PMS

Pressure Monitoring Sensor

SP27

High integrated single-chip PMS sensor with a low power embedded micro-controller

SP270 1300kPa Version A5

ROM Library Function Guide

Revision 1.0, 2012-02-07

Sense & Control

Edition 2012-02-07

Published by
Infineon Technologies AG
81726 Munich, Germany

Legal Disclaimer

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the device, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation, warranties of non-infringement of intellectual property rights of any third party.

Information

For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office.

Infineon Technologies components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

SP27 High integrated single-chip PMS sensor with a low power embedded micro-controller
CONFIDENTIAL
Revision History: 2012-02-07, Revision 1.0

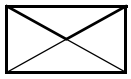
Previous Revision: Revision 0.1

Page	Subjects (major changes since last revision)
Page 39	Execution Time of the function Powerdown updated
Page 41	Execution Time of the function IntervalTimerCalibration updated
Page 50	ReadID Input and Output Parameters updated
Page 52	FW_Revision_Nb Input and Output Parameters updated
Page 57	Execution Time of the function FlashUserConfigSectorLine updated
Page 61	Ressource Usage of the function FlashSetLock updated
Page 61	Execution Time of the function FlashSetLock updated

We Listen to Your Comments

Any information within this document that you feel is wrong, unclear or missing at all?
Your feedback will help us to continuously improve the quality of this document.
Please send your proposal (including a reference to this document) to:

sensors@infineon.com



Co. TEAKLITE™ of CEVA, Inc. TEKTRONIX™ of Tektronix Inc. TOKO™ of TOKO KABUSHIKI KAISHA TA. UNIX™ of X/Open Company Limited. VERILOG™, PALLADIUM™ of Cadence Design Systems, Inc. VLYNQ™ of Texas Instruments Incorporated. VXWORKS™, WIND RIVER™ of WIND RIVER SYSTEMS, INC. ZETEX™ of Diodes Zetex Limited.

Last Trademarks Update 2011-02-24

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 General Considerations

This document describes the ROM Library functions that are available in the SP27 Version A5 step device. When the ROM Library Function **"FW_Revision_Nb()" on Page 52** is called, these devices will return the value $0A32_H$ for the ROM revision.

In order for the application to use these ROM Library functions, the following files have to be included to the software project:

- SP27_ROMLibrary.h (the header the function prototypes)
- SP27_ROMLibrary.lib (the precompiled ROM Library functions)

Notes

1. *The application must ensure that no IDLE-RESUME event source is active before any call of the ROM Library_functions.*
2. *All typical charge consumptions and typical execution times stated throughout this document were obtained from measurements of 10 devices at V_{CC} = 3.0 V and $T_{Ambient}$= 25°C*
3. *Analysis of General Register, SFR, and Stack resource usage was performed on object code generated by Keil uVision3 (Assembler version: 8.00d, Compiler version: 8.08, Linker version: 6.05, Hex converter version: 2.6.0.1).*

1.2 Type definitions

The following table defines the parameter types used throughout this document.

Table 1 Definition of types

Type	Description	Range	
		Minimum	Maximum
Unsigned char	8 bit value without sign bit	0	255
Signed char	8 bit value with sign bit	-128	127
Unsigned int	16 bit value without sign bit	0	65,535
Signed int	16 bit value with sign bit	-32,768	32,767
Unsigned long	32 bit value without sign bit	0	4,294,967,295
Signed long	32 bit value with sign bit	-2,147,483,648	2,147,483,647

The Keil C51 Compiler stores data in big endian format (MSB first).

1.3 Wakeup Handler

The Wakeup-Handler is executed every time the device wakes up from POWER DOWN state. Possible wakeup sources are listed in SP27 Datasheet **[1]**.

Table 2 Wakeup Handler

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Wakeup from POWER DOWN state (independent of Wakeup source)	t_{wakeup}	–	0.85	1.8	ms	

1.4 Restricted RAM and FLASH areas

The ROM Library functions use certain address areas in RAM and FLASH. To ensure that the ROM Library functions operate properly, the application may not use these locations for storage, nor alter the contents of these memory locations

1.4.1 Restricted RAM areas

The RAM area $C1_H$ through FF_H (upper 63 Bytes) of the 256 Bytes RAM is used and overwritten by the ROM Library functions.

1.4.2 Restricted FLASH areas

The FLASH address $57FF_H$ is reserved for Lockbyte 3. This value must not be changed by the application otherwise it might result in an unintentionally locked FLASH User Configuration Sector. Locking this sector is irreversible and shall only be done by either programming the Lockbyte 3 together with writing and locking the FLASH Code Sector or by a dedicated ROM Library function **FlashSetLock()**.

*Note: If **Erase_UserConfigSector()** or **WriteFlashUserConfigSectorLine()** are used by the application it has to be ensured that these restricted Flash locations are restored to the proper values.*

1.5 8051 Instruction Set Summary

As the SP27 incorporates an 8051 compatible microcontroller, **Figure 1** shows the SP27 OpCode Map.

Most Significant Nybble

Least Significant Nybble

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP 1/1	AJMP *page0* 2/2	LJMP *addr16* 3/2	RR *A* 1/1	INC *A* 1/1	INC *dir* 2/1	INC *@R0* 1/1	INC *@R1* 1/1	INC *R0* 1/1	INC *R1* 1/1	INC *R2* 1/1	INC *R3* 1/1	INC *R4* 1/1	INC *R5* 1/1	INC *R6* 1/1	INC *R7* 1/1
1	JBC *bit,rel* 3/2	ACALL *page0* 2/2	LCALL *addr16* 3/2	RRC *A* 1/1	DEC *A* 1/1	DEC *dir* 2/1	DEC *@R0* 1/1	DEC *@R1* 1/1	DEC *R0* 1/1	DEC *R1* 1/1	DEC *R2* 1/1	DEC *R3* 1/1	DEC *R4* 1/1	DEC *R5* 1/1	DEC *R6* 1/1	DEC *R7* 1/1
2	JB *bit,rel* 3/2	AJMP *page1* 2/2	RET 1/2	RL *A* 1/1	ADD *A,#imm* 2/1	ADD *A,dir* 2/1	ADD *A,@R0* 1/1	ADD *A,@R1* 1/1	ADD *A,R0* 1/1	ADD *A,R1* 1/1	ADD *A,R2* 1/1	ADD *A,R3* 1/1	ADD *A,R4* 1/1	ADD *A,R5* 1/1	ADD *A,R6* 1/1	ADD *A,R7* 1/1
3	JNB *bit,rel* 3/2	ACALL *page1* 2/2	RETI 1/2	RLC *A* 1/1	ADDC *A,#imm* 2/1	ADDC *A,dir* 2/1	ADDC *A,@R0* 1/1	ADDC *A,@R1* 1/1	ADDC *A,R0* 1/1	ADDC *A,R1* 1/1	ADDC *A,R2* 1/1	ADDC *A,R3* 1/1	ADDC *A,R4* 1/1	ADDC *A,R5* 1/1	ADDC *A,R6* 1/1	ADDC *A,R7* 1/1
4	JC *rel* 2/2	AJMP *page2* 2/2	ORL *dir,A* 2/1	ORL *dir,#imm* 3/2	ORL *A,#imm* 2/1	ORL *A,dir* 2/1	ORL *A,@R0* 1/1	ORL *A,@R1* 1/1	ORL *A,R0* 1/1	ORL *A,R1* 1/1	ORL *A,R2* 1/1	ORL *A,R3* 1/1	ORL *A,R4* 1/1	ORL *A,R5* 1/1	ORL *A,R6* 1/1	ORL *A,R7* 1/1
5	JNC *rel* 2/2	ACALL *page2* 2/2	ANL *dir,A* 2/1	ANL *dir,#imm* 3/2	ANL *A,#imm* 2/1	ANL *A,dir* 2/1	ANL *A,@R0* 1/1	ANL *A,@R1* 1/1	ANL *A,R0* 1/1	ANL *A,R1* 1/1	ANL *A,R2* 1/1	ANL *A,R3* 1/1	ANL *A,R4* 1/1	ANL *A,R5* 1/1	ANL *A,R6* 1/1	ANL *A,R7* 1/1
6	JZ *rel* 2/2	AJMP *page3* 2/2	XRL *dir,A* 2/1	XRL *dir,#imm* 3/2	XRL *A,#imm* 2/1	XRL *A,dir* 2/1	XRL *A,@R0* 1/1	XRL *A,@R1* 1/1	XRL *A,R0* 1/1	XRL *A,R1* 1/1	XRL *A,R2* 1/1	XRL *A,R3* 1/1	XRL *A,R4* 1/1	XRL *A,R5* 1/1	XRL *A,R6* 1/1	XRL *A,R7* 1/1
7	JNZ *rel* 2/2	ACALL *page3* 2/2	ORL *C,bit* 2/2*	JMP *@A+DPTR* 1/2	MOV *A,#imm* 2/1	MOV *dir,#imm* 3/2	MOV *@R0,#imm* 2/1	MOV *@R1,#imm* 2/1	MOV *R0,#imm* 2/1	MOV *R1,#imm* 2/1	MOV *R2,#imm* 2/1	MOV *R3,#imm* 2/1	MOV *R4,#imm* 2/1	MOV *R5,#imm* 2/1	MOV *R6,#imm* 2/1	MOV *R7,#imm* 2/1
8	SJMP *rel* 2/2	AJMP *page4* 2/2	ANL *C,bit* 2/2*	MOVC *A,@A+PC* 1/2	DIV *AB* 1/4	MOV *dir,dir* 3/2	MOV *dir,@R0* 2/2*	MOV *dir,@R1* 2/2*	MOV *dir,R0* 2/2*	MOV *dir,R1* 2/2*	MOV *dir,R2* 2/2*	MOV *dir,R3* 2/2*	MOV *dir,R4* 2/2*	MOV *dir,R5* 2/2*	MOV *dir,R6* 2/2*	MOV *dir,R7* 2/2*
9	MOV *DPTR,#imm* 3/2	ACALL *page4* 2/2	MOV *bit,C* 2/2*	MOVC *A,@A+DPTR* 1/2	SUBB *A,#imm* 2/1	SUBB *A,dir* 2/1	SUBB *A,@R0* 1/1	SUBB *A,@R1* 1/1	SUBB *A,R0* 1/1	SUBB *A,R1* 1/1	SUBB *A,R2* 1/1	SUBB *A,R3* 1/1	SUBB *A,R4* 1/1	SUBB *A,R5* 1/1	SUBB *A,R6* 1/1	SUBB *A,R7* 1/1
A	ORL *C,bit* 2/2*	AJMP *page5* 2/2	MOV *C,bit* 2/1	INC *DPTR* 1/2*	MUL *AB* 1/4		MOV *@R0,dir* 2/2*	MOV *@R1,dir* 2/2*	MOV *R0,dir* 2/2*	MOV *R1,dir* 2/2*	MOV *R2,dir* 2/2*	MOV *R3,dir* 2/2*	MOV *R4,dir* 2/2*	MOV *R5,dir* 2/2*	MOV *R6,dir* 2/2*	MOV *R7,dir* 2/2*
B	ANL *C,bit* 2/2*	ACALL *page5* 2/2	CPL *bit* 2/1	CPL *C* 1/1	CJNE *A,#imm* 3/2	CJNE *A,dir* 3/2	CJNE *@R0,#imm* 3/2	CJNE *@R1,#imm* 3/2	CJNE *R0,#imm* 3/2	CJNE *R1,#imm* 3/2	CJNE *R2,#imm* 3/2	CJNE *R3,#imm* 3/2	CJNE *R4,#imm* 3/2	CJNE *R5,#imm* 3/2	CJNE *R6,#imm* 3/2	CJNE *R7,#imm* 3/2
C	PUSH *dir* 2/2*	AJMP *page6* 2/2	CLR *bit* 2/1	CLR *C* 1/1	SWAP *A* 1/1	XCH *A,dir* 2/1	XCH *A,@R0* 1/1	XCH *A,@R1* 1/1	XCH *A,R0* 1/1	XCH *A,R1* 1/1	XCH *A,R2* 1/1	XCH *A,R3* 1/1	XCH *A,R4* 1/1	XCH *A,R5* 1/1	XCH *A,R6* 1/1	XCH *A,R7* 1/1
D	POP *dir* 2/2*	ACALL *page6* 2/2	SETB *bit* 2/1	SETB *C* 1/1	DA *A* 1/1	DJNZ *dir,rel* 3/2	XCHD *A,@R0* 1/1	XCHD *A,@R1* 1/1	DJNZ *R0,rel* 2/2	DJNZ *R1,rel* 2/2	DJNZ *R2,rel* 2/2	DJNZ *R3,rel* 2/2	DJNZ *R4,rel* 2/2	DJNZ *R5,rel* 2/2	DJNZ *R6,rel* 2/2	DJNZ *R7,rel* 2/2
E	MOVX *A,@DPTR* 1/2	AJMP *page7* 2/2	MOVX *A,@R0* 1/2	MOVX *A,@R1* 1/2	CLR *A* 1/1	MOV *A,dir* 2/1	MOV *A,@R0* 1/1	MOV *A,@R1* 1/1	MOV *A,R0* 1/1	MOV *A,R1* 1/1	MOV *A,R2* 1/1	MOV *A,R3* 1/1	MOV *A,R4* 1/1	MOV *A,R5* 1/1	MOV *A,R6* 1/1	MOV *A,R7* 1/1
F	MOVX *@DPTR,A* 1/2	ACALL *page7* 2/2	MOVX *@R0,A* 1/2	MOVX *@R1,A* 1/2	CPL *A* 1/1	MOV *dir,A* 2/1	MOV *@R0,A* 1/1	MOV *@R1,A* 1/1	MOV *R0,A* 1/1	MOV *R1,A* 1/1	MOV *R2,A* 1/1	MOV *R3,A* 1/1	MOV *R4,A* 1/1	MOV *R5,A* 1/1	MOV *R6,A* 1/1	MOV *R7,A* 1/1

