

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

Files removed/replaced:

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

Files with diff/changelog:

- CLASSB\_SRAM/applications/CLASSB\_SRAM/classb\_sram.c
- CLASSB\_SRAM/applications/CLASSB\_SRAM/classb\_sram.h
- CLASSB\_SRAM/applications/CLASSB\_SRAM/main\_sram.c
- CLASSB\_SRAM/documentation/CLASSB\_SRAM.rst

## Diff of classb\_sram.c:

```
----- CLASSB_SRAM/applications/CLASSB_SRAM/classb_sram.c -----
@@ -2,23 +2,23 @@
/**
 * \file
 *
 - * \version 1.2
 + * \version 1.3
 *
 * \brief
 *
      SRAM test based on the March X algorithm.
 - *
 - * \par Application note:
 - *   AVR1610: Guide to IEC60730 Class B compliance with XMEGA
 + *
 + * \par Application note:
 + *   AN2632: Guide to IEC60730 Class B compliance with TinyAVR 1-series
 *
 * \par Documentation
 - *   For comprehensive code documentation, supported compilers, compiler
 + *   For comprehensive code documentation, supported compilers, compiler
 *   settings and supported devices see readme.html
 *
 * \author
 - *   Atmel Corporation: http://www.atmel.com \n
 - *   Support email: avr@atmel.com
 - *
 - * Copyright (C) 2012 Atmel Corporation. All rights reserved.
 + *   Microchip Technology: http://www.microchip.com
 + *   Support at http://www.microchip.com/support/
 + *
 + * Copyright (C) 2019 Microchip Technology. All rights reserved.
 *
 * \page License
 *
@@ -32,16 +32,16 @@
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 - * 3. The name of Atmel may not be used to endorse or promote products derived
 + * 3. The name of Microchip may not be used to endorse or promote products derived
 * from this software without specific prior written permission.
 *
 * 4. This software may only be redistributed and used in connection with an
 - * Atmel AVR product.
 + * Microchip AVR product.
 *
 - * THIS SOFTWARE IS PROVIDED BY ATMEL "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 ATMEL 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
@@ -51,54 +51,53 @@
* DAMAGE.
*/
#include "avr_compiler.h"
#include "classb_compiler.h"
#include "classb_sram.h"
#include "error_handler.h"
#include "classb_error_handler.h"

//\ingroup classb_sram
//@{

#if defined(__ICCAVR__)
- #pragma location=INTERNAL_SRAM_START
- __no_init uint8_t classb_buffer[CLASSB_SEC_SIZE+CLASSB_OVERLAP_SIZE];
-#elif defined(__GNUC__)
- static uint8_t classb_buffer[CLASSB_SEC_SIZE+CLASSB_OVERLAP_SIZE] __attribute__((section
(".classb_sram_buffer")));
-#elif defined(__DOXYGEN__)
- //! @cond IAR
-
- //! \internal
-
- //! @endcond
-
- //! \brief Buffer of SRAM memory that is used to store the information from
- //! sections that are tested.
- //!
- //! The buffer is placed in the beginning of SRAM (see \c INTERNAL_SRAM_START). The size
- //! in bytes should be \c INTERNAL_SRAM_SIZE/CLASSB_NSECS+CLASSB_OVERLAP_SIZE. As an example, let
us
- //! suppose the device has \c INTERNAL_SRAM_SIZE=4096 Bytes of SRAM, \ref CLASSB_NSECS = 8 and
- //! \ref CLASSB_OVERLAP=25. Given that the number of sections is a divisor of \c INTERNAL_SRAM_SIZE,
- //! there would be in principle exactly 8 sections, each with a size of 512 Bytes. For 25%
- //! overlap we need to extend the first section (the buffer) by 128 Bytes, and the second
- //! section is shortened accordingly. Therefore, we would have to reserve 512+128=640 Bytes for
- //! the buffer.
- //!
- //! The implementation depends on the compiler:
- //! For IAR it is easy because variables can be placed in a specific address using the a location
- //! pragma directive.
- //! For GCC the following steps should be followed:
- //! - Firstly, it is necessary to define in <em> AVR/GNU C linker > Memory settings </em> a section
- //! that starts at \c INTERNAL_SRAM_START, for example:
- //! <tt> .classb_sram_buffer=0x3E00</tt>.
- //! - Secondly, the buffer variable has to be placed in that memory section. This can be done
- //! appending this to the definition of the variable:
- //! <tt> __attribute__((section(".classb_sram_buffer")))</tt>.
- //! - Thirdly, the <tt>.data</tt> section has to be moved so that there is no overlap:
- //! the start should be <tt>INTERNAL_SRAM_START + buffer_size</tt>. For example
- //! 640=0x280 and then we would set
- //! <tt> .data=0x2280</tt>.
- //! <tt> uint8_t classb_buffer[CLASSB_SEC_SIZE+CLASSB_OVERLAP_SIZE];
-#endif
+ #pragma location = INTERNAL_SRAM_START
+ __no_init uint8_t classb_buffer[CLASSB_SEC_SIZE + CLASSB_OVERLAP_SIZE];
+ #elif defined(__GNUC__)
+ static uint8_t classb_buffer[CLASSB_SEC_SIZE + CLASSB_OVERLAP_SIZE] __attribute__((section(".classb_sram_buffer")));
+ #elif defined(__DOXYGEN__)
+ //! @cond IAR
+
+ //! \internal
+
+ //! @endcond
+
+ //! \brief Buffer of SRAM memory that is used to store the information from
+ //! sections that are tested.

