

# IEC 60730B Safety Routines for Kinetis MCUs

by: Pavel Krenek

## 1 Introduction

The purpose of the IEC 60730 safety routines for household appliances is to ensure safe and reliable operation of their products. Freescale has developed safety routines to help manufacturers of automatic control systems meet the IEC 60730 standard class B in the large appliance and industrial control market.

Typical applications for Freescale safety routines are: cooking products, boiler and heater control, dishwashers, household actuators, dryers, motor control, refrigerators and freezers, elevators, vacuum cleaners, automatic gates and washing machines. The routines' task is to check proper functioning of registers, RAM, and flash memory. These routines have been developed for the 32-bit microcontrollers based on ARM<sup>®</sup>Cortex<sup>®</sup> M0+ and Cortex M4 cores.

For industrial safety applications, electronic manufacturers are required to design safety measures that monitor and verify key components of the embedded system into their automatic electronic control systems.

These periodic safety routines will help Freescale customers in attaining IEC 60730 class B certification.

### Contents

1. Introduction . . . . .	1
2. Kinetis MCUs based on ARM technology . . . . .	2
3. Kinetis registers . . . . .	3
4. Kinetis memory . . . . .	5
5. Kinetis safety routines . . . . .	6
6. CPU and FPU registers self-test for "stuck at" faults . . . . .	6
7. RAM memory test . . . . .	9
8. Watchdog modules test . . . . .	13
9. Flash memory test . . . . .	18
10. Clock test . . . . .	19
11. Revision history . . . . .	20

There are three classes of IEC 60730 standards:

- class A: not intended to be relied upon for the safety of the equipment
- class B: to prevent unsafe operation of the controlled equipment
- class C: to prevent special hazards

Table 1 indicates the requirements for each routine considering the given class. All tests described by this application note are outlined in black.

**Table 1. IEC60730 household standards supported devices**

Test	IEC 60730 Class B				IEC 60730 Class C		
	8-bit S08	16-bit DSC568XXX	32-bit MCF51xx	32-bit Kinetis	8-bit S08	16-bit DSC568xxx	32-bit MCF51xx
<b>IEC 60730 Periodic Test</b>	8-bit S08	16-bit DSC568XXX	32-bit MCF51xx	32-bit Kinetis	8-bit S08	16-bit DSC568xxx	32-bit MCF51xx
<b>CPU Register Test</b>	Stuck at	Stuck at	Stuck at	Stuck at	Walkpat	Walkpat	Walkpat
<b>CPU Instruction Test</b>	N/A	N/A	N/A	N/A	CPU Instr. Test	CPU Instr. Test	CPU Instr. Test
<b>RAM Test</b>	March C, X	March C	March C	March C, X	Walkpat	Walkpat	Walkpat
<b>Flash Test</b>	CRC 16-bit	CRC 16-bit	CRC 16-bit	CRC 16-bit	CRC 16-bit	CRC 16-bit	CRC 16-bit
<b>Watchdog Test</b>	Timeout and Reset	Timeout and Reset	Timeout and Reset	Timeout and Reset	Timeout and Reset	Timeout and Reset	Timeout and Reset
<b>Clock Test</b>	-	-	-	Independent clock testing	-	-	-

## 2 Kinetis MCUs based on ARM technology

Kinetis portfolio of ARM Cortex MCUs consists of multiple hardware and software-compatible ARM Cortex-M0+ and ARM Cortex-M4 MCU families with exceptional low-power performance, memory scalability, and feature integration. Families range from the entry-level ARM Cortex-M0+ Kinetis L Series to the high-performance, feature-rich ARM Cortex-M4 Kinetis K and include a wide selection of analog, communication, HMI, connectivity, and security features.

Kinetis L based on Cortex-M0+ core has the following features and benefits:

- Tight integration of system peripherals – reduces size and development costs
- Thumb instruction set – combines high code density with 32-bit performance
- Support for single-cycle I/O access
- Power control optimization of system components
- Integrated sleep modes for low power consumption
- Fast code execution – enables running the processor with a slower clock or increasing sleep mode time
- Optimized code fetching – reduces flash and ROM power consumption

- Hardware multiplier
- Deterministic, high-performance interrupt handling for time-critical applications
- Deterministic instruction-cycle timing
- Support for system-level debug authentication
- Serial Wire Debug – reduces the number of pins required for debugging
- Support for optional instruction trace

Kinetis K based on Cortex-M4 core has the following features and benefits:

- Tight integration of system peripherals – reduces size and development costs
- Thumb instruction set – combines high code density with 32-bit performance
- IEEE<sup>®</sup>754-compliant FPU
- Code-patch ability for ROM system updates
- Power control optimization of system components
- Integrated sleep modes for low power consumption
- Fast code execution – permits slower processor clock or increases sleep mode time
- Hardware division and fast multiplier
- Deterministic, high-performance interrupt handling for time-critical applications
- Optional Memory Protection Unit (MPU) for safety-critical applications
- Extensive debug and trace capabilities:
  - Serial Wire Debug and Serial Wire Trace – reduce the number of pins required for debugging, tracing, and code profiling

The following text describes the Kinetis registers, memory and watchdog; their functionality is checked using safety routines.

### 3 Kinetis registers

The core registers that are considered part of the core-programming model are shown in [Figure 1](#).

The ARM Cortex-M0+ and ARM Cortex-M4 MCUs consist of the following set of registers:

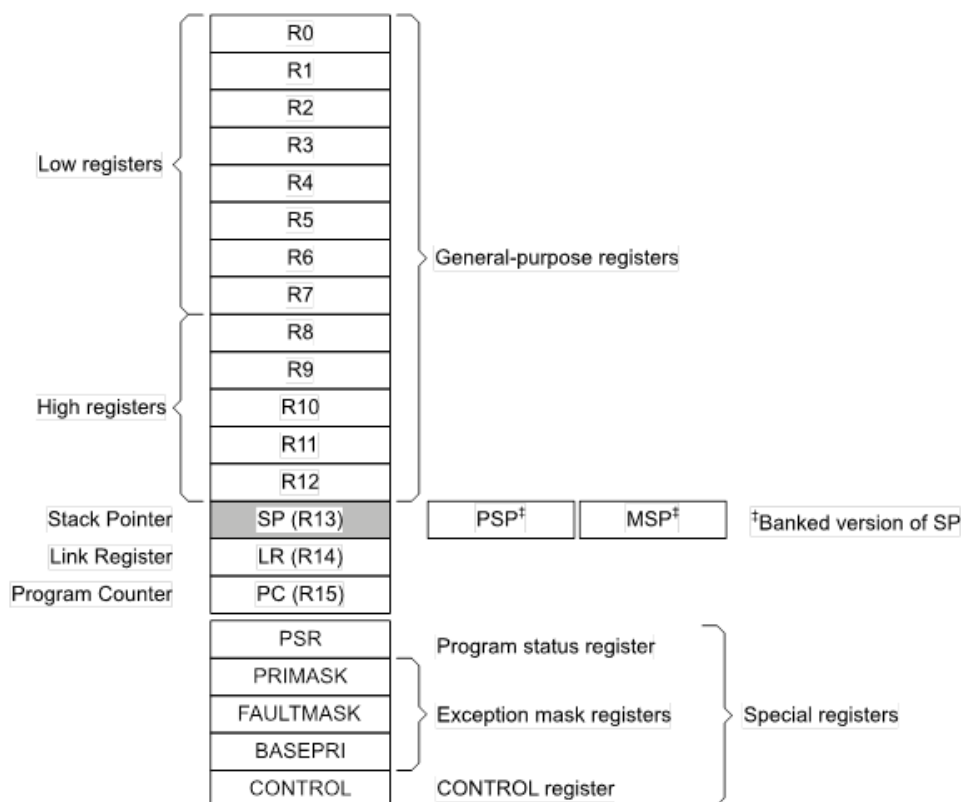
- Thirteen 32-bit general-purpose registers for data operations
- Two stack pointers for using in thread mode:
  - main stack pointer (MSP)
  - process stack pointer (PSP)
- One link register (LR) – stores the return information for subroutines, function calls, and exceptions
- One 32-bit program counter (PC) register – contains the current program address
- Three program status registers (PSR):
  - application program status register (APSR)
  - interrupt program status register (IPSR)
  - execution program status register (EPSR)