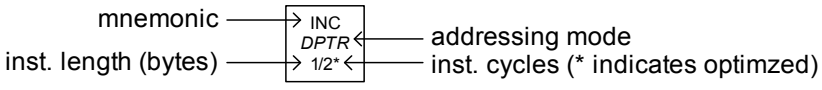


Figure 1 SP27 OpCode Map

2 ROM Library Functions

The following library functions are available for application usage:

Table 3 ROM Library functions

ROM Library function	Description	Page
Meas_Sensor()	Measures the ambient air pressure	Page 16
Meas_Pressure()	Measures the ambient air pressure	Page 22
Meas_Temperature()	Measures the ambient temperature	Page 28
Raw_Temperature()	Measures the raw temperature	Page 31
Comp_Temperature()	Compensates raw temperature data	Page 33
ADC_Selftest()	Returns the delta of ADC test measurements	Page 36
Powerdown()	Forces the device to POWER DOWN state	Page 38
IntervalTimerCalibration()	Calibrates the Interval Timer precounter	Page 40
SMulIntInt()	Multiplies two signed values (16 bit * 16 bit)	Page 42
UDivLongLong()	Divides two unsigned values (32 bit : 32 bit)	Page 44
UDivIntInt()	Divides two unsigned values (16 bit : 16 bit)	Page 46
CRC8_Calc()	Calculates an 8 Bit CRC with polynom 83_H	Page 48
Read_ID()	Returns the unique device ID	Page 50
FW_Revision_Nb()	Returns the ROM- and Flash library revision number	Page 52
Erase_UserConfigSector()	Erases the FLASH user configuration sector	Page 54
WriteFlashUserConfigSectorLine()	Writes one FLASH line (32 Bytes) in the FLASH user configuration sector	Page 56
WriteFlashCodeSectorLine()	Writes one FLASH line (32 Bytes) in the FLASH Code Sector 0.	Page 58
FlashSetLock()	Sets the Lockbyte to protect the User Configuration sector	Page 60
ECC_Check()	Evaluates the ECC result bit	Page 62
CRC16_Check()	Evaluates the CRC16 result of a memory block	Page 64
GetCompValue()	Returns a compensated value from look up table	Page 66
Wait100usMultiples()	Performs a delay of 100 µs multiples	Page 70

2.1 Meas_Sensor()

2.1.1 Description

This function performs a pressure sensor measurement determined by **SensorConfig**.

The pressure result can be either:

- Compensated for sensitivity, offset and temperature
- Output as raw value without performing the compensation

The function can measure pressure with up to 64 samples at a specified sample rate. The function will return the average (arithmetic mean) pressure value in order to compensate for noise. The function performs a new temperature measurement for compensating the pressure measurements, unless a previously measured raw temperature value is supplied. Number of samples and raw temperature source are both determined by **SensorConfig**. SampleRate controls how frequently the measurement acquisitions occur and will not influence the measurement result under stable input pressure conditions. The device is set to IDLE mode during the delay between measurement acquisitions.

2.1.2 Actions

Pressure Measurement

- Measure pressure sensor with channel and number of samples given by **SensorConfig**
- Check wire bonds between the ASIC and the sensor die
- Average the measurement result(s), or (optionally) use the maximum raw measurement, to obtain a raw pressure value
- Sensor gain is automatically configured
- (optionally) Perform a raw temperature measurement
- (optionally) Compensate the raw pressure value

2.1.3 Prototype

unsigned char **Meas_Sensor** (unsigned int **SensorConfig**, unsigned char **SampRate**, signed int idata ***SensorResult**)

2.1.4 Inputs

Table 4 Meas_Sensor: Input Parameters

Register / Address	Type	Name	Description
R6 (MSB) R7 (LSB)	unsigned int	SensorConfig	Defines Sensor Configuration (refer to **Table 5** and **Table 6**)
R5	unsigned char	SampleRate	Defines the number of system clock cycles (12 MHz RC Oscillator) divided by 8 which is waited between consecutive measurements. *Note: Only applicable if SensorConfig.2-0 [2POWN2:0] greater than 1 Sample*
R3	signed int idata*	SensorResult	Pointer to an integer array in RAM to receive the measurement result
*(SensorResult+2)	signed int	RawTemperature	Previous Raw Temperature value can optionally be used as input parameter. Refer to bit RAWTemp in SensorConfig.

Table 5 Meas_Sensor: Input Parameter: SensorConfig[15:8]

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
RES	**RES**	**RES**	**RES**	**RES**	**RES**	**RES**	**RAWTemp**
RES		The RES have to be set to 0_B					
RAWTemp		Selects source of raw temperature data for compensation 0_B: Perform new raw temperature measurement 1_B: Use raw temperature data supplied in *(SensorResult+2)					

Table 6 Meas_Sensor: Input Parameter: SensorConfig[7:0]

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PSensor	**RAW**	**RES**	**RES**	**RES**	**2POWN.2**	**2POWN.1**	**2POWN.0**
PSensor		PSensor has to be set to 1_B					
RAW		Defines if the raw ADC result is compensated 0_B: Temperature compensation is performed. Returns compensated & raw value 1_B: No compensation is performed. Returns only raw value. If SensorConfig bit RAWTemp is set to 1_B, no RAW temperature measurement is performed. *Note: See **Table 8** for impact on ROM Library function output*					
RES		RES bits have to be set to 0_B					
2POWN[2:0]		Number of ADC measurements that are taken and averaged 111_B: 64 Samples 110_B: 64 Samples 101_B: 32 Samples 100_B: 16 Samples 011_B: 8 Samples 010_B: 4 Samples 001_B: 2 Samples 000_B: 1 Sample					

Table 7 Meas_Sensor: Input Parameter: SampleRate

Bit7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SR.7	**SR.6**	**SR.5**	**SR.4**	**SR.3**	**SR.2**	**SR.1**	**SR0**
SR[7:0]		Number of systemclock cycles divided by 8 between two consecutive samples (only applicable if more than one sample is taken): 00_H: No delay (fastest possible sample rate) $01_H..4F_H$: Not allowed $50_H..FF_H$: 1/sample rate (sample rate in systemclocks divided by 8)					

