

This document highlights the changes between v1.0 and v1.1 of the Class B Library – WDT.

Files removed/replaced:

- CLASSB\_WDT/applications/CLASSB\_WDT/error\_handler.h replaced by CLASSB/classb\_error\_handler.h
- CLASSB\_WDT/applications/CLASSB\_WDT/Tiny817xpro.h replaced by CLASSB/utis/oled1\_xpro\_attiny817.h
- CLASSB\_WDT/applications/CLASSB\_WDT/avr\_compiler.h replaced by CLASSB/utis/classb\_compiler.h

Files with diff/changelog:

- CLASSB\_WDT/applications/CLASSB\_WDT/classb\_wdt\_test.c
- CLASSB\_WDT/applications/CLASSB\_WDT/classb\_wdt\_test.h
- CLASSB\_WDT/applications/CLASSB\_WDT/main\_wdt.c
- CLASSB\_WDT/documentation/CLASSB\_WDT.rst

## Diff of classb\_wdt\_test.c:

```
----- CLASSB_WDT/applications/CLASSB_WDT/classb_wdt_test.c -----
index a05a461..9abc4c9 100644
@@ -5,20 +5,20 @@
 * \version 1.1
 *
 * \brief
- * Watchdog timer self-diagnostic routine.
+ * Watchdog timer self-diagnostic routine.
 *
- * \par Application note:
- * AVR1610: Guide to IEC60730 Class B compliance with TinyAVR 1-series
+ * \par Application note:
+ * 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
@@ -51,8 +51,11 @@
 * DAMAGE.
 */
+
+ #include "classb_wdt_test.h"
- #include "Tiny817xpro.h"
+ #include "oled1_xpro_attiny817.h"
+ #include "ccp.h"

//! \addtogroup classb_wdt
//@{
```

```

@@ -69,7 +72,7 @@ NO_INIT volatile classb_preinit_teststate_t classb_wdt_teststate;

//! \brief Number of TC periods in a WDT period.
//! This variable is not initialized and, therefore, can be used across resets.
-NO_INIT volatile uint16_t classb_tc_count;
+NO_INIT volatile uint8_t classb_tc_count;
//@}

//! \endcond
@@ -83,6 +86,7 @@ NO_INIT volatile uint16_t classb_tc_count;
*      -# WDT issues a system reset after timeout.
*      -# WDT timing and that it can be reset.
*      -# WDT issues a system reset if it is untimely reset (WDT window mode).
+ * -# WDT issues a system reset if no WDR is executed (WDT window mode)
* If any of these tests should fail, \ref CLASSB_ERROR_HANDLER_WDT() would be called.
* By default the device will simply hang.
*
@@ -91,6 +95,7 @@ NO_INIT volatile uint16_t classb_tc_count;
* -# Check that WDT can issue a system reset. Set test state 1 and system reset.
* -# Check that WDT can be reset. Set test state 2 and system reset.
* -# Check that window mode works correctly. Set test state 3 and system reset.
+ * -# Check that WDT issues a system reset if no WDR is executed in windowmode. Set test state 4 and system reset
* -# Set up WDT in windows mode with periods CLASSB_WDT_WPER and CLASSB_WDT_PER.
*      Set test state "ok" and continue to main().
*
@@ -111,30 +116,49 @@ NO_INIT volatile uint16_t classb_tc_count;
#endif
// TCA overflow time in seconds
#define TCA_OVERFLOW_TIME 0.001
-// Period value written to TCA to achieve overflow time
+// Period value written to TCA to achieve overflow time
#define PER_VALUE (uint16_t)(TCA_OVERFLOW_TIME * F_CPU)
-// Delay to allow WDT to sync two consecutive WDR instructions
-#define WDT_SYNC_DELAY (uint16_t)(F_CPU / 1100)

-// Top value of TCA overflows before WDT is assumed not to be able to reset device ~25ms
-#define TC_MAX_COUNT_BEFORE_ERROR 25
-// Wait time for reset when WDT is in window mode ~10ms
-#define TC_COUNT_BEFORE_SAFE_RESET 10
+// Maximum value of TCA overflows before WDT is assumed not to be able to reset device ~11ms
+// Taking the frequency tolerance into account.
+// WDT_PERIOD_8CLK_gc + ~35% frequency tolerance of clock
+// TC_MAX_COUNT_BEFORE_ERROR = 8ms + 2.8 = 10.8ms ~11ms
+#define TC_MAX_COUNT_BEFORE_ERROR 11
+
+// Minimum value of TCA overflows before WDT is allowed to reset device ~5ms
+// Taking the frequency tolerance into account.
+// WDT_PERIOD_8CLK_gc - ~35% frequency tolerance of clock
+// = 8ms - 35% tolerance = 8ms - 2.8 = 5.2ms ~ 5ms
+#define TC_MIN_COUNT_BEFORE_ERROR 5
+
+// The WDR instruction will need 2 to 3 cycles of the WDT clock in order to be synchronized.
+#define WDR_SYNC_DELAY 3

//! \brief Configure TCA
//! Set period so that TCA overflows approximately every 1ms
static inline void configure_TCA_for_time_taking()
{
    // Set period value
-    TCA0_SINGLE_PER = PER_VALUE;
+    TCA0.SINGLE.PER = PER_VALUE;
    // Enable TCA and start counting
-    TCA0_SINGLE_CTRLA = TCA_SINGLE_ENABLE_bm;
+    TCA0.SINGLE.CTRLA = TCA_SINGLE_ENABLE_bm;
}

//! \brief Macro to reset TCA to zero and start counting from zero
#define restart_TCA()
-    TCA0_SINGLE_CTRLSET = TCA_SINGLE_CMD_RESTART_gc;
-    TCA0_SINGLE_INTFLAGS = TCA_SINGLE_OVF_bm

