

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

CPU Program Counter Register 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 devices

1	CPU PROGRAM COUNTER TEST ARCHITECTURE	4
2	CPU PROGRAM COUNTER TEST IN COMPLIANCE WITH IEC/UL STANDARDS.....	8
3	CPU PROGRAM COUNTER TEST IMPLEMENTATION.....	8
3.1	IEC60730B_CM4_PC_INIT().....	10
3.2	IEC60730B_CM4_PC_TEST().....	11
3.3	IEC60730B_CM7_PC_TEST().....	12
3.4	IEC60730B_PC_OBJECT()	13
4	PROGRAM COUNTER TEST MODULE TEST CONCEPT.....	14
5	PROGRAM COUNTER TEST VALIDATION	14

1 CPU Program Counter Test Architecture

The CPU Program Counter register test procedure tests the CPU Program Counter register for stuck at condition. Program Counter register test can be performed once after the microcontroller reset and also during runtime.

The identification of the safety error is ensured by the specific FAIL return in the case of the CPU program counter register not working correctly. The application developer must assess the return value of the test function, and if it is equal to the FAIL return, then the jump into the safety error handling function will occur. The safety error handling function may be specific according to the application and is not a part of our library. The main purpose of this function is to put the application into a safety state. Because of unpredictable behavior if the PC is corrupted, there is one additional identification mechanism. During the test, the flag parameter is set to 1 and after successful execution it is cleared.

By contrast to other CPU registers, the PC cannot simply be filled with a test pattern. It is necessary to force the CPU (program flow) to access the corresponding address which is testing the pattern to verify the program counter functionality.

Two types of PC test were developed. First type (called CM4 type) works well with Kinetis devices that have RAM memory mapped around the address 0x2000_0000. Two addresses, one above and the other below 0x2000_0000 are used to test as many bits of the program counter register as possible.

The second type (called CM7 type) of test was developed because of devices that do not have such memory mapping. It is universal and can be well used also instead of the first type.

In the first type, there is an initialization function that needs to be called first. This function copies the short test routine from the FLASH memory into two places in the RAM memory. The addresses of these places must be declared in the linker configuration file. The test function forces the PC to jump to these places to perform the test.

In the second type, the PC test works without an initialization function. Another object, the short function, is written in a separate file. This object needs to be placed in an appropriate address in the FLASH memory by declaring it in the linker configuration file. The test function uses address of this routine and also appropriate address in the RAM memory for testing the PC.

Block diagrams for the program counter register tests are shown in the following figures:

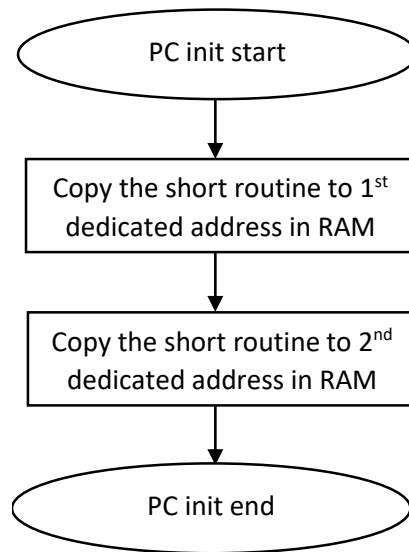
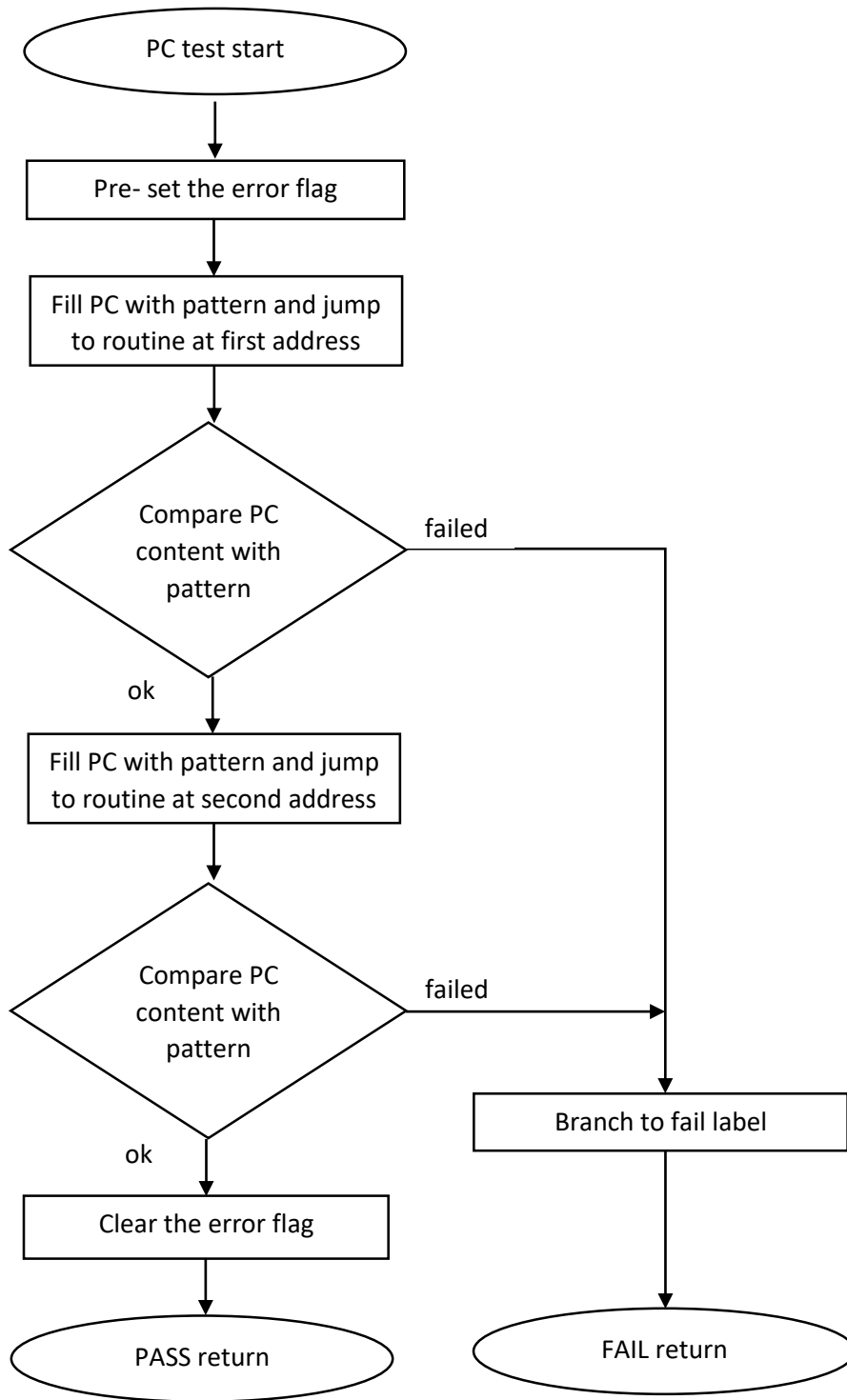


Figure 1. Block diagram for IEC60730B_CM4_PC_Init()

