
12	CFG12	DISCHARGE_TIME<7:0>							
13	CFG13	C_STARTONPIN			C_TRIG_SEL			DISCHARGE_TIME <9:8>	
14	CFG14	PRECHARGE_TIME<7:0>							

15	CFG15			C_FAKE				PRECHARGE_TIME <9:8>	
16	CFG16	FULLCHARGE_TIME <7:0>							
17	CFG17		C_REF_SEL					DISCHARGE_TIME <9:8>	
18	CFG18	C_G_OP_ RUN	C_G_OP_ EXT	C_G_EN					
19	CFG19	C_G_OP_VU		C_G_OP_ATTN		C_G_TIME			
20	CFG20	R_CY					C_G_OP_TR		
21	CFG21	R_TRIG_PREDIV<7:0>							
22	CFG22		R_TRIG_SEL			R_AVRG		R_TRIG_PREDIV <9:8>	
23	CFG23	R_PORT_EN		R_PORT_ EN_IMES	R_PORT_ EN_IREF		R_FAKE	R_STARTONPIN	
24	CFG24			TDC_CHAN_EN		TDC_ ALUPER MOPEN	TDC_ NOISE_ DIS	TDC_MUPU_SPEED	
25	CFG25	TDC_MUPU_NO							
26	CFG26	TDC_QHA_SEL						TDC_ NOISE_ CY_DIS	
27	CFG27	DSP_MOFLO_EN				DSP_SPEED		PG1x PG3	PG0x PG2
28	CFG28	WD_DIS							
29	CFG29	DSP_STARTONPIN				DSP_FF_IN			
30	CFG30	PG5_ INTN_EN	PG4_ INTN_EN			DSP_START_EN			
31	CFG31	PI1_ TOGGLE_ EN	PI0_ TOGGLE_ EN	PI0_RES		PI0_ PDM_SEL	PI0_CLK_SEL		
32	CFG32			PI1_RES		PI1_ PDM_SEL	PI1_CLK_SEL		
33	CFG33	PG_DIR_IN				PG_PU			
34	CFG34	INT_ TRIG_BG	DSP_ TRIG_BG	BG_ PERM	AUTO START				
35[1]	CFG35	CDC_GAIN_CORR<7:0>							
36	CFG36	-							

[1] Registers 35 to 42 depend on the firmware. The content shown is the one for the standard firmware

37	CFG37	-							
38	CFG38	BG_TIME							
39	CFG39	PULSE_SEL1				PULSE_SEL0			
40	CFG40	C_SENSE_SEL							
41	CFG41	R_SENSE_SEL							
42	CFG42		ALARM1 _SEL ECT		ALARM0 _SEL ECT	EN_ ASYNC_ RD	HS_ MODE_ SEL	R_ MEDIAN_ EN	C_ MEDIAN_ EN
…									
47	CFG47								RUNBIT
48	CFG48					MEM_LOCK			
49	CFG49	SERIAL_NUMBER<7:0>							
50	CFG50	SERIAL_NUMBER<15:8>							
…									
54	CFG54	MEM_CTRL							
…									
62	CFG62	CHARGE_PUMP<7:0>							
63	CFG63	CHARGE_PUMP<15:8>							

6.2 Read Registers

The content of the result registers depends strongly on the firmware which is used. For the standard firmware the result registers show the capacitance ratios.

Addr	Name	<D7>	<D6>	<D5>	<D4>	<D3>	<D2>	<D1>	<D0>
0	RES0	7							0
1		15							8
2		23							16
3		31							24
4	RES1	7							0
5		15							8
6		23							16
7		31							24

8	**RES2**	7							0
9		15							8
10		23							16
11		31							24
12	**RES3**	7							0
13		15							8
14		23							16
15		31							24
16	**RES4**	7							0
17		15							8
18		23							16
19		31							24
20	**RES5**	7							0
21		15							8
22		23							16
23		31							24
24	**RES6**	7							0
25		15							8
26		23							16
27		31							24
28	**RES7**	7							0
29		15							8
30		23							16
31		31							24
32	**STATUS_0**	POR_ FLAG_ WDG	POR_ FLAG_ CONFIG	IR_ FLAG_ COLL	AUTOBO OT		RDC_ READY	CDC_ ACTIVE	RUNBIT
33	**STATUS_1**					RDC_ER R	MUP_ ERR	ERR_ OVFL	COMB_ ERR
34	**STATUS_2**		C_ PORTERR _INT	C_ PORTERR 5	C_ PORTERR 4	C_ PORTERR 3	C_ PORTERR 2	C_ PORTERR 1	C_ PORTERR 0

6.3 Detailed Configuration Register Description

6.3.1 Configuration Register 0

Table 13: Configuration Register 0

Address 0x00			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:6	**I2C_A**	0 to 3	Complement to the I²C-address
5:2	**OLF_FTUNE**	0: Minimum 7: Typ., recommended 15: Maximum	Fine-tune the low-frequency clock
1:0	**OLF_CTUNE**	0: 10 kHz 1: 50 kHz 2: 100 kHz 3: 200 kHz	Coarse -tune the low-frequency clock

6.3.2 Configuration Register 1

Table 14: Configuration Register 1

Address 0x01			
Bits	**Bit Name**	**Settings**	**Bit Description**
7	**OX_DIS**	Default : 0	Disable the OX clock
5	**OX_DIV4**	0: No division; f_ox = 2 MHz 1: Division by 4; f_ox = 0.5MHz	OX clock frequency : Raw freq./ 4
4	**OX_AUTOSTOP_DIS**	Default: 0	ScioSense internal bits
3	**OX_STOP**	Default: 0	ScioSense internal bits
2:0	**OX_RUN**	0: Generator off 6: OX latency = 1 / f_{OLF} 3: OX latency = 2 / f_{OLF} 2: OX latency = 31 / f_{OLF} 1: OX runs permanently	Control the permanency or the latency of the OX generator. Latency means an oscillator settling time before a measurement starts

6.3.3 Configuration Register 2

Table 15: Configuration Register 2

Address 0x02			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:6	**RDCHG_INT_SEL1**	0: 180 kΩ 1: 90 kΩ 2: 30 kΩ (default) 3: 10 kΩ	Same, but for ports PC4 – PC5
5:4	**RDCHG_INT_SEL0**		Choice of one out of 4 on-chip discharge resistors for the CDC ports PC0 – PC3 plus internal port PC6
3	**RDCHG_INT_EN**	0: Off 1: Internal on (default	Enable internal discharge resistors
1	**RDCHG_EXT_EN**	0: Off (default) 1: External on	Enable external discharge resistor switching on PCAUX during discharge phase (High-Z during pre- and full-charge phase) Note: **AUX_PD_DIS** (PCAUX pull down disable) : 1

6.3.4 Configuration Register 3

Table 16: Configuration Register 3

Address 0x03			
Bits	**Bit Name**	**Settings**	**Bit Description**
6	**AUX_PD_DIS**	0: Pull-down active 1: Pull-down disabled	Disable pull-down at PCAUX
5	**AUX_CINT**	0: Normal (default) 1: Aux during c-internal active	Activates auxiliary port PCAUX during internal C-reference conversion only
4:3	**RDCHG_OPEN**	2: Recommended	ScioSense internal bits
2	**RDCHG_PERM_EN**	0: Off (default) 1: On	Keep the chip-internal discharge resistor permanently connected.
1	**RDCHG_EXT_PERM**	0: Off (default) 1: On	Activates auxiliary port PCAUX permanently for a) permanently discharge or b) to add an offset capacitance to every charge/discharge cycle Note: **AUX_PD_DIS** (PCAUX pull down disable) : 1
0	**RCHG_SEL**	0: 10 kΩ (default) 1: 180 kΩ	Choice of one out of 2 on-chip charging resistors for the CDC, permitting to limit the charging current, avoiding transients

6.3.5 Configuration Register 4

Table 17: Configuration Register 4

Address 0x04			
Bits	**Bit Name**	**Settings**	**Bit Description**
7	**C_REF_INT**	0: External reference at PC0/GND or PC0/PC1) 1: Internal reference	Use on-chip reference capacitor at CDC special ports PC6
5	**C_COMP_EXT**	0: Idle 1: Active; must be avoided when **C_FLOATING**==0	Activate the compensation mechanism for off-chip parasitic capacitances
4	**C_COMP_INT**	0: Idle 1: Active	Activate the compensation mechanism for on-chip parasitic capacitances and gain compensation
1	**C_DIFFERENTIAL**	0: Ordinary 1: Differential	Select between single or differential sensors
0	**C_FLOATING**	0: Grounded 1: Floating	Select between grounded or floating sensors

6.3.6 Configuration Register 5

Table 18: Configuration Register 5

Address 0x05			
Bits	**Bit Name**	**Settings**	**Bit Description**
7	**CY_PRE_MR1_SHORT**	0: Normal (recommended) 1: Reduced	Reduce delays between internal clock paths
5	**C_PORT_PAT**	0: Normal 1: Alternating order of ports	The order of the measured ports will be reversed after each sequence. If **C_ PORT_PAT** is activated then **C_AVRG + C_ FAKE** should be an even number
3	**CY_HFCLK_SEL**[2]	0: OLF 1: OHF	Clock source for the CDC
2	**CY_DIV4_DIS**	0: Off 1: On	Quadruple the clock period (only in combination with **CY_HFCLK_ SEL** == 1)
1	**CY_PRE_LONG**	0: Off, recommended 1: On	Adds safety delay between internal clock paths
0	**C_DC_BALANCE**	0: Off ("single HiZ) 1: DC free ("both HiZ)	Only for differential floating mode (other modes are DC free), changes port control to eliminate DC at Capacity sense

[2] CY_HFCLK_SEL and CY_DIV4_DIS are combined in the evaluation software as 'Cycle Clock Select'

6.3.7 Configuration Register 6

Table 19: Configuration Register 6

Address 0x06			
Bits	**Bit Name**	**Settings**	**Bit Description**
5:0	**C_PORT_EN**	0x00: All off, the CDC will not work 0x01: Only port PC0 is activated etc. 0x3F: All ports activated	Enables bitwise the CDC ports from PC0 to PC5, bit #0 for port PC0, #1 for PC1 etc.

6.3.8 Configuration Registers 7, 8

Table 20: Configuration Register 7, 8

Address 0x07, 0x08			
Bits	**Bit Name**	**Settings**	**Bit Description**
12:0	**C_AVRG**	0, 1: Sample size = 1 2: Sample size = 2 3: Sample size = 3 ... 8191 : Maximum sample size	Sample size for averaging (calculating the mean value) over CDC measurements

6.3.9 Configuration Registers 9, 10, 11

Table 21: Configuration Registers 9, 10, 11

Address 0x09 to 0x0B			
Bits	**Bit Name**	**Settings**	**Bit Description**
22:0	**CONV_TIME**	Concerning CDC, a particular period for triggering the measurements $T_{conv./seq}$ = 2 * **CONV_ TIME**[..] / f_{OLF}	Conversion trigger period or: sequence period (in stretched mode)

6.3.10 Configuration Register 12

Table 22: Configuration Register 12

Address 0x0C			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:0 & Reg.13 : 1:0	**DISCHARGE_TIME**	OLF: $T_{discharge}$ = (**DISCHARGE_TIME** + 1) * $T_{cycleclock}$ OHF: $T_{discharge}$ = (**DISCHARGE_TIME** + 0) * $T_{cycleclock}$ 1023: Off	Sets CDC discharge time Tdischarge. Time interval reserved for discharge time measurement.

6.3.11 Configuration Register 13

Table 23: Configuration Register 13

Address 0x0D			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:6	**C_STARTONPIN**	0: PG0, 1 : PG1, 2: PG2, 3 : PG3	Selection of the GPIO port that permits triggering a CDC start
4:2	**C_TRIG_SEL**	0: Continuous 1: Read triggered 2 : Timer triggered 3 : Timer triggered (stretched) 4 : n.d. 5: Pin triggered 6: Opcode triggered (7 : continuous_exp, not recommended)	CDC Trigger Mode
1:0 & Reg12: 7:0	**DISCHARGE_TIME**	See register 12	See register 12

6.3.12 Configuration Register 14

Table 24: Configuration Register 14

Address 0x0E			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:0 & Reg.15 : 1:0	**PRECHARGE_TIME**	OLF: $T_{precharge}$ = (**PRECHARGE_TIME** + 1) * $T_{cycleclock}$ OHF & **FULLCHARGE _TIME** = 1023: $T_{precharge}$ = (**PRECHARGE_TIME** + 2) * $T_{cycleclock}$ OLF & **FULLCHARGE _TIME**= 0x3FF: $T_{precharge}$ = (**PRECHARGE_TIME** + 1) * $T_{cycleclock}$ 1023: Off	Sets CDC discharge time $T_{precharge}$. Time interval reserved for discharge time measurement.

6.3.13 Configuration Register 15

Table 25: Configuration Register 15

Address 0x0F			
Bits	**Bit Name**	**Settings**	**Bit Description**
5:2	**C_FAKE**	0: None 1: 1 fake … 15: 15 fakes	Number of ”fake“ or ”warm-up” measurements for the CDC, performed just before the “real” ones; the “fake” values do not count
1:0 & Reg14: 7:0	**PRECHARGE_TIME**	See register 14	See register 14

6.3.14 Configuration Register 16

Table 26: Configuration Register 16

Address 0x10			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:0 & Reg.17 : 1:0	**FULLCHARGE_TIME**	OLF: $T_{fullcharge}$ = (**FULLCHARGE_TIME** + 1) * $T_{cycleclock}$ OHF: $T_{fullcharge}$ = (**FULLCHARGE_TIME** + 2) * $T_{cycleclock}$	Sets CDC discharge time $t_{fullcharge}$. Time interval reserved for discharge time measurement.

6.3.15 Configuration Register 17

Table 27: Configuration Register 17

Address 0x11			
Bits	**Bit Name**	**Settings**	**Bit Description**
6:2	**C_REF_SEL**	0: Minimum 1: Approx.1pF ... 31 : Maximum (approx. 31pF)	Setting the on-chip reference capacitor for the CDC Note: Step width varies. C_{ref} ~ 0.959 pF*C_REF_SEL + 3.23 pF
1:0 & Reg.16 : 7:0	**FULLCHARGE_TIME**	See register 16	See register 16

6.3.16 Configuration Register 18

Table 28: Configuration Register 18

Address 0x12			
Bits	**Bit Name**	**Settings**	**Bit Description**
7	**C_G_OP_RUN**	0: Permanent 1: Pulsed (set OP to sleep mode between conversions)	Guard: OP Mode
6	**C_G_OP_EXT**	0: Internal OP 1: External OP, PG3 as C_G_MUX_SEL	Guard: Activate external OP
5:0	**C_G_EN**	b'xxxxx1 : Activates port PC0 b'xxxx1x : Activates port PC1 b'xxx1xx : Activates port PC2 b'xx1xxx : Activates port PC3 b'x1xxxx : Activates port PC4 b'1xxxxx : Activates port PC5	Guard Enable, for each port

6.3.17 Configuration Register 19

Table 29: Configuration Register 19

Address 0x13			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:6	**C_G_OP_VU**	0: x 1.00 1: x 1.01 2: x 1.02 3: x 1.03	Guard: OP gain (from Sense Port to Guard)
5:4	**C_G_OP_ATTN**	0: 0.5 aF 1: 1.0 aF 2: 1.5 aF 3: 2.0 aF	Guard: OP attenuation
3:0	**C_G_TIME**	t : **C_G_TIME** * 500ns	Guard: Time during Precharge to switch Guard Port from "direct connected" to OP

6.3.18 Configuration Register 20

Table 30: Configuration Register 20

Address 0x14					
Bits	**Bit Name**	**Settings**			**Bit Description**
7	**R_CY**	OLF f	R_CY=0	R_CY=1	Cycle-time for the RDC Precharge/Charge/ Discharge, depending on OLF frequency
		10 kHz	100 µs	200 µs	
		50 kHz	20 µs	40 µs	
		100 kHz	10 µs	20 µs	
		200 kHz	20 µs	40 µs	
2:0	**C_G_OP_TR**	0: … 7: Recommended			Guard OP current trim

6.3.19 Configuration Register 21

Table 31: Configuration Register 21

Address 0x15			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:0 & Reg22: 1:0	**R_TRIG_PREDIV**	0, 1: Every signal triggers 2: Every 2nd signal triggers 3: Every 3rd signal triggers ... 1023: Maximum factor	Pre-divider, permits to make less temperature measurements than capacitance measurements. This is a factor between measurement rates of CDC over RDC. It is used also as OLF clock divider if OLF is used as trigger source.

6.3.20 Configuration Register 22

Table 32: Configuration Register 22

Address 0x16			
Bits	**Bit Name**	**Settings**	**Bit Description**
6:4	**R_TRIG_SEL**	0: Off 1: Timer triggered 3 : Pin triggered 5: CDC asynchronous (recommended) 6: CDC synchronous	Trigger source selection for the RDC 5 & 6: triggered by the end of CDC conversion Note: Trigger by opcode works at any time, independent form the setting here
3:2	**R_AVRG**	0: Not averaged 1: 4-fold averaged 2 : 8-fold averaged 3 : 16-fold averaged	Sample size for the mean value calculation (averaging) in the RDC part
1:0 & Reg21: 7:0	**R_TRIG_PREDIV**	0, 1: Every signal trigger 2: Every 2nd signal triggers 3 : Every 3rd signal triggers ... 1023: Maximum factor	Pre-divider, permits to make less temperature measurements than capacitance measurements. This is a factor between measurement rates of CDC over RDC. It is used also as OLF clock divider if OLF is used as trigger source.

6.3.21 Configuration Register 23

Table 33: Configuration Register 23

Address 0x17			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:6	**R_PORT_EN**	‘bx0: Disabled ‘bx1: Activates port PT0REF ‘b0x : Disabled ‘b1x: Activates port PT1	Port activation for the RDC part
5	**R_PORT_EN_IMES**	0: Disabled 1: Enabled	Port activation for internal aluminum temperature sensor
4	**R_PORT_EN_IREF**	0: Disabled	Port activation for internal reference resistor

		1: Enabled	
3	**-**		-
2	**R_FAKE**	0: 2 fake cycles per average value 1: 8 fake cycle per average value	Number of ”fake“ or ”warm-up” measurements for the RDC, performed just before the “real” ones; the “fake” values do not count
1:0	**R_STARTONPIN**	0: PG0 1: PG1 2: PG2 3: PG3	Selection of the GPIO port that permits triggering a RDC start

6.3.22 Configuration Register 24

Table 34: Configuration Register 24

Address 0x18			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:6	**-**	-	-
5:4	**TDC_CHAN_EN**	Mandatory : 3	ScioSense internal bits
3	**TDC_ALUPERMOPEN**	Mandatory : 0	ScioSense internal bits
2	**TDC_NOISE_DIS**	Mandatory : 0	ScioSense internal bits
1:0	**TDC_MUPU_SPEED**	Mandatory : 3	ScioSense

6.3.23 Configuration Register 25

Table 35: Configuration Register 25

Address 0x19			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:2	**TDC_MUPU_NO**	Mandatory : 1	ScioSense internal bits
1:0	**-**	-	-

6.3.24 Configuration Register 26

Table 36: Configuration Register 26

Address 0x1A			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:2	**TDC_QHA_SEL**	Mandatory : 20	ScioSense internal bits
1	**TDC_NOISE_CY_DIS**	Mandatory : 1	ScioSense internal bits
0	-	-	-

6.3.25 Configuration Register 27

Table 37: Configuration Register 27

Address 0x1B			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:6	**DSP_MOFLO_EN**	0: Off 3: On	Enable the mono-flop (anti-bouncing filter)" in the GPIO pulse line.
5:4	**-**	-	-
3:2	**DSP_SPEED**	0: Fastest 1: Fast 2: Slow, recommended 3 : Slowest	DSP speed
1	**PG1xPG3**	0 : Pulse output at PG3 1 : Pulse output at PG1	Switch PG1/PG3 wiring to/from DSP
0	**PG0xPG2**	0 : Pulse output at PG2 1 : Pulse output at PG0	Switch PG0/PG2 wiring to/from DSP

6.3.26 Configuration Register 28

Table 38: Configuration Register 28

Address 0x1C			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:0	**WD_DIS**	0x5A : Watchdog disabled (off) 0x00 (recommended) / else : Watchdog enabled	Watchdog Disable, to disable Watchdog 0x5A has to be written to this register. The watchdog period is between 9s and 15s

6.3.27 Configuration Register 29

Table 39: Configuration Register 29

Address 0x1D			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:4	**DSP_STARTONPIN**	Bitwise PG0 to PG3	Pin mask for starting the DSP - This mask permits assigning one or more GPIO pins to start the DSP
3:0	**DSP_FF_IN**	Bitwise DSP_IN_0 to DSP_ IN_3	Pin mask for flip-flop activation

6.3.28 Configuration Register 30

Table 40: Configuration Register 30

Address 0x1E			
Bits	**Bit Name**	**Settings**	**Bit Description**
7	**PG5_INTN_EN**	0: PG5 normal operation 1: PG5 <== INTN	Route INTN Signal to PG5

6	**PG5_INTN_EN**	0: PG4 normal operation 1: PG4 <== INTN	Route INTN Signal to PG4
5:3	**-**	-	-
2:0	**DSP_START_EN**	'bxxx1 : Trigger by end of CDC 'bxx1x : Trigger by end of RDC (recommended) 'bx1xx : Trigger by timer	DSP Trigger Enable

6.3.29 Configuration Register 31

Table 41: Configuration Register 31

Address 0x1F			
Bits	**Bit Name**	**Settings**	**Bit Description**
7	**PI1_TOGGLE_EN**	0: Normal operation 1: Toggle flip flop active	Activates toggle flip flop at Pulse Interface 1 Output especially for PDM to create 1:1 duty factor
6	**PI0_TOGGLE_EN**	0: Normal operation 1: Toggle flip flop active	Activates toggle flip flop at Pulse Interface 0 Output especially for PDM to create 1:1 duty factor
5:4	**PI0_RES**	0: 10 bit 1: 12 bit 2: 14 bit 3: 16 bit	Resolution of the pulse-code interfaces
3	**PI0_PDM_SEL**	0: PWM 1: PDM	Pulse Interface 0 PWM / PDM switch
2:0	**PI0_CLK_SEL**	0: Off 1: OLF / 1 2: OLF / 2 3: OLF / 4 4: OX / 1 5: OX / 2 6: OX / 4 7: n.d.	Pulse Interface 0 Clock Select

6.3.30 Configuration Register 32

Table 42: Configuration Register 32

Address 0x20			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:6	**-**	-	-
5:4	**PI1_RES**	0: 10 bit 1: 12 bit 2: 14 bit 3: 16 bit	Resolution of the pulse-code interfaces
3	**PI1_PDM_SEL**	0: PWM 1: PDM	Pulse Interface 1 PWM / PDM switch
2:0	**PI1_CLK_SEL**	0: Off 1: OLF / 1 2: OLF / 2 3: OLF / 4 4: OX / 1 5: OX / 2 6: OX / 4 7: n.d.	Pulse Interface 1 Clock Select

6.3.31 Configuration Register 33

Table 43: Configuration Register 33

Address 0x21			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:$	**PG_DIR_IN**	0 : Output 1 : Input	Toggles general-purpose port direction between input and output #4: PG0 #5: PG1 #6: PG2 #7: PG3
3:0	**PG_PU**	0 : Pull-up disabled 1 : Pull-up active	Activates protective pull-up resistors at general-purpose ports #0: PG0 #1: PG1 #2: PG2 #3: PG3

6.3.32 Configuration Register 34

Table 44: Configuration Register 34

Address 0x22			
Bits	**Bit Name**	**Settings**	**Bit Description**
7	**INT_TRIG_BG**	0: Disabled 1: Enabled	End of Reading triggers Bandgap
6	**DSP_TRIG_BG**	0: Disabled 1: Enabled	Bandgap refresh is triggered by the DSP bit setting
5	**BG_PERM**	1 : Bandgap permanent enabled 0 : Bandgap pulsed	Activate Bandgap permanently. With **BG_PERM** = 1 the current consumption rises by approx. 20 µA
4	**AUTOSTART**	0: Disabled 1: CDC trigger after Power On	For standalone operation, triggers CDC after Power On
3:0	**-**	Mandatory : 7	ScioSense internal bit

6.3.33 Configuration Register 35

Table 45: Configuration Register 35

Address 0x23			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:0	**CDC_GAIN_CORR[7:0]**	Recommended 1.25 ==> 0x40	Firmware defined configuration of the gain correction factor. Bits 0 to 7 of 8fpp 0 : 0 n : 1 + n/256

6.3.34 Configuration Register 36

Table 46: Configuration Register 36

Address 0x24			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:0	-	-	Not used

6.3.35 Configuration Register 37

Table 47: Configuration Register 37

Address 0x25			
Bits	**Bit Name**	**Settings**	**Bit Description**

7:0	-	-	Not used

6.3.36 Configuration Register 38

Table 48: Configuration Register 38

Address 0x26			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:0	**BG_TIME**	0: Recommended	Firmware defined

6.3.37 Configuration Register 39

Table 49: Configuration Register 39

Address 0x27			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:4	**PULSE_SEL1**	0 to 5: Res0 to Res5 (C0..5/Cref @PCap04_standard) 6: Res6 (PT1/Ref @PCap04_ standard) 7: Res7 (Alu/Ref @PCap04_ standard)	Firmware defined, select source for Pulse IF 1
3:0	**PULSE_SEL0**	0 to 5: Res0 to Res5 (C0..5/Cref @PCap04_standard) 6: Res6 (PT1/Ref @PCap04_ standard) 7: Res7 (Alu/Ref @PCap04_ standard)	Firmware defined, select source for Pulse IF 0

6.3.38 Configuration Register 40

Table 50: Configuration Register 40

Address 0x28			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:0	**C_SENSE_SEL**	PCap04_linearize firmware only, select C ratio for linearization 0..5 : C0 to 5 / Cref	Firmware defined

6.3.39 Configuration Register 41

Table 51: Configuration Register 41

Address 0x29			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:0	**R_SENSE_SEL**	PCap04_linearize firmware only, select R ratio for temperature determination	Firmware defined

6.3.40 Configuration Register 42

Table 52: Configuration Register 42

Address 0x2A			
Bits	**Bit Name**	**Settings**	**Bit Description**
7	**ALARM1_SELECT**	PCap04_linearize firmware only: Select 0: Z 1: Theta	Firmware defined. Select source of alarm signals at PG0 and PG1 Active High
6	**ALARM1_POLARITY**		
5	**ALARM0_SELECT**		
4	**ALARM0_POLARITY**		
3	**EN_ASYNC_READ**	1 : Active	Values in result registers Res0 to Res7 are only updated if the previous value has been read
2	**HS_ MODE_SEL**	0 : Mandatory	ScioSense internal bot
1	**R_ MEDIAN_EN**	PCap04_linearize firmware only	Enable median filters for ci/ri in linearize firmware
0	**C_ MEDIAN_EN**		

6.3.41 Configuration Register 47

Table 53: Configuration Register 47

Address 0x2F			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:1	-	-	Not used
0	**RUNBIT**	0: Off = the chip system is idle and protected 1: On = the protection is removed, and the system may run	On/off switch for front-end and DSP: It should be ”off“ during programming and any registry modification, thus protecting the chip from any undesirable/unspecified states

6.3.42 Configuration Register 48

Table 54: Configuration Register 48

Address 0x30			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:4	-	-	Not used
3:0	**MEM_LOCK**	'bxxx1 : NVRAM range `d0 to 703 'bxx1x : NVRAM range `d704 to 831 'bx1xx : NVRAM range `d832 to 959 'b1xxx : NVRAM range `d960 to 1007 and NVRAM range `d1022 to 1023	Data secure function to safe parts of NVRAM from reading and writing via SIF

6.3.43 Configuration Register 49

Table 55: Configuration Register 49

Address 0x31			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:0	**SERIAL_NUMBER[7:0]**	Free disposal for customer. Could only be written as far as the byte is zero. Afterwards it could just be cleared by an complete Erase	Free disposal for customer. Could only be written as far as the byte is zero. Afterwards it could just be cleared by an complete Erase

6.3.44 Configuration Register 50

Table 56: Configuration Register 50

Address 0x32			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:0	**SERIAL_NUMBER[15:8]**	Free disposal for customer. Could only be written as far as the byte is zero. Afterwards it could just be cleared by an complete Erase	Free disposal for customer. Could only be written as far as the byte is zero. Afterwards it could just be cleared by an complete Erase

6.3.45 Configuration Register 51 to 53

Table 57: Configuration Register 51 to 53

Address 0x33 to 0x35			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:0	-	0: mandatory	ScioSense internal use

6.3.46 Configuration Register 54

Table 58: Configuration Register 54

Address 0x36			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:0	**MEM**_CTRL	0x2d : NVRAM store enable 0x59 : NVRAM recall enable 0xb8 : NVRAM erase Register is reset automatically after following SIF activity	Memory control

6.3.47 Configuration Register 55 to 61

Table 59: Configuration Register 55 to 61

Address 0x37 to 0x3D			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:0	-	0: mandatory	ScioSense internal use

6.3.48 Configuration Register 62

Table 60: Configuration Register 62

Address 0x3E			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:0	**CHARGE_PUMP[7:0]**	Individual, device-specific setting. Not allowed to be changed	Lower byte of NVRAM charge pump trim

6.3.49 Configuration Register 63

Table 61: Configuration Register 63

Address 0x3F			
Bits	**Bit Name**	**Settings**	**Bit Description**
7:0	**CHARGE**_PUMP[15:8]	Individual, device-specific setting. Not allowed to be changed	Higher byte of NVRAM charge pump trim

Important Note: We guarantee the data for data retention and endurance only under the assumption, that the customer does not change the registers 62 and 63. In addition, it is mandatory to follow the given procedure for ERASE NVRAM as described in section NVRAM and ROM precisely. Otherwise, **we do no longer guarantee** the data retention time and endurance cycles.

6.4 Detailed Read Register Description

PCap04 has 35 byte of RAM for read access, combined as quadruples of 4 byte. The read registers are made of 7 result registers. Addresses 32 to 34 contain the status register.

6.4.1 Result Registers

The content of the results registers depend on the firmware. The following describes the result registers as they are used by the standard firmware.

