

– preliminary –

E909.05 and E909.06 firmware additional

Using Firmware

User space mapping, I²C- and SPI-Protocol

Interrupt- and Callbackhandling

Version 4.01
29.07.2010



Mechaless Systems GmbH
Albert-Nestler-Str. 10
76131 Karlsruhe
Germany

Contents

1	LIMITED WARRANTY	4
2	Introduction	5
2.1	The firmware API	5
3	Working with the HALIOS[®]-Chip firmware API	6
3.1	Integrating the HALIOS [®] -Chip firmware library into the application project	6
4	Firmware reference	7
4.1	The Special Function Register area (SFR)	7
4.1.1	Buffer size of SFR and User space	9
4.1.2	Read constant values	9
4.1.3	SYSTEM STATE	10
4.1.4	COMM_DEVICE	10
4.1.5	SAMPLE_TIME	11
4.1.6	Special mode, Special in, Special out	13
4.1.7	TRIMOSCI	14
4.1.8	TRIMWKUP	14
4.1.9	TIME_STAMP	14
4.1.10	SFREQ - HALIOS [®] sending frequency	14
4.1.11	GYRATOR_SETTLING	15
4.1.12	LOOPCOUNT	15
4.1.13	LEDCONF0 .. 11	15
4.1.14	CLOCKCONF 0 .. 11	15
4.1.15	CONF_A 0 .. 11	15
4.1.16	CONF_B 0 .. 11	15
4.1.17	CONF_COMP 0 .. 11	15
4.1.18	PRE_AMP 0 .. 11	16
4.1.19	REPEAT_LOOP	16
4.1.20	START 0 .. 11	16
4.1.21	CNTRESULT 0 .. 11	16
4.1.22	MEANRESULT 0 .. 11	16
4.2	The User Space	16
4.3	The I²C protocol	16
4.3.1	The I ² C module	17
4.3.2	Read command: 0x00/0x01	17
4.3.3	Write command: 0x80/0x81	18

4.4	The SPI protocol	19
4.4.1	Configuration of SPI module	20
4.4.2	The read command: 0x0dnn	20
4.4.3	The write command: 0x8dnn	20
4.4.4	SPI example sequence	21
4.5	Using own interrupt functions.....	22
4.6	Using interrupt callbacks.....	23

1 LIMITED WARRANTY

THERE IS NO WARRANTY FOR THE CORRECTNESS OF THIS PUBLICATION OR THE RELATED CODE EXAMPLES OR RELATED PROGRAMS TO THE EXTENT PERMITTED BY APPLICABLE LAW EXCEPT WHEN OTHERWISE STATED IN WRITING. THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PUBLICATION OR THE PUBLISHED CODE EXAMPLES OR RELATED PROGRAMS "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PUBLICATION OR THE PUBLISHED CODE EXAMPLES OR RELATED PROGRAMS IS WITH YOU. SHOULD THE PUBLICATION OR THE PUBLISHED CODE EXAMPLES OR THE RELATED PROGRAMS PROVE TO BE INCORRECT OR DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT, UNLESS REQUIRED BY APPLICABLE LAW, WILL THE COPYRIGHT HOLDER OR ANY OTHER PARTY BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THIS PUBLICATION OR THE PUBLISHED CODE EXAMPLES OR THE RELATED PROGRAMS (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR FAILURE OF THE PROGRAMS/PROGRAM EXAMPLES TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF THE COPYRIGHT HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Mechaless Systems GmbH
Albert-Nestler-Str. 10
76131 Karlsruhe
Germany

<http://www.mechaless.eu>
info@mechaless.eu

Phone.: +49 (0)721 / 62698-0
Fax: +49 (0)721 / 62698-11

Additional information can be found in the following documents:

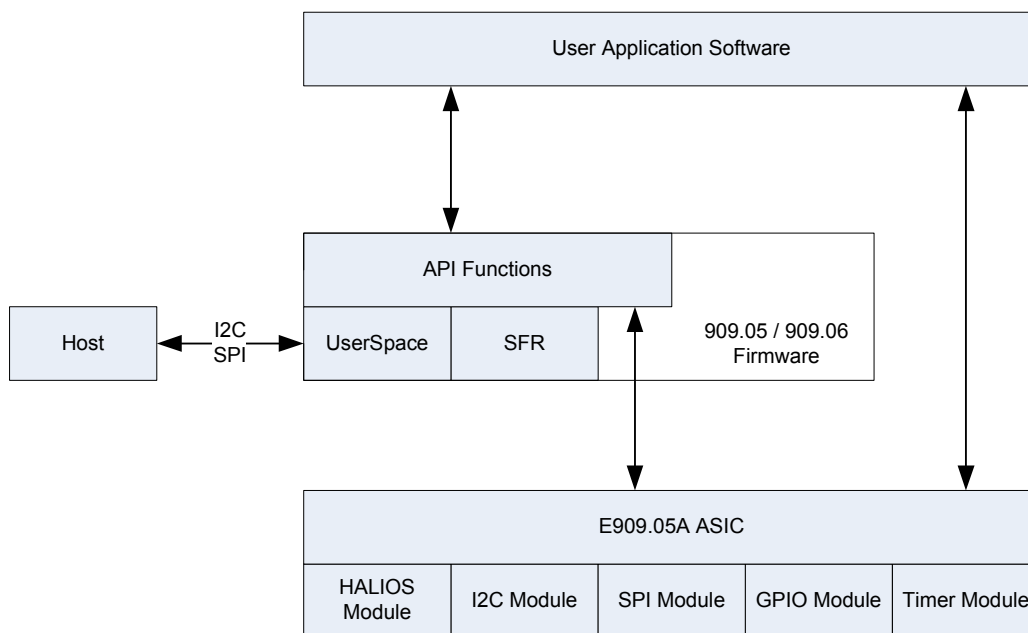
- HALIOS® Basics
- HACo Operating Instructions
- Data specification sheet regarding IC used (e.g. E909.05 or E909.06)
- Optic basic principles
- E909.05A USB library specification
- HALIOS@Tools library specification
- Coordinates 3D library specification

2 Introduction

The HALIOS[®]-Chip is an integrated circuit implementing the sensor system, a 16 BIT CPU core (EL16) and some peripherals. The 16 BIT EL16 CPU core of the HALIOS[®]-Chip can be programmed in C. The API of the firmware library makes available the HALIOS[®] sensor system and the peripherals of the HALIOS[®]-Chip for the application software engineer. Further more a communication protocol is implemented to afford the communication between the HAcO PC software and the HALIOS[®]-Chip firmware.

2.1 The firmware API

The firmware is an easy-to-use interface to the HALIOS[®]-Chip and its parts. It consists of well documented function calls, which a client application can use via calls to perform several tasks with the HALIOS[®]-Chip. In a layer-based view of the system, the firmware API can be seen as a layer which abstracts complex functionality. The firmware API provides a simple way to use the HALIOS[®] sensor system, the STOP timer, the TIMER 1 module, the GPIO module, the I²C and SPI interface and the IRQ module.



Picture 1: Layer view of the 909.05 and 909.06 firmware

3 Working with the HALIOS[®]-Chip firmware API

Integration of the HALIOS[®]-Chip firmware API into the system requires that the system designer implements the components discussed below, together with a proper handling of their implementation aspects.

3.1 Integrating the HALIOS[®]-Chip firmware library into the application project

Every application software for the HALIOS[®]-Chip has to include the following header file (distributed by Mechaless Systems GmbH):

```
firmware.h      Header file for all firmwarefunctions
```

For the library these folders are needed (distributed by Mechaless Systems GmbH):

```
include        All needed header files  
misc          common used files for all libraries  
lib           contains the libraries
```

With the HALIOS[®]-Chip firmware library the following files are also delivered by Mechaless Systems GmbH with the application notes:

```
main.c        a sample application note how to use the API functions of the firmware,  
              interrupt functions and callbacks  
main.h        some constants for the sample application note  
config.h     contain the usage of interrupt handling. If you want use your own interrupt  
              function, please change the 0 to 1 at the corresponding define (See  
              application note "a2_use_isr"
```

For development you can use a GCC IDE (only E909.05). Mechaless Systems GmbH provide this at the HALIO-Evaluation CD or at our homepage.

The recommended C-IDE is the IAR Embedded Workbench. An adapted trial-version can be obtained from ELMOS Semiconductor AG.

4 Firmware reference

The interfaces to the firmware beside the API are the I²C and the SPI device. The 909.06 also provides an SCI interface for LIN communication. The communication devices exclusively have access to the firmware internal special function register area (SFR) and the application wide User space (see Picture 1: Layer view).

4.1 The Special Function Register area (SFR)

To control the HALIOS[®] regulation and some other internal states, the firmware manages a section of the available RAM space for internal usage.