2.1.5 Outputs

Table 8 Meas_Sensor: Output values

Register/ Address	Type	Name	Description
R7	unsigned char	StatusByte	0000.0000_B: Success $xxxx.xxx1_B$: Underflow of ADC Result[1)] $xxxx.xx1x_B$: Overflow of ADC Result[1)] $xxxx.x1xx_B$: Sensor Fault Wire Bond Check
*(SensorResult+0)	signed int	Compensated Pressure	If Input Bit SensorConfig.7[Sens_Type] == 1; If Input Bit SensorConfig.6[RAW] == 0: 8000_H = -2048.0 kPa (Only theoretical) 0000_H = 0.0 kPa $7FFF_H$ = 2047.9375 kPa (= 2048 kPa - 1 LSB where 1 LSB = 1/16 kPa) If Input Bit SensorConfig.6[RAW] == 1: 8000_H since no compensation is performed
*(SensorResult+1)	signed int	Raw Pressure	10 Bit ADC Result Value: $0000.00xx.xxxx.xxxx_B$
*(SensorResult+2)	signed int	Raw Temperature	If Input Bit SensorConfig.6[RAW] == 0: 16 Bit scaled signed ADC Result Value If Input Bit SensorConfig.6[RAW] == 1: 8000_H since no temperature measurement is performed

1) If the sensor measurement result is within the input range for which its accuracy is specified, then the ADC underflow/overflow condition is due to a measurement failure and the measurement results are not valid. Otherwise, if the sensor measurement result is outside of this input range, then the ADC underflow/overflow bits may be ignored.

2.1.6 Resource Usage

Table 9 Meas_Sensor: Resources

Type	Used or Modified
Registers	R0, R1, R2, R3, R4, R5, R6, R7
SFR	ACC, B, CFG0[1)], CFG1, CFG2, DIVIC, DPH, DPL, PSW, TCON[1)], TH0[1)], TL0[1)], TMOD[1)]
Stack	13 Bytes[2)]

1) Only affected if more than 1 sample is taken
2) Two additional bytes, not included here, are needed to call the library function

2.1.7 Execution Information

Table 10 Meas_Sensor: Pressure Measurement: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time Compensated with new raw temperature measurement, 2 samples	$t_{comp,new}$	–	916	1016	µs	DIVIC = 00_H, SensorConfig = 0081_H, SampleRate = 00_H
Execution Time Compensated when supplied with previously obtained raw temperature value, 2 samples	$t_{comp,prev}$	–	686	768	µs	DIVIC = 00_H, SensorConfig = 0181_H, SampleRate = 00_H
Execution Time Raw (uncompensated) pressure result, 2 samples	t_{RAW}	–	304	332	µs	DIVIC = 00_H, SensorConfig = $00C1_H$, SampleRate = 00_H
Execution Time for each additional sample for averaging	t_{sample}	–	54	60	µs	SampleRate = 00_H Only 1, 2, 4, 8, 16, 32, 64 samples possible
Charge Consumption Compensated with new raw temperature measurement, 2 samples	$Q_{comp,new}$	–	1,62	2,535	µC	DIVIC = 00_H, SensorConfig = 0081_H, SampleRate = 00_H
Charge Consumption Compensated when supplied with previously obtained raw temperature value, 2 samples	$Q_{comp,prev}$	–	1,175	1,889	µC	DIVIC = 00_H, SensorConfig = 0181_H, SampleRate = 00_H
Charge Consumption Raw (uncompensated) pressure result, 2 samples	Q_{RAW}	–	0,549	0,821	µC	DIVIC = 00_H, SensorConfig = $00C1_H$, SampleRate = 00_H
Charge Consumption for each additional sample for averaging	Q_{sample}	–	0,118	0,151	µC	SampleRate = 00_H Only 1, 2, 4, 8, 16, 32, 64 samples possible

2.2 Meas_Pressure()

2.2.1 Description

This function performs a pressure sensor measurement.

The result can be either:

- Compensated for sensitivity, offset and temperature
- Output as raw value without performing the compensation

The function can measure pressure with up to 64 samples at a specified sample rate. The function will return the average (arithmetic mean) pressure value in order to compensate for noise. The function performs a new temperature measurement for compensating the pressure measurements, unless a previously measured raw temperature value is supplied. Number of samples and raw temperature source are both determined by **SensorConfig**. SampleRate controls how frequently the measurement acquisitions occur and will not influence the measurement result under stable input pressure conditions. The device is set to IDLE mode during the delay between measurement acquisitions.

2.2.2 Actions

- Measure pressure sensor with gain, channel and number of samples for averaging given by **SensorConfig**
- Check wire bonds between the ASIC and the sensor die
- Average the measurement result(s) to obtain a raw pressure value
- (optionally) Perform a raw temperature measurement
- (optionally) Compensate the raw pressure value

2.2.3 Prototype

unsigned char **Meas_Pressure** (unsigned int **SensorConfig**, unsigned char **SampRate**, signed int idata ***PressResult**)

2.2.4 Inputs

Table 11 Meas_Pressure: Input Parameters

Register / Address	Type	Name	Description
R6 (MSB) R7 (LSB)	unsigned int	SensorConfig	Defines Sensor Configuration (refer to **Table 12** and **Table 13**)
R5	unsigned char	SampleRate	Defines the number of system clock cycles (12 MHz RC Oscillator) divided by 8 which is waited between consecutive measurements. *Note: Only applicable if SensorConfig.2-0 [2POWN2:0] greater than 1 Sample*
R3	signed int idata*	PressResult	Pointer to an integer array in RAM to receive the measurement result
*(PressResult+2)	signed int	Raw Temperature	Previous Raw Temperature value can optionally be used as input parameter. Refer to bit RAWTemp in SensorConfig.

Table 12 Meas_Pressure: Input Parameter: SensorConfig[15:8]

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
RES	**RES**	**RES**	**RES**	**RES**	**RES**	**RES**	**RAWTemp**
RES		RES bits have to be set to 0_B					
RAWTemp		Selects source of raw temperature data for compensation 0_B: Perform new raw temperature measurement 1_B: Use raw temperature data supplied in *(PressResult+2)					

Table 13 Meas_Pressure: Input Parameter: SensorConfig[7:0]

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Sens_Type	**RAW**	**RES**	**RES**	**RES**	**2POWN.2**	**2POWN.1**	**2POWN.0**
Sens_Type		Select Pressure Measurement Must be set to 1_B for Pressure Measurement					
RAW		Defines if the raw ADC result is compensated 0_B: Temperature compensation is performed. Returns compensated & raw value 1_B: No compensation is performed. Returns only raw value. If SensorConfig bit RAWTemp is set to 1_B, no RAW temperature measurement is performed. *Note: See **Table 15** for impact on ROM Library function output*					
RES		RES bits have to be set to 0_B					
2POWN[2:0]		Number of ADC measurements that are taken and averaged 111_B: 64 Samples 110_B: 64 Samples 101_B: 32 Samples 100_B: 16 Samples 011_B: 8 Samples 010_B: 4 Samples 001_B: 2 Samples 000_B: 1 Sample					

Table 14 Meas_Pressure: Input Parameter: SampleRate

Bit7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SR.7	**SR.6**	**SR.5**	**SR.4**	**SR.3**	**SR.2**	**SR.1**	**SR0**
SR[7:0]		Number of systemclock cycles divided by 8 between two consecutive samples (only applicable if more than one sample is taken and averaged): 00_H: No delay (fastest possible sample rate) $01_H..4F_H$: Not allowed $50_H..FF_H$: 1/sample rate (sample rate in systemclocks divided by 8)					

2.2.5 Outputs

Table 15 Meas_Pressure: Output values

Register/ Address	Type	Name	Description
R7	unsigned char	StatusByte	0000.0000_B: Success $xxxx.xxx1_B$: Underflow of ADC Result[1)] $xxxx.xx1x_B$: Overflow of ADC Result[1)] $xxxx.x1xx_B$: Sensor Fault Wire Bond Check
*(PressResult+0)	signed int	Compensated Pressure	If Input Bit SensorConfig.6[RAW] == 0: 8000_H = -2048.0 kPa (Only theoretical) 0000_H = 0.0 kPa $7FFF_H$ = 2047.9375 kPa (= 2048 kPa - 1 LSB where 1 LSB = 1/16 kPa) If Input Bit SensorConfig.6[RAW] == 1: 8000_H since no compensation is performed
*(PressResult+1)	signed int	Raw Pressure	10 Bit ADC Result Value: $0000.00xx.xxxx.xxxx_B$
*(PressResult+2)	signed int	Raw Temperature	If Input Bit SensorConfig.6[RAW] == 0: 16 Bit scaled signed ADC Result Value If Input Bit SensorConfig.6[RAW] == 1: 8000_H since no temperature measurement is performed

1) If the pressure measurement result is within the input range for which its accuracy is specified, then the ADC underflow/overflow condition is due to a measurement failure and the measurement results are not valid. Otherwise, if the pressure measurement result is outside of this input range, then the ADC underflow/overflow bits and the pressure measurement results may be ignored.

