

This document highlights the changes between v1.0 and v1.1 of the Class B Library - Interrupts.

Files removed/replaced:

- CLASSB\_INTERRUPTS/applications/CLASSB\_INTERRUPTS/error\_handler.h replaced by CLASSB/classb\_error\_handler.h
- CLASSB\_INTERRUPTS/applications/CLASSB\_INTERRUPTS/Tiny817xpro.h replaced by CLASSB/utils/oled1\_xpro\_attiny817.h
- CLASSB\_INTERRUPTS/applications/CLASSB\_INTERRUPTS/avr\_compiler.h replaced by CLASSB/utils/classb\_compiler.h
- CLASSB\_INTERRUPTS/applications/CLASSB\_INTERRUPTS/classb\_rtc\_common.h replaced by CLASSB/classb\_rtc.h

Files with diff/changelog:

- CLASSB\_INTERRUPTS/applications/CLASSB\_INTERRUPTS/classb\_interrupt\_monitor.h
- CLASSB\_INTERRUPTS/applications/CLASSB\_INTERRUPTS/main\_interrupts.c
- CLASSB\_INTERRUPTS/documentation/CLASSB\_INTERRUPTS.rst

Diff of classb\_interrupt\_monitor.h:

```
- CLASSB_INTERRUPTS/applications/CLASSB_INTERRUPTS/classb_interrupt_monitor.h -
index bec3769..9122e3d 100644
@@ -2,23 +2,23 @@
/**
 * \file
 *
- * \version 1.1
+ * \version 1.2
 *
 * \brief
 *
 * This file contains settings and definitions for the interrupt monitor.
 *
 * \par Application note:
- * AVR1610: Guide to IEC60730 Class B compliance with TinyAVR 1-series
+ * AN2632: Guide to IEC60730 Class B compliance with TinyAVR 1-series
 *
 * \par Documentation
 * For comprehensive code documentation, supported compilers, compiler
 * settings and supported devices see readme.html
 *
 * \author
- * Microchip Technology: http://www.microchip.com \n
- * Support at http://www.microchip.com/support/ \n
+ * Microchip Technology: http://www.microchip.com
+ * Support at http://www.microchip.com/support/
 *
- * Copyright (C) 2017 Microchip Technology. All rights reserved.
+ * Copyright (C) 2019 Microchip Technology. All rights reserved.
 *
 * \page License
 *
@@ -38,10 +38,10 @@
 * 4. This software may only be redistributed and used in connection with an
 * Microchip AVR product.
 *
- * THIS SOFTWARE IS PROVIDED BY Microchip "AS IS" AND ANY EXPRESS OR IMPLIED
+ * THIS SOFTWARE IS PROVIDED BY MICROCHIP "AS IS" AND ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE
- * EXPRESSLY AND SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL Microchip BE LIABLE FOR
+ * EXPRESSLY AND SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
@@ -54,47 +54,46 @@
#endif CLASSB_INTERRUPT_MONITOR_H_
#define CLASSB_INTERRUPT_MONITOR_H_
```

```

#include "avr_compiler.h"
#include "classb_rtc_common.h"
#include "error_handler.h"
-
#include "classb_compiler.h"
#include "classb_rtc.h"
#include "classb_error_handler.h"

//! \defgroup int_monitor Interrupt Monitor
-//!
+//!
//! \brief A test for the frequency of interrupts.
-//!
-//! Each time an interrupt is executed a counter is increased. The interrupt monitor
+//!
+//! Each time an interrupt is executed a counter is increased. The interrupt monitor
//! is executed periodically to check that the counters are within a configurable range.
-//! The Real Time Counter is used to run \ref classb_intmon_callback() periodically. The
-//! Interrupt monitor is enabled after calling \ref classb_rtc_setup(). Note that
-//! \ref CLASSB_INT_MON should be defined. See \ref rtc_driver for more details.
-//!
-//! In order to monitor an interrupt, the following steps should be followed:
-//! -# Register the interrupt in \ref classb_int_identifiers.
-//! -# The main application registers the interrupt by calling
-//! \ref classb_intmon_reg_int(). This gives the monitor information on the
+//! The Real Time Counter is used to run \ref classb_intmon_callback() periodically. The
+//! Interrupt monitor is enabled after calling \ref classb_rtc_setup(). Note that
+//! \ref CLASSB_INT_MON should be defined. See \ref rtc_driver for more details.
+//!
+//! In order to monitor an interrupt, the following steps should be followed:
+//! -# Register the interrupt in \ref classb_int_identifiers.
+//! -# The main application registers the interrupt by calling
+//! \ref classb_intmon_reg_int(). This gives the monitor information on the
//! interrupt, for example the expected frequency.
-//! -# The interrupts that have to be monitored should call \ref classb_intmon_increase()
-//! on each execution.
-//! -# The main application requests that the monitor starts checking the interrupt.
-//! This is done by changing the interrupt state to \c ENABLE with
-//! \ref classb_intmon_set_state(). The next time the interrupt monitor runs the
+//! -# The interrupts that have to be monitored should call \ref classb_intmon_increase()
+//! on each execution.
+//! -# The main application requests that the monitor starts checking the interrupt.
+//! This is done by changing the interrupt state to \c ENABLE with
+//! \ref classb_intmon_set_state(). The next time the interrupt monitor runs the
//! state of the interrupt will be set to \c ON.
-//! -# If at some point an interrupt should stop being monitored, the main application
-//! can change the state to \c DISABLE. The interrupt monitor will change
-//! the state to \c OFF the next time it is executed.
-//!
-//! The fact that the main application has to request to start monitoring an
-//! interrupt (and that it is the monitor sets the \c ON state) ensures that the
-//! interrupt counter is synchronized with the interrupt monitor, i.e. the interrupt
-//! counter starts being increased exactly after a period of the interrupt monitor.
-//!
-//! Note that the interrupt counter is only increased if the interrupt is \c ON.
-//! When an interrupt is \c OFF, the counter should be zero and otherwise the error
-//! flag will be set. Further, enabling an interrupt that is \c ON or disabling an
+//! -# If at some point an interrupt should stop being monitored, the main application
+//! can change the state to \c DISABLE. The interrupt monitor will change
+//! the state to \c OFF the next time it is executed.
+//!
+//! The fact that the main application has to request to start monitoring an
+//! interrupt (and that it is the monitor sets the \c ON state) ensures that the
+//! interrupt counter is synchronized with the interrupt monitor, i.e. the interrupt
+//! counter starts being increased exactly after a period of the interrupt monitor.
+//!
+//! Note that the interrupt counter is only increased if the interrupt is \c ON.
+//! When an interrupt is \c OFF, the counter should be zero and otherwise the error
+//! flag will be set. Further, enabling an interrupt that is \c ON or disabling an

