SMART ARM-based Microcontrollers

AT12869: ICM Usage on SAM S/E/V70/71 Microcontrollers

APPLICATION NOTE

Introduction

This application note explains the main features of Integrity Check Monitor (ICM) module on Atmel®| SMART SAM V71/V70/E70/S70 family devices. For more details about ICM module, refer the complete datasheet of the specific device datasheet.

Features

In addition to the main features of ICM, this application note explains the steps to use ICM module to perform the following tasks:

- Integrity check on Internal SRAM (Non Contiguous)
- Integrity check on Internal Flash (Contiguous)
- Hashing using ICM
- Security Access on ICM registers
- Active Monitoring using ICM

Table of Contents

1. Prerequisites

- **Hardware Prerequisites**
  - Atmel SAM V71 Xplained ULTRA Kit
  - Interfacing cables - one Micro-USB type-B cable
- **Software Prerequisites**
  - Atmel Studio 7.0 or later
  - ASF 3.28.1 or later
  - Tera Term or any other terminal emulator (communications) program

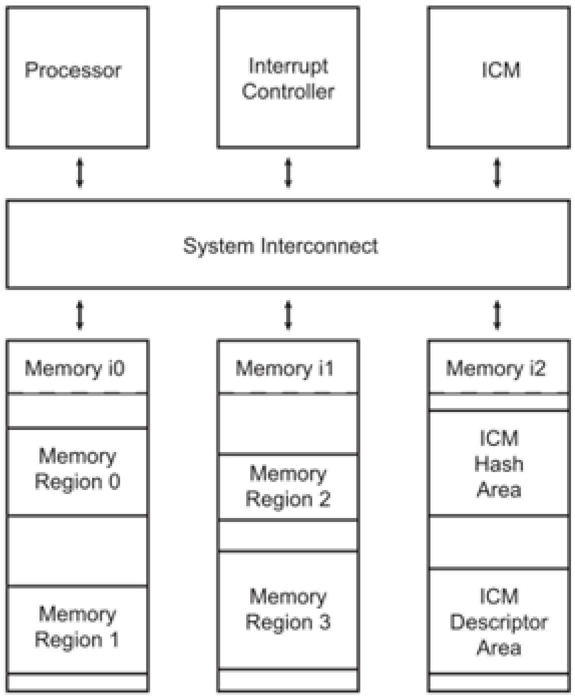
2. ICM Overview

The Integrity Check Monitor (ICM) is a DMA controller that performs hash calculation on multiple memory regions using the transfer descriptors located in memory (ICM Descriptor Area). The Hash function is based on the Secure Hash Algorithm (SHA). The ICM controller integrates the following two modes of operation:

1. Hash a list of memory regions and save the digests to memory (ICM Hash Area)
2. Active monitoring of the memory: In this mode, the hash function is evaluated and compared to the digest located at a predefined memory address (ICM Hash Area). If a mismatch occurs, an interrupt is raised.

The following figure illustrates an example of four-region monitoring. The Hash and Descriptor areas are located in Memory instance *i2* and the four regions are split in memory instances *i0* and *i1*.

Figure 2-1. Four Region Monitoring Example



The ICM SHA engine is compliant with the American FIPS (Federal Information Processing Standard) Publication 180-2 specification.

The following terms are concise definitions of the ICM concepts used throughout this document:

- Region - a partition of instruction or data memory space
- Region Descriptor - a data structure stored in memory, defining region attributes
- Context Registers - a set of ICM non-memory-mapped, internal registers which are automatically loaded, containing the attributes of the region being processed
- Main List - a list of region descriptors. Each element associates the start address of a region with a set of attributes
- Secondary List - a linked list defined on a per region basis that describes the memory layout of the region (when the region is non-contiguous)
- Hash Area - a predefined memory space where the region hash results (digest) are stored

The working of ICM related to Hashing a memory area or Active monitoring of memory region is explained in detail in the corresponding device datasheet. Refer the device datasheet for more information.

3. Setup

The overview of this section is as follows:

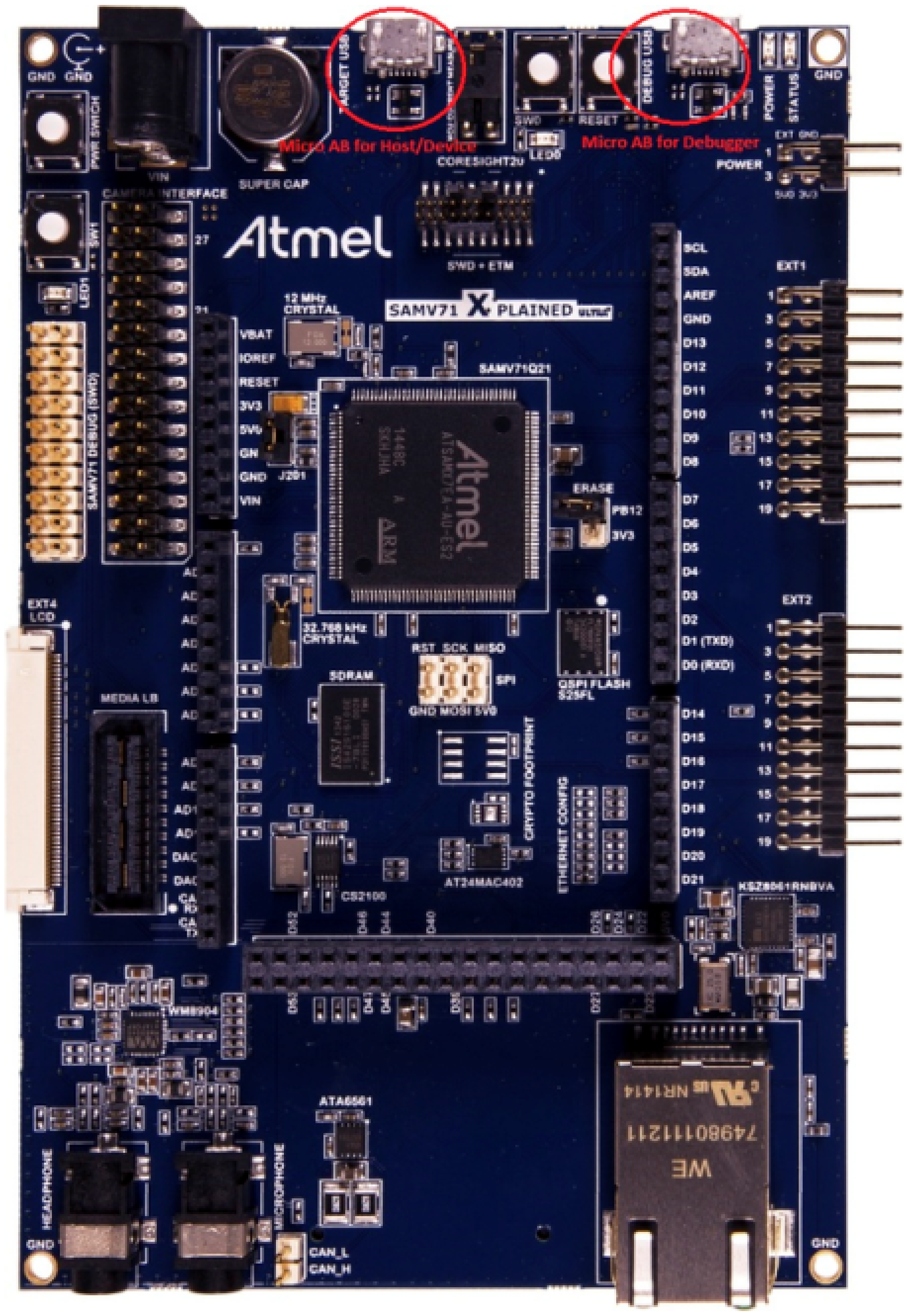
- Hardware Setup
- Software Setup

3.1. Hardware Setup

The SAM V71 Xplained Ultra kit is used to execute the example application. This evaluation kit allows connecting multiple external components using a wing connector. A wing board is a self-contained board that can be connected to the kit using a wing connector. The SAM V71 Xplained Ultra has two wing connector marked as EXT1 and EXT2.

There are two USB ports on the SAM V71 Xplained Ultra board - **DEBUG USB** and **TARGET USB**. For debugging/programming using the Embedded debugger EDBG, **DEBUG USB** port must be connected.

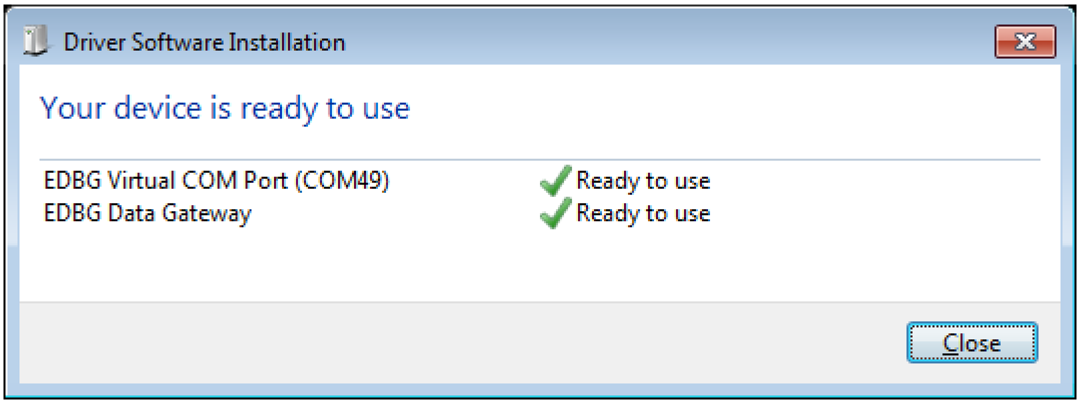
Figure 3-1. Micro-AB Connectors on SAM V71 Xplained Ultra



3.2. Software Setup

After the SAM V71 Xplained Ultra kit is connected to the PC, the Windows® Task bar will pop-up a message as shown in the following figure.

Figure 3-2. SAM V71 Xplained Ultra Driver Installation


Driver Software Installation

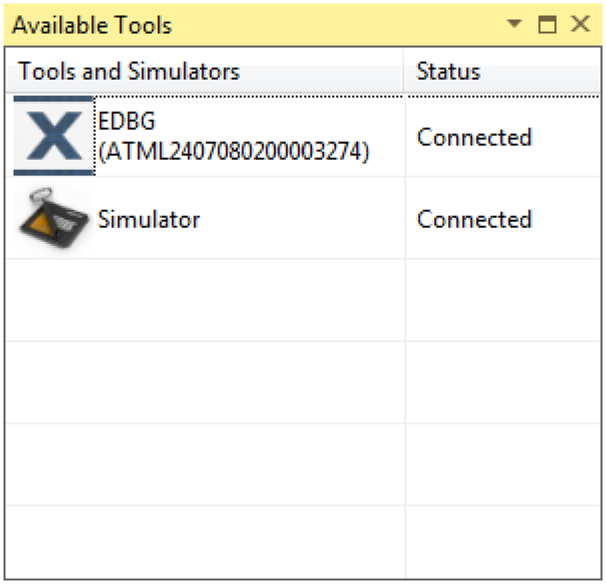
Your device is ready to use

EDBG Virtual COM Port (COM49)	✓ Ready to use
EDBG Data Gateway	✓ Ready to use

To ensure that the EDBG tool is detected in Atmel Studio,

1. Open Atmel Studio 7.0, Go to **View** → **Available Atmel Tools**. The EDBG should get listed in the tools as **EDBG** and the tool status should display as **Connected**. This indicates that the tool is communicating with Atmel Studio, as expected.