The SFR has following structure:

Address	Name	Comment	Usage
0	BUFSIZE	Low Byte: Size of gui_sfr_buffer High Byte: Size of gui_user_buffer	Internal
1	READ_CONST_CMD	0: Constant output off 1: Chip name 2: Last Reset Reason 3: Installed library 4: Library version 5: Project number 6: Maximum Loops 7: Maximum repeated measurement	Constant reading mechanism
2	READ_CONST_CMD_DETAIL	Details for READ_CONST_CMD	
3	READ_CONST_STATUS_HIGH	Higher 16_bit of constant	
4	READ_CONST_STATUS_LOW	Lower 16-bit of constant	
5	SYSTEM_STATUS	0: RUN: keep running 1: STANDBY: go to standby 2: STOP: stop the main clock 3: OFF: stop wakeup clock	Firmware configuration
6	COMM_DEVICE	0: I ² C and SPI disabled BIT0: I ² C enable BIT1: SPI enable	
7	SAMPLE_TIME	Must be a value of {2, 4, 6, ..., 32}	
8	SPECIAL_MODE	0: Special mode off 1: I ² C status (Special in = I2C_CTRL, Special out = I2C_STATUS) 2: Read RAM (Special in = address, Special out = result) 3: Write RAM (Special in = address, Special out = value) 4: Write Flash (Special out = result)	Special mode

Address	Name	Comment	Usage
		5: Restore Flash (Special out = result) 6: Check optical System	
9	SPECIAL_INDATA	Input for special mode	
10	SPECIAL_OUTDATA	Output for special mode	
11	TRIMOSCI	Trim the master clock oscillator for the EL16 CPU (Only at E909.05)	Oscillator trim values for E909.05A
12	TRIMWKUP	Trim the oscillator for the wakeup timer. (Only at E909.05)	
13	TIME_STAMP	Timestamp for the communication device	Global HALIOS® register
14	HALIOS_FREQ	HALIOS® send frequency select (sfreq) Frequency = FSYS / (sfreq * 16); FSYS = 8MHz Sfreq = 3..7 (Resetvalue = 4)	
15	GYRATOR_SETTLING	Gyrator Settling mode during Warmuploop (BIT 10 at register Start Value Counter)	
16	LOOPCOUNT	Amount of active loops in the system (1 .. Maximum Loops)	
17	LEDCONF 0	LED and measurement configuration	Loop dependency HALIOS® register
18	CLOCKCONF 0	Measurement Configuration HALIOS® Clock	
19	CONF_A 0	Current configuration phase A	
20	CONF_B 0	Current configuration phase B	
21	CONF_COMP 0	Current configuration compensator offset	
22	PRE_AMP 0	Preamplifier Configuration	
23	REPEAT_LOOP	How often this loop will be measured	
24	START 0	Start value counter	Loop
25	CNTRESULT 0	Measurement result: counter value	Results
26	MEANRESULT 0	Measurement result: mean value	
...	...	Configurations and Results for Loops 1 to 14	
167	LEDCONF 15	LED and measurement configuration	Loop dependency HALIOS® register
168	CLOCKCONF 15	Measurement Configuration HALIOS® Clock	
169	CONF_A 15	Current configuration phase A	
170	CONF_B 15	Current configuration phase B	
171	CONF_COMP 15	Current configuration compensator offset	
172	PRE_AMP 15	Preamplifier Configuration	
173	REPEAT_LOOP 15	How often this loop will by measured	
174	START 15	Start value counter	Loop
175	CNTRESULT 15	Measurement result: counter value	Results

Address	Name	Comment	Usage
176	MEANRESULT 11	Measurement result: mean value	

Table 1: Table for SFR addresses

4.1.1 Buffer size of SFR and User space

(Address 0)

This configuration word contains a constant word. It shows how many words are reserved in the SRAM area for the SFR in the low byte and in high byte the size of User space. If high byte is 0, no User space exist.

4.1.2 Read constant values

(Address 1..4)

With the command word READ_CONST_CMD the read out of several constant values from the ASIC is started. After the firmware set the READ_CONST_STATUS registers this register is set return to 0.

The following constants are readable:

- 0: No change will be done at the high and low READ_CONST_STATUS registers.
- 1: Read out the “Chip name” from the ASIC. The output is ASCII-coded. With the register READ_CONST_CMD_DETAIL you define which part of name will be read (Set only one bit):
 - BIT0: READ_CONST_STATUS_LOW shows the first two letters
READ_CONST_STATUS_HIGH shows second two letters
 - BIT1: READ_CONST_STATUS_LOW shows third two letters
READ_CONST_STATUS_HIGH shows fourth two letters
 - Other or combines: Both registers will set to 0xFFFF
- 2: The READ_CONST_STATUS_LOW register shows the “Last Reset Reason”. READ_CONST_STATUS_HIGH will set to 0. For details please see chapter “Analog Control Module” / “Register reset source status”. READ_CONST_CMD_DETAIL will be ignored.
- 3: The READ_CONST_STATUS_LOW register shows which HALIOS® libraries are installed in the device. Each installed library sets a bit in this register. The bit pattern shows which libraries are installed. READ_CONST_STATUS_HIGH will set to 0. READ_CONST_CMD_DETAIL will be ignored. “Table 2” shows the bit patterns for the libraries:

Bit pattern	Library name
0x0001	Firmware library
0x0002	HALIOS® Tools library
0x0004	Coordinates 3D library
0x0008	USB library
0x0010	Reserved

.....
0x4000	Reserved
0x8000	Application version

Table 2: Libraries bit pattern

If for example the firmware library and the HALIOS® Tools library are installed, the READ_CONST_STATUS_LOW register contains the value:

```
Example:
0x0001 + 0x0002 = 0x0003
```

- 4: Set the Library version number to the READ_CONST_STATUS_LOW register. The READ_CONST_CMD_DETAIL register define which library version number will be set. For details see “Table 2”. All unknown values at READ_CONST_CMD_DETAIL will cause 0xFFFF. The library version of the lowest bit will be shown.

```
Example:
411 (0x019B) is v4.11
```

- 5: It is possible to set a string of eight characters to the firmware (use function `setProjectNumber(const uint8_t* pui8_String)` to set this string). The result at READ_CONST_STATUS_LOW and READ_CONST_STATUS_HIGH is ASCII coded. The READ_CONST_CMD_DETAIL will define which values are set:
 - BIT0: READ_CONST_STATUS_LOW shows first two character
READ_CONST_STATUS_HIGH shows second two character
 - BIT1: READ_CONST_STATUS_LOW shows third two character
READ_CONST_STATUS_HIGH shows fourth two character
 - Other or combines: Both registers set to 0xFFFF
- 6: Maximum number of Loops is possible to set for measurement.
- 7: Maximum number of repeats for repeated measurement.

4.1.3 SYSTEM STATE

(Address 5)

This configuration word switches between the possible system states. The mode is used during the function `deviceWaitForTimer()`.

- RUN: System stay in RUN-MODE.
- STANDBY: The CPU is stopped, main clock and wakeup clock are running
- STOP: The CPU and the main clock is stopped (Only I²C and wakeup interrupt can break this mode.)
- OFF: The CPU and the wakeup is stopped (Only I²C can break this mode)

4.1.4 COMM_DEVICE

(Address 6)

This configuration word determines which communication device is active (combinations are possible):

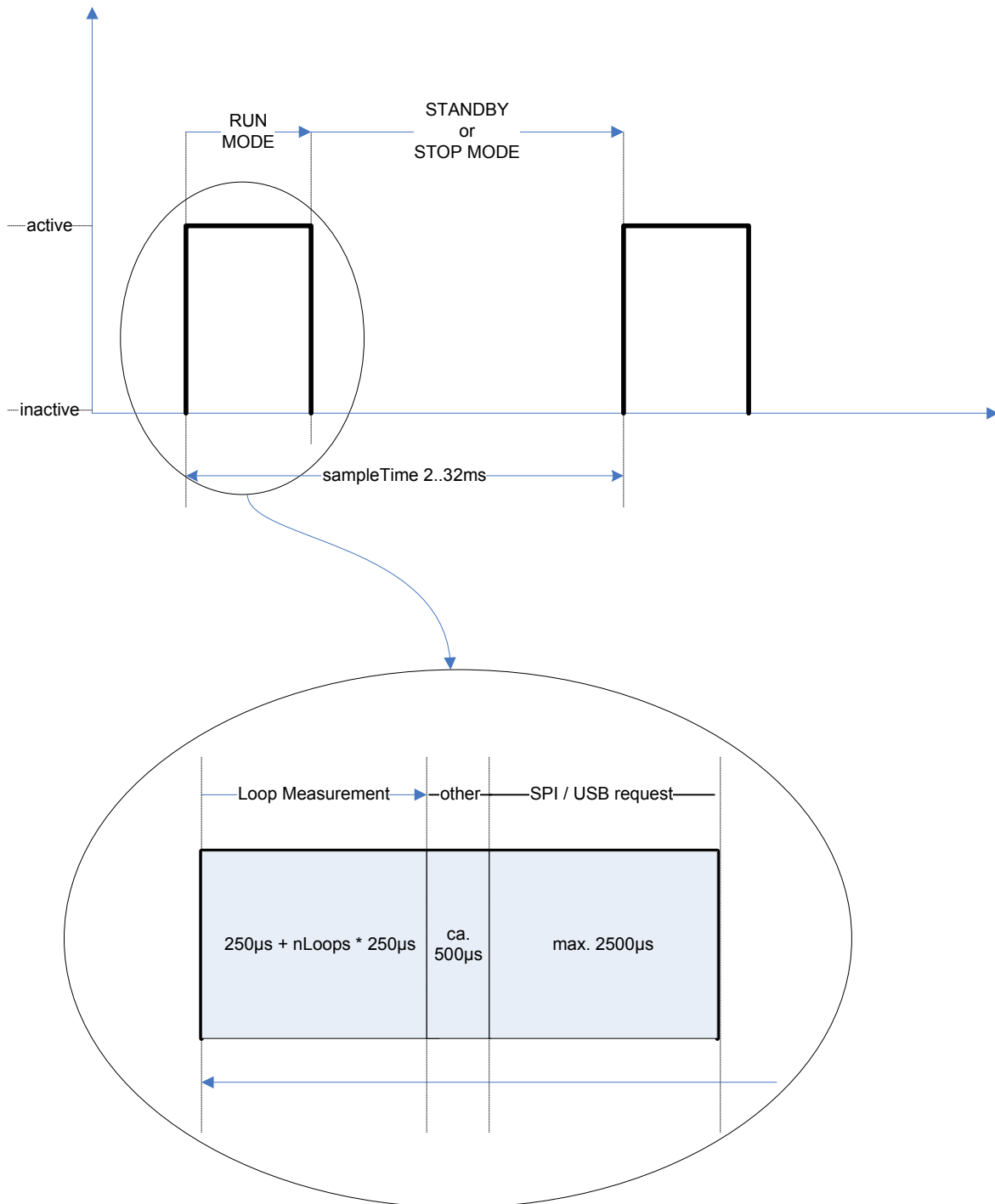
- 0 I²C and SPI interface are disabled
- BIT0 I²C slave interface is enabled. All Interrupts will be set for interrupt triggered handling by firmware. As interface you can access to SFR and User space.
- BIT1 SPI is enabled. GPIOs will set the I/O functionality for SPI. Using SPI as slave.
- Other bits will ignore