```

```

+      TCA0.SINGLE.CTRLSET = TCA_SINGLE_CMD_RESTART_gc;
+      TCA0.SINGLE.INTFLAGS = TCA_SINGLE_OVF_bm;
+
+void watchdog_delay(uint8_t i)
+{
+    restart_TCA();
+    while (i) {
+        while (!(TCA0.SINGLE.INTFLAGS & TCA_SINGLE_OVF_bm))
+        ;
+        TCA0.SINGLE.INTFLAGS = TCA_SINGLE_OVF_bm;
+        i--;
+    }
+}

//! \brief WDT test
//!
@@ -152,6 +176,8 @@ static inline void configure_TCA_for_time_taking()
//! The next step is to test that window mode works by attempting to reset the WDT counter in the closed
//! part of the WDT counter window. The correct response of the device should be to reset the device.
//!
+//! The next step will test that WDT will issue a system reset in windowmode if no WDR instruction is executed.
+//!
//! If the device and WDT was reset correctly in each step of the test the WDT is configured with the
//! application defined WDT period and window. The application WDT window and period settings can be
//! found in classb_wdt_test.h: CLASSB_WDT_WPER, and CLASSB_WDT_PER these should be modified to suite
@@ -161,7 +187,7 @@ static inline void configure_TCA_for_time_taking()
classb_wdt_test(void)
{
    // This variable is used to count TC periods
-    register uint16_t counter;
+    register uint8_t counter;

    configure_TCA_for_time_taking();

@@ -174,30 +200,36 @@ classb_wdt_test(void)
    RSTCTRL_RSTFR = (RSTCTRL_UPDIRF_bm | RSTCTRL_EXTRF_bm | RSTCTRL_PORF_bm);
    // Set the next state of the test
    classb_wdt_teststate = TEST_WDT_1;

-
    // Configure the TC, which is used as an independent time source.
    // In this section we are going to measure the number of TC periods
    // before the WDT timeout issues a system reset.

    // WDT Configuration:
-    _PROTECTED_WRITE(WDT_CTRLA, WDT_PERIOD_8CLK_gc);
+    ccp_write_io((void*)&(WDT_CTRLA), WDT_PERIOD_8CLK_gc);

+    // Wait for WDT to sync registers
+    while (WDT_STATUS & WDT_SYNCBUSY_bm)
+    ;

    // Count number of TC periods until a WDT timeout.
    // There is a configurable maximum number of TC periods before
    // an error in the WDT is assumed.

+
    classb_tc_count = 0;
    counter = TC_MAX_COUNT_BEFORE_ERROR;
    restart_TCA();
    while (counter--) {
        classb_tc_count++;
-        while (!(TCA0_SINGLE_INTFLAGS & TCA_SINGLE_OVF_bm))
+        while (!(TCA0.SINGLE.INTFLAGS & TCA_SINGLE_OVF_bm))
        ;
-        TCA0_SINGLE_INTFLAGS = TCA_SINGLE_OVF_bm;
+        TCA0.SINGLE.INTFLAGS = TCA_SINGLE_OVF_bm;
    }

+
    // This should only be executed if there is an error in the WDT,
    // i.e. if the WDT did not timeout before the maximum number of
    // TC periods was exceeded.