   Figure 3-3. EDBG under Available Atmel Tools


Available Tools

Tools and Simulators	Status
EDBG (ATML2407080200003274)	Connected
Simulator	Connected

2. If the tool is not displayed in **Available Atmel Tools**, disconnect the tool and reconnect again.
3. Right click the tool in the **Available Tools** list, click **Upgrade**. This will check if the firmware in the tool is up-to-date. Click the **Upgrade** button to upgrade the firmware of the tool to latest version. If you get **Upgrade Failed** error, power cycle the device and then try upgrading again.

4. ICM Features Demonstration

The following features are demonstrated in this application note.

1. The ICM performs manual monitoring of Internal SRAM (for both contiguous and non-contiguous memory)
2. The ICM performs manual monitoring of Internal Flash (for contiguous memory)
3. The ICM generates Hash using SHA engine, this can be used for verifying content
4. The ICM provides security access to ICM registers. Registers are not modified unintentionally.
5. The ICM provides active monitoring of Internal SRAM

The various features are explained along with relevant code snippets to demonstrate the features explained in the application note.

4.1. SRAM Integrity Check Example

The ICM can monitor contiguous and non-contiguous memory. The non-contiguous memory access is achieved by using secondary list access provided by the ICM. The contiguous memory monitoring is verified using a simply array as the main list to be monitored. Additional memories are added using secondary list to show case non-contagious memory monitoring. The example application monitors the arrays(memory) that are defined in the internal SRAM. When the data is modified, the digest mismatch interrupt is invoked to indicate which region that was modified by the user.

4.1.1. ICM Configuration

- ICM uses two regions - region0 and region1 in the ICM main list to be monitored
- region0 monitors message_sha array
- region1 monitors message_sha_sram array and also monitors using the secondary list message_sha_sram_2
- Compare mode set to true for both region0 and region1
- Enable and register callbacks to be triggered on Region digest mismatch for region0 and region1
- Non-Active Monitoring configuration, requires ICM to be enabled to start monitoring
- After Initial Configuration, the program modifies the following regions:
  – Main list of region1
  – Secondary list of region1
  – Main list of region0

4.1.2. Code Snippet – I (ICM Global variables – Commonly used across all examples)

```
/* Is set to 1 when a ICM interrupt happens */
static volatile uint32_t gs_icm_triggered = 0U;
static volatile uint32_t gs_icm_hash_invoked = 0U;

/* Test page start address. */
#define TEST_PAGE_ADDRESS (IFLASH_ADDR + IFLASH_SIZE - IFLASH_PAGE_SIZE * 4)
typedef unsigned long UL;

/* Hash area */
COMPILER_ALIGNED(128)
uint32_t output_sha[0x20];

#define MAX_TEST 4

/* Region descriptor in main list */
COMPILER_ALIGNED(64)
struct icm_region_descriptor_main_list reg_descriptor[2];
```

```
COMPILER_ALIGNED(4)
/* Region descriptor in secondary list */
struct icm_region_descriptor_sec_list reg_descriptor_sec;

#define TEST_ARRAY {0x80636261,
0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000, \
            0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000, \
            0x00000000,0x18000000}

/* Memory region area */ // SRAM
uint32_t message_sha[16] = TEST_ARRAY;

//define a memory in SRAM
/* Memory region area */ // SRAM
uint32_t message_sha_sram[16] = TEST_ARRAY;

//define a memory in SRAM
/* Memory region area */ // SRAM
uint32_t message_sha_sram_2[16] = TEST_ARRAY;
```

4.1.3. Code Snippet – II (ICM Configuration APIs and callback handlers – Commonly used across all examples)

```
 // ICM Initialization
static void do_icm_init(enum icm_algo ualgo)
{
    /* ICM configuration */
    struct icm_config icm_cfg;

    /* ICM initialization */
    icm_cfg.is_write_back= false;
    icm_cfg.is_dis_end_mon = false;
    icm_cfg.is_sec_list_branch = false;//secondary list supported.
    icm_cfg.bbc = 0;
    icm_cfg.is_auto_mode = false; // manual mode
    icm_cfg.is_dual_buf = false;
    icm_cfg.is_user_hash = false;
    icm_cfg.ualgo = ualgo;
    icm_cfg.hash_area_val = 0;
    icm_cfg.des_area_val = 0;
    icm_init(ICM, &icm_cfg);
}

// Region Descriptior Initialization.
static void set_region_descriptor_default(struct icm_region_descriptor_main_list *
reg_descriptor)
{
    reg_descriptor->start_addr = NULL; // To be defined.
    reg_descriptor->cfg.is_compare_mode = false; // To compare digest
    reg_descriptor->cfg.is_end_mon = false; //End of Monitoring
    // For more details on below parameters, refer the ICM header file and datasheet.
    reg_descriptor->cfg.is_wrap = false;
    reg_descriptor->cfg.reg_hash_int_en = false;
    reg_descriptor->cfg.dig_mis_int_en = false;
    reg_descriptor->cfg.bus_err_int_en = false;
    reg_descriptor->cfg.wrap_con_int_en = false;
    reg_descriptor->cfg.ebit_con_int_en = false;
    reg_descriptor->cfg.status_upt_con_int_en = false;
    reg_descriptor->cfg.is_pro_dly = false;
    reg_descriptor->cfg.mem_reg_val = 0;
    reg_descriptor->cfg.algo = ICM_SHA_1;
    reg_descriptor->tran_size = 0;
    reg_descriptor->next_addr = NULL;
}

// Callback handler on region digest mismatch
static void reg_dig_mismatch_handler(uint8_t reg_num)
{
    gs_icm_triggered = 0;
    if(reg_num == ICM_REGION_NUM_0)
        printf( " Memory region0 is modified \n\r");
    if(reg_num == ICM_REGION_NUM_1)
        printf( " Memory region1 is modified \n\r");
    gs_icm_triggered = 1;
}
```