**Figure 2.** Block diagram for IEC60730B_CM4_PC_Test()

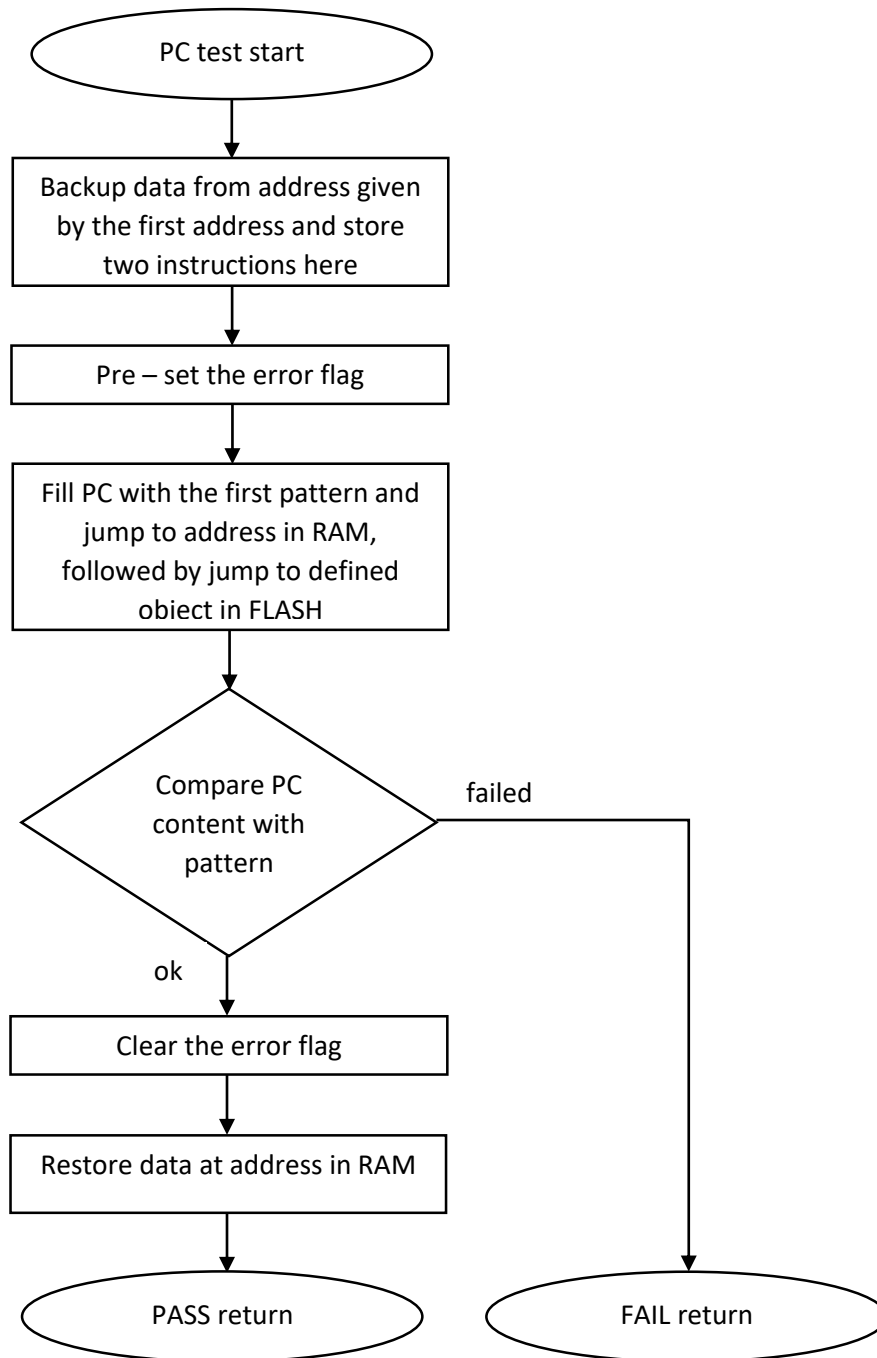


Figure 3. Block diagram for IEC60730B_CM7_PC_Test()

2 CPU Program Counter 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 Program Counter Test in Compliance with the IEC/UL Standards

Test	Component	Fault/Error	SW / HW Class	Acceptable Measures
CPU	CPU (1.3 – Programme Counter)	Stuck at	B/R.1	Periodic self test

3 CPU Program Counter Test Implementation

Test functions for the Program Counter are placed in *IEC60730_B_CM4_CM7_pc.S* and are written as assembler functions. The header file with the test patterns and the function prototypes is *IEC60730_B_CM4_CM7_pc.h*. *IEC60730_B_CM4_CM7.h* and *asm_mac_common.h* are files that must also be placed in the application. For the second test type, the file *IEC60730B_B_CM4_CM7_pc_object.S* must be placed in the appropriate address in FLASH memory.

Implementation of the first type can be as follows:

To initialize the CPU Program Counter test:

IEC60730B_CM4_PC_Init()

To test the CPU Program Counter after reset and during runtime:

IEC60730B_CM4_PC_Test()

The principle of this test case is based on placing two test functions to specific variable memory (RAM) addresses. The test patterns are chosen in order to test 1 GB on the invariable memory (flash). The test patterns change 30 bits of the low significant part of 32 bit program counter.

