

Application Note for E909.05 and E909.6 for HALIOS tools 4.0

Mechaless Systems GmbH

Names of all products in this publication are used for identification purposes only and are trademarks and/or registered trademarks of their respective owners. Mechaless Systems GmbH makes no claim of ownership or corporate association with the products or companies that own them.

Copyright© 2008 by Mechaless Systems GmbH
Albert-Nestler-Strasse 10
D 76131 Karlsruhe
Germany

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.

Chapter 1

Application Note for E909.05 and E909.6 for HALIOS tools File Index

1.1 Application Note for E909.05 and E909.6 for HALIOS tools File List

Here is a list of all documented files with brief descriptions:

main.c (Application example to demonstrate the usage of the HALIOS tools library (lib_haliostools) for HALIOS IC E909.05 and E909.06)	3
main.h	18

Chapter 2

Application Note for E909.05 and E909.6 for HALIOS tools File Documentation

2.1 main.c File Reference

Application example to demonstrate the usage of the HALIOS tools library (lib_haliostools) for HALIOS IC E909.05 and E909.06.

```
#include "firmware.h"  
#include "main.h"  
#include "user_space.h"  
#include "haliostools.h"  
#include "usb.h"
```

Defines

- #define `USB_PIN` BIT1

Functions

- const uint16_t `gui_applicationVersion` `__attribute__((section(".application_version")))`
- void `isr_gpio_falling` (void)
- void `isr_wakeup` (void)
- int `main` (int argc, char *argv[])

Variables

- volatile uint16_t [gui_doUsb](#) = 1
- volatile uint16_t [gui_measurement](#) = 0
- const char [gArc_project_number](#) [] = "0908503"

2.1.1 Detailed Description

Application example to demonstrate the usage of the HALIOS tools library (lib_haliostools) for HALIOS IC E909.05 and E909.06.

Author:

Florian Degler, Mechaless Systems GmbH

Date:

Created: 2008-11-18

Author:

Roland Muenzer, Media System Consulting

Date:

Changed: 2008-12-03 Reworked documentation

Author:

Florian Degler, Mechaless Systems GmbH

Date:

Changed: 2010-05-28 Reworked for firmware V4.0

Author:

Markus Kilian, Mechaless Systems GmbH

Date:

Changed: 2010-05-31 Reworked for firmware V4.0

Id

[main.c](#),v 1.1 2010/07/13 10:11:37 mki Exp

Definition in file [main.c](#).

2.1.2 Define Documentation

2.1.2.1 #define USB_PIN BIT1

Define the hardware input pin which is connected to IC Max3420 for USB-request. For base-board its always GPIO_1

Definition at line 67 of file main.c.

2.1.3 Function Documentation

2.1.3.1 const uint16_t gui_applicationVersion __attribute__((section(".application_version")))

Set a project application version number. Set to a fix area at FLASH to make possible read out in output file and verify the flashed code.

2.1.3.2 void isr_gpio_falling (void)

Interrupt function Falling edge at Pin 1 is a USB-request from Master

```
/**  
    if (P0NEDGE_STAT & USB_PIN)  
    {  
        gui_doUsb = 1;  
        g_status0.wakeupEnd = 1;  
    }  
  
    P0NEDGE_CLR = 0x3F;  
    /**
```

Definition at line 73 of file main.c.

References gui_doUsb, and USB_PIN.

2.1.3.3 void isr_wakeup (void)

Interrupt function Wakeup occurred - Set wukeupEnd and do a measurement

Definition at line 92 of file main.c.

References gui_measurement.