```

```

-/*! Interrupt that is \c OFF will call the error handler if \ref CLASSB_STRICT is defined.
+/*! The error handler related to this test is CLASSB_ERROR_HANDLER_INTERRUPT().
+/*! The error handler related to this test is CLASSB_ERROR_HANDLER_INTERRUPT().
-/*!
+/*!
+/*! \addtogroup int_monitor
@@ -105,21 +104,23 @@
-/*!{
+/*!
+/*! \brief Enumeration of interrupt identifiers.
-/*!
+/*! This enumeration holds the identifiers for the interrupts that should be monitored.
-/*! These identifiers are used in the interrupt when calling functions related to the
+/*! interrupt monitor: \ref classb_intmon_reg_int(), \ref classb_intmon_increase() and
-/*! \ref classb_intmon_set_state(). Interrupt identifiers are included before
+/*!
+/*! This enumeration holds the identifiers for the interrupts that should be monitored.
+/*! These identifiers are used in the interrupt when calling functions related to the
+/*! interrupt monitor: \ref classb_intmon_reg_int(), \ref classb_intmon_increase() and
+/*! \ref classb_intmon_set_state(). Interrupt identifiers are included before
+/*! \ref N_INTERRUPTS, so that it will hold the total number of registered interrupts.
-enum classb_int_identifiers { MY_INTERRUPT, /*!< Identifier of the interrupt
+enum classb_int_identifiers { MY_INTERRUPT, /*!< Identifier of the interrupt
+                                                                    N_INTERRUPTS /*!< This will keep the number of
registered interrupts
+                                                                    };
+enum classb_int_identifiers
+{
+    MY_INTERRUPT, /*!< Identifier of the interrupt
+    N_INTERRUPTS /*!< This will keep the number of registered interrupts
+};
-/*! \brief Behavior for re-enabling or re-disabling monitoring of interrupts.
+/*! If this is defined, enabling an interrupt that is on \c ON state or disabling
+/*! \brief Behavior for re-enabling or re-disabling monitoring of interrupts.
+/*! If this is defined, enabling an interrupt that is on \c ON state or disabling
+/*! an interrupt that is on \c OFF state will call the error handler.
-#define CLASSB_STRICT
+#define CLASSB_STRICT
+/*!}
+@@ -130,11 +131,13 @@ enum classb_int_identifiers { MY_INTERRUPT, /*!< Identifier of the interrupt
+/*! \brief Enumeration of interrupt states.
+*/
-enum classb_int_states {ON, /*!< \internal Interrupt is being monitored (should only be set by the interrupt monitor).
+enum classb_int_states {ON, /*!< \internal Interrupt is being monitored (should only be set by the interrupt monitor).
+                                                                    OFF, /*!< \internal Interrupt is not being monitored (should only be
set by the interrupt monitor).
+                                                                    ENABLE, /*!< Interrupt should start being monitored (can be set by
the user application).
+                                                                    DISABLE /*!< Interrupt should stop being monitored (can be set by
the user application).
+                                                                    };
+enum classb_int_states
+{
+    ON,          /*!< \internal Interrupt is being monitored (should only be set by the interrupt monitor).
+    OFF,         /*!< \internal Interrupt is not being monitored (should only be set by the interrupt monitor).
+    ENABLE,      /*!< Interrupt should start being monitored (can be set by the user application).
+    DISABLE     /*!< Interrupt should stop being monitored (can be set by the user application).
+};
+/*!
+* \internal \brief Data structure for the interrupts that are monitored.
+@@ -142,32 +145,32 @@ enum classb_int_states {ON, /*!< \internal Interrupt is being monitored (should
+* The main application has to register the interrupt by calling \ref classb_intmon_reg_int().
+* It is that function that sets the values of the structure.
```

```

*/
-struct intmon_interrupt {
-    /*! \brief Expected number of interrupts in the monitor period.
-    */
+struct intmon_interrupt
+{
+    /*! \brief Expected number of interrupts in the monitor period.
+    */
+    /*! The monitor period is defined by \ref CLASSB_RTC_INT_PERIOD and \ref CLASSB_RTC_FREQ.
+    uint16_t n;
-    /*! \brief Interrupt counter.
-    */
-    /*! This holds the number of interrupt occurrences in the current monitor period.
+    /*! \brief Interrupt counter.
+    */
+    /*! This holds the number of interrupt occurrences in the current monitor period.
+    uint16_t c;
-    -
-    /*! \brief Limit for deviation in the counter.
-    */
-    /*! Limit for deviation in the counter.
-    uint16_t l;
-    -
+    +
+    /*! \brief Limit for deviation in the counter.
+    */
+    /*! Limit for deviation in the counter.
+    uint16_t l;
+    +
+    /*! \brief State of the interrupt
-    */
+    */
+    /*! State of the interrupt
-    enum classb_int_states s;
+    enum classb_int_states s;
+};

//@}
/*! \brief Array of data structures for the interrupts that should be monitored
struct intmon_interrupt monitored_interrupts[N_INTERRUPTS];

-
/** \brief Registers an interrupt.
*
* This function registers the information that the monitor needs in order to
@@ -182,14 +185,13 @@ struct intmon_interrupt monitored_interrupts[N_INTERRUPTS];
* \param tolerance The allowed deviation (%) with respect to interrupt_counter for the interrupt counter.
* \callergraph
*/
-static inline void classb_intmon_reg_int(enum classb_int_identifiers identifier, uint16_t reference, uint8_t tolerance)
+static inline void classb_intmon_reg_int(enum classb_int_identifiers identifier, uint16_t reference, uint8_t tolerance)
{
-
+
+    monitored_interrupts[identifier].n = reference;
+    monitored_interrupts[identifier].c = 0;
+    monitored_interrupts[identifier].l = (reference * tolerance) / 100;
+    monitored_interrupts[identifier].s = OFF;
-
}

/*! \brief Increases the interrupt counter of the specified interrupt.
@@ -201,12 +203,11 @@ static inline void classb_intmon_reg_int(enum classb_int_identifiers identifier,
*
* \callergraph
*/
-static inline void classb_intmon_increase( enum classb_int_identifiers identifier )
-{-
+static inline void classb_intmon_increase(enum classb_int_identifiers identifier)
+{