## Kinetis registers

- One PRIMASK register – prevents activation of all exceptions with configurable priority
- One CONTROL register – controls the stack used, and optionally the code privilege level, when the processor is in Thread mode
- 32 single-precision extension registers – can be accessed as 16 doubleword registers for load, store, and move operations

The Cortex-M4F FPU implements the FPv4-SP floating-point extension. The FPU fully supports single-precision add, subtract, multiply, divide, multiply and accumulate, and square root operations. It also provides conversions between the fixed-point and floating-point data formats, and floating-point constant instructions.

The FPU provides floating-point computation functionality that is compliant with the *ANSI / IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic*, referred to as the IEEE 754 standard.



**Figure 1. ARM Cortex-M4 registers**

## 4 Kinetis memory

This section describes the processor memory map, the behavior of memory accesses, and the bit-banding features. The processor has a fixed memory map that provides up to 4 GB of addressable memory. The ARM Cortex-M0+ system memory map is ARM6-M architecture compliant, and is common both to the debugger and processor accesses. The ARM Cortex-M4 system memory map is ARM7-M architecture compliant. For more information about the memory architecture see the ARM6-M or ARM7-M Architecture Reference Manual.

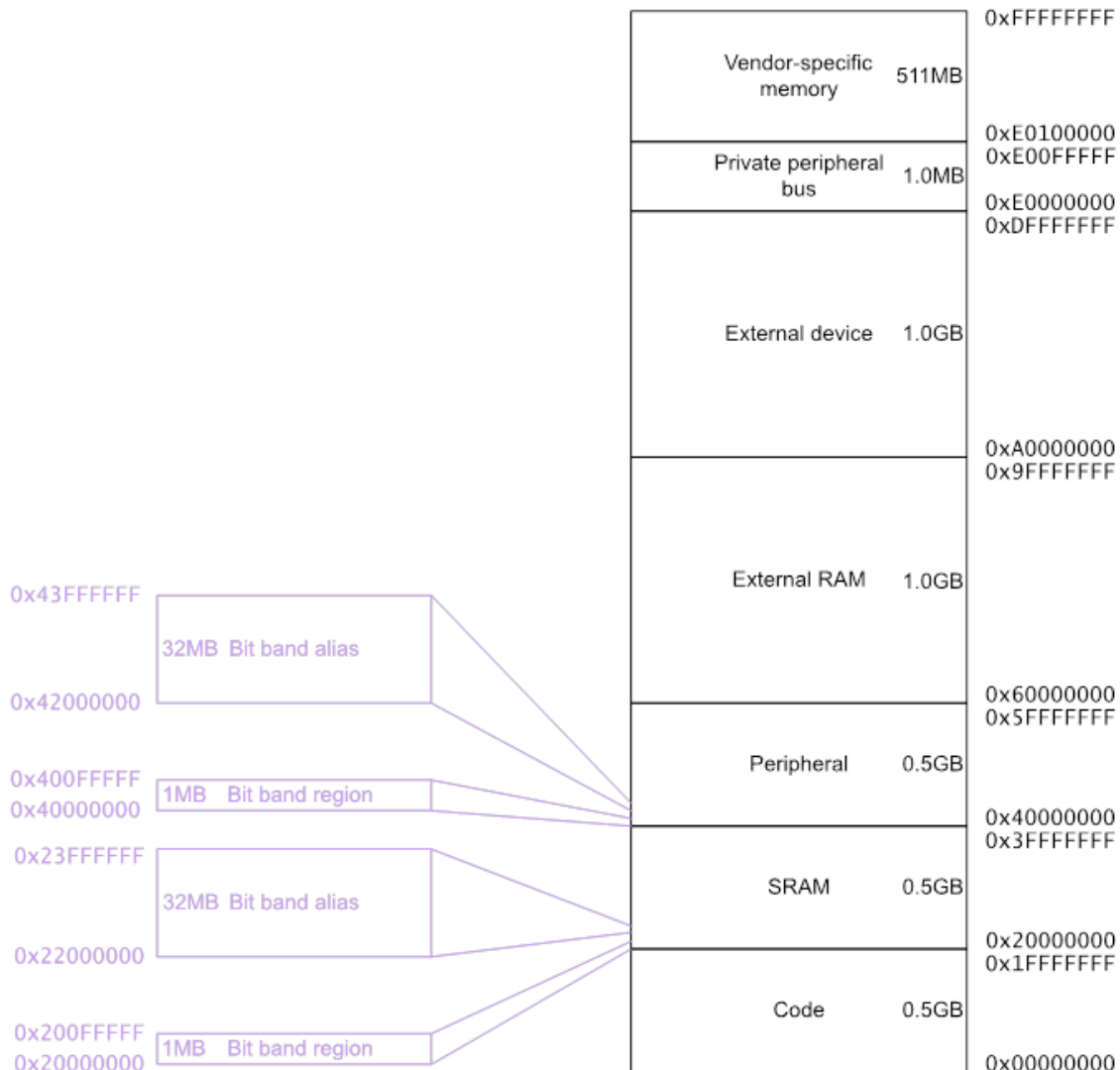


Figure 2. Kinetis memory region

## 5 Kinetis safety routines

The Kinetis safety routines consist of the following five tests:

- Watchdog 2008 (Kinetis K, M, V) module and COP (Kinetis L) module timeout test (watchdog module with windowing feature) and module for Kinetis E
- CPU register self-test for "stuck at" faults
- March X and March C RAM tests
- Flash memory CRC test
- Clock test

All of the tests above can be run automatically by using the self-test.

## 6 CPU and FPU registers self-test for “stuck at” faults

A periodic self-test has been designed to test all previously mentioned registers. The self-test is expected to be ran immediately after any reset of the MCU, but can also be used in an application, as long as the interrupts are disabled and the stack pointer is reset to the top of the stack.

Each CPU register and program counter is checked for “stuck at” faults by using the 0x55 and 0xAA bit patterns. The registers are first tested for the 0x55 value and then for the 0xAA value. This tests all bits for zero and one conditions. If an error is found, it goes through the error routine, and returns the error code of a particular register.

All the CPU registers can be tested using one of the following functions:

- IEC60730B\_KINETIS\_CpuRegisterTest – used for testing of all registers
- IEC60730B\_KINETIS\_CpuRegisterTestFPU – can be used to test the MCU with ARM Cortex-M4 and the floating-point unit for “stuck at” faults of all FPU registers
- IEC60730B\_KINETIS\_ProgramCounterTest – this function performs program counter test for CORTEX M4 or CORTEX M0+ MCUs

### NOTE

The final settings of the tested core are performed by using pre-processor macros in the header files.

