

**Core self-test library compliant with IEC 60730, IEC 60335 UL 60730,
UL 1998 documentation**

Invariable Memory Test

Document revision history

Date	Author	Version	Notes
11/2015	Jozef Sedlak	0.1	Initial version
11/2015	Jozef Sedlak	1.0	Version for certification
10/2016	Jozef Sedlak	1.1	NXP
11/2018	Jozef Sedlak	3.0	Release with new supported compilers

1	INVARIABLE MEMORY TEST ARCHITECTURE	4
2	INVARIABLE MEMORY TEST IN COMPLIANCE WITH IEC/UL STANDARDS	4
3	INVARIABLE MEMORY TEST IMPLEMENTATION	5
3.1	COMPUTING OF CRC VALUE IN LINKING PHASE OF APPLICATION	5
3.2	TEST PERFORMED ONCE AFTER MICROCONTROLLER RESET	7
3.3	RUNTIME TEST.....	8
3.4	IEC60730B_CM4_CM7_FLASH_HWTEST()	9
3.5	IEC60730B_CM4_CM7_FLASH_SWTEST().....	10
3.6	IEC60730B_CM4_CM7_FLASH_SWTEST_32().....	11
3.7	IEC60730B_CM4_CM7_FLASH_HWTEST_DCP()	12
4	INVARIABLE MEMORY TEST MODULE TEST CONCEPT	13
5	INVARIABLE MEMORY TEST VALIDATION	13

1 Invariable Memory Test Architecture

The invariable memory on the CM4 and CM7 based Kinetis microcontrollers is the on-chip flash. The principle of the invariable memory test is to check whether there is a change in memory content during the application run. Several checksum methods can be used for this purpose. The checksum is an algorithm that calculates a signature of data placed in tested memory. The signature of this memory block is then periodically calculated and compared with the original signature.

The signature for assigned memory is calculated in the linking phase of an application. The signature must be saved in the invariable memory, however in a different area than that which the checksum is calculated for. In runtime and after reset, the same algorithm must be implemented in the application to calculate the checksum. Results are compared and if not equal, a safety error state occurs.

The algorithm that calculates a checksum parameter (signature) by IAR linker must be set to use the 16-bit CRC polynomial (0x1021) to generate a CRC code for error detection. The same algorithm is implemented in HW–CRC module. When DCP module is used for CRC calculation, then 32-bit CRC must be set.

If the HW CRC module, or DCP module cannot be used for the invariable memory test, there is a possibility to use the software version of test. It has same functionality, does not require hardware support, but it is much slower.

2 Invariable Memory Test in compliance with IEC/UL standards

The performed overload test fulfils safety requirements according to IEC 60730-1, IEC 60335, UL 60730, and UL 1998 standards as described in the following table:

Table 1. CPU Registers Test in Compliance with the IEC and UL Standards

Test	Component	Fault/Error	SW / HW Class	Acceptable Measures
Invariable memory	4.1 – Invariable memory	All single bit faults	B/R.1	Periodic modified checksum

3 Invariable Memory Test Implementation

Test functions for flash memory are placed in IEC60730_B_CM4_CM7_flash.S and IEC60730_B_CM4_CM7_flash_DCP.c. The header file with definitions and function prototypes is IEC60730_B_CM4_CM7_flash.h. IEC60730_B_CM4_CM7.h and asm_mac_common.h are files that need to be placed in application as well. Functions defined in IEC60730_B_CM4_CM7_flash.S:

- IEC60730B_CM4_CM7_Flash_HWTest()
- IEC60730B_CM4_CM7_Flash_SWTest()
- IEC60730B_CM4_CM7_Flash_SWTest_32()

Functions defined in IEC60730_B_CM4_CM7_flash_DCP.:

- IEC60730B_CM7_Flash_HWTest_DCP()

3.1 Computing of CRC value in linking phase of application

Following example is valid for IAR IDE. Implementation in Keil IDE and MCUXpresso IDE differ in post-build phase.