Test patterns:

- Test pattern 1: 0x1FFFFFF8
- Test pattern 2: 0x20000006

This test can be used to check the program counter can address up to 1 GB of the source code size. The use of the function is the same after reset and during runtime. The developer must pay attention in case of use of the function during runtime (because of interrupts) as described in the implementation chapter. The user must ensure that the area in RAM that is used for the test cannot be rewritten in the application.

The following is an example of the function calling:

```
#include "IEC60730_B_CM4_CM7.h"
#define IEC60730B_CFG_PC_ADDR0    (0x1FFFFFF8)
#define IEC60730B_CFG_PC_ADDR1    (0x20000006)
extern unsigned long PC_test_flag; /* from Linker configuration file */
const unsigned long Program_Counter_test_flag = (unsigned long)&PC_test_flag;
#define PC_TEST_FLAG ((unsigned long *) Program_Counter_test_flag)

IEC60730B_CM4_PC_Init (IEC60730B_CFG_PC_ADDR0, IEC60730B_CFG_PC_ADDR1);
IEC60730B_pc_test_result = IEC60730B_CM4_PC_Test(IEC60730B_CFG_PC_ADDR0,
IEC60730B_CFG_PC_ADDR1, PC_TEST_FLAG);
if(IEC60730B_ST_PC_FAIL == IEC60730B_pc_test_result)
    SafetyError();
```

Implementation example of the second type:

The only function that is handled in the application is:

```
IEC60730B_CM7_PC_Test()
```

The user must place an appropriate pattern as a first input. If needed, the function can be called more times in sequence with different patterns. Note that the test pattern must be the real address in the RAM and must be even-numbered. The *IEC60730B_B_CM4_CM7_pc_object.S* file must be placed at an appropriate address in the FLASH memory.

The following is an example of the function calling:

```
#include "IEC60730_B_CM4_CM7.h"
extern unsigned long PC_test_flag; /* from Linker configuration file */
const unsigned long Program_Counter_test_flag = (unsigned long)&PC_test_flag;
#define PC_TEST_FLAG ((unsigned long *) Program_Counter_test_flag)

IEC60730B_pc_test_result = IEC60730B_CM7_PC_Test(0xAAAA, IEC60730B_PC_object, PC_TEST_FLAG);
if(IEC60730B_ST_PC_FAIL == IEC60730B_pc_test_result)
    SafetyError();
```

3.1 IEC60730B_CM4_PC_Init()

This function initializes the CPU program counter test for CM4 core-based devices. This function expects that the linker reserved RAM memory from the address 0x1FFFFFF8 to 0x20000013.

This example shows how to configure the linker configuration file (KV46_256KB_Safety_common.icf) for the IAR tool:

```
/* memory regions */
define symbol __ICFEDIT_region_RAM_start__          = 0x1FFFC410;
define symbol __ICFEDIT_region_RAM_end__            = 0x1FFFFFF7;
define symbol __ICFEDIT_region_RAM_PCTEST_start__   = 0x1FFFFFF8;
define symbol __ICFEDIT_region_RAM_PCTEST_end__     = 0x20000013;
define symbol __region_RAM2_start__                 = 0x20000014;
define symbol __region_RAM2_end__                   = 0x20003FDB;
define exported symbol PC_test_flag                 = 0x20003FDC;
define region RAM_region = mem:[from __ICFEDIT_region_RAM_start__ to
__ICFEDIT_region_RAM_end__]|mem:[from __ICFEDIT_region_RAM2_start__ to
__ICFEDIT_region_RAM2_end__];
define region PCTEST_region = mem:[from __ICFEDIT_region_RAM_PCTEST_start__ to
__ICFEDIT_region_RAM_PCTEST_end__];
/* Define memory block */
define block CodeRelocateRam { section .text; };
define block PCTEST_block { section .pctest; };
/* Put data into memory blocks */
place in RAM_region { readwrite, block CodeRelocateRam, block HEAP, block CSTACK, zi,rw };
place in PCTEST_region { block PCTEST_block; }
```

Function prototype:

void IEC60730B_CM4_PC_Init (unsigned long pattern1, unsigned long pattern2).