4.1.5 SAMPLE_TIME

(Address 7)

This configuration word contains the sample time in milliseconds. Valid values are 2, 4, 6, ..., 32. With the API function `paramGetSampleTime()` the sample time can be read out and with `paramSetSampleTime()` it can be set. The API function `deviceWaitForTimer()` activates the configured power saving mode (see 4.1.3). The `WAKEUP_TIMER` interrupt wakes the CPU up when the sample time is elapsed (Attention: Not in OFF-mode).

Device Timing



Picture 2: Timing diagram for firmware

4.1.6 Special mode, Special in, Special out

(Address 8, 9, 10)

This registers are only useful for software engineering purposes. Please be careful by modifying this register otherwise data stored in FLASH can be lost.

Special mode (8)	Special in (9)	Special out (10)	Comment
0	Not used.	Not used.	Special mode off.
1	Content of the register I2C_CTRL (0x0164).	Content of the register I2C_STATUS (0x0166).	By troubles with the I ² C communication this information can be helpful.
2	Address to read out.	Contents from the given address.	This is useful for monitoring online the hardware registers of the HALIOS [®] -Chip. After the result is written into "Special out" "Special mode" is written back to 0 by the firmware.
3	Address to write to.	Value to write into the address.	This is useful to change online hardware registers of the E909.05A. After the value from "Special out" is written at the address stored in "Special in" "Special mode" is written back to 0 by the firmware.
4	Not used.	>0: Amount of flashed words -1: internal error -2: error while erasing -3: error while flashing	Write the contents of the SFR and the User Space area into FLASH. If the flash write function succeeds in "Special out" the amount of flashed words is written. This functionality can be also used with the API function deviceFlash ().
5	Not used.	>0: Amount of copied words -1: no data found	If valid data are found in FLASH than they are copied into the SFR and User Space area. This functionality can be also used with the API function deviceRestore ().
6	Not used	Result of checking the optical system 0: All parts of the optical system works correct BIT0: LED1 maybe damaged BIT1: LED2 maybe damaged BIT2: LED3 maybe damaged BIT3: LED4 maybe damaged BIT4: LEDC maybe damaged	This call the function haliosCheck()

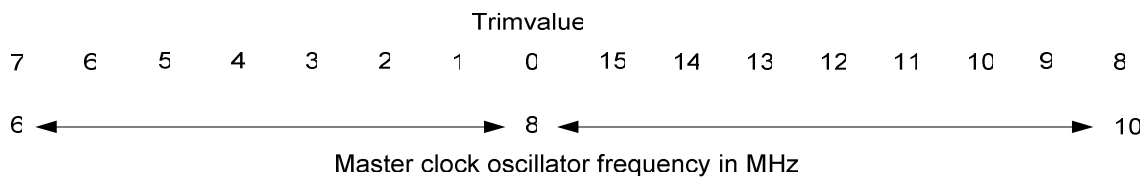
		BIT5: Photo diode damaged Combination is possible	
Any other values.	Not used.	Not used.	No function.