Table 62: Result Registers

Name	Length	Format	Bit Description

	Bits		Grounded C	Differential C
Res0	32	E.g. C0 = Unsigned fixed-point number: 5 bits integer 27 bits fractional Min = 0x0 = = 0.0000000 Max = 0xFFFFFFFF = 31.9999995	Ratio C0 / Cref	Ratio C1 / C0
Res1			Ratio C1 / Cref	C3 / C2
Res2			C2 / Cref	C5 / C4
Res3			C3 / Cref	
Res4			C4 / Cref	Zero
Res5			C5 / Cref	Zero
Res6			PT1/PTref	PT1/PTref
Res7			PTinternal/PTref	PTinternal/PTref

The user is free to assign any data to the results registers in his own firmware.

6.4.2 Status Registers

Table 63: Register STATUS_0

Bits	Bit Name	Bit Description
7	**POR_Flag_Wdog**	A watchdog overflow has been detected and has provoked a power-on reset. Perhaps the firmware has hung up in an unwanted endless loop or, more likely, a CDC/RDC trigger signal has been lost.
6	**POR_Flag_Config**	One or more configuration bits toggled by interferences and has provoked a power-on-reset.
5	**POR_CDC_DSP_COLL**	If a CDC sequence is triggered while DSP is still active an Initial Reset is provoked.
4	**AutoBoot busy**	
3		
2	**RDC ready**	
1	**CDC active**	Warning: Traffic on interface may enhance noise in measurement
0	**RUNBIT**	The RUNBIT from write register 47 is mirrored here

Table 64: Register STATUS_1

Bits	Bit Name	Bit Description
7:4	n.c.	Test bits
3	**RDC_Err**	Some kind of error occurred when the RDC unit was busy
2	**Mup_Err**	A particular kind of TDC error occurred when the CDC unit was busy
1	**Err_Ovfl**	An overflow error occurred when the CDC unit was busy
0	**Comb_Err**	All error bits, from here onward, disjunctively combined (using bit-or)

Table 65: Register STATUS_2

Bits	Bit Name	Port	Bit Description
7	-	-	Not used
6	**C_PortError** Internal Reference	PC internal ref	In the CDC unit, one or several ports are affected by some error like a short-circuit to ground. May also be a charge/ discharge resistivity too big, a capacitance too big, or an ill-defined precharge/fullcharge/discharge time.
5	**C_PortError5**	PC5	
4	**C_PortError4**	PC4	
3	**C_PortError3**	PC3	
2	**C_PortError2**	PC2	
1	**C_PortError1**	PC1	
0	**C_PortError0**	PC0	

7 Principles of Operation

7.1 Converter Frontend

The device uses “discharge time measurement” as a principle for measuring either capacitances (CDC unit) or resistances (RDC unit). It addresses all ports (PC...,PT...) in time multiplex, the time measurement being done by means of a high-resolution TDC (time-to-digital converter).

7.2 Capacitance-to-Digital Converter (CDC)

7.2.1 Measuring Principle

In PCap04, capacitance measurement is done by measuring discharge times of RC-networks. The measurements are ratiometric. This means that the capacitors are compared to a fixed reference or, like in differential sensors, to capacitors with change in opposite direction. Thanks to the short time intervals and special compensation methods, the ratio of discharge times is directly proportional to the ratio of capacitors. The discharge time is defined by the capacitor and the selected discharge resistor.

$$\frac{\tau_N}{\tau_{ref}} = \frac{C_N}{C_{ref}} \tau = k \times R \times C$$

7.2.1.1 Connecting Sensors

PCap04 can handle single and differential sensors in grounded or floating connection. PCap04 has integrated reference capacitors. They are programmable in a range from 1 pF to 31 pF in steps of 1pF.

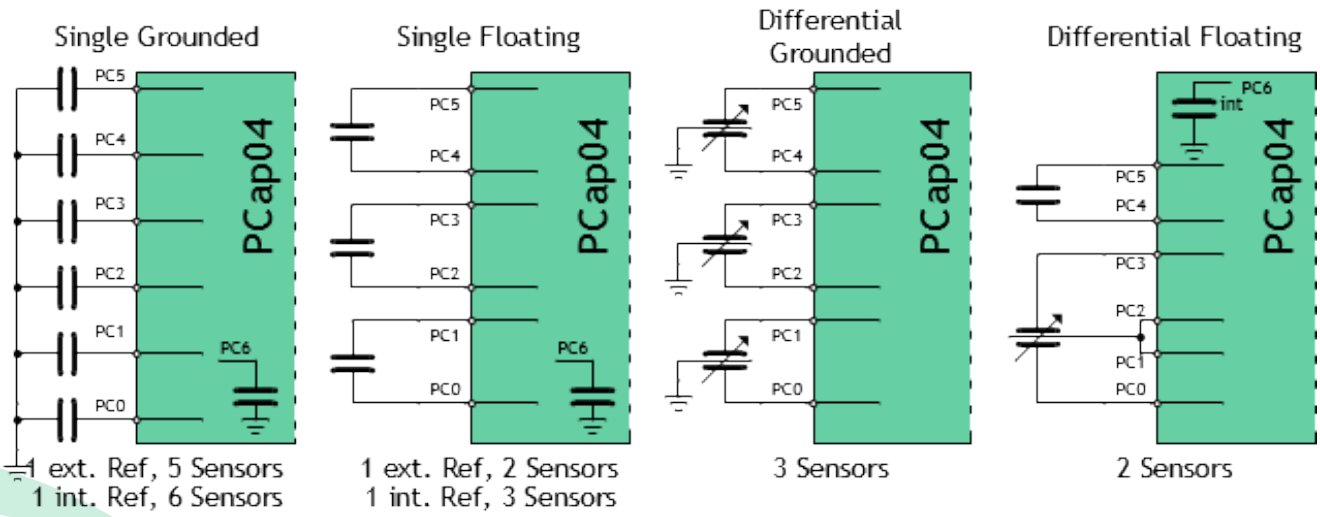


Figure 5: Connecting sensors

7.2.1.2 Discharge Resistors

The PCap04 has two sets of discharge resistors already integrated. One resistor set (10k / 30k / 90k / 180k ohm) is for measurements on port PC0 to PC3 and the internal reference port PC6. The other resistor set (10k / 30k / 90k / 180k ohm) is for ports PC4 and PC5. This

way, it is possible to measure different sensors with strongly deviated capacitance like pressure and humidity with one and the same chip. Parameters RCHG_xxx select the resistors.

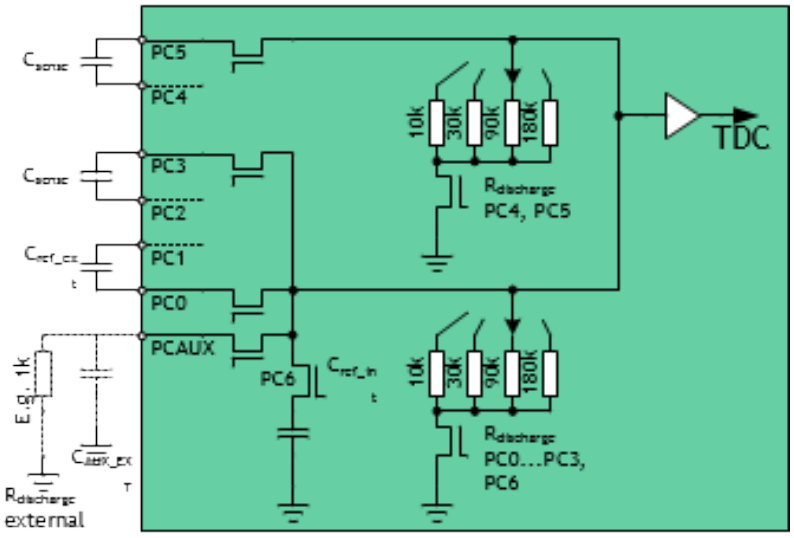


Figure 6: Integrated Discharge Resistors

The user can define which resistor he uses for the internal compensation measurement. It is selected by **DSP_RDCHG_COMP_INT_SEL** (0 = RCHG0 (default), 1 = RCHG1).

There is the possibility to use an external discharge resistor for handling big capacitances.

7.2.1.3 Cycle

In PCap04 the measurement principle is based on a three-step cycle.

- In the pre-charge phase the capacitor is charged up via a series resistor to a level close to Vdd. The resistor reduces the charge current and reduces the mechanical stress on the sensing capacitor. This can be necessary in some MEMS applications. Further, this is a measure to detect a short circuit and to limit the current even in such an error case.
- In full-charge phase, the capacitor is charged up finally to Vdd without any series resistor.
- Then, in the third step, the capacitor is discharged via the discharge resistor down to 0V. The CDC measures the time interval until a trigger level is reached. All this is called a single “cycle”.

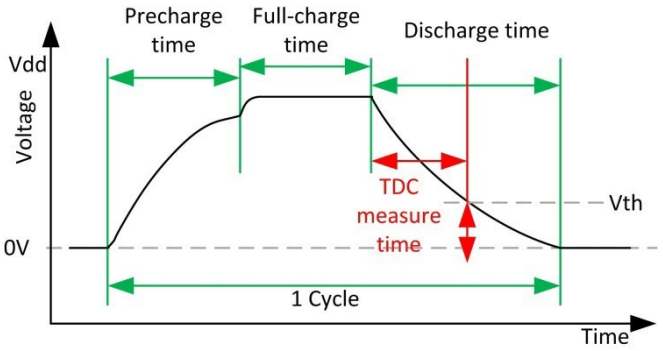


Figure 7: Single Cycle Timing

In applications that do not need the slow charge up but high conversion rate, it is possible to disable the pre-charge option and to start charge up directly without any series resistor.

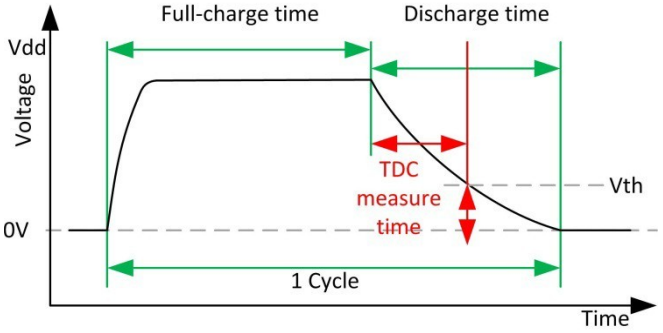


Figure 8: Single Cycle, Fast Charge

In both cases the capacitors are discharged for the full discharge time period and then connected to GND.

In case of a short circuit, the voltage never reaches the trigger level of the comparator. In such a case the measurement cycle is stopped, no low-resistive full-charge follows. This way, the current in a short-circuit case is limited.

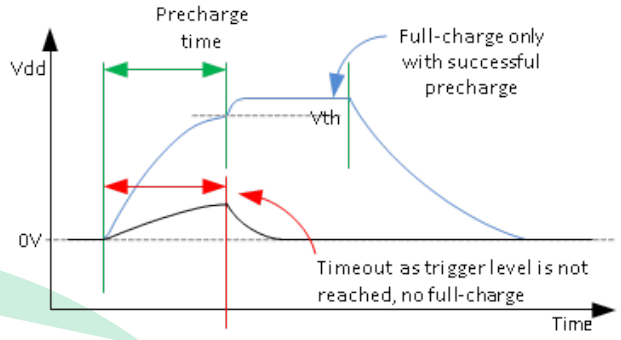


Figure 9: Short Circuit Detection

7.2.1.4 Sequence

A “sequence” is made of a set of cycles, namely those for the various active ports as well as combinations of them as given by the compensation measurements. The number and kind of single cycles depend on the way of how the sensors are connected, the number of capacitors and the selected compensation options.

For grounded sensors, the sequence starts always with PC0 (reference) and then one or more of the other 5 ports. Normally, internal compensation is activated. So the sequence ends with the measurement Cint of the internal stray capacitance/delays.

For compensating internal parasitic capacitance and the comparator delay the CDC measures the discharge time with all ports being off (Cint).

The following figure shows the sequence for a grounded sensor with internal compensation.

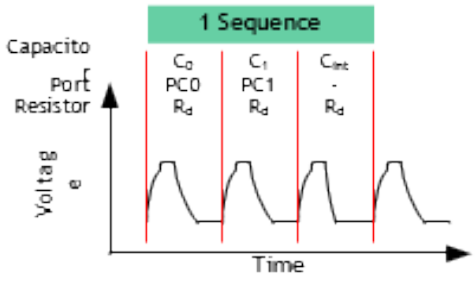


Figure 10: Sequence Grounded

For floating sensors, the sequence starts always with PC0/PC1 (reference), followed by one to three pairs of ports for the sensors. Normally, both compensations (internal and external) are activated.

For compensation of external parasitic capacitances the CDC makes a measurement for each capacitor with both ports being opened. So, for each capacitor 3 measurements are made, e.g. PC0, PC1 and PC0+PC1. The sequence ends with the internal compensation measurement Cint. The following figures show the sequence for 1 floating sensor with full compensation.

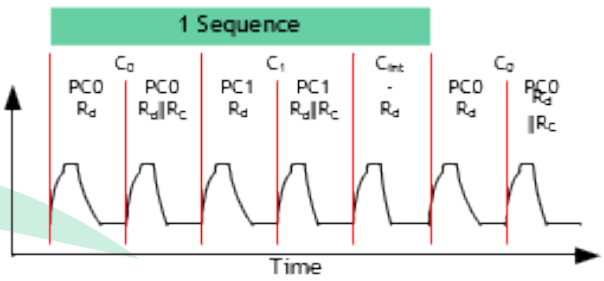


Figure 11: Sequence Floating

7.2.1.5 Conversion

Finally, the combination of various sequences and delays in between the sequences define a single “conversion”. Once triggered, a conversion is automatically completed, including all fake measurements and all real measurements defined by sample size for averaging. At the end of a conversion the measurement results are ready for further processing and readout. The end of the conversion is indicated by flag to the DSP and also the RDC unit.

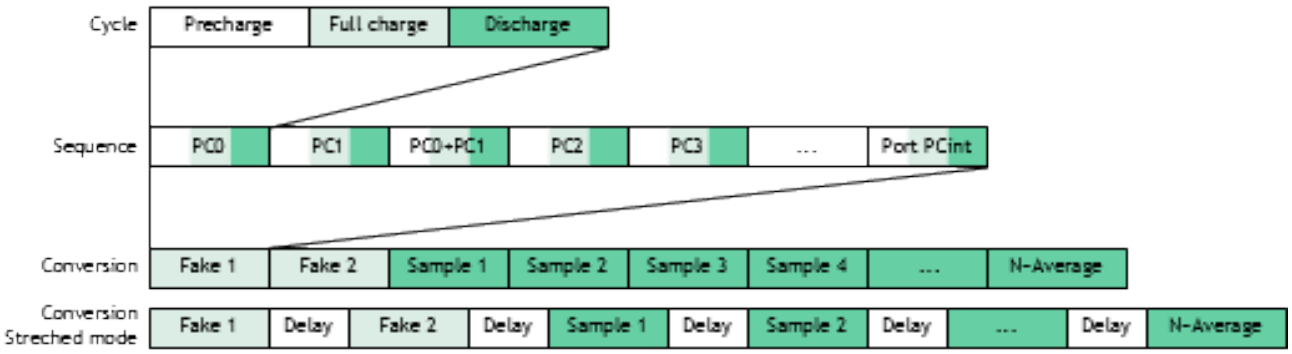


Figure 12: Cycle-Sequence-Conversion

Various sources are available to trigger the CDC.

- Continuous mode: In this mode the CDC starts automatically after POR. The end of the DSP processing triggers directly the next measurement. This mode allows the fastest sample rate, but with the risk that communication will continue while the next measurement is started. Noise will increase.
- Read triggered mode: In this mode the very first measurement is triggered via SIF. When the measurement is completed the interrupt goes LOW. An external microcontroller can react on this and read the data. The end of the read via SIF triggers the next measurement. This mode allows fastest measurement without having the interface disturb the measurement.
- Timer triggered mode: In this mode the PCAP04 timer triggers the measurements. This is preferred in applications with low sample rate.
- Opcode or pin triggered mode: In this mode the external microcontroller has full control on when measurements are triggered. This might be valuable when the measurement needs to be synchronized with other tasks.

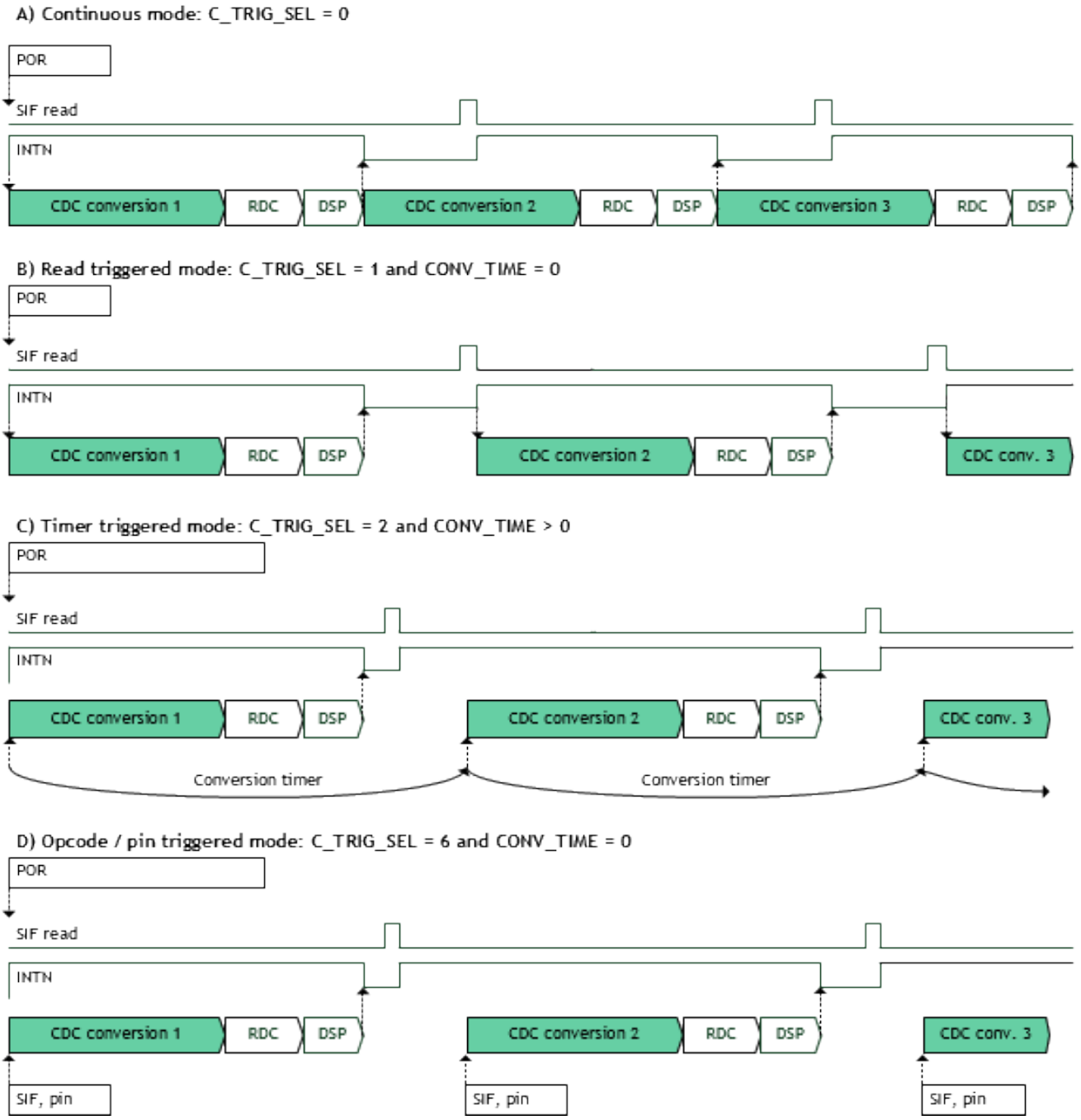


Figure 13: Conversion Modes

7.2.2 CDC Compensation Options

7.2.2.1 Internal Compensation

For the internal compensation measurement, both switches A1 and A0 are open. Only the internal parasitic capacitance and the comparator propagation delay will thus be measured.

It is recommended to have internal compensation active in any application.

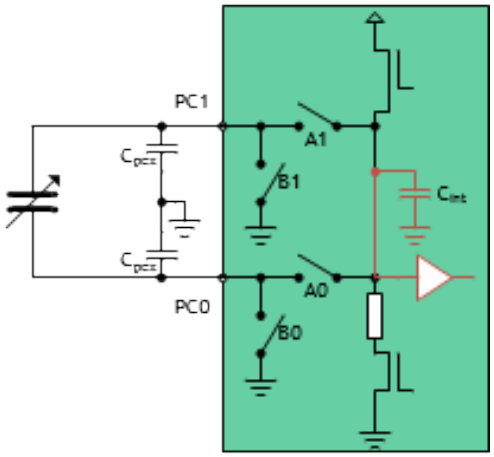


Figure 14: Internal Compensation Measurement

7.2.2.2 External Compensation

With floating capacitors we have the additional option to compensate external parasitic capacitances against ground. On the PCB, the wire capacitance typically refers to ground. For long wires, it is recommended to use shields which should be grounded at their PCB side.

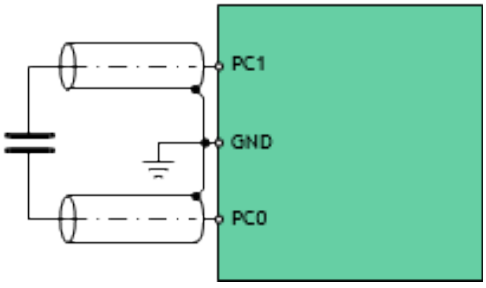


Figure 15: Shielded Cables

Three measurements are necessary for each capacitor in case of floating sensors. This is shown in the Figure 81. First, the electrode at PC0 is loaded to VDD18. The discharge time is defined by the sensor capacitance, the parasitic capacitance of the connection, including the chip pad, and the internal capacitance. Second, the same measurement is done for the electrode at PC1. Third, both electrodes are set at VDD18. Therefore the field across the sensor is zero and has no impact.

The discharge time includes only the connection and pad capacitance as well as the internal capacitance. Now it is possible to correct mathematically for the parasitic capacitance. This correction is covered by the ScioSense firmware.

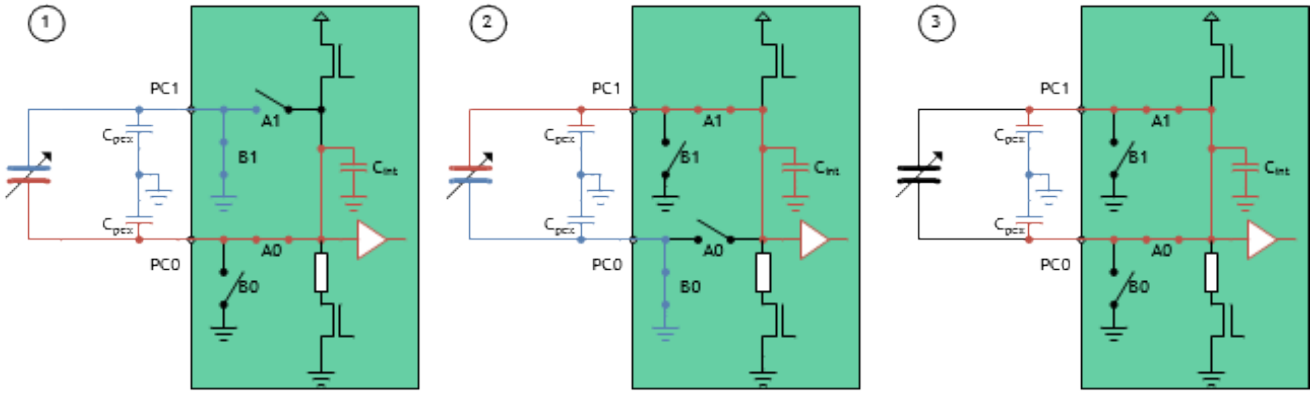


Figure 16: External Compensation

7.2.2.3 DC Balance

When driving floating sensors the sensors' supply is typically DC free.

With differential floating sensors symmetry would be broken. Therefore, PCap04 has the possibility to change the port controlling that the sensors are operated DC free (set by **C_DC_ BALANCE**).

In applications with grounded sensors the sensors cannot be DC fee by principle.

7.2.2.4 Gain Correction

Comparable to classical A/D converters, the PCap04 shows a gain error. But in case of PCap04 the gain error is mainly given by internal parasitic capacitances and the propagation delay of the internal comparator. With internal compensation being active this delay is subtracted from the original measurement.

The temperature drift can be approximated linearly and corrected mathematically just by a gain factor. In the standard firmware the gain correction factor could be set in register 35 with fpp 8. The correction factor depends on the discharge time and therefore the RC combination. It has to be evaluated individually for every single application. E.g., with 22 pF and 30 kOhm the correction factor is 1.25 (0x40).

CDC_GAIN_CORR = (gain_corr-1)*256

Empirical method to find the right gain correction factor:

Replace the sensor with a temperature stable capacitor of the same size (ceramic C0G) as your reference capacitor. (Therefore: quotient = 1, gain = 0). Set the gain correction factor to 1.0. Put the system (PCap04 on PCB) into a temperature chamber and measure the offset drift over temperature. Add an additional temperature stable capacitor to simulate your gain. Measure the gain drift. Increase the gain correction factor and measure the gain drift again. With a gain correction factor >1.0 the gain drift will decrease. If the

gain correction factor is set too big then you will see a negative gain drift due to over compensation. The right gain correction factor is found, if the drift is reduced to what you measured at the initial offset drift measurement. Write back the new **CDC_GAIN_CORR** value into register 35.

7.2.3 CDC Important Parameters

7.2.3.1 Cycle Clock

The basic period t_{cycle} that defines the cycle time can be derived from the low frequency oscillator or the high frequency oscillator. Parameters **CY_HFCLK_SEL** selects in between the two, parameter **CY_DIV4_DIS** select between the original 2 MHz or a 0.5 MHz generated by a divider by 4.

Table 66: Configuration of Cycle Clock

CY_HFCLK_SEL	CY_DIV4_DIS	Cycle Time Base
0	0	$t_{cycle} = t_{OLF}$; t_{OLF} = period low-frequency oscillator
1	0	$t_{cycle} = 4*t_{OHF}$; t_{OHF} = period high-frequency oscillator.
1	1	$t_{cycle} = t_{OHF}$; t_{OHF} = period high-frequency oscillator.

7.2.3.2 Cycle Time

The pre-charge, full-charge and discharge times of a single cycle are defined in multiples of t_{cycle}. Those are selected by:

Table 67: Configuration of Cycle Time

Reg.	Configuration Parameter	Description
14, 15	**PRECHARGE_TIME**	Time to charge via resistor for current limitation. Depends on the cycle clock and, with OHF, on the FULLCHARGE_TIME. 1023 = No pre-charge phase OLF: 0 to 1022: $T_{precharge}$ = (**PRECHARGE_TIME** + 1) * t_{cycle} OHF & FULLCHARGE_TIME = 1023: 0 to 1022: $T_{precharge}$ = (**PRECHARGE_TIME** + 2) * t_{cycle} OHF & FULLCHARGE_TIME != 1023: 0 to 1022: $T_{precharge}$ = (**PRECHARGE_TIME** + 1) * t_{cycle}
16, 17	**FULLCHARGE_TIME**	Time for final charge without current limitation. Depends on the cycle clock. 1023 = No full-charge phase OLF: 0 to 1022: $T_{fullcharge}$ = (**FULLCHARGE_TIME** + 1) * t_{cycle} OHF: 0 to 1022: $T_{fullcharge}$ = (**FULLCHARGE_TIME** + 2) * t_{cycle}
12, 13	**DISCHARGE_TIME**	Time to discharge the capacitor. Depends on the cycle clock. OLF: 0 to 1023: $T_{discharge}$ = (**DISCHARGE_TIME** + 1) * t_{cycle} OHF: 0 to 1023: $T_{discharge}$ = (**DISCHARGE_TIME** + 0) * t_{cycle}

7.2.3.3 Sequence

The length of a sequence depends on the kind and number of sensors, the selected compensation methods and the averaging sample size. The following parameters affect the sequence:

Table 68: Sequence Configuration

Reg.	Configuration Parameter	Description
6	**C_PORT_EN**	Bitwise enable of the capacitance ports PC0 to PC5 0: Port disabled 1: Port active
4	**C_REF_INT**	Switches between external and internal reference capacitors. Cannot be used with differential sensors. 0: External, PC0 or PC0 & PC1 1: Internal, PC6
4	**C_DIFFERENTIAL**	Switches between single and differential sensors 0: Single 1: Differential
4	**C_FLOATING**	Switches between grounded and floating sensors 0: Grounded 1: Floating
4	**C_COMP_INT**	Turns on compensation of internal capacitances/delays 0: Off 1: On, recommended
4	**C_COMP_EXT**	Turns on compensation of external parasitic capacitances. Available only with floating sensors. 0: Off 1: On, recommended
5	**C_DC_BALANCE**	Changes Port Control for Differential Floating Mode 0: Off (single HighZ) 1: DC-Free (both HighZ)

7.2.3.4 Conversion

The duration of a full conversion has a lower limit given by the number of fake measurements, the averaging and eventually an inter-sequence delay:

Table 69: Conversion Configuration

Reg.	Configuration Parameter	Description
15	**C_FAKE**	Number of fake measurements (cycles with results being ignored) 0: No dummy cycles 1: 1 dummy cycle … 15: 15 dummy cycles
7, 8	**C_AVRG**	Sample size for averaging within one conversion.

		0: = No averaging to 8191: Maximum sample size

The Start of the next conversion depends on the selection of the measurement trigger:

Table 70: Conversion Configuration

Reg.	Configuration Parameter	Description
13	**C_TRIG_SEL**	First trigger selection for CDC trigger 0: Continuous 1: Read triggered 2: Timer triggered 3: Timer triggered (stretched) 4 : n.d. 5: Pin triggered 6: Opcode triggered (7: continuous_exp, not recommended)
13	**C_STARTONPIN**	Selects the GPIO that triggers the CDC measurement
9, 10, 11	**CONV_TIME**	Sets the conversion time in multiples of twice the period of the low-frequency clock. t_{conv} = 2 * **CONV_TIME** * t_{ofl}

7.2.4 Guarding

Depending on the sensor topology it may be necessary to add an active shield to suppress disturbing capacitances. The guard electrode is an additional metal area behind the sensing electrode, which is kept at the same potential as the sensing electrode. So all material in between the sensing electrode and the guard electrode are potential-free and therefore have no capacitive effect. It eliminates material-depending temperature drifts of the sensor PCB. Further, things behind the shield are separated from the senor by the potential-free zone.