```

```

if (monitored_interrupts[identifier].s == ON)
    monitored_interrupts[identifier].c++;
}

/*! \brief Set a state for the specified interrupt.
 * @ -221,47 +222,45 @@ static inline void classb_intmon_increase( enum classb_int_identifiers identifie
 *      \callergraph
 */
-static inline void classb_intmon_set_state( enum classb_int_identifiers identifier, enum classb_int_states state )
+static inline void classb_intmon_set_state(enum classb_int_identifiers identifier, enum classb_int_states state)
{
-
+
    switch (state)
    {
-         case ENABLE:
+         case ENABLE:
#ifdef CLASSB_STRICT
            if (monitored_interrupts[identifier].s != OFF)
            {
                CLASSB_ERROR_HANDLER_INTERRUPT();
                break;
            }
-#endif
+
            if (monitored_interrupts[identifier].s != OFF)
            {
                CLASSB_ERROR_HANDLER_INTERRUPT();
                break;
            }
+
+        #endif
+
            break;

-         case DISABLE:
+         case DISABLE:
#ifdef CLASSB_STRICT
            if (monitored_interrupts[identifier].s != ON)
            {
                CLASSB_ERROR_HANDLER_INTERRUPT();
                break;
            }
-#endif
+
            break;

-         default:
+         default:
            if (monitored_interrupts[identifier].s != ON)
            {
                CLASSB_ERROR_HANDLER_INTERRUPT();
                break;
            }
+
+        #endif
+
            break;

-     default:
+     default:
            CLASSB_ERROR_HANDLER_INTERRUPT();
        }
-
+
    // Set the new state only if CLASSB_CONDITION1_INTERRUPT is true.
-   if(CLASSB_CONDITION1_INTERRUPT)
+   if (CLASSB_CONDITION1_INTERRUPT)
    {
-       monitored_interrupts[identifier].s = state;
+       monitored_interrupts[identifier].s = state;
    }
}

-
+