```

```

+//!
+//! The buffer is placed in the beginning of SRAM (see \c INTERNAL_SRAM_START). The size
+//! in bytes should be \c INTERNAL_SRAM_SIZE/CLASSB_NSECS+CLASSB_OVERLAP_SIZE. As an example, let us
+//! suppose the device has \c INTERNAL_SRAM_SIZE=4096 Bytes of SRAM, \ref CLASSB_NSECS = 8 and
+//! \ref CLASSB_OVERLAP=25. Given that the number of sections is a divisor of \c INTERNAL_SRAM_SIZE,
+//! there would be in principle exactly 8 sections, each with a size of 512 Bytes. For 25%
+//! overlap we need to extend the first section (the buffer) by 128 Bytes, and the second
+//! section is shortened accordingly. Therefore, we would have to reserve 512+128=640 Bytes for
+//! the buffer.
+//!
+//! The implementation depends on the compiler:
+//! For IAR it is easy because variables can be placed in a specific address using the a location
+//! pragma directive.
+//! For GCC the following steps should be followed:
+//! - Firstly, it is necessary to define in <em> AVR/GNU C linker > Memory settings </em> a section
+//! that starts at \c INTERNAL_SRAM_START, for example:
+//! <tt> .classb_sram_buffer=0x3E00</tt>.
+//! - Secondly, the buffer variable has to be placed in that memory section. This can be done
+//! appending this to the definition of the variable:
+//! <tt> __attribute__((section(".classb_sram_buffer")))</tt>.
+//! - Thirdly, the <tt>.data</tt> section has to be moved so that there is no overlap:
+//! the start should be <tt>INTERNAL_SRAM_START + buffer_size</tt>. For example
+//! 640=0x280 and then we would set
+//! <tt>.data=0x2280</tt>.
+uint8_t classb_buffer[CLASSB_SEC_SIZE + CLASSB_OVERLAP_SIZE];
+#endif

/* \brief This function executes March X test for a memory section at a time.
*
@@ -108,7 +107,7 @@
* is <tt>CLASSB_SEC_SIZE + CLASSB_OVERLAP_SIZE</tt>.
* - The algorithm is set to start \c CLASSB_OVERLAP_SIZE bytes before the start of the current section,
* thus overlapping \c CLASSB_OVERLAP_SIZE bytes with the previous section.
-
+
* Special cases of memory sections are:
* - The first section to test is the buffer segment, which starts at \c INTERNAL_SRAM_START
* and ends at <tt>CLASSB_SEC_SIZE + CLASSB_OVERLAP_SIZE</tt>. In this case there is no previous segment, so no
@@ -121,143 +120,157 @@
* - The "remainder" segment, if any. This section has a size of \c CLASSB_SEC_REM, so the total
* number of bytes will be <tt>CLASSB_SEC_REM + CLASSB_OVERLAP_SIZE</tt>.
*/
-void classb_sram_test()
+void classb_sram_test()
{
-
- // This variable keeps track of the section to test.
+
+ // This variable keeps track of the section to test.
+ static uint8_t current_section = 0;
-
- switch (current_section)
- {
+
+ switch (current_section) {
+ case 0:
- // Test the buffer, which starts at INTERNAL_SRAM_START and ends at CLASSB_SEC_SIZE +
- CLASSB_OVERLAP_SIZE. There is no overlap with previous segments.
+ // Test the buffer, which starts at INTERNAL_SRAM_START and ends at CLASSB_SEC_SIZE +
+ CLASSB_OVERLAP_SIZE.
+ // There is no overlap with previous segments.
+ classb_marchX((uint8_t *)INTERNAL_SRAM_START, classb_buffer, CLASSB_SEC_SIZE +
+ CLASSB_OVERLAP_SIZE);
+ break;
- case 1:
- // Test the first section, which size shrunk from below by the buffer when there is overlap.
+ // Test the first section, which size shrunk from below by the buffer when there is overlap.
+ // In order to overlap with the buffer, we simply start at INTERNAL_SRAM_START + CLASSB_SEC_SIZE.
+ classb_marchX((uint8_t *)INTERNAL_SRAM_START + CLASSB_SEC_SIZE, classb_buffer,
+ CLASSB_SEC_SIZE);