Table 3: Special modes

4.1.7 TRIMOSCI

(Address 11)

This function is only used at E909.05. Trims the master clock oscillator in the way shown in "Picture 3":

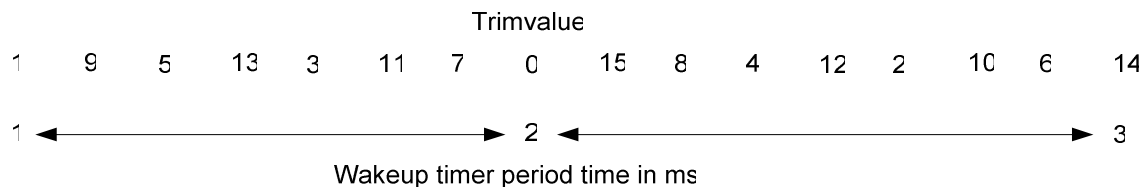


Picture 3: Trim values for the master clock oscillator

4.1.8 TRIMWKUP

(Address 12)

This function is only used in E909.05. It trims the wakeup timer period time in the way shown in "Picture 4":



Picture 4: Trim values for the wakeup timer period

4.1.9 TIME_STAMP

(Address 13)

When the measurement is completed this word is increased by one. So every measurement has its own time stamp. Therefore the user is able to do time measurements by multiplication the difference between two time stamps with the sample time.

4.1.10 SFREQ - HALIOS® sending frequency

(Address 14)

With this word the HALIOS® frequency can be configured. The standard value is 4. It is possible to set this register to a value between 3 and 7.

The equation is:

$$\text{Frequency} = \frac{FSYS}{16 * SFREQ}; FSYS = 8MHz$$

With standard this is:

$$\frac{8000kHz}{16 * 4} = 125kHz$$

4.1.11 GYRATOR_SETTLING

(Address 15)

If this word is set to 1 the fast gyrator settling is active (E909.06 only). For details please refer to data sheet.

4.1.12 LOOPCOUNT

(Address 16)

This configuration word determines how many loops are active. This value can be set with the API function `haliosSetLoopCount()`.

4.1.13 LEDCONF0 .. 11

(Address 17 + (loopindex * 10))

Stores the LED and measurement configuration of the assigned loop.

4.1.14 CLOCKCONF 0 .. 11

(Address 18 + (loopindex * 10))

Store the configuration for the polarity for each LED. (Not used at e909.05)

4.1.15 CONF_A 0 .. 11

(Address 19 + (loopindex * 10))

Stores the current configuration in phase A for range and offset (see specification from the device for details).

4.1.16 CONF_B 0 .. 11

(Address 20 + (loopindex * 10))

Stores the current configuration in phase B for range and offset.

4.1.17 CONF_COMP 0 .. 11

(Address 21 + (loopindex * 10))

Stores the current configuration for the compensator offset and the DC_OFFSET for temperature compensation.

4.1.18 PRE_AMP 0 .. 11

(Address 22 + (loopindex * 10))

Stores the configuration for the preamplifier. (Not used at e909.05)

4.1.19 REPEAT_LOOP

(Address 24 + (loopindex * 10))

The number of repeats for this loop. Standard is 0. Than this loop is measured one time. Set this to value higher than 0 to reduce noise. After the last measurement the result is the average from the number of measurements. The value is set here is the exponent to 2:

- 1: This loop is measured twice.
- 2: This loop is measured four times.
- 3: This loop is measured eight times.

Attention: Take care to the runtime of the system. Each measurement needs 250 µs!

4.1.20 START 0 .. 11

(Address 24 + (loopindex * 10))

Stores the start value and the step size of the assigned measurement.

4.1.21 CNTRESULT 0 .. 11

(Address 25 + (loopindex * 10))

Stores the 10 bit measurement result.

4.1.22 MEANRESULT 0 .. 11

(Address 26 + (loopindex * 10))

Stores the 12 bit measurement result.

4.2 The User Space

For data (variables, parameters) of the application software that should be accessed, e.g. via HALIOS Config (HACo) software or via I²C. This area is called *User space*.

With the API functions paramSetValue() and paramGetValue() it is possible to modify the User Space by the application software.

4.3 The I²C protocol

The I²C protocol contains four commands:

- Read command for SFR: 0x00
- Write command for SFR: 0x80
- Read command for User space: 0x01
- Write command for User space: 0x81

All other command bytes can be used with the callback "callback_i2c_rx" as user defined commands.

4.3.1 The I²C module

For a detailed description of the I²C interface please refer to the data sheet.