The checksum of a memory block must be calculated before it has been written into flash memory. The result of the CRC calculation must be stored in the flash, but in different area as what the checksum was calculated for. A good method is to define a small block in flash (ROM) memory where the result of checksum will be stored. To do this, you must edit a linker configuration file. The path to the linker configuration file can be found in: Project > Options > Linker > Config. File name extension is .icf. For this example, CHECKSUM block, with .checksum section is defined.

```
define symbol __FlashCRC_start__ = 0x3fff0;  
define symbol __FlashCRC_end__ = 0x3fff;  
define region CRC_region = mem:[from __FlashCRC_start__ to __FlashCRC_end__];  
define block CHECKSUM { section .checksum };  
place in CRC_region { block CHECKSUM };
```

The input parameters for CRC calculation must be set up in the linker option tabs: Project > Options > Linker. There are two options for setting up calculation parameters. The first option is used to calculate the checksum for one block of memory in your application. The parameters are filled in Checksum subtab. For this example, the start and end addresses are: 0x510 and 0x3000. Unused memory will be filled with 0xFF. The checksum will be stored with 16 bits. The checksum algorithm is CRC16 with the standard 0x1021 polynomial. Initial seed is 0. Block size for particular calculation is 8-bit. The variable for the result is __checksum.

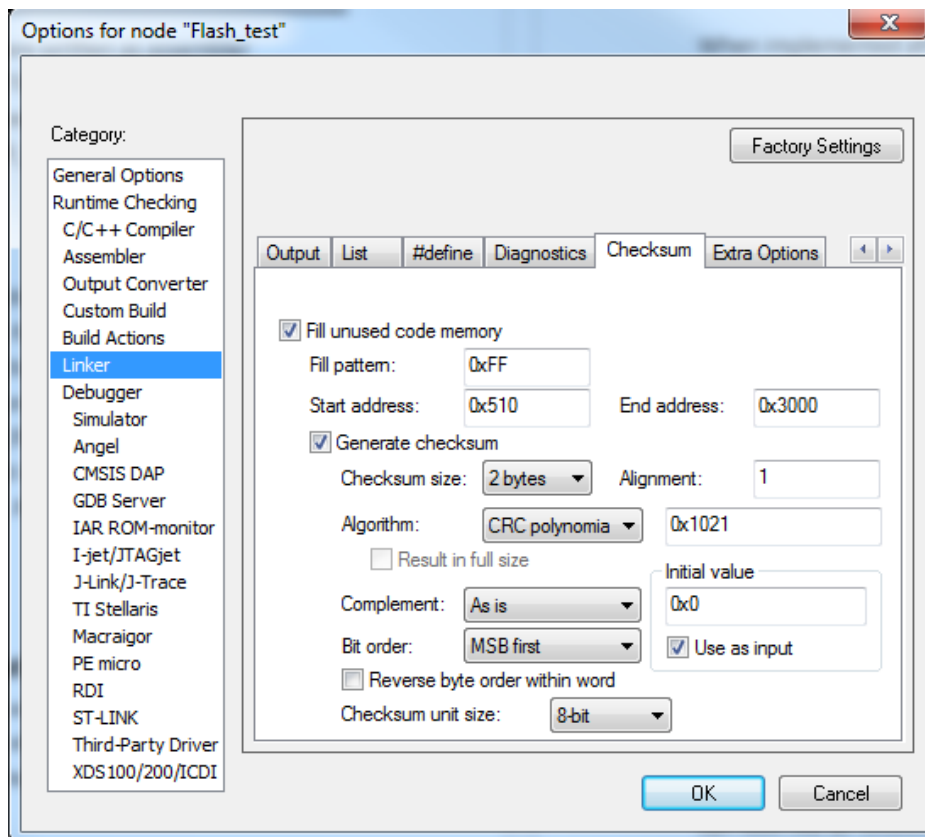


Figure 1. Checksum settings for linker

The constant variable name (`__checksum`)

must be written into Project > Options > Linker > Input > Keep symbols.

The following lines must be placed into the source code, to have `__checksum` variable available in application.

```
#pragma section = ".checksum"
```

```
#pragma location = ".checksum"
```

```
extern unsigned short const __checksum;
```

If CRC value calculation for more memory blocks is needed, the following approach must be used. There must be enough space in the block defined in the linker configuration file. For this example, the parameters for the calculations are the same as in the previous example and the addresses of blocks are: (0x510 – 0x610, 0x620 – 0x720, 0x730 – 0x830). The variables are as follows: (`__checksum_first`, `__checksum_second`, `__checksum_third`). In this case linker command lines directives are used: Project > Options > Linker > Extra Options. Allow the use of command line options and enter the following lines there. Note that options in the Checksum subtab must be unchecked.

```
—fill 0xFF;0x510-0x610
```

```
—checksum __checksum_first:2,crc16,0x0;0x510-0x610
```

```
—place_holder __checksum_first,2,.checksum,4
```

```
—fill 0xFF;0x620-0x720
```

```
—checksum __checksum_second:2,crc16,0x0;0x620-0x720
```

```
—place_holder __checksum_second,2,.checksum,4
```

```
—fill 0xFF;0x730-0x830  
—checksum __checksum_third:2,crc16,0x0;0x730-0x830  
—place_holder __checksum_third,2,.checksum,4
```

Project > Options > Linker > Input

Write:

```
__checksum_first  
__checksum_second  
__checksum_third
```

in the Keep symbols block.