```

```

        break;
    case CLASSB_NSECS:
        // We test the last section of size SRAM_SIZE % CLASSB_NSECS
        // Limit size to the amount of memory remaining when dividing SRAM_SIZE with CLASSB_NSECS.
        - classb_marchX((uint8_t *)INTERNAL_SRAM_START + CLASSB_NSECS * CLASSB_SEC_SIZE -
CLASSB_OVERLAP_SIZE, classb_buffer, CLASSB_SEC_REM + CLASSB_OVERLAP_SIZE);
        + classb_marchX((uint8_t *)INTERNAL_SRAM_START + CLASSB_NSECS * CLASSB_SEC_SIZE -
CLASSB_OVERLAP_SIZE,
        + classb_buffer,
        + CLASSB_SEC_REM + CLASSB_OVERLAP_SIZE);
        break;
    default:
        - // Sections in the middle. We start CLASSB_OVERLAP_SIZE before the segment and test
CLASSB_SEC_SIZE+CLASSB_OVERLAP_SIZE bytes
        - classb_marchX((uint8_t *)INTERNAL_SRAM_START + current_section * CLASSB_SEC_SIZE -
CLASSB_OVERLAP_SIZE, classb_buffer, CLASSB_SEC_SIZE + CLASSB_OVERLAP_SIZE);
        + // Sections in the middle. We start CLASSB_OVERLAP_SIZE before the segment and test
        + // CLASSB_SEC_SIZE+CLASSB_OVERLAP_SIZE bytes
        + classb_marchX((uint8_t *)INTERNAL_SRAM_START + current_section * CLASSB_SEC_SIZE -
CLASSB_OVERLAP_SIZE,
        + classb_buffer,
        + CLASSB_SEC_SIZE + CLASSB_OVERLAP_SIZE);
        break;
    }
}

// Increase section count for next iteration, or reset if all memory is tested.
current_section++;
- if (current_section > CLASSB_NSEC_TOTAL-1)
+ if (current_section > CLASSB_NSEC_TOTAL - 1)
    current_section = 0;
}

-
+
-/*! \internal\brief This function executes the the March X algorithm in a section of
+/*! \internal\brief This function executes the the March X algorithm in a section of
 * SRAM memory.
 *
 * The following steps are followed:
 * -# The content of the area to be tested is copied to the buffer. This does not
 * apply when the buffer is tested.
 * -# The March X algorithm is applied to the section to test for coupling faults.
 * -# Memory content is copied back from the buffer. This does not apply when the
 * buffer is tested.
 *
 * This function requires that all variables are placed in registers by the compiler.
 * This is achieved by using the \c register specifier (probably not necessary if the right
 * level of optimization is chosen).
 *
 * The following steps are followed:
 * -# The content of the area to be tested is copied to the buffer and then the buffer content is compared to the current section
 * to make sure the copy was successful. This does not apply when the buffer is tested.
 * -# The March X algorithm is applied to the section to test for coupling faults.
 * -# Memory content is copied back from the buffer and compared to make sure the copy is OK.
 * This does not apply when the buffer is tested.
 *
 * This function requires that all variables are placed in registers by the compiler.
 * This is achieved by using the \c register specifier (probably not necessary if the right
 * level of optimization is chosen).
 *
 * If there should be an error in SRAM a local variable (implemented in a register)
 * would be flagged. This would lead to the error handler \ref CLASSB_ERROR_HANDLER_SRAM()
 *
 * If there should be an error in SRAM a local variable (implemented in a register)
 * it would be flagged. This would lead to the error handler \ref CLASSB_ERROR_HANDLER_SRAM()
 * being called.
 *
 * \param p_sram Pointer to first byte in memory area to be tested
 * \param p_buffer Pointer to first byte in the buffer
 * \param size Size of area to be tested in bytes.
 */