2.1.3.4 int main (int argc, char * argv[])

main

Parameters:

← *argc* dummy parameter

← *argv* dummy parameter

```

**/
uint16_t ui_cnt;
loopConf_t t_loopConf;

/* variables for filter and calibration functions */
uint16_t ui_filtervalue, ui_quiescent;
uint16_t ui_autocatch = 0;
uint16_t ui_quis_min;
Calib_Result_t t_calib_result = Calib_Nothing_Done;
uint16_t ui_lastCalibTimestamp[LOOPMAXCOUNT];
uint16_t ui_oldCalTime = 0;

/**
 * Initializes the HALIOS SFRs and set up the basic functions of hardware.
 * @n It is recommend to call this function as first call.
 *
 * @post The system is configured:
 * - The trimmvalues are read from InfoBlock and set to
 * mclk and wkclk (only at (E909.05)
 * - Following interrupts are enabled:
 *   - HALIOS measurement ready
 *   - wakeup timer
 * - Following GPIO settings are used:
 *   - The RDY_PIN will set as output,
 *     if no readypin is needed set RDY_PIN as 0
 * - Wakup timer enabled and set to 10 ms, used for sample time
 * - One HALIOS loop enabled and set up (one LED against compensator).
 *
 * @param [in] BIT0 Set a GPIO as trigger pin for measurement, use only one bit.
 *           If not needed set to 0.
 */
haliosInitialize(BIT0);

/**
 * Set the projectnumber (eight characters) to g_sfr.project_number to make
 * readable about the constant reading mechanism @ref paramCheckSfr.

```

```
*
* @param[in] gArc_project_number Pointer to a string. The maximum numbers of eight ch
*/
paramSetProjectNr((uint8_t*)(gArc_project_number));

/** Setup the register of the watchdog timer0.
*
* Configure the watchdog in milliseconds (ms).
*
* @param[in] 500 Watchdogtime in ms.
* @n Must be smaller than 500 seconds (s)!
* @n Higher Values will ignore and set to 500 s
*/
deviceSetWatchdogTime (500UL);

/** set IO port function to GPIO for all pins */
POCFG = 0;

/**
* Define which communication device will be used and enable or disable the
* related interrupts.
* @n This function is optional. If this function is not called, communication
* devices set all to off.
*
* @param[in] DEVCOM_I2C set communication to I2C
* - For no communication device use (@ref DEVCOM_NO_COMM)
* - For I2C (@ref DEVCOM_I2C)
* - For SPI (@ref DEVCOM_SPI).
* - For SPI and I2C (@ref DEVCOM_I2C | @ref DEVCOM_SPI)
*/
deviceSetCommDevice(DEVCOM_I2C);

/**
* Call this function to show the last reset reason at a pin
* by a significant bit pattern.
* @n This function is optional. Use only if you don't want to
* do your own fail state.
* @n
* @n Count the blink sequence of the output pin:
* - 4 times blinking: watchdog reset
* - 5 times blinking: CPU register parity error
* - 6 times blinking: FLASH uncorrectable bit error
* - 7 times blinking: RAM perity error
* - 8 times blinking: Trap
* @n @n
* @param[in] outputPin Define the pin which should do the failState show
* @param[in] inputPin Define the pin which break the failState show.
* Set to 0 if now break is required
*/
failState(BIT2, BIT3);
```

```

/**
 * Compute the checksum over all words in "Parameter FLASH Area".
 * If the Checksum proofs "Valid Data", data is copied from the
 * "Parameter FLASH Area" into RAM.
 *
 * @return
 *     - -1: No valid data found.
 *     - else: Number of copied words.
 */
if (deviceRestore() == -1)
{
    /**
     * Set the sample time in milli seconds. The wakeup timer
     * of the Analog Control Module is used for the timing.
     * Depending on the communication device the micro-controller
     * switches to STANDBY or STOP mode.
     *
     * @note time in milli seconds, must be between 2 and 32, only even
     * values are accepted. (See also description of the Analog Control Module).
     */
    paramSetSampleTime(8);

    /**
     * Set the amount of active loops.
     *
     * @param[in] count Amount of active loops. @a count must be less or equal to
     * @ref LOOPMAXCOUNT.
     * @return An element of the @a HaliosCode enumeration:
     *     - HALIOS_OK: No error occurred
     *     - HALIOS_PARAM: Wrong parameter for count passed.
     */
    haliosSetLoopCount(8);

    /**
     * Configuration of the 1st loop.
     * This is an example how to use type loopConf_t for loop configuration.
     * The values are indices for the LED current of the ASIC.
     */
    t_loopConf.loopNr = 0;
    t_loopConf.ledConf = H_LED3B | H_LED5A | H_AON | H_ACCON;
    t_loopConf.phaseA.range = 10;
    t_loopConf.phaseA.offset = 22;
    t_loopConf.phaseB.range = 15;
    t_loopConf.phaseB.offset = 15;
    t_loopConf.iConfC = 0;
    t_loopConf.DC_offset = 0;
    t_loopConf.PreAmp = 0;
    t_loopConf.ClockConf = 0;

    /**
     * Store the configuration data into the virtuel loops at SFR by using

