

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

**Watchdog 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, Test for MKE1xF devices added
11/2018	Jozef Sedlak	3.0	Release with new compilers support

<b>1</b>	<b>WATCHDOG TEST ARCHITECTURE.....</b>	<b>4</b>
<b>2</b>	<b>WATCHDOG TEST IN COMPLIANCE WITH IEC/UL STANDARDS.....</b>	<b>4</b>
<b>3</b>	<b>WATCHDOG TEST IMPLEMENTATION .....</b>	<b>5</b>
3.1	IEC60730B_CM4_CM7_WATCHDOG_TEST_SETUP().....	7
3.2	IEC60730B_CM4_CM7_WATCHDOG_TEST_SETUP_KE() .....	8
3.3	IEC60730B_CM4_CM7_WATCHDOG_TEST_SETUP_K32W() .....	9
3.4	IEC60730B_CM4_CM7_WATCHDOG_TEST_SETUP_RT().....	10
3.5	IEC60730B_CM4_CM7_WATCHDOG_TEST_CHECK() .....	11
3.6	IEC60730B_CM4_CM7_WATCHDOG_TEST_CHECK_KE() .....	12
3.7	IEC60730B_CM4_CM7_WATCHDOG_TEST_CHECK_K32W().....	13
3.8	IEC60730B_CM4_CM7_WATCHDOG_TEST_CHECK_RT() .....	14
<b>4</b>	<b>WATCHDOG TEST MODULE TEST CONCEPT .....</b>	<b>15</b>
<b>6-</b>	<b>WATCHDOG TEST VALIDATION.....</b>	<b>15</b>

## 1 Watchdog Test Architecture

---

The watchdog test provides testing of the watchdog timer functionality. The test checks whether the watchdog timer is able to cause a reset and whether the reset happens at the expected time. Before the start of the test, the watchdog must be configured for use in the respective application. The next step before the test is the setup of the independent timer (LPTMR or GPT), which is used for watchdog timeout comparison. The first function for watchdog testing is called after that. This function refreshes the watchdog timer, activates the timer and captures the its counter value during an endless loop. This function should be called only once after power-on reset (POR). After the watchdog reset, the second function must be called. This function should be called after every reset except the POR. This function checks whether the captured timer counter value corresponds to the expected watchdog timeout value. The next check is whether the number of watchdog resets does not exceed the limit value. The user can choose what action must be made after an incorrect result. Due to safety requirements, the user has limited options for choosing the clock source for the watchdog and also for the timer. The first condition is that the watchdog timer clock cannot be the same as the watchdog bus interface clock. Check the device reference manual for watchdog timer clock source options. The second condition is that the watchdog timer clock cannot be the same as the timer clock.

## 2 Watchdog Test in compliance with IEC/UL standards

---

Watchdog test is not directly specified in IEC60730 - annex H table, but it partially fulfils safety requirements according to IEC 60730-1, IEC 60335, UL 60730 and UL 1998 standards as described in the following table:

**Table 1. Watchdog Test in Compliance with the Standards**

Test	Component	Fault/Error	SW / HW Class	Acceptable Measures
Watchdog test	3. Clock	Wrong frequency	B/R.1	Frequency monitoring
Watchdog test	8. Monitoring devices and comparators	Any output outside the static and dynamic functional specification	B/R.1	Tested monitoring

### 3 Watchdog Test Implementation

---

Test functions for the watchdog are placed in *IEC60730\_B\_CM4\_CM7\_wdg.c*. The header file is *IEC60730\_B\_CM4\_CM7\_wdg.h*. *IEC60730\_B\_CM4\_CM7.h* and *asm\_mac\_common.h* are files that need to be placed in the application as well.

The user must have available space in RAM memory, which is not corrupted after non-POR.

This memory is used for the user-defined variable of *WD\_Test\_Str* type, which is the structure with three members. It is defined in *IEC60730\_B\_CM4\_CM7\_wdg.h*.

The important condition for performing the watchdog test is to have configured watchdog module and the device timer before the beginning of the test.

Watchdog module in MKE1xF devices differs from the watchdog implemented in another CM4 core-based devices. The differences in implementation are explained below.