4.1.4. Code Snippet – III (ICM SRAM Monitoring Test)

```
// ICM SRAM
static void run_icm_sram_test()
{
        // Perform default ICM Initialization.
        do_icm_init(ICM_SHA_1);

        /* Set region0 & 1 default descriptor values */
        //set default configuration first.
        set_region_descriptor_default(&reg_descriptor[0]);
        set_region_descriptor_default(&reg_descriptor[1]);
        reg_descriptor[0].start_addr = (uint32_t)message_sha; // Region 0 Monitoring address
        /* Set region1 descriptor in main list */
        reg_descriptor[1].start_addr = (uint32_t)message_sha_sram; // Region 1 Mon-itoring
address
        reg_descriptor[1].next_addr = &reg_descriptor_sec;

        /* Set region1 descriptor in secondary list */
        reg_descriptor_sec.start_addr = (uint32_t)message_sha_sram_2; // Region 1 Secondary
list
        reg_descriptor_sec.tran_size = 0;
        reg_descriptor_sec.next_addr = NULL; // The list ends here..

        /* Set region descriptor start address */
        icm_set_reg_des_addr(ICM, (uint32_t)&reg_descriptor[0]);

        /* Set hash area start address */
        // hashed data will be stored here and compared on digest mismatch interrupt
        icm_set_hash_area_addr(ICM, (uint32_t)output_sha);

        /* Enable ICM */
        icm_enable(ICM);
        delay_ms(200);
        /* Set callback function for digest mismatch interrupt handler */
        icm_set_callback(ICM, reg_dig_mismatch_handler, ICM_REGION_NUM_0,
        ICM_INTERRUPT_RDM, 1);
        icm_set_callback(ICM, reg_dig_mismatch_handler, ICM_REGION_NUM_1,
        ICM_INTERRUPT_RDM, 1);
        /* Set region monitoring mode to compare mode */
        reg_descriptor[0].cfg.is_compare_mode = true;
        reg_descriptor[1].cfg.is_compare_mode = true;
        /* Modify memory region value */
        message_sha_sram[0] = 0x8063626; //region 1 modified.
        /* Enable ICM to identify change in memory modified (not automatic) */
        icm_enable(ICM);
        delay_ms(1000);
        /* Modify memory region value */
        message_sha_sram[0] = 0x80636261;
        message_sha_sram_2[0] = 0x8063626;  //region 1 modified.
        /* Enable ICM */
        icm_enable(ICM);
        delay_ms(1000);
        /* Modify memory region value */
        message_sha_sram_2[0] = 0x80636261;
        message_sha[0] = 0x8063626; //region 0 modified.
        /* Enable ICM */
        icm_enable(ICM);
}
```

```
/**
 * \brief ICM Features Test
 */
int main(void)
{
    const usart_serial_options_t usart_serial_options = {
        .baudrate   = CONF_TEST_BAUDRATE,
#ifdef CONF_UART_CHAR_LENGTH
        .charlength = CONF_UART_CHAR_LENGTH,
#endif
        .paritytype = CONF_TEST_PARITY,
#ifdef CONF_UART_STOP_BITS
```

```
        .stopbits = CONF_UART_STOP_BITS,
#endif
    };

    sysclk_init();
    board_init();

    sysclk_enable_peripheral_clock(CONSOLE_UART_ID);
    stdio_serial_init(CONF_TEST_USART, &usart_serial_options);
    run_icm_sram_test(); // Based on the test, use the corresponding method to test.
    while(1);
}
```

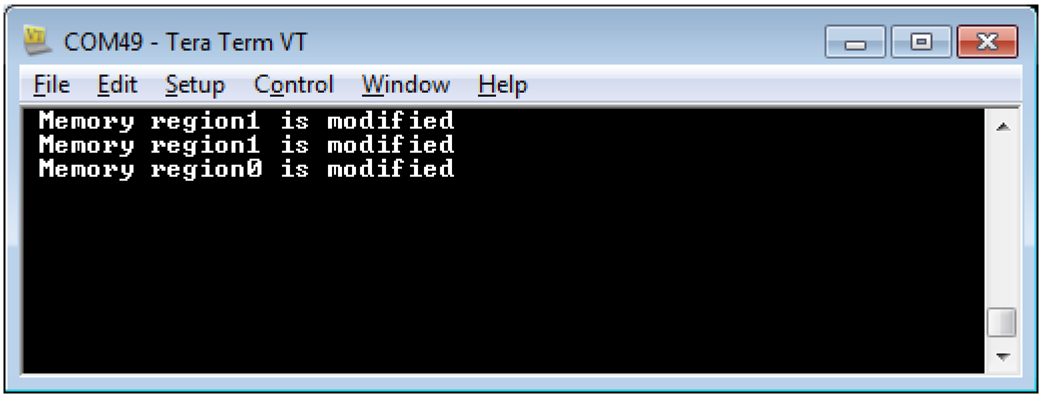
.

4.1.5. Output

After the initial configuration

- When the message_sha_sram array is modified, the program triggers the message “Memory region1 is modified” (main list of region1)
- When the message_sha_sram_2 array is modified, the program triggers the message “Memory region1 is modified” (secondary list of region1)
- When the message_sha array is modified, the program triggers the message “Memory region0 is modified” (main list of region0)

Figure 4-1. Output



```
Memory region1 is modified
Memory region1 is modified
Memory region0 is modified
```

4.2. Flash Integrity Check Example

The ICM can monitor both internal SRAM and internal Flash. The Flash is accessed using the available Flash APIs. The example uses the Flash address location TEST_PAGE_ADDRESS (symbolic constant, storing flash address in code), which is configurable. The program copies a predefined array into the flash location and the contents of the array are monitored by ICM. When the flash location is updated, the corresponding digest mismatch interrupt is invoked to indicate the region modified by the user.