In PCap04 the driver for the guard is integrated. This amplifier needs to have a low-capacitive input to not disturb the measurement path. Gain is ideally = 1. The gain can be set > 1 (overcompensation) by means of C_G_OP_VU so that in combination with an external voltage divider it is possible to match the port and wire resistance. The guard is connected to pin PCAUX (other functions of this pin are then not available). In-active ports are also switched to guard so that there is no additional capacitance seen between guard and inactive ports.

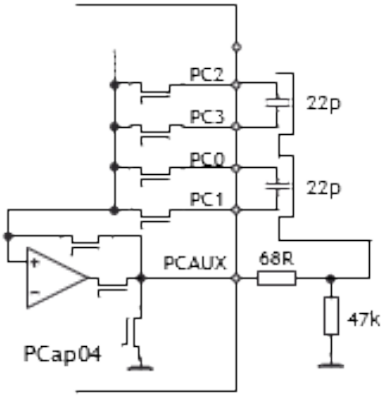


Figure 17: Guarding with Floating Capacitors

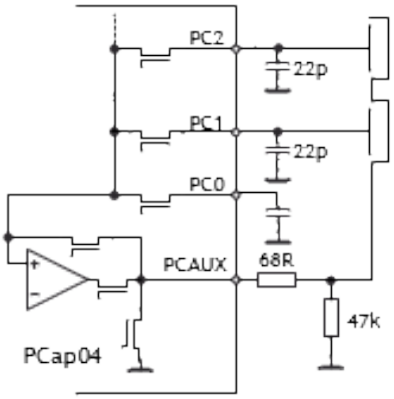


Figure 18: Guarding with Grounded Capacitors

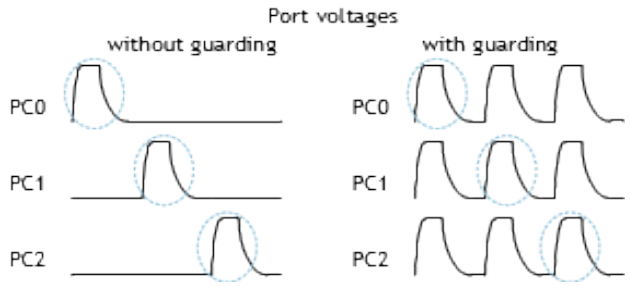


Figure 19: Port Voltages with Guarding

With guarding for multiple ports, all ports are set to VDD18 (dummy guard), even the ones that are not measured. This ensures that electric fields between those ports and the active port are zero. Internally, those ports are not connected to the time measuring path but to the guard driver.

Note: due to the dummy guard for multiple ports being activated, guarding can't be used in combination with differential floating sensors (where the ports PC1 and PC2 are shortened).

If the guard electrode is so big that it cannot be driven by the internal amplifier it is possible to add an external one. An external analog-multiplexer will be needed, too. The

external amplifier is connected to PCAUX. The SEL port of the multiplexer is connected to PG3.

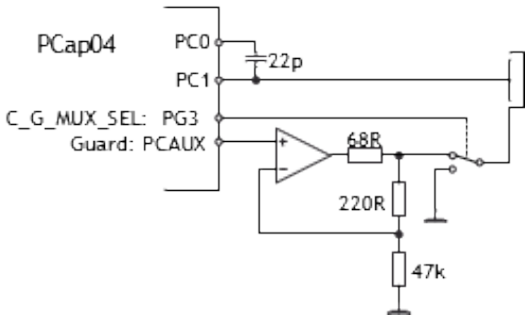


Figure 20: Guarding with External Amplifier

Table 71: Important Parameters are Set in Register 18

Reg.	Configuration Parameter	Description
18	**C_G_EN[5:0]**	Individual guard enable for each port PC0 to PC5
18	**C_G_OP_RUN**	0: Permanent: Guarding OP is permanent activated (additional power consumption) 1: Pulsed: Guarding OP set to sleep mode between CDC conversions
19	**C_G_OP_ATTN**	Capacitive attenuation of guarding OP: 0: 0.5pF 1: 1.0pF 2: 1.5pF 3: 2.0pF
19	**C_G_OP_VU**	OP Gain (from sense port to guard) 0 : 1.00 1: 1.01 2: 1.02 3: 1.03
20	**C_G_OP_TR**	Trim power consumption and driving strength of guarding OP 0: Min. … 7: Max.
18	**C_G_OP_EXT**	Switch between internal guarding OP and an optional external OP 0: Internal OP 1: External OP, PG3 as **C_G_MUX_SEL**
19	**C_G_TIME**	$t_{pp1} = t_{OHF}$ * **C_G_TIME, PRECHARGE_TIME > C_G_TIME**

- **C_G_EN**[5:0] guarding is activated individually for each port PC0 to PC5. One or more (except differential floating) ports can be enabled. The guard output (PCAUX) drives voltage the enabled ports are enabled.
- **C_G_TIME** controls the pre-charge phase. Because internal circuits the pre-charge phase is divided into two phases:

- Pre-charge phase 1: PCAUX is directly connected with active PC.
- Pre-charge phase 2: PCAUX is driven by OP, V_PCAUX = gain_guardOP * V_PCactive. PCAUX is driven by OP until finishing current port cycle.

As a consequence, the CDC pre-charge time setting, **PRECHARGE_TIME**, needs to be minimum 1 and bigger than **C_G_TIME**.

Attention: For guarding, the internal OX/OHF is mandatory:

- **OX_RUN** > 0, OX_DIS = 0; OX_STOP = 0; OX_DIV4 = 0;
- **CY_HFCLK_SEL** = 1;

7.3 RDC Resistance-to-Digital Converter

7.3.1 Measuring Principle

In PCap04 resistance measurement is also done by measuring discharge times. The measurements are ratiometric. This means, the temperature-sensitive resistances are compared with a fixed reference. The ratio of discharge times is directly proportional to the ratio of resistors. The discharge time is defined by the resistors and the load capacitance.

$$\frac{\tau_N}{\tau_{ref}} = \frac{R_N}{T_{ref}} \tau = k \times R \times C$$

7.3.1.1 Connecting Sensors

The chip device has two on-chip resistor elements for the measurement of temperature, an aluminum strip with TK ≈ 2800ppm/K as a sensor and a poly-silicon resistor with TK “close to zero” as a reference. In the range 0°C to 100°C the aluminum sensor can be well approximated by a linear function of temperature.

As an alternative, it is possible to connect up to two external sensors. One of those can be used as external reference alternately. External and internal thermometers/reference may be mixed, e.g. an external PT1000 may be compared to the internal Poly-Si resistor.

The chip has an internal capacitor of about 94 pF. In combination with the internal resistors the discharge time is about 500 ns and the typical resolution of the resistance ratio is better than 13 bits. For precision measurements we recommend to connect an external capacitor of about 10 nF. With 10 nF the discharge time is in the order of 20 µs and the resolution in the order of 15 bits.

Discharge time must not exceed 20 µs. For the capacitor, C0G ceramics yields best performance, while X7R material yields fair results.

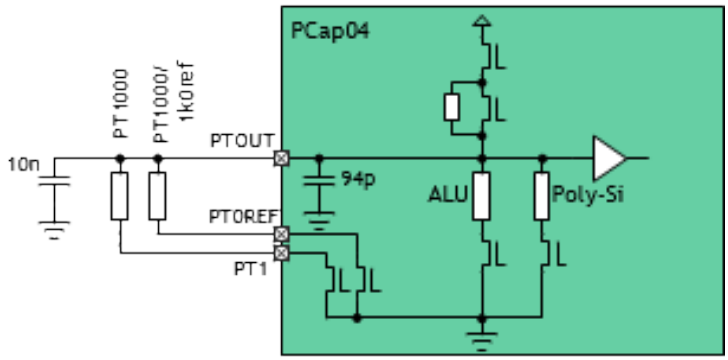


Figure 21: Connecting Temperature Sensors

Note: The RDC measurement is based on an AC principle. Long cables with their parasitic capacitance and resistance will disturb and it is recommended to have short cables (≤ 0.5 m), ideally twisted and shielded.

7.3.1.2 Cycle & Conversion

In PCap04 the resistance measurement is running in three phases, like in capacitance measurement:

Pre-charge – Full-charge – Discharge. The timing is based on the internal low-frequency oscillator (OLF). The duration of full and discharge phases can be 1 or 2 periods of this reference. The conversion starts with 2 or 8 fake measurements to improve the stability of data. For each single conversion the averaging can be selected with sample size 1, 4, 8 or 16.

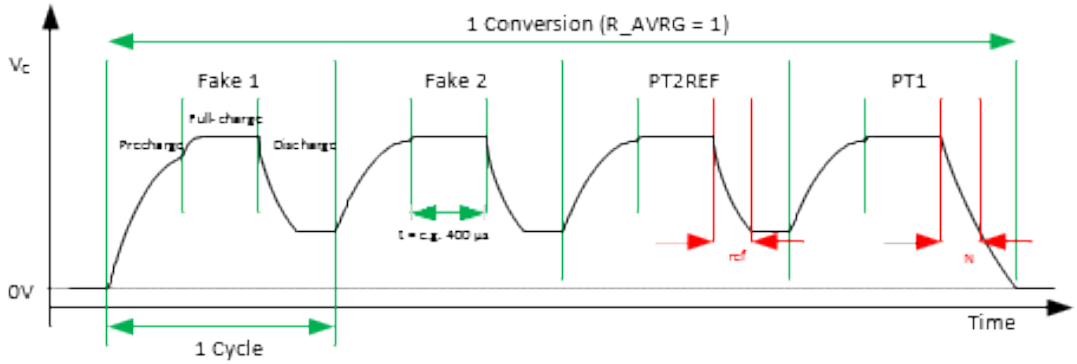


Figure 22: RDC Conversion, R_AVRG = 1, Reference and sensor, 2 fake measurements

7.3.1.3 Trigger

Various sources can trigger the RDC. The trigger rate can be set to a divider of the CDC trigger rate by means of parameter **R_ TRIG_PREDIV** (1 to 1023).

Parameter **R_TRIG_SEL** defines the various possibilities to trigger a resistance measurement:

- Serial Interface command, PIN or DSP (Figure 23)
- Timer triggered, based on the OLF (Figure 23)

- CDC end of conversion
    - Asynchronous: The DSP is triggered by the RDC end of conversion. If RDC rate is less than CDC rate the DSP is triggered directly from the CDC for inactive RDC conversions (Figure 24).
    - Synchronous: The DSP is triggered by the RDC end of conversion. Assuming that RDC rate is less than the CDC rate, the inactive RDC conversions are replaced by a delay (Figure 25)

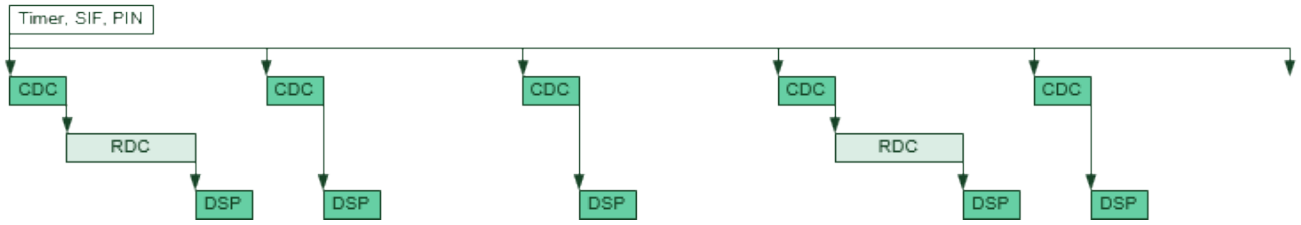
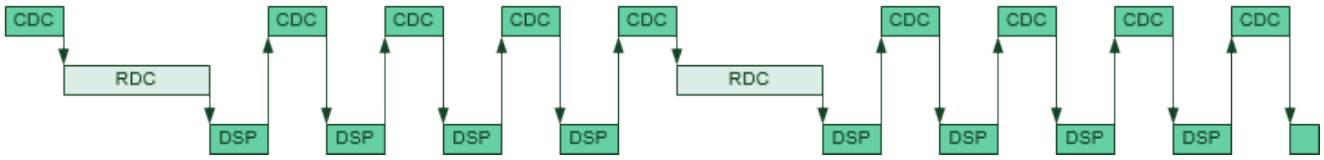
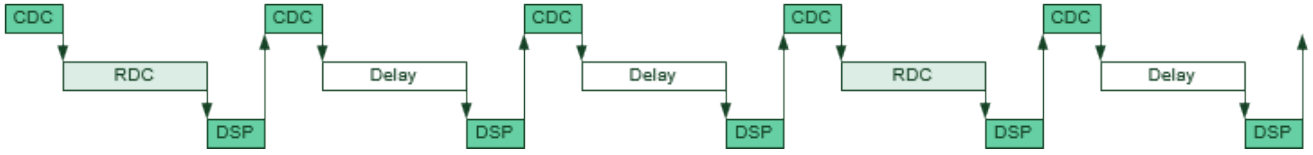


Figure 23: RDC Timing, Triggered by Timer, SIF or Pin



R_TRIG_PREDIV = 3, R_TRIG_SEL = 3'b101, DSP_START_EN: CDC_TRIG_EN = 0, RDC_TRIG_EN = 1

Figure 24: RDC Timing, Triggered by DSP in Asynchronous Mode



R_TRIG_PREDIV = 3, R_TRIG_SEL = 3'b110, DSP_START_EN: CDC_TRIG_EN = 1, RDC_TRIG_EN = 0

Figure 25: RDC Timing, Triggered by DSP in Synchronous Mode

7.3.2 RDC Important Parameters

7.3.2.1 Cycle Clock

The base frequency for the temperature measurement is the low frequency oscillator (OLF). A further bit, R_CY, specifies whether 1 or 2 periods define the length of the three phases.

Table 72: Configuration of Cycle Clock

OLF Frequency	$t_{precharge} = t_{fullcharge} = t_{discharge}$	
	R_CY = 0	R_CY = 1
10 kHz	100 µs	200 µs

50 kHz	20 µs	40 µs
100 kHz	10 µs	20 µs
200 kHz	20 µs	40 µs

7.3.2.2 Sequence

The major settings for the sequence are the number of ports, the fakes, the reference and averaging.

Table 73: Sequence Configuration

Reg.	Configuration Parameter	Description
23	**R_PORT_EN**	Enable ports PT0REF, PT1
23	**R_PORT_EN_IREF**	Enable the internal reference resistor
23	**R_PORT_EN_IMES**	Enable the internal temperature sensor
22	**R_AVRG**	Set averaging for T measurement
23	**R_FAKE**	Set number of fake measurements

7.3.2.3 Conversion

Table 74: Conversion Configuration

Reg.	Configuration Parameter	Description
22	**R_TRIG_SEL**	Selection of trigger source for RDC unit
21, 22	**R_TRIG_PREDIV**	Pre-divider to set the RDC rate as fraction of the CDC rate but also to the **OLF_CLK** when **OLF_CLK** is selected as RDC Trigger 0 = 1: RDC conversion with each CDC conversion 2: RDC conversion every second CDC conversion … 1023: Maximum setting
23	**R_STARTONPIN**	Select Pin for pin triggered

7.3.3 RDC Results, Ratios

PCap04_standard and PCap04_linearize firmware are determining ratios between sense ports and reference ports.

If the internal reference port is activated (**R_PROT_EN_IREF**: 1) for all other ports (internal sense, PT0 and PT1) the internal reference is automatically selected for ratios.

If **R_PORT_EN_IREF**: 0, external port PT0/Ref is selected as reference value for all other ports (internal sense, PT1)

7.4 Interfaces (Serial & PDM/PWM)

7.4.1 Serial Interfaces (SIF)

Two types of serial interfaces are available for communication with a microcontroller and for programming the device: SPI and I²C. Only one interface is available at a time, selected by pin **IIC_EN**. On both interfaces, the PCap04 can operate as slave only.

Table 75: Serial Interface Selection

Pin	Description
IIC_EN = GROUND	4-wire SPI interface General-purpose I/O pins PG0 and PG1 are not available
IIC_EN = VDD	2-wire I²C interface All general-purpose I/O pins are available

IIC_EN may not be floating. Connect IIC_EN to VDD if there is no need for a controller interface.

The serial interfaces allow read access to the read registers (results and status), read/write access to the configuration registers and read/write access to the NVRAM (explicitly the SRAM part of the NVRAM).

All commands for write or read to memory or configuration / read registers may use explicit addressing or address auto-increment.

Table 76: Opcodes

Description	Byte 2								Byte 1			Byte 0
wr_mem (NVRAM)	1	0	1	0	0	0	add<9...0>[1]					data<7...0>
rd_mem (NVRAM)	0	0	1	0	0	0	add<9...0>[12]					data<7...0>
Write configuration	1	0	1	0	0	0	1	1	1	1	add<5...0>[12]	data<7...0>
Read configuration	0	0	1	0	0	0	1	1	1	1	add<5...0>[12]	data<7...0>
Read result (RAM)	0	1	add<5...0>[12,2]						data<7...0>			

[1] Auto-incremental write/read is supported by both, SPI and I²C

[2] Address range for Read result is 0 to 24 (8 x 32bit result registers and 3 x 8 bit status registers)

POR (Power-on Reset)	1	0	0	0	1	0	0	0								
Initialize	1	0	0	0	1	0	1	0								
CDC Start conversion	1	0	0	0	1	1	0	0								
RDC Start conversion	1	0	0	0	1	1	1	0								
dsp_trig	1	0	0	0	1	1	0	1								
nv_store[3]	1	0	0	1	0	1	1	0								
nv_recall[14]	1	0	0	1	1	0	0	1								
nv_erase[14]	1	0	0	1	1	1	0	0								
Test read	0	1	1	1	1	1	1	0	0	0	0	1	0	0	0	1

The serial interface is tested most easily by performing a test read:

Write opcode 0x7e to SIF and read 1 byte. Compare this byte to following patterns:

- 0x11: Expected value, read cycle performed correctly
- 0x88: Failure: there is a big/little-endian swap
- 0xEE: Failure: during read cycle all bits are inverted
- 0x77: Failure: inverted bits and bit/little-endian swap

7.4.1.1 System Reset

In case the PCap04 is operated as a slave, not in self-boot mode, it is necessary to do the following actions after applying power:

- Send opcode Initialize via the serial interface, opcode 0x8A.
- Write the firmware into the SRAM by means of opcode “Write to SRAM“.
- Write the configuration registers by means of opcode “Write Config”. Register 47 with the RUNBIT has to be the last one in order.
- Send a start command, opcode 0x8C

There is a difference between a hardware power-on and sending the POR opcode. During a hardware power-on the chip runs with the bandgap in continuous mode. Therefore VDD18 is generated immediately and the chip is operational after the specified 4 ms. In case of a software POR it depends on the bandgap configuration in advance. It the bandgap was running in pulsed mode then it takes the period of the bandgap refresh until VDD188 will be generated and the chip is operative. This may take in worst case seconds. In most situations it will be fully sufficient to send a Initialize opcode instead of a POR. But as an

[3] Set MEM_CTRL before using nv_store, nv_recall or nv_erase

alternative, the user can send first an opcode to change the bandgap setting to permanent and then send the POR opcode.

7.4.1.2 Synchronous Read

For best results it is recommended to read all values from the result registers synchronized by the INTN signal. The INTN signal can be routed to PG4 or PG5 by **PG4_INTN_EN** and **PG5_INTN_ EN** at Register 30. The INTN signal is low active, which means

the negative edge of INTN signals new values are available at Res0 to Res7. The INTN is set back to High by a positive edge at SSN (SPI) or a stop condition (I^2C).

7.4.1.3 Asynchronous Read

If it is not possible to read synchronously as described above for any reason, asynchronous read (**EN_ASYNC_RD**) has to be enabled in register 42. In this mode values in result registers Res0 to Res7 are only updated if the previous value has been read (INTN is reset to High by a positive edge of SSN or a stop condition).

7.4.2 I^2C Compatible Interface

The present paragraph outlines the PCap04 device specific use of the I^2C interface. The external I^2C master begins the communication by creating a start condition, a falling edge on the SDA line while SCL is HIGH. It stops the communication by a stop condition, a rising edge on the SDA line while SCK is high. Data bits are transferred with the rising edge of SCK.

On I^2C buses, every slave holds an individual 7-bit device address[4] with 5 fixed and 2 configurable bits. This address has always to be sent as the first byte after the start condition, the eighth bit indicating the direction of the following data transfer (R=read=1 and W=write=0).

Table 77: Address Byte

MSB							LSB
0	1	0	1	0	A1	A0	R/W
Fixed					Variable		key

[4] In the former silicon version V1 there was a bug. PCAP04 responded to any address. See error sheet PCap04_ES000131_1-01. This bug is removed in silicon revision V2 as described here.

7.4.2.1 I²C Timing

The address byte is followed by the opcode and eventually the payload. Each byte is followed by an acknowledge bit (= 0, when a slave acknowledges).

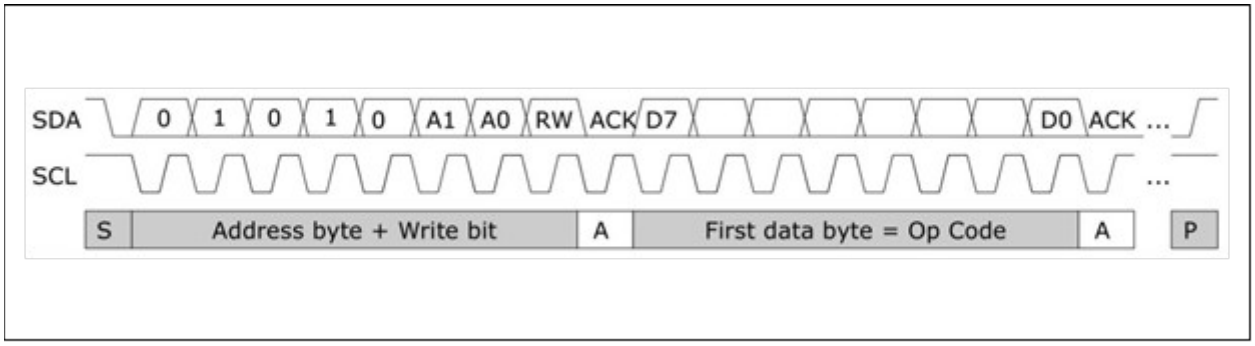
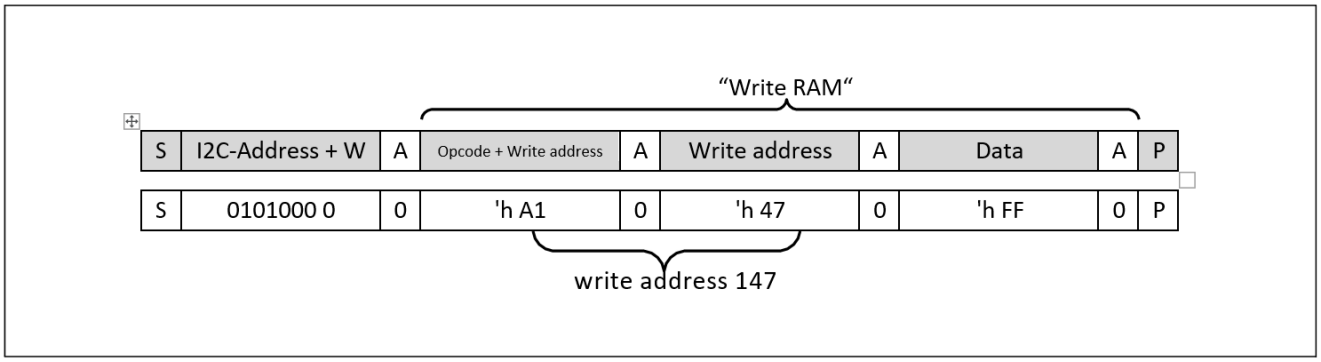


Figure 26: I²C Typical Sequence

7.4.2.2 I²C Write

During write transactions, the master alone sends data, the addressed slave just sends the acknowledge bits. The master first sends the slave address plus the write bit. Then it sends the PCap04 specific opcode including the register address in the slave. Finally it sends the payload ("Data").

Incremental writing is possible, means, for a consecutive set of data only the start address has to be sent and a various number of data could be sent in one row.

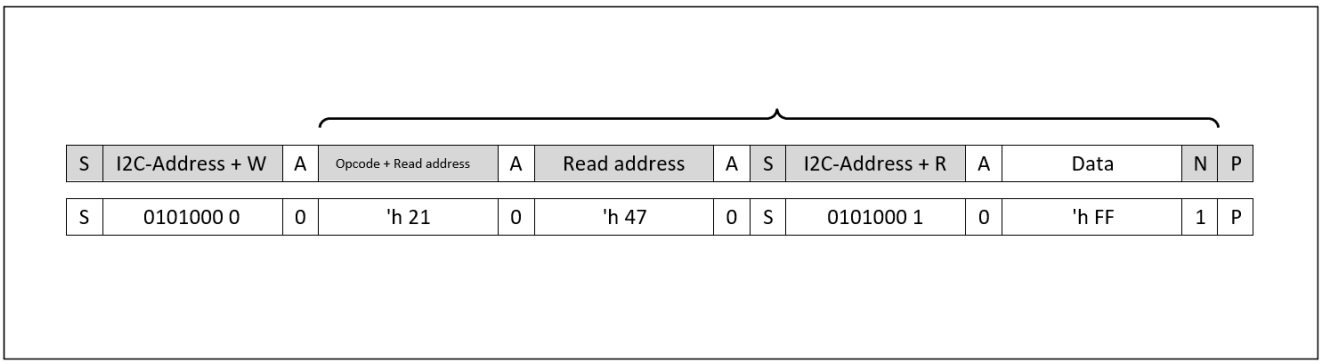


Example: Write 0xFF as a datum to the SRAM at address 0x147.

Figure 27: I²C Write Procedure

7.4.2.3 I²C Read

During read transactions, the direction of communication has to be commuted. Therefore, the master creates again a start condition (resp. restart: start without stop condition in between) and sends the slave address plus the read bit to switch into read mode. Figure 105 shows an example with op code "read from SRAM".



S	I2C-Address + W	A	Opcode + Read address	A	Read address	A	S	I2C-Address + R	A	Data	N	P
S	0101000 0	0	'h 21	0	'h 47	0	S	0101000 1	0	'h FF	1	P

Example: Read from SRAM address 0x147, we find 0xFF having been programmed before.

Figure 28: I²C Read Procedure

After arrival of the first (or any) data byte, the master may either signal

- Not-Acknowledge = N = 1 to indicate “end read“, “stop sending“ to the slave, or
- Acknowledge = A = 0 to indicate “continue in automatic address-increment mode” and thus receive many bytes in a row. As one can see, automatic address increment is particularly useful and efficient with the I²C interface.

7.4.3 SPI Interface

Clock Polarity, Clock Phase and Bit Order: The following choices are necessary for successful operation.

7.4.3.1 SPI Settings

Table 78: SPI Settings

SPI - Parameter	Description	Setting
CPOL	Clock polarity	0
CPHA	Clock phase	1
Mode	SPI Mode	1
DORD	Bit sequence order	0, MSB first

7.4.3.2 SPI Timing

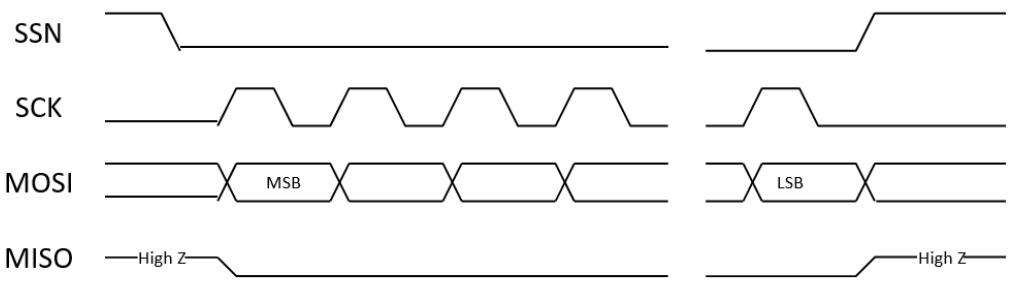


Figure 29: SPI Write Procedure

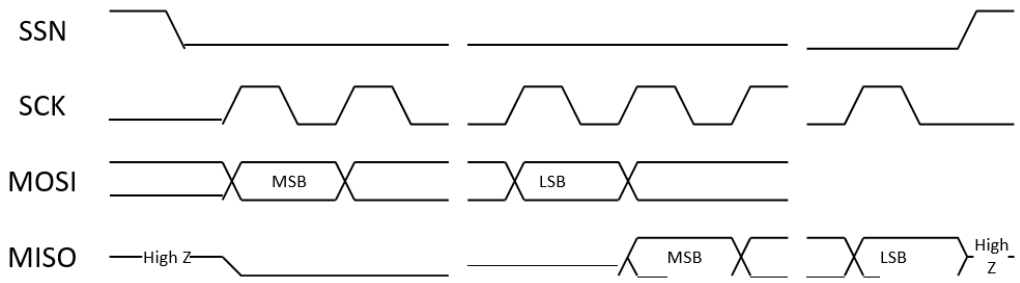


Figure 30: SPI Read Procedure

Table 79: SPI Timing

Name	Symbol	VDD=2.2V	VDD=3.0V	VDD=3.6V	Units
Serial clock frequency	$f_{SPI-bus}$	10	17	20	MHz
Serial clock pulse width HI state	t_{pwh}	50	30	25	ns
Serial clock pulse width LO state	t_{pwl}	50	30	25	ns
SSN enable-to-valid latch	t_{sussn}	10	8	7	ns
SSN pulse width between write cycles	t_{pwssn}	50	30	25	ns
Data setup time prior to clock edge	t_{sud}	7	6	5	ns
Data hold time after clock edge	t_{hd}	5	4	3	ns
Data valid after clock edge	t_{vd}	40	26	16	ns

7.4.4 GPIO and PDM/PWM

This section is about the general purpose ports and their use as Pulse-Density / Pulse Width Modulated outputs (PDM/PWM). PCap04 is very flexible with assignment of the various GPIO pins to the DSP inputs/outputs. The following table shows the 6 general purpose ports and their possible assignment.