```

```

* a struct @ref LoopConf.
*
* @param[in] t_LoopConfig The LED and current configuration.
*
* @return An element of the @ref HaliosCode enumeration:
*   - HALIOS_OK:      No error occurred
*   - HALIOS_PARAM:  Wrong parameter in @a t_LoopConfig passed.
*/
haliosLoopInit(t_loopConf);

/**
* Store the configuration data into the virtuel loops at SFR by direct access.
*
* @note No validation check will done. It is recommand to use
* the function @ref haliosLoopInit.
*
* @param[in] loopNr      0 .. @ref LOOPMAXCOUNT
* @param[in] ledConf     LED and measurement configuration.
* @param[in] iClockConf  Measurement Configuration HALIOS Clock
* @param[in] iConfA      Current configuration for phase A.
* @param[in] iConfB      Current configuration for phase B.
* @param[in] iConfC      Current configuration for the compensator offset.
* @param[in] iPreAmp     Preamplifier Configuration
*/
haliosLoopInitialize(1, 20993, 0, 875, 495, 27, 0);
haliosLoopInitialize(2, 20996, 0, 810, 495, 25, 0);
haliosLoopInitialize(3, 21056, 0, 908, 495, 29, 0);
haliosLoopInitialize(4, 21077, 0, 3, 1023, 127, 0);
haliosLoopInitialize(5, 20993, 0, 287, 31, 127, 0);
haliosLoopInitialize(6, 21077, 0, 259, 1023, 127, 0);
haliosLoopInitialize(7, 20993, 0, 127, 31, 127, 0);

/**
* Set System Status to be used for @ref deviceWaitForTimer during wait
* until timer has elapsed or a interrupt wakes up the system.
* @n This function is optional. If not called system status is STANDBY.
* @n
* @param[in] SystemStatus  Selects system mode for deviceWaitForTimer
* - DEVSET_RUN:           Keep System in RUN Mode in deviceWaitForTimer
* - DEVSET_STANDBY:       Switch to STANDBY Mode in deviceWaitForTimer
* - DEVSET_STOP:          Switch to STOP Mode in deviceWaitForTimer
* - DEVSET_OFF:           Switch to OFF Mode in deviceWaitForTimer
*
* Keep in mind that spi-usb communication only works in RUN and in STANDBY mode.
*/
deviceSetSystemStatus(DEVSET_STANDBY);

/** Settings for filter and calibration in the user space */
paramSetValue(RAM_FILT_BORDER, HALIOS_FILT_8); /* filter depth */
paramSetValue(RAM_FILT_BREAK, 10); /* filter break */
paramSetValue(RAM_CAL_TUBE, 32); /* tube around target value */

