

HALIOS[®] Tools

Software-Specification

Version 4.0
04.06.2010



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

Contents

1	Foreword.....	3
2	Introduction	4
2.1	Overview.....	4
3	Working with the HALIOS® Tools library	5
3.1	Integrating the library into the application project	5
4	HALIOS® Tools reference	6
4.1	Get HALIOS® into the operating point	6
4.2	Filtering the loop values	6
4.3	Calibrating the loop values.....	8
4.4	Initialize temperature compensation.....	9
4.5	Temperature compensation.....	10
4.6	LevelSwitch module	10

1 Foreword

This document describes the contents of the HALIOS® Tools library in detail and shows an example how to use this library. Please read this guide completely and carefully. Further information can be found on our website: <http://www.mechaless.com>

If you have any questions or comments regarding this product or documentation, please contact:

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

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

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)
- E909.05 or E909.06 firmware library documentation
- HALIOS firmware specification
- USB library documentation

2 Introduction

This library contains some functions to make it easier to work with the HALIOS[®] sensor signals.

2.1 Overview

The library HALIOS[®] tools depends on the HALIOS[®] Firmware The application has to link against following libraries:

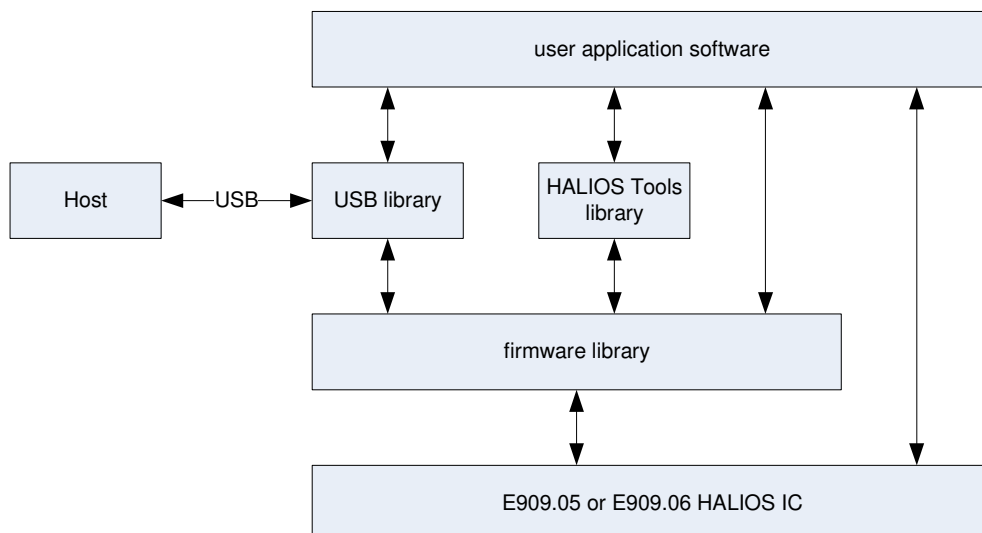
- ▶ lib_firmware_05 or lib_firmware_06
- ▶ lib_haliostools

For USB communication with HALIOS[®] Config tool the application has to be linked against

- ▶ lib_usb

too.

In **“Fehler! Verweisquelle konnte nicht gefunden werden.”** a layer view of an application with E909.05 or E909.06 is shown. This application uses the firmware for E909.05 or E909.06, the USB library, the HALIOS[®] Tools library and the Coordinates3D library.