2.2.6 Resource Usage

Table 16 Meas_Pressure: Resources

Type	Used or Modified
Registers	R0, R1, R2, R3, R4, R5, R6, R7
SFR	ACC, B, CFG0[1)], CFG1, CFG2, DIVIC, DPH, DPL, PSW, TCON[1)], TH0[1)], TL0[1)], TMOD[1)]
Stack	11 Bytes

1) Only affected if more than 1 sample is taken

2.2.7 Execution Information

Table 17 Meas_Pressure: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time Compensated with new raw temperature measurement, 2 samples	$t_{comp,new}$	–	918	1017	µs	DIVIC = 00_H, SensorConfig = 0081_H, SampleRate = 00_H
Execution Time Compensated when supplied with previously obtained raw temperature value, 2 samples	$t_{comp,prev}$	–	689	770	µs	DIVIC = 00_H, SensorConfig = 0181_H, SampleRate = 00_H
Execution Time Raw (uncompensated) pressure result, 2 samples	t_{RAW}	–	305	334	µs	DIVIC = 00_H, SensorConfig = $00C1_H$, SampleRate = 00_H
Execution Time for each additional sample for averaging	t_{sample}	–	54	60	µs	SampleRate = 00_H Only 1, 2, 4, 8, 16, 32, 64 samples possible
Charge Consumption Compensated with new raw temperature measurement, 2 samples	$Q_{comp,new}$	–	1,622	2,539	µC	DIVIC = 00_H, SensorConfig = 0081_H, SampleRate = 00_H
Charge Consumption Compensated when supplied with previously obtained raw temperature value, 2 samples	$Q_{comp,prev}$	–	1,173	1,9	µC	DIVIC = 00_H, SensorConfig = 0181_H, SampleRate = 00_H
Charge Consumption Raw (uncompensated) pressure result, 2 samples	Q_{RAW}	–	0,551	0,825	µC	DIVIC = 00_H, SensorConfig = $00C1_H$, SampleRate = 00_H
Charge Consumption for each additional sample for averaging	Q_{sample}	–	0,118	0,151	µC	SampleRate = 00_H Only 1, 2, 4, 8, 16, 32, 64 samples possible

2.2.8 Code Example

```
// Library function prototypes
#include "SP27_ROMLibrary.h"

void main()
{
     // Return value of pressure measurement is stored in StatusByte
     unsigned char StatusByte;

     // Input parameters for pressure measurement
     unsigned int SensorConfig = 0x0081;
     unsigned char SampRate = 0x00;

     // struct for pressure measurement results
     struct{
           signed int Pressure;
           signed int Raw_pressure;
           signed int Raw_temperature;
     } idata Press_Result;

     // Pressure measurement function call
     StatusByte = Meas_Pressure(SensorConfig, SampRate, &Press_Result.Pressure);

     if(!StatusByte){
           // Pressure measurement was successful
     }
     else{
           // Pressure measurement was not successful, underflow or
           // overflow of ADC result, Sensor Fault Wire Bond Check
     }
}
```

Figure 2 Code example for usage of Meas_Pressure()

2.3 Meas_Temperature()

2.3.1 Description

This function performs a temperature measurement and returns both raw and compensated temperature results. The Compensated Temperature result is compensated for sensitivity and offset errors. The Raw Temperature result is uncompensated and may be used as input for **Meas_Sensor()**, , and **Meas_Pressure()**.

2.3.2 Actions

- Measures the temperature sensor with 2 ADC samples for averaging (arithmetic mean).
- Compensate for offset using calibration data stored in FLASH (Compensated Temperature)

2.3.3 Prototype

unsigned char **Meas_Temperature** (signed int idata * **Temp_Result**)

2.3.4 Inputs

Table 18 Meas_Temperature: Input Parameters

Register / Address	Type	Name	Description
R7	signed int idata*	Temp_Result	Pointer to an integer array in RAM to receive the measurement result

2.3.5 Outputs

Table 19 Meas_Temperature: Output values

Register/ Address	Type	Name	Description
R7	unsigned char	StatusByte	0000.0000_B: Success $xxxx.xxx1_B$: Underflow of ADC Result $xxxx.xx1x_B$: Overflow of ADC Result
*Temp_Result+0	signed int	Compensated Temperature	8000_H = -256.0 °C 0000_H = 0.0 °C $7FFF_H$ = 255.9921875 °C (= 256 °C - 1 LSB where 1 LSB = 1/128 °C)
*Temp_Result+1	signed int	Raw Temperature	16 Bit scaled signed ADC Result Value

2.3.6 Resource Usage

Table 20 Meas_Temperature: Resources

Type	Used or Modified
Registers	R0, R1, R2, R3, R4, R5, R6, R7
SFR	ACC, B, CFG0, CFG1, CFG2, DIVIC, DPH, DPL, PSW
Stack	9 Bytes[1)]

1) Two additional bytes, not included here, are needed to call the library function

2.3.7 Execution Information

Table 21 Meas_Temperature: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time	t_{comp}	–	668	726	µs	DIVIC = 00_H, by default 2 ADC measurements are taken and averaged
Charge Consumption	Q_{comp}	–	1,2	1,82	µC	DIVIC = 00_H, by default 2 ADC measurements are taken and averaged

2.3.8 Code Example

```
// Library function prototypes
#include "SP27_ROMLibrary.h"

void main()
{
     // Return value of temperature measurement is stored in StatusByte
     unsigned char StatusByte;

     // struct for temperature measurement results
     struct{
           signed int Temperature;
           signed int Raw_temperature;
     } idata Temp_Result;

     // Temperature measurement function call
     StatusByte = Meas_Temperature(&Temp_Result.Temperature);

     if(!StatusByte){
           // Temperature measurement was successful
     }
     else{
           // Temperature measurement was not successful, underflow or
           // overflow of ADC result
     }
}
```

Figure 3 Code example for usage of Meas_Temperature()

2.4 Raw_Temperature()

2.4.1 Description

This function performs a raw temperature measurement. The Raw Temperature result is uncompensated and may be used as input for **Meas_Sensor()**, **Meas_Pressure()**, and **Comp_Temperature()**.

2.4.2 Actions

- Measures the temperature sensor with 2 ADC samples for averaging (arithmetic mean)

2.4.3 Prototype

unsigned char **Raw_Temperature** (signed int idata * **TempResult**)

2.4.4 Inputs

Table 22 Raw_Temperature: Input Parameters

Register / Address	Type	Name	Description
R7	signed int idata*	TempResult	Pointer to an integer array in RAM to receive the measurement result

2.4.5 Outputs

Table 23 Raw_Temperature: Output values

Register/ Address	Type	Name	Description
R7	unsigned char	StatusByte	0000.0000_B: Success $xxxx.xxx1_B$: Underflow of ADC Result $xxxx.xx1x_B$: Overflow of ADC Result
*Temp_Result+0	signed int	Raw Temperature	16 Bit scaled signed ADC Result Value

2.4.6 Resource Usage

Table 24 Raw_Temperature: Resources

Type	Used or Modified
Registers	R0, R3, R4, R5, R6, R7
SFR	ACC, B, CFG0, CFG1, CFG2, DPH, DPL, PSW
Stack	5 Bytes[1)]

1) Two additional bytes, not included here, are needed to call the library function

2.4.7 Execution Information

Table 25 Raw_Temperature: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time	t_{RAW}	–	274	299	µs	DIVIC = 00_H, by default 2 ADC measurements are taken and averaged
Charge Consumption	Q_{RAW}	–	0,513	0,773	µC	DIVIC = 00_H, by default 2 ADC measurements are taken and averaged

2.5 Comp_Temperature()

2.5.1 Description

This function will convert a previously obtained raw temperature value into a compensated temperature value. This function can be called after a pressure or raw temperature measurement to compensate the raw temperature. Thus, the current ambient temperature is obtained without taking an extra temperature measurement.

2.5.2 Actions

- Compensate raw temperature data using calibration data stored in FLASH.

2.5.3 Prototype

unsigned char **Comp_Temperature** (signed int idata **TempRawIn,** signed int idata * **TempResult**)

2.5.4 Inputs

Table 26 Comp_Temperature: Input Parameters

Register / Address	Type	Name	Description
R6(MSB), R7(LSB)	signed int idata	TempRawIn	Raw measurement value (gathered from Meas_Pressure() or Raw_Temperature)
R5	signed int idata*	Temp_Result	Pointer to an integer array in RAM to receive the measurement result

2.5.5 Outputs

Table 27 Comp_Temperature: Output values

Register/ Address	Type	Name	Description
R7	unsigned char	StatusByte	0000.0000_B: Success $xxxx.xxx1_B$: Underflow of Result $xxxx.xx1x_B$: Overflow of Result
*Temp_Result	signed int	Compensated Temperature	8000_H = -256.0 °C 0000_H = 0.0 °C $7FFF_H$ = 255.9921875 °C (= 256 °C - 1 LSB where 1 LSB = 1/128 °C)
*Temp_Result+1	signed int	Raw Temperature	16 Bit scaled signed ADC Result Value

2.5.6 Resource Usage

Table 28 Comp_Temperature: Resources

Type	Used or Modified
Registers	R0, R1, R2, R3, R4, R5, R6, R7
SFR	ACC, B, DPH, DPL, PSW
Stack	4 Bytes[1)]

1) Two additional bytes, not included here, are needed to call the library function

2.5.7 Execution Information

Table 29 Comp_Temperature: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time	t	–	420	454	µs	DIVIC = 00_H
Charge Consumption	Q	–	0,63	1,089	µC	DIVIC = 00_H

2.5.8 Code Example

```
// Library function prototypes
#include "SP27_ROMLibrary.h"

void main()
{
     // Return value of pressure and temperature measurement is stored in StatusByte
     unsigned char StatusByte;

     // Input parameters for pressure measurement
     unsigned int SensorConfig = 0x0081;
     unsigned char SampRate = 0x00;

     // struct for pressure measurement results
     struct{
           signed int Pressure;
           signed int Raw_pressure;
           signed int Raw_temperature;
     } idata Press_Result;

     // struct for compensated temperature results
     struct{
           signed int Temperature;
           signed int Raw_temperature;
     } idata Temp_Result;

     // Pressure measurement function call
     StatusByte = Meas_Pressure(SensorConfig, SampRate, &Press_Result.Pressure);

     if(!StatusByte){
           // Pressure measurement was successful
     }
     else{
           // Pressure measurement was not successful, underflow or
           // overflow of ADC result, Sensor Fault Wire Bond Check
     }

     // Compensate Temperature function call
     StatusByte = Comp_Temperature(Press_Result.Raw_temperature, &Temp_Result.Temperature);

     if(!StatusByte){
           // Temperature compensation was successful
     }
     else{
           // Temperature compensation was not successful, underflow or
           // overflow during compensation occurred
     }
}
```

Figure 4 Code example for usage of Comp_Temperature()

2.6 ADC_Selftest()

2.6.1 Description

The ADC self test is a combination of three measurements that use various channels as input and reference for the ADC. The output of this function is the delta deviation from the ideal value.