```

```

paramSetValue(RAM_CAL_TIME, 300);           /* time for calibration */
paramSetValue(RAM_CAL_DCNT, 8);           /* value for movement detection with 1
paramSetValue(RAM_CAL_TARGET_VALUE, 100);   /* target value for calibration */

/** Set quiescent-value for the loops */
for (ui_cnt = 0; ui_cnt < haliosGetLoopCount(); ui_cnt++)
    paramSetValue(RAM_QUIESCENT_LOOP0 + (ui_cnt * BLOCK_SIZE), paramGetValue(RAM_C

/** Switch Calibration
 * Application options are:
 * CAL_START - calibration on start up
 * CAL_TIME - calibration for time
 * CAL_AUTO_CATCH - enable autocatch function
 */
paramSetValue(RAM_CAL_SETUP, ( CAL_START | CAL_TIME ));

/** Set the time (maximum time, some USB controller call more
than this value!) the PC requests for new values. */
paramSetValue(RAM_USB_CALL_TIME, 8);
}

/**
 * Check the contents of SFR and does any special functions.
 * If the content of a SFR register has changed the new values will be copied
 * into the corresponding firmware functions or corresponding hardware registers.
 * - Set size of SFR and user space to address @ref BUFFSIZE at SFR
 * - Set constant reading values to SFR controled by @ref READ_CONST_CMD
 * - Set systemStatus
 * - Set Communication device
 * - Set sampletime
 * - Use spezial functions (use careful)
 * - Set main clock (ANALOG_MCLK) (Only E909.05)
 * - Set wakeup clock (ANALOG_WKCLK) (Only E909.05)
 * - Set HALIOS frequency (Only E909.06)
 * - Set number of Loops to g_sfr.loopCount
 */
paramCheckSfr();

/** Set GPIO 2..5 as output pins */
PDIR &= ~(BIT2 | BIT3 | BIT4 | BIT5);

/**
 * @brief Function o init the HALIOS tools
 *
 * Initialize the structures for filtering and calibration.
 */
init_haliostools();

/**
 * @brief Warmup the HALIOS loops.

```

```
*
* Function from HALIOS Tools. Do some measurements for each loop to ensure
* that the measuerment counter has reached its actual value.
*
* @param[in] times How many times to start an empty measurement to warm up the
*             loops:
*             - HALIOS_WARMUP_FULL: 6 times for a full range of 1024 steps
*             - HALIOS_WARMUP_HALF: 3 times for a half range of 512 steps
*/
haliosWarmup(HALIOS_WARMUP_FULL);

/**
 * Force a calibration for each active loop.
 */
if (paramGetValue(RAM_CAL_SETUP) & CAL_START)
{
    for (ui_cnt = 0; ui_cnt < paramGetSFR(LOOPCOUNT); ++ui_cnt)
    {
        t_calib_result = haliosCompCalib(ui_cnt, haliosGetResult(ui_cnt), \
            paramGetValue(RAM_CAL_TARGET_VALUE),
            paramGetValue(RAM_CAL_TUBE), 0, 1023);
        paramSetValue( ((ui_cnt * BLOCK_SIZE) + RAM QUIESCENT_LOOP0), g_calib[ui_cnt].
    }
}

for(ui_cnt = 0; ui_cnt < LOOPMAXCOUNT; ui_cnt++)
{
    /* Initialize the last calibration-time-stamp variable for calibration */
    ui_lastCalibTimestamp[ui_cnt] = 0;
}

#if (USB != USB_OFF)
/**
 * Initialize the SPI module and the MAX3420E SPI-USB bridge.
 *
 * @post GPIO 2..5 configured for SPI
 */
usbInitialize(USB_PART_ON, USB_PIN, paramGetValue(RAM_USB_CALL_TIME));

/* set interrupt for falling signal on the interrupt request pin */
PONEDGE_EN |= USB_PIN;
/* set interrupt mask for falling signal on a GPIO */
IRQ_MASK_H |= VBH_GPIO_FALLING;
#endif

/** Set application bit and Version */
g_sfr.inst_libs |= BIT15;
deviceCheckVersion(BIT15, gui_applicationVersion);

/**
 *
```

```

* Do the measurement in an endless loop
*
*/
while (1)
{
    /**
    * Start and retrigger the watchdog timer. This is an inline function.
    *
    * @note At E909.06: After first call of watchdog it is not possible
    * to disable the watchdog or change the watchdog time.
    *
    */
    KICKDOG();

    /**
    * Check the contents of SFR and does any special functions.
    * If the content of a SFR register has changed the new values will be copied
    * into the corresponding firmware functions or corresponding hardware registers.
    * - Set size of SFR and user space to address @ref BUFFSIZE at SFR
    * - Set constant reading values to SFR controled by @ref READ_CONST_CMD
    * - Set systemStatus
    * - Set Communication device
    * - Set sampletime
    * - Use spezial functions (use careful)
    * - Set main clock (ANALOG_MCLK) (Only E909.05)
    * - Set wakeup clock (ANALOG_WKCLK) (Only E909.05)
    * - Set HALIOS frequency (Only E909.06)
    * - Set number of Loops to g_sfr.loopCount
    */
    paramCheckSfr();

    if (gui_measurment == 1)
    {
        gui_measurment = 0;

        /**
        * Do the HALIOS measurement of all configurated loops.
        * - Enable the analog part
        * - Start one Warmup to engage the analog part
        * - Start the configured measurements
        * - disable the analog part
        * - count up the @ref TIME_STAMP
        *
        * When haliosMeasure() is called with parameter HALIOS_RDYON,
        * the configured PIN in haliosInitialize() will be switched on
        * when entering the haliosMeasure() function,
        * and will be switched off when haliosMeasure() is left.
        *
        * @param[in] readyPin @ref HaliosCode
        * - @ref HALIOS_RDYON GPIO is used as ready pin.