```

```

classb_wdt_teststate = FAULT_WDT;
}
// If a watchdog reset has occurred, this is not the first iteration.
else if (RSTCTRL_RSTFR & RSTCTRL_WDRF_bm) {
@@ -207,130 +239,156 @@ classb_wdt_test(void)
// Handle the different test stages.
switch (classb_wdt_teststate) {
case TEST_WDT_1:
{
// test WDT timing and that WDT can be reset
// Assume watchdog fault for now.
classb_wdt_teststate = FAULT_WDT;

// Make sure that the estimated WDT period is also higher than expected
if (classb_tc_count >= TC_COUNT_BEFORE_SAFE_RESET) {
// Make sure that the estimated WDT period is equal or lower than measured WDT period
if (classb_tc_count >= TC_MIN_COUNT_BEFORE_ERROR){
// WDT Configuration
// Enable WDT with configurable period.
_PROTECTED_WRITE(WDT_CTRLA, WDT_PERIOD_8CLK_gc);
// Wait approximately 0.75 * T_WDT, where T_WDT is the estimated
ccp_write_io((void*)&(WDT_CTRLA), WDT_PERIOD_8CLK_gc);
// Wait for WDT to sync registers
while (WDT_STATUS & WDT_SYNCBUSY_bm)
;
// Wait approximately 0.75 * T_WDT - WDR_SYNC_DELAY, where T_WDT is the
estimated

// WDT period, before a WDT reset. This checks that the WDT does
// not expires earlier than expected.
counter = classb_tc_count;
counter += (classb_tc_count >> 1);
counter >>= 1;
restart_TCA();
while (counter--) {
while (!(TCA0_SINGLE_INTFLAGS & TCA_SINGLE_OVF_bm))
;
TCA0_SINGLE_INTFLAGS = TCA_SINGLE_OVF_bm;
}
// WDR_SYNC_DELAY needs to be subtracted to avoid WDT to reset during
synchronization after WDT reset.

counter = (((classb_tc_count*3)/4) - WDR_SYNC_DELAY);
watchdog_delay(counter);
watchdog_reset();
// The WDR instruction will need 2 to 3 cycles of the WDT clock in order to be
synchronized.

watchdog_delay(WDR_SYNC_DELAY);

// Wait approximately 0.75 * T_WDT
counter = classb_tc_count;
counter += (classb_tc_count >> 1);
counter >>= 1;
while (counter--) {
while (!(TCA0_SINGLE_INTFLAGS & TCA_SINGLE_OVF_bm))
;
TCA0_SINGLE_INTFLAGS = TCA_SINGLE_OVF_bm;
}
counter = ((classb_tc_count*3)/4);
watchdog_delay(counter);

// Exiting the above while loop should only occur if WDT reset worked, otherwise there
would

// have been a system reset before this point (this would be
// executed approximately 1.5*T_WDT after the WDT was initially set).
// have been a system reset before this point
// (this would be executed approximately 1.5*T_WDT after the WDT was initially set).
// Note that in that case the state would still be |FAULT_WDT|.
// Set next state and wait for the WDT to issue a system reset.
// That should approximately happen in 0.25*T_WDT