4.2.1. ICM Configuration

- ICM uses the region region0 in the ICM main list to be monitored
- region0 monitors contents of TEST_PAGE_ADDRESS
- Compare mode set to true for region0
- Enable and register callbacks to be triggered on Region mismatch for region0
- Non Active Monitoring configuration requires ICM to be enabled to start monitoring the regions
- After Initial Configuration, the program modifies main list of region0

4.2.2. Code Snippet – I (Flash Initialization and Writing Flash with default content)

```
static void run_icm_flash_test(void)
{

    /* ICM configuration */
    struct icm_config icm_cfg;

    uint32_t ul_test_page_addr = TEST_PAGE_ADDRESS;
    uint32_t *pul_test_page = (uint32_t *) ul_test_page_addr;
    uint32_t ul_rc;
    uint32_t ul_idx;


    /* Initialize flash: 6 wait states for flash writing. */
    ul_rc = flash_init(FLASH_ACCESS_MODE_128, 6);
    if (ul_rc != FLASH_RC_OK) {
        printf("-F- Initialization error %lu\n\r", (UL)ul_rc);
        return ;
    }

    /* Unlock page */
    printf("-I- Unlocking test page: 0x%08lu\r\n", ul_test_page_addr);
    ul_rc = flash_unlock(ul_test_page_addr,
            ul_test_page_addr + IFLASH_PAGE_SIZE - 1, 0, 0);
    if (ul_rc != FLASH_RC_OK) {
        printf("-F- Unlock error %lu\n\r", (UL)ul_rc);
        return ;
    }

    ul_rc = flash_erase_sector(ul_test_page_addr);
    if (ul_rc != FLASH_RC_OK) {
        printf("-F- Flash programming error %lu\n\r", (UL)ul_rc);
        return ;
    }

    ul_rc = flash_write(ul_test_page_addr, message_sha,
            64, 0);

    if (ul_rc != FLASH_RC_OK) {
        printf("-F- Flash programming error %lu\n\r", (UL)ul_rc);
        return ;
    }

    /* Validate page */
    printf("-I- Checking page/flash write contents are successful ");
    for (ul_idx = 0; ul_idx < 16; ul_idx++) {
        printf(".");
        if (pul_test_page[ul_idx] != message_sha[ul_idx]) {
            printf("\n\r-F- data error at %d \n\r", (int) ul_idx);
            return ;
        }
    }
    printf("OK\n\r");
```

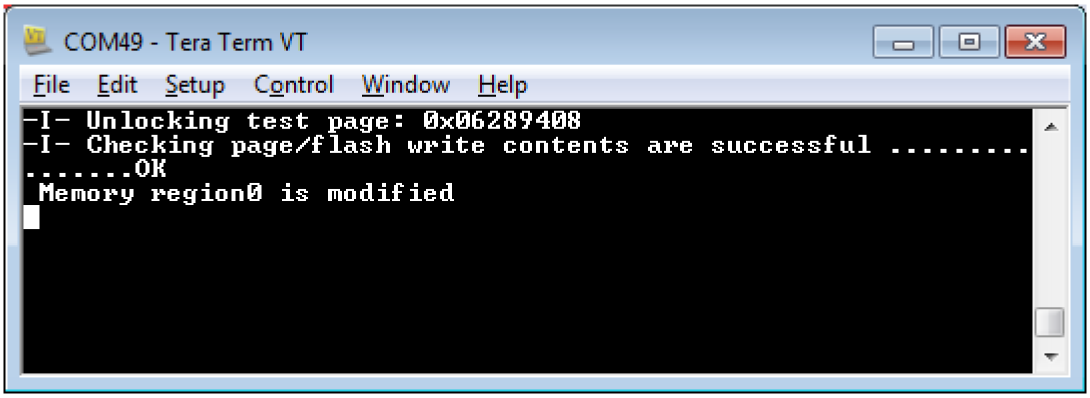
4.2.3. Code Snippet-II (Continued...ICM Configuration and Flash Monitoring)

```
/* ICM initialization */
    do_icm_init(ICM_SHA_1);

    /* Set region0 descriptor */

    /* Set region descriptor */
    set_region_descriptor_default(&reg_descriptor[0]);
    reg_descriptor[0].start_addr = (uint32_t)TEST_PAGE_ADDRESS; // Memory to be hashed
    reg_descriptor[0].cfg.is_end_mon = true;

    /* Set region descriptor start address */
    icm_set_reg_des_addr(ICM, (uint32_t)&reg_descriptor[0]);

    /* Set hash area start address */
    icm_set_hash_area_addr(ICM, (uint32_t)output_sha);

    /* Enable ICM */
    icm_enable(ICM);
    delay_ms(200);

    /* Set region monitoring mode to compare mode */
    reg_descriptor[0].cfg.is_compare_mode = true;

    /* Set callback function for digest mismatch interrupt handler */
    icm_set_callback(ICM, reg_dig_mismatch_handler, ICM_REGION_NUM_0,
    ICM_INTERRUPT_RDM, 1);

    /* Modify memory region value */
    message_sha[0] = 0x8063626;

    ul_rc = flash_write(ul_test_page_addr, message_sha,    64, 0);

    /* Enable ICM */
    icm_enable(ICM);
}
```

4.2.4. Output

After the initial configuration, the program modifies the contents of TEST_PAGE_ADDRESS (Flash location). This triggers the Message **Memory region0 is modified** (main list of region0).