Table 80: General Purpose Port Assignment

External Port Name	Description	Direction In or Out
PG0	SSN (in SPI-Mode), serial select	In
	DSPØ or DSP2, I/O for the DSP	In[16] / Out
	FF0 or FF2, I/O for the DSP with Flip-Flop	In[5]
	Pulse0, PDM or PWM output	Out
PG1	MISO (in SPI-Mode)	Out
	DSP1 or DSP3, I/O for the DSP	In[16] / Out
	FF1 or FF3, I/O for the DSP with Flip-Flop	In[16]
	Pulse1, PDM or PWM output	Out
PG2	DSPØ or DSP2, I/O for the DSP	In[16] / Out
	FF0 or FF2, I/O for the DSP with Flip-Flop	In[16]
	Pulse0, PDM or PWM output	Out
PG3	DSP1 or DSP3, I/O for the DSP	In[16]/ Out
	FF1 or FF3, I/O for the DSP with Flip-Flop	In[16]
	Pulse1, PDM or PWM output	Out
	C_G_MUX_SEL output	Out
PG4	DSP4 (output only)	Out
	INTN	Out
PG5	DSP5 (output only)	Out
	INTN	Out

Table 81: General Purpose Port Assignment

Reg	Parameter	Settings	Description

[5] These ports provide an optional debouncing filter and an optional pull-up resistor.

27	DSP_MOFLO_EN	Bit 6 for PG0 Bit 7 for PG1	Activates anti-bouncing filter in PG0 and PG1 lines
27	PG0xPG2	0: PG0 1: PG2	The pulse codes can be output at ports PG0 & PG1 or PG2 & PG3. In I²C mode they can be optionally given out on PG2 and PG3, instead of PG0 and PG1.
27	PG1xPG3	0: PG1 1: PG3	
29	DSP_FF_IN	Bit 0 : PG0 Bit 1 : PG1 Bit 2 : PG2 Bit 3 : PG3	Pin mask for latching flip-flop activation
30	PG4_INTN_EN	Bit 6	Activates INTN at port PG4
30	PG5_INTN_EN	Bit 7	Activates INTN at port PG5
33	PG_DIR_IN	0: Output 1: Input	Toggles outputs to inputs (PG3/bit7 to PG0/bit4).
33	PG_PU	Bit 0 : PG0 Bit 1 : PG1 Bit 2 : PG2 Bit 3 : PG3	Activates pull-up resistors in PG0 to PG3 lines; useful for mechanical switches.

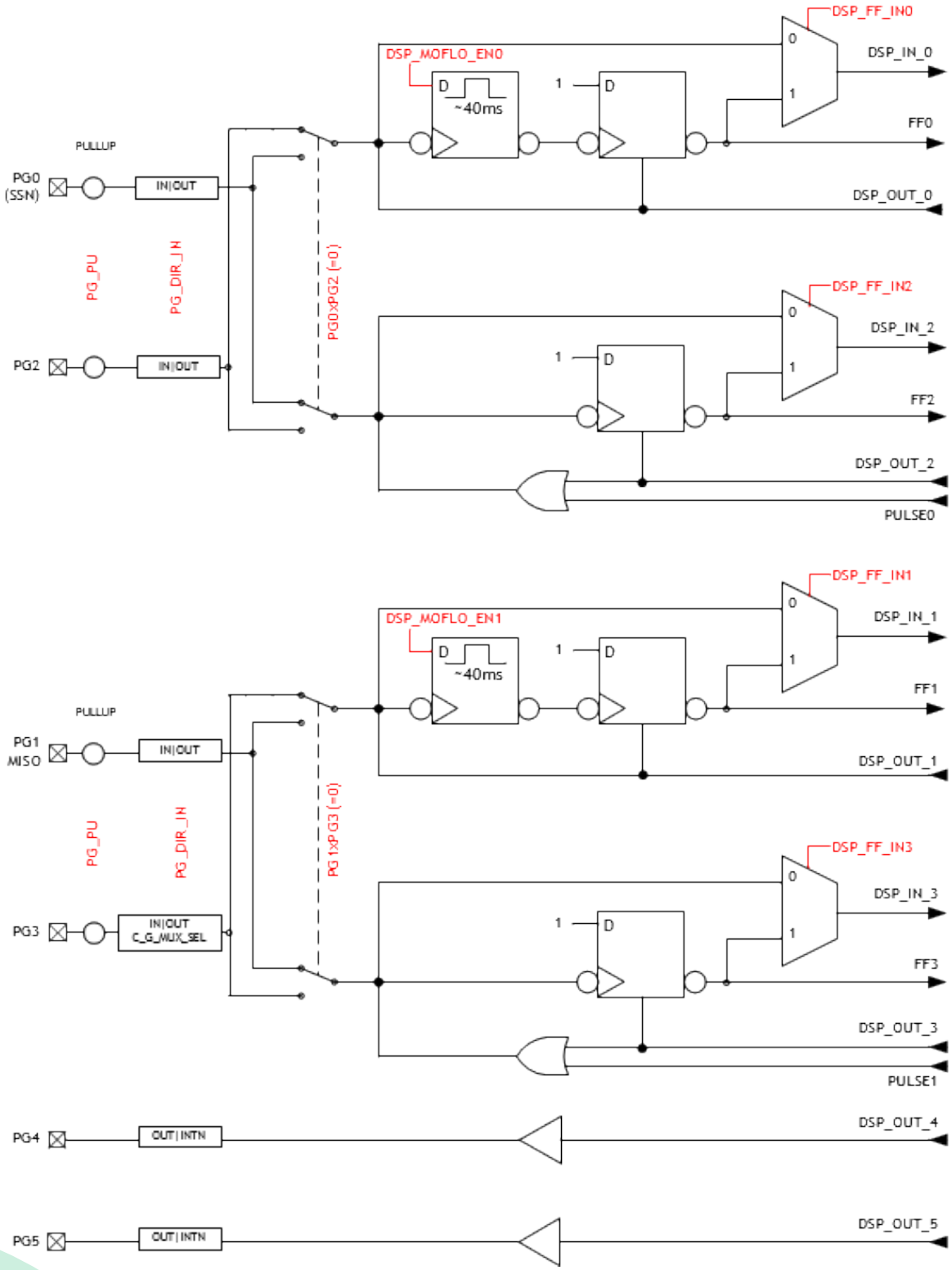


Figure 31: GPIO Assignment

7.4.4.1 Debouncing Filter

There is a possibility to activate a 40ms debounce filter (“monoflop“) for the ports in case these are used as push button inputs. This might be useful especially in case the DSP is started by the pins (signals FF0, FF2). Figure 32 shows the effect of the monoflop filter.

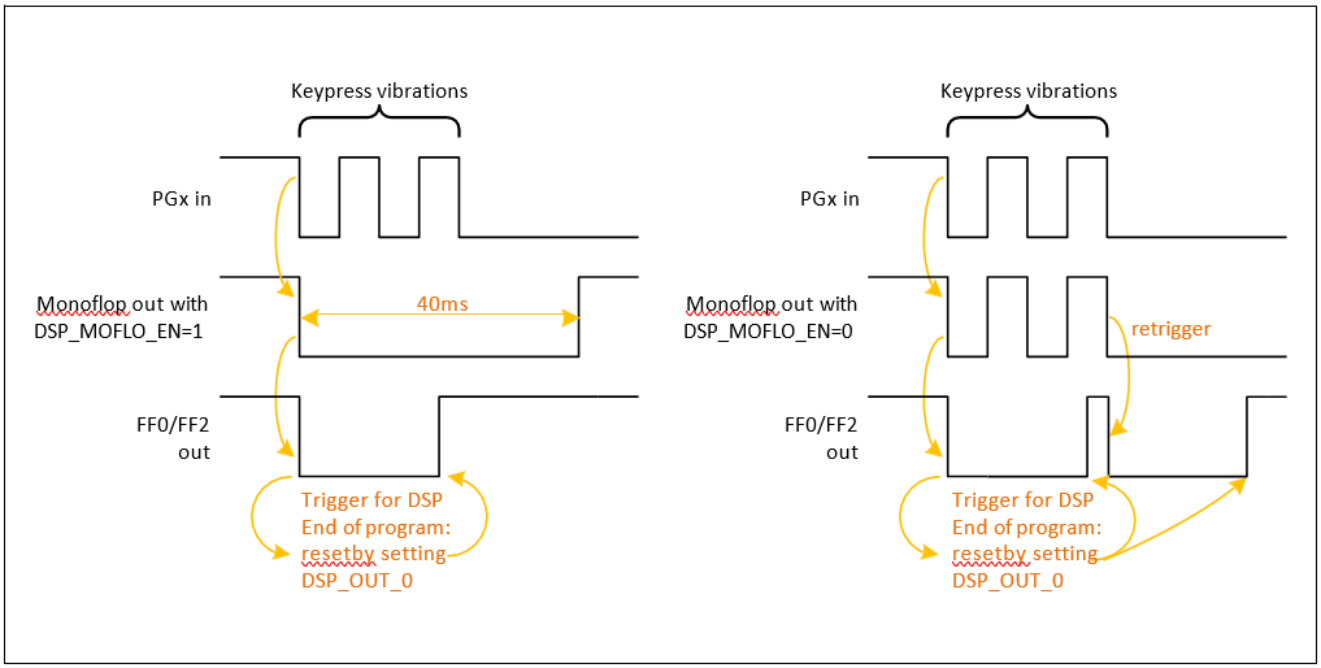


Figure 32: Port Trigger Timing

7.4.4.2 PDM and PWM

There is a possibility to generate two pulse density modulated output signals. In general, PDM is preferred because of better noise behaviour. The output is based on the content of RAM registers **PI0_REF**, **PI1_REF** (DSP write addresses 109, 110. Width 16 bit each). The content of those RAM cells depends on the firmware. The description in this datasheet is based on the standard firmware, which writes the capacitance ration to **PI0_ REF**, the Resistance ratio to PI1_REF.

The pulse interfaces can be switched on individually. The resolution can be programmed from 10 to 16 bit. There is a broad range of clock signals that can be selected as base for the pulse interfaces, derived from the 50 kHz low-frequency oscillator or the 2 MHz internal oscillator. The output pins may be PG0 or PG2 and PG1 or PG3.

The PDM signal can be converted into an analog voltage by means of a simple RC-filter. A first-order filter made of 220kΩ / 100nF is sufficient. By the choice or the resistor and capacitor values the user can optimize for reaction time vs. ripple.

Filter configuration instructions:

The resistor should be 50kOhm. The internal DC resistance of the output buffer is typ. 100Ω.

1. Settling time (for PDM and PWM)
   If the output value changes, the settling time to reach 90% is 2.3 x Tau, Tau = R x C
   Example: 200kΩ x 100nF x 2.3 = 50ms
   The smaller is Tau the faster is the settling but the higher is the ripple.
2. Voltage Ripple
   Calculation Method:

$$v_{pp} = V_{DD} \times \left(1 - e^{\frac{-1}{f_0 \times R \times C}}\right) \quad \text{, with } f_0 \ll \frac{1}{R \times C} \text{ this is } v_{pp} = \frac{V_{DD}}{f_0 \times R \times C}$$

$$v_{pp} = ripple\ voltage,\ f_0 = \frac{1}{t_{pulse\ width}}(PWM),\ f_0 = \frac{f_{clk}}{2^{PWM\ resolution(bit)}}(PDM)$$

In the standard firmware, the result of measurement from capacitance or temperature is a 32-bit value. The DSP linearizes this 32-bit result to a value according to the resolution settings of the pulse interface. The parameters pi<n>_result0, pi<n>_ result1, pi<n>_pulse0 and pi<n>_pulse1 of the linear function are configurable in NVRAM, calibration space 800 to 822. The parameters are describing the edges for a simple scaling (1st order linearization), whereby pi<n>_pulse0 is also the minimum clipping values and pi<n>_pulse1 the maximum clipping value for the pulse output. Pi<n>_pulse0 must be always smaller than pi<n>_pulse1. For negative slopes just pi<n>_result0 has to be larger than pi<n>_result1. A 12-bit resolution thus limits the result value between 0 and 4096. For lower-bit resolutions, the range reduces accordingly.

The pulse_out is determined like this:

$$pi_n out = \frac{pi_n pulse\,1 - pi_n pulse\,0}{pi_n result\,1 - pi_n result\,0} \cdot (result - pi_n result\,0) + pi_n pulse\,0$$

$$pi_n result\,1 > pi_n result\,0$$

$$0 \le pi_n pulse\,1 \le 2^{pulse\,resolution}$$

$$0 \le pi_n pulse\,0 \le 2^{pulse\ resolution}$$

The following figure depicts how the result is processed to generate the pulsed output.

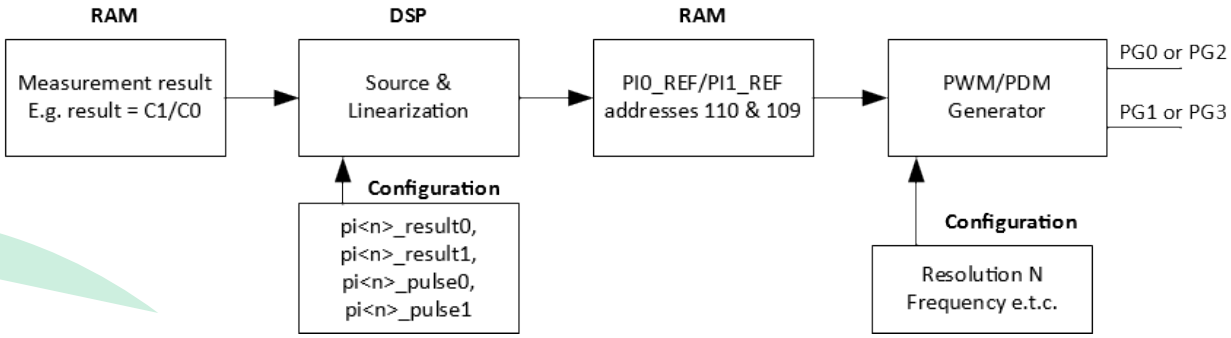


Figure 33: PWM and PDM pulse generation

Figure 34 shows a sample linear function and its parameters graphically. In this graph, the result C1/C0 has been taken on the x-axis, assuming that this result is to be pulse modulated. A 12 bit resolution has been configured.

The settings for the PDM and PWM interface are made in configuration registers 27 and 29 to 33.

The lower limit (pi<n>_pulse0) of the valid range corresponds to 0% modulation (all bits are 0), The upper limit (pi<n>_pulse1) of the valid range corresponds to 100% modulation (all bits are 1), and this is the maximum possible value of output. 12 bit resolution implies that this maximum value is 4095. For lower-bit resolutions, this maximum value will come down accordingly. In terms of voltage, the two limits correspond to 0V and VDD.

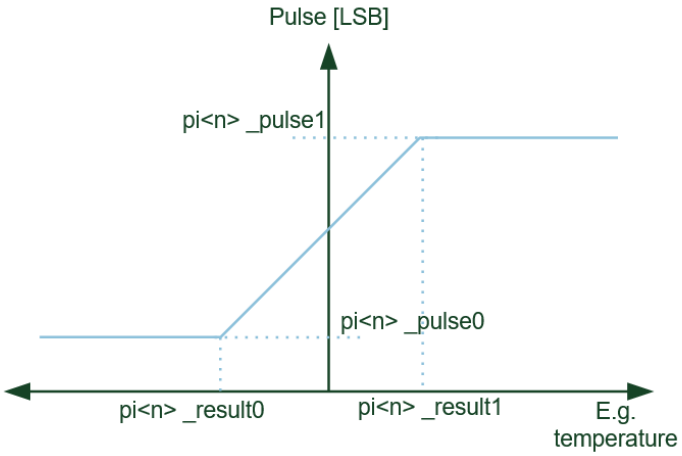


Figure 34: PWM and PDM Linearization

Applications:

- A typical case would be outputting capacitance results through PG0 and temperature results through PG1. Calculation and transfer to the output registers will be performed by firmware.
- Main applications will be that an analog interface is demanded by the final customer.
- Applications where the serial interface cannot be used due to speed limitations or other reasons.
- Finally, a temperature-coded pulse stream could be low-pass filtered and then directly used for temperature control.

Please note that the entire linearization task as described here is performed by firmware, especially the PCap04_standard and PCap04_linearization firmware.

7.5 Oscillators

PCap04 offers a low frequency oscillator (**OLF_CLK**) and an integrated 2 MHz high frequency oscillator (**OHF_CLK**). **OLF_ CLK** is running all the time and cannot be turned off.

OLF_CLK is used for:

- CDC cycle time
- RDC cycle time

- PDM/PWM time base
- Watchdog for standalone applications

The **OHF_CLK** can be used alternatively for

- CDC cycle time
- PDM/PWM time base

The **OLF_CLK** can be trimmed for various typical frequencies:

Table 82: OLF Trimming

OLF_CTUNE	OLF_FTUNE	OLF Frequency
0 (10kHz)	1	5kHz
	7	10kHz
1 (50kHz)	0	28kHz
	3	48kHz
2 (100kHz)	4	100kHz
3 (200kHz)	5	200kHz

Note: The internal oscillators are not very precise and stable. The frequency varies from chip to chip, with temperature and voltage.

- Variation over batch ± 20%
- Variation with temperature ± 5%,
- Variation with voltage. VDD ± 2%

The OHF can be switched off, turned on with delay before further tasks like measurement follow, or turned on continuously:

OX_RUN[2:0]

0: Generator off
6: OX latency = 1 / f_{OLF}
3: OX latency = 2 / f_{OLF}
2: OX latency = 31 / f_{OLF}
1: OX runs in permanence

By means of **OX_DIV4** it can be divided by 4 to generate 500 kHz.

7.6 DSP & Memory

This section describes the 32bit-DSP of the PCap04.

A 32-bit digital signal processor (DSP) in Harvard architecture was integrated to the PCap04. It is responsible for taking the information from the CDC and RDC measuring units, for processing the data and making them available to the user interface. Both, the CDC/RDC raw data as well as the data processed by the DSP are stored in the RAM. The program for the DSP is stored either in the NVRAM. The DSP can collect various status information from a set of 64 I/O Bits and write back 16 of those. This way the DSP can react on and also control the GPIO pins of PCap04. The DSP is internally clocked at approximately 60MHz. The internal clock is stopped through a firmware command, to save power. The DSP starts again upon a GPIO signal or an “end of measurement” condition.

In its simplest form, the DSP transfers the pure time measurement information from the CDC/RDC to the read registers without any further processing. The next higher step is to calculate the capacitance ratios including the information from the compensation measurements, as it is provided in ScioSense standard firmware version PCap04_standard_v01.hex. Finally, ScioSense provides a ready-made linearize firmware that performs a linearization via polynomial of third degree and temperature compensation via polynomial of second degree. Many functional blocks for the linearization firmware are implemented as ROM code. This way, the main firmware can be very compact and can fit into the 1k NVRAM.

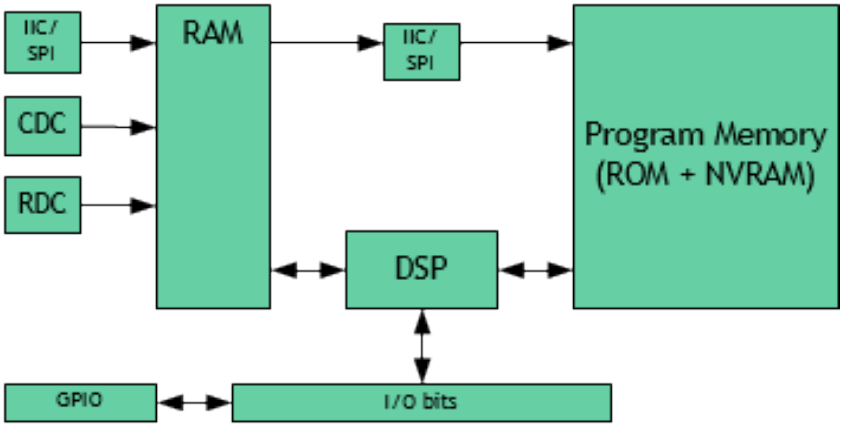


Figure 35: DSP Enabling

The content of the read registers will always depend on the firmware in use. With the standard firmware it will be the pure capacitance and resistance ratios. With the linearization firmware it might the linearized and calibrated result, e.g. a pressure given in Pascal or humidity given in percent.

The DSP is ScioSense proprietary to cover low-power tasks as well as very high data rates. It is programmed in Assembler. A user-friendly assembler software with a graphical interface, help text pop-ups as well as sample code sustain programming efforts.

7.6.1 DSP & Environment

The DSP reads the RDC and CDC raw data from the RAM, processes then and writes the results back to the RAM. The program is stored in the NVRAM. It may use subroutines that are available from the ROM. The DSP reacts on flags and controls flags. It controls the GPIO and accordingly the PDM/PWM interface.

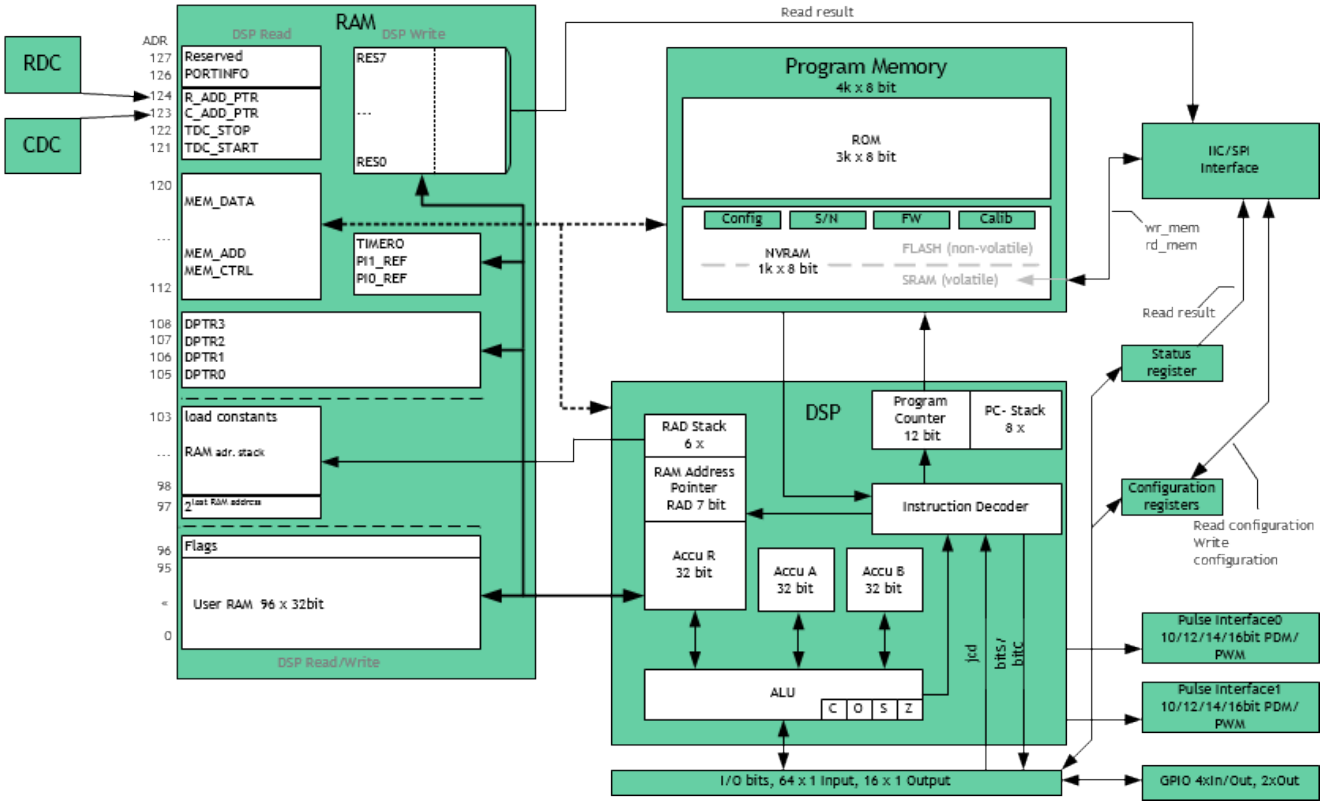


Figure 36: DSP Environment

The DSP is designed in Harvard architecture for 32 bit wide parallel data processing. It is coupled to a 128 x 32 bit RAM, 96 x 32 bit thereof freely accessible. In read access, the DSP can get the MEM_DATA from address space 112 to 120, the CDC- and RDC-frontend control from address space 121 to 124. By write access the DSP provides the output data to the PDM/PWM interfaces (addresses 109, 110).

A detailed description of the RAM is given is section 2.1. The DSP operates with two accumulators A and B and has direct access to the RAM, which can be seen as a third accumulator. The RAM address pointer is of 7 bit size, and there is a 6-fold stack for RAM addresses.

The program counter has 12 bit and there is an 8-fold stack for the program counter.

Finally, the DSP can get a lot of information from the 64 I/O bits. The read information covers the ALU status, trigger information, some of the configuration bits and the information about the status of the GPIOs. 16 of those bits can be used as outputs, setting the GPIOs and also some internal information. The DSP can read these bits by means of

instruction jcd (conditional jump) and set those bits by means of instructions bitS/bitC (bit Set/Clear).

The ALU flags overflow, carry, equal/not equal and pos./neg. are used directly as condition for the jcd instructions and are also mirrored in the I/O bits.

7.6.2 RAM Structure

The RAM plays a key role. It is made of 128 words with size of maximum 32 bit. The DSP has free write and read access to registers address 0 to 96 of those words, all 32 bits wide. The RAM space addresses 97 to 104, 109 to 111 and 115 and higher has different meaning for read and write operations.

The main data in the read section are the raw data as they come from the CDC and the RDC. Also the parameters are in the RAM as part of the configuration registers and they are set via the serial interface or copied from the NVRAM.

The DSP reads the raw data, does the data processing and writes back the results into the write section of the RAM. From there, the user can read the final results through the serial interface.

Some of the RAM cells are dedicated to special functions and will be described in the following in detail.

Table 83: RAM Structure in Detail

RAM: DSP Read				RAM: DSP Write		
Addr	Description	Bits		Addr	Description	Bits
127	Reserved					-
126	PORTINFO	24				
124	R_ADD_PTR	2				
123	C_ADD_PTR	4				
122	TDC_STOP	27				
121	TDC_START	27				
120	MEM_DATA_u08b	32				
119	MEM_DATA_u16b	32				
118	MEM_DATA_u24b	32				
117	MEM_DATA_s08b	32				
116	MEM_DATA_s16b	32				
115	MEM_DATA_s24b	32				
114	MEM_DATA (wr & 4Byte rd)					32

113	MEM_ADD					10
112	MEM_CTRL					16
				111	TIMER0	16
				110	PI1_REF	16
				109	PI0_REF	16
108	DPTR3					7
107	DPTR2					7
106	DPTR1					7
105	DPTR0					7
				104	RES7	32
103	rad_stack_6b	6		103	RES6	32
102	rad_stack_12b	12		102	RES5	32
101	rad_stack_18b	18		101	RES4	32
100	rad_stack_24b	24		100	RES3	32
99	rad_stack_30b	30		99	RES2	32
98	rad_stack_32b	32		98	RES1	32
97	2last_RAM_address	32		97	RES0	32
96	Flags & extended GPIO's					32
95	(free) User RAM					32
…	…					32
0	(free) User RAM					32

7.6.2.1 Registers 0 to 95, User RAM

This is normal RAM space without any special functions. It is readable and writable via instruction rad. Example: Add content of RAM address 12 and 13 and write the result into RAM address 13

```
rad 12
move a, r
rad         13
add         r, a
```

Table 84: Flags

Bit	Flag Name	Default (after Reset)	Description
0	FIRSTART_N	0	Indicates first DSP-Trigger until set to 1 by firmware
1..2	Free to use	0	
3	RDCHG_COM_INT_SEL	0	0: Use RDCHG_IN_SEL0 1: Use RDCHG_IN_SEL1 For internal compensation
4	Free to use	0	
5	FLAG_CDC_INV	0	Temporary parameter to ROM routine _ROM_cdc 0 : Determine inverse ratios reference/sense 1 : Determine ratios sense/reference
	SIGNED_VALUE_NV	0	Temporary parameter to ROM routines _ROM_NVblock_copy_32b_ _ROM_NVblock_copy_24b_ _ROM_NVblock_copy_16b_ _ROM_NVblock_copy_08b_ 0 : Treat data from NVRAM as unsigned 1 : Treat data from NVRAM as signed
6..7	Reserved	0	Temporarily used flags within ROM routines
8	RST_RDC	Pulsed	Temperature reset. This flag has to be set 1, after each RDC measurement. Otherwise a new RDC measurement is not possible. This flag is set back to 0 automatically
9..15	Reserved		
16..31	Free to use	Unknown	

7.6.2.2 DSP Read Register 97

This register is there to get the N-th power of 2. The exponent N needs to be written to the RAD stack. The result can be read from register 81. In the assembler, the necessary three instructions are merged into one:

```
load2exp   a, 10 ; a = 2^10 =  1024
```

A very simple and efficient method to set an accumulator = 1 is

```
load2exp    b, 0 ; b = 2^0 = 1
```

7.6.2.3 DSP Read Registers 98 to 103

These registers contain the content of the RAM address stack. The 32 bit data is made of the 6 last 6-bit RAM addresses. This address can be used to load 32 bit constants from the program memory into the data space. The necessary instructions are merged into one single instruction by the assembler. (Hint: The assembler accepts negative values as well as decimal and

hexadecimal numbers. Depending on the constant to be load, the assembler converts this instruction into 3 to 8 operations)

```
load a, 1715956  ; a = 1715956
```

is the same as

```
rad 0x06   ; 0x06 * 218
rad 0x22   ;+ 0x22 * 212
rad 0x3b   ;+ 0x3b * 26
rad 0x34   ;+  0x34 = 1715956
rad 100    ; rad_stack_24b move a, r
```

7.6.2.4 DSP Read/Write Registers 105 to 108, Data Pointer

These registers may be used for indirect addressing. They are 7 bits wide.