```

```

+ // Set next test state
+ classb_wdt_teststate = TEST_WDT_2;
+
+ // Wait approx 0.5*T_WDT
+ counter = (classb_tc_count >> 1);
- while (counter--) {
-     while (!(TCA0_SINGLE_INTFLAGS & TCA_SINGLE_OVF_bm))
-     {
-         ;
-         TCA0_SINGLE_INTFLAGS = TCA_SINGLE_OVF_bm;
-     }
+ watchdog_delay(counter);
+
+ }
+ // Set error flag if WDT has not issued a reset.
+ classb_wdt_teststate = FAULT_WDT;
+ break;
-
- // test WDT with window mode.
+ }
+ // test that WDT works correctly in window mode.
+ case TEST_WDT_2:
-
+ {
+     // Assume watchdog fault for now.
+     classb_wdt_teststate = FAULT_WDT;
+
+     // Set Window and period for WDT
+     _PROTECTED_WRITE(WDT_CTRLA, WDT_PERIOD_8CLK_gc | WDT_WINDOW_8CLK_gc);
+     ccp_write_io((void*)&(WDT_CTRLA), (uint8_t) WDT_PERIOD_8CLK_gc | (uint8_t) WDT_WINDOW_8CLK_gc);
+     // Wait for WDT to sync registers
+     while (WDT_STATUS & WDT_SYNCBUSY_bm)
+     {
+         ;
+     }
+     // Set next test state and do a reset of the WDT immediately.
+     // This should issue a system reset with any window mode settings.
+     classb_wdt_teststate = TEST_WDT_3;
+     // Issue WDR instruction to activate the first window
+     watchdog_reset();
+     // Wait for WDR instruction to be synchronized to the WDT clock domain
+
+     uint8_t delay_misec = 10;
+     while (delay_misec--) {
+         while (!(TCA0_SINGLE_INTFLAGS & TCA_SINGLE_OVF_bm))
+         {
+             ;
+             TCA0_SINGLE_INTFLAGS = TCA_SINGLE_OVF_bm;
+         }
+     }
+     // Test WDT open window first by adding a delay to ensure the WDT is in open window
+     // Add a delay of approximately 1.25 * T_WDT to reach the open window.
+     counter = classb_tc_count + classb_tc_count/4;
+     watchdog_delay(counter);
+
+     // Issue WDR instruction
+     watchdog_reset();
+     // Wait for WDR instruction to be synchronized to the WDT clock domain
+
+     delay_misec = 4;
+     while (delay_misec--) {
+         while (!(TCA0_SINGLE_INTFLAGS & TCA_SINGLE_OVF_bm))
+         {
+             ;
+             TCA0_SINGLE_INTFLAGS = TCA_SINGLE_OVF_bm;
+         }
+     }
+     // Issue WDR instruction
+     // The WDR instruction will need 2 to 3 cycles of the WDT clock in order to be synchronized.
+     watchdog_delay(WDR_SYNC_DELAY);
+
+     // WDT is now reset and back in the closed window.
+     // Add a delay of approximately 0.5 * T_WDT and still remain inside the closed window.
+     counter += classb_tc_count/2;
+     watchdog_delay(counter);
+
+     // Set next test state and do a reset of the WDT immediately.