```

```

*           - @ref HALIOS_RDYOFF GPIO is not used as ready pin.
*/
haliosMeasure(HALIOS_RDYON);

/**
* Filter the loops and check the calibration.
*/
ui_autocatch = 0;
for (ui_cnt = 0; ui_cnt < paramGetSFR(LOOPCOUNT); ++ui_cnt)
{
    /**
    * @brief Filter the loop with a low pass filter.
    *
    * @param[in] loopNr      Number of the loop (0 .. LOOPCOUNT).
    * @param[in] border3db  The 3dB border of the low pass filter.
    * @param[in] filterBreak If the derivation of the raw loop value is high
    *                        than filterBreak the filtered value is omitted
    *                        the raw loop value will be written to loopFilter
    *                        negative value for filterBreak disables the filter
    *                        break mechanism.
    *
    * @return      filter_value the software filtered value
    */
    ui_filtervalue = haliosFilterLoop(ui_cnt, \
        (HALIOS_FILT)paramGetValue(RAM_FILT_BORDER), \
        paramGetValue(RAM_FILT_BREAK));

    /** Set filtervalue to user space */
    paramSetValue( (RAM_FILT_LOOP0 + (ui_cnt * BLOCK_SIZE)) , ui_filtervalue);

    if (paramGetValue(RAM_CAL_SETUP) & CAL_TIME)
    {
        /**
        * When the autocatch function cause a calibration it sets the time for
        * So this forces a calibration immediately.
        * The variable ui_lastCalibTimestamp prevents that autocatch enforces
        * That's necessary because if no sensor is connected or the optical c
        * value is below quiescent value and probably around zero.
        */
        if (paramGetValue(RAM_CAL_SETUP) & CAL_AUTO_CATCH)
        {
            if ( paramGetValue(RAM QUIESCENT_LOOP0 + (ui_cnt * BLOCK_SIZE)) <=
            {
                ui_quis_min = paramGetValue(RAM QUIESCENT_LOOP0 + (ui_cnt * BL
            }
            else
            {
                ui_quis_min = paramGetValue(RAM QUIESCENT_LOOP0 + (ui_cnt * BL
            }
        }

        /**