Load a register with the address you want to manipulate:

```
load a, <myaddress>        load a, <myaddress>
rad DPTR0                  rad 105
move r, a                  move r, a
```

Load a RAM address pointer with content of DPTR0:

rad _at_DPTR0 ; now ram address pointer is set to content of DPTR0

Hint: In the <pcap_standard.h> "_at_DPTR0" to "_at_DPTR3" are set to values of 284 to 287. These are no valid RAM addresses but just indicators to the assembler to generate the corresponding opcodes.

Example direct memory address: Copy a memory block from one address to another:

```
__sub_dma :                                                    ; DPTR1 : source_address; DPTR0 :
not b                          ; initialize loop counter       destination address; b: length of dma
inc b                          ; with -  <length>
sub_dma_loop :                 ;
rad  _at_DPTR1                 ; copy a : @DPTR1
move a, r
rad _at_DPTR0                  ; copy @DPTR0 : a
move r, a

rad DPTR0                      ; increment destination
inc r                          ; address
rad DPTR1                      ; increment source add
inc r                          ;
inc b                          ; increment loop
                               ; counter
jNE       sub_dma_loop         ; loop body
jrt
```

7.6.2.5 DSP Read Register 126, PORTINFO (PORTERR<7...0>, PORTMASK<7...0>)

The low 8 bits mirror the port enable setting as defined by configuration parameter C_PORT_EN in register 12.

Bits 8 to 17 are error flags for the capacitance ports including the internal reference ports.

7.6.2.6 DSP Write Registers 97 to 104, RES00...RES07

These are the result registers to which the DSP has to write the output data so that the user can read those through the SPI/IIC interface as Res 0 to Res 7.

All addresses are 32 bit wide.

!!! Attention: These Registers are write only! The DSP cannot read from these Registers !!!

7.6.2.7 DSP Write Registers 109, 110, PI0_REF...PI1_REF

These registers contain the data that is used to generate the PWM/PDM output signals. After the DSP has calculated and scaled the output data, it writes those into these two registers. The data are 16 bit wide.

7.6.2.8 DSP Write Register 111, TIMER0

The DSP has a 16-bit timer based on the OLF clock. This timer may be used to generate long delays while the DSP is halted. Bit #3 (timer) in DSP_START_EN must be set!

By writing a value to Register 111 the timer starts to count up from 0 each OLF-clock cycle until the written value has been reached. Then a DSP_START_TRIG is generated.

If the DSP is not halted the TIMER0_IRQ_N Flag could be tested anyway.

Example 1 (without halting DSP):

```
CONST       wait_time_1ms 50 ; 50*20µs (@50kHz)
…
load a, wait_time_1ms rad TIMER0
move r, a

timer_wait_loop:
jcd TIMER0_IRQ_N, timer_wait_loop
```

Example 2 (with halting DSP, DSP run on internal oscillator):

```
CONST       wait_time_1ms 50       ; 50*20µs (@50kHz)
…
ORG 0
jcd TIMER0_IRQ_N, Skip_Timer0_process
            jsb Triggered_by_Timer0
Skip_Timer0_process:
…
load a, wait_time rad TIMER0
move r, a
stop

Triggered_by_Timer0:              ; subroutine
```

7.6.2.9 DSP Read/Write Registers 112 to 120, MEM_CTRL, MEM_ADD, MEM_DATA

Those registers are used by ROM routines for the transfer of data between NVRAM and RAM. It is possible to transfer data between NVRAM and accumulators a and b, from 1 to 4 bytes, signed and unsigned.

- MEM_CTRL: defines the operation. The four options are
  - MEM_STORE: write to NVRAM
  - MEM_RECALL: read from NVRAM
  - MEM_WE: enable writing
  - MEM_WR_PROTECT: protect against arbitrary writing
- MEM_ADD: defines the target address in the NVRAM
- MEM_DATA_xxx: registers for write or read data. Data that shall be written into NVRAM data need to be in RAM address 114. Data that have been read from NVRAM can be found in addresses 114 to 120, depending on the format.

These calls are used e.g. to copy constants and calibration data from the NVRAM to the RAM.

Example: Copying the NV_C_sens_sel register to RAM

```
load        a, NV_C_sens_sel            ; mem_add : NV_C_sens_sel
rad         mem_add
move        r, a
jsb         _ROM_memory_rd_a_u08b__     ; A contains content  of NV_C_sens_sel
```

```
rad         RAM_C_sens_sel
move        r, a
```

These registers may be used by the DSP to change the configuration on the fly.

Important note: After writing to the NVRAM it is necessary to do an ini_reset. Therefore the DSP has to do the following sequence of setting flags DSP_6 and DSP_7.

```
; Initreset
bitC 7
bitC 6
bitS 6
bitC 6
bitS 6
```

7.6.2.10 DSP Read Registers 121 to 124, TDC_START, TDC_STOP,C_ADD_PTR, R_ADD_PTR

ScioSense internal data, used by ROM routine __tdc_dispatch__

7.6.3 NVRAM and ROM

The total program memory is made of 1k NVRAM and 3k ROM. The NVRAM holds configuration data, 960 byte of user code and some special registers. The ROM holds useful mathematical routines that make programming very efficient.

7.6.3.1 NVRAM Structure

The user space is split in three sections. The reason is that the NVRAM can be read/write protected by section. A big section of 704 byte is for program code, two smaller sections of 128 byte may be used for calibration data or additional firmware.

Table 85: NVRAM Organization

Address		NVRAM (1k x 8 bit)		Memory Lock
Decimal	Hexadecimal	Contents	Length [Byte]	Settings
1023 to 1022	3FF to 3FE	CHARGE_PUMP	2	MEM_LOCK<3>
1021 to 1011	3FD to 3F3	Reserved	11	
1010 to 1009	3F2 to 3F1	S/N customer	2	
1008	3F0	MEM_LOCK	1	
1007 to 960	3EF to 3C0	Configuration Registry	48	MEM_LOCK<3>
959 to 832	3BF to 340	User Space (FW/CAL1)	128	MEM_LOCK<2>
831 to 704	33F to 2C0	User Space (FW/CAL0)	128	MEM_LOCK<1>
703 to 0	2BF to 0	User Space (FW)	704	MEM_LOCK<0>

The NVRAM consists of two parts: a volatile SRAM and a non-volatile memory (FLASH). There is a store/recall method to copy (store) the complete SRAM content to FLASH or to recall it from FLASH back to SRAM.

Different methods of operation apply:

- Stand Alone:
  Configuration data, firmware and calibration values are stored once to non-volatile memory and autoboot is selected. After a power-on the device starts immediately with the measurement.
- Pre-Configured:
  Configuration data, firmware and calibration values are stored once to non-volatile memory, RUNBIT and autoboot are disabled. After power-on, the device is programmed and configured, but in idle mode, waiting for instructions.
- Pure Slave:
  Configuration data, firmware and calibration values are written to the SRAM (volatile memory) after each power on by an external µC.

7.6.3.2 NVRAM Access

There are three commands available to handle the NVRAM, Store, Recall & Erase, each one protected to avoid accidental trigger during communication over the serial interface. It is mandatory to send first an activation code to register 54 (M**EM_CTRL**). This is followed by the according opcode (‖ = termination of SIF, e.g. setting SSN = HIGH):

- Store SRAM content into NVRAM:
  - Activation code in **MEM_CTRL**: 0x2D
  - Store NVRAM opcode: 0x96
    (Send via SIF: 0xA3F62D ‖ 0x96, wait minimum 12 ms)
- Recall from NVRAM into SRAM:
  - Activation in MEM_CTRL: 0x59
  - Recall from NVRAM opcode: 0x99 (
    Send via SIF: 0xA3F659 ‖ 0x99)
- Erase NVRAM:
  - Read trim bits adr 1022&1023 and Unique ID from adr 954 to 959
  - Activation in MEM_CTR: 0xB8 Erase NVRAM opcode: 0x9C
    (Send via SIF: 0xA3F6B8 ‖ 0x9C, wait minimum 12ms)
  - Write back trim bits adr 1022&1023 and Unique ID to adr 954 to 959
  - Activation code in MEM_CTRL: 0x2D
  - Store NVRAM opcode: 0x96

Important Note: We guarantee the data for data retention and endurance only under the assumption, that the customer does **not** change the registers 62 and 63 and NVRAM adr 954 to 959 (Unique ID). In addition, it is mandatory to follow the given procedure for ERASE NVRAM as described in section NVRAM and ROM precisely. Otherwise, we do no longer guarantee the data retention time and endurance cycles.

7.6.3.3 ROM Structure

The limitation in size for the NVRAM is compensated by having many functions integrated hard-wired in 3k ROM. The ROM routines range from simple shift functions over filters to polynomial linearization of 4th degree. This allows to keep the user code very compact.

The assembler comes with header file PCap04_ROM_addresses_standard.h that lists the jump-in addresses for the various ROM routines. For details see section Sample Code / Libraries.

7.6.3.4 DSP Inputs & Outputs

The DSP has access to 64 bits of information on ALU status, start trigger, configuration, input / output pins.

This information can be interpreted by means of instructions jcd or conditional jump.

Instruction conditional jump looks like:

```
jcd     p1,p2: if p1 ==1 then jump to p2, p1 = flag number
```

16 of those bits can be set by the DSP, e.g. to set a GPIO or to select between RDC and CDC data. The bits are controlled by means of instructions bitS / bitC (bit Set/bit Clear).

Table 86: DSP Inputs / Outputs

Bit Name	Description	Type	Read Bit #	Write Bit #
DSP_OUT<7...0>	Status feedback of the 8 general DSP outputs (Write bits 0 to 7).	IN	56 to 63	
SIF_TRIGGERED_N[6]	Flag = LOW indicates that a falling edge at a pin or an SPI/IIC opcode has started the DSP. This flag is reset by a STOP instruction at the end of the firmware.	Start trigger	55	
PIN_TRIGGERED_N[17]	Flag = LOW indicates a GPIO has started the DSP		54	
TDC_TRIGGERED_R_N[17]	Flag = Low indicates that a single time-values from Resistance (Temperature) measurement are available and must be processed (done by ROM routine _ROM_tdc_dispatch)			
TDC_TRIGGERED_C_N[17]	Flag = LOW indicates that time-values from Capacitance measurement are available and must be processed (done by ROM routine _ ROM_tdc_dispatch)	Start trigger	52	

[6] A negative edge on those inputs start the DSP. The status of the start trigger is memorized till the next reset or stop of the DSP. The start trigger information can be read from inputs 48 to 55 by jcd.

Bit Name	Description	Type	Read Bit #	Write Bit #
INTN_TRIGGERED_N[17]	Flag = LOW indicates the DSP is started by rising edge of INTN-Signal	Start trigger	51	
TIMER0_IRQ_N[17]	Flag = LOW indicates the DSP is started by the internal timer	Start trigger	50	
RDC_TRIGGERED_N[17]	Flag = LOW indicates that an RDC measurement has started the DSP. Therefore, DSP_ STARTONTEMP has to be set (configuration register 8). This flag is reset by a STOP instruction at the end of the firmware.	Start trigger	49	
CDC_TRIGGERED_N[17]	Indicates the DSP is started by the end of the capacitance conversion.	Start trigger	48	
ALU_OFL_N	ALU flags for overflow, carry, equal and sign. The ALU flags are used by the jump instruction of the assembler	Status	47	
ALU_OFL		Status	46	
ALU_CAR_N		Status	45	
ALU_CAR		Status	44	
ALU_EQ / ALU_ZERO		Status	43	
ALU_NE / ALU_ZERO_N		Status	42	
ALU_POS		Status	41	
ALU_NEG		Status	40	
FLAGREG_N[7:0]	Lower 8bits, inverted Flags from FLAGREG (register 96)	Flag	32..39	
AWAKE_TRIGGERED_N	After setting RUNBIT to 1 the DSP is triggered immediately. This flag shows this trigger source. Used for initialize raw result registers before first measurement (used by ROM routine _ROM_tdc_dispatch)	Start Trigger	31	
TDC_RDY	Flag = Low indicates TDC-Ring oscillator is running	Status	28	
POR_CDC_DSP_COLL	Flag = Low indicates reset forced by a CDC / DSP collision	Status	27	
LAST_CYCLE_ACTIVE_N	Flag = Low indicates that this is the last CDC measurement in the current sequence (used for ROM routine _ROM_tdc_dispatch)	Status	26	
CYC_ACTIVE	Flag = bit 23 of status register. Indicates that the CDC frontend is active. (not negated)	Status	25	

Bit Name	Description	Type	Read Bit #	Write Bit #
POR_FLAG_WD	Flag = Low indicates a reset was forced by watchdog timeout	Status	24	
POR_FLAG_PARITY	Flag = Low indicates a reset was forced by one or more configuration bits toggled by interferences.	Status	23	
CONTINUOUS_N	Low : Continuous mode is activated	Config Reg	22	
AUTOSTART_N	Bit from configuration register	Config Reg	21	
C_REF_INT	Bit from configuration register	Config Reg	20	
TIMER_TRIG_DSP			19	
(TRUE)	Constant 1, usable for “goto” jcd TRUE, <jump_ address>		18	
INT_TRIG_BG_N	Bit from configuration register	Config Reg	17	
CDC_TRIG_BG_N	Bit from configuration register	Config Reg	16	
C_COMP_EXT_N	Bit from configuration register	Config Reg	15	
C_COMP_IN_N	Bit from configuration register	Config Reg	14	
C_SINGLE / C_DIFFERENTIAL_N	Bit from configuration register	Config Reg	13	
C_GROUNDED / C_FLOATING_N	Bit from configuration register	Config Reg	12	
ERR_OVFLN	Flag = bit 16 of status register. Indicates an overflow or other error in the TDC.	Status	11	
COMB_ERRN	Flag = bit 16 of status register. This is a combined condition of all known error conditions.	Status	10	
CYC_ACTIVE_N	Flag = bit 23 of status register. Indicates that the CDC frontend is active. (negated)	Status	9	
SIF_RES_RD_BSY		Status	8	
RAM_BUSY	Indicates, NVRAM is busy	Status	7	
Interrupt_In	Port INTN will be reset by a positive edge on SSN (SPI) or a stop condition (I²C), with this flag the current status of INTN can be	Status	6	

Bit Name	Description	Type	Read Bit #	Write Bit #
TEMPERR_N	Flag = bit 3 of status register 1. Indicates whether an error occurred during the temperature measurement. 0 : Error, 1 : No error	Status	5	
RDC_BUSY	Flag = bit 2 of status register. Indicates RDC unit is busy. 0 : Measurement done, 1 : Measurement	Status	4	
TRIG_BG	This parameter starts the Bandgap (to synchronize with measurement) (pulse, automatically set to 0)	Out		15
(MEM_PUSH)	Reserved, only usable by ROM routines	Out		14
RST_CDC	CDC reset. This flag has to be set 1, after each CDC measurement. Otherwise, a new CDC measurement is not possible. This flag is set back to 0 automatically	Out		13
(MEM_RD)	Reserved, only usable by ROM routines	Out		12
Interrupt_Out	Sets the interrupt (pin PG4 or PG5, see register 30) (pulse, automatically set to 0)	Out		11
(PAGE)	Reserved, do not use	Out		10
TRIG_RDC	This bit starts a new RDC measurement. (pulsed, automatically set to 0)	Out		9
TRIG_CDC	This bit starts a new CDC measurement (pulsed, automatically set to 0)	Out		8
DSP_7	Those two outputs are used by the DSP for Reset watchdog INI_RESET by DSP Pattern combination of both Outputs are used to prevent these actions triggered accidentally); Initreset bitC 7 bitC 6 bitS 6 bitC 6 bitS 6	Out		7
DSP_6		Out		6
DSP_5	Sets the general purpose output pin PG5	Out		5
DSP_4	Sets the general purpose output pin PG4	Out		4

Bit Name	Description	Type	Read Bit #	Write Bit #
DSP_3	When the Pulse1 is switched OFF then this bit can be used to set and clear the general purpose output pin PG3. When the Pulse1 is ON then this bit must be cleared so that the Pulse1 output appears on PG3.	In/Out	3	3
DSP_2	When the Pulse0 is switched OFF then this bit can be used to set and clear the general purpose output pin PG2. When the Pulse0 is ON then this bit must be cleared so that the Pulse0 output appears on PG2	In/Out	2	2
DSP_1	Set or read the general purpose I/Os at pins PG0 & PG1. The assignment is programmable and shown in detail below.	In/Out	1	1
DSP_0		In/Out	0	0

7.6.3.5 ALU Flags

Every ALU operation sets flags. The ALU has four flags: overflow, carry, equal and sign. The following table shows an overview:

Table 87: ALU Flags

Flag	Description	Format	Modified by Instructions:	Interpreted by Instructions:	Range
ON	No Overflow	signed	add, sub, mult, div	jOvIC, jOvIS	>= -2^{31} and <= $2^{31}-1$
O	Overflow				< -2^{31} and > $2^{31}-1$
CN	No Carry[7]	unsigned	add, sub, mult, div	jCarC, jCarS	< 2^{32}
C	Carry[18]				>= 2^{32}
Z	Equal / Zero	signed / unsigned	add, sub, mult, div, move, shiftL, shiftR	jEQ, jNE	== 0
ZN	Not Equal / Not Zero				!=0
S	Positive	signed	add, sub, mult, div, move, shiftL, shiftR	jPos, jNeg	>= 0
SN	Negative				< 0

E.g. for A - B: if A ≥ B ↓ C = 1; if A < B ↓ C = 0.

In other words, the carry C is actually the status of the carry of the addition operation A+ 2´s complement (B).

[7] . During addition, the carry C is set when a carry-over takes place from the most significant bit, else C remains at 0. During subtraction, carry C is by default 1. Carry C is cleared only when the minuend < subtrahend.

7.6.3.6 DSP Configuration

Configuration register 8 defines the DSP operation. Relevant bits are:

DSP_SRAM_SEL, **DSP_START**, **DSP_STARTONOVL**, **DSP_STARTONTEMP**, **DSP_STARTPIN**, **DSP_WATCHDOG_LENGTH**, **DSP_SPEED**

Table 88: DSP Configuration

Reg	Parameter	Settings	Description
27	DSP_SPEED	0: Fastest 1: Fast 2: Recommended 3: Low-current (slow)	Setting the DSP speed
29	DSP_STARTONPIN	0: FF0 1: FF1 2: FF2 3: FF3	Pin mask for DSP trigger
30	DSP_START_EN<2..0>		DSP trigger enable 'bxxx1 : Trigger by end of CDC 'bxx1x : Trigger by end of RDC (recommended) 'bx1xx : Trigger by timer 'b1xxx : Obsolete
34	DSP_TRIG_BG	0: Disabled 1: Enabled	Bandgap refresh is triggered by start of DSP determination.

7.6.3.7 DSP Start

There are various options to trigger the DSP. In slave operation:

- Trigger by external controller. This is done by sending opcode “CDC Start conversion” or “DSP_TRIG”.

In stand-alone operation:

- Trigger by pin. The trigger pin is selected between pins PG0 to PG3 by configuration parameters **DSP_STARTPIN** and **PG0_X_PG2/PG1_X_PG3**. Signal FFx triggers the DSP. FFx has to be reset in the firmware by setting DSP_x, e.g.
  BitS DSP_2
  BitC DSP_2
- Trigger by the end of
  CDC
  RDC
  Timer
  Or by an interrupt. The option is selected by configuration parameter **DSP_START_EN**.

(Hint: DSP is also triggered by

- Toggling RUNBIT from 0 to 1. This is indicated by Flag **“AWAKE_TRIGGERED_N”**.
- After each CDC or RDC cycle. This is indicated by Flags **TDC_C_TRIGGERED_N** and **TDC_R_TRIGGERED_N**)

7.6.3.8 Watchdog

The watchdog is based on the OLF clock and counts always, even if the DSP is halted. If the DSP doesn't reset the Watchdog within 9s to 15s a power-on reset is generated => auto-boot. Status Flag **POR_FLAG_WDOG** is set.

The watchdog is implemented to handle situations where no CDC or RDC is running.

In applications as slave, the watchdog has to be disabled. This can be done by writing a 0x5A to WD_DIS. If the watchdog is used disarm the watchdog in advance to any SIF-Communication.

7.6.4 Instruction Set

The complete instruction set of the PCap04 consists of 29 core instructions that have unique op-code decoded by the CPU. Further, ScioSense offers a set of libraries including common constant definitions and mathematical operations

The library family is intended to be continuously expanded and be a great help during software development.

Table 89: Instruction Set

Simple Arithmetic	Miscellaneous	RAM Access	Bitwise Operation
add	init	rad	not
sign	nop	clear	and
sub	rst	load	or
inc	stop	load2exp	xor
	wdr	mov push pop	
Complex Arithmetic	**Shift & Rotate**	**Unconditional Jump**	**Bitwise**
div	shiftL	goto jsb	bitC
mult	shiftR	jrt	bitS
Conditional Jump			
jcd	jEQ	jOflC	
jCarC	jNE	jOflS	
jCarS	jNeg	jPOS	

and	Bitwise AND
Syntax:	and p1,p2
Parameters:	p1 = ACCU [a,b,r] p2 = ACCU [a,b,r] p1 != p2
Calculus:	p1 : p1 & p2
Flags affected:	C O S Z
Bytes:	1
Description:	Bitwise AND (conjunction)
Category:	Bitwise operation

add	Addition
Syntax:	add p1,p2
Parameters:	p1 = ACCU [a,b,r] p2 = ACCU [a,b,r]
Calculus:	p1 : p1 + p2
Flags affected:	C O S Z
Bytes:	1
Description:	Addition of two registers
Category:	Simple arithmetic

bitC	Clear single bit
Syntax:	bitC p1
Parameters:	p1 = number 0 to 15
Calculus:	Set bit number p1 of the DSP output bits bit = 0
Flags affected:	-
Bytes:	1
Description:	Clear a single bit in the DSP output bits
Category:	Bitwise

bitS	Set single bit
Syntax:	bitS p1
Parameters:	p1 = number 0 to 15
Calculus:	Set bit number p1 of the DSP output bits bit = 1
Flags affected:	-
Bytes:	1
Description:	Set a single bit in the DSP output bits
Category:	Bitwise

clear	Clear register
Syntax:	clear p1
Parameters:	p1 = ACCU [a,b,r]
Calculus:	p1 : 0
Flags affected:	S Z
Bytes:	2
Description:	Clear addressed register to 0
Category:	RAM access

div	Unsigned division
Syntax:	div
Parameters:	-
Calculus:	Single div code: b : (a/r), a : Remainder * 2 N div codes: b : (a/r)*2^(N-1), a : Remainder * (2^N)
Flags affected:	S Z
Bytes:	1
Description:	Unsigned division of two 32-bits registers. When the div opcode is used once, the resulting quotient is assigned to register 'b'. The remainder can be calculated from 'a'. When N div opcodes are used one after another, the result in b : (a/r)*2^(N-1). See also ROM routine div_xx. Before executing the first division step, the following conditions must be satisfied: 'b' = 0, and 0<'a'<2*'r'. If this condition is not satisfied, you can shift 'a' until this is satisfied. After shifting, if a -> a* (2^ea) and r -> r* (2^er), then the resulting quotient b for N division steps is b: (a/r) * 2^(1+ea-er-N) a = Remainder * (2^N)
Category:	Complex arithmetic

inc	Increment register
Syntax:	inc p1
Parameters:	p1 = ACCU [a,b,r]
Calculus:	p1 : p1 + 1
Flags affected:	C O S Z
Bytes:	1
Description:	Increment register
Category:	Simple arithmetic

init	Init reset
Syntax:	Init
Parameters:	-
Calculus:	-
Flags affected:	C O S Z
Bytes:	1
Description:	Initialization and reset. Sets back CDC, RDC and CPU. Copies configuration from NVRAM into configuration registers.
Category:	Miscellaneous

jCarC	Jump on Carry Clear
Syntax:	jCarC p1
Parameters:	p1 = jumplabel
Calculus:	if (carry == 0) PC : p1
Flags affected:	-
Bytes:	2
Description:	Jump on carry clear. Program counter will be set to target address if carry is clear. The target address is given by using a jumplabel. The conditional jump does not serve the stack. Therefore it is not possible to return by jrt. If the target address is beyond the range of current address (PC) -128/+127 bytes, then the assembler software will substitute this opcode for the following optimization: jCarS new_label jsb p1 jrt new_label: …..... In this case the stack will be loaded with p1, and therefore the stack capacity will be reduced by one.
Category:	Conditional jump

jCarS	Jump on Carry Set
Syntax:	jCarS p1
Parameters:	p1 = jumplabel
Calculus:	if (carry == 1) PC : p1
Flags affected:	-
Bytes:	2
Description:	Jump on carry set. Program counter will be set to target address if carry is set. The target address is given by using a jumplabel. The conditional jump does not serve the stack. Therefore it is not possible to return by jrt. If the target address is beyond the range of current address (PC) -128/+127 bytes, then the assembler software will substitute this opcode for the following optimization: jCarC new_label jsb p1 jrt new_label: …..... In this case the stack will be loaded with p1, and therefore the stack capacity will be reduced by one.
Category:	Conditional jump

jcd	Conditional Jump
Syntax:	jcd p1, p2
Parameters:	p1 = Flag or input port bit [63...0]. See section 2.3 for DSP Inputs. p2 = jumplabel
Calculus:	If (p1 == 1) PC : p2
Flags affected:	-
Bytes:	2
Description:	Program counter is set to target address if the bit given by p1 is set to one. The target address is given by using a jumplabel. The conditional jump does not serve the stack. Therefore it is not possible to return by jrt. If the target address is beyond the range of current address (PC) -128/+127 bytes, then the assembler software will substitute this opcode for the following optimization: jcd p1, new_label1 jsb new_label2 jrt new_label1: jsb p2 jrt new_label2: ;...
Category:	Conditional jump

jEQ	Jump on Equal
Syntax:	jEQ p1
Parameters:	p1 = jumplabel
Calculus:	if (Z == 0) PC : p1
Flags affected:	-
Bytes:	2
Description:	Jump on equal resp. zero. Program counter will be set to target address if the foregoing result is zero. The target address is given by using a jumplabel. The conditional jump does not serve the stack. Therefore it is not possible to return by jrt. If the target address is beyond the range of current address (PC) -128/+127 bytes, then the assembler software will substitute this opcode for the following optimization: jNE new_label jsb p1 jrt new_label: …… In this case the stack will be loaded with p1, and therefore the stack capacity will be reduced by one.
Category:	Conditional jump
jNE	Jump on Not Equal
Syntax:	jNE p1