#### Implementation for MKV3x, MKV4x, MKV5x devices:

*IEC60730B\_CM4\_CM7\_watchdog\_test\_setup ()*

*IEC60730B\_CM4\_CM7\_watchdog\_test\_check ()*

#### Implementation for MKE1xF devices:

*IEC60730B\_CM4\_CM7\_watchdog\_test\_setup\_ke ()*

*IEC60730B\_CM4\_CM7\_watchdog\_test\_check\_ke ()*

#### Implementation for MK32W0x/cm4 devices:

*IEC60730B\_CM4\_CM7\_watchdog\_test\_setup\_k32w ()*

*IEC60730B\_CM4\_CM7\_watchdog\_test\_check\_k32w ()*

#### Implementation for MIMXRT devices:

*IEC60730B\_CM4\_CM7\_watchdog\_test\_setup\_RT ()*

*IEC60730B\_CM4\_CM7\_watchdog\_test\_check\_RT ()*

- GPT is used as independent timer

#### Structure defined in *IEC60730\_B\_CM4\_CM7\_wdg.h*:

```
typedef struct {
    unsigned long counter;
    unsigned long resets;
    unsigned long wd_test_uncomplete_flag;
} WD_Test_Str;
```

Handling of the functions must be ensured by the user. To identify what has been the source of the reset, the reset control module (RCM, MSMC, SRC) must be used. The common configuration is that if an unwanted result has been founded by the “check” function, the program stays stacked in an endless loop in the function. This causes the application to stay in the loop of watchdog resets. By entering the zero as fourth input value of check function, the endless loop is not activated. In that case, the user must ensure that the application is put into safe state.

Below is an example of the watchdog test implementation (MKV3x, MKV4x, MKV5x):

```
#include "IEC60730_B_CM4_CM7.h"
#define WATCHDOG_ON
#define Watchdog_refresh WDOG_REFRESH = 0xA602; WDOG_REFRESH = 0xB480

extern unsigned long WD_TEST_BACKUP; /* from Linker configuration file */
const unsigned long WD_backup_address = (unsigned long)&WD_TEST_BACKUP;

#define WATCHDOG_TEST_VARIABLES ((WD_Test_Str *) WD_backup_address)

#define WD_TEST_LIMIT_HIGH    3400
#define WD_TEST_LIMIT_LOW    3000
#define ENDLESS_LOOP_ENABLE    1 /* set 1 or 0 */
#define WATCHDOG_RESETS_LIMIT 1000
#define WATCHDOG_TIMEOUT_VALUE 100

MCG->C1 |= MCG_C1_IRCLKEN_MASK; /* MCGIRCLK active */
MCG->C2 &= (~MCG_C2_IRCS_MASK); /* slow reference clock selected */
SIM->SCGC5 |= SIM_SCGC5_LPTMR_MASK; /* enable clock gate to LPTMR */
LPTMR0->CSR = 0; /* time counter mode */
LPTMR0->CSR = LPTMR_CSR_TCF_MASK | LPTMR_CSR_TFC_MASK; /* CNR reset on overflow */
LPTMR0->PSR |= LPTMR_PSR_PBYP_MASK; /* prescaler bypassed */
LPTMR0->PSR &= (~LPTMR_PSR_PCS_MASK); /* clear prescaler clock */
LPTMR0->PSR |= LPTMR_PSR_PCS(0); /* select the clock input */
LPTMR0->CMR = 0; /* clear the compare register */
LPTMR0->CSR |= LPTMR_CSR_TEN_MASK; /* enable timer */

WatchdogEnable(WDOG, WATCHDOG_TIMEOUT_VALUE);

if(RCM_SRS0_POR_MASK==( RCM_SRS0_POR_MASK &RCM->SRS0)) /* if POR reset */
{
    IEC60730B_CM4_CM7_watchdog_test_setup(WATCHDOG_TEST_VARIABLES);
}
```

```
if(RCM_SRS0_POR_MASK!=(RCM_SRS0_POR_MASK &RCM->SRS0)) /* if non-POR reset */
{
    IEC60730B_CM4_CM7_watchdog_test_check(WD_TEST_LIMIT_HIGH, WD_TEST_LIMIT_LOW, \
    WATCHDOG_RESETS_LIMIT, ENDLESS_LOOP_ENABLE, WATCHDOG_TEST_VARIABLES);
}
```

### 3.1 IEC60730B\_CM4\_CM7\_watchdog\_test\_setup()

Can be used for MKV3x, MKV4x, MKV5x devices. This function clears the reset counter, which is a member of the WD\_Test\_Str structure. It refreshes the watchdog to start counting from zero. It starts the LPTMR, which must be configured before the function call occurs. Within the waiting endless loop, the value from the LPTMR is periodically stored in the reserved area in the RAM.

#### **Function prototype:**

void IEC60730B\_CM4\_CM7\_watchdog\_test\_setup (WD\_Test\_Str\* pWatchdogBackup).