Picture 1: Layer model

3 Working with the HALIOS® Tools library

Integration of the HALIOS® Tools library 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 library into the application project

Any application software has to include the following header file (distributed by Mechaless Systems GmbH):

For E909.05 or E909.06:

```
firmware.h          (hardware specification)
usb.h               (header file of the USB library)
haliostools.h      (header file of the HALIOS® Tools library)
```

Firmware libraries for IAR:

```
lib_firmware05.r43 or lib_firmware06.r43  (the firmware library)
lib_callback.r43                          (fill callback functions if user don't define)
lib_usb.r43                               (only needed for USB communication with HAcO)
lib_haliostools.r43                       (haliosTools)
```

Firmware libraries for GCC:

```
lib_firmware05.a or lib_firmware06.a      (the firmware library)
lib_callback.a                            (fill callback functions if user don't define)
lib_usb.a                                 (only needed for USB communication with HAcO)
lib_haliostools.a                        (haliosTools)
```

To test and for development you can use the GCC IDE. The recommended C-IDE for further software development is the IAR Embedded Workbench.

4 HALIOS® Tools reference

The HALIOS® Tools are a library which contains functions to help working with the HALIOS® sensor signals. They cover the domains filtering, calibrating and temperature compensation of the signals.

4.1 Get HALIOS® into the operating point

During one measurement, started with the function call of “haliosMeasure()”, the analog HALIOS® peripheral does 25 measurements. In each measurement the HALIOS® module is able to do 1, 2, 4 or 8 steps. This means when acceleration is turned off that after each haliosMeasure() call the sensor signal can change at maximum at 25 counter values. When acceleration is turned on and the HALIOS® module is not in his operating point than one haliosMeasure() call can enforce a maximum change of 64 counter values. When the HALIOS® module is already in his operating point one haliosMeasure() call can enforce a maximum change of 200 counter values.

To relieve the application software engineer from this detail the HALIOS® Tools library has the function

```
void haliosWarmup(uint16_t times)
```

which do an empty measurement for each configured loop multiple times. For easy usage of the function two constants are defined for the parameter times:

```
#define HALIOS_WARMUP_FULL 6
#define HALIOS_WARMUP_HALF 4
```

The constant HALIOS_WARMUP_FULL enforces haliosMeasure() to do as much measurements as needed to reach a maximum change of 1024 counter values. HALIOS_WARMUP_HALF therefore enforces a maximum change of 512 counter values.

4.2 Filtering the loop values

One main part of the HALIOS® Tools library is the low pass filter function. The signature of the filter function is as follows:

```
void haliosFilterLoop(uint16_t loopNr, uint16_t *loopFilt, HALIOS_FILT border3db, int filterBreak, haliosFilt *filter);
```

loopNr is the number of the loop which has to be filtered. The value should be between 0 and 11.

*loopFilt is the filtering result.

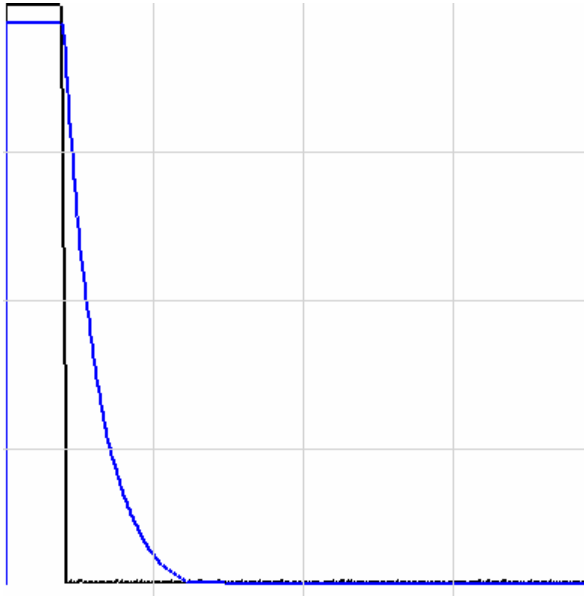
border3db is the three decibel barrier frequency of the low pass filter. The possible values for the barrier frequency are stored in the enumeration HALIOS_FILT:

```
typedef enum {
    HALIOS_FILT_0,      /**< Filtering off. */
    HALIOS_FILT_2,      /**< 3dB border at 2*SampleTime. */
    HALIOS_FILT_4,      /**< 3dB border at 5*SampleTime. */
    HALIOS_FILT_8,      /**< 3dB border at 10*SampleTime. */
    HALIOS_FILT_16,     /**< 3dB border at 20*SampleTime. */
    HALIOS_FILT_32,     /**< 3dB border at 39*SampleTime. */
    HALIOS_FILT_64      /**< 3dB border at 79*SampleTime. */
} HALIOS_FILT;
```