```

```

* Force calibration when current value
* is below saved quiescent value
*/
if ((ui_autocatch == 0) && ((ui_filtervalue < ui_quis_min) || (ui_c
    && (ui_lastCalibTimestamp[ui_cnt] > 50))
    {
        ui_autocatch = 1;
        ui_oldCalTime = paramGetValue(RAM_CAL_TIME);
        paramSetValue(RAM_CAL_TIME, 0);
        ui_lastCalibTimestamp[ui_cnt] = 0;
    }
else if ((ui_lastCalibTimestamp[ui_cnt] <= 50) \
    && ((ui_filtervalue < ui_quis_min) || (ui_quis_min == 0)))
    {
        ui_lastCalibTimestamp[ui_cnt]++;
    }
else if ((ui_lastCalibTimestamp[ui_cnt] != 0) && (ui_filtervalue >
    {
        ui_lastCalibTimestamp[ui_cnt] = 0;
    }
}

/**
* @brief
* Calibrate the passed loop. This function counts the calls for each l
* The function checks if a movement can be detected. In case of a mov
* counter will reset. If no movement for countEnd has been detected th
* checks if the loop is in the tube around the target value (target va
* If the loop is outside the tube the compensator offset will be chang
* to reach the target value again. In case of a balanced loop the
* offset from both sender will be influenced.
* This is important when a static object has been detected or the opt
* of the sensor has changed.
*
* @param[in] nr          number of the loop (0 .. LOOPCOUNT)
* @param[in] loopValue   actual value of the signal
* @param[in] target      target value for the idle loop
* @param[in] tube        If the loop is within the tube borders (tar
*                        < loopValue < target+tube) then the actual
*                        value is fetched as the new reference value
*                        derivation is smaller than maxDCnt. If the
*                        value is outside the calibration tube the
*                        compensator offset current is calibrated.
* @param[in] countEnd    If the loop value is count times between th
*                        calibTube then a new reference value is det
*                        If count is zero the function immediatly st
*                        calibration.
* @param[in] maxDCnt     If the derivation is greater than maxDCnt t
*                        calibration is aborted and counter gets a r
*/
t_calib_result = haliosCompCalib(ui_cnt, ui_filtervalue, \

```

```

        paramGetValue(RAM_CAL_TARGET_VALUE),
        paramGetValue(RAM_CAL_TUBE), \
            ((uint32_t)(paramGetValue(RAM_CAL_TIME)) * 100), \
            paramGetValue(RAM_CAL_DCNT));

    /** Save the new quiescent value to user space */
    if (t_calib_result != Calib_Nothing_Done)
    {
        paramSetValue(RAM QUIESCENT_LOOP0 + (ui_cnt * BLOCK_SIZE), g_calib
    }

    /** Set calibTime to old value */
    if (ui_oldCalTime != 0)
    {
        paramSetValue(RAM_CAL_TIME, ui_oldCalTime);
        ui_oldCalTime = 0;
    }
} /* calibration time */

/** Compute amplitude for the loop */
ui_quiescent = paramGetValue(RAM QUIESCENT_LOOP0 + (ui_cnt * BLOCK_SIZE));
if (ui_filtervalue > ui_quiescent)
{
    paramSetValue( RAM_AMPLITUDE_LOOP0 + (ui_cnt * BLOCK_SIZE)
, (ui_filtervalue - ui_quiescent) );
}
else
{
    paramSetValue( RAM_AMPLITUDE_LOOP0 + (ui_cnt * BLOCK_SIZE) , 0);
}

}/* filter loops and check calibration */

/**
 * This function switches an output pin on or off in dependence of
 * the loop value of the assigned loop. There are four LevelSwitch
 * modules. Each of these modules could be configured separately:
 * - Assign a loop to the module
 * - Define a high and a low treshhold
 *
 * @note startAddress Start for parameter in User_space
 *
 * @note return Error code:
 * - 0: No error
 * - 1: Border is not set
 * - 2: Border is out of range
 */
    paramSetValue(RAM_LEVEL_SWITCH_RETURN, \
deviceLevelSwitch(RAM_LEVEL_SWITCH));

```

```

        /**
        * If USB is switched on the output pins from MAX3420 are used.
        * Otherwise the GPIO from HALIOS IC are used.
        */
#if (USB != USB_OFF)
        /* output pins from max3420 are used */
        wreg(rGPIO, paramGetValue(RAM_LEVEL_SWITCH_RESULT));
#else
        /* IO2..5 are used */
        P0OUT = (P0OUT & 0xffc3) | paramGetValue(RAM_LEVEL_SWITCH_RESULT) << 2;
#endif
    } /* if measuerment */

#if (USB != USB_OFF)

    /**
    * If Interrupt falling edge was caused by Pin 1 do a USB transfer
    */
    if (gui_doUsb == 1)
    {
        /**
        * If USB-request occured during a measurement
        * clear the wakeupEnd flag
        */
        g_status0.wakeupEnd = 0;

        gui_doUsb = 0;

        /** Do transmission */
        usbHacoHandleIrqs();
    }
#endif

    /**
    * Wait until the timer has elapsed.
    */
    deviceWaitForTimer();

    /**

```

Definition at line 105 of file main.c.