#### **Function inputs:**

pWatchdogBackup - pointer to structure of WD\_Test\_Str type, defined in the header file

#### **Function output:**

Void. The function does not return a value.

#### **Function performance:**

The function duration depends on the watchdog counter timeout value. The function size is 50 bytes.<sup>2</sup>

#### **Calling restrictions:**

The watchdog timer and the LPTMR must be configured correctly. The variable of WD\_Test\_Str type must be declared and placed in RAM area that is not overwritten during application startup. Interrupts should be disabled.

### 3.2 IEC60730B\_CM4\_CM7\_watchdog\_test\_setup\_ke()

Can be used for MKE1xF devices. This function clears the reset counter, which is a member of the WD\_Test\_Str structure. It refreshes the watchdog to start counting from zero. It starts the LPTMR, which must be configured before the function call occurs. Within the waiting endless loop, the value from the LPTMR is periodically stored in the reserved area in the RAM.

**Function prototype:**

```
void IEC60730B_CM4_CM7_watchdog_test_setup_ke (WD_Test_Str* pWatchdogBackup).
```

**Function inputs:**

pWatchdogBackup - pointer to structure of WD\_Test\_Str type, defined in the header file

**Function output:**

Void. The function does not return a value.

**Function performance:**

The function duration depends on the watchdog counter timeout value. The function size is 40 bytes.<sup>2</sup>

**Calling restrictions:**

The watchdog timer and the LPTMR must be configured correctly. The variable of WD\_Test\_Str type must be declared and placed into RAM area that is not overwritten during application startup. Interrupts should be disabled.



### 3.3 IEC60730B\_CM4\_CM7\_watchdog\_test\_setup\_k32w()

Can be used for MK32W0x/cm4 devices. This function clears the reset counter, which is a member of the WD\_Test\_Str structure. It refreshes the watchdog to start counting from zero. It starts the LPTMR, which must be configured before the function call occurs. Within the waiting endless loop, the value from the LPTMR is periodically stored in the reserved area in the RAM.

#### **Function prototype:**

```
void IEC60730B_CM4_CM7_watchdog_test_setup_ke (WD_Test_Str* pWatchdogBackup,  
unsigned long* pWDOG, unsigned long* pLPTMR).
```

#### **Function inputs:**

pWatchdogBackup - pointer to structure of WD\_Test\_Str type, defined in the header file

pWDOG – pointer to Watchdog module base address

pLPTMR – pointer to LPTMR module base address

#### **Function output:**

Void. This function does not return a value.

#### **Function performance:**

The function duration depends on the watchdog counter timeout value. The function size is 36 bytes.<sup>2</sup>

#### **Calling restrictions:**

The watchdog timer and the LPTMR must be configured correctly. The variable of WD\_Test\_Str type must be declared and placed into RAM area that is not overwritten during application startup. Interrupts should be disabled.

### 3.4 IEC60730B\_CM4\_CM7\_watchdog\_test\_setup\_RT()

Can be used for MIMXRT devices. This function clears the reset counter, which is a member of the WD\_Test\_Str structure. It refreshes the watchdog to start counting from zero. It starts the GPT, which must be configured before the function call occurs. Within the waiting endless loop, the value from the GPT is periodically stored in the reserved area in the RAM.

**Function prototype:**

```
void IEC60730B_CM4_CM7_watchdog_test_setup_ke (WD_Test_Str* pWatchdogBackup).
```

**Function inputs:**

pWatchdogBackup - pointer to structure of WD\_Test\_Str type, defined in the header file

**Function output:**

Void. This function does not return a value.

**Function performance:**

The function duration depends on the watchdog counter timeout value. The function size is 30 bytes.<sup>2</sup>

**Calling restrictions:**

Watchdog test works only if the application is loaded from Flash memory. The watchdog timer and the GPT must be configured correctly. The variable of WD\_Test\_Str type must be declared and placed into RAM area that is not overwritten during application startup. Interrupts should be disabled.