The ARM Cortex-M4 with FPU (floating-point unit) consists of 16 general-purpose 32-bit registers and three other specific registers.

### 6.1 32-bit data registers

The FPU provides an extension register file that contains 32 single-precision registers.

- S0-S32
- FPSCR register

The S0-S32 registers are tested using a simple algorithm load-test pattern, and checking whether it loaded correctly. For a complete test, two patterns are used, 0x55555555 and 0xAAAAAAAA. They test all bits in register for one and zero. The FPSCR register is tested by patterns 0x55400015 and 0xA280008A.

### 6.1.1 Data registers

The 32-bit data registers are checked by writing pattern 0x55555555 to the register and comparing the value with an immediate value in another data register. The register value is then loaded with a "complemented" value of 0xAAAAAAAA and verified with immediate value of 0xAAAAAAAA in another data register.

The Kinetis MCU with floating point unit (FPU) includes an advanced set of sixteen 64-bit registers. This FPU set of registers can be accessed as 32-bit or 64-bit register. This set of registers is tested using the same principle as general-purpose 32-bit registers (R0-R12). The algorithm of CPU register testing is shown in [Figure 3](#).

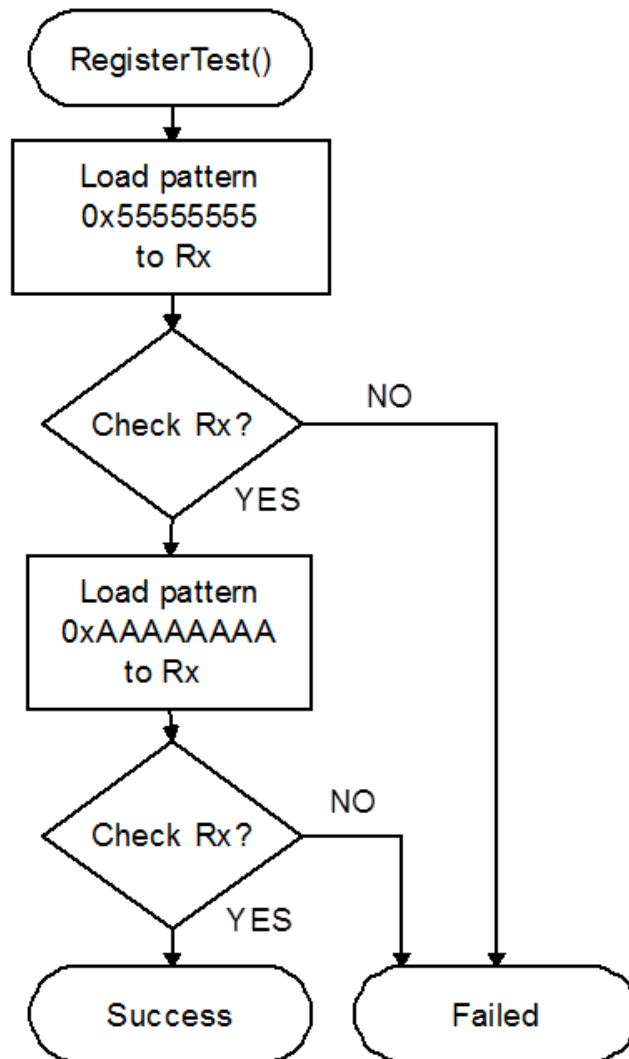


Figure 3. CPU register testing algorithm

## 6.1.2 Program counter register

Due to Kinetis memory organization and Kinetis CPU behavior, it is not possible to test all bits of the PC register. There exists an issue with linker file and flash memory allocation for small pieces of test code. In this case, the flash memory used for testing of the PC register causes the fragmentation of user application and problems in user code. For this reason, the test code can be placed in RAM memory only, and therefore three MSB bits of valid address range cannot be tested. In addition, the LSB bit of PC register cannot be tested, because the Kinetis core cannot acknowledge odd addresses.

In short, the 32-bit PC register only uses 29-bits to address the complete address range of flash and RAM of the Cortex-M4 and M0+.

## 6.1.3 Stack pointer register

Both the ARM Cortex-M4 and ARM Cortex-M0+ kernels have two stack pointers: main stack pointer (MSP) and process stack pointer (PSP). The handler mode uses MSP and the process mode uses MSP or PSP. R13 indicates current SP.

The SP 32-bit register is tested for “stuck at” faults. The first byte of the register cannot be tested, because the register is aligned to four bytes. The main stack pointer (MSP) is tested, and the register is compared to the appropriate pattern in the data register. Similar procedure is used to test the process stack register (PSP).

## 6.1.4 Special registers

The ARM Cortex-M4 and ARM Cortex-M0+ include several special registers. [Table 2](#) describes all information about those special registers.

**Table 2. ARM Cortex-M4 and ARM Cortex-M0+ special registers**

Register	Description	ARM Cortex-M4	ARM Cortex-M0+
PRIMASK	prevents activation of all exceptions with configurable priority	X	X
PSR	includes three registers (APSR, IPSR, EPSR) which contain the current state of the condition flags from previous instruction executions	X	X
FAULTMASK	prevents activation of all exceptions except for Non-Maskable Interrupt (NMI)	X	–
BASEPRI	serves for settings of 16 interrupt priority levels	X	–

### NOTE

All special registers are tested with the same method as data registers.

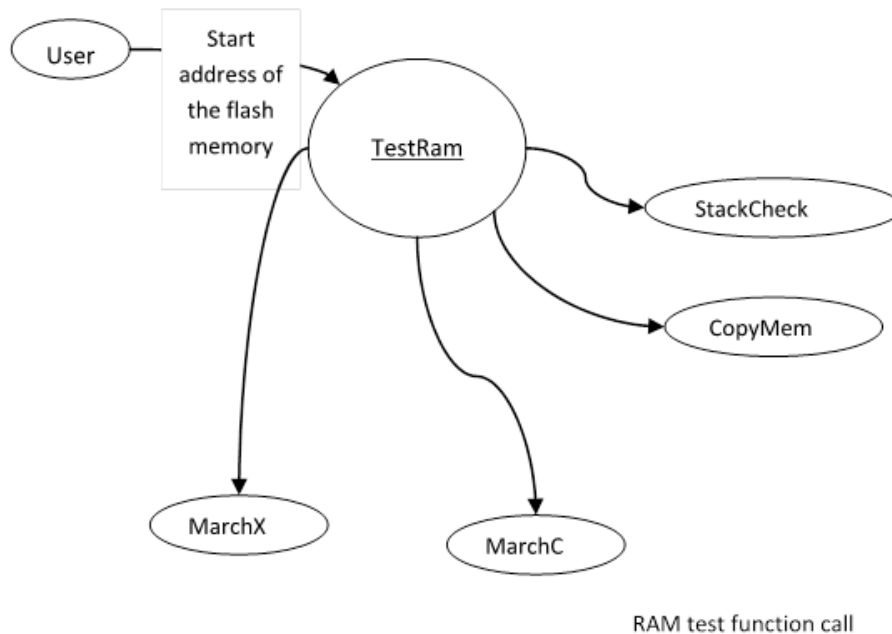


## 7 RAM memory test

Two different algorithms are implemented to test the RAM, as described in following subsections.