If e.g. the value HALIOS_FILT_32 is chosen and the sample time is configured to 10 milliseconds than the three decibel barrier frequency of the low pass filter is

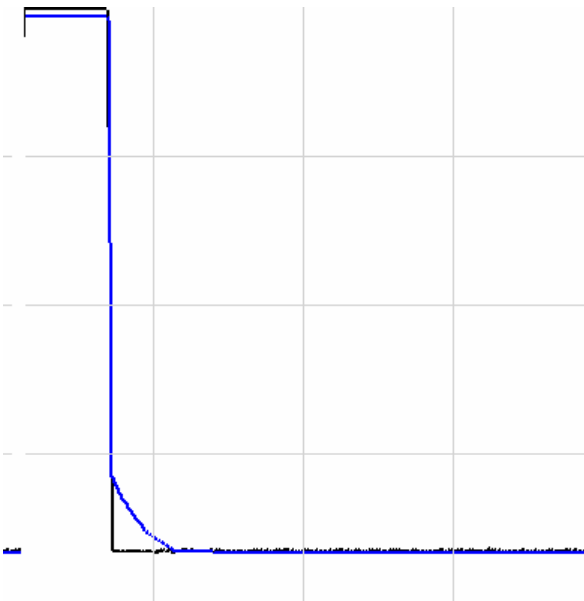
$$f_{3dB} = \frac{1}{39 \cdot 10ms} \approx 2.56Hz$$

In Picture “Picture 2” this behaviour is illustrated at a signal which is similar to a step response.



Picture 2: lowpass filter without filterbreak

To accelerate the filter function the parameter `filterBreak` is used. If the derivation of the raw loop value is higher than `filterBreak` the filtered value is omitted and the raw loop value will be written to `*loopFilt`. A negative value for `filterBreak` disables the filter break mechanism. In “Picture 3” is the same signal with the same filter parameters illustrated, the only difference to the above example is the `filterBreak` is set to 200.



Picture 3: lowpass filter with filterbreak

In this case the filter function is shut off until the difference between the filtered and the raw loop value is greater than 200 counter values.

The parameter `*filter` is a pointer to structure witch store the values of each loop several. This is the exact value of filter in fix-point exposure with 11-bit accuracy. It is recommended to initialize this value by leftshift the current loopvalue and set to this structure.

The filter function `haliosFilterLoop()` works correctly only when it is called periodically in the main loop of the E909.05A application software.

4.3 Calibrating the loop values

The second main part of the HALIOS® Tools library is the calibrating function. Calibration provides compensation a change in the optical coupling during the application. The signature of the calibration function is as follows:

```
Uint16_t haliosCompCalib(uint16_t nr, uint16_t loopValue, int16_t target, int16_t tube,
uint32_t countEnd, uint16_t maxDcnt, haliosCalib *calib);
```

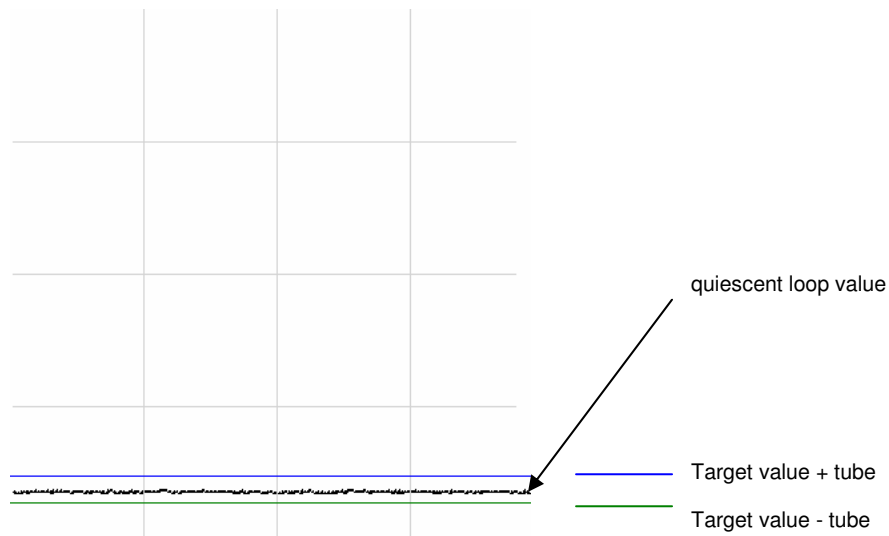
Follow parameters are assigned to the function:

- | | |
|--|---|
| <code>nr</code> | - The number of the loop which has to be calibrated and has to be a number between 0 and 11. |
| <code>loopValue</code> | - The actual value of the signal. It is recommended that the filtered loop value is used. |
| <code>target</code> | - The target value for the calibration. During calibration the internal algorithm tries to set the signal in the tube near to the target value. |
| <code>tube</code> | - If the loop is within the <code>tube</code> borders ($target - tube < loopValue < target + tube$) then the actual loop value is fetched as the new reference value and stored in <code>calib->refVal</code> . |
| <code>countEnd</code> | Value sets the time for calibration. If the internal calibration counter for each loop is higher than this value a calibration is forced. |
| <code>maxDcnt</code> | - Value for movement detection. If the derivation of the loop (<code>calib->dcnt</code>) is higher than this value a movement has been detected. Than the counter value (<code>calib->count</code>) is set to zero. |
| <code>calib</code> | Structure to remember the values when the function is called on the next sample. Please refer also to the header file <i>halioostools.h</i> . |
| <code>calib->refVal</code> | - It is the reference value for the calibration. The value is stored with the actual loop value when a calibration is forced and no movement has been detected. |
| <code>calib->loopLast</code> | - Value stores the last loop value. |
| <code>calib->dcnt</code> | - Value to store the derivation of the loop. Value will be reset when the algebraic sign of the derivation changes. |
| <code>calib->count</code> | - Counter value which is incremented on every function call. A calibration is forced when <code>calib->dcnt</code> exceeds <code>maxDcnt</code> and the signal is out of the <code>tube</code> . |
| <code>calib->autocatch_timestamp</code> | - Value is used in the application to avoid <i>autocatch</i> on every sample. The auto-catch function is implemented in the main application. If auto-catch occurs the application sets the value for <code>countEnd</code> temporarily to zero to force a calibration. |

Important hints:

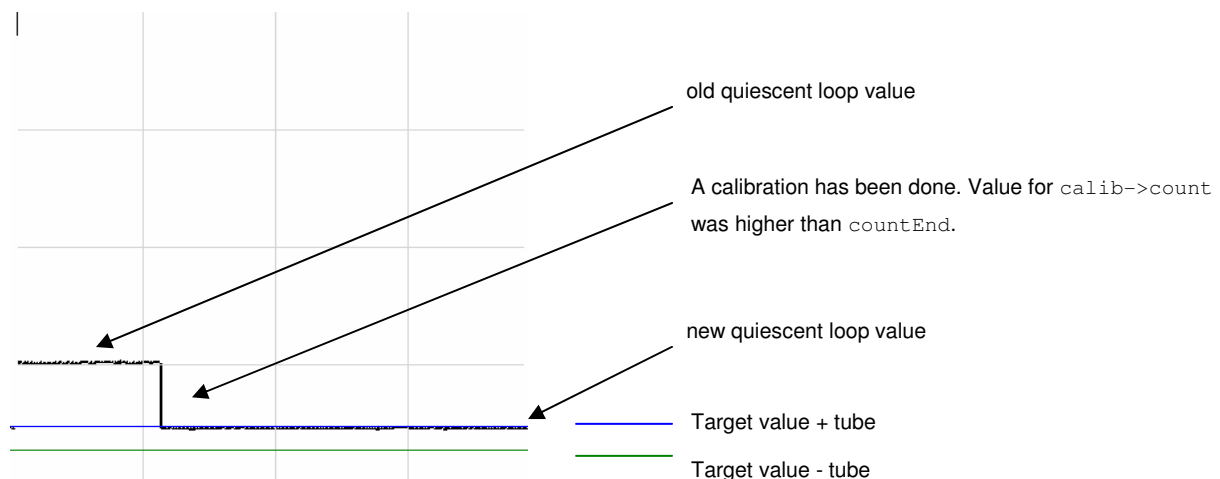
The calibration function `haliosCompCalib()` only works correctly when it is called periodically in the main loop of the application software. The `tube` and the value for `maxDcnt` have always to be higher than the noise on the raw signal. It is recommended to use filtered loop values for calibration. Otherwise the movement detection does not work properly. Due to noise on the raw signal the algebraic sign can change although the signal rises or falls weakly. After a calibration has been done the function returns a value which is not zero. It is recommended set the quiescent value in the application with the new reference value (`calib->refVal`) of the calibration structure.