```

```

-static inline uint16_t abs_diff(uint16_t a, uint16_t b)
+static inline uint16_t abs_diff(uint16_t a, uint16_t b)
{
-     return (a > b)?(a - b):(b - a);
+     return (a > b) ? (a - b) : (b - a);
}

/*! \brief The interrupt monitor.
@@ -276,48 +275,48 @@ static inline uint16_t abs_diff(uint16_t a, uint16_t b)
*
* \callergraph
*/
-static inline void classb_intmon_callback()
+static inline void classb_intmon_callback()
{
-
-     for (uint8_t i = 0; i < N_INTERRUPTS; i++ )
+     for (uint8_t i = 0; i < N_INTERRUPTS; i++)
+     {
-
-
+         switch (monitored_interrupts[i].s)
+         {
-             case ON:
-                 // Check whether the counter is within the allowed range
-                 if ( (uint16_t) abs_diff(monitored_interrupts[i].c, monitored_interrupts[i].n) >
monitored_interrupts[i].l) {
-                     CLASSB_ERROR_HANDLER_INTERRUPT();
-                     break;
-                 }
-                 // Reset counter
-                 monitored_interrupts[i].c = 0;
-                 break;
-             case OFF:
-                 // The counter is only increased when the state is ON.
-                 if (monitored_interrupts[i].c)
-                     CLASSB_ERROR_HANDLER_INTERRUPT();
-                 break;
-             case ENABLE:
-
-                 // Change state
-                 monitored_interrupts[i].s = ON;
-                 break;
-             case DISABLE:
-                 // Change state and reset the counter
-                 monitored_interrupts[i].s = OFF;
-                 monitored_interrupts[i].c = 0;
+         case ON:
+             // Check whether the counter is within the allowed range
+             if ((uint16_t)abs_diff(monitored_interrupts[i].c, monitored_interrupts[i].n) > monitored_interrupts[i].l)
+             {
+                 CLASSB_ERROR_HANDLER_INTERRUPT();
+                 break;
+             }
+             default:
+                 CLASSB_ERROR_HANDLER_INTERRUPT();
+             }
+             // Reset counter
+             monitored_interrupts[i].c = 0;
+             break;
+         case OFF:
+             // The counter is only increased when the state is ON.
+             if (monitored_interrupts[i].c)
+                 CLASSB_ERROR_HANDLER_INTERRUPT();
+             break;
+         case ENABLE:
+             // Change state
+             monitored_interrupts[i].s = ON;
+             break;
+         case DISABLE:
+
+             // Change state and reset the counter
+             monitored_interrupts[i].s = OFF;
+             monitored_interrupts[i].c = 0;
+         }
+     }
}