2.6.2 Actions

- Perform three ADC measurements
- Calculate delta from ideal value

2.6.3 Prototype

unsigned char **ADC_Selftest**(signed int idata * **Delta**)

2.6.4 Inputs

Table 30 ADC_Selftest: Input Parameters

Register / Address	Type	Name	Description
R7	signed int idata*	Delta	Pointer to an integer array in RAM to receive the ADC measurement result

2.6.5 Outputs

Table 31 ADC_Selftest: Output values

Register/ Address	Type	Name	Description
R7	unsigned char	StatusByte	0000.0000_B: Success $xxxx.xxx1_B$: Underflow of ADC Result $xxxx.xx1x_B$: Overflow of ADC Result
*Delta	signed int idata*	Delta	The ADC can be considered as working when the value of Delta is between +/-6 LSBs after a call of this function during which the supply voltage was constant.

2.6.6 Resource Usage

Table 32 ADC_Selftest: Resources

Type	Used or Modified
Registers	R0, R1, R2, R3, R4, R5, R6, R7
SFR	ACC, B, CFG0, CFG1, CFG2, DIVIC, DPH, DPL,PSW
Stack	7 Bytes[1)]

1) Two additional bytes, not included here, are needed to call the library function

2.6.7 Execution Information

Table 33 ADC_Selftest: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time	t	–	645	701	µs	DIVIC = 00_H, 3 Numbers of ADC measurements are taken
Charge Consumption	Q	–	1,2	1,84	µC	DIVIC = 00_H 3 ADC measurements are taken

2.7 Powerdown()

2.7.1 Description

This function forces the device to POWER DOWN state.

2.7.2 Actions

- If an RF Transmission is in process, wait until it has completed
- If the SFR ITPR has been updated, wait for the Interval Timer to initialize
- Enter POWER DOWN state

2.7.3 Prototype

void **Powerdown**(void)

2.7.4 Inputs

Table 34 Powerdown: Input Parameters

Register / Address	Type	Name	Description
None	---	---	---

2.7.5 Outputs

Table 35 Powerdown: Output values

Register/ Address	Type	Name	Description
None	---	---	---

2.7.6 Resource Usage

Table 36 Powerdown: Resources

Type	Used or Modified
Registers	---
SFR	ACC, CFG0, CFG1, RFS
Stack	0 Bytes[1)]

1) Two additional bytes, not included here, are needed to call the library function

2.7.7 Execution Information

Table 37 Powerdown: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time	t	–	11,36	21,6	µs	DIVIC = 00_H, ITinit = 0, RFSE = 1 before entering PDWN
Charge Consumption	Q	–	0,018	0,053	µC	DIVIC = 00_H, ITinit = 0, RFSE = 1 before entering PDWN

2.8 IntervalTimerCalibration()

2.8.1 Description

This function initiates a calibration of the Interval Timer precounter (ITPL and ITPH) to obtain a specific interval timer timebase between 1Hz and 20Hz. The function can work with 12MHz RC Clock source, utilizing a special timer mode.

2.8.2 Actions

- Calibrate the interval timer precounter (SFR ITPL, SFR ITPH) to the value passed in WU_Frequency

2.8.3 Prototype

signed char **IntervalTimerCalibration**(unsigned char **WU_Frequency**)

2.8.4 Inputs

Table 38 IntervalTimerCalibration: Input Parameters

Register / Address	Type	Name	Description
R7	unsigned char	WU_Frequency	Base frequency of the interval timer precounter [Hz] 1dec: 1 Hz (precounter time ~1000 ms) 2dec: 2 Hz (precounter time ~500 ms) ... 19dec: 19 Hz (precounter time ~53 ms) 20dec: 20 Hz (precounter time ~50 ms)

2.8.5 Outputs

Table 39 IntervalTimerCalibration: Output values

Register/ Address	Type	Name	Description
R7	signed char	StatusByte	StatusByte: 0: Success -1: Error -2: Input parameter out of rang2

2.8.6 Resource Usage

Table 40 IntervalTimerCalibration: Resources

Type	Used or Modified
Registers	R0, R1, R2, R3, R4, R5, R6, R7
SFR	ACC, B, CFG0, DPTR, ITPH, ITPL, LFOOTP, PSW, TCON, TH0, TH1, TL0, TL1, TMOD
Stack	2 Bytes[1)]

1) Two additional bytes, not included here, are needed to call the library function

2.8.7 Execution Information

Table 41 IntervalTimerCalibration: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time[1)]	t_{12Mhz}	–	1190	1680	µs	DIVIC = 00_H, Clock Source is 12 MHz RC HF Oscillator
Charge Consumption[1)]	Q_{12MHz}	–	1,23	2,56	µC	DIVIC = 00_H, Clock Source is 12 MHz RC HF Oscillator

1) Values are obtained by extrapolation, not by characterization

2.9 SMulIntInt() (16Bit * 16Bit)

2.9.1 Description

This function multiplies the first signed int value (16 bit) Multiplicand1 by the second signed int value (16bit) Multiplicand2 and produces a 32-bit signed result.

2.9.2 Actions

- Perform multiplication

2.9.3 Prototype

void **SMulIntInt**(signed int idata * **Multiplicand1**, signed int idata * **Multiplicand2**, signed long idata * **Product**)

2.9.4 Inputs

Table 42 SMulIntInt: Input Parameters

Register / Address	Type	Name	Description
R7	signed int idata*	Multiplicand1	Pointer to Multiplicand1
R5	signed int idata*	Multiplicand2	Pointer to Multiplicand2
R3	signed long idata*	Product	Pointer to an long array in RAM to the 32 Bit multiplication Product (Multiplicand1 * Multiplicand2)

2.9.5 Outputs

Table 43 SMulIntInt: Output values

Register/ Address	Type	Name	Description
None	---	---	---

2.9.6 Resource Usage

Table 44 SMulIntInt: Resources

Type	Used or Modified
Registers	R0, R1, R2, R3, R4, R5, R6, R7
SFR	ACC, B, PSW
Stack	2 Bytes[1)]

1) Two additional bytes, not included here, are needed to call the library function

2.9.7 Execution Information

Table 45 SMulIntInt: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time	t	–	75	81	µs	DIVIC = 00_H
Charge Consumption	Q	–	0,112	0,194	µC	DIVIC = 00_H

2.10 UDivLongLong() (32Bit : 32Bit)

2.10.1 Description

This function divides the unsigned long value (32 bit) Dividend by the unsigned long value (32 bit) Divisor.

2.10.2 Actions

- Perform division

2.10.3 Prototype

unsigned long **UDivLongLong**(unsigned long idata * **Dividend**, unsigned long idata * **Divisor**)

2.10.4 Inputs

Table 46 UDivLongLong: Input Parameters

Register / Address	Type	Name	Description
R7	unsigned long idata*	Dividend	Pointer to 32 bit Dividend
R5	unsigned long idata*	Divisor	Pointer to 32 bit Divisor

2.10.5 Outputs

Table 47 UDivLongLong: Output values

Register/ Address	Type	Name	Description
R4(MSB), R5, R6, R7(LSB)	unsigned long	Quotient	32 bit Quotient of the division (Dividend / Divisor)

Note: The output value for the remainder can be found in R0 (MSB), R1, R2, R3 (LSB).

2.10.6 Resource Usage

Table 48 UDivLongLong: Resources

Type	Used or Modified
Registers	R0, R1, R2, R3, R4, R5, R6, R7
SFR	ACC, B, DPH, DPL, PSW
Stack	0 Bytes[1)]

1) Two additional bytes, not included here, are needed to call the library function

2.10.7 Execution Information

Table 49 UDivLongLong: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time	t	–	387	418	µs	DIVIC = 00_H
Charge Consumption	Q	–	0,581	1,004	µC	DIVIC = 00_H

2.11 UDivIntInt() (16Bit : 16Bit)

2.11.1 Description

This function divides the unsigned int value (16 bit) Dividend by the unsigned int value (16 bit) Divisor.

2.11.2 Actions

- Perform division

2.11.3 Prototype

unsigned int **UDivIntInt**(unsigned int **Dividend**, unsigned int **Divisor**)

2.11.4 Inputs

Table 50 UDivIntInt: Input Parameters

Register / Address	Type	Name	Description
R6(MSB) R7(LSB)	unsigned int	Dividend	16 bit Dividend
R4(MSB) R5(LSB)	unsigned int	Divisor	16 bit Divisor

2.11.5 Outputs

Table 51 UDivIntInt: Output values

Register/ Address	Type	Name	Description
R6(MSB), R7(LSB)	unsigned int	Quotient	16 bit Quotient of the division (Dividend / Divisor)

Note: The output value for the remainder can be found in R4 (MSB) and R5 (LSB).

2.11.6 Resource Usage

Table 52 UDivIntInt: Resources

Type	Used or Modified
Registers	R0, R4, R5, R6, R7
SFR	ACC, B, PSW
Stack	0 Bytes[1)]

1) Two additional bytes, not included here, are needed to call the library function

2.11.7 Execution Information

Table 53 UDivIntInt: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time	t	–	104	113	µs	DIVIC = 00_H
Charge Consumption	Q	–	0,156	0,270	µC	DIVIC = 00_H

2.12 CRC8_Calc()

2.12.1 Description

This function calculates the CRC-8 checksum for a memory area in RAM using a fixed polynomial (x^8+x^2+x+1). The CRC-8 calculation starts with a defined preload value.

2.12.2 Actions

- Calculate CRC-8

2.12.3 Prototype

unsigned char **CRC8_Calc**(unsigned char **Preload**, unsigned char idata * **BlockStart**, unsigned char **BlockLength**)

2.12.4 Inputs

Table 54 CRC8_Calc: Input Parameters

Register / Address	Type	Name	Description
R7	unsigned char	Preload	Preload Value for the CRC Calculation. According to CCITT a value FF_H is recommended.
R5	unsigned char idata*	BlockStart	Pointer to first Byte of the Data that is to be used for calculating checksum
R3	unsigned char	BlockLength	Length in Bytes of Block that is used for calculation of the checksum, starting with *BlockStart.