In the firmware the 7-bit slave address for I²C device default is defined with 0x58. The complete address byte with READ/WRITE bit is defined as

```
I2C SLAVE ADDRESS + READ-BIT = 0xb1  
I2C SLAVE ADDRESS + WRITE-BIT = 0xb0
```

To set an another slave address in the I²C control register (I2C_CTRL) you have to do this after calling the initialization function, like in the example below (otherwise it will be overwritten in the initialization function):

```
...  
// Initialize the E909.05A device  
haliosInitialize();  
// Clear the address field  
I2C_CTRL &= ~(0x03 << I2C_SLAVE_ADDR);  
// Set the address to 0x59  
I2C_CTRL |= 0x01 << I2C_SLAVE_ADDR;  
...
```

4.3.2 Read command: 0x00/0x01

With 0x00 communicate to the SFR, 0x01 communicate to User space. With the read command a word (16 BIT) of the SFR or the User space can be read out.

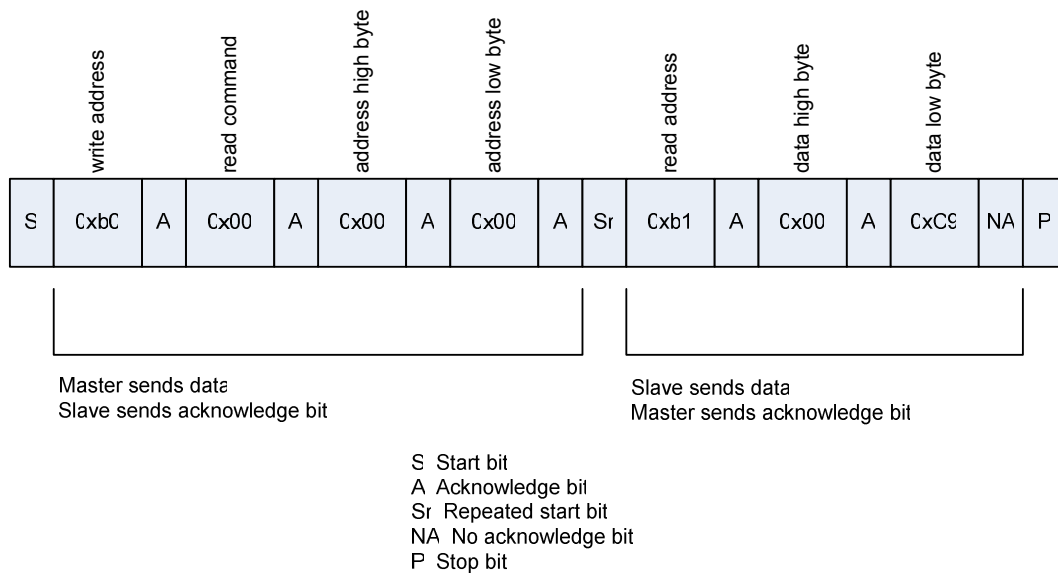
Command	
Byte 1	0x00
Byte 2	The high-byte of the address
Byte 3	The low-byte of the address

Table 4: I²C read command

Response	
Byte 1	The high-byte of the data
Byte 2	The low-byte of the data

Table 5: I²C read response

Example: Read data from address 0x0000 out (result is 0x00c9 = 201)



Picture 5: I²C read command example

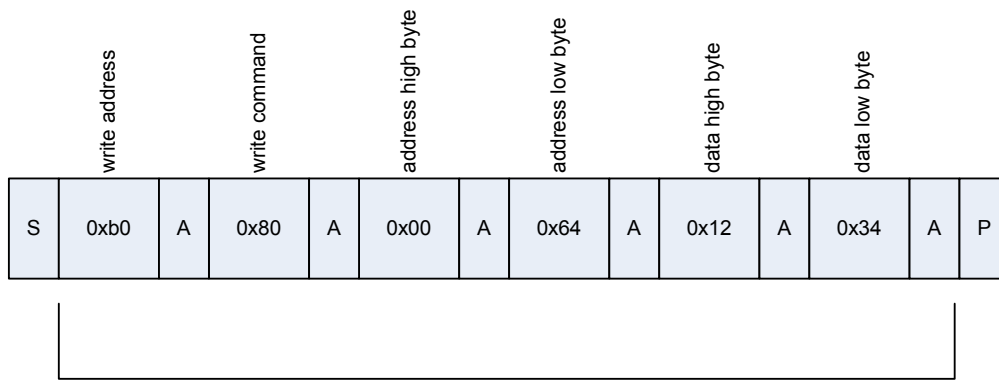
4.3.3 Write command: 0x80/0x81

With the write command a word (16 BIT) is written from the I²C host to the SFR (0x80) or the User space (0x81).