Function inputs:

pattern1 – First testing address. 0x1FFFFFF8

pattern2 – Second testing address. 0x20000006

Function output:

Void. The function does not return a value.

Function performance:

The function duration is approximately 243 cycles (3.04 μ s)¹. Function size is 42 bytes.²

Calling restrictions:

The function needs to have reserved RAM memory from address 0x1FFFFFF8 to 0x20000013.

3.2 IEC60730B_CM4_PC_Test()

This function checks the Program Counter register for CM4 core-based devices. The Program Counter register is tested according to the block diagram in [Figure 2](#).

Function prototype:

```
IEC60730B_RESULT IEC60730B_CM4_PC_Test(unsigned long pattern1, unsigned long pattern2, unsigned long* PC_flag);
```

Function inputs:

pattern1 – First testing address. 0x1FFFFFF8

pattern2 – Second testing address. 0x20000006

* PC_flag –pointer to address in memory (that is not accessible by startup code) used as flag

Function output:

```
typedef unsigned long IEC60730B_RESULT;
```

It can have two values:

IEC60730B_ST_PC_FAIL (0x00000201)

IEC60730B_ST_PC_PASS (0)

In case of incorrect test execution, PC_flag has a value of 1.

Function performance:

The function duration is approximately 90 cycles, (1.13 μ s)¹. The function size is 52 bytes.²

Calling restrictions:

This function cannot be interrupted. This function can be called after the initialization routine of the CPU Program Counter test IEC60730B_PC_Init () has been called. Its dedicated place in RAM (PCTEST_region) cannot be corrupted after initialization.

3.3 IEC60730B_CM7_PC_Test()

This function performs the second version of the Program Counter test. The Program Counter register is tested according to the block diagram in [Figure 3](#).

Function prototype:

```
IEC60730B_RESULT IEC60730B_CM7_PC_Test(unsigned long pattern1, tFcn_pc pObject_function,  
unsigned long* PC_flag);
```

Function inputs:

pattern1 – Appropriate address in RAM memory
pObject_function – pointer to the function. It is always: IEC60730B_PC_object – auxiliary function from *IEC60730B_B_CM4_CM7_pc_object.S* file
* PC_flag – pointer to address in memory (that is not accessible by startup code) used as flag

Function output:

```
typedef unsigned long IEC60730B_RESULT;
```

It can have two values:

IEC60730B_ST_PC_FAIL (0x00000201)

IEC60730B_ST_PC_PASS (0)

In case of incorrect test execution, PC_flag has a value of 1.

Function performance:

The function duration is approximately 34 cycles, (1.08 μ s)². The function size is 87 bytes.²

Calling restrictions:

This function cannot be interrupted.

3.4 IEC60730B_PC_object()

This function is used to perform the PC test, it should be called only by the *IEC60730B_CM7_PC_Test()* function. The user must place this function at a reliable address in the FLASH memory.

The following example shows how to place the function at the desired address in the linker configuration file (*KV58_1MB_Safety_common.icf*) for the IAR tool:

```
define symbol __PC_test_start__          = 0x100FFFE0;
define symbol __PC_test_end__            = 0x100FFFFF;
define region PC_region = mem:[from __PC_test_start__ to __PC_test_end__];
define block PC_TEST { section .text object IEC60730_B_CM4_CM7_pc_object.o};
place in PC_region { block PC_TEST};
```

Function prototype:

```
void IEC60730B_PC_object(void);
```

Function inputs:

Void. No inputs.

Function output:

Void. The function does not return any value.

Function performance:

The function duration is included in the duration of the *IEC60730B_CM7_PC_Test()* function. Its size is 16 bytes.²

Calling restrictions:

None

1 – The number of cycles and the execution time were measured with the use of a MKV31 / 80 MHz CPU clock / 20 MHz flash clock.

2 – Function compiled by IAR v8.22.2

4 Program Counter Test Module test concept

Content moved to TestReport_IEC60730B_ProgramCounter_rev3_0 document

5 Program Counter Test Validation

Content moved to TestReport_IEC60730B_ProgramCounter_rev3_0 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.