2.12.5 Outputs

Table 55 CRC8_Calc: Output values

Register/ Address	Type	Name	Description
R7	unsigned char	CRC_Result	Calculated CRC8 checksum

2.12.6 Resource Usage

Table 56 CRC8_Calc: Resources

Type	Used or Modified
Registers	R0, R1, R2, R3, R4, R5, R7
SFR	ACC, PSW
Stack	0 Bytes[1)]

1) Two additional bytes, not included here, are needed to call the library function

2.12.7 Execution Information

Table 57 CRC8_Calc: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time	t	–	8 + x*9	9 + x*10	µs	DIVIC = 00_H, x is the number of bytes
Charge Consumption	Q	–	0,012 + x*0,013	0,021 + x*0,023	µC	DIVIC = 00_H, x is the number of bytes

2.13 Read_ID()

2.13.1 Description

This function returns the unique serial number of the device and a product description code.

2.13.2 Actions

- Read 4-byte ID (from Flash Address: 5850_H (MSB)...5853_H (LSB))
- Read 1 byte product code (from Flash Address: $584F_H$)

2.13.3 Prototype

void **Read_ID** (struct ID_Struct idata * idata **ID_Result**)

Note: Structure ID_Struct is defined in SP27_ROMLibrary.h

2.13.4 Inputs

Table 58 Read_ID: Input Parameters

Register / Address	Type	Name	Description
R7	idata*	ID_Result	Pointer to a structure according to the following definition: idata struct ID_Struct {unsigned long ID; unsigned char ProdCode;}

2.13.5 Outputs

Table 59 Read_ID: Input Parameters

Register / Address	Type	Name	Description
R7	idata*	ID_Result	Pointer to a structure according to the following definition: idata struct ID_Struct {unsigned long ID; unsigned char ProdCode;} Product Code pressure range indication: $xxxx.x000_B$: reserved $xxxx.x100_B$: reserved $xxxx.x001_B$: reserved $xxxx.x101_B$: reserved $xxxx.xx10_B$: 1300kPa, Green Package[1)] $xxxx.xx11_B$: reserved

1) For definition “Green Package” please refer to **[1]**

2.13.6 Resource Usage

Table 60 Read_ID: Resources

Type	Used or Modified
Registers	R0, R3, R4, R5, R6, R7
SFR	ACC, DPH, DPL, PSW
Stack	2 Bytes[1)]

1) Two additional bytes, not included here, are needed to call the library function

2.13.7 Execution Information

Table 61 Read_ID: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time	t	–	32	35	µs	DIVIC = 00_H
Charge Consumption	Q	–	0,048	0,082	µC	DIVIC = 00_H

2.13.8 Code Example

```
// Library function prototypes
#include "SP27_ROMLibrary.h"

// ID structure is defined in SP27_ROMLibrary.h
idata ID_Struct ID_Result;

void main()
{
     // Read Sensor ID function call
     Read_ID(&ID_Result);
}
```

Figure 5 Code example for usage of Read_ID()

2.14 FW_Revision_Nb()

2.14.1 Description

This function returns the ROM Library revision number and the FLASH Library revision number.

2.14.2 Actions

- Read 2-byte Revision Number from ROM and return it
- Read 2-byte Revision Number from FLASH Library and return it

2.14.3 Prototype

void **FW_Revision_Nb** (unsigned int idata * **ROM_Rev**, unsigned int idata * **Lib_Rev**)

2.14.4 Inputs

Table 62 FW_Revision_Nb: Input Parameters

Register/ Address	Type	Name	Description
R4(MSB), R5(LSB)	unsigned int idata*	Lib_Rev	Pointer to the RAM location where the FLASH Library code revision will be returned
R6(MSB), R7(LSB)	unsigned int idata*	ROM_Rev	Pointer to the RAM location where the ROM code revision will be returned

2.14.5 Outputs

Table 63 FW_Revision_Nb: Input Parameters

Register/ Address	Type	Name	Description
R4(MSB), R5(LSB)	unsigned int idata*	Lib_Rev	Pointer to the RAM location where the FLASH Library code revision will be returned $xxxx_H$ = SP27 FLASH Library version
R6(MSB), R7(LSB)	unsigned int idata*	ROM_Rev	Pointer to the RAM location where the ROM code revision will be returned $0A32_H$ = SP27 ROM Library Version A5

2.14.6 Resource Usage

Table 64 FW_Revision_Nb: Resources

Type	Used or Modified
Registers	R0, R5, R7
SFR	ACC
Stack	2 Bytes[1)]

1) Two additional bytes, not included here, are needed to call the library function

2.14.7 Execution Information

Table 65 FW_Revision_Nb: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time	t	–	12	13	µs	
Charge Consumption	Q	–	0,018	0,030	µC	

2.15 Erase_UserConfigSector()

2.15.1 Description

This function erases the FLASH user configuration sector located at FLASH address 5780_H -- $57FF_H$ if the Lockbyte 3 is not set. If Lockbyte 3 is set this function will return -1 without any action.
This function returns -1 and has no effect if executed in DEBUG mode.

Note: The application software has to ensure that FLASH is only programmed or erased when all required environmental conditions are fulfilled. Special care has to be taken that ambient temperature T_{FL}, supply voltage V_{batFL} and Endurance En_{FL} are within specified range (see [1]).

2.15.2 Actions

- Erase the FLASH user configuration sector

2.15.3 Prototype

signed char **Erase_UserConfigSector** (void)

2.15.4 Inputs

Table 66 Erase_UserConfigSector: Input Parameters

Register / Address	Type	Name	Description
None	---	---	---

2.15.5 Outputs

Table 67 Erase_UserConfigSector: Output values

Register/ Address	Name	Type	Description
R7	signed char	Statusbyte	0: success -1: failed

2.15.6 Resource Usage

Table 68 Erase_UserConfigSector: Resources

Type	Used or Modified
Registers	R0, R1, R2, R3, R4, R5, R6, R7
SFR	ACC, B, CRC0, CRC1, CRCC, CRCD, DIVIC, DPH, DPL, PSW, TCON, TH0, TH1, TL0, TL1
Stack	5 Bytes[1)]

1) Two additional bytes, not included here, are needed to call the library function

2.15.7 Execution Information

Table 69 Erase_UserConfigSector: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time	t	–	103000	111300	µs	DIVIC = 00_H
Charge Consumption	Q	–	222	358	µC	DIVIC = 00_H

2.16 WriteFlashUserConfigSectorLine()

2.16.1 Description

This function writes one line in the FLASH user configuration sector located at FLASH address 5780_H -- $57FF_H$ if the Lockbyte 3 is not set. If Lockbyte 3 is set this function will return -1 without any action.
This function returns -1 and has no effect if executed in DEBUG mode.

The written data is verified after the programming. In case the verification fails this function will return -1.

Note: The application software has to ensure that FLASH is only programmed or erased when all required environmental conditions are fulfilled. Special care has to be taken that ambient temperature T_{FL}, supply voltage V_{ccFL} and Endurance En_{FL} are within specified range (see [1]).

Note: Before written new data, the Flash User Config Sector needs to be erased by the user.

2.16.2 Actions

- Write one 32 Byte FLASH Line of the FLASH user configuration sector

2.16.3 Prototype

signed char **WriteFlashUserConfigurationSectorLine** (unsigned int **Startaddress**, unsigned char idata * **WrData**)

2.16.4 Inputs

Table 70 WriteFlashUserConfigurationSectorLine: Input Parameters

Register / Address	Type	Name	Description
R7 R6	unsigned int	Startaddress	Startaddress 5780_H: FLASH Line 0 $57A0_H$: FLASH Line 1 $57C0_H$: FLASH Line 2 $57E0_H$: FLASH Line 3
R5	unsigned char idata*	WrData	Pointer to first Byte of the 32 Byte Data array that is going to be written to the FLASH Line.

2.16.5 Outputs

Table 71 WriteFlashUserConfigurationSectorLine: Output values

Register/ Address	Type	Name	Description
R7	signed char	Statusbyte	0: success -1: failed

2.16.6 Resource Usage

Table 72 WriteFlashUserConfigurationSectorLine: Resources

Type	Used or Modified
Registers	R0, R1, R2, R3, R4, R5, R6, R7
SFR	ACC, CFG1, CRC0, CRC1, CRCC, CRCD, DIVIC, DPH, DPL, PSW, TCON, TH0, TH1, TL0, TL1, TMOD
Stack	5 Bytes[1)]

1) Two additional bytes, not included here, are needed to call the library function

2.16.7 Execution Information

Table 73 WriteFlashUserConfigurationSectorLine: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time[1)]	t	–	3917	4260	µs	DIVIC = 00_H, FLASH written with all bit 1
Charge Consumption[1)]	Q	–	8,97	12	µC	DIVIC = 00_H, FLASH written with all bit 1

1) Values are obtained by extrapolation, not by characterization

2.17 WriteFlashCodeSectorLine()

2.17.1 Description

This function offers the possibility to write one Code Sector (sector 0) flash line. It enables the application program to modify the code sector, thus enabling code patches or configuration blocks to be applied to the Flash during application execution.

Some requirements to use this function are mandatory:

- The code sector must be unlocked
- The flash-line to be written must be already in erased-state
- No provision for erasing the Code Sector via ROM library call will be provided
- The function can not be called in debug mode

This function writes one line (32 bytes) to the Flash Code Sector located at Flash address 4000_H -- $575F_H$. If the SP37 is in RUN mode and Lockbyte 2 is not set the write is performed, otherwise this function will return -1. The line contents are verified after the entire line has been written. In the event that the write verification fails, this function will return -2.

Note: The application software has to ensure that Flash is only programmed or erased when all required environmental conditions are fulfilled. Special care has to be taken that ambient temperature T_{FL}, supply voltage and Endurance En_{FL} are within specified range (see [1]).