```

```

-void classb_marchX(register volatile uint8_t * p_sram, register volatile uint8_t * p_buffer, register uint16_t size)
-
-    register uint16_t i = 0;
-    register uint8_t error = 0;
+void classb_marchX(register volatile uint8_t *p_sram, register volatile uint8_t *p_buffer, register uint16_t size)
+{
+    register uint16_t i = 0;
+    register uint8_t error = 0;
+
+    // Save content of the section: copy to buffer unless we test the buffer
+    if (p_buffer != p_sram)
+        for (uint16_t i = 0; i < size; i++)
+            *(p_buffer+i) = *(p_sram+i);
+
+    {
+        for (uint16_t i = 0; i < size; i++)
+            *(p_buffer + i) = *(p_sram + i);
+        // Check that saved content is uncorrupted
+        for (i = 0; i < size; i++) {
+            uint8_t temp_buffer = *(p_buffer + i);
+            if (temp_buffer != *(p_sram + i))
+                error = 1;
+        }
+    }
+
+
+    // Test phase 1: write 0 to all bit locations.
+    // Test phase 1: write 0 to all bit locations.
+    for (i = 0; i < size; i++)
+        *(p_sram+i) = 0x00;
+
+        *(p_sram + i) = 0x00;
+
+    // Test phase 2: read 0, write FF.
+    for (i = 0; i < size; i++)
+    {
+        if (*(p_sram+i) != 0x00)
+            // Test phase 2: read 0, write FF.
+            for (i = 0; i < size; i++) {
+                if (*(p_sram + i) != 0x00)
+                    error = 1;
+                else
+                    *(p_sram+i) = 0xFF;
+                else
+                    *(p_sram + i) = 0xFF;
+            }
+
+
+    // Test phase 3: read FF, write 0 (reverse order).
+    for(i = size ; i>0; i--)
+    {
+        if (*(p_sram+i-1) != 0xFF)
+            error = 1;
+        else
+            *(p_sram+i-1) = 0x00;
+
+
+    // Test phase 3: read FF, write 0 (reverse order).
+    for (i = size; i > 0; i--) {
+        if (*(p_sram + i - 1) != 0xFF)
+            error = 1;
+        else
+            *(p_sram + i - 1) = 0x00;
+    }
+
+
+    // Test phase 4: read 0.
+
+    // Test phase 4: read 0.
+    for (i = 0; i < size; i++)
+        if (*(p_sram+i) != 0x00)

```

```

-             error = 1;
+             if (*(p_sram + i) != 0x00)
+                 error = 1;
-
-#ifndef CLASSB_SRAM_INTRAWORD_TEST
// Intra-word march test.
for (i = 0; i < size; i++) {
-     *(p_sram+i) = 0x55;
-     if (*(p_sram+i) != 0x55)
-         error = 1;
+     *(p_sram + i) = 0x55;
+     if (*(p_sram + i) != 0x55)
+         error = 1;
-
-     *(p_sram+i) = 0xAA;
-     if (*(p_sram+i) != 0xAA)
-         error = 1;
+     *(p_sram + i) = 0xAA;
+     if (*(p_sram + i) != 0xAA)
+         error = 1;
-
-     *(p_sram+i) = 0x33;
-     if (*(p_sram+i) != 0x33)
-         error = 1;
-
-     *(p_sram+i) = 0xCC;
-     if (*(p_sram+i) != 0xCC)
-         error = 1;
-
-     *(p_sram+i) = 0xF0;
-     if (*(p_sram+i) != 0xF0)
-         error = 1;
-     *(p_sram+i) = 0x0F;
-     if (*(p_sram+i) != 0x0F)
-         error = 1;
-
+     *(p_sram + i) = 0x33;
+     if (*(p_sram + i) != 0x33)
+         error = 1;
+
+     *(p_sram + i) = 0xCC;
+     if (*(p_sram + i) != 0xCC)
+         error = 1;
+
+     *(p_sram + i) = 0xF0;
+     if (*(p_sram + i) != 0xF0)
+         error = 1;
+     *(p_sram + i) = 0x0F;
+     if (*(p_sram + i) != 0x0F)
+         error = 1;
+
+ }
+
+ }

#endif

// Restore content of the section: copy from buffer, unless buffer is tested
if (p_buffer != p_sram)
+ {
+     for (i = 0; i < size; i++)
-         *(p_sram+i) = *(p_buffer+i);
+         *(p_sram + i) = *(p_buffer + i);
+
+     // Check that restored content is uncorrupted
+     for (i = 0; i < size; i++) {
+         uint8_t temp_buffer = *(p_buffer + i);
+         if (temp_buffer != *(p_sram + i))
+             error = 1;
+     }
+ }

```

```

        // Call the error handler if there was an error.
-       if (error)
+       if (error)
            CLASSB_ERROR_HANDLER_SRAM();
    }

-//@}
\ No newline at end of file
+//@}

```

## Diff of classb\_sram.h:

```

----- CLASSB_SRAM/applications/CLASSB_SRAM/classb_sram.h -----
index da60a76..a6403f1 100644
@@ -1,23 +1,23 @@
/* This file has been prepared for Doxygen automatic documentation generation.*/
/**
- * \file
+ * \file
- * \version 1.1
+ * \version 1.2
- * \brief Settings for the SRAM test.
+ * \brief Settings for the SRAM test.
- *
+ *
- * \par Application note:
+ * \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
+ * \par Documentation
- *   For comprehensive code documentation, supported compilers, compiler
+ *   For comprehensive code documentation, supported compilers, compiler
- *   settings and supported devices see readme.html
+ *   settings and supported devices see readme.html
- *
+ *
- * \author
+ * \author
- *   Microchip Technology: http://www.microchip.com \n
+ *   Microchip Technology: http://www.microchip.com \n
- *   Support at http://www.microchip.com/support/ \n
+ *   Support at http://www.microchip.com/support/ \n
- *
+ *
- * Copyright (C) 2017 Microchip Technology. All rights reserved.
+ * Copyright (C) 2017 Microchip Technology. All rights reserved.
- *   Microchip Technology: http://www.microchip.com
+ *   Microchip Technology: http://www.microchip.com
- *   Support at http://www.microchip.com/support/
+ *   Support at http://www.microchip.com/support/
- *
+ *
- * Copyright (C) 2019 Microchip Technology. All rights reserved.
+ * Copyright (C) 2019 Microchip Technology. All rights reserved.
- *
+ *
- * \page License
+ * \page License
- *
+ *
@@ -37,10 +37,10 @@
- * 4. This software may only be redistributed and used in connection with an
+ * 4. This software may only be redistributed and used in connection with an
- * Microchip AVR product.
+ * 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
+ * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
- * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE
+ * 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
+ * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
- * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
+ * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
- * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
+ * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
@@ -53,48 +53,48 @@
#ifndef _SRAM_H_
#define _SRAM_H_
-
+
-//! \defgroup classb_sram Internal SRAM Test
+//! \defgroup classb_sram Internal SRAM Test
-//!
+//!
-//! \brief This self-diagnostic test checks the internal SRAM memory for faults.
+//! \brief This self-diagnostic test checks the internal SRAM memory for faults.
-//!
+//!