Figure 4-2. Output



```
-I- Unlocking test page: 0x06289408
-I- Checking page/flash write contents are successful .........
........OK
 Memory region0 is modified
```

4.3. Hash Generation Example

The SHA (Secure Hash Algorithm) is one of the various cryptographic hash functions. A cryptographic hash is like a signature for a text or a data file. For example, SHA-256 algorithm generates an almost-unique, fixed size 256-bit (32-byte) hash. Hash is a one way function – it cannot be decrypted. Hence, this is suitable for password validation, challenge hash authentication, anti-tamper, digital signatures. The ICM contains SHA-1, SHA-224 and SHA-256 engines. The example shows the Hash generation for SHA-224 and SHA-256.

After specifying the region and algorithm to be hashed, the hash value generated by ICM module is validated with a known set of Hash value for the specific memory region. The example displays whether Hash Generated by ICM matches with predefined Hash value set inside the program for the specific memory range.

Note: We can use online SHA tools to generate hash value for a stream.

The example uses the Hash value available in the datasheet under the section, **Message Digest Example**.

4.3.1. ICM Configuration

- ICM uses region region0 in the ICM main list to be monitored

- Enable and register callbacks to be triggered on Hash completion for Region0
- After Initial Configuration, the program performs hash for SHA-224 and SHA-256 and validates the result with known set of values on SHA-224 and SHA-256.

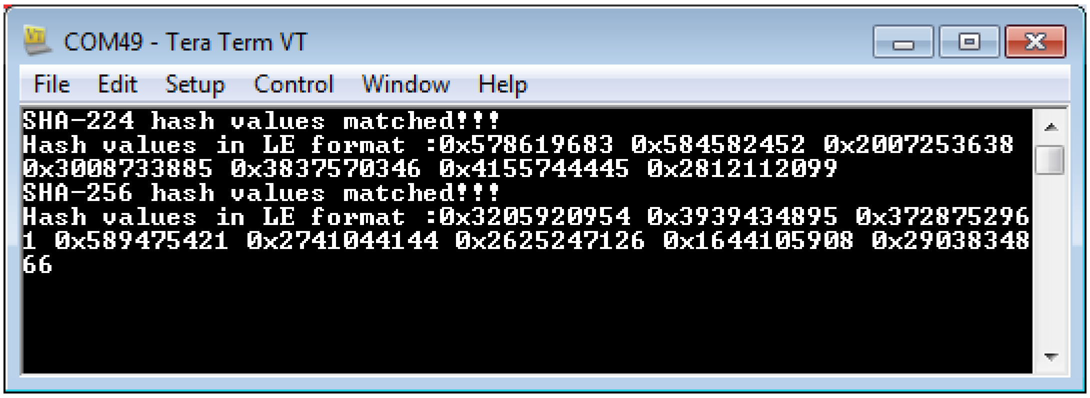
4.3.2. Code Snippet – ICM Hash Generation and Verification

```
static void run_icm_hash_test(void)
{
    //perform default ICM initialization.
    do_icm_init(ICM_SHA_1);
    /* Set region descriptor */
    set_region_descriptor_default(&reg_descriptor[0]);
    reg_descriptor[0].start_addr = (uint32_t)message_sha; // Memory to be hashed
    reg_descriptor[0].cfg.is_end_mon = true;

    /* Set region descriptor start addres */
    icm_set_reg_des_addr(ICM, (uint32_t)&reg_descriptor);

    /* Set hash area start addres */
    icm_set_hash_area_addr(ICM, (uint32_t)output_sha);

    /* Set callback function for region hash complete interrupt handler */
    icm_set_callback(ICM, reg_hash_complete_handler,
            ICM_REGION_NUM_0, ICM_INTERRUPT_RHC, true);

    //hash test completed..
    do_hash_test(ICM_SHA_224);
    do_hash_test(ICM_SHA_256);
}

static void do_hash_test(enum icm_algo algo)
{
    const char *temp[]={"SHA-256","SHA-224"};
    int iMax= (algo== ICM_SHA_256)?8:7;
    gs_icm_hash_invoked=0;
    reg_descriptor[0].cfg.algo = algo;
    /* Enable ICM */
    icm_enable(ICM);
    while(!gs_icm_hash_invoked); // wait till hash complete.
    if(!gs_icm_triggered)
    {
        printf("%s hash values mismatch!!!\r\n", (algo == ICM_SHA_256)?temp[0]:temp[1]);
    }
    else
        printf("%s hash values matched!!!\r\n", (algo == ICM_SHA_256)?temp[0]:temp[1]);

        printf("Hash values in LE format :");

        for( int i=0;i<iMax;i++)
            printf("0x%lu ", output_sha[i]);
        printf("\r\n");

}
static void reg_hash_complete_handler(uint8_t reg_num)
{
    UNUSED(reg_num);
    gs_icm_triggered=0;
    if((output_sha[0] == 0x227D0923) &&
            (output_sha[1] == 0x22D80534) &&
            (output_sha[2] == 0x77A44286) &&
            (output_sha[3] == 0xB355A2BD) &&
            (output_sha[4] == 0xE4BCAD2A) &&
            (output_sha[5] == 0xF7B3A0BD) &&
            (output_sha[6] == 0xA79D6CE3)) {
        gs_icm_triggered = 1;
    }
    if((output_sha[0] == 0xBF1678BA) &&
            (output_sha[1] == 0xEACF018F) &&
            (output_sha[2] == 0xDE404141) &&
            (output_sha[3] == 0x2322AE5D) &&
            (output_sha[4] == 0xA36103B0) &&
            (output_sha[5] == 0x9C7A1796) &&
            (output_sha[6] == 0x61FF10B4) &&
            (output_sha[7] == 0xAD1500F2)) {
        gs_icm_triggered = 1;
    }
    gs_icm_hash_invoked=1;
}
```

4.3.3. Output

After the initial configuration, the program validates whether the generated hash value matches with predefined values for region0. The program also prints the Hash value of 224-bit/256-bit data in Little Endian (LE) format.