References CAL_AUTO_CATCH, CAL_START, CAL_TIME, gArc_project_number, gui_doUsb, gui_measurment, and USB_PIN.

2.1.4 Variable Documentation

2.1.4.1 volatile uint16_t [gui_doUsb](#) = 1

Global variable for communication between Interrupt and USB-Part in main

Definition at line 39 of file main.c.

2.1.4.2 volatile uint16_t [gui_measurment](#) = 0

Global variable for synchronize the measueremt with configured sample time

Definition at line 45 of file main.c.

2.2 main.h File Reference

Defines

- #define `APPLICATION_VERSION` 101UL
- #define `USB_OFF` 1
- #define `USB_HACO` 2
- #define `USB_MOUSE` 3
- #define `USB_KEYB` 4
- #define `USB` `USB_HACO`
- #define `LIN_OFF` 0
- #define `LIN_ON` 1
- #define `LIN` `LIN_OFF`
- #define `CAL_OFF` 0
- #define `CAL_START` BIT0
- #define `CAL_TIME` BIT1
- #define `CAL_AUTO_CATCH` BIT2

2.2.1 Detailed Description

Header file for the example application.

Author:

Miroslav Ostric, Mechaless Systems GmbH

Date:

Created: 2007-03-13

Author:

Roland Muenzer, Media System Consulting

Date:

Changed: 2008-11-26 added comments, added missing include "firmware.h"

Author:

Florian Degler, Mechaless Systems GmbH

Date:

Changed: 2010-05-28 Reworked for firmware V4.0

Author:

Markus Kilian, Mechaless Systems GmbH

Date:

Changed: 2010-05-31 Reworked for firmware V4.0 added comments, removed obsolete include "firmware.h"

Author:

Markus Kilian, Mechaless Systems GmbH

Date:

Changed: 2010-07-13 Due to compatibility for GCC firmware library 4.01 available. Application version set to 1.01.

Definition in file [main.h](#).

2.2.2 Define Documentation

2.2.2.1 `#define APPLICATION_VERSION 101UL`

Version number for the application.

Definition at line 27 of file main.h.

2.2.2.2 `#define USB_OFF 1`

Standalone application, no USB support.

Definition at line 30 of file main.h.

2.2.2.3 #define USB_HACO 2

USB support for the MAX3420E USB-SPI bridge, e.g. like on the E909.05A baseboard.

Definition at line 36 of file main.h.

2.2.2.4 #define USB USB_HACO

Software switch to choose between standalone mode and USB support.

Definition at line 41 of file main.h.

2.2.2.5 #define LIN_OFF 0

Definitions for LIN module

Keep in mind: LIN can only be used with E909.06

Definition at line 49 of file main.h.

2.2.2.6 #define CAL_OFF 0

Bit definitions for calibration

Definition at line 57 of file main.h.

Index

`__attribute__`
main.c, 5

APPLICATION_VERSION
main.h, 19

CAL_OFF
main.h, 20

gui_doUsb
main.c, 16

gui_measurment
main.c, 17

isr_gpio_falling
main.c, 5

isr_wakeup
main.c, 5

LIN_OFF
main.h, 20

main
main.c, 6

main.c, 3
__attribute__, 5
gui_doUsb, 16
gui_measurment, 17
isr_gpio_falling, 5
isr_wakeup, 5
main, 6
USB_PIN, 5

main.h, 18
APPLICATION_VERSION, 19
CAL_OFF, 20
LIN_OFF, 20
USB, 20

USB_HACO, 19
USB_OFF, 19

USB
main.h, 20

USB_HACO
main.h, 19

USB_OFF
main.h, 19

USB_PIN
main.c, 5