Picture 4 shows the tube for 40 counter values around the target value of 100.



Picture 4: Tube for autocalib

If the loop value is outside the calibration tube for `countEnd` times the compensator offset current is changed in the way that the signal comes back between the tube borders, as shown in “Picture 5”.



Picture 5: working autocalib

This calibration is only enforced when no object above the sensor area is detected within `countEnd` times. If the loop value is still in tube for `countEnd` times the reference value (`calib->refVal`) for the calibration is stored. The function returns 0 when nothing is done with the offset from compensator. When the offset is modified the function will returns 1.

4.4 Initialize temperature compensation

This function will assist you to initialize the structure for the temperature compensation. The signature of this function is as follows:

```
void haliosTempCompInit(uint16_t loopValue, haliosTempcomp *tempcomp);
```

`loopValue` is the current loop value.

`*tempcomp` is a pointer to the structure witch stores the values to compensate the temperature.

This function has to call during the initialize and every time when something is modified at the parameters from the loops.

4.5 Temperature compensation

This function is to compensate temperature caused differences at the raw values from a HALIOS®-loop. The signature of this function is as follows:

```
uint16_t haliosTempComp(uint16_t loopValue, uint32_t countEnd, uint16_t maxDCnt, uint16_t maxCnt, haliosTempcomp *tempcomp);
```

loopValue is the raw value you will to compensate.

countEnd is the end value of call-timer to detect temperature caused differences.

maxDCnt is the maximum expected differences caused by temperature. All differences higher than maxDCnt during the call-timer not reach countEnd are interpreted as moving.

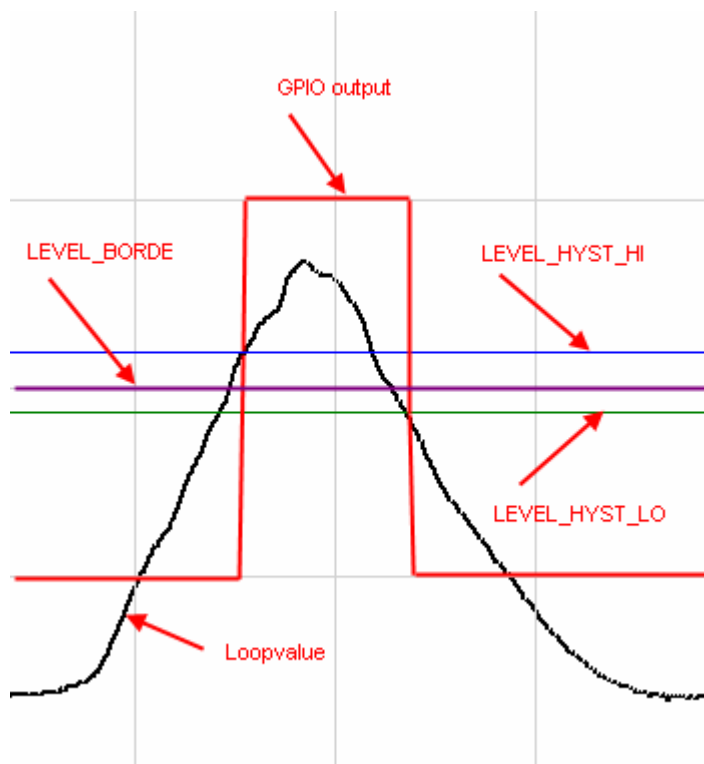
maxCnt is the maximum expected difference witch caused by temperature during movement.