The test routines are destructive in their nature. To execute a transparent test pattern, the RAM is partitioned into four segments (0 – 3). The segment 3 is used as a redundant area for copying other segments temporarily, while the test pattern is being executed on them.

The function first checks the location of SP; in case it is located at segment 3, the function will not proceed and will return MEMORY\_FAIL\_SP error code. Else, the segment 3 will be checked and the function applied.



**Figure 4. RAM test function**

The function verifies the memory segment from the address given by IEC60730\_B\_RAM\_START\_RAM with the length of IEC60730\_B\_RAM\_END\_RAM. This test is destructive and works as follows:

- STEP1 – copies the particular memory segment to the segment 3 (backup memory)
- STEP2 – the particular test routine is called; this routine is defined by IEC60730B\_KINETIS\_RamTestBlockMarch\_
- STEP3 – copies the particular memory from backup to memory segment (back to original)
- STEP4 – the backup copy of memory is copied back from the segment 3
- STEP5 – in case the SP was moved to the segment 3, it is moved back (StackCheck)

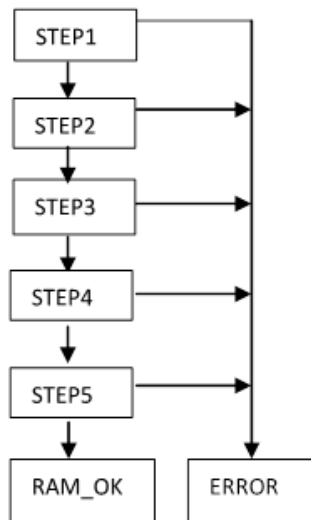


Figure 5. RAM memory testing diagram

## 7.1 March X

The March X test pattern is an industry standard for checking RAM memory arrays for “address decoder” and “stuck at” faults.

The RAM safety function uses the March X algorithm to check the memory. The function verifies the memory segment from the address given by pMemStart with the length of uintLength. This test is destructive and works as follows:

- STEP1 – writes all zeros to array; it sets the registers and does a loop to clear each memory address
- STEP2 – starting at lowest address, reads zeros, writes ones, increments up array
- STEP3 – starting at highest address, reads ones, writes zeros, decrements down array
- STEP6 – reads all zeros from array

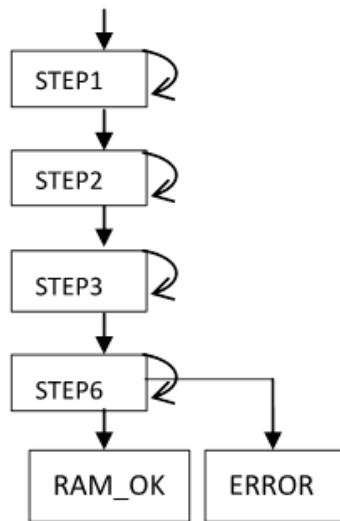


Figure 6. March X RAM memory testing diagram

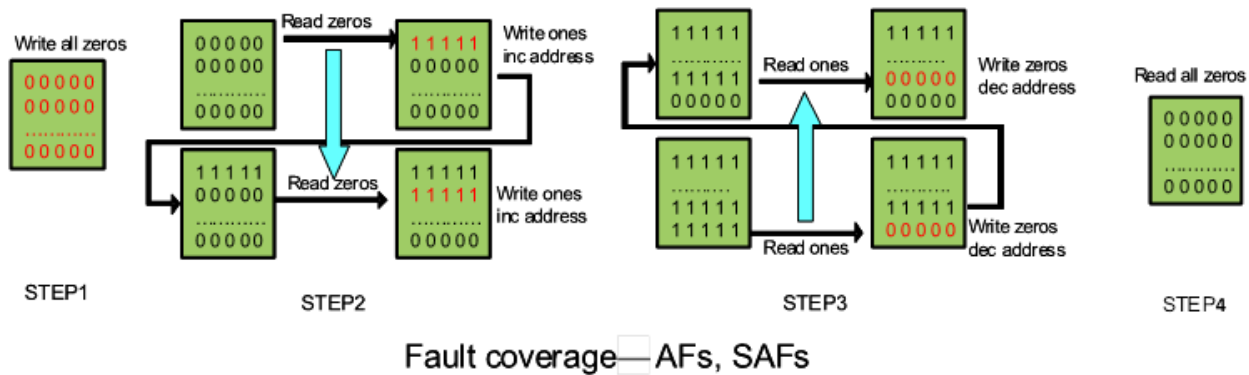


Figure 7. March X

The March X test is carried out by changing the RAM address value in debug mode. The test is executed for each bit of the selected RAM address by setting it to zero when it is one, and vice versa. The same process is repeated for all registers.

## 7.2 March C

The March C test pattern (van der Goor, 1991) is an industry standard for checking RAM memory arrays for “d.c.” faults. The RAM testing function verifies the memory segment from the address given by IEC60730\_B\_RAM\_START\_RAM with the length of memory (IEC60730\_B\_RAM\_END\_RAM). This test is destructive and works as follows:

- STEP1 – writes all zeros to array
- STEP2 – starting at lowest address, reads zeros, writes ones, increments up array
- STEP3 – starting at lowest address, reads ones, writes zeros, increments up array
- STEP4 – starting at highest address, reads zeros, writes ones, decrements down array
- STEP5 – starting at highest address, reads ones, writes zeros, decrements down array
- STEP6 – reads all zeros from array

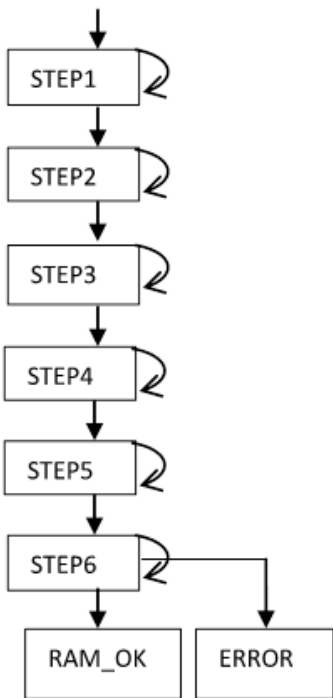
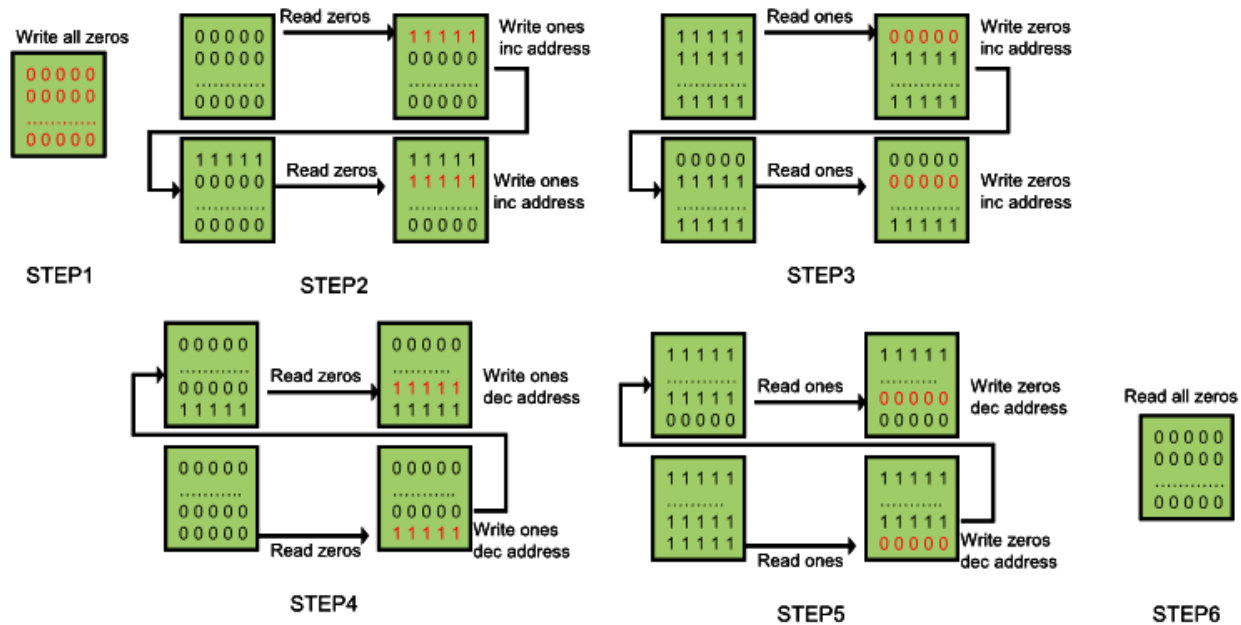