```

```

+ // This should issue a system reset since WDT will be in closed window.
+ classb_wdt_teststate = TEST_WDT_3;
+ watchdog_reset();
+ // The WDR instruction will need 2 to 3 cycles of the WDT clock in order to be synchronized.
+ // Wait for an additional 0.25 * T_WDT just to avoid timing issues with reset vs execution of
instructions
+
+ watchdog_delay(WDR_SYNC_DELAY + classb_tc_count/4);
+
+ // If there was a problem with the WDT in window mode, this would be executed.
+ // Set error flag if WDT has not issued a reset.
+ classb_wdt_teststate = FAULT_WDT;
+ break;
+
+ }
+
+ // continue to test WDT with window mode.
+ case TEST_WDT_3:
+ {
+
+ // Assume watchdog fault for now.
+ classb_wdt_teststate = FAULT_WDT;
+
+ // Set Window and period for WDT
+ ccp_write_io((void*)&(WDT.CTRLA), (uint8_t) WDT_PERIOD_8CLK_gc | (uint8_t) WDT_WINDOW_8CLK_gc);
+ // Wait for WDT to sync registers
+ while (WDT_STATUS & WDT_SYNCBUSY_bm)
+ ;
+ // Test that WDT issues a system reset after timeout if no WDR is executed.
+ // Wait 1.75 * T_WDT
+ counter = classb_tc_count + ((classb_tc_count*3)/4);
+ watchdog_delay(counter);
+
+ classb_wdt_teststate = TEST_WDT_4;
+
+ // Wait 0.5 * T_WDT
+ counter = classb_tc_count/2;
+ watchdog_delay(counter);
+
+ // If there was a problem with the WDT in window mode, this would be executed.
+ // Wait for 0.25 * T_WDT just to avoid timing issues with reset vs execution of instructions
+ // then set fault state if WDT module has not reset the device as expected.
- counter = classb_tc_count >> 2;
- restart_TCA();
- while (counter--) {
-     while (!(TCA0_SINGLE_INTFLAGS & TCA_SINGLE_OVF_bm))
-     ;
-     TCA0_SINGLE_INTFLAGS = TCA_SINGLE_OVF_bm;
- }
+ counter = classb_tc_count/4;
+ watchdog_delay(counter);
+
+ // Set error flag if WDT has not issued a reset.
+ classb_wdt_teststate = FAULT_WDT;
+ break;
+
+ }
+
+ // After the test the WDT should be left enabled for the main application.
- case TEST_WDT_3:
+ case TEST_WDT_4:
+ {
+
+ // WDT configuration for the main application: WDT in normal mode
+ _PROTECTED_WRITE(WDT_CTRLA, CLASSB_WDT_PER | CLASSB_WDT_WPER);
+ ccp_write_io((void*)&(WDT.CTRLA), (uint8_t) CLASSB_WDT_PER | (uint8_t) CLASSB_WDT_WPER);
+ // Wait for WDT to sync registers
+ while (WDT_STATUS & WDT_SYNCBUSY_bm)
+ ;
+ classb_error = 0;
+ // The test of the WDT is finished and there was no error
+ classb_wdt_teststate = TEST_WDT_OK;
+ break;
-
+ }

```

```

// Handle WDT reset under normal operation. If the program reaches
// this point is because the main application did not update the
// WDT correctly and this led to system reset. Note that the WDT
// needs to be setup again.
case TEST_WDT_OK:
+   {
+       // Configurable actions
+       CLASSB_ACTIONS_WDT_RUNTIME_FAILURE();
+       break;
-   }
+   // Otherwise assume error
+   default:
+   {
+       classb_wdt_teststate = FAULT_WDT;
+       break;
+   }
+ }

// Handle other reset reasons, i.e. software or brown-out.
@@ -350,4 +408,4 @@ classb_wdt_test(void)
#if defined(__ICCAVR__)
    return 1;
#endif
-}
+}
\ No newline at end of file

```

## Diff of classb\_wdt\_test.h:

```

----- CLASSB_WDT/applications/CLASSB_WDT/classb_wdt_test.h -----
index ebc5d68..ffb067e 100644
@@ -1,21 +1,27 @@
-/* This file has been prepared for Doxygen automatic documentation generation.*/
+/* This file has been prepared for Doxygen automatic documentation generation.*/
/**
 * \file
 *
 * - * \brief Settings for the watchdog timer test.
 * - *
+ * \version 1.1
+ *
+ * \brief
+ *   Header file for compiler compatibility
+ *
+ *   This file contains some general definitions and macros to ensure code
+ *   compatibility with both IAR and GCC.
+ *
 * \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