```

```

+                // Change state and reset the counter
+                monitored_interrupts[i].s = OFF;
+                monitored_interrupts[i].c = 0;
+                break;
+            default:
+                CLASSB_ERROR_HANDLER_INTERRUPT();
+        }
-
+        // If CLASSB_CONDITION2_INTERRUPT is true, there is no need to check the other interrupts.
+        if (CLASSB_CONDITION2_INTERRUPT)
-            break;
+            break;
+    }
}

//@}

-
-#endif /* CLASSB_INTERRUPT_MONITOR_H_ */
\ No newline at end of file

```

## Diff of main\_interrupts.c:

```

----- CLASSB_INTERRUPTS/applications/CLASSB_INTERRUPTS/main_interrupts.c -----
index 32f5846..3b2b53d 100644
@@ -2,30 +2,30 @@
/**
 * \file
 *
- * \version 1.1
+ * \version 1.2
 *
 * \brief
- *
- * This in an demo application for the interrupt monitor.
- *
- * This application sets up a periodic interrupt: a timer/counter (TC)
- * overflow interrupt. Further, the application configures the
- * interrupt monitor in order to check that the frequency of the
- * interrupt is correct. Two button interrupts are configured. The
- * first one changes the frequency of the TC interrupt. The second
- * one deactivates the interrupt in the monitor.
+ *
+ * This in an demo application for the interrupt monitor.
+ *
+ * This application sets up a periodic interrupt: a timer/counter (TC)
+ * overflow interrupt. Further, the application configures the
+ * interrupt monitor in order to check that the frequency of the
+ * interrupt is correct. Two button interrupts are configured. The
+ * first one changes the frequency of the TC interrupt. The second
+ * one deactivates the interrupt in the monitor.
 *
 * \par Application note:
- * AVR1610: Guide to IEC60730 Class B compliance with TinyAVR 1-series
+ * AN2632: Guide to IEC60730 Class B compliance with TinyAVR 1-series
 *
 * \par Documentation
 * For comprehensive code documentation, supported compilers, compiler
 * settings and supported devices see readme.html
 *
 * \author
- * Microchip Technology: http://www.microchip.com \n
- * Support at http://www.microchip.com/support/ \n
+ * Microchip Technology: http://www.microchip.com
+ * Support at http://www.microchip.com/support/
 *
- * Copyright (C) 2017 Microchip Technology. All rights reserved.
+ * Copyright (C) 2019 Microchip Technology. All rights reserved.