### 3.5 IEC60730B\_CM4\_CM7\_watchdog\_test\_check()

This function compares the captured value of the LPTMR counter with pre-calculated limit values and checks whether the watchdog reset counter overflows. In case that the function is called after non-watchdog reset, the `wd_test_uncomplete_flag` is set. With the `endless_loop_enable` parameter, endless loop within the function is enabled or disabled, (by setting it to 1 or 0 value). If enabled, the function ends up in endless loop under these conditions:

- entering after non-watchdog or non-POR reset
- counter from watchdog test does not fit the limit values
- watchdog resets exceed the defined limit value

#### **Function prototype:**

```
void IEC60730B_CM4_CM7_watchdog_test_check(unsigned long limit_high, unsigned long limit_low,  
unsigned long resets_limit, unsigned long endless_loop_enable, WD_Test_Str* pWatchdogBackup);
```

#### **Function inputs:**

`limit_high`: pre-calculated upper limit value of LPTMR counter in Watchdog test

`limit_low`: pre-calculated low limit value of LPTMR counter in Watchdog test

`resets_limit`: defined limit of Watchdog resets

`endless loop enable`: if this is 1, endless loop is activated in the function

`pWatchdogBackup` - pointer to structure of `WD_Test_Str` type, defined in the header file

#### **Function output:**

Void. The function does not return a value.

#### **Function performance:**

Function duration is approximately 61 cycles (0.76  $\mu$ s)<sup>1</sup>. Function size is 76 bytes.<sup>2</sup>

#### **Calling restrictions:**

The respective setup function must be executed first.

### 3.6 IEC60730B\_CM4\_CM7\_watchdog\_test\_check\_ke()

This function compares the captured value of the LPTMR counter with pre-calculated limit values and checks whether the watchdog reset counter overflows. In case that the function is called after non-watchdog reset, the `wd_test_uncomplete_flag` is set. With the `endless_loop_enable` parameter, endless loop within the function is enabled or disabled, (by setting it to 1 or 0 value). If enabled, the function ends up in endless loop under these conditions:

- entering after non-watchdog or non-POR reset
- counter from watchdog test does not fit the limit values
- watchdog resets exceed the defined limit value

#### **Function prototype:**

```
void IEC60730B_CM4_CM7_watchdog_test_check_ke(unsigned long limit_high, unsigned long limit_low, unsigned long resets_limit, unsigned long endless_loop_enable, WD_Test_Str* pWatchdogBackup);
```

#### **Function inputs:**

`limit_high`: pre-calculated upper limit value of LPTMR counter in Watchdog test

`limit_low`: pre-calculated low limit value of LPTMR counter in Watchdog test

`resets_limit`: defined limit of Watchdog resets

`endless loop enable`: if this is 1, endless loop is activated in the function

`pWatchdogBackup` - pointer to structure of `WD_Test_Str` type, defined in the header file

#### **Function output:**

Void. The function does not return a value.

#### **Function performance:**

Function duration is approximately 63 cycles (0.63  $\mu$ s)<sup>3</sup>. Function size is 74 bytes.<sup>2</sup>

#### **Calling restrictions:**

The respective setup function must be executed first.

### 3.7 IEC60730B\_CM4\_CM7\_watchdog\_test\_check\_k32w()

This function compares the captured value of the LPTMR counter with pre-calculated limit values and checks whether the watchdog reset counter overflows. In case that the function is called after non-watchdog reset, the `wd_test_uncomplete_flag` is set. With the `endless_loop_enable` parameter, endless loop within the function is enabled or disabled, (by setting it to 1 or 0 value). If enabled, the function ends up in endless loop under these conditions:

- entering after non-watchdog or non-POR reset
- counter from watchdog test does not fit the limit values
- watchdog resets exceed the defined limit value

#### **Function prototype:**

```
void IEC60730B_CM4_CM7_watchdog_test_check_k32w(unsigned long limit_high, unsigned long limit_low, unsigned long resets_limit, unsigned long endless_loop_enable, WD_Test_Str* pWatchdogBackup);
```

#### **Function inputs:**

`limit_high`: pre-calculated upper limit value of LPTMR counter in Watchdog test

`limit_low`: pre-calculated low limit value of LPTMR counter in Watchdog test

`resets_limit`: defined limit of Watchdog resets

`endless loop enable`: if this is 1, endless loop is activated in the function

`pWatchdogBackup` - pointer to structure of `WD_Test_Str` type, defined in the header file

#### **Function output:**

Void. The function does not return a value.

#### **Function performance:**

Function duration is approximately 82 cycles (1.71  $\mu$ s)<sup>4</sup>. Function size is 74 bytes.<sup>2</sup>

#### **Calling restrictions:**

The respective setup function must be executed first.

### 3.8 IEC60730B\_CM4\_CM7\_watchdog\_test\_check\_RT()

This function compares the captured value of the GPT counter with pre-calculated limit values and checks whether the watchdog reset counter overflows. In case that the function is called after non-watchdog reset, the `wd_test_uncomplete_flag` is set. With the `endless_loop