Note: Each Flash Code Sector Line may be written only once. Once a line is written with data, it must not be written again. there is no corresponding ROM function to erase the Flash Code Sector under application program control.

2.17.2 Actions

Write one Code Sector flash line.

2.17.3 Prototype

signed char **WriteFlashCodeSectorLine** (unsigned int **StartAddress**, unsigned char idata ***WrData**)

2.17.4 Inputs

Table 74 WriteFlashCodeSectorLine: Input Parameters

Register / Address	Type	Name	Description
R6, R7	unsigned int	Startaddress	Must be between 4000_H and $575F_H$ (the line containing Lockbyte2 is excluding) Must be multiple of 32
R5	unsigned char idata*	WrData	Pointer to 32 Byte buffer in RAM holding the contents to be written to the specified Flash line.

2.17.5 Outputs

Table 75 WriteFlashCodeSectorLine: Output values

Register/ Address	Type	Name	Description
R7	signed char	Statusbyte	0: success -1: Lockbyte 2 is set (Code Sector Lock), without writing anything; or the function was called while the device is in debug Mode. -2: After data verification, errors are detected -3: The StartAddress" is not multiple of 32 or the "StartAddress" is out-of-range (nothing was performed); or "

2.17.6 Resource Usage

Table 76 WriteFlashCodeSectorLine: Resources

Type	Used or Modified
Registers	R0, R1, R2, R3, R4, R5, R6, R7
SFR	ACC, PSW, DPTR, CFG1, TCON, TMOD, TH0, TL0, TH1, TL1, DSR, CRC0, CRC1, CRCD, CRCC, DIVIC
Stack	5 Bytes[1)]

1) Two additional bytes, not included here, are needed to call the library function

2.17.7 Execution Information

Table 77 WriteFlashCodeSectorLine: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time	t	–	3850	4317	µs	DIVIC = 00_H, FLASH written with all bit 1
Charge Consumption	Q	–	7,857	11,379	µC	DIVIC = 00_H, FLASH written with all bit 1

2.18 FlashSetLock()

2.18.1 Description

This function sets the Lockbyte 3 which protects the User Configuration sector. This function returns -1 and has no effect if executed in DEBUG mode.

Attention: This function shows only effect if the Lockbyte 2 that protects the Code Sector is set.

Note: The application software has to ensure that FLASH is only programmed or erased when all required environmental conditions are fulfilled. Special care has to be taken that ambient temperature T_{FL}, supply voltage V_{ccFL} and Endurance En_{FL} are within specified range (see [1]).
This function returns -1 and has no effect if executed in DEBUG mode.

2.18.2 Actions

- Set the Lockbyte 3 protecting the User Configuration sector

2.18.3 Prototype

signed char **FlashSetLock**(void)

2.18.4 Inputs

Table 78 FlashSetLock: Input Parameters

Register / Address	Type	Name	Description
None	---	---	---

2.18.5 Outputs

Table 79 FlashSetLock: Output values

Register/ Address	Type	Name	Description
R7	signed char	Statusbyte	0: success -1: failed

2.18.6 Resource Usage

Table 80 FlashSetLock: Resources

Type	Used or Modified
Registers	R0, R1, R2, R3, R4, R5, R6, R7
SFR	ACC, B, CFG1, CRC0, CRC1, CRCC, CRCD, DIVIC, DPH, DPL, TCON, TH1, TL1
Stack	8 Bytes[1)]
RAM[2)]	0x10 to 0x8F

1) Two additional bytes, not included here, are needed to call the library function
2) The FlashSetLock() overwrites RAM addresses 0x10 to 0x8F during its execution. For this reason it is recommended to perform a software reset (CFG2.0[RESET] == 1) after this function returns.

2.18.7 Execution Information

Table 81 FlashSetLock: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time[1)]	t	–	116030	126010	µs	DIVIC = 00_H
Charge Consumption[1)]	Q	–	291	404	µC	DIVIC = 00_H

1) Values are obtained by extrapolation, not by characterization

2.19 ECC_Check()

2.19.1 Description

This function evaluates the **E**rror **C**ode **C**orrection (ECC) result bit. This bit shows if an error in the read/executed FLASH has been detected since the last call of this function.

Note: When a FLASH byte is programmed, the ECC unit generates and stores four ECC bits. With these additional bits the ECC is able to correct single bit errors and detects two bit errors. The ECC is calculated from read/executed FLASH and the result is stored in the ECC result bit. It cannot distinguish between a single bit or two bit error.

2.19.2 Actions

- Check the ECC result bit

2.19.3 Prototype

signed char **ECC_Check**(void)

2.19.4 Inputs

Table 82 ECC_Check: Input Parameters

Register / Address	Type	Name	Description
None	---	---	---

2.19.5 Outputs

Table 83 ECC_Check: Output values

Register/ Address	Type	Name	Description
R7	signed char	Statusbyte	0: success -1: ECC error detected

2.19.6 Resource Usage

Table 84 ECC_Check: Resources

Type	Used or Modified
Registers	R7
SFR	ACC, PSW
Stack	0 Bytes[1)]

1) Two additional bytes, not included here, are needed to call the library function

2.19.7 Execution Information

Table 85 ECC_Check: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time	t	–	6	7	µs	DIVIC = 00_H
Charge Consumption	Q	–	0,009	0,016	µC	DIVIC = 00_H

2.19.8 Code Example

```
// Library function prototypes
#include "SP27_ROMLibrary.h"

void main()
{
     // Return value of ECC check is stored in StatusByte
     unsigned char StatusByte;

     // ECC check function call to reset the ECC check function
     StatusByte = ECC_Check();

     // CRC16 check function call to test User Data Sector
     StatusByte = CRC16_Check(0x5780,0x80);

     // ECC check function call
     StatusByte += ECC_Check();

     if(!StatusByte){
          // Both ECC and CRC16 check were successful
     }
     else{
          // ECC and or CRC16 check was not successful
     }
}
```

Figure 6 Code example for usage of the functions ECC_Check() and CRC16_Check()

2.20 CRC16_Check()

2.20.1 Description

This function computes a 16-bit CRC of a code block from a start address with a defined length from FLASH, which includes a pre-computed 16-bit CRC, which is stored in the last 2 Bytes of the memory block ((StartAddr + Length - 2) is CRC High) and ((StartAddr + Length - 1) is CRC Low).

Note: The pre-computed CRC value has to be written by the application for code blocks in FLASH.

2.20.2 Actions

- Calculate 16-bit CRC
- Compare CRC with pre-computed CRC located in the last 2 Bytes of the code block

2.20.3 Prototype

signed char **CRC16_Check**(unsigned char code * **StartAddr**, unsigned int **Length**)

2.20.4 Inputs

Table 86 CRC16_Check: Input Parameters

Register / Address	Type	Name	Description
R6 (MSB) R7 (LSB)	unsigned char code*	StartAddr	Pointer to first Byte of the Data that is used for calculating the checksum
R4 (MSB) R5 (LSB)	unsigned int	Length	Length in Bytes of the code block, including the pre-computed CRC value, that is used for calculating the checksum.

2.20.5 Outputs

Table 87 CRC16_Check: Output values

Register/ Address	Type	Name	Description
R7	signed char	Statusbyte	0: CRC matches -1: CRC does not match

2.20.6 Resource Usage

Table 88 CRC16_Check: Resources

Type	Used or Modified
Registers	R4, R5, R6, R7
SFR	ACC, B, CRC0, CRC1, CRCD, DPH, DPL, PSW
Stack	0 Bytes[1)]

1) Two additional bytes, not included here, are needed to call the library function

2.20.7 Execution Information

Table 89 CRC16_Check: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time	t	–	22 + x*8	24 + x*9	µs	DIVIC = 00_H, x is the number of bytes
Charge Consumption	Q	–	0,033 + x*0,012	0,056 + x*0,020	µC	DIVIC = 00_H, x is the number of bytes

2.20.8 Code Example

2.21 GetCompValue()

2.21.1 Description

This function retrieves an 8 bit value from a 2 dimensional lookup table depending on input values Value1 and Value2. The lookup table is a M by N matrix and can be of any size from 2 x 2 up to 15 x 15 holding 225 different values in its maximum configuration. This function may be used to obtain a compensation value from a lookup table with 2 threshold types (e.g. temperature and voltage), according to the measured input values (Value1 and Value2). The lookup table has to be stored in the FLASH.

The input values are compared against threshold points defined in the lookup table, and the thresholds must be sorted in increasing order. The column is chosen by Value1 thresholds and the row is selected by Value2 thresholds respectively. If Value1 is lower than its lowest threshold then the left-most column is selected, and likewise if Value2 is lower than its lowest threshold the top-most row is selected.

The matrix size is always 1 larger than the number of threshold points. Thus, a 12 by 7 matrix will have 6 threshold points for Value1 (column) and 11 threshold points for Value2 (row).