```

```

*
* \page License
*
@@ -45,10 +45,10 @@
* 4. This software may only be redistributed and used in connection with an
* Microchip AVR product.
*
- * THIS SOFTWARE IS PROVIDED BY Microchip "AS IS" AND ANY EXPRESS OR IMPLIED
+ * THIS SOFTWARE IS PROVIDED BY MICROCHIP "AS IS" AND ANY EXPRESS OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
* MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE
- * EXPRESSLY AND SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL Microchip BE LIABLE FOR
+ * EXPRESSLY AND SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR
* ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
* SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
@@ -58,10 +58,10 @@
* DAMAGE.
*/

#include "avr_compiler.h"
#include "classb_compiler.h"
#include "classb_interrupt_monitor.h"
#include "Tiny817xpro.h"
#include "classb_rtc_common.h"
#include "oled1_xpro_attiny817.h"
#include "classb_rtc.h"

// \name Configuration parameters
// @
@@ -69,20 +69,17 @@
// \brief Frequency in Hz of the T/C interrupt.
//
// This is used to compute the value written to the PER register.
#define F_TC_INT 30
#define F_TC_INT 30

// \brief Tolerance for number of T/C interrupts (%).
-//
-#define TC_INT_TOL 15
+//
+#define TC_INT_TOL 15

// \brief TC prescaler
//
// The TC runs on the system clock scaled down by this parameter.
-// Possible values are 1, 2, 4, 8, 64, 256 or 1024.
#define TC_PRESCALER 1
-
-// \brief This multiplies the TC period in SW0 interrupt.
#define PER_CHANGE 4
+// Possible values are 1, 2, 4, 8, 64, 256 or 1024.
#define TC_PRESCALER 1

// @
@@ -90,24 +87,27 @@
// \name Internal parameters
// @
-// \brief Reference for number of TC interrupts
-//
-#define TC_INT_COUNT_REF (uint32_t) ((1e0L * 37))
+// \brief Reference for number of TC interrupts
+//
+#define TC_INT_COUNT_REF (uint32_t) ((1e0L * 37))

// \brief TC frequency
-//

```



```

+//!
//! The frequency of the TC is F_CPU divided by the prescaling factor.
-#define TC_FREQ (uint32_t) (F_CPU/TC_PRESCALER)
+#define TC_FREQ (uint32_t) (F_CPU / TC_PRESCALER)

//! \brief TC period
-//!
+//!
//! The 16-bit TC will generate an interrupt when the count reaches this value.
-#define TC_PER (uint16_t) (TC_FREQ / F_TC_INT)
+#define TC_PER (uint16_t) (TC_FREQ / F_TC_INT)

-#define CHANGED_PERIOD (uint16_t)(TC_FREQ / F_TC_INT)
+//! \brief TC fail period
+//!
+//! A period that will result in an interrupt count outside the accepted range
+#define TC_PER_FAIL (TC_PER * 5)

-//@}
+//@}

//! \name Global variables
//@{
@@ -118,131 +118,129 @@ NO_INIT volatile uint8_t classb_error;
//@}

/*! \brief Setup TC interrupt
- *!
- *! This sets up the TC so that it generates an overflow interrupt periodically.
- *! This interrupt will be checked with the interrupt monitor.
+ *!
+ *! This sets up the TC so that it generates an overflow interrupt periodically.
+ *! This interrupt will be checked with the interrupt monitor.
*/
static inline void example_tc_configure(void)
{
-    TCA0_SINGLE_PER = TC_PER;
-    TCA0_SINGLE_INTCTRL = TCA_SINGLE_OVF_bm;
+    TCA0.SINGLE.PER = TC_PER;
+    TCA0.SINGLE.INTCTRL = TCA_SINGLE_OVF_bm;
}

static inline void example_tc_enable(void)
{
-    TCA0_SINGLE_CTRLA = TCA_SINGLE_ENABLE_bm;
+    TCA0.SINGLE.CTRLA = TCA_SINGLE_ENABLE_bm;
}

int main(void)
{
-    // Initialize error variabel
+    // Initialize error variable to 0 at bootup in this example
    classb_error = 0;

+
    // Configure the RTC. The monitor is called back from the RTC interrupt.
    classb_rtc_configure();

    // Enable RTC
-    RTC_CTRLA = RTC_RTCEN_bm;
-
+    RTC.CTRLA = RTC_RTCEN_bm;
+
    // Configure Buton 1 used to induce error
-    configure_button1();
    // Configure Button 2 to disable Interrupt monitor
+    configure_button1();
    configure_button2();
    // Configure LED1 to indicate error
-    configure_LED1();
    // Configure LED2 to indicate disabeling of interrupt monitor of TCA