The following lines must be entered in the source code, such that `__checksum_first`, `__checksum_second` and `__checksum_third` variables are available in the application.

```
#pragma section = ".checksum"  
#pragma location = ".checksum"  
extern unsigned short const __checksum_first;  
extern unsigned short const __checksum_second;  
extern unsigned short const __checksum_third;
```

3.2 Test performed once after microcontroller reset

When implemented after reset, or when there is no restriction on the execution time, function call can be as follows: ¹

```
#include "IEC60730_B_CM4_CM7.h"  
#pragma section = ".checksum"  
#pragma location = ".checksum"  
extern unsigned short const __checksum;
```

```
if((uint16_t)__checksum != IEC60730B_CM4_CM7_Flash_HWTest(start_address, size, CRC_BASE,  
start_seed))  
SafetyError();
```

Where:

`__checksum` - the constant variable with CRC value computed in linking phase of application.
`start_address` - the initial address of memory block to be tested.
`size` - the size of memory block to be tested. (first address – end address + 1)
`CRC_BASE` - base address of the CRC HW module
`start_seed` - the start condition seed. For the used algorithm it needs to be 0.

3.3 Runtime test

In application runtime with limited time for execution, the CRC is computed in sequence. It means that input parameters have different meanings in comparison with calling after reset. The implementation is as follows: ¹

```
#include "IEC60730_B_CM4_CM7.h"
#pragma section = ".checksum"
#pragma location = ".checksum"
extern unsigned short const __checksum;

flash_crc.part_crc = IEC60730B_CM4_CM7_Flash_HWTest(flash_crc.actual_address, \
flash_crc.block_size, CRC_BASE, flash_crc.part_crc);
if(IEC60730B_ST_FLASH_FAIL == Flash_test_handling(__checksum, &flash_crc))
SafetyError();
```

Where:

__checksum	- the constant variable with CRC value computed in linking phase of application.
flash_crc.part_crc	- particular CRC result and seed parameter for next iteration.
flash_crc.actual_address	- the actual address of memory block to be tested.
CRC_BASE	- base address of the CRC module
flash_crc.block_size	- the size of memory block to be tested.

The handling of the function must be carried out by the application developer. When the checksum of a block is calculated in more iterations, the result from the first iteration (function call) is the seed value for the next function call. After the last part of memory is processed with the test function, the result is the final checksum of the whole tested memory block.

3.4 IEC60730B_CM4_CM7_Flash_HWTest()

This function calculates the 16-bit CRC polynomial (0x1021) with the use of the CRC module.

Function prototype:

```
unsigned short IEC60730B_CM4_CM7_Flash_HWTest(const unsigned int startAdd, \  
const unsigned int size, const unsigned int moduleAddress, const unsigned short crcVal);
```

Function inputs:

startAdd – first address of memory block to be tested
size – size of block to be tested
moduleAddress – base address of the CRC module
crcVal – input seed value for calculation (for the first iteration it is 0, for the next iterations it is the result from the previous function call)

Function output:

16-bit checksum of the block of memory defined by input parameters

Function performance:

Function size is 44 bytes.²

Function duration depends on the defined block size. Several examples are shown in table 2.³

Table 2. Duration of IEC60730B_CM4_CM7_Flash_HWTest() in dependence of tested block size

Block size (Bytes)	Clock cycles	Execution time (approximately)
0x10	180	2.25 µs
0x20	311	3.88 µs
0x50	710	8.87 µs

Calling restrictions:

The function cannot be interrupted with function that changes the content or setup of HW CRC module.

3.5 IEC60730B_CM4_CM7_Flash_SWTest()

This function calculates the 16-bit CRC polynomial (0x1021) without using the hardware.