```

```

*
@@ -35,10 +41,10 @@
* 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
@@ -48,44 +54,45 @@
* DAMAGE.
*/

+
+
#ifndef CLASSB_WDT_TEST_H
#define CLASSB_WDT_TEST_H

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

//          CLASSB_WDT_PER contains the desired WDT timeout period for the application
//          the WDT will be set to this value after the test is complete.

/*! \defgroup classb_wdt Watchdog Timer Test
-/*!
-/*! \brief This test checks that the watchdog timer (WDT) is working.
-/*! The WDT is a system function for monitoring correct program operation that allows
-/*! recovering from error situations such as runaway or deadlocked code. The
-/*! self-diagnostic test \ref classb_wdt_test() is executed before the main application
-/*! and it makes sure that:
+/*!
+/*! \brief This test checks that the watchdog timer (WDT) is working.
+/*! The WDT is a system function for monitoring correct program operation that allows
+/*! recovering from error situations such as runaway or deadlocked code. The
+/*! self-diagnostic test \ref classb_wdt_test() is executed before the main application
+/*! and it makes sure that:
-/*! - a system reset is issued after WDT timeout
-/*! - the WDT can be reset
-/*! - the device is reset upon untimely WDT reset in window mode
-/*!
-/*! The test relies on a Timer Counter (TC) to check the timing of the WDT
-/*! oscillator. Note that the TC has a clock source independent from the
-/*! WDT. The oscillator used with the TC is implicitly tested as well: if the
-/*! frequency difference between TC and WDT is more than 50%, the error state is set.
-/*!
-/*! Errors are handled by \ref CLASSB_ERROR_HANDLER_WDT() and there are two configurable
-/*! actions:
+/*!
+/*! The test relies on a Timer Counter (TC) to check the timing of the WDT
+/*! oscillator. Note that the TC has a clock source independent from the
+/*! WDT. The oscillator used with the TC is implicitly tested as well: if the
+/*! frequency difference between TC and WDT is more than 50%, the error state is set.
+/*!
+/*! Errors are handled by \ref CLASSB_ERROR_HANDLER_WDT() and there are two configurable
+/*! actions:
-/*! -# \ref CLASSB_ACTIONS_WDT_RUNTIME_FAILURE() processes systems resets caused by a working WDT.
-/*! -# \ref CLASSB_ACTIONS_WDT_OTHER_FAILURE() deals with other reset causes, e.g. brown-out or
+/*! -# \ref CLASSB_ACTIONS_WDT_OTHER_FAILURE() deals with other reset causes, e.g. brown-out or
-/*! software reset.
-/*!
-/*! In addition to error handler and configurable actions, the user should configure the

```



```

+//!
+//! In addition to error handler and configurable actions, the user should configure the
+//! WDT periods \ref CLASSB_WDT_WPER and \ref CLASSB_WDT_PER.
-//!
-//! \note The WDT should be left enabled at the end of this test and be active at all times. There
-//! are a number of Class B tests that can potentially take longer time than a set WDT period, see
-//! for example \ref classb_crc. If this is the case, a possible solution is to
+//!
+//! \note The WDT should be left enabled at the end of this test and be active at all times. There
+//! are a number of Class B tests that can potentially take longer time than a set WDT period, see
+//! for example \ref classb_crc. If this is the case, a possible solution is to
+//! increase the WDT period before the Class B test and decrease it again afterwards.
+//!
+//@{
+// @ -93,57 +100,55 @@
+//! \name Configuration settings
+//@{
+//! \brief This is the closed period, where WDT cannot be reset.
-//!
-//! This should be given as one of the group configuration settings. The total timeout
-//! period is the sum of the open and closed periods. In order to comply with the standard,
+//!
+//! This should be given as one of the group configuration settings. The total timeout
+//! period is the sum of the open and closed periods. In order to comply with the standard,
+//! this should be at least 50% of the total period.
-#define CLASSB_WDT_WPER WDT_WINDOW_256CLK_gc
+#define CLASSB_WDT_WPER WDT_WINDOW_256CLK_gc

+//! \brief This is the open period, where WDT has to be reset.
-//!
-//! This should be given as one of the group configuration settings. The total timeout
+//!
+//! This should be given as one of the group configuration settings. The total timeout
+//! period is the sum of the open and closed periods. In order to comply with
+//! the standard, this should be no greater than 50% of the total period.
-#define CLASSB_WDT_PER WDT_PERIOD_256CLK_gc
-//@}
+#define CLASSB_WDT_PER WDT_PERIOD_256CLK_gc
+//@}
+//! \internal \name Settings that should not be modified
+//@{

+//@}

-
-
-//! \internal \brief This enum has the valid test states for the WDT test.
+//! \internal \brief This enum has the valid test states for the WDT test.
typedef enum classb_preinit_teststate {
    FAULT_WDT,
    TEST_WDT_1,
    TEST_WDT_2,
    TEST_WDT_3,
+    TEST_WDT_4,
+    TEST_WDT_OK,
} classb_preinit_teststate_t;

-
-
-#if defined(__DOXYGEN__)
-    //! \name Class B Test
-    //@{
-    void classb_wdt_test (void);
-    //@}
+//! \name Class B Test
+//@{
+void classb_wdt_test(void);
+//@}
+
+
+//elif defined(__GNUC__)
+    // Pre-init test function prototype for GCC.
+    void __attribute__((__naked__, section(".init1"))) classb_wdt_test (void);
+    // Pre-init test function type and name for GCC.