```

```

+ // Configure LED3 to indicate button press
+ configure_LED1();
+ configure_LED2();
+ configure_LED3();

// Config of TCA used for this example
example_tc_configure();

- // Register the interrupt in the monitor
- classb_intmon_reg_int(MY_INTERRUPT, TC_INT_COUNT_REF, TC_INT_TOL);
-
+ classb_intmon_reg_int(MY_INTERRUPT, TC_INT_COUNT_REF, TC_INT_TOL);
+ // Wait For RTC to be started so that RTC and TCA is in sync
- classb_wait_for_rtc_sync();
-
+ classb_wait_for_rtc_sync();
+ // Enable TCA
+ example_tc_enable();

+ // Global interrupts enabled in the system
+ sei();

-
+ // Enable the monitoring of the TC interrupt
+ classb_intmon_set_state(MY_INTERRUPT, ENABLE);

- while(!classb_error) {
-     // Do nothing
- };
-
- // If this is executed there has been an error. Turn off LEDs
+
+ // Main loop
+ while(!classb_error) {
+     // Press BUTTON1 to introduce the interrupt error
+ };
+
+ // If this the code reaches this step there has been an error. Turn off LEDs 1 and 3
+ LED1_OFF;
+ LED3_OFF;
+ }

-
- /*! \brief Interrupt for SW0 press: change TC period.
- *
- * This interrupt can be used to test the interrupt monitor. After pressing button 0 the
- * period of the TC is changed, which means that the frequency of the TC interrupt is
- * modified. This should be detected by the interrupt monitor, which should then set
- * the global error flag \ref classb_error.
+ /*! \brief Interrupt for BUTTON press: Change TC period or disable TC interrupt monitoring.
+ *
+ * This interrupt can be used to test the interrupt monitor. After pressing button 0 the
+ * period of the TC is changed, which means that the frequency of the TC interrupt is
+ * modified. This should be detected by the interrupt monitor, which should then set
+ * the global error flag \ref classb_error.
+ *
- * If the TC interrupt does not need to be monitored anymore the state of the interrupt
- * can be changed to DISABLE with button 2. Which in this case disables it until device
+ * If the TC interrupt does not need to be monitored anymore the state of the interrupt
+ * can be changed to DISABLE.
+ * \note This interrupt could be executed more than once because of button bouncing.
+ * If \ref CLASSB_STRICT is defined the monitor would then set \ref classb_error.
+ */
ISR(PORTA_PORT_vect)
- {
-     // Clear Interrupt flag

```

```

-     BUTTON1_PORT.INTFLAGS = BUTTON1_PIN;
-     // Change period to induce error
-     TCA0_SINGLE_PER = CHANGED_PERIOD;
-}
-
-/*! \brief Interrupt for SW1 press: disable TC interrupt monitoring.
- *
- * If the TC interrupt does not need to be monitored anymore the state of the interrupt
- * can be changed to DISABLE.
- * \note This interrupt could be executed more than once because of button bouncing.
- * If \ref CLASSB_STRICT is defined the monitor would then set \ref classb_error.
- */
-ISR(PORTB_PORT_vect)
{
    // Read Interrupt flag
    uint8_t pinflags = BUTTON1_PORT.INTFLAGS;
-    LED3_OFF;
-    if (pinflags == BUTTON1_PIN)
-    {
-        TCA0_SINGLE_PER = CHANGED_PERIOD;
-    }
-    else
-    {
+
+        // Debounce
+        _delay_ms(200);
+
+        if(pinflags == BUTTON1_PIN) {
+            TCA0.SINGLE.PER = TC_PER_FAIL;
+        } else { //pinflags == BUTTON2_PIN
+            classb_intmon_set_state(MY_INTERRUPT, DISABLE);
+            LED2_OFF;
+        }
+
+        BUTTON1_PORT.INTFLAGS = pinflags;
    }
}

-
-/*! \brief TC overflow interrupt
- *
- * The interrupt counter is incremented by calling \ref classb_intmon_increase().
- * After that an LED is toggled.
+ * The interrupt counter is incremented by calling \ref classb_intmon_increase().
+ */
+ISR(TCA0_OVF_vect)
+{
+    // Clear Interrupt flag
-    TCA0_SINGLE_INTFLAGS = TCA_SINGLE_OVF_bm;
-    // Register interrump event with monitor
+    TCA0.SINGLE.INTFLAGS = TCA_SINGLE_OVF_bm;
+    // Register interrupt event with monitor
+    classb_intmon_increase(MY_INTERRUPT);
-    // Toggle LED.
-    PORTA_OUTTGL = LED1_PIN;
+}

+/*! \brief RTC overflow interrupt
+ *
+ * The interrupt counter monitored by calling \ref classb_intmon_callback().
+ * After that LED3 is toggled.
+ */
+ISR(RTC_CNT_vect)
+{
+    // Clear the interrupt flag
-    RTC_INTFLAGS = RTC_OVF_bm | RTC_CMP_bm;
+    RTC.INTFLAGS = RTC_OVF_bm | RTC_CMP_bm;
+
+    // Run interrupt monitor
+    classb_intmon_callback();