Figure 8. March C RAM memory testing diagram



March C (van der Goor, 1991)

Figure 9. March C

The March C test is carried out by changing RAM address value in debug mode. The test is executed for each bit of the chosen RAM address by setting it to zero when it is one, and vice versa. The same process is repeated for all the registers.

Function IEC60730B\_KINETIS\_RamInit – this routine initializes the RAM test system variables.

Function IEC60730B\_KINETIS\_RamTestComplete – this test performs complete testing of the RAM in defined ranges. Testing algorithm can be either March C or March X. The settings have to be defined in the library configuration file.

Function IEC60730B\_KINETIS\_RamTestBlock – this routine tests the actual segment of the RAM area, which includes the RAM test backup area. The routines are used for continuous RAM testing during the runtime of the application. The routine test returns the following results:

- IEC60730\_FALSE – routine ends badly (something is wrong in RAM)
- IEC60730\_TRUE – all is OK

Function IEC60730B\_KINETIS\_RamGetLastTestedBlock – this routine returns the actual test address of the block memory.

The RAM memory tests are provided by simple algorithms March C and March X. Two patterns, 0x00000000 and 0xFFFFFFFF, are used for the complete test, it tests all the bits in the register for one and zero. The memory failure is simulated by writing a modified value into the register.

## 8 Watchdog modules test

### 8.1 Watchdog timer module (WDOG)

The watchdog timer keeps a watch on the system functioning and resets it in case of its failure. Some reasons for such failures are: run-away software code and the stoppage of the system clock that can lead to serious consequences in a safety-critical system. In such cases, the watchdog brings the system into a safe state of operation. The watchdog monitors the operation of the system by expecting periodic communication from the software, generally known as servicing or refreshing the watchdog. If this periodic refreshing does not occur, the watchdog resets the system.

The features of the Watchdog Timer (WDOG) include:

- Independent clock source input (independent from CPU / bus clock); choice between two clock sources:
  - LPO oscillator
  - external system clock
- Unlock sequence for allowing updates to write-once WDOG control / configuration bits
- All WDOG control / configuration bits are writable only once within 256 bus-clock cycles of being unlocked
  - after unlocking, these bits need to be updated within 256 bus-clock cycles; failure to update these bits resets the system
- Programmable time-out period, specified in terms of number of WDOG clock cycles

- Ability to test WDOG timer and reset with a flag indicating watchdog test
- Windowed refresh option
- Robust refresh mechanism
- Count of WDOG resets as they occur
- Configurable interrupt on time-out to provide debug breadcrumbs – this is followed by a reset after 256 bus-clock cycles

The watchdog is a fail-safe mechanism that brings the system into a known initial state in case of its failure due to CPU clock stopping or a run-away condition in code execution. In its simplest form, the watchdog timer runs continuously off a clock source and expects to be serviced periodically, failing which resets the system. This ensures that the software is executed correctly and has not ran away in an unintended direction. Software can adjust the period of servicing or the time-out value for the watchdog timer to meet the needs of the application.

A windowed mode of operation can be selected. This mode expects the servicing to be done only in a particular window of the time-out period. An attempt to service the watchdog outside this window results in a reset. By operating in this mode, it is possible to get an indication of whether the code is running faster than expected or not. The window length is also user-programmable.

If the system fails to update / refresh the watchdog due to an unknown and persistent cause, it will be caught in an endless cycle of resets from the watchdog. To analyze the cause of such conditions, the watchdog can be programmed to first issue an interrupt, followed by a reset after a while. In the interrupt service routine, the software can analyze the system stack to help with debugging.

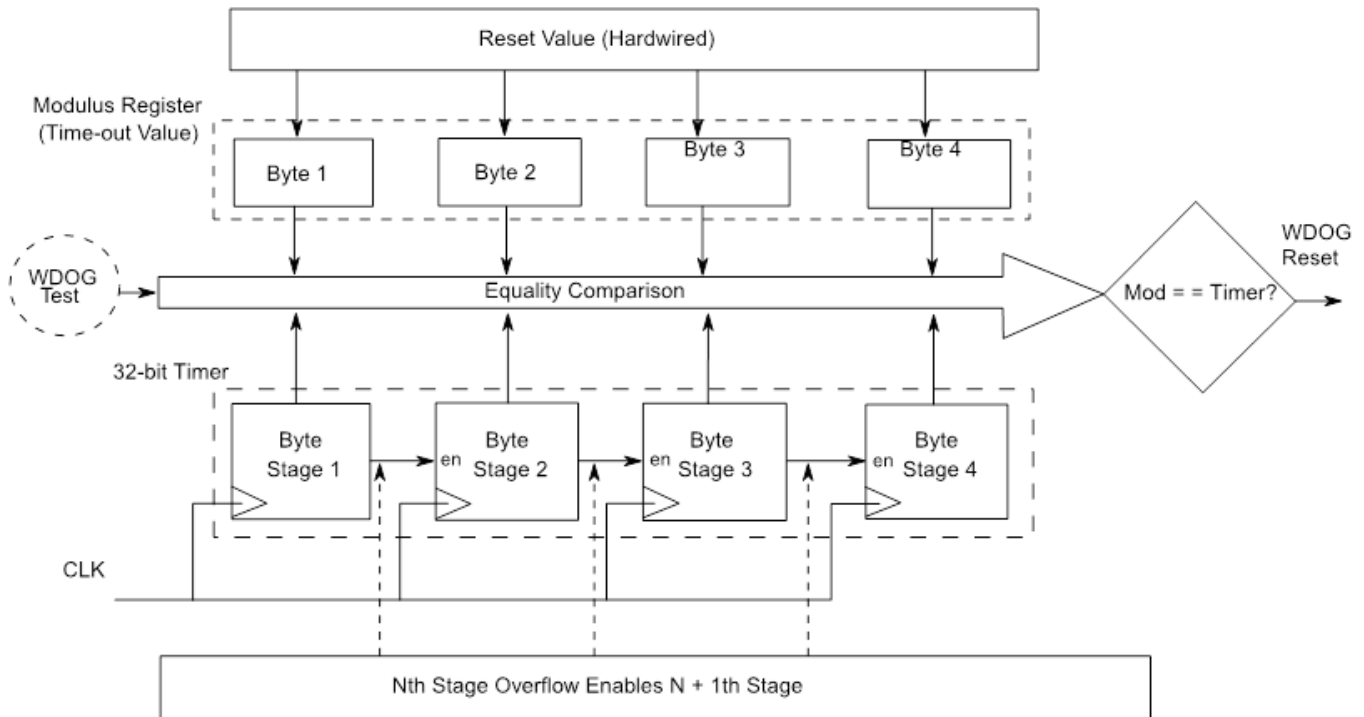
To enhance the independence of watchdog from the system, it runs on an independent LPO oscillator clock. You can also switch over to an alternate clock source through a control register bit, if required.