Figure 4-3. Output



```
SHA-224 hash values matched!!!
Hash values in LE format :0x578619683 0x584582452 0x2007253638
0x3008733885 0x3837570346 0x4155744445 0x2812112099
SHA-256 hash values matched!!!
Hash values in LE format :0x3205920954 0x3939434895 0x372875296
1 0x589475421 0x2741044144 0x2625247126 0x1644105908 0x29038348
66
```

4.4. Security Feature Example

When an undefined register is accessed, the URAD bit in the Interrupt Status Register (ICM_ISR) is set if it is unmasked. This is used when unidentified access to ICM registers are triggered to modify ICM configuration after ICM module is enabled.

4.4.1. ICM Configuration

- ICM uses region0 in the ICM main list to be monitored
- The region0 monitors message_sha array (Contiguous)
- Compare mode set to true for region0
- Enable and register callbacks to be triggered on undefined register access
- After Initial Configuration, the program modifies the Hash engine used for SHA-256. This triggers the callback on undefined register access, indicating security issue

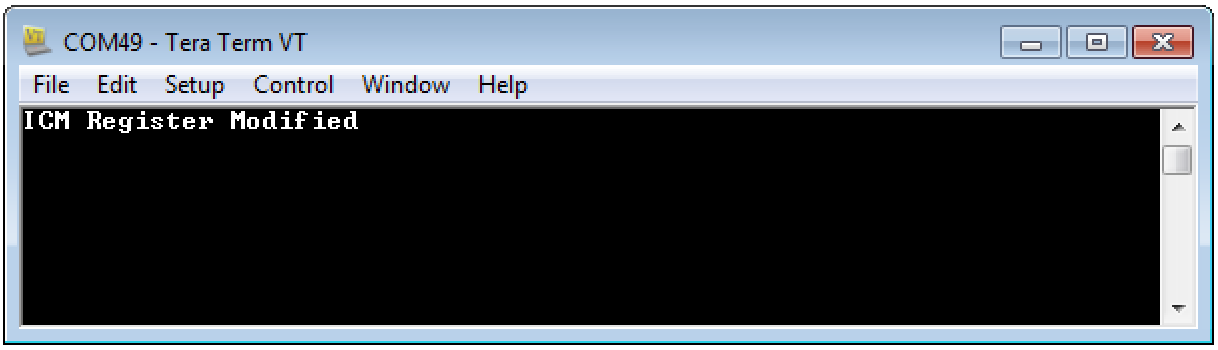
4.4.2. Code Snippet- ICM Security Access Example

```
static void run_icm_security_test(void)
{
        /* ICM configuration */
        struct icm_config icm_cfg;
        /* ICM initialization */
        icm_cfg.is_write_back= false;
        icm_cfg.is_dis_end_mon = false;
        icm_cfg.is_sec_list_branch = false;
        icm_cfg.bbc = 0;
        icm_cfg.is_auto_mode = true;
        icm_cfg.is_dual_buf = false;
        icm_cfg.is_user_hash = false;
        icm_cfg.ualgo = ICM_SHA_1;
        icm_cfg.hash_area_val = 0;
        icm_cfg.des_area_val = 0;
        icm_init(ICM, &icm_cfg);

        /* Set region0 descriptor */
        set_region_descriptor_default(&reg_descriptor[0]);
        reg_descriptor[0].start_addr = (uint32_t)message_sha;
        /* Set region monitoring mode to compare mode and use corresponding configurations to
set ICM in auto mode */
        reg_descriptor[0].cfg.is_compare_mode = true;
        reg_descriptor[0].cfg.is_wrap = true;
        reg_descriptor[0].cfg.is_end_mon = false;
        reg_descriptor[0].next_addr = NULL;

        /* Set region descriptor start address */
        icm_set_reg_des_addr(ICM, (uint32_t)&reg_descriptor[0]);

        /* Set hash area start address */
        icm_set_hash_area_addr(ICM, (uint32_t)output_sha);

        /* Set callback function for register modified interrupt handler */
        icm_set_callback(ICM, reg_urad_handler,    ICM_REGION_NUM_0, ICM_INTERRUPT_URAD, 1);

        /* Enable ICM */
        icm_enable(ICM);
        delay_ms(2000);
        /* Modify Register setting */
        icm_set_algo(ICM,ICM_CFG_UALGO_SHA256);
        delay_ms(2000);
}

static void reg_urad_handler(uint8_t reg_num)
{
    if(reg_num == ICM_REGION_NUM_0)
    {
           printf("ICM Register Modified \r\n");
        icm_reset(ICM);
    }
}
```

4.4.3. Output

After the initial configuration, the program modifies contents of ICM CFG register. This triggers the Message `ICM Register Modified`.

Figure 4-4. Output



```
ICM Register Modified
```

4.5. Auto Monitoring Feature Example

The ICM can perform Active monitoring of contiguous and non-contiguous memory.

The ASCD bit of the ICM_CFG register is used to activate the ICM Automatic Mode. When ICM_CFG.ASCD is set, the ICM performs the following actions:

- The ICM controller passes through the Main List once with CDWBN bit in the context register at 0 (i.e. Write Back activated) and EOM bit in context register at 0.
- When WRAP = 1 in ICM_RCFG, the ICM controller enters active monitoring with CDWBN bit in context register now set and EOM bit in context register cleared. The bits CDWBN and EOM in ICM_RCFG have no effect.

The example monitors the arrays defined in the internal SRAM. When the data is modified, the digest mismatch interrupt is invoked to indicate the region that was modified by the user.