```

```

-// The test \ref classb_sram_test() divides the internal SRAM into \ref CLASSB_NSECS sections that are
-// tested in turns with a March X algorithm (see \ref marchx). The simplest behavior of the test
-// is when there is no overlap between memory sections. In this case all sections have the same
-// size, except possibly the last one (see \ref CLASSB_NSECS). The first memory section (referred to as
-// the buffer) is reserved: it is used by the test to store the content of the other sections
-// while they are being tested (for more details see \ref classb_buffer).
-//
-// If there is overlap (see \ref CLASSB_OVERLAP), every time a memory section is tested a part of the
-// previous section is tested as well. Note that this does not apply to the buffer, since it is
-// the first section. The size of the buffer is then expanded with respect to the previous case.
-// Further, the size of the second section is decreased correspondingly.
-//
-// If there should be an error in internal SRAM the error handler \ref CLASSB_ERROR_HANDLER_SRAM()
+//
+// The test \ref classb_sram_test() divides the internal SRAM into \ref CLASSB_NSECS sections that are
+// tested in turns with a March X algorithm (see \ref marchx). The simplest behavior of the test
+// is when there is no overlap between memory sections. In this case all sections have the same
+// size, except possibly the last one (see \ref CLASSB_NSECS). The first memory section (referred to as
+// the buffer) is reserved: it is used by the test to store the content of the other sections
+// while they are being tested (for more details see \ref classb_buffer).
+//
+// If there is overlap (see \ref CLASSB_OVERLAP), every time a memory section is tested a part of the
+// previous section is tested as well. Note that this does not apply to the buffer, since it is
+// the first section. The size of the buffer is then expanded with respect to the previous case.
+// Further, the size of the second section is decreased correspondingly.
+//
+// If there should be an error in internal SRAM the error handler \ref CLASSB_ERROR_HANDLER_SRAM()
// would be called.
-//
+//
+// \note Interrupts must be disabled during this test.
-//
+//
+// \section marchx March X
-//
-// The chosen algorithm is <em>March X</em>. This consists on the following steps:
-// \f[ \Updownarrow (w_{\textbf{D}}); \Uparrow(r_{\textbf{D}}, w_{\bar{\textbf{D}}}); \Downarrow(r_{\bar{\textbf{D}}}, w_{\textbf{D}});
\Updownarrow(r_{\textbf{D}}) \,; \f]
-// where \f$w_{\f}$ denotes a write operation, \f$r_{\f}$ denotes a read operation,
+//
+// The chosen algorithm is <em>March X</em>. This consists on the following steps:
+// \f[ \Updownarrow (w_{\textbf{D}}); \Uparrow(r_{\textbf{D}}, w_{\bar{\textbf{D}}}); \Downarrow(r_{\bar{\textbf{D}}},
+// w_{\textbf{D}}); \Updownarrow(r_{\textbf{D}}) \,; \f]
+// where \f$w_{\f}$ denotes a write operation, \f$r_{\f}$ denotes a read operation,
+// \f$\textbf{D}\f$ is any data background, \f$\bar{\textbf{D}}\f$ is the complement
-// of \f$\textbf{D}\f$ and the arrows refer to the addressing order. In our implementation
+// of \f$\textbf{D}\f$ and the arrows refer to the addressing order. In our implementation
+// we have chosen \f$\textbf{D} = \text{0x00}\f$.
+//
+// In order to detect some intra-word CFs that are not considered by march x fault model,
+// we have included the following optional march element:
-//
+//
+// \f[ \Updownarrow (w_{\textbf{D}_0}, r_{\textbf{D}_0}) \ldots w_{\textbf{D}_d}, r_{\textbf{D}_d}) \,; \f]
-// where the background sequence is \f$\text{0x00, 0xFF, 0x55, 0xAA, 0x33, 0xCC, 0x0F, 0xF0}\f$.
-// This intra-word test is only executed if \ref CLASSB_SRAM_INTRAWORD_TEST is defined. The elements
-// that correspond to the first two data backgrounds are redundant because the first part of the
-// test includes them. This optional test will detect all intra-word state CFs considered by the
+// where the background sequence is \f$\text{0x00, 0xFF, 0x55, 0xAA, 0x33, 0xCC, 0x0F, 0xF0}\f$.
+// This intra-word test is only executed if \ref CLASSB_SRAM_INTRAWORD_TEST is defined. The elements
+// that correspond to the first two data backgrounds are redundant because the first part of the
+// test includes them. This optional test will detect all intra-word state CFs considered by the
+// unrestricted CF model.
-//
-//
+//
+//
+// @{}
+//
+// \name Configuration settings