```

```

-          PORTA_OUTTGL = LED1_PIN;
-}
\ No newline at end of file
+          // Toggle LED3
+          if(!classb_error) {
+              LED3_PORT.IN |= LED3_PIN;
+          }
+}
+}

```

## Diff of CLASSB\_INTERRUPTS.rst:

```

----- CLASSB_INTERRUPTS/documentation/CLASSB_INTERRUPTS.rst -----
index 8495ed5..d8a6ddc 100644
@@ -1,7 +1,7 @@
Introduction
=====

-This in an demo application for the interrupt monitor.
+This in a demo application for the Class B interrupt monitor.

The application is made to execute on a ATtiny817 Xplained Pro
with the OLED1 Xplained connected to the EXT1 header.
@@@ -9,23 +9,43 @@@ with the OLED1 Xplained connected to the EXT1 header.
This application sets up a periodic interrupt: a timer/counter (TC)
overflow interrupt. Further, the application configures the
interrupt monitor in order to check that the frequency of the
-interrupt is correct. Two button interrupts are configured. The
-first one changes the frequency of the TC interrupt. The second
-one deactivates the interrupt in the monitor.
+interrupt is correct. This is done periodically by means of an RTC interrupt.
+
+LEDs 1 and 2 are enabled during startup, while LED 3 will toggle each time the interrupt monitor is executed. If the error flag
should be
+set, LEDs 1 and 3 will be turned off.
+
+Two button interrupts are configured in this example:
+ - Pressing button 1 will change the timer period, which means that the frequency of the TC interrupt is modified.
+   This will be detected by the interrupt monitor, which should then set the global error flag classb_error.
+ - Pressing button 2 will change the interrupt monitor state to DISABLE and future changes in frequency will not cause an error.
This will turn off LED2, but the other LEDs will still operate.
+ If CLASSB_STRICT is defined and the monitor is already disabled, a second attempt at disabling the monitor will set the error
flag.
+
+A reset or power cycling is needed to clear the classb_error flag.

Related documents / Application notes
-----

-This application is described in the following application note: To be published
+This application is described in the following application note:
+
+`Guide to IEC 60730 Class B Compliance with tinyAVR 1-series
<http://www.microchip.com/wwwappnotes/appnotes.aspx?appnote=en604502>` _

Supported evaluation kit
-----

- ATTiny817-XPRO

-Running the demo
-----
+Running the demo using GCC
+-----

1. Press Download Pack and save the .atzip file
2. Import .atzip file into Atmel Studio 7, File->Import->Atmel Start Project.
3. Build and flash into supported evaluation board

```

+

+Running the demo using IAR

+-----

+

+1. Check "IAR Embedded Workbench", press Download Pack and save the .atzip file

+2. Follow the steps on how to download and import a Start project into IAR found in "How to open in IDEs"

+ found under export project.

+3. Change linker file using iar\_cfgtiny817\_classB.xcl located in utils folder.

+4. Build and flash into supported evaluation board.

\ No newline at end of file