*tempcomp is a pointer to the structure witch stores the values to compensate the temperature for each loop. To compensate more than one loop it is recommended to hold this structure In the main-routine of your application.

The function haliosTempComp() works only correctly when it is called periodically in the main loop of the E909.05A application software.

4.6 LevelSwitch module

The LevelSwitch module makes it possible to react with a pin change level when the loop value crosses a defined border value. It is additionally possible to define a hysteresis about the border value with a top and a bottom border value to avoid strobos on the output pin. In "Picture 6" an example is figured out, where for better understanding the reaction of the output pin is overlaid over the loop curve.



Picture 6: LevelSwitch module

In the HALIOS Tools are four independent LevelSwitch modules implemented. The configuration of each module is possible over the parameter values which are stored and accessible in the UserSpace of the Firmware. With the function

```
void deviceLevelSwitchInit(uint16_t startAddress);
```

these parameters are initialized and stored in the UserSpace beginning at 'startAddress'. This function has to be called once in the initialization routine of the application.

With the function

```
DeviceLevel deviceLevelSwitch(uint16_t startAddress);
```

the stored parameters are processed and the output pins are switched in dependency of the assigned loop values.

Table "Table 1: LevelSwitch parameters" lists the existing parameters.

Offset + startAddress	Description
0	Configuration word for module 0
1	LEVEL_BORDER for module 0
2	LEVEL_HYST_HI for module 0
3	LEVEL_HYST_LO for module 0
...	...
12	Configuration word for module 3
13	LEVEL_BORDER for module 3
14	LEVEL_HYST_HI for module 3
15	LEVEL_HYST_LO for module 3

Table 1: LevelSwitch parameters

Table "" explains the configuration word of the LevelSwitch modules.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved			LOOP_NUMBER			reserved			INVERT			USB		ACTIVE	

Table 2: LevelSwitch configuration word

ACTIVE

- 0: LevelSwitch module disabled.
- 1: LevelSwitch module enabled.

USB

- 0: GPIO pins 2..5 of the E909.05A are used by the LevelSwitch modules 0..3.
- 1: Output pins 0..3 of the MAX3420E are used by the LevelSwitch modules 0..3.

INVERT

- 0: Set assigned output pin high when loop value is over LEVEL_HYST_HI and low if under LEVEL_HYST_LO.
- 1: Set assigned output pin low when loop value is over LEVEL_HYST_HI and high if under LEVEL_HYST_LO.

LOOP_NUMBER

- 0..11: Assign the loop to the LevelSwitch module.

Appendix

A. Record of Revisions

Version	Date	Author	State	Comment
1.0	04.12.2007	MOST	Release	
2.0	2008-04-11	MOST	Release	Chapter 5 removed.
3.0	2008-08-08	FDE	Release	Filter, Calibration are edited and temperature compensation is added
3.01	2008-12-03	FDE	Release	Chapter Sample App removed: see e909.05_application_note
3.02	2009-12-10	FDE	Release	Chapter 4.2 LevelSwitch module added
3.03	2010-03-11	MKI	Release	Chapter 4.3 calibrating the loop values reworked
4.00	2010-05-04	MKI	Release	Documentation for firmware 4.0 reworked
4.01	2010-03-17	MKI	Release	Mark-up removed