```



@@ -102,11 +102,19 @@

/\*! \brief Number of sections the SRAM is divided into for testing.

/\*!

-/\*! It is advisable that \c INTERNAL\_SRAM\_SIZE is divisible by the number of  
-/\*! sections and, therefore, recommended values are 2, 4, 8, 16, etc. Otherwise an extra  
-/\*! section will be added with the remainder of the division as size. Note that the higher  
-/\*! the number of sections the smaller the size of \ref classb\_buffer, i.e. the section of memory  
-/\*! that is reserved for the test) and the faster each partial test is completed.

+/\*! It is advisable that \c INTERNAL\_SRAM\_SIZE is divisible by the number of  
+/\*! sections and, therefore, recommended values are 2, 4, 8, 16, etc. Otherwise an extra  
+/\*! section will be added with the remainder of the division as size. Note that the higher  
+/\*! the number of sections the smaller the size of \ref classb\_buffer, i.e. the section of memory  
+/\*! that is reserved for the test) and the faster each partial test is completed.

+/\*!

+/\*!

+/\*! IMPORTANT!

+/\*! When \c CLASSB\_NSECS is changed, its important to do changes in project settings as well to avoid

+/\*! that important register values are stored in the classb\_buffer. If this is not fixed and registers

+/\*! are stored in the classb\_buffer, the code can stop working since the register are overwritten and not restored.

+

+/\*! Look in main\_sram.c on how to calculate the data section to avoid overlap with the buffer.

#define CLASSB\_NSECS 8

/\*\*

@@ -117,49 +125,47 @@

#define CLASSB\_OVERLAP 25UL

#ifdef \_\_DOXYGEN\_\_

-/\*! \brief If defined an intra-word test will be added after the inter-word test.

-/\*!

-/\*! Given the layout of the memory, the probability of intra-word coupling faults

-/\*! is greatly diminished. However, for extra safety the test can be expanded to

-/\*! check some intra word coupling faults.

-#define CLASSB\_SRAM\_INTRAWORD\_TEST

+/\*! \brief If defined an intra-word test will be added after the inter-word test.

+/\*!

+/\*! Given the layout of the memory, the probability of intra-word coupling faults

+/\*! is greatly diminished. However, for extra safety the test can be expanded to

+/\*! check some intra word coupling faults.

+#define CLASSB\_SRAM\_INTRAWORD\_TEST

#else

-// #define CLASSB\_SRAM\_INTRAWORD\_TEST

+#define CLASSB\_SRAM\_INTRAWORD\_TEST

#endif

//@}

-

/\*! \internal

/\*! \name Constants that are automatically computed.

-//@{

-/\*! \internal The size of each segment in bytes

+//@{

+/\*! \internal The size of each segment in bytes

#define CLASSB\_SEC\_SIZE (INTERNAL\_SRAM\_SIZE / CLASSB\_NSECS)

-/\*! \internal The size of the last segment in bytes (when \c INTERNAL\_SRAM\_SIZE is not divisible by \ref CLASSB\_NSECS)

+/\*! \internal The size of the last segment in bytes (when \c INTERNAL\_SRAM\_SIZE is not divisible by \ref CLASSB\_NSECS)

#define CLASSB\_SEC\_REM (INTERNAL\_SRAM\_SIZE % CLASSB\_NSECS)

-/\*! \internal Size of overlap in bytes. When testing a memory section, the algorithm starts \c CLASSB\_OVERLAP\_SIZE

+/\*! \internal Size of overlap in bytes. When testing a memory section, the algorithm starts \c CLASSB\_OVERLAP\_SIZE

/\*! bytes behind the start of the section.

-#define CLASSB\_OVERLAP\_SIZE (CLASSB\_SEC\_SIZE\*CLASSB\_OVERLAP)/100

+#define CLASSB\_OVERLAP\_SIZE (CLASSB\_SEC\_SIZE \* CLASSB\_OVERLAP) / 100

/\*! \internal Total number of segments, including remainder if present.