```

```

-      #define classb_wdt_test void classb_wdt_test
+// Pre-init test function prototype for GCC.
+void __attribute__((__naked__, section(".init1"))) classb_wdt_test(void);
+// Pre-init test function type and name for GCC.
+#define classb_wdt_test void classb_wdt_test
+elif defined(__ICCAVR__)
-      // Pre-init test function type and name for IAR.
-      #define classb_wdt_test uint8_t __low_level_init
+// Pre-init test function type and name for IAR.
+#define classb_wdt_test uint8_t __low_level_init
+else
-      #error Unknown compiler!
+error Unknown compiler!
#endif

#if defined(__GNUC__) && !defined(__OPTIMIZE__)
-# error Optimization must be enabled to successfully write to protected registers, due to timing constraints.
+error Optimization must be enabled to successfully write to protected registers, due to timing constraints.
#endif

-
+//@}

-#endif // #define CLASSB_WDT_TEST_H
\ No newline at end of file
+#endif // #define CLASSB_WDT_TEST_H

```

## Diff of main\_wdt.c:

```

----- CLASSB_WDT/applications/CLASSB_WDT/main_wdt.c -----
index cc861b2..bcfa987 100644
@@ -1,43 +1,46 @@
-/* This file has been prepared for Doxygen automatic documentation generation.*/
+* This file has been prepared for Doxygen automatic documentation generation.*/
/**
 * \file
 *
+ * \version 1.1
+ *
 * \brief
- *
+ * This is the demo application for the watchdog timer (WDT) test.
+ * This is the demo application for the watchdog timer (WDT) test.
+ *
- *
+ * The self-diagnostic routine for the WDT is executed before
+ * The self-diagnostic routine for the WDT is executed before
+ * the main function is called. If the WDT is not working properly,
+ * the device will hang.
+ *
- *
+ * This demo application sets up an interrupt for SW0, switches
+ * an LED on and then stays in a loop as long as the system is
+ * on a working state. In this loop the WDT is reset according to
+ * its open and closed period settings. If the error flag should be
+ * set, the main program would exit the loop and switch off the LED.
+ *
- *
+ * The following procedure can be followed in order to test that
+ * the correct behavior of the WDT. When button 1 is pressed, the
+ * WDT is reset too early and it will issue a system reset. When
+ * button 2 is pressed, the WDT is reset too late and a system reset
+ * will be issued as well. This "unexpected" reset should be caught
+ * by the self-diagnostic routine, which in this demo is configured
+ * to set the error flag, setup the WDT and continue to the main application.
+ *
+ *
+ * This demo application sets up an interrupt for button 1 and button 2
+ * on OLED1 X plained pro, switches LED 1 on and then stays in a loop
+ * as long as the system is in a working state.
+ * In this loop the WDT is reset according to
+ * its open and closed period settings. If the error flag should be

```

```

+ *          set, the main program would exit the loop and switch off LED 1.
+ *
+ *          The following procedure can be followed in order to test that
+ *          the correct behavior of the WDT. When button 1 is pressed, the
+ *          WDT is reset too early and it will issue a system reset. When
+ *          button 2 is pressed, the WDT is reset too late and a system reset
+ *          will be issued. This "unexpected" reset should be caught
+ *          by the self-diagnostic routine, which in this demo is configured
+ *          to set the error flag, setup the WDT and continue to the main application.
+ *          Therefore, after pressing any of the buttons the system is set on
+ *          a safe state.
+ *
- *
+ *
+ * \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
+ *
@@ -57,10 +60,10 @@
+ * 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
@@ -69,13 +72,11 @@
+ * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
+ * DAMAGE.
+ */
-#include "avr_compiler.h"
-#include "util/delay.h"
-#include "Tiny817xpro.h"

-NO_INIT volatile uint8_t classb_error;
+#include "classb_compiler.h"
+#include "oled1_xpro_attiny817.h"

+NO_INIT volatile uint8_t classb_error;

//! This delay is longer than the closed window setting, but shorter than the total period.
void correct_timing()
@@ -87,7 +88,7 @@ void fast_timing()
{
    _delay_ms(128);
}
-/*! This delay is longer than the total period.
+/*! This delay is longer than the total period.
+void slow_timing()
{