4.5.1. ICM Configuration

- ICM uses region0 for monitoring
- The region0 monitors message_sha array defined in SRAM
- Compare mode set to true for region0
- Enable and register callbacks to be triggered on Region mismatch for region0
- Automatic configuration is set on ICM
- ICM is enabled
- After Initial Configuration, the program modifies the following region:
  - Main list of region0

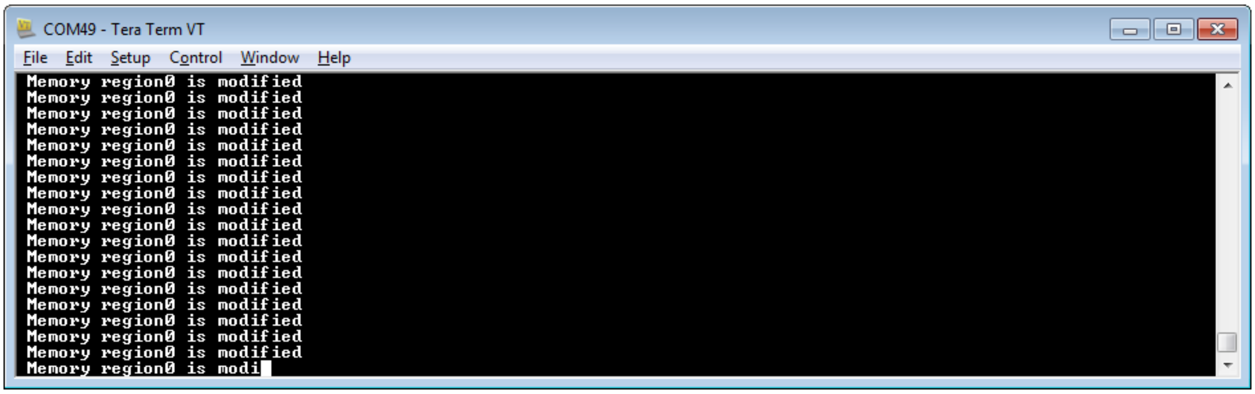
4.5.2. Code Snippet – ICM Active Monitoring

```
static void run_icm_auto_test(void)
{
        /* ICM configuration */
        struct icm_config icm_cfg;
        /* ICM initialization */
        icm_cfg.is_write_back= false;
        icm_cfg.is_dis_end_mon = false;
        icm_cfg.is_sec_list_branch = false;
        icm_cfg.bbc = 0;
        icm_cfg.is_auto_mode = true;
        icm_cfg.is_dual_buf = false;
        icm_cfg.is_user_hash = false;
        icm_cfg.ualgo = ICM_SHA_1;
        icm_cfg.hash_area_val = 0;
        icm_cfg.des_area_val = 0;
        icm_init(ICM, &icm_cfg);


        /* Set region0 descriptor */
        set_region_descriptor_default(&reg_descriptor[0]);
        reg_descriptor[0].start_addr = (uint32_t)message_sha;
                /* Set region monitoring mode to compare mode */
        reg_descriptor[0].cfg.is_compare_mode = true;
        reg_descriptor[0].cfg.is_wrap = true;
        reg_descriptor[0].cfg.is_end_mon = false;
        reg_descriptor[0].next_addr = NULL;

        /* Set region descriptor start address */
        icm_set_reg_des_addr(ICM, (uint32_t)&reg_descriptor[0]);

        /* Set hash area start address */
        icm_set_hash_area_addr(ICM, (uint32_t)output_sha);

        /* Set callback function for digest mismatch and register modified interrupt handler
*/
        icm_set_callback(ICM, reg_dig_mismatch_handler, ICM_REGION_NUM_0,ICM_INTERRUPT_RDM,
1);

        /* Enable ICM */
        icm_enable(ICM);

        message_sha[0] = 0x8063626;
        delay_ms(2000);
}
```

4.5.3. Output

After initial configuration, the program modifies contents of the region0. This triggers the Message "Memory region0 is modified". Since this mode is used in Active Monitoring, the same message is displayed repeatedly.

Figure 4-5. Output



```
Memory region0 is modified
Memory region0 is modified
Memory region0 is modified
Memory region0 is modified
Memory region0 is modified
Memory region0 is modified
Memory region0 is modified
Memory region0 is modified
Memory region0 is modified
Memory region0 is modified
Memory region0 is modified
Memory region0 is modified
Memory region0 is modified
Memory region0 is modified
Memory region0 is modified
Memory region0 is modified
Memory region0 is modified
Memory region0 is modified
Memory region0 is modi
```

5. References

5.1. Atmel SAM S7/ E7/ V7 Datasheets

The complete datasheet for each device contains block diagrams of the peripherals and details about implementing firmware for the device. It also contains the electrical specifications and expected characteristics of the device.

The complete datasheets are available in the Documents section of the product pages

- SAM S70: http://www.atmel.com/products/microcontrollers/arm/sam-s.aspx?tab=overview#sams70
- SAM E70: http://www.atmel.com/products/microcontrollers/arm/sam-e.aspx#same70
- SAM V70: http://www.atmel.com/products/microcontrollers/arm/sam-v-mcus.aspx#samv70
- SAM V71: http://www.atmel.com/products/microcontrollers/arm/sam-v-mcus.aspx#samv71

5.2. Reference Application Notes

1. Getting Started with SAM V71 Microcontrollers
2. **Atmel AT12874**: Getting Started with SAM S70/E70

These application notes can be downloaded from http://www.atmel.com/products/microcontrollers/arm/default.aspx?tab=documents.

5.3. ASF User Manual

1. **Atmel AVR4029**: Atmel Software Framework - Getting Started
2. **Atmel AVR4030**: AVR Software Framework - Reference Manual

These documents can be downloaded from http://www.atmel.com/tools/avrsoftwareframework.aspx?tab=documents.

6. Revision History

Doc Rev.	Date	Comments
42703A	4/2016	Initial document release.