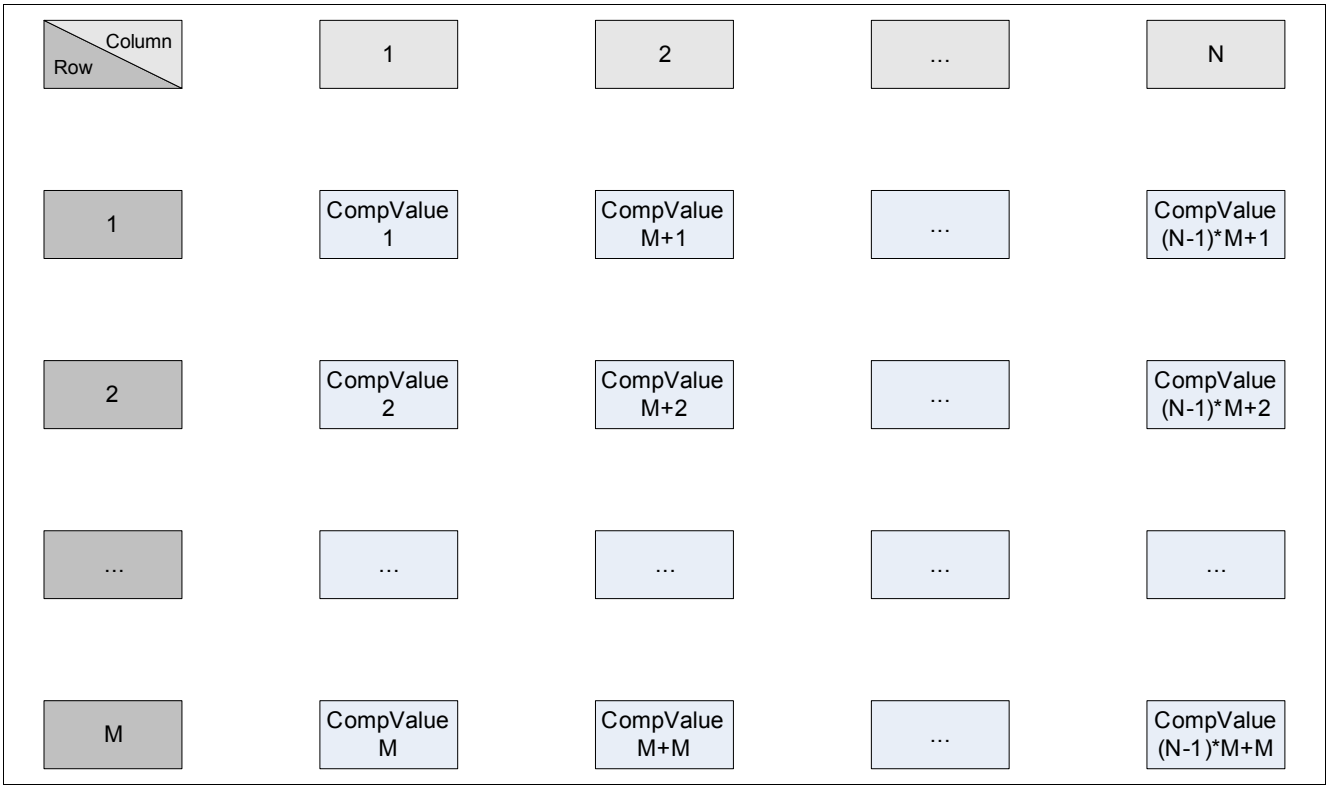


Figure 7 M by N matrix

2.21.2 Actions

- Compare Value1 with defined thresholds from threshold type 1
- Compare Value2 with defined thresholds from threshold type 2
- Return compensated value

2.21.3 Prototype

unsigned char **GetCompValue**(unsigned char code * **TablePointer**, unsigned char **Value1**, unsigned char **Value2**)

2.21.4 Inputs

Table 90 GetCompValue: Input Parameters

Register / Address	Type	Name	Description
R6 (MSB) R7 (LSB)	unsigned char code*	TablePointer	Pointer to the first Byte of lookup table
R5	unsigned char	Value1	Measured Value1 for comparison with thresholds from threshold type 1 to select column
R3	unsigned char	Value2	Measured Value2 for comparison with thresholds from threshold type 2 to select row

Figure 8 shows how the compensated value table must appear in FLASH memory.

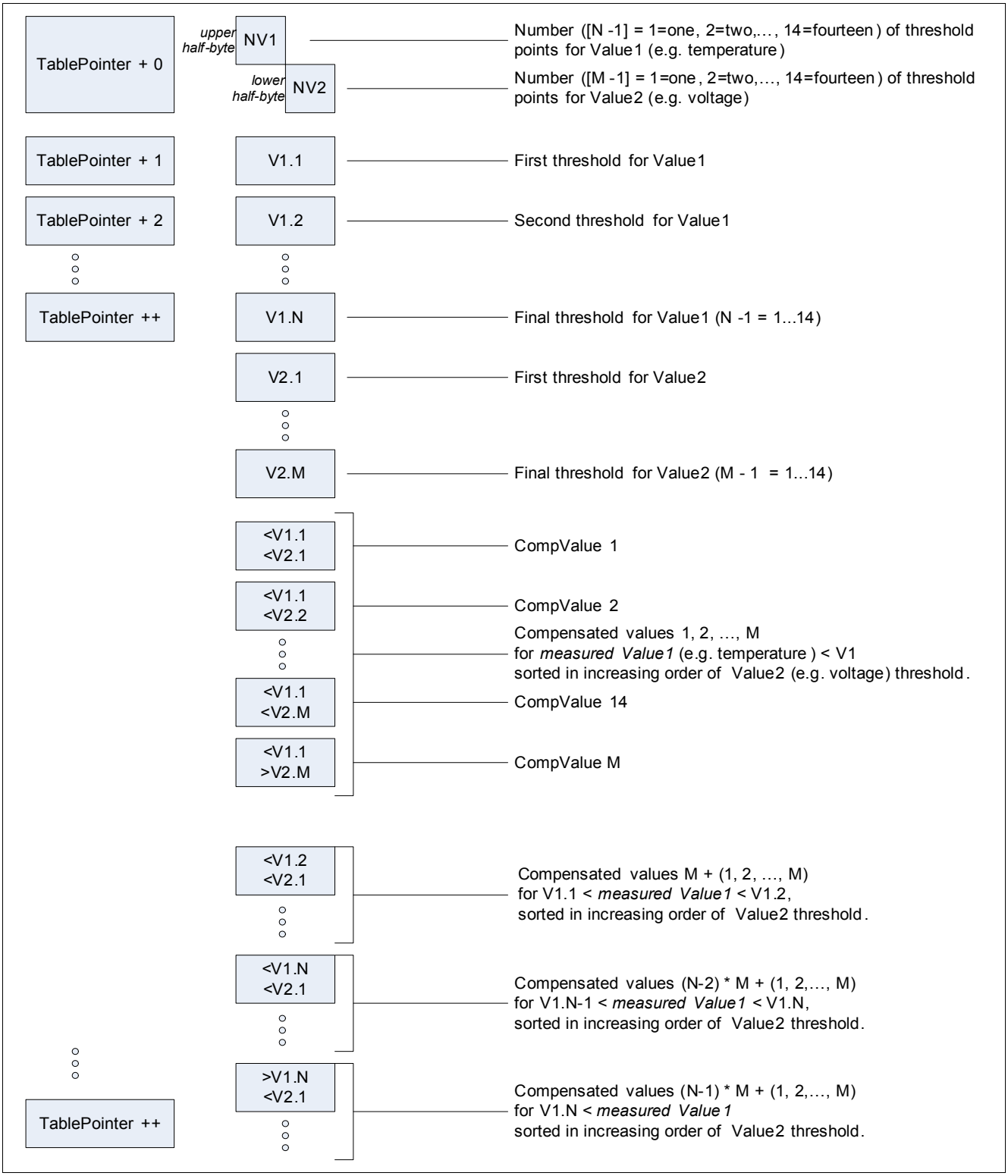


Figure 8 Lookup table organization

2.21.5 Outputs

Table 91 GetCompValue: Output values

Register/ Address	Type	Name	Description
R7	unsigned char	Compensated Value	Returns the compensated value from the lookup table defined by selected column and row

2.21.6 Resource Usage

Table 92 GetCompValue: Resources

Type	Used or Modified
Registers	R0, R1, R3, R5, R6, R7
SFR	ACC, DPH, DPL, PSW, SP
Stack	2 Bytes[1)]

1) Two additional bytes, not included here, are needed to call the library function

2.21.7 Execution Information

Table 93 GetCompValue: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time	t	–	619	669	µs	DIVIC = 00_H
Charge Consumption	Q	–	0,929	1,606	µC	DIVIC = 00_H

2.22 Wait100usMultiples()

2.22.1 Description

This function performs a delay of 100 µs multiples. It clears the CPU clock divider SFR DIVIC, initializes the timers according to the delay time, and uses IDLE state during the delay time. In case an unexpected resume event occurs, the IDLE state will be left and the function remain in RUN state until the delay is elapsed. The total duration of the delay can be determined by an offset time of 28 µs plus the multiplication of the input parameter value with 100µs. The maximum delay time is limited in practice by the watchdog timer, which may be reset by the application prior to the function call.

2.22.2 Actions

- Set SFR DIVIC = 00_B
- Configure timers according to the delay time
- Enter IDLE state
- Delay until timer is elapsed

2.22.3 Prototype

void **Wait100usMultiples**(unsigned int **Counter**)

2.22.4 Inputs

Table 94 Wait100usMultiples: Input Parameters

Register / Address	Type	Name	Description
R6 (MSB) R7 (LSB)	unsigned int	Counter	Number of 100µs multiples. Duration[µs] = Counter x 100µs

2.22.5 Outputs

Table 95 Wait100usMultiples: Output values

Register/Address	Type	Name	Description

2.22.6 Resource Usage

Table 96 Wait100usMultiples: Resources

Type	Used or Modified
Registers	R6, R7
SFR	ACC, B, CFG0, CFG2, DIVIC, DPTR, PSW, TCON, TH0, TL0, TH1, TL1, TMOD
Stack	0 Bytes[1)]

1) Two additional bytes, not included here, are needed to call the library function

2.22.7 Execution Information

Table 97 Wait100usMultiples: Execution Time and Charge Consumption

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Execution Time	t_{100}	–	131	138	µs	DIVIC is 00_H, counter = 1
	t_{600}	–	636	682	µs	DIVIC is 00_H, counter = 6
	t_{+100}	–	100	109	µs	DIVIC is 00_H, counter is increased
Charge Consumption	Q_{100}	–	0,113	0,19	µC	DIVIC is 00_H, counter = 1
	Q_{600}	–	0,393	0,788	µC	DIVIC is 00_H, counter = 6
	Q_{+100}	–	0,057	0,12	µC	DIVIC is 00_H, counter is increased

3 Reference Documents

This section contains documents used for cross- reference throughout this document.

[1] SP27 Datasheet

w w w . i n f i n e o n . c o m

Published by Infineon Technologies AG