### 8.1.1 Watchdog testing

To test the watchdog, its main timer and its associated compare and reset logic must be tested. As for now, two tests are implemented for the watchdog (quick test and byte test). While there is a control bit provided to put the watchdog into the test mode (functional), there is also an overriding test-disable control bit which, once set, disables the test mode permanently until reset.

This test is based on embedded test performance of watchdog model 2008. Advanced byte test is set for the IEC60730.

The byte test implements a more thorough test of the watchdog timer. In this test, the timer is split up into its constituent byte-wide stages that are run independently and tested for time-out against the corresponding byte of the time-out value register. The splitting concept is explained in [Figure 10](#).



**Figure 10. Watchdog timer byte splitting**

Watchdog timer module 2008 is included on all Freescale Kinetis K and Kinetis M devices. This module offers two methods of testing the reset functionality.

The principle of the first method is setting the smallest delay possible to reset the watchdog counter and waiting in the loop. Another part of the test is focused on testing the windowing mode. This is done by setting the window value and initializing the watchdog before the upcounting procedure of the window value. If the watchdog reset occurred, the test continues with testing the watchdog time. If an issue is found at any part of the test, the function returns to the actual state of test which is described in the following paragraph. This test is implemented in the following function:

Function IEC60730B\_KINETIS\_Wdg2008Test – the basic test of the watchdog module model 2008:

- IEC60730\_COP\_TRUE – the test was carried out successfully
- IEC60730\_COP\_FALSE – the test was carried out and watchdog is not working properly
- IEC60730\_WIN\_FALSE – the test of the windowing mode has failed
- IEC60730\_COP\_APP – the test was started from an application source

Figure 11 shows the algorithm of the watchdog module basic testing, which is the same as for COP watchdog.

The principle of the second test is testing each part of the 32-bit watchdog counter independently. The watchdog module includes a 32-bit counter, which allows separate testing of the 8-bit counter. That means that all eight bits of the timer are tested separately, and after a correct result, another testing of the counter section is started. If any issues are detected, the function returns FALSE result.

If the reset of the watchdog was caused by an application, then the test function will return an appropriate value. This method of testing guarantees that the watchdog is working correctly at all time.

Function IEC60730B\_KINETIS\_Wdg2008ByteTest – this routine tests the watchdog by itself using four runs / resets to carry out the test successfully.

- IEC60730\_COP\_TRUE – the test was carried out successfully
- IEC60730\_COP\_FALSE – the test was carried out and the watchdog is not working properly
- IEC60730\_COP\_APP – the test was started by an application source, the test was not carried out and returned a value of 3

The algorithm of the watchdog module byte test is shown in [Figure 11](#).

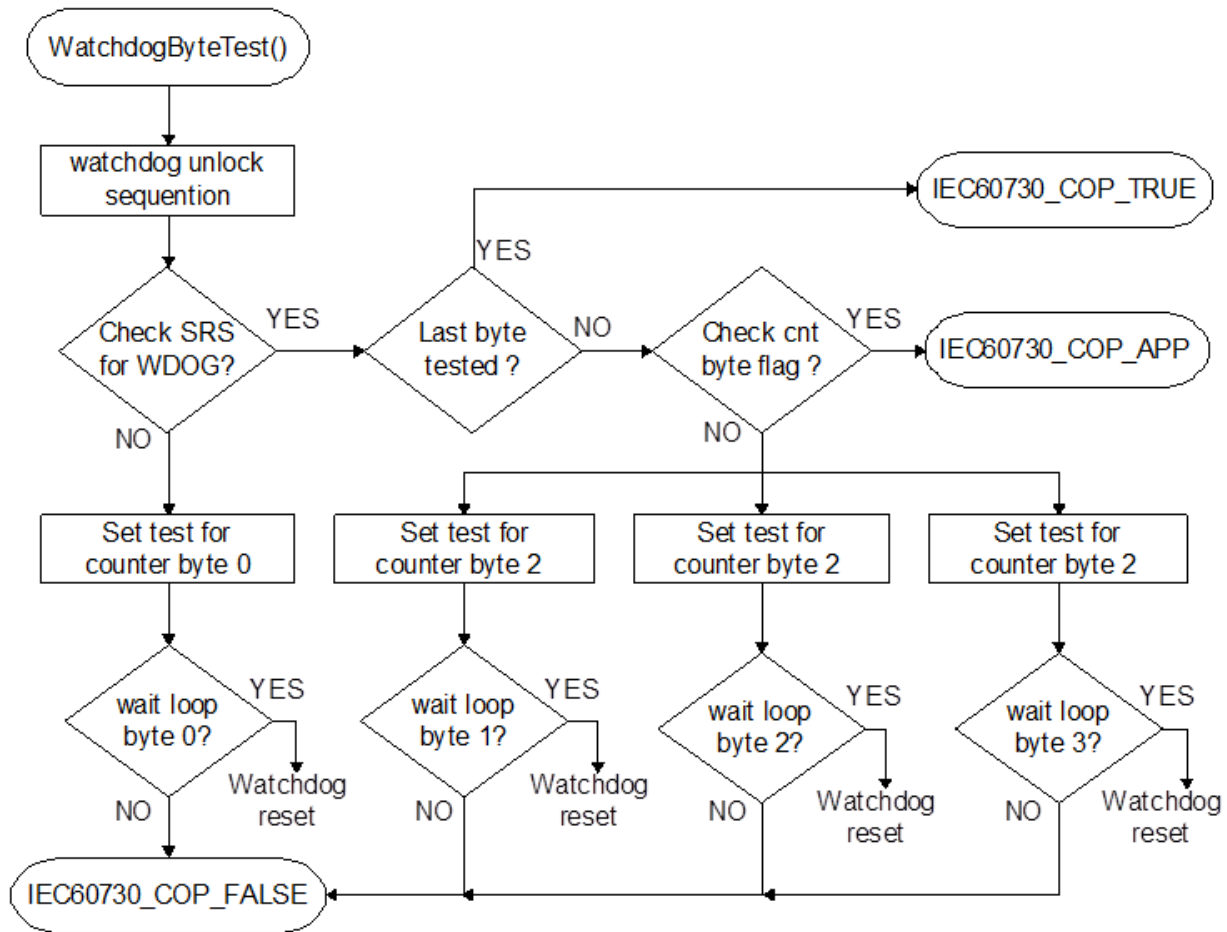


Figure 11. Algorithm of the byte watchdog module test



## 8.2 Computer operating properly (COP) watchdog timer module

The watchdog module is intended to force a system reset when the application software fails to execute as expected. To prevent a system reset from the COP timer (when it is enabled), the application software must reset the COP counter periodically. If the application program gets lost and fails to reset the COP counter before it times out, a system reset is generated to force the system back to a known starting point.

Watchdog cannot be disabled before this test and can be disabled only after the test (one-time programming). The COP control register can be written only once after a reset. This test is designed only for testing immediately after reset.

Function IEC60730B\_KINETIS\_CopTest tests the complete COP module itself using four runs / resets to carry out the test successfully. This function is also focused on testing the window mode and COP mode.

Windowing mode is tested first and the COP mode testing follows. The COP is tested by setting a short counter value and waiting in the loop for overflow. It is checked after a restart by the COP bit in SRS and also by using unique constant placed in RAM. If the COP does not overflow, the function returns FALSE value of windowing test.

The COP overflow mode is tested once the windowing test is carried out successfully.

The IEC60730B\_KINETIS\_CopTest function can recognize the source of the root cause; in case it is the application, the function returns value IEC60730\_COP\_APP.

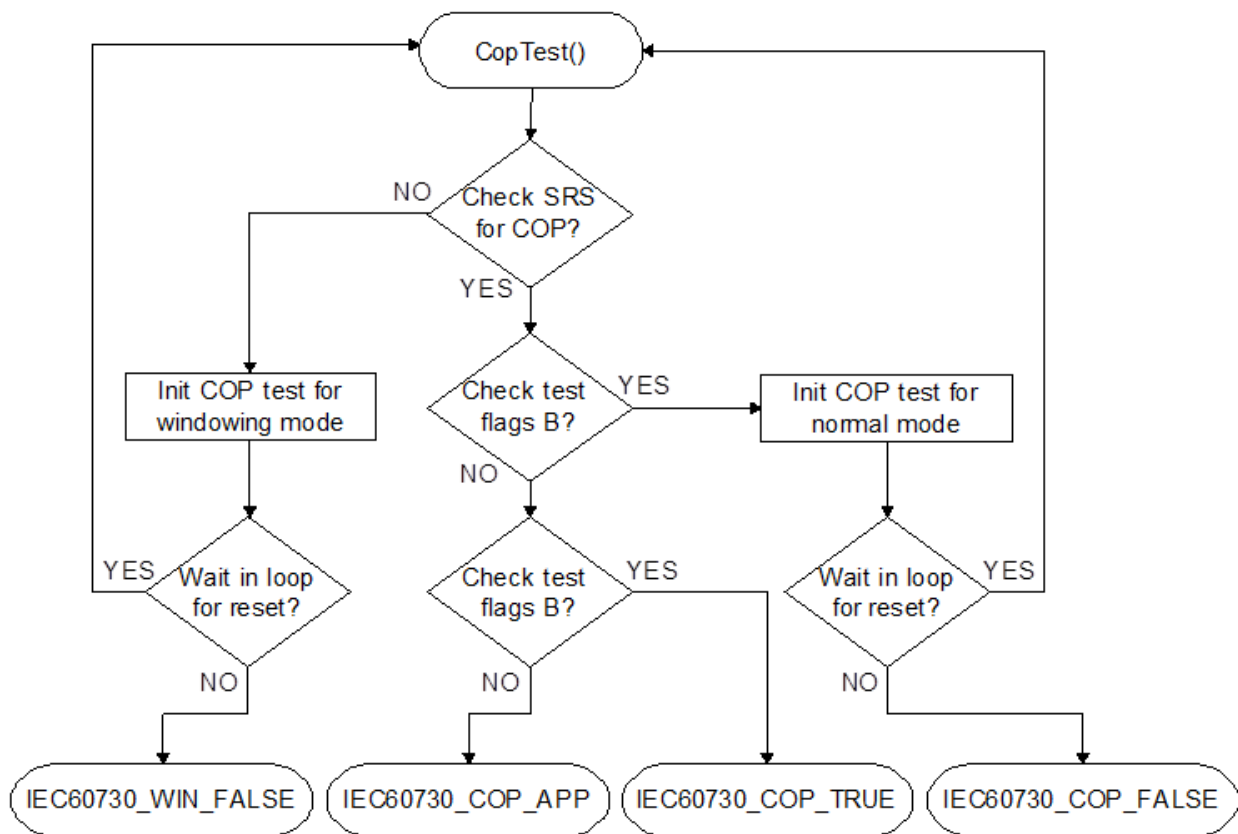


Figure 12. Algorithm of the COP module testing

The IEC60730B\_KINETIS\_CopTest routine can return the following results of test:

- IEC60730\_COP\_TRUE(0) – the test was carried out successfully
- IEC60730\_COP\_FALSE(1) – the test was carried out and COP is not working properly
- IEC60730\_WIN\_FALSE(2) – the test of the windowing mode has failed
- IEC60730\_COP\_APP(3) – the test was started by an application source

## 9 Flash memory test

Flash memory fault / error control technique is based on testing of a single word representing the contents of all words in the memory. During the self-test, a checksum is calculated from the same algorithm, and compared to the saved checksum. The saved checksum value must be placed in an independent section of memory. This technique recognizes all the “odd” errors and some of the “even” errors. A CRC (16-bit) signature of the invariable memory is a preferred method of ensuring that there is not even a single fault.

For a run-time calculation of the CRC signature, either a hardware CRC module can be used, or the CRC signature can be calculated using a software algorithm. The difference between calculating the CRC using a hardware module vs. using a software algorithm is that using the hardware CRC module is 12 times faster. Some of the Kinetis families do not include a hardware CRC module (Kinetis L).

The principle of the flash memory testing is shown in [Figure 13](#):

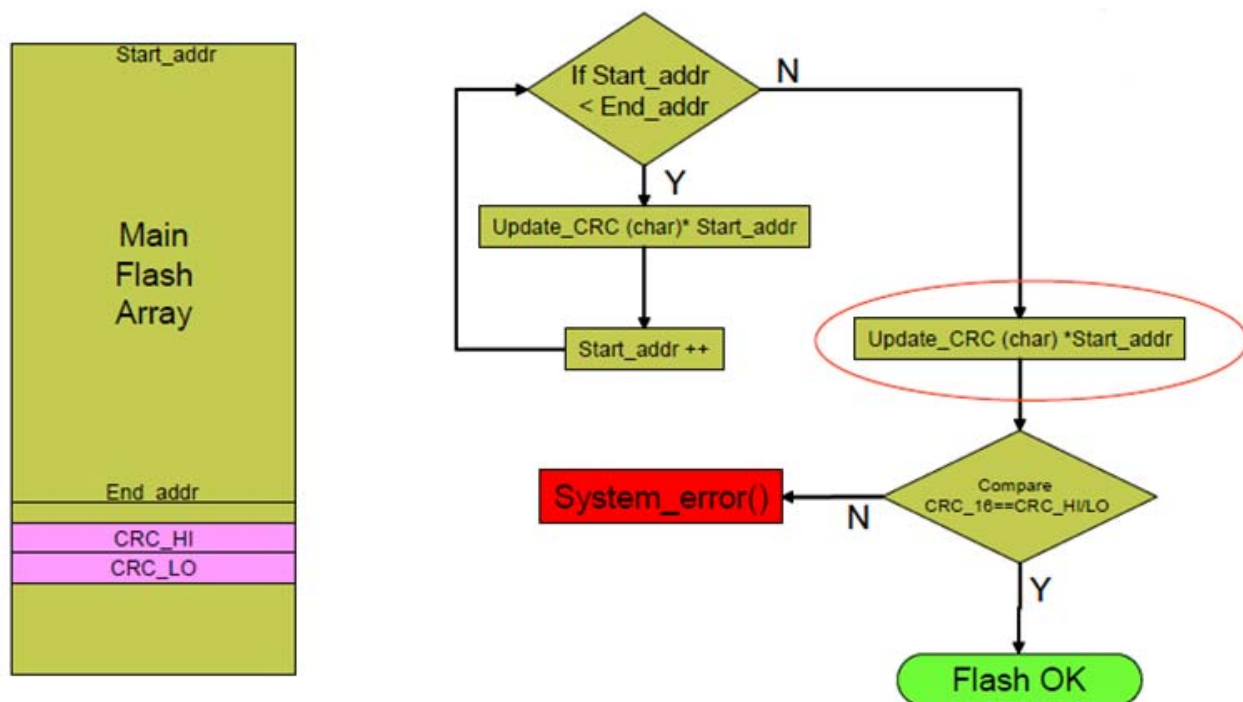


Figure 13. Algorithm of flash memory testing

Two methods of test function implementation can be used in the application.

The first method provides a complete flash memory test in the defined range of the flash memory. This test should be carried out before starting the application due to a long duration of the test. If only shorter memory sections are tested, it is possible to use the complete test in the run-time application.

- IEC60730B\_KINETIS\_FlashCRC16Test
  - This function calculates a CRC of the current flash memory segment and stores it into a defined RAM section. After that, the CRC is compared with the CRC value calculated by an external tool. This routine has to be used only as a run-time test. To calculate the CRC, a 16-bit CRC-CCIT (0x1021) is used. Prior to this test, the test init function must be called.

The second method of testing has to be used only in run-time of the application. The CRC signature is calculated per parts during the run-time, and saved into the RAM memory. If the whole range of flash memory is calculated, the value is compared to the complete CRC of the current segment. This function is not as time-consuming, and can be designed depending on the application. Before using the run-time function, initialization must be carried out.

- IEC60730B\_KINETIS\_FlashCRC16TestRuntimeInit
  - This function saves all input parameters (start address, size, golden CRC) into a dedicated RAM which is reserved for Flash memory test (file IEC60730\_B\_cfg.h). This routine must be called once prior to run-time testing function (IEC60730B\_KINETIS\_FlashCRC16TestRuntimeInit()).
- IEC60730B\_KINETIS\_FlashCRC16TestRuntime
  - This function calculates the CRC of the current flash memory block and stores it into a defined RAM section. After that, the CRC is compared to the CRC value calculated by an external tool. This routine has to be used only as a run-time test. The 16-bit CRC-CCIT (0x1021) is used for the CRC calculation. Prior to this test, the test init function IEC60730B\_KINETIS\_FlashCRC16TestRuntimeInit() must be called.

## 10 Clock test

The main principle of the clock testing is comparing the internal free-running timer to periodically-generated interrupts. Both peripherals are clocked by independent clocks. The internal system timer (SYSTICK) is clocked by internal core system clock. Periodical interrupt is clocked by the low-power oscillator (LPO) and it generates a periodic interrupt of 128 ms. After each period, the value from system timer is checked and compared to user-defined deviation.

Interrupts are pre-defined using real-time counter (RTC) and low-power timer (LPTMR). Another types of periodic interrupts with defined range can also be used. The clock testing is based on run-time testing, and the main test function can be called in the application anytime.

The initialization function must be called before the first clock checking.

- IEC60730B\_KINETIS\_ClockTestInit
  - This function provides initialization of real-time clock (RTC) or low-power timer (LPTMR) and system timer (SYSTICK) peripherals for the clock testing. It must be called prior to the clock self-test.

## Revision history

- IEC60730B\_ClockErrorCallback
  - This callback function can be placed into the user code to signalize the clock test error.
- IEC60730B\_KINETIS\_ClockTestIsr
  - This function provides calculating SYSTICK time ticks in defined range (RTC, LPTMR – 128 ms). This function also evaluates the results and stores the result value into the RAM.

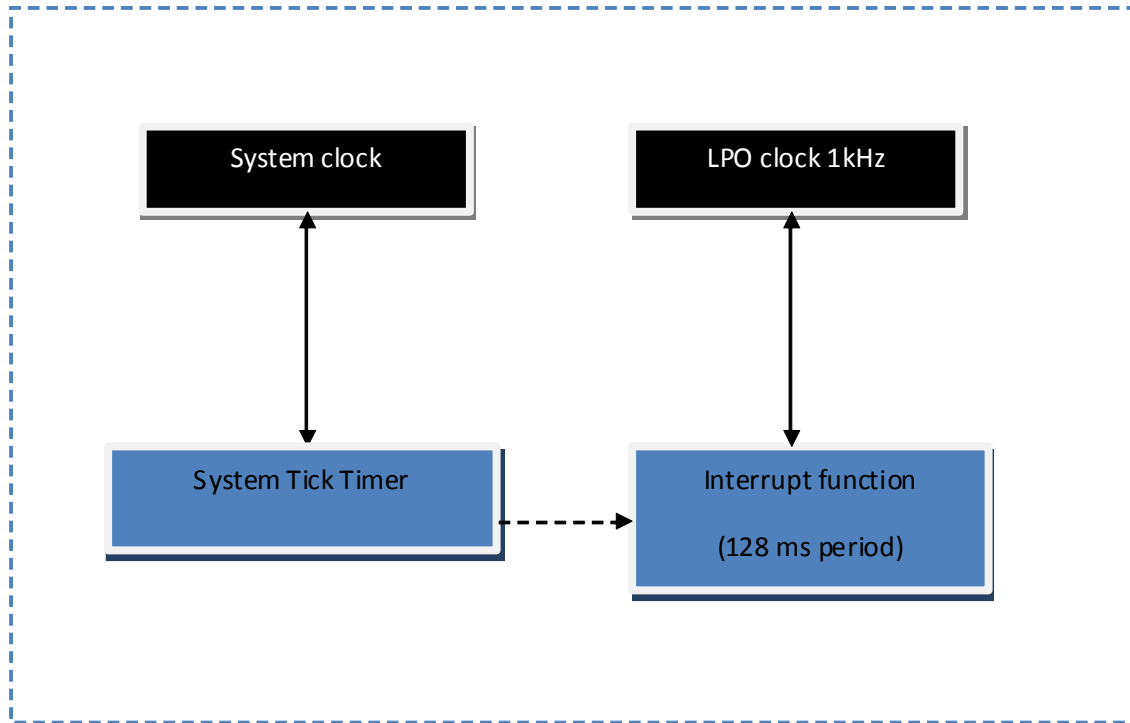


Figure 14. Clock test block schematic

## 11 Revision history

Table 3. Revision history

Rev. No.	Date	Change description
0	04/2014	Initial release
1	11/2014	New chapters added

**How to Reach Us:**

**Home Page:**  
[freescale.com](http://freescale.com)

**Web Support:**  
[freescale.com/support](http://freescale.com/support)

Information in this document is provided solely to enable system and software implementers to use Freescale 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.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale 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 Freescale 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. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: [freescale.com/SalesTermsandConditions](http://freescale.com/SalesTermsandConditions).

Freescale, the Freescale logo, and Kinetis are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners. ARM and Cortex are the registered trademarks of ARM Limited. ARM Logo is registered trademark of ARM Limited  
© 2014 Freescale Semiconductor, Inc.