#if (CLASSB\_SEC\_REM == 0)

-# define CLASSB\_NSEC\_TOTAL CLASSB\_NSECS

+#define CLASSB\_NSEC\_TOTAL CLASSB\_NSECS

```

#else
-# define CLASSB_NSEC_TOTAL CLASSB_NSECS + 1
+#define CLASSB_NSEC_TOTAL CLASSB_NSECS + 1
#endif
/*@}

/*! \name Class B Test
/*@{
-void classb_sram_test( void );
+void classb_sram_test(void);
/*@}

/*! \internal\name March X Algorithm
/*@{
-void classb_marchX(register volatile uint8_t * p_sram, register volatile uint8_t * p_buffer, register uint16_t size);
+void classb_marchX(register volatile uint8_t *p_sram, register volatile uint8_t *p_buffer, register uint16_t size);
/*@}
-
-
+
+
/*@}
-#endif
\ No newline at end of file
+#endif

```

## Diff of main\_sram.c:

```

----- CLASSB_SRAM/applications/CLASSB_SRAM/main_sram.c -----
index 75983e5..fc5d53f 100644
@@ -2,53 +2,67 @@
/**
 * \file
 *
- * \version 1.2
+ * \version 1.3
 *
 * \brief
 *
 * Example application for the SRAM test.
 *
- * This example shows how the SRAM test can be embedded in an application.
- * The application lights up an LED that signals correct behavior of the
- * system. Then it enters the main loop, where the SRAM test is called
- * periodically. This test is based on March X and it checks a segment of
- * SRAM memory at a time. After the SRAM test, a second LED is toggled in
- * order to visualize when the partial tests are finished. If errors were
- * found, the global error flag would be set by the test. This would lead
- * to the application leaving the main loop and switching off the first LED.
- *
- * The configuration for memory size, amount of segments, buffer size, etc.
- * can be set up in classb_sram.h. Note that if GCC is used and the size of
- * the test buffer changes (as a result of a change in the number of segments
- * or the amount of segment overlap), the new buffer size must be configured
+ * This example shows how the SRAM test can be embedded in an application.
+ * The application lights up LED1 that signals correct behavior of the
+ * system. Then it enters the main loop, where the SRAM test is called
+ * periodically. This test is based on March X algorithm and it checks a segment of
+ * SRAM memory at a time.
+ * If errors were found, the global error flag would be set by the test. This would lead
+ * to the application leaving the main loop and switching off LED1.
+ *
+ * The configuration for memory size, amount of segments, buffer size, etc.
+ * can be set up in classb_sram.h. Note that if GCC is used and the size of
+ * the test buffer changes (as a result of a change in the number of segments
+ * or the amount of segment overlap), the new buffer size must be configured
+ * in the linker as when using GCC.
+ *
- * the memory configuration sets up a buffer at the start of SRAM. The address to

```

```

+ * The memory configuration sets up a buffer at the start of SRAM. The address to
+ * the start of SRAM can be different from device to device. See the respective datasheet.
- * For the tiny817 the start address is given as 3E00, the linker option to use for the
- * start of the buffer is:
+ *
+ * For the tiny817 the start address is given as 3E00.'
+ * In project properties -> Toolchain -> AVR/GNU Linker -> Memory Settings you must add two memory
segments in SRAM segments.
+ *
+ * The first memory segment is for the start of .classb_sram_buffer and should be added as:
+ * .classb_sram_buffer=0x803E00
+ *
- * -Wl,-section-start=.classb_sram_buffer=0x803E00
+ * The second segment is to define the data segment so it doesn't overlap the classb_sram_buffer.
+ * This buffer must be adjusted to reflect the number of segments CLASSB_NSECS
+ *
+ * The way to calculate where the data segment starts is like this:
+ * .data = 1.25*(INTERNAL_SRAM_SIZE / CLASSB_NSECS) + 0x3e00
+ *
+ * Example 1:
+ * INTERNAL_SRAM_SIZE = 512(0x200), CLASSB_NSECS = 8
+ * 1.25*(512/8) = 80 = 0x50 + 0x3e00 = 0x3E50
+ *
+ * .data=0x3e50
+ *
+ * Example 2:
+ * INTERNAL_SRAM_SIZE = 512(0x200), CLASSB_NSECS = 13
+ * 1.25*(512/13) = 49,23 = 0x31 + 0x3e00 = 0x3E31
+ *
- * Now the data section needs to be changed so that it does not overlap the buffer section.
- * In this example the buffer is 64 bytes and 16 bytes overlap in hex this is an offset of
- * 0x59 bytes. The data section start must be changed by the following linker option:
+ * .data=0x3e31
+ *
- * -Wl,-section-start=.data=0x803E50
+ *
+ * For IAR the above is not needed.
- *
- * \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
+ * 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
- *
- * Copyright (C) 2017 Microchip Technology. All rights reserved.
+ * Microchip Technology: http://www.microchip.com
+ * Support at http://www.microchip.com/support/
+ *
+ * Copyright (C) 2019 Microchip Technology. All rights reserved.
+ *
+ * \page License
+ *
@@ -68,10 +82,10 @@
+ * 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
@@ -81,33 +95,31 @@
+ * DAMAGE.
+ */
-#include "avr_compiler.h"
+#include "classb_compiler.h"
#include "classb_sram.h"
-#include "Tiny817xpro.h"
+#include "oled1_xpro_attiny817.h"

/** \brief Global error variable */
volatile uint8_t classb_error;

-
-/*! \brief
+/*! \brief
+ * Main entry point of the application. Sets up board hardware,
+ * and waits until an error has occurred.
- * \callgraph
+ * \call graph
+ */
int main(void)
{
- // Initialize error variabel
+ // Initialize error variable
classb_error = 0;
configure_LED1();
- while(!classb_error)
- {
- //disable interrupts
+ while (!classb_error) {
+ // disable interrupts
cli();
classb_sram_test();
- //enable interrupts
+ // enable interrupts
sei();
};
-
+ // Turn off LED to indicate error.
LED1_OFF;
-}
\ No newline at end of file
+}