```

```

        _delay_ms(800);
@@ -105,23 +106,21 @@ int main(void)
    // Configure Button 2 to make the CPU reset the WDT to late
    configure_button2();

-    // Set the delay to the correct timing
+    // Set the delay to the correct timing
    our_delay = &correct_timing;
    // Enable interrupts
    sei();

-
-    while(classb_error!=1) {
-        //This calls one of the three delay functions
-        our_delay();
-        //LEDPORT.OUTTGL = PIN1_bm;
-        watchdog_reset();
+
+        while (classb_error != 1) {
+            // This calls one of the three delay functions
+            our_delay();
+            watchdog_reset();
+        };

-        // If we reach here, an error has been detected. Turn off LED.
+        // If we reach here, an error has been detected. Turn off LED 1.
        LED1_OFF;
    }

-
    /*! \brief Button interrupt
    * Set the delay to longer or shorter to trip the WDT error
    */
    @@ -129,16 +128,14 @@ ISR(PORTA_PORT_vect)
    {
        // read interrupt flag
        uint8_t intf = BUTTON1_PORT.INTFLAGS;
-        // If button 1 set delay to shorter than the window to rest the WDT to fast
-        if (intf == BUTTON1_PIN)
-        {
+        // If button 1 set delay to shorter than the window to rest the WDT to fast
+        if (intf == BUTTON1_PIN) {
            our_delay = &fast_timing;
        }
-        // If button 2 make the delay longer than the WDT period
-        if (intf == BUTTON2_PIN)
-        {
+        // If button 2 make the delay longer than the WDT period
+        if (intf == BUTTON2_PIN) {
            our_delay = &slow_timing;
        }
        // Clear interrupt flag
        BUTTON1_PORT.INTFLAGS = intf;
-    }
    \ No newline at end of file
+}

```

## Diff of CLASSB\_WDT.rst:

```

----- CLASSB_WDT/documentation/CLASSB_WDT.rst -----
index 98aecf1..bd40550 100644
@@ -10,26 +10,29 @@ The self-diagnostic routine for the WDT is executed before
the main function is called. If the WDT is not working properly,
the device will hang.

-This demo application sets up an interrupt for SW0, switches
-an LED on and then stays in a loop as long as the system is
-on a working state. In this loop the WDT is reset according to

```

-its open and closed period settings. If the error flag should be  
-set, the main program would exit the loop and switch off the LED.

-

-The following procedure can be followed in order to test that  
-the correct behavior of the WDT. When button 1 is pressed, the  
-WDT is reset too early and it will issue a system reset. When  
-button 2 is pressed, the WDT is reset too late and a system reset  
-will be issued as well. This "unexpected" reset should be caught  
-by the self-diagnostic routine, which in this demo is configured  
-to set the error flag, setup the WDT and continue to the main application.

+This demo application sets up an interrupt for button 1 and button 2  
+on OLE1 X plained pro,switches LED 1 on and then stays in a loop  
+as long as the system is in a working state.

+In this loop the WDT is reset according to  
+its open and closed period settings. If the error flag should be  
+set, the main program would exit the loop and switch off LED 1.

+

+The following procedure can be followed in order to test that  
+the correct behavior of the WDT. When button 1 is pressed, the  
+WDT is reset too early and it will issue a system reset. When  
+button 2 is pressed, the WDT is reset too late and a system reset  
+will be issued. This "unexpected" reset should be caught  
+by the self-diagnostic routine, which in this demo is configured  
+to set the error flag, setup the WDT and continue to the main application.

Therefore, after pressing any of the buttons the system is set on

-a safe state.

+a safe state.

#### 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

-----

@ @ -37,9 +40,19 @ @ 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

+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.