Parameters:	p1 = jumplabel
Calculus:	if (Z == 1) PC : p1
Flags affected:	-
Bytes:	2
Description:	Jump on not equal resp. not zero. Program counter will be set to target address if the foregoing result is zero. The target address is given by using a jumplabel. The conditional jump does not serve the stack. Therefore it is not possible to return by jrt. If the target address is beyond the range of current address (PC) -128/+127 bytes, then the assembler software will substitute this opcode for the following optimization: jEQ new_label jsb p1 jrt new_label: …… In this case the stack will be loaded with p1, and therefore the stack capacity will be reduced by one.
Category:	Conditional jump

jNeg	Jump on Negative
Syntax:	jNeg p1
Parameters:	p1 = jumplabel
Calculus:	if (S == 1) PC : p1
Flags affected:	-
Bytes:	2
Description:	Jump on negative. Program counter will be set to target address if the foregoing result is negative (Bit 31 == 1). The target address is given by using a jumplabel. If the target address is beyond the range of current address (PC) -128/+127 bytes, then the assembler software will substitute this opcode for the following optimization: jPos new_label jsb p1 jrt new_label: …… In this case the stack will be loaded with p1, and therefore the stack capacity will be reduced by one.
Category:	Conditional jump

jOvlC	Jump on Overflow Clear
Syntax:	jOvlC p1
Parameters:	p1 = jumplabel
Calculus:	if (O == 0) PC : p1
Flags affected:	-
Bytes:	2
Description:	Jump on overflow clear. Program counter will be set to target address if the overflow flag of the foregoing operation is clear. The target address is given by using a jumplabel. The conditional jump does not serve the stack. Therefore it is not possible to return by jrt. If the target address is beyond the range of current address (PC) -128/+127 bytes, then the assembler software will substitute this opcode for the following optimization: jOflS new_label jsb p1 jrt new_label: ……. In this case the stack will be loaded with p1, and therefore the stack capacity will be reduced by one.
Category:	Conditional jump

jOvlS	Jump on Overflow Set
Syntax:	jOvlS p1
Parameters:	p1 = jumplabel
Calculus:	if (O == 1) PC : p1
Flags affected:	-
Bytes:	2
Description:	Jump on overflow set. Program counter will be set to target address if the overflow flag of the foregoing operation is set. The target address is given by using a jumplabel. The conditional jump does not serve the stack. Therefore it is not possible to return by jrt. If the target address is beyond the range of current address (PC) -128/+127 bytes, then the assembler software will substitute this opcode for the following optimization: jOflC new_label jsb p1 jrt new_label: ……. In this case the stack will be loaded with p1, and therefore the stack capacity will be reduced by one.
Category:	Conditional jump

jPos	Jump on Positive
Syntax:	jPos p1

Parameters:	p1 = jumplabel
Calculus:	if (S == 0) PC : p1
Flags affected:	-
Bytes:	2
Description:	Jump on positive. Program counter will be set to target address if the foregoing result is positive (Bit 31 == 0). The target address is given by using a jumplabel. The conditional jump does not serve the stack. Therefore it is not possible to return by jrt. If the target address is beyond the range of current address (PC) -128/+127 bytes, then the assembler software will substitute this opcode for the following optimization: jNeg new_label jsb p1 jrt new_label: …… In this case the stack will be loaded with p1, and therefore the stack capacity will be reduced by one.
Category:	Conditional jump

jrt	Return from subroutine
Syntax:	jrt
Parameters:	-
Calculus:	PC : PC from jsub-call
Flags affected:	-
Bytes:	1
Description:	Return from subroutine. A subroutine can be called via ‘jsb’ and exited by using jrt. The program is continued at the next command following the jsb-call. You have to close a subroutine with jrt - otherwise there will be no jump back. The stack is decremented by 1.
Category:	Unconditional Jump

goto	Unconditional relative Jump
Syntax:	goto p1
Parameters:	p1 = jumplabel
Calculus:	PC : p1
Flags affected:	-
Bytes:	2
Description:	Jump to jumplabel. Program counter will be set to target address. The target address is given by using a jumplabel. The goto command does not serve the stack. Therefore it is not possible to return by jrt. If the target address is beyond the range of current address (PC) -128/+127 bytes, then the assembler software will substitute this opcode for the following optimization: goto new_label1 jsb new_label2 jrt new_label1: jsb p2 jrt new_label2: ;...
Category:	Unconditional Jump

jsb	Unconditional Jump
Syntax:	jsb p1
Parameters:	p1 = jumplabel
Calculus:	PC : PC from jsub-call
Flags affected:	-
Bytes:	2
Description:	Jump to subroutine without condition. The programm counter is loaded by the address given through the jumplabel. The subroutine is processed until the keyword ‘jrt’ occurs. Then a jump back is performed and the next command after the jsub-call is executed. This opcode needs temporarily a place in the program counter stack (explanation see below). The stack is incremented by 1.
Category:	Unconditional Jump

load	Load Accumulator
Syntax:	load p1,p2
Parameters:	p1 = ACCU [a,b] p2 = 6..32-bit integer number (positive/negative, decimal or hexadecimal)
Calculus:	p1 : p2
Flags affected:	S Z
Bytes:	3..8 (depending on p2)
Description:	Move constant to p1 (p1=ACCU, p2=NUMBER) The following instruction is not allowed: load r, NUMBER This instruction is a macro that is replaced by the following opcodes: rad NUMBER[23:18] rad NUMBER[17:12] rad NUMBER[11:6] rad NUMBER[5:0] rad rad_stack_24b move [a, b], r Here the 24-bits number is split into four pieces, the symbol [xx:yy] indicates the individual bit range belonging to each piece. Please notice that the ram address pointer is changed during the operations, keep this in mind while coding.
Category:	RAM access

load2exp	Load Accumulator with 2exp
Syntax:	load2exp p1,p2
Parameters:	p1 = ACCU [a,b] p2 = 6-bit number
Calculus:	p1 : 2^p2
Flags affected:	S Z
Bytes:	2
Description:	Move 2^(p2) to p1(p1=ACCU, p2=NUMBER) The following instruction is not allowed: load r, NUMBER This instruction is a macro that is replaced by the following opcodes: rad NUMBER[5:0] rad load2exp move [a, b], r
Category:	RAM access

mov	Move
Syntax:	mov p1,p2
Parameters:	p1 = ACCU [a,b,r] p2 = ACCU [a,b,r]
Calculus:	p1 : p2
Flags affected:	S Z
Bytes:	1
Description:	Move content of p2 to p1 Assembler will understand also the old opcode move
Category:	RAM access

mult	Multiply
Syntax:	mult
Parameters:	-
Calculus:	ab : (b * r)
Flags affected:	S Z
Bytes:	1
Description:	Unsigned multiplication of the content of ab and r registers. ab is the composition of the registers a and b, forming an 64-bits long register, where 'a' takes the most significant bits, and register 'b' takes the less significant ones. The result is stored in the composed register a and b. The register 'a' must be previously cleared. This instruction only executes one multiplication step, to execute a full 32-bits multiplication, this instruction must be executed 32 times. This has the disadvantage of being tedious to code, but also has the advantage of executing only the amount of arithmetic needed, if you do not need a 32-bits multiplication but N, where N<32, then you have only to execute N multiplication steps in order to complete the full N-bits multiplication. After one multiplication step, register 'a' contains ((a+(b[0]*r))>>1), and register 'b' contains { a[0], b[3:1] }. For example: lets denote the individual bits of register 'a' as a[31], a[30], a[29].....a[2], a[1], a[0], and lets denote a range of bits of 'a' as: a[3:0], meaning the 4 less significant bits of register 'a'. Then, after one multiplication step, a[30:0] = (a[31:0] + r[31:0] * b[0]) >> 1, where >> 1, means right shift by one position; the value of a[31] is zero, and b[31] = (a[0] + r[0] * b[0]), and b[30:0] = b[311]. The register r remains unchanged.
Category:	Complex arithmetic

nop	No operation
Syntax:	-

Parameters:	-
Calculus:	-
Flags affected:	-
Bytes:	1
Description:	Placeholder code or timing adjust (no function)
Category:	Miscellaneous

not	Bitwise NOT
Syntax:	not p1
Parameters:	p1 = ACCU [a,b,r]
Calculus:	p1 : ~ p1
Flags affected:	C O S Z
Bytes:	1
Description:	Invert register (negation)
Category:	Bitwise operation

or	Bitwise OR
Syntax:	or p1,p2
Parameters:	p1 = ACCU [a,b,r] p2 = ACCU [a,b,r] p1 != p2
Calculus:	p1 : p1 | p2
Flags affected:	C O S Z
Bytes:	1
Description:	Bitwise OR (disjunction)
Category:	Bitwise operation

pop	Remove address
Syntax:	pop
Parameters:	-
Calculus:	-
Flags affected:	-
Bytes:	1
Description:	Roll back ram-address stack
Category:	RAM access

rst	Power On Reset
Syntax:	rst
Parameters:	-
Calculus:	-
Flags affected:	S Z
Bytes:	5
Description:	This is a symbolic opcode which is equivalent to the following instruction sequence: bitC 54 bitC 55 bitS 55 bitS 54 bitC 55 The assembler understands also the old powerOnReset
Category:	Miscellaneous

push	Put data into stack memory
Syntax:	push p1
Parameters:	p1 = NUMBER [6-bit]
Calculus:	-
Flags affected:	-
Bytes:	1
Description:	Writes p1 to RAM address stack (range: 0 to 63). Commit constant value to ROM routines. push 22 ; number of fractional digits for CDC ratios push 4 ; Port Number for reference value (PC4) jsb _ROM_CDC__ Note: for advanced users only. Better use rad
Category:	RAM access

rad	Set RAM Address Pointer
Syntax:	rad p1
Parameters:	p1 = NUMBER [7-bit]
Calculus:	-
Flags affected:	-
Bytes:	2
Description:	Set pointer to RAM address (range: 0 to 127) in the RAM address stack. Note: Internally the RAM is made of 2 pages, 64 words each. The assembler translates the combination of a bitS/C and a push instruction into the rad instruction. rad 15 move r, b will move the content of b the address 15
Category:	RAM access

wdr	Clear watch dog timer
Syntax:	wdr
Parameters:	-
Calculus:	-
Flags affected:	-
Bytes:	5
Description:	Clear watchdog timer. This is a symbolic opcode which is equivalent to the following instruction sequence: bitC 54 bitC 55 bitS 54 bitS 55 bitC 54 The assembler understands also the old resetWDG opcode
Category:	Miscellaneous

shiftL	Shift Left
Syntax:	shiftL p1
Parameters:	p1 = ACCU [a, b]
Calculus:	p1 : p1<< 1
Flags affected:	S Z
Bytes:	1
Description:	Shift p1 left --> shift p1 register to the left, fill LSB with 0, MSB is placed in carry register
Category:	Shift and rotate

shiftR	Shift Right
Syntax:	shiftR p1
Parameters:	p1 = ACCU [a, b]
Calculus:	p1: p1>> 1
Flags affected:	S Z
Bytes:	1
Description:	Signed shift right of p1 --> shift p1 right, MSB is duplicated according to whether the number is positive or negative
Category:	Shift and rotate

sign	Sign
Syntax:	sign p1

Parameters:	p1 = ACCU [a,b]
Calculus:	If SF = 0 => p1 : |p1|, SF : S(p1) If SF = 1 => p1 : - |p1|, SF : S(p1)
Flags affected:	S Z SF
Bytes:	1
Description:	The intention of this opcode is to take the absolute value of one parameter before multiplication or division and to restore the sign after this operation. Assuming the Signum flag is zero, the absolute value of accumulator is taken and the sign from accumulator is stored to SF. At the second time this opcode is used the sign to p1 will be restored from SF Zero is assumed to be positive.
Category:	Simple arithmetic

stop	Stop
Syntax:	stop
Parameters:	-
Calculus:	-
Flags affected:	-
Bytes:	1
Description:	Stop of the PCAP-Controller. The clock generator is stopped, the PCAP-Controller go to standby. A restart can be achieved by an external event like ‘watchdog timer’, ‘external switch’ or ‘new capacitive measurement results’. Usually this opcode is the last command in the assembler listing.
Category:	Miscellaneous

sub	Subtraction
Syntax:	sub p1,p2
Parameters:	p1 = ACCU [a,b,r] p2 = ACCU [a,b,r] p1 != p2
Calculus:	p1: p1 - p2
Flags affected:	C O S Z
Bytes:	1
Description:	Subtraction of 2 registers. The following instructions are not allowed: sub a,a. sub b,b. sub r,r
Category:	Simple arithmetic

xor	Bitwise XOR
Syntax:	xor p1,p2
Parameters:	p1 = ACCU [a,b,r] p2 = ACCU [a,b,r] p1 != p2
Calculus:	p1 : p1 ^ p2
Flags affected:	C O S Z
Bytes:	1
Description:	Bitwise XOR (antivalence)
Category:	Bitwise operation

7.6.5 Instruction Details

7.6.5.1 Pointer

```
; Copying the Cratio results to the persistent bank in RAM load b, 6
load       a,  __sub_cdc_C0_Ratio_temp rad  DPTR1
move   r, a
load       a, C0_Ratio_RAM rad   DPTR0
move   r, a
jsb        _ROM_dma__
```

rad _at_DPTR0 to rad _at_DPTR3 are special instructions for indirect addressing.

_at_DPTR0 to _at_DPTR3 are special RAM addresses 284 and 287 that have been defined in the firmware.

RAM addresses 105 to 108 are used as data pointers, named DPTR0 to DTPTR3.

By means of

```
rad       DPTR0
move      r, a
```

an address is loaded into DPTR0. With rad _at_DPTR0 the address in DPTR0 is loaded.

Example 1: copy sequentially RAM-content from one address-space to another

```
load      a, C0_ratio
rad       DPTR1
move      r, a
load      a, RES0
rad       DPTR0
move      r, a
load      b, 8
jsb       __ROM_dma__      ; call ROM routine
```

Example 2: Copying the Rratio results to the persistent bank into RAM

```
rad       4
          rad  rad_stack_6b
          move b, r
          load a, __sub_rdc_R0_Ratio_temp  ; Source  for copy rad     DPTR1
move   r, a

rad       R0_Ratio_RAM              ; Destination for copy rad
          rad_stack_6b
          move a, r
rad       DPTR0 move   r, a
jsb       _ROM_dma__
```

7.6.5.2 Call of a subroutine

Transfer Constants with Push and Pop

```
push      FPP_CRATIO                ; Stack - 1 ---> Number of fpp in result
push      C_REF_PORT_NUMBER         ; Stack - 0 ---> Reference Port Number
jsb       _ROM_cdc__                ; Calling ROM routine for Ratio   calculation

_ROM_cdc__:
bitS PAGESEL_OUT
rad       rad_stack_6b              ; (Stack -  0) contains Reference Port Number
move      b, r
pop
rad       __sub_cdc_RefPort
move      r, b                      ; Temporarily saving the Reference Port Number in RAM
pop
pop
rad       rad_stack_6b         ; (Stack - 1) contains number of fractional digits in the result
(cdc_fpp)
move      b, r                      ; B = The number of fractional digits ,    Result_fpp
```

7.6.5.3 mult

The instruction “mult” is just a single multiplication step. To do a complete 32-bit multiplication this instruction has to be done 32 times. The multiplicands are in accumulators b and r. Every step takes the lowest bit of b. If it is one, r is added to accumulator a, else nothing is added. Thereafter a and b are shifted right. The lowest bit of a becomes the highest bit of b. Before the first step of the multiplication, a has to be cleared. The final result is spread over both accumulators a and b.

The use of mult is simplified by using the ROM routines mult_ 01 to mult_32.

In many cases it will not be necessary to do the full 32 multiplication steps but much fewer. The necessary number of steps is given by the number of significant bits of b and also the necessary significant number of bits of the result.

But, if the multiplication steps are less than 32, the result might be spread between accumulators a and b. Doing an appropriate right shift of the multiplicand in r, and the appropriate number of multiplication steps, it is possible to ensure that the result is either fully in a or in b.

7.6.5.4 Handover of Constants by Push & Pop

A simple method to hand over constants with a value 0 to 63 is using push & pop instructions. The following shows an example for calling a subroutine.

Subroutine call:

```
push       FPP_CRATIO            ; Stack - 1 -> Number of fpp in the result
push       C_REF_PORT_NUMBER     ; Stack - 0 -> Reference Port  Number
jsb        _ROM_cdc__            ; Calling ROM routine for Ratio calculation

_ROM_cdc__:
rad        rad_stack_6b          ; (Stack - 0) contains Reference Port Number
move       b, r
pop
rad        __sub_cdc_RefPort
move       r, b                  ; Temporarily saving the Reference Port Number in RAM
pop
pop
rad        rad_stack_6b          ; (Stack - 1) contains number of fractional digits in the result
(cdc_fpp)
move       b,r                   ; B = The number of fractional digits , Result_fpp
```

7.6.5.5 div

The instruction “div” is, like the multiplication, just a single step of a complete division. The necessary number of steps for a complete division depends on the accuracy of the result. The dividend is in accumulator a, the divisor is in accumulator r. Every division step contains following actions:

- leftshift b
- compare a and r. If a is bigger or equal to r then r is subtracted from a and One is added to b
- leftshift a

Start Conditions: 0 < a < 2*r, b = 0

Again, multiple division steps are implemented in ROM library to be easily used by customers, calling div_01 to div_32. A call of function e.g. div_24 out of this library will do a sequence of 24 division steps. The result is found in b, the remainder in a.

With N division steps the result in b: $(a/r)+2^{(N-1)}$, a: remainder$*2^{N}$.

Example 1: a = 2, r = 6, Integer division

Table 90: Example: Division 2/6

Steps	a = 2	b	r = 6	
	000000..000010	0..00000	0..0110	a < r, leftshift b, a
1	000000..000100	0..00000	0..0110	a < r, leftshift b, a
2	000000..001000	0..00000	0..0110	leftshift b, a >= r: a-=r, b+=1, leftshift a
3	000000..000100	0..00001	0..0110	a < r, leftshift b, a
4	000000..001000	0..00010	0..0110	leftshift b, a >= r: a-=r, b+=1, leftshift a
5	000000..000100	0..00101	0..0110	

Quotient = b * $2^{(1\text{-steps})}$ = 0.3125, Remainder = a * $2^{(\text{-steps})}$ = 4 * 2^{-5} = 0.125

The following two, more complex examples show a nice advantage of division over multiplication: The resolution in bit is directly given by the number of division steps. With this knowledge, assembly programs can be written very effectively. It is easy to use only the number of division steps that is necessary.

Example 2: A = 8.75, R = 7.1875, Fractional number division, A & R with 4 fractional digits each.

$8.75/7.1875 = a*2^{expA} / r*2^{expR} = a * 2^{-4} / r * 2^{-4}$

Table 91: Example: Division 8.75/7.1875

Steps	a = 140	b	r = 115	
	1000 1100	0000 0000	0111 0011	leftshift b, a >= r: a-=r, b+=1, leftshift a
1	0011 0010	0000 0001	0111 0011	a < r, leftshift b, a
2	0110 0100	0000 0010	0111 0011	a < r, leftshift b, a
3	1100 1000	0000 0100	0111 0011	leftshift b, a >= r: a-=r, b+=1, leftshift a
4	1010 1010	0000 1001	0111 0011	leftshift b, a >= r: a-=r, b+=1, leftshift a
5	0110 1110	0001 0011	0111 0011	a < r, leftshift b, a
6	1101 1100	0010 0110	0111 0011	leftshift b, a >= r: a-=r, b+=1, leftshift a
7	1101 0010	0100 1101	0111 0011	leftshift b, a >= r: a-=r, b+=1, leftshift a
8	1011 1110	1001 1011	0111 0011	

Quotient = $b * 2^{(1+expA-expR-steps)} = 155 * 2^{(1-4+4-8)} = 1.2109$ Remainder = $a*2^{(-steps-expR)} = 190*2^{-12} = 0.0463$

Example 3: A = 20, R = 1.2, Fractional number division, R < A.

A and R are shifted to left to display the fractional digits of R. Further, R has to be shifted to the left till it is bigger than A/2.

$20/1.2 = a*2^{expA} / r*2^{expR} = a*2^{-4} / r*2^{-8}$

Table 92: Example: 20/1.2

Steps	a = 320	b	r = 307	
	0001 0100 0000	0000 0000 0000	0001 0011 0011	leftshift b, a >= r: a-=r, b+=1, leftshift a
1	0000 0001 1010	0000 0000 0001	0001 0011 0011	a < r, leftshift b, a
2	0000 0011 0100	0000 0000 0010	0001 0011 0011	a < r, leftshift b, a
3	0000 0110 1000	0000 0000 0100	0001 0011 0011	a < r, leftshift b, a
4	0000 1101 0000	0000 0000 1000	0001 0011 0011	a < r, leftshift b, a
5	0001 1010 0000	0000 0001 0000	0001 0011 0011	leftshift b, a >= r: a-=r, b+=1, leftshift a
6	0000 1101 1010	0000 0010 0001	0001 0011 0011	a < r, leftshift b, a
7	0001 1011 0100	0000 0100 0010	0001 0011 0011	leftshift b, a >= r: a-=r, b+=1, leftshift a
8	0001 0000 0010	0000 1000 0101	0001 0011 0011	a < r, leftshift b, a
9	0010 0000 0100	0001 0000 1010	0001 0011 0011	leftshift b, a >= r: a-=r, b+=1, leftshift a
10	0001 1010 0010	0010 0001 0101	0001 0011 0011	leftshift b, a >= r: a-=r, b+=1, leftshift a
11	0000 1101 1110	0100 0010 1011	0001 0011 0011	a < r, leftshift b, a
12	0001 1011 1100	1000 0101 0110	0001 0011 0011	

$Quotient = b * 2^{(1+expA-expR-steps)} = 2134 * 2^{(1-4+8-12)} = 16.6719$

The remainder is, as always, smaller than the denominator divided by 2steps e.g. in the present case, remainder < $1.2 / 2^{12} = 0{,}0003$

Steps = 1 + expA – expB – expRes

7.6.6 ROM Routines

The following routines are implemented as ROM code:

Table 93: ROM Routines

ROM Routine	Address		ROM Routine	Address
_ROM_Version	1024		_ROM_median	1804
_ROM_tdc_dispatch	1029		_ROM_rdc	1936
_ROM_cdc_cycle	1047		_ROM_rdc_inverse	1962
_ROM_rdc_cycle	1068		_ROM_cdc	2021
_ROM_cdc_initialize	1086			
_ROM_rdc_initialize	1111			
shiftR_B_32 to _00	1140			
shiftL_B_32 to _00	1173			
shiftR_A_32 to _00	1206		_ROM_memory_rd_a_32b to _ROM_memory_wr_08b	2470
shiftL_A_32 to _00	1239		_ROM_memory_store	2650
mult_32 to _00	1272		_ROM_memory_recall	2673
div_33 to _00	1305		_ROM_2PT_Calibration	2696
_ROM_div_variable	1339		_ROM_polynomial_3rd_degree	2825
_ROM_mult_variable	1396		sub_polynomial_load_coeffs	2874
_ROM_shift_a_variable	1452		_ROM_polynomial_4th_degree	2923
_ROM_shift_b_variable	1478		_ROM_pulse	2969
_ROM_dma	1504		_ROM_pulse_loaded_cal_vals	2990
_ROM_ln	1520		_ROM_NVblock_copy_32b_ to _ROM_NVblock_copy_08b_	3138
_ROM_log10	1547		_ROM_capacitance_polynomial	3236
_ROM_ld	1574		_ROM_capacitance_polynomial_4d	3437
_ROM_signed24_to_signed32	1648			

Handover parameters and RAM addresses for ROM routines

Table 94: Parameters and RAM for Key ROM Routines

ROM Routine	Parameter	RAM Address
_ROM_cdc	sub_cdc_C0_Ratio_temp to sub_cdc_C5_Ratio_temp	58 .. 63
_ROM_rdc _ROM_rdc_inverse	sub_rdc_R0_Ratio_temp to sub_rdc_R3_Ratio_temp	64.. 67
_ROM_mult_variable	sub_standard_multiplier	82
_ROM_div_variable	sub_standard_divisor	82

In the following we give a detailed description of the ROM routines. It is recommended to always include the following files to declare addresses to ROM routines:

```
#device PCap04v2                    ; former #device PCap04v1
#include <pcap_standard.h>
#include <PCap04_ROM_addresses_standard.h>
```

7.6.6.1 _ROM_Version__

Function:	This routine gives back the 8-bit version number of the ROM in the chip
Input parameters:	-
Output/Return value:	A-Accu: ROM version
Prerequisites	None
Dependency on other .h	None
Function call	`Jsb _ROM_version ; address 0x400`
Temporary memory usage	-

Example:	`jsb _ROM_Version__`
	`rad RES5`
	`move r, a`

7.6.6.2 _ROM_tdc_dispatch__

Function:	TDC library, to be called at the very beginning. Calls subroutines _ROM_cdc_ cycle or _ROM_rdc-cylcle and _tdc_awake , depending on the trigger
Input parameters:	-
Output/Return value:	-
Prerequisites	None
Dependency on other .h	None
Function call	`Jsb _ROM_tdc_dispatch__`
Temporary memory usage	-

Example:	`org 0`
	`__SOP__: ; Start of program`
	`jsb _ROM_tdc_dispatch__`

7.6.6.3 __u0_tdc_dispatch__

Function:	alternative to __tdc_dispatch__ to allow customer to collect all TDC-raw values independently.
Input parameters:	u0_NoValues (address Mi - 1) Number of raw values, depends on frontend settings
Output/Return value:	-
Prerequisites	None
Dependency on other .h	None
Function call	`jsb _ROM_tdc_dispatch__`
Temporary memory usage	u0_TDC_index_counter: RAM address u0_NoValues - 1

Example:	`CONST u0_Mfloat ; <start_address for TDC raw values,`
	`; eg. TM0 – 14>`
	`Org 0`
	`__SOP__: ; Start of program`
	`rad u0_Mfloat`
	`jsb _u0_tdc_dispatch__`
	`…`
	
	`MK_Initialize: ; Initialize procedure`
	`;…`
	`load b, 8 ;equal configured RAW values minus 1`
	`rad u0_NoValues`
	`move r, b`
	`;…`
	`jrt`

Ram offset of raw data in floating mode.

RAM Offset	_ROM_tdc_dispatch		_u0_tdc_dispatch	
0	M0+M1	PC0, PC1	M0	PC0
1	M2+M3	PC2, PC3	M1	PC1
2	M4+M5	PC4, PC5	M01	PC0, PC1
3	M01	PC0, PC1	M2	PC2
4	M23	PC2, PC3	M3	PC3
5	M45	PC4, PC5	M23	PC2, PC3
6	M6.1+M6.2	PC6	M6.1	PC6
7	Mi	-	Mi	Mi

7.6.6.4 _ROM_cdc_cycle_

Function:	Calculates the Start-Stop difference in the TDC values of the capacitance measurement ports and accumulates this in the measurement value RAM register (depending on the C_ADD_PTR) respective to the port being measured. If the calculated discharge time is too large, then the result is replaced with 0xFFFFFFFF thus indicating overflow. The DSP stops if there are more measurements to be done. In case of a last measurement, the DSP returns to where the ROM routine was called from continues to execute the post processing firmware, if any.
Input parameters:	-
Output/Return value:	The CDC discharge time measurement value is available in the respective RAM (88 - 95)
Prerequisites	This function has to be called after a measurement cycle at every capacitive port.
Dependency on other .h	None

Function call	`push <MSB(RAM_startaddress)>`
	`push <LSB(RAM startaddress)>`
	`jsb _ROM_cdc_cycle__`

Temporary memory usage	DPTR0 is overwritten in this routine.

Example:	`push 1 ; bit6 of starting address`
	`push M0 ; bit 5..0 of starting address`
	`jsb _ROM_cdc_cycle`
	`(not recommended, use _ROM_tdc_dispatch__ instead as this one calls _u0_cdc_cycle among others)`

7.6.6.5 _u0_cdc_cycle_

Function:	Alternative to _ROM_cdc_cycle
Input parameters:	-
Output/Return value:	The CDC discharge time measurement value is available in the respective RAM (88 - 95)
Prerequisites	This function has to be called after a measurement cycle at every capacitive port.
Dependency on other .h	None

Function call	`push <MSB(RAM_startaddress)>`
	`push <LSB(RAM startaddress)>`
	`jsb _u0_cdc_cycle__`

Temporary memory usage	DPTR0 is overwritten in this routine.

Example:	`push 1 ; bit6 of starting address`
	`push M0 ; bit 5..0 of starting address`
	`jsb _u0_cdc_cycle`
	`(not recommended, use _u0_tdc_dispatch__ instead. This one calls _u0_cdc_cycle among others)`

7.6.6.6 __ROM_Rdc_cycle__

Function:	Calculates the Start-Stop difference in the TDC values of the temperature measurement ports and accumulates this in the measurement value RAM register (depending on the R_ADD_PTR) respective to the port being measured. If the calculated discharge time is too large, then the result is replaced with 0xFFFFFFFF thus indicating overflow. If the end of an RDC measurement triggered the DSP, then this ROM routine call will stop the DSP after doing the above tasks. Else, the DSP returns to where the ROM routine was called and continues to execute the firmware.
Input parameters:	-
Output/Return value:	The RDC discharge time measurement value is available in the respective RAM (84 – 87)
Prerequisites	This function has to be called after a measurement cycle at every resistive port.
Dependency on other .h	None

Function call	`push <MSB(RAM_startaddress)>`
	`push <LSB(RAM startaddress)>`
	`jsb _ROM_rdc_cycle__`

Temporary memory usage	DPTR0 is overwritten in this routine.

Example:	`push  1  ; bit6 of starting  address`
	`push  TM0  ; bit 5..0 of starting address`
	`jsb  _ROM_rdc_cycle`
	`(not recommended, use _ROM_tdc_dispatch__ instead)`

7.6.6.7 __ROM_cdc_initialize__

Function:	This ROM routine clears all the measurement value RAM registers, Addresses 88-95 to 0.
Input parameters:	The starting address of the RAM registers (88) which are to be cleared, is to be pushed into the RAM address stack before calling this subroutine.
Output/Return value:	-
Prerequisites	-
Dependency on other .h	None

Function call	`push <MSB(RAM_startaddress)>`
	`push <LSB(RAM startaddress)>`
	`jsb _ROM_cdc_initialize__`

Temporary memory usage	The addresses DPTR0 and _at_DPTR0 are overwritten in this ROM routine.

Example:	`push  1  ; bit6 of starting  address`
	`push  M0  ; bit 5..0 of starting address`
	`jsb  _u0_cdc_initialize`

7.6.6.8 _u0_cdc_initialize__

Function:	Alternative to _ROM_cdc_initialize
Input parameters:	The starting address of the RAM registers (88) which are to be cleared, is to be pushed into the RAM address stack before calling this subroutine.
Output/Return value:	-
Prerequisites	-
Dependency on other .h	None

Function call	`push <MSB(RAM_startaddress)>`
	`push <LSB(RAM startaddress)>`
	`jsb _ROM_cdc_initialize__`

Temporary memory usage	The addresses DPTR0 and _at_DPTR0 are overwritten in this ROM routine.

Example:	`push 1 ; bit6 of starting address`
	`push M0 ; bit 5..0 of starting address`
	`jsb _u0_cdc_initialize`

7.6.6.9 _ROM_rdc_initialize__

Function:	This ROM routine clears all the measurement value RAM registers, Addresses 84-87 to 0.
Input parameters:	The starting address of the RAM registers (84) which are to be cleared, is to be pushed into the RAM address stack before calling this subroutine.
Output/Return value:	-
Prerequisites	None
Dependency on other .h	None

Function call	`push <MSB(RAM_startaddress)>`
	`push <LSB(RAM startaddress)>`
	`jsb _ROM_rdc_initialize__`

Temporary memory usage	The addresses DPTR0 and _at_DPTR0 are overwritten in this ROM routine.

