Application Note for E909.05 and E909.6 for USB 4.0

Mechaless Systems GmbH

LIMITED WARRANTY

THERE IS NO WARRANTY FOR THE CORRECTNESS OF THIS PUBLICATION OR THE RELATED CODE EXAMPLES OR RELATED PROGRAMS TO THE EXTENT PERMITED 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 NECCESSARY 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, INCIDENTICAL 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 USB File Index

1.1 Application Note for E909.05 and E909.6 for USB File List

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

main.c (Application example to demonstrate the usage of of the USB library (lib_usb) for HALIOS IC E909.05 end E909.06. Following libraries are used: lib_-firmware - firmware API functions for HALIOS IC lib_callback - callback functions for interrupt routines lib_usb - SPI-USB communication) 3
main.h 15

Chapter 2

Application Note for E909.05 and E909.6 for USB File Documentation

2.1 main.c File Reference

Application example to demonstrate the usage of of the USB library (lib_usb) for HALIOS IC E909.05 end E909.06. Following libraries are used: lib_firmware - firmware API functions for HALIOS IC lib_callback - callback functions for interrupt routines lib_usb - SPI-USB communication.

```
#include "firmware.h"
#include "main.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_measurment = 0
- const char gArc_project_number [ ] = "0908503"

2.1.1 Detailed Description

Application example to demonstrate the usage of of the USB library (lib_usb) for HALIOS IC E909.05 end E909.06. Following libraries are used: lib_firmware - firmware API functions for HALIOS IC lib_callback - callback functions for interrupt routines lib_usb - SPI-USB communication.

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 1.2 2010/06/18 15:30:23 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 68 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 74 of file main.c.

References gui_doUsb, and USB_PIN.

2.1.3.3 void isr_wakeup (void)

Interrupt function Wakeup occured - Set wukeupEnd and do a measurement

Definition at line 93 of file main.c.

References gui_measurment.

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

main

Parameters:

← ***argc*** dummy parameter

← ***argv*** dummy parameter

```
**/
    loopConf_t t_loopConf;

#if (USB == USB_KEYB)
    uint16_t ui_cnt = 0;
    uint16_t ui_stateUsbKeybOut = 0;
#elif (USB == USB_MOUSE)

#endif

    /**
    * Initializes the HALIOS SFRs and set up the basic functions of hardware.
    * @n It is recommand 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 */
  P0CFG = 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 occured
    *      - HALIOS_PARAM: Wrong parameter for count passed.
    */
    haliosSetLoopCount(4);

    /**
    * 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 = 15;
    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 occured
*      - 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);

/**
* 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);

/**
 * Write USB call time to user space.
 * Set the time (maximum time, some USB controller call more
 * than this value!) the PC requests for new values.
 *
 *  Write values to user space are readable using I2C Protocol or
 *  USB (and HACo).
 *
 *  The size of user space is 255 words
 */
```

```
        paramSetValue(0, 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();

#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(0));

    /* set interrupt for falling signal on the interrupt request pin */
    P0NEDGE_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);

        } /* if measuerment */

#if (USB == USB_KEYB)
        switch (ui_stateUsbKeybOut)
        {
            case 0:
                if (haliosGetResult(0) > 600)
```

```
    {
        ui_stateUsbKeybOut = 1;
    }
    break;

case 1:
    ui_cnt = setKey(KEY_LEFT_SHIFT, KEY_H);
    if (ui_cnt == 1)
        ui_stateUsbKeybOut = 2;
    break;

case 2:
    ui_cnt = setKey(0, KEY_A);
    if (ui_cnt == 1)
        ui_stateUsbKeybOut = 3;
    break;

case 3:
    ui_cnt = setKey(0, KEY_L);
    if (ui_cnt == 1)
        ui_stateUsbKeybOut = 4;
    break;

case 4:
    ui_cnt = setKey(0, KEY_L);
    if (ui_cnt == 1)
        ui_stateUsbKeybOut = 5;
    break;

case 5:
    ui_cnt = setKey(0, KEY_O);
    if (ui_cnt == 1)
        ui_stateUsbKeybOut = 6;
    break;

case 6:
    ui_cnt = setKey(0, KEY_SPACE);
    if (ui_cnt == 1)
        ui_stateUsbKeybOut = 7;
    break;

case 7:
    ui_cnt = setKey(KEY_LEFT_SHIFT, KEY_W);
    if (ui_cnt == 1)
        ui_stateUsbKeybOut = 8;
    break;

case 8:
    ui_cnt = setKey(0, KEY_O);
    if (ui_cnt == 1)
        ui_stateUsbKeybOut = 9;
```

```
                break;

            case 9:
                ui_cnt = setKey(0, KEY_R);
                if (ui_cnt == 1)
                    ui_stateUsbKeybOut = 10;
                break;

            case 10:
                ui_cnt = setKey(0, KEY_L);
                if (ui_cnt == 1)
                    ui_stateUsbKeybOut = 11;
                break;

            case 11:
                ui_cnt = setKey(0, KEY_D);
                if (ui_cnt == 1)
                    ui_stateUsbKeybOut = 12;
                break;

            case 12:
                ui_cnt = setKey(0, KEY_ENTER);
                if (ui_cnt == 1)
                    ui_stateUsbKeybOut = 13;
                break;

            case 13:
                if (haliosGetResult(0) < 390)
                {
                    ui_stateUsbKeybOut = 0;
                }
                break;

            default:
                ui_stateUsbKeybOut = 0;
                break;
        }



#elif (USB == USB_MOUSE)

#endif /* (USB == USB_KEYB) */

#if (USB != USB_OFF)

        /**
         * If Interrupt falling edge was caused by USB Pin, do an 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 */
#if (USB == USB_HACO)
            usbHacoHandleIrqs();
#elif (USB == USB_KEYB)
            usbKeybHandleIrqs();
#elif (USB == USB_MOUSE)
#warning see application note usb_mousedemo
#endif /* (USB == USB_HACO) */

        }

#endif /* (USB != USB_OFF) */

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

        /**
```

Definition at line 106 of file main.c.

References 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 40 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 46 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

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 35 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 40 of file main.h.

Index

__attribute__
    main.c, 5

APPLICATION_VERSION
    main.h, 16

gui_doUsb
    main.c, 14
gui_measurment
    main.c, 14

isr_gpio_falling
    main.c, 5
isr_wakeup
    main.c, 5

main
    main.c, 6
main.c, 3
    __attribute__, 5
    gui_doUsb, 14
    gui_measurment, 14
    isr_gpio_falling, 5
    isr_wakeup, 5
    main, 6
    USB_PIN, 5
main.h, 15
    APPLICATION_VERSION, 16
    USB, 16
    USB_HACO, 16
    USB_OFF, 16

USB
    main.h, 16
USB_HACO
    main.h, 16
USB_OFF
    main.h, 16
USB_PIN
    main.c, 5