Function prototype:

*unsigned short IEC60730B_CM4_CM7_Flash_SWTest(const unsigned int startAdd, *
const unsigned int size, unsigned int moduleAddress, const unsigned short crcVal);

Function inputs:

startAdd – first address of memory block to be tested
size – size of block to be tested
module_address - this parameter has no influence, its purpose is only better compatibility with the *IEC60730B_CM4_CM7_Flash_HWTest* function
crcVal – input seed value for calculation (for the first iteration is 0, for next iterations it is the result from the previous function call)

Function output:

16-bit checksum of the block of memory defined by input parameters

Function performance:

Function size is 54 bytes.²

The function duration depends on defined block size. Several examples are shown in table 3.³

Table 3. Duration of IEC60730B_Flash_SWTest() in dependence of tested block size.

Block size (Bytes)	Clock cycles	Execution time (approximately)
0x4	490	6.12 µs
0x8	980	12.25 µs
0x10	1920	24 µs

Calling restrictions:

None

3.6 IEC60730B_CM4_CM7_Flash_SWTest_32()

This function calculates the 32-bit CRC polynomial (0x04C11DB7) without using hardware.

Function prototype:

*unsigned short IEC60730B_CM4_CM7_Flash_SWTest_32(const unsigned long startAdd, *
const unsigned long size, unsigned long moduleAddress, const unsigned long crcVal);

Function inputs:

startAdd – first address of memory block to be tested
size – size of block to be tested
module_address - this parameter has no influence, its purpose is only better compatibility with the
IEC60730B_CM4_CM7_Flash_HWTest function
crcVal – input seed value for calculation (for the first iteration is 0, for next iterations it is the
result from the previous function call)

Function output:

32-bit checksum of the block of memory defined by input parameters

Function performance:

Function size is 48 bytes.²

The function duration depends on defined block size. Several examples are shown in table 4.³

Table 4. Duration of IEC60730B_Flash_SWTest_32() in dependence of tested block size.

Block size (Bytes)	Clock cycles	Execution time (approximately)
0x4	480	6 µs
0x8	914	11.425 µs
0x10	1820	22.75 µs

Calling restrictions:

None

3.7 IEC60730B_CM4_CM7_Flash_HWTest_DCP()

This function calculates the 32-bit CRC polynomial (0x04C11DB7) with the use of the DCP (Data co-processor) module.

Function prototype:

*void IEC60730B_CM4_CM7_Flash_HWTest_DCP (const unsigned long startAdd, const unsigned long size, const unsigned long crcVal, const dcp_channels_t channel, flash_dcp_state_t *psDCPState, const unsigned long tag);*

Function inputs:

startAdd - first address of the tested memory
size - size of tested block of memory
crcVal - start condition seed
channel - DCP channel used for calculation (enumeration type)
psFlashDCPState - variable which stores state & result of each DCP channel
tag - differentiates calculation on the same channel

Function output:

Void. Function does not return any value. Checksum is stored into the defined variable of *flash_dcp_state_t* type.

Function performance:

Function size is 472 bytes.²

Function duration depends on the defined block size. Several examples are shown in table 5.⁴

Table 5. Duration of IEC60730B_CM4_CM7_Flash_HWTest_DCP() in dependence of tested block size

Block size (Bytes)	Clock cycles	Execution time (approximately)
0x10	57	2.375 µs
0x20	57	2.375 µs
0x50	67	2.80 µs
0x500	261	10.88 µs

Calling restrictions:

The function cannot be interrupted with function that changes the content or setup of HW DCP module. Data block which should be calculated must be aligned to 4 bytes.

1 – The same implementation example is used for IEC60730B_CM4_CM7_Flash_SWTest()

2 – Function compiled by IAR v8.22.2

3 – The number of cycles and the execution time were measured by MKV31 / 80 MHz CPU clock / 20 MHz flash clock.

4- The number of cycles and the execution time were measured by MIMXRT1050 / 600 MHz CPU clock

4 Invariable Memory Test Module test concept

Content moved to document

5 Invariable Memory Test Validation

Content moved to document

Table V. Validation

Date	Validated by	Validated Revision Of Document	Validated Version Of Source Code	Validation Result
11/2015	Jaroslav Lepka	1.0	1.0	P
11/2016	Pavel Sustek	1.1	1.0	P
11/2018	Jaroslav Lepka	3.0	3.0	P

Validation result options:

P – Passed

F – Failed

N/A – Not applicable

How to Reach Us:

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address:
nxp.com/SalesTermsandConditions.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, Freescale, and the Freescale logo are trademarks of NXP B.V. All other product or service names are the property of their respective owners.

ARM, the ARM logo, and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2016 NXP B.V.