Example:	`push 1 ; bit6 of starting address`
	`push TM0 ; bit 5..0 of starting address`
	`jsb _ROM_rdc_initialize`

7.6.6.10 shiftL_A_xx; shiftR_A_xx; shiftL_B_xx; shiftR_B_xx

Function:	Function to call a fixed number xx of shift steps of the A-Accu or B-Accu to the left or right
Input parameters:	A-Accu or B-Accu: Value to shift xx times
Output/Return value:	shiftL_A_01 A : A * 2 ^ (1) shiftL_A_02 A : A * 2 ^ (2) shiftL_A_03 A : A * 2 ^ (3) shiftL_A_31 A : A * 2 ^ (31)
Prerequisites	None
Dependency on other .h	None

Function call	`jsb shiftL_A_01 jsb shiftR_A_01`
	`.....`
	`jsb shiftL_A_32 jsb shiftR_A_32`
	
	`jsb shiftL_B_01 jsb shiftR_B_01`
	`.....`
	`jsb shiftL_B_32 jsb shiftR_B_32`

Temporary memory usage	
Example:	`load b, dp_const_m` `jsb shiftL_B_08`

7.6.6.11 mult_xx

Function:	Function to call a variable no of multiplication steps
Input parameters:	B-Accu : Multiplier 1 R-Accu : Multiplier 2
Output/Return value:	A-Accu : Result of multiplication
Prerequisites	A-Accu : 0 B >= 0 R >= 0
Dependency on other .h	None

Function call	`jsb mult_01`
	`..`
	`jsb mult_32`

Temporary memory usage	

Example:	`sign b ; Store sign of multiplier b`
	`; take absolute value from b`
	`clear a`
	`rad __sub_standard_multiplier__`
	`jsb mult_15 ; a2 * theta in A-Akku`
	`sign a ;Restoring sign of the multiplier to the result`

7.6.6.12 div_xx

Function:	div_00 to div_33: Function to call a fixed no of division steps
Input parameters:	A-Accu: Dividend r : Divisor
Output/Return value:	B-Accu: Dividend / divisor
Prerequisites	B-Accu 0 0<'a'<2*'r' respectively result b needs to be 0 < b < 2
Dependency on other .h	None
Function call	`jsb div_xx`
Temporary memory usage	

Example:	`rad M_internal_ref`
	`move b, r`
	`shiftL b`
	`shiftL b ; b : b + 4, to make sure dividend > 2*a`
	`rad 0`
	`move r, b`
	`rad M0`
	`move a, r ; a = M0`
	`clear b`
	`rad 0 ; r = M_internal_ref`
	`jsb div_29 ; b = a/r = M0/M_internal_ref`

7.6.6.13 _ROM_div_variable__

Function:	Function to call a variable no of division steps
Input parameters:	B-Accu : No of division steps A-Accu : Dividend arguments - 0 : Divisor (RAM-address 82)
Output/Return value:	B-Accu : Dividend / divisor
Prerequisites	0<'a'<2*'r' respectively result b needs to be 0 < b < 2
Dependency on other .h	None
Function call	`jsb _ROM_div_variable__`
Temporary memory usage	

Example:	`load a, 3 ; divisor = 3`
	`rad _arguments__ ; hand over via ramaddress 82`
	`move r, a`
	`load a, 4 ; dividend = 4`
	`load b, 23 ; 23 division steps`
	`jsb _ROM_div_variable__;`
	`rad RES00`
	`move r, b`
	
	`; B : 0x555555__; B : 4/3 = 1.333333 (fpp22)`

7.6.6.14 __ROM_mult_variable__

Function:	Function to call a variable no of multiplication steps
Input parameters:	B-Accu : Multiplier 1 A-Accu : Number of multiplication steps arguments - 0 : Multiplier 2 (RAM-address 82)
Output/Return value:	A-Accu : Result of multiplication
Prerequisites	Multiplier 1 > 0 Multiplier 2 > 0
Dependency on other .h	None
Function call	`jsb _ROM_mult_variable__`
Temporary memory usage	

Example:	`load b, 3 ; multiplier 1 = 3`
	`rad _arguments__ ; hand over via ramaddress 82 move r, b`
	`load b, 4 ; multiplier 2 = 4`
	`load a, 32 ; 32 muliplication steps`
	`jsb _ROM_mult_variable__;`
	`rad RES00`
	`move r, b`
	
	`; B : 12__; B : 4*3 = 12 (fpp22)`

7.6.6.15 __ROM_shift_a_variable__

Function:	Function to call a variable no of shift A-Accu steps
Input parameters:	A-Accu : Value to shift B-Accu : Number of shift steps B > 0 : Left shift B < 0 : Right shift
Output/Return value:	A : A * 2 ^ (B)
Prerequisites	None
Dependency on other .h	None
Function call	`jsb _ROM_shift_a_variable__`
Temporary memory usage	

Example:	`rad MyVar ; a : MyVar;`
	`move a, r`
	`load b, -4 ; set up 4fold right shift`
	`jsb _ROM_shift_a_variable__`
	`rad RES00`
	`move r, a`

7.6.6.16 __ROM_shift_b_variable__

Function:	Function to call a variable no of shift B-Accu steps
Input parameters:	B-Accu : Value to shift A-Accu : Number of shift steps A > 0 : Left shift A < 0 : Right shift
Output/Return value:	B : B * 2 ^ (A)
Prerequisites	None
Dependency on other .h	None
Function call	`jsb _ROM_shift_b_variable__`
Temporary memory usage	

Example:	`rad MyVar ; b : MyVar;`
	`move b, r`
	`load a, 13 ; set up 13fold left shift`
	`jsb _ROM_shift_b_variable__`
	`rad RES00`
	`move r, b`

7.6.6.17 __ROM_dma__

Function:	„Direct Memory Access“ – This routine copies sequential RAM-content from one address-space to another. The number of RAM values to be copied can be specified.
Input parameters:	B-Accu :Number of values to be copied DPTR1 :Source RAM block address DPTR0 :Destination RAM block address
Output/Return value:	The contents, i.e. the specified number of values are copied from the source RAM block to the destination RAM block.
Prerequisites	-
Function call	`jsb ROM_dma__`
Temporary memory usage	-
RAM permanently changed?	Yes, the destination RAM block

Example	`; Copying the Cratio results to the persistent bank`
	`; in RAM (cells __sub_cdc_C0_Ratio_temp to __sub_cdc_C0_Ratio_temp +5 into cells C0_Ratio_RAM to C0_Ratio_RAM + 5)`
	`Load b, 6`
	`load a, __sub_cdc_C0_Ratio_temp`
	`rad DPTR1`
	`move r, a`
	`load a, C0_Ratio_RAM`
	`rad DPTR0`
	`move r, a`
	`jsb _ROM_dma__`

7.6.6.18 __ROM_ln__, __ROM_log10__, __ROM_ld__

Function:	Calculation of logarithm log10 (logarithm to base 10) ln (logarithm naturalis) ld (logarithm dualis, to base 2) AccuA = ld(AccuA) for all logarithms, first ld(x) is determined. For log10 and ln afterwards the return value is divided by ld(e(1)) or ld(10) formula for ld(x) of http://de.wikipedia.org/wiki/Logarithmus#Nat.C3.BCrlicher_Logarithmus
Input parameters:	A-Accu : Containing parameter for logarithm dualis with 11fpp
Output/Return value:	A-Accu : Signed 16bit Value
Prerequisites	

Function call	`jsb ROM_ld`
	`jsb ROM_ln__`
	`jsb ROM_log10__`

Temporary memory usage	__temporary_variables - 3 to temporary_variables_
RAM permanently changed?	

Example	`load a, 0xA01 ; A: 2.500977 (=2561*2^-10)`
	`jsb __ROM_ld__`
	`rad RES00`
	`move r, a ; A: 1.322492 (0x1528E*2^-16)`

7.6.6.19 __ROM_signed24_to_signed32__

Function:	Type cast from 24bit signed to 32bit signed. This function is to cast a 24bit signed value e to a 32 bit signed value.
Input parameters:	Accu B : Signed 24 bit value
Output/Return value:	Accu B : Signed 32bit value
Prerequisites	None
Dependency on other .h	None
Function call	`_ROM_signed24_to_signed32__`
Temporary memory usage	1
Example:	`Jsb _ROM_signed24_to_signed32__`

7.6.6.20 __ROM_median__

Function:	Median-filter: this is a quasi-median-filter. The depth of filter is defined by arguments via stack. Each new Value (X) will be compared with the current median value. Is the new value smaller or equal to the median value the last value at the list will be replaced by X. Otherwise the firstvalue at the list will be replaced by X. Afterwards the complete list is sorted. The value at the very middle of the list is returned as new median.
Input parameters:	B-Accu : Filter input (and output) Stack - 2 : Start address of filter memory Stack - 1 : Middle section of filter memory Stack - 0 : Last address of filter memory The filter order is defined by last address - start address. The necessary memory section has to be reserved for and must not be used otherwise
Output/Return value:	B-Accu: Filter output (and input)
Prerequisites	None
Dependency on other .h	None
Function call	`jsb _ROM_median__`
Temporary memory usage	__temporary_variables - 4 to temporary_variables FILTER_START to FILTER_STOP

Example:	`CONST FILTER_ORDER 5 CONST FILTER_START 12`
	`CONST FILTER_MIDDLE 12 + (FILTER_ORDER/2)`
	`CONST FILTER_STOP 12 + FILTER_ORER`
	`rad Filter_input_value move b, r`
	`push FILTER_START`
	`push FILTER_MIDDLE`
	`push FILTER_STOP`
	`jsb _ ROM_median__`

7.6.6.21 __ROM_cdc__

Function:	This routine contains the subroutine to determine the capacitor ratios (or inverse ratios); depending on the measurement scheme and compensation modes. Improved algorithm fpr floating mode in V2.
Input parameters:	Stack 1 : CDC_FPP number of fractional digits in the result. Stack 0 : C_REF_PORT_NUMBER reference port number

	(0 to 6 for grounded, 0 to 2 for floating). A-Accu : sub_cdc_gain_corr factor for gain correction, applied on Mi, 8fpp 0 0 : gain corr = 1 else 1+ (A-Accu) FLAG_CDC_INV (Bit 5) of FLAGREG: 0 : inverse capacitance ration, 1 : capacitance rations
Output/Return value:	The addresses sub_cdc_C0_Ratio_temp to sub_cdc_C5_Ratio_temp are updated with relevant capacitance ratios (or inverse ones)
Prerequisites	
Function call	`jsb _ROM_cdc__`
Temporary memory usage	Addresses 58 to 72

Example:	`load2exp a, FLAG_CDC_INV`
	`not a ; Clearing FLAG_CDC_INV (Bit 5) of FLAGREG`
	`rad FLAGREG`
	`and r, a`
	`load a, 0x40 ; default gain_corr 1.25`
	`rad 27 ; Stack - 1 ---> Number of fpp in the result`
	`rad 0 ; Stack - 0 -> Ref. port number`
	`jsb _ROM_cdc__ ; Calling ROM routine for Ratio`
	`; calculation rad __sub_cdc_C1_Ratio_temp`
	`move a, r ; save Ratio from CDC`

7.6.6.22 _ROM_rdc__ ; _ROM_rdc_inverse__

Function:	Subroutine to calculate the resistance ratios or inverse ratios for the temperature measurement ports TM0 -> Internal Reference Resistance Port; TM1 -> External Resistance Port, PT0 TM2 -> External Resistance Port, PT1 TM3 -> Internal Sensor Resistance Port
Input parameters:	Stack 0 : FPP_RRATIO fixed point position of result A-Accu :REF_PORT_NUMBER reference, 0 = internal 1=external
Output/Return value:	__sub_rdc_R0_Ratio_temp TM0/Ref or Ref/TM0 __sub_rdc_R1_Ratio_temp TM1/Ref or Ref/TM1 __sub_rdc_R2_Ratio_temp TM2/Ref or Ref/TM2 __sub_rdc_R3_Ratio_temp TM3/Ref or Ref/TM3
Prerequisites	None
Dependency on other .h	#include <memory.h>
Function call	`jsb _ROM_rdc__` `jsb _ROM_rdc_inverse__`
Temporary memory usage	__temporary_variables - 8 to temporary_variables_

Example:	`load a, 0 ; Reference Port Number = 0`
	`rad 25 ; Number of fpp in the result`
	`; A-Akku: Reference Port Number (0 ... 3)`
	`; Stack - 0: Value of fpp of result = __rdc_fpp__`
	`jsb _ROM_rdc__ ; Determine RDC ratios`
	`;jsb _ROM_rdc_inverse__ ; To determine inverse-RDC ratios`
	`rad __sub_rdc_R1_Ratio_temp ; save ratio from RDC`
	`move a, r`

7.6.6.23 _ROM_memory_ (read/write volatile memory)

Function:	Functions for memory NVRAM access

	read functions: * signed/unsinged, 8/16/24/32bit RAM into Accu A or B write functions: * 8/16/24/32 bit (write data from RAM cell mem_data) The address in RAM cell mem_add is automatically incremented with read/write operations.
Input parameters:	-
Output/Return value:	Accu-A or Accu-B contains the specified value from the NVRAM
Prerequisites	None
Dependency on other .h	#include <memory.h>
Function call	jsb _ROM_memory_rd_a_32b__ jsb _ROM_memory_rd_a_u24b__ ; jsb _ROM_memory_r jsb _ROM_memory_rd_a_u16b__ ; jsb _ROM_memory_r jsb _ROM_memory_rd_a_u08b__ ; jsb _ROM_memory_r jsb _ROM_memory_rd_b_32b__ jsb _ROM_memory_rd_b_u24b__ _ROM_memory_wr_32b__; _ROM_memory_wr_24b__ _ROM_memory_wr_16b__; _ROM_memory_wr_08b__
Temporary memory usage	
Example:	Single read: load a, <read_mem_add> ; mem_add : <read_mem_add> rad mem_add move r, a jsb _ROM_memory_rd_a_u24b__ Single write: load a, MEM_WE ; enable memory writing ;(disable write protection) rad mem_ctrl move r, a load a, <write_address> ; mem_add : <write_mem_add> rad mem_add move r, a load a, <data to store> ; mem_data : <data to store> rad mem_data move r, a jsb _ROM_memory_wr_32b__ ; optional: set write protection load a, MEM_WR_PROTECT rad mem_ctrl move r, a Autoincrement: load a, 900 ; address (decimal) 900 rad mem_add move r, a load a, 0x012345; data : 0x012345 rad mem_data move r, a jsb _ROM_memory_wr_24b__ ; add 900: 0x45 ; add 901: 0x23 ; add 902: 0x01 ; mem_add after this operation : 903

7.6.6.24 _ROM_MEMORY_RECALL/STORE

Function:	Functions for nonvolatile memory access All nonvolatile accesses are secured by RAM cell mem_ctrl. If a store or a recall should be activated this has to be enabled by dedicated codes to mem_ctrl to prevent such actions by accident. Store: mem_ctrl : MEM_STORE (0x2d00) Recall: mem_ctrl : MEM_RECALL (0x5900) Note: there is a bug in the ROM code, which appears in case the bit 7 is set before the _ROM_memory_store_ routine is called. Adding bitC 7 before the call solves the issue and the data is stored in the non-volatile part correctly.
Input parameters:	
Output/Return value:	
Prerequisites	None
Dependency on other .h	<memory.h>
Function call	`jsb _ROM_memory_store__` `jsb _ROM_memory_recall__`
Temporary memory usage	

Example	`Store:`
	`bitC  7`
	`load  a, MEM_STORE`
	`rad  mem_ctrl`
	`move  r, a`
	`jsb _  ROM_memory_store__`
	
	`Recall:`
	`load  a, MEM_RECALL`
	`rad  mem_ctrl`
	`move  r, a`
	`jsb _  ROM_memory_recall__`

7.6.6.25 _ROM_2pt_calibration

Function:	This function performs a two point calibration with the given set of input values. The mathematical formula of the function is given below (calculation of xi): (xi_at_ccp1 - xi_at_ccp2) xi = ---------- . (ci - ci_at_ccp1) + xi_at_ccp1 (ci_at_ccp1 - ci_at_ccp2) The input values, i.e., xi_at_ccp1, xi_at_ccp2, ci_at_ccp1, ci_at_ccp2 are defined in the calibration data
Input parameters:	A-Accu : |FPP_ci| - |FPP_ccp| FPP_ci : No. of fractional digits in the input capacitance ratio FPP_ccp : No. of fractional digits in the calibration value of capacitance ratio DPTR0 : Start address of the RAM containing the constants in 4 consecutive addresses in the following order: xi_at_ccp1, xi_at_ccp2, ci_at_ccp1, ci_at_ccp2 Stack-0 : RAM address of Input Capacitance Ratio ci NOTE: It is a must that ALL the 4 calibration values have the same fpp !!!
Output/Return value:	A-Accu :xi with FPP_ccp fractional digits
Prerequisites	All the four calibration values must have the same fpp.
Function call	_ROM_2pt_calibration__
Temporary memory usage	__temporary_variables - 8 to temporary_variables_

Example:	load a, FPP_difference; A Accu : (|FPP_ci| - |FPP_ccp|)
	rad xi_at_ccp1 ; Start address containing the list of
	; calibration values
	move b, r
	rad DPTR0 ; DPTR0 now contains starting address of the
	; calibration values
	move r, b
	rad C1_ratio
	
	; Inputs :
	; A Accu : (FPP_ci - FPP_ccp)
	; DPTR0 : Start address of the Calibration values in RAM
	; Stack-0 : RAM address of Input Capacitance Ratio ci
	; Output :
	; A Accu : xi with FPP_ccp fractional digits
	jsb _ROM_2PT_Calibration
	rad 2pt_result ; Storing the returned value in RAM
	move r, a

7.6.6.26 _ROM_polynomial_3rd_degree

Function:	This function calculates the third degree polynomial for a given input value, with known coefficients. The mathematical formula of the function is given below: an = ((cc3n / xi + cc2n)/xi + cc1n)/xi + cc0n where cc3n, cc2n, cc1n and cc0n are the coefficients of the third degree polynomial. xi is the given input value which can be a capacitance or resistance ratio for example. This can be used for capacitance and resistance polynomials of this form.
Input parameters:	DPTR0: start address of the memory containing the following : cc3n: 3rd degree coefficient cn_div3n: division steps for cc3n cc2n: 2nd degree coefficient cn_div2n: division steps for cc2n cc1n: 1st degree coefficient cn_div1n: division steps for cc1n cc0n: constant coefficient A-Accu: Input Capacitance or Resistance Ratio (or Inverse Ratio)
Output/Return value:	A-Accu: Result of the polynomial
Prerequisites	None
Dependency on other .h	None
Function call	`jsb _ROM_polynomial_3rd_degree`
Temporary memory usage	___temporary_variables_

Example:	The following example calculated temperature using the temperature polynomial
	`rad Ratio_temp ; Resistance ratio`
	`move a, r ; A-Akku contains R_ratio`
	
	`load b, cc3n_address ; Start address containing the list`
	`; of calibration values`
	`rad DPTR0 ; DPTR0 now contains starting address`
	`move r, b ; of the coefficient and steps list`
	
	`; Input : DPTR0: start address of the list memory`
	`; A-Akku: Input Capacitance or Resistance Ratio`
	`; (or Inverse Ratio)`
	`jsb _ROM_polynomial_3rd_degree ; A = theta`
	`rad theta`
	`move r, a`

7.6.6.27 _ROM_polynomial_4th_degree

Function:	This function calculates the third degree polynomial for a given input value, with known coefficients. The mathematical formula of the function is given below: an = (((cc4n / xi + cc3n)/xi + cc2n)/xi + cc1n)/xi + cc0n where cc4n, cc3n , cc2n, cc1n and cc0n are the coefficients of the third degree polynomial. xi is the given input value which can be a capacitance or resistance ratio for example. This can be used for capacitance and resistance polynomials of this form.
Input parameters:	DPTR0: start address of the memory containing the following : cc4n: 4th degree coefficient cn_div4n: division steps for cc4n cc3n: 3rd degree coefficient cn_div3n: division steps for cc3n cc2n: 2nd degree coefficient cn_div2n: division steps for cc2n cc1n: 1st degree coefficient cn_div1n: division steps for cc1n cc0n: constant coefficient A-Accu: Input Capacitance or Resistance Ratio (or Inverse Ratio)
Output/Return value:	A-Accu: Result of the polynomial
Prerequisites	None
Dependency on other .h	None
Function call	`jsb _ROM_polynomial_4th_degree`
Temporary memory usage	__temporary_variables -3 to temporary_variables_

Example:	The following example calculated temperature using the temperature polynomial
	`rad Ratio_temp ; Resistance ratio`
	`move a, r ; A-Akku contains R_ratio`
	
	`load b, cc4n_address ; Start address containing the list`
	`; of calibration values`
	`rad DPTR0 ; DPTR0 now contains starting address`
	`move r, b ; of the coefficient and steps list`
	
	`; Input : DPTR0: start address of the list memory`
	`; A-Akku: Input Capacitance or Resistance Ratio`
	`; (or Inverse Ratio)`
	`jsb _ROM_polynomial_4th_degree ; A = theta`
	`rad theta`
	`move r, a`

7.6.6.28 __ROM_pulse__

Function:	This function determines the pulse output, given two input co-ordinates (result_ 1, pulse_out_1) and (result_2, pulse_out_2) and the current result_n. The input coordinates are copied from the NVRAM, given the address and must fulfill the pre-requisite. The result_n, result_1 and result_2 can be the result from the capacitance measurement (like Humidity, Pressure etc) or the temperature measurement (theta). The "result_n" is converted into "Pulse_out" which can then be assigned to PULSE0 or PULSE1 output. The mathematical formula of the function is given below: Pulse_out = (pulse_out_1 - pulse_out_2) ---------- * (result_n - result_1) + (pulse_out_1) (result_1 - result_2) Additionally Pulse_out is limited to the range between pulse_out_min and pulse_out_max. The constants pulse_out_1, pulse_out_2, result_1, result_2, pulse_out_min and pulse_out_min are defined in the calibration memoryand have to be copied to the RAM before calling this function. !!! **NOTE :** result_n and result_1 must have the same format.
Input parameters:	A-Accu: Value of result_n B-Accu: 10-bit NVRAM Start Address containing the list of calibration values
Output/Return value:	A-Accu: Pulse_out value as integer
Prerequisites	result_1: 4 bytes (Same fpp as result_2 and result_n) result_2 : 4 bytes (Same fpp as result_1 and result_n) pulse_out_1: 2 bytes (Integer) pulse_out_2: 2 bytes (Integer) pulse_out_max: 2 bytes (Integer) pulse_out_min: 2 bytes (Integer)
Dependency on other .h	-
Function call	`jsb _ROM_pulse__`
Temporary memory usage	__temporary_variables - 8 to temporary_variables_

Example:	`rad Z_result ; Value to be given as pulse output`
	`move a, r`
	
	`load b, result1_NVaddress ; Starting address in NVRAM`
	`; containing constants`
	
	`; Input: B-Accu: 10 bit NVRAM Address`
	`; A-Accu: Value of result_n`
	`jsb _ROM_pulse__ ; Output in A-Accu is an integer`
	`rad PULSE0 ; Pulse output on PULSE0`
	`move r, a`

7.6.6.29 _ROM_pulse_loaded_cal_vals

Function:	This ROM routine has the same functionality as the _ROM_pulse routine. Only difference is that the constant values have to be copied from the calibration NVRAM memory to the RAM by the firmware, before this routine is called.
Input parameters:	A-Akku: Value of result_n DPTR0: Start address of the RAM containing the constant values in 6 consecutive addresses in the following order: result_1 (Same fpp as result_2 and result_n) result_2 Same fpp as result_1 and result_n) pulse_out_1 (Integer) pulse_out_2 (Integer) pulse_out_max (Integer) pulse_out_min (Integer)
Output/Return value:	A-Akku: Pulse_out value as integer
Prerequisites	The coordinate values must be copied from the NVRAM to a RAM space by the firmware.
Dependency on other .h	-
Function call	`jsb _ROM_ pulse_loaded_cal_vals`
Temporary memory usage	__temporary_variables - 8 to temporary_variables_

Example:	`rad Z_result ; Value to be given as pulse output`
	`move a, r`
	
	`load b, result1_RAMaddress ; Starting address in RAM`
	`; containing constants`
	
	`rad DPTR0 move r, b`
	`; Input : DPTR0: Starting RAM Address`
	`; A-Accu : Value of result_n`
	`jsb _ROM_pulse_loaded_cal_vals`
	`; Output in A-Accu is an integer`
	`rad PULSE0 ; Pulse output on PULSE0`
	`move r, a`

7.6.6.30 _ROM_NVblock_copy__

Function:	Copy a block of data from NVRAM to RAM
Input parameters:	DPTR0: Start address of the RAM B-Accu : Starting address of the NVRAM A-Accu: Count of the number of values to be copied SIGNED_VALUE_NV in FLAGREG must be set (for reading signed values) or cleared (for reading unsigned values) - relevant only for 24/16/08 bit values
Output/Return value:	RAM contains a copy of the specified number of values from the NVRAM
Prerequisites	None
Dependency on other .h	None

Function call	`jsb _ROM_NVblock_copy_32b_;`
	`jsb _ROM_NVblock_copy_24b_;`
	`jsb _ROM_NVblock_copy_16b_;`
	`jsb _ROM_NVblock_copy_08b_;`

Temporary memory usage	___temporary_variables_

Example:	This example copies 4 values, 32 bits each from the NVRAM to the RAM address starting at xi_at_ccp1:
	`load a, RAM_address`
	`; DPTR0 <-- starting RAM address rad DPTR0`
	`move r, a`
	`load2exp a, 2 ; Count = 4`
	`load b, NVRAM_address ; Starting NVRAM`
	
	`; Subroutine to copy values from NVRAM -> RAM,`
	`;Returns current NVRAM address in B-Akku`
	`jsb _ROM_NVblock_copy_32b_`

7.6.6.31 _ROM_capacitance_polynomial

Function:	This function calculates the value of the capacitance polynomial, Z_result = (((a2 * theta)+ a1) * theta)+a0 given the capacitance ratio or inverse (Cratio) and temperature (theta). a2, a1 and a0 are 3rd degree polynomials of the Cratio value an = ((cc3n / xi + cc2n)/xi + cc1n)/xi + cc0n where cc3n .... cc0n are the coefficients of the third degree polynomial in the NVRAM
Input parameters:	DPTR0: Address of the capacitance ratio or inverse(Cratio) DPTR1: Address of the temperature (theta) B-Accu Starting address of coefficient values in NVRAM Arg_6: Z_min Arg_7 : Z_max FLAGREG, Bit 7 (LIN_3BYTE_COEFF) : 1 -> 3 bytes in each of the coefficients 0 -> 4 bytes in each of the coefficients
Output/Return value:	A-Accu: Z_result = result
Prerequisites	-
Function call	`Jsub _ROM_capacitance_polynomial__`
Temporary memory usage	__temporary_variables - 8 to temporary_variables Arg_0 to Arg_7

Example:	`;  Copying Z_min and Z_max to  Arg_6 and  Arg_7`
	`load  a, Arg_6 ; DPTR0 <-- starting argument`
	`; memory address (RAM)`
	`rad  DPTR0`
	`move  r, a`
	`load2exp  a, 1`
	`load  b, NV_Cal_vals  ; Starting address of Calibration`
	`; values in  NVRAM`
	`jsb  _ROM_NVblock_copy_32b_  ;  Subroutine to`
	`; and  Z_max values from NVRAM -> argument  RAM`
	`;---------------------`
	`load  a, C1_ratio`
	
	`rad  DPTR0  ; DPTR0: Address of  the capacitance`
	`; ratio or inverse(Cratio)`
	`move  r, a`
	`load  a, theta  ; DPTR1: Address of  the temperature`
	`rad  DPTR1`
	`move  r, a`
	`load  b,  NV_QUADRATIC_COEFFS  ; Address of`
	`;in NVRAM  in B-Accu`
	`jsb  _ROM_capacitance_polynomial`
	`rad  Z_result`
	`move  r, a`

7.6.6.32 _ROM_capacitance_polynomial_4d

Function:	This function calculates the value of the capacitance polynomial, Z_result = (((a2 * theta)+ a1) * theta)+a0 given the capacitance ratio or inverse (Cratio) and temperature (theta). a2, a1 and a0 are 4th degree polynomials of the Cratio value an = (((cc4n / xi + cc3n)/xi + cc2n)/xi + cc1n)/xi + cc0n where cc4n .... cc0n are the coefficients of the fourth degree polynomial in the NVRAM
Input parameters:	DPTR0: Address of the capacitance ratio or inverse(Cratio) DPTR1: Address of the temperature (theta) B-Accu Starting address of coefficient values in NVRAM Arg_6: Z_min Arg_7 : Z_max
Output/Return value:	A-Accu: Z_result = result of the polynomial
Prerequisites	-
Function call	`jsub _ROM_capacitance_polynomial__`
Temporary memory usage	__temporary_variables - 8 to temporary_variables Arg_0 to Arg_7

Example:	`;  Copying Z_min and Z_max to Arg_6 and Arg_7`
	`load  a,  Arg_6  ; DPTR0 <-- starting argument`
	`; memory  address (RAM)`
	`rad  DPTR0`
	`move  r, a`
	`load2exp  a, 1  ; Count =  2`
	`load  b, NV_Cal_vals  ; Starting address of Calibration`
	`; values in  NVRAM`
	`jsb  _ROM_NVblock_copy_32b_ ;  Subroutine to copy Z_min and`
	`Z_max values from NVRAM -> argument RAM`
	`;--------------------`
	`load  a, C1_ratio`
	
	`rad  DPTR0  ; DPTR0: Address of  the capacitance`
	`; ratio or inverse(Cratio)`
	`move  r, a`
	`load  a, theta  ; DPTR1: Address of  the temperature`
	`rad  DPTR1`
	`move  r, a`
	`load  b,  NV_QUADRATIC_COEFFS  ; Address of`
	`;in NVRAM  in B-Accu`
	`jsb  _ROM_capacitance_polynomial_4d`
	`rad  Z_result`
	`move  r, a`

7.6.7 Assembly Programs

The PCap04 assembler is a multi-pass assembler that translates assembly language files into HEX files as they will be downloaded into the device. For convenience, the assembler can include header files. The user can write his own header files but also integrate the library files as they are provided by ScioSense.

The assembly program is made of many statements which contain instructions and directives. In the former section we explained the instructions in detail. In the following sections we describe the directives and some sample code.

Each line of the assembly program can contain only one directive or instruction statement. Statements must be contained in exactly one line.

Symbols

A symbol is a name that represents a value. Symbols are composed of up to 31 characters from the following list:

A - Z, a - z, 0 - 9, _

Symbols are not allowed to start with numbers. The assembler is case sensitive, so care has to be taken for this.

Numbers

Numbers can be specified in hexadecimal or decimal. Decimal have no additional specifier. Hexadecimals are specified by leading “0x”.

Expressions and Operators

An expression is a combination of symbols, numbers and operators. Expressions are evaluated at assembly time and can be used to calculate values that otherwise would be difficult to be determined.

The following operators are available with the given precedence:

Table 95: Operators

Level	Operator	Description
1	()	Brackets, specify order of execution
2	* /	Multiplication, Division
3	+ —	Addition, Subtraction

Example:

```
CONST       value   1
Equal       ((value + 3)/2)
```

Directives

The assembler directives define the way the assembly language instructions are processed. They also provide the possibility to define constants, to reserve memory space and to control the placement of the code. Directives do not produce executable code.

The following table provides an overview of the assembler directives.

Table 96: Directives

Directive	Description	Example
CONST	Constant definition, CONST [name] [value] value might be a number, a constant, a sum of both	CONST Slope 42 CONST Slope constant + 1
LABEL:	Label for target address of jump instructions. Labels end with a colon. All rules that apply to symbol names also apply to labels.	jsb LABEL1 LABEL1: ...
;	Comment, lines of text that might be implemented to explain the code. It begins with a semicolon character. The semicolon and all subsequent characters in this line will be ignored by the assembler. A comment can appear on a line itself or follow an instruction.	; this is a comment
org	Sets a new origin in program memory for subsequent statements.	org 0x23 equal 0x332211 ; write 0x11 to address 0x23, ; 0x22 to address 0x24 ...
equal	Insert three bytes of user defined data in program memory, starting at the address as defined by org.	
#device	Directive definition, #device [dev_name] might be the name of the used device. Defines which Library is used for the assembler and defines the subdirectory \lib\[dev_name] for the included header or library file.	#device PCap03-Z
#include	Include the header or library file named in the brackets < > or quotation marks " ". The code will be added at the line of the include command. Names in brackets refer to the ScioSense library with the defined subdirectory \lib\[dev_name]. Without using #device directive it refers to the fixed subdirectory \lib. In quotation marks the might be just the file name in case it is in the same folder as the program, but also the complete path.	#include <rdc.h> #include "rdc.h"
#ifdef #elseif #endif	Directive to implement code or not, depending on the value of the symbol following the #ifdef directive. Use e.g. to include header files only once into a program.	#ifdef standard_h #else #define standard_h ... #endif
#define	Defines a symbol that will be interpreted as true when being analysed by the #ifdef directive	

7.6.8 Sample Code

In the following we show some sample code for programming loops in the various kinds, for the use of the load instruction and the rotate instruction.