```

## Diff of CLASSB\_SRAM.rst:

```

----- CLASSB_SRAM/documentation/CLASSB_SRAM.rst -----
index fd43478..f6f1a00 100644
@@ -6,40 +6,55 @@ Example application for the SRAM test.
The application is made to execute on a ATtiny817 Xplained Pro
with the OLED1 Xplained connected to the EXT1 header.

-This example shows how the SRAM test can be embedded in an application.
-The application lights up an LED that signals correct behavior of the
-system. Then it enters the main loop, where the SRAM test is called
-periodically. This test is based on March X and it checks a segment of
-SRAM memory at a time. If errors are found, the global error flag will
-be set. This would lead to the application leaving the main loop and
-switching off the first LED.
-
-The configuration for memory size, amount of segments, buffer size, etc.
-can be set up in classb_sram.h. Note that if GCC is used and the size of

```

-the test buffer changes (as a result of a change in the number of segments  
-or the amount of segment overlap), the new buffer size must be configured  
-in the linker as well when using GCC.

-

-The memory configuration sets up a buffer at the start of SRAM. The address to

+This example shows how the SRAM test can be embedded in an application.

+The application lights up LED1 that signals correct behavior of the

+system. Then it enters the main loop, where the SRAM test is called

+periodically. This test is based on March X algorithm and it checks a segment of

+SRAM memory at a time.

+If errors were found, the global error flag would be set by the test. This would lead

+to the application leaving the main loop and switching off LED1.

+

+The configuration for memory size, amount of segments, buffer size, etc.

+can be set up in classb\_sram.h. Note that if GCC is used and the size of

+the test buffer changes (as a result of a change in the number of segments

+or the amount of segment overlap), the new buffer size must be configured

+in the linker as when using GCC.

+

+The memory configuration sets up a buffer at the start of SRAM. The address to

the start of SRAM can be different from device to device. See the respective datasheet.

-For the tiny817 the start address is given as 3E00, the linker option to use for the

-start of the buffer is:

-

--Wl,-section-start=.classb\_sram\_buffer=0x803E00

-

-Now the data section needs to be changed so that it does not overlap the buffer section.

-In this example the buffer is 64 bytes and 16 bytes overlap in hex this is an offset of

-0x59 bytes. The data section start must be changed by the following linker option:

-

--Wl,-section-start=.data=0x803E50

+

+For the tiny817 the start address is given as 3E00.'

+In project properties -> Toolchain -> AVR/GNU Linker -> Memory Settings you must add two memory segments in SRAM segments.

+

+The first memory segment is for the start of .classb\_sram\_buffer and should be added as:

+.classb\_sram\_buffer=0x803E00

+

+The second segment is to define the data segment so it doesn't overlap the classb\_sram\_buffer.

+This buffer must be adjusted to reflect the number of segments CLASSB\_NSECS

+

+The way to calculate where the data segment starts is like this:

+.data = 1.25\*(INTERNAL\_SRAM\_SIZE / CLASSB\_NSECS) + 0x3e00

+

+Example 1:

+INTERNAL\_SRAM\_SIZE = 512(0x200), CLASSB\_NSECS = 8

+1.25\*(512/8) = 80 = 0x50 + 0x3e00 = 0x3E50

+

+.data=0x3e50

+

+Example 2:

+INTERNAL\_SRAM\_SIZE = 512(0x200), CLASSB\_NSECS = 13

+1.25\*(512/13) = 49.23 = 0x31 + 0x3e00 = 0x3E31

+

+.data=0x3e31

For IAR the above is not needed.

-

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

-----  
@@ -47,8 +62,8 @@ 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.

@@ -56,3 +71,13 @@ Running the demo

In Toolchain go to AVR/GNU Linker -> Miscellaneous and paste in:

-Wl,-section-start=.classb\_sram\_buffer=0x803E00 -Wl,-section-start=.data=0x803E50

4. 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.