Command	
Byte 1	0x80
Byte 2	The high-byte of the address
Byte 3	The low-byte of the address
Byte 4	The high-byte of the data word
Byte 5	The low-byte of the data word

Table 6: I²C write command

Example: Write the data word 0x1234 to the address 0x0064 in SFR



Master sends data
Slave sends acknowledge bit

S: Start bit
A: Acknowledge bit
Sr: Repeated start bit
NA: No acknowledge bit
P: Stop bit

Picture 6: I²C write command example

4.4 The SPI protocol

The SPI protocol contains four commands to access SFR and User space area (nn is the address):

- Read command from SFR: 0x00nn
- Read command from User space: 0x08nn
- Write command to SFR: 0x80nn
- Write command to User space: 0x88nn

Used Symbols: 0xcdnn

- c: command; 0 → read, 8 → write
- d: destination; 0 → SFR, 8 → User space
- nn: address; range 0x00 until 0xff

The read command performs two actions:

1. Read out data from the given address
2. Set this address for the write command, too.

4.4.1 Configuration of SPI module

For a detailed description of the SPI interface see data sheet.

The firmware configures the SPI module in the following way:

- E909.05: GPIO 2..5 are configured as SPI function pins
- E909.06: GPIO 4..7 are configured as SPI function pins
- MSB first
- 1st edge shift, 2nd edge sample
- Clock off level 0
- Slave mode
- Low and high water FIFO levels set to 2 → SPI in 16 bit mode.

4.4.2 The read command: 0x0dnn

With the read command a word (16 BIT) of the SFR or of the User Space is read out.

Command	
Word 1	0x0dnn

Table 7: SPI read command

Valid read commands are: 0x0d00 to 0x0dff.

Response	
Word 1	The data word

Table 8: SPI read response

4.4.3 The write command: 0x8dnn

With the write command a word (16 BIT) is written byte-wise to the SFR or the user space.

Command		
Word 1	0x8dnn	The low-byte of the Data word
Word 2	0x8dnn	The high-byte of the data word

Table 9: SPI write command

4.4.4 SPI example sequence

Modify and verify the sample time (address: 7 = 0x0007):

- Write the new sample time (4 ms)
- Read the written sample time out

Attention: Because of the synchronous behaviour of the SPI interface, the read command must be sent twice!

•

Command	Response	Comment
0x0007	0xnxxx	Set the address, response is from last command
0x8004	0x000a	Write data low-byte, response from last read command
0x8000	0x0004	Write data high-byte, response from last write command
0x0007	0x0000	Read data word, response from last write command
0x0007	0x0004	Read data word, response from last read command

Table 10: SPI example

4.5 Using own interrupt functions

Change the corresponding define in config.h to that interrupt you want to use your own function from 0 to 1.

You have to set the name of your function like that define without the USE_X_.

Example:

In config.h:

```
/** Interrupt vector number 9, timer1 event. */  
#define USE_X_isr_timer1_event 1
```

Add this to main.c:

```
void isr_timer1_event ( void )  
{  
    P0OUT ^= BIT0;  
  
    /*  
     * You have to clear the interrupt flag before returning to the  
     * application.  
     */  
    TIMER1_IRQCLR = TIMER_IRQCLR;  
}
```

4.6 Using interrupt callbacks

With callbacks it is possible to extend the interrupt functionality instead of replace the firmware function. In this way the application software can extend the I²C or SPI protocol with user defined commands without losing the existing protocol.

If you want use one of the callbacks you have to deselect the callback-library by set the switch in UESTudio under Project \ Project Settings... \ Registercard Settings \ Compiler Options \ “Use Callback Lib” to “no”. In IAR you only have to remove the libcallback.r43.

Then you have to include the following functions. In this example a pin toggle after each measurement of the HALIOS[®] part.

```
/**
 * dummy function that provides no functionality
 */
void callback_i2c_wakeup ( void )
{
}

/**
 * dummy function that provides no functionality
 */
void callback_i2c_rx ( void )
{
}

/**
 * dummy function that provides no functionality
 */
void callback_i2c_high_water ( void )
{
}

/**
 * dummy function that provides no functionality
 */
void callback_spi_high_water ( void )
{
}

/**
 * dummy function that provides no functionality
 */
void callback_wakeup ( void )
{
}

/**
 * This callback function will be called when a measurement has finished.
 * Keep in mind, that this function is executed in the interrupt handler
 * and thus should be kept as short as possible.
 */
void callback_measure_rdy ( void )
{
    /** toggle GPIO PIN2 */
    P0OUT ^= BIT2;
}
```