7.6.8.1 "for" Loop

Table 97: For Loop

Assembler	C-Equivalent	Comment
load a, max	for(index=-max; index < 0; index++)	max : number of repetitions
not a	{..}	2nd complement for max (~max+1)
inc a		
rad index		store (-max) to index
move r, a		
do:		loop body
;{..}		
rad index		
inc r		loop increment
jCarC do		repeat while index < 0

7.6.8.2 "while" Loop

Table 98: While Loop

Assembler	C-Equivalent	Comment
do:	while (expression)	
rad expression	{..}	
move a, r		activate Status Flags for „expression".
jEQ done		Jump if expression == 0
;{..}		loop body
clear a		unconditional jump without writing to program counter stack
jEQ do		
done;		

7.6.8.3 "do - while" Loop

Table 99: Do-While Loop

Assembler	C-Equivalent	Comment
do: ;{..} rad expression move a, r jNE do	do {..} while (expression) 	 loop body activate Status Flags jump if expression != 0

7.6.8.4 Rotate Right A to B

To rotate a value right from Akku A to Akku B, Akku B and R must be set to zero. Afterwards with each mult command a single „rotate right from A to B" is done. This function could be used e.g. to shift a 8-bit value to the highest byte in the register.

Table 100: Rotate

Assembler	C-Equivalent	Comment
load a, 0xa3 clear b move r, b mult ; (8x) mult … mult	A = <U8bC> b = a << 40	

7.6.8.5 “do - while” with 2 pointers

Table 101: Do-While Loop with Pointers

Assembler	C-Equivalent	Comment
load a, MW7	loopLimit = *MW7	load max-address for ptrSource
rad loopLimit		
move r, a		
load a, MW0	ptrSource = *MW0;	load ptrSource with source address
rad DPTR0		
move r, a		
load a, RES0	ptrSink = *Res0;	load ptrSink with sink address
rad DPTR1		
move r, a		
do:		loop body
rad _at_DPTR0	do { *ptrSink++ = *ptrSource++ }	load value from source
move a, r		
rad _at_DPTR1		write value to sink
move r, a		
rad loopLimit		write max-address to a
move a, r		
rad DPTR1		
inc r		increment sink address
rad DPTR0		
inc r		increment source address
sub a, r		limitLoop - ptrSource
jCarS do		repeat loop if ptrSource <= max-address
	while (ptrSource <= MW7)	

7.6.9 Libraries

The PICOCAP assembler offers the possibility to implement library files. With these libraries the firmware can be written in a modular manner. Common library files are for definitions of variable and constant names.

When the DSP has to be programmed by the user for a specific application or when the firmware ought to be modified, these library functions can be simply integrated into the application program without any major tailoring. They save programming effort for known, repeatedly used, important functions. Some library files are interdependent on other file(s) from the library.

The library functions are called header files (they have *.h extension) in the assembler software and have to be included in the main *.asm program. The path for the library files should be \lib\[dev_name] in the folder where the assembler is.

The following are the device related header files that we supply together with the assembler as part of the evaluation kit:

- pcap_standard.h
- PCap04_ROM_addresses_standard.h
- pcap_config.h

The input parameters, output parameters, effect on RAM contents etc. for each of these library functions are explained in the tables below.

Note: In the standard firmware and in all the library files, the notation “ufdN” is used as a comment. This shows if the parameter is signed or unsigned and the number of fractional digits in the number, N. For e.g. ufd21 indicates that the parameter is an unsigned fixed point number with 21 fractional digits. If the u at the beginning is missing, it is a signed number.

7.6.9.1 pcap_standard.h

Function:	This is a standard library for PCap04 firmware projects. It contains the major address mappings and constant names for the PCap04. Note: This file should be always included. It contains no commands, so no pro- gram space is wasted
Definitions (examples):	… ;- Temp. Variables and Arguments Def. ---------------*/ CONST __arguments__ 82 CONST __number_of_arguments__10 CONST __temporary_variables____arguments__ - __number_of_arguments__ CONST Arg_0 __arguments__ - 8 CONST Arg_1 __arguments__ - 7 … ; - RAM-Addresses OUT (&IN) ------------------------*/ CONST FLAGREG96 CONST RES00 97 CONST RES01 98 CONST RES02 99 …

7.6.9.2 PCap04_ROM_addresses_standard.h

Function:	This file declares the jump labels for ROM routines
Definitions (examples):	… CONST _ROM_dma__0x5d1; 1489 CONST _ROM_ln__0x5e1 ; 1505 CONST _ROM_log10__0x5fc; 1532 … CONST _ROM_cdc__0x7d4; 2004 CONST __sub_cdc_C0_Ratio_temp0x3A; 58 CONST __sub_cdc_C1_Ratio_temp0x3B; 59 …

7.6.9.3 pcap_config.h

Function:	Configuration register addresses and names for pcap04. It is for convenience, contains no commands and so no program space is wasted.

Definitions (examples):	`CONST CFG_ADD_OFFSET 960 ; Start address of config in NVRAM`
	`; -------- Register 13`
	`CONST CFG_ADD_C_TRIG_SEL 13 + CFG_ADD_OFFSET`
	`; -------- Register 42`
	`CONST CFG_ADD_EXTERNAL_FLAGS 42+ CFG_ADD_OFFSET`
	`CONST CFG_BM_C_MEDIAN_EN0x01 ; Bit Mask for C_MEDIAN_EN`
	`; Enable median filter for CDC values`
	`...`

8 Application Information

8.1 Schematic

Pcap04 needs only a few external components for operation. Of importance is a sufficient buffering of the supply voltage. We recommend 10 µF for VDD33 and 4.7 µF for VDD18. A simple RC network may be used for integration of the PDM outputs to generate analog output signal.

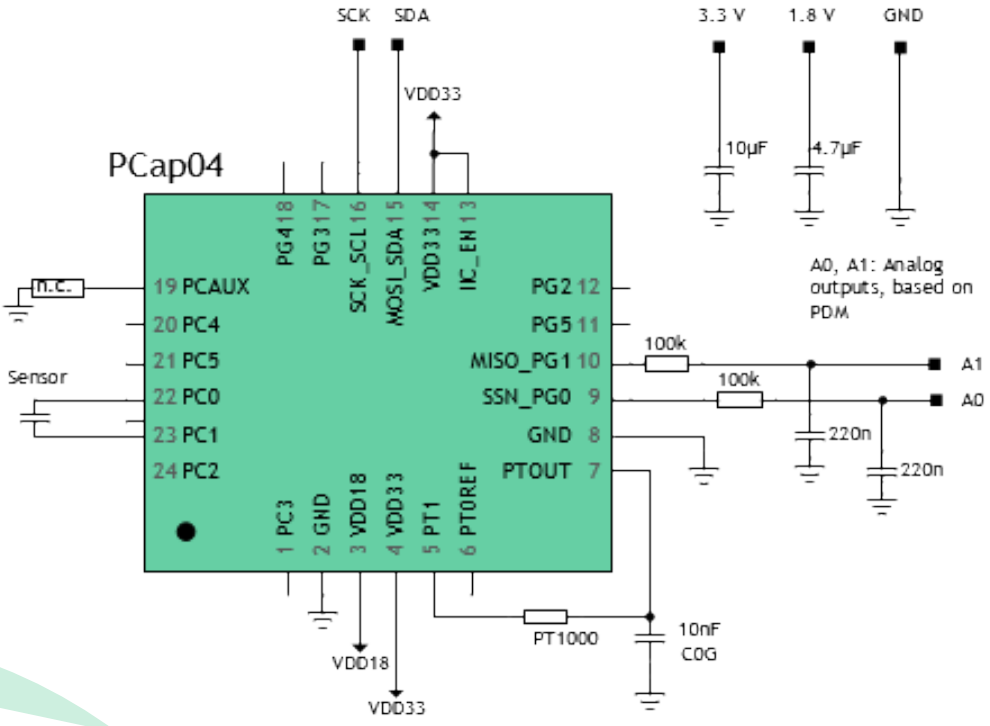


Figure 37: Typical Schematics with I²C Interface and PDM Analog Outputs

8.2 Minimized Bonding

Pcap04 is designed that for compact one-sensor applications the die may be bonded on two sides only.

Minimum: GND: #3, 4, 21,22, VDD33: #7, 24, VDD18: #5, 6 have to be connected.

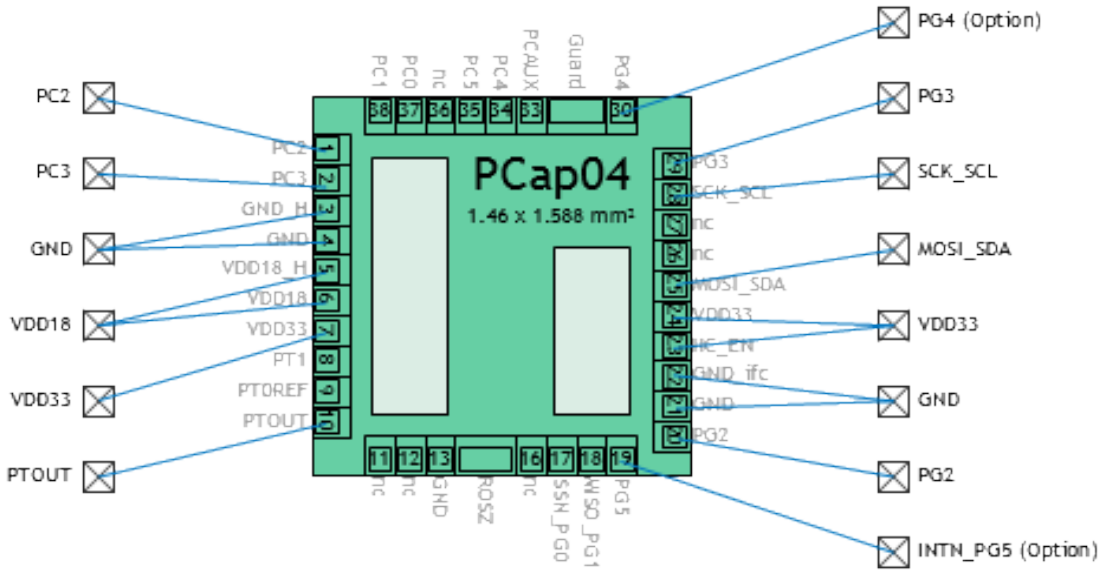
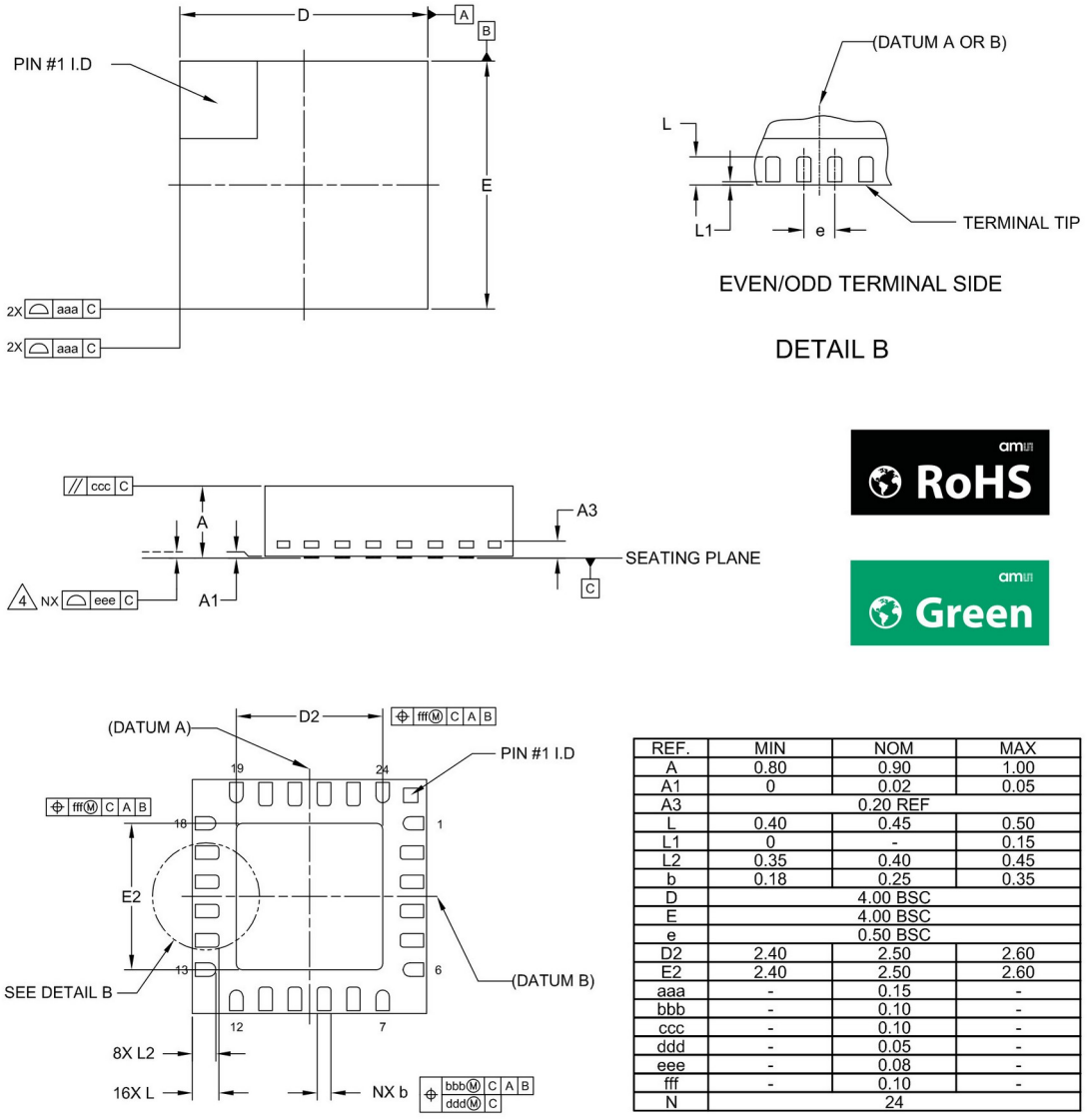


Figure 38: 2-Side Bonding

9 Package

9.1 Drawings & Markings



REF.	MIN	NOM	MAX
A	0.80	0.90	1.00
A1	0	0.02	0.05
A3		0.20 REF	
L	0.40	0.45	0.50
L1	0	-	0.15
L2	0.35	0.40	0.45
b	0.18	0.25	0.35
D		4.00 BSC	
E		4.00 BSC	
e		0.50 BSC	
D2	2.40	2.50	2.60
E2	2.40	2.50	2.60
aaa	-	0.15	-
bbb	-	0.10	-
ccc	-	0.10	-
ddd	-	0.05	-
eee	-	0.08	-
fff	-	0.10	-
N		24	

Figure 39: Package Drawing (QFN24)

Note(s):

1. Dimensioning and tolerancing conform to ASME Y14.5M-1994
2. All dimensions are in millimeters (angles are in degrees).
3. Dimension b applies to metallized terminal and is measured between 0.25mm and 0.30mm from terminal tip. Dimension L1 represents terminal full back from package edge up to 0.15mm is acceptable.
4. Coplanarity applies to the exposed heat slug as well as the terminal.
5. Radius on terminal is optional.
6. N id the total number of terminals.

Figure 40: Package Marking

Table 102: Package Code

PCAP04	T	PP	V2	YY	WW	X	ZZ
Part	Temperature A : -40°C to 125°C B : -40°C to 85°C	Package QF : QFN24	Silicon revision	Year	Week	Assembly Plant Identifier	Assembly Traceability Code

9.2 PCB Pad Layout

Caution: The center pad is internally connected to GND. No wires other than GND are allowed underneath. It is recommended to not solder the center pad. Too much solder paste could reduce solder quality. Suitable socket: e.g. Plastronics 32QN50S15050D

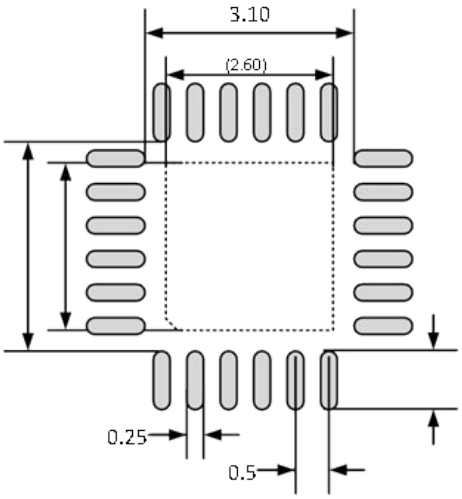


Figure 41: Landing Pattern (dimensions in mm)

9.3 Tape & Reel Information

PCAP04 is shipped in 7'' reels with 1000 chips per reel.

The tape-and-reel configuration is used for transport and storage from the manufacturer to the customer, and for use in the customer manufacturing plant. The configuration is designed for feeding components to automatic-placement machines for surface mounting on board assemblies. The complete configuration consists of a carrier tape with sequential individual cavities that hold individual components, and a cover tape that seals the carrier tape to retain the components in the cavities. Single reels are packed into dry-pack and inserted into intermediate boxes before shipping.

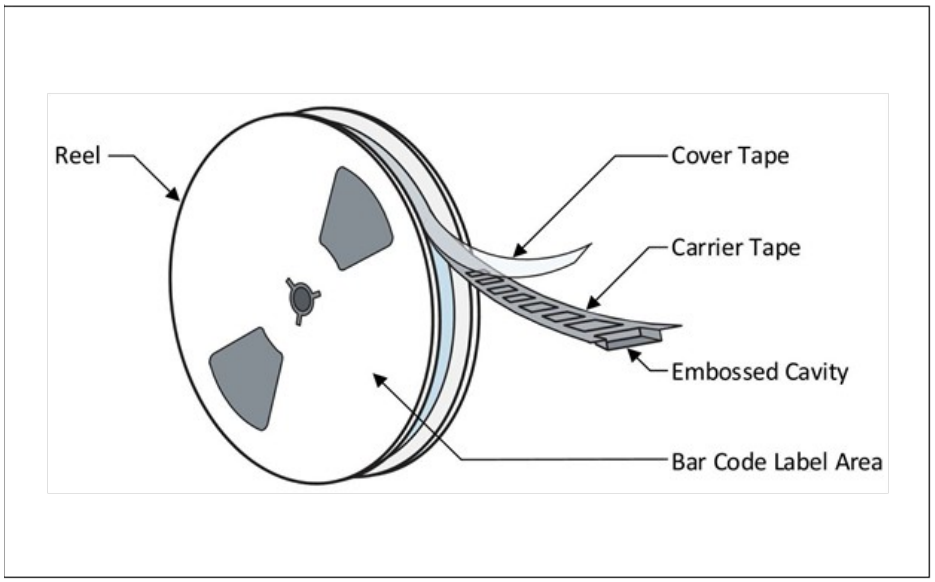


Figure 42: Reel

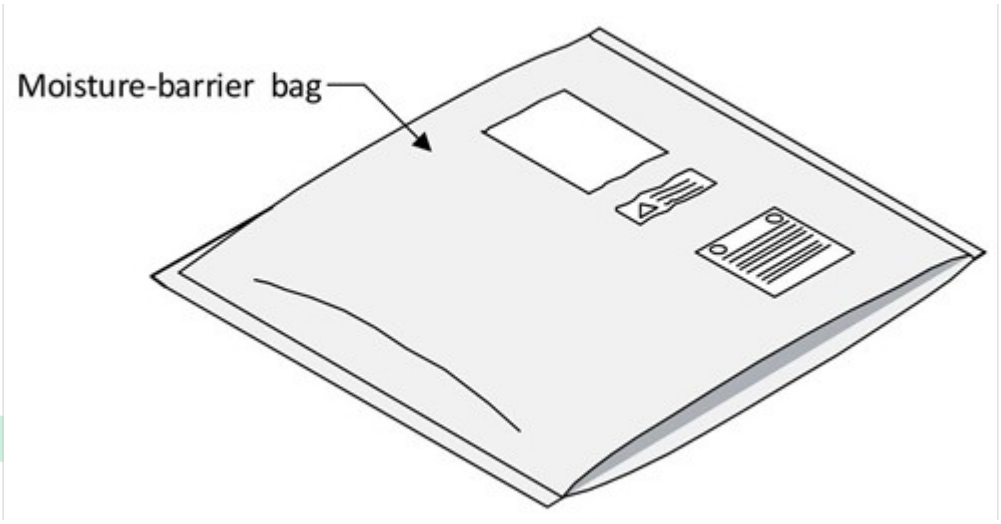


Figure 43: Single Reel in Dry Bag

Carrier tape is widely used for presenting devices to pick-and-place machines for automatic placement onto printed circuit boards.

A0 7.25
B0 7.25
K0 1.10

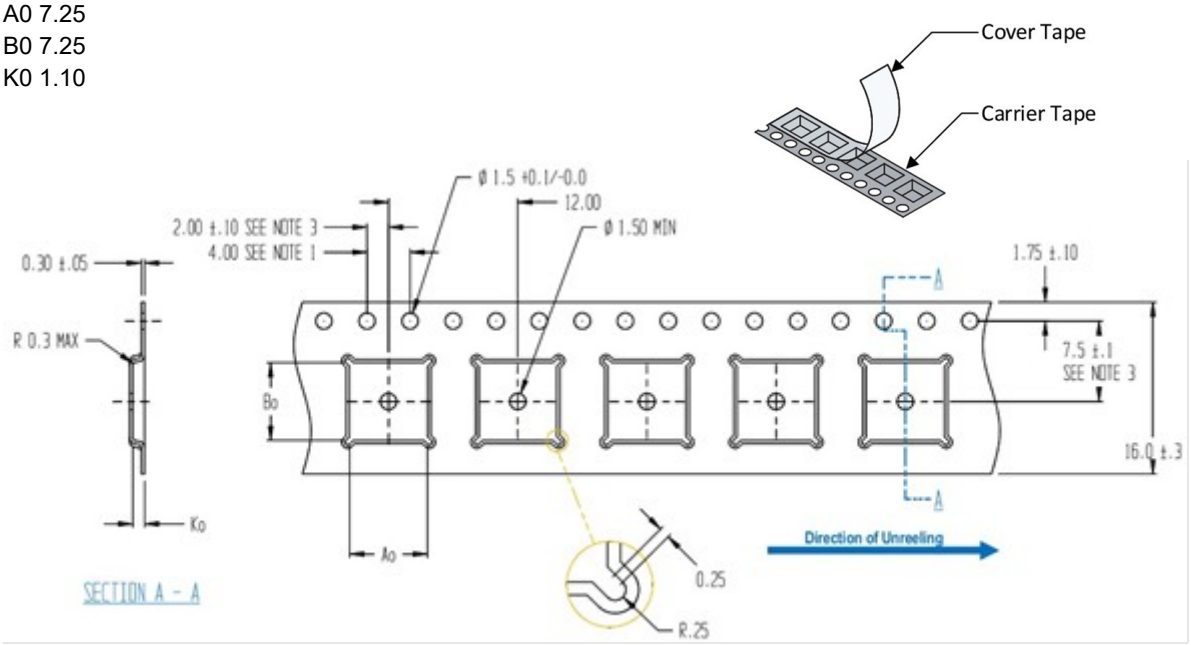


Figure 44: Carrier Tape QFN24

Note(s):

1. Sprocket hole pitch cumulative tolerance ±0.2
2. Camber in Compliance with EIA 481
3. Pocket position relative to sprocket hole measured as true position of pocket, not pocket hole
4. All dimensions in mm

10 Soldering information

IPC/JEDEC J-STD-020

The reflow peak soldering temperature (body temperature) is specified according to IPC/JEDEC J-STD-020 “Moisture/Reflow Sensitivity Classification for Non-hermetic Solid State Surface Mount Devices.” The lead finish for Pb-free leaded packages is “Matte Tin” (100% Sn)

11 Ordering information

Please note that the material ID will change with the switch from V1 to V2 as described in product change notification SC-001623-CN.

Table 103: Ordering information

Part Number	Description		Material ID	Delivery Form	Delivery Quantity
PCAP04-AQFM-24	PCAP04-AQFM-24 QFN24 LF T&RDP	V1[1] V2	502040014 502040033	7" T&R	1000 pcs
PCAP04-ASDW-100	PCAP04-ASDW-100 Dice in waffle pack (100 dies		502040029 502040030	Dice	100 dice
PCAP04-ASWB	PCAP04-ASWB Sorted wafer in box	V1[19] V2	502040013 502040034	Wafer Box	10000 dice
PCAP04-BQFM-24	PCAP04-BQFM-24 QFN24 LF T&RDP	V1[19] V2	502040021 502040039	7" T&R	1000 pcs
PCAP04-BSWB-290	PCAP04-BSWB Sorted wafer in box, 290µ inked		502040015	Wafer Box	10000[2] dice
PCAP04-BSWB-735	PCAP04-BSWB-735 Sorted wafer in box, 735µ (not grinded), not inked SW	V1[19] V2	502040028 502040035	Wafer Box	10000 dice
PCAP04-EVA-BOARD	PCap04-EVA-BOARD		220300002		
PCAP04-EVA-KIT	PCap04-EVA-KIT V1.0		220300003		

[1] V1 will be replaced by V2 as announced by product change notification SC-001623-CN
[2] The number of good dice varies from wafer to wafer and is in the order of 10500 dice.

12 RoHS Compliance & ScioSense Green Statement

RoHS: The term RoHS compliant means that Sciosense B.V. products fully comply with current RoHS directives. Our semiconductor products do not contain any chemicals for all 6 substance categories, including the requirement that lead does not exceed 0.1% by weight in homogeneous materials. Where designed to be soldered at high temperatures, RoHS compliant products are suitable for use in specified lead-free processes.

ScioSense Green (RoHS compliant and no Sb/Br): ScioSense Green defines that in addition to RoHS compliance, our products are free of Bromine (Br) and Antimony (Sb) based flame retardants (Br or Sb do not exceed 0.1% by weight in homogeneous material).

Important Information: The information provided in this statement represents Sciosense B.V. knowledge and belief as of the date that it is provided. Sciosense B.V. bases its knowledge and belief on information provided by third parties, and makes no representation or warranty as to the accuracy of such information. Efforts are underway to better integrate information from third parties. Sciosense B.V. has taken and continues to take reasonable steps to provide representative and accurate information but may not have conducted destructive testing or chemical analysis on incoming materials and chemicals. Sciosense B.V. and Sciosense B.V. suppliers consider certain information to be proprietary, and thus CAS numbers and other limited information may not be available for release.

13 Copyrights & Disclaimer

Devices sold by Sciosense B.V. are covered by the warranty and patent indemnification provisions appearing in its General Terms of Trade. Sciosense B.V. makes no warranty, express, statutory, implied, or by description regarding the information set forth herein. Sciosense B.V. reserves the right to change specifications and prices at any time and without notice. Therefore, prior to designing this product into a system, it is necessary to check with Sciosense B.V. for current information. This product is intended for use in commercial applications. Applications requiring extended temperature range, unusual environmental requirements, or high reliability applications, such as military, medical life-support or life-sustaining equipment are specifically not recommended without additional processing by Sciosense B.V. for each application. This product is provided by Sciosense B.V. “AS IS” and any express or implied warranties, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose are disclaimed.

Sciosense B.V. shall not be liable to recipient or any third party for any damages, including but not limited to personal injury, property damage, loss of profits, loss of use, interruption of business or indirect, special, incidental or consequential damages, of any

kind, in connection with or arising out of the furnishing, performance or use of the technical data herein. No obligation or liability to recipient or any third party shall arise or flow out of Sciosense B.V. rendering of technical or other services.

14 Document status

Table 104: Document status

Document Status	Product Status	Definition
Product Preview	Pre-Development	Information in this datasheet is based on product ideas in the planning phase of development. All specifications are design goals without any warranty and are subject to change without notice.
Preliminary Datasheet	Pre-Production	Information in this datasheet is based on products in the design, validation or qualification phase of development. The performance and parameters shown in this document are preliminary without any warranty and are subject to change without notice.
Datasheet	Production	Information in this datasheet is based on products in ramp-up to full production or full production which conform to specifications in accordance with the terms of ScioSense B.V. standard warranty as given in the General Terms of Trade.
Datasheet (Discontinued)	Discontinued	Information in this datasheet is based on products which conform to specifications in accordance with the terms of ScioSense B.V. standard warranty as given in the General Terms of Trade, but these products have been superseded and should not be used for new designs.

15 Revision information

Table 105: Revision history

Revision	Date	Comment	Page
V3	02 Nov 2018	V3 (former 1.03)	All
V4, V5		Internal versions only, not released	
V6	25 Sep 2023	In version V2 we removed the I2C error as described in error sheet ES000131. I2C is now working as described	60
		ROM routines _u0_tdc_dispatch, _u0_cdc_cycle and _u0_cdc_initialize added	112, 113, 114
		Cref calculation added	27
		Table 80 OLF_tune values corrected	69
		_ROM_MEMORY_STORE__ bug and solution descsribed.	124
		R_DCHG_SEL register 3 corrected	23
		_ROM_In.. routines corrected to 11fpp	120
		Section System Reset moved, Initialize preferred and difference hardware POR and software POR description added	59
		Improved algorithm in cdc.lib	121
		Wafer and dice thickness updated	4
		Guard not to be used with differential floating mode PRECHARGE_TIME > C_G_TIME	51, 52

Note(s) and/or Footnote(s):

1. Page and figure numbers for the previous version may differ from page and figure numbers in the current revision.
2. Correction of typographical errors is not explicitly mentioned.

Address: Sciosense B.V.
High Tech Campus 10
5656 AE Eindhoven
The Netherlands

Contact: www.sciosense.com
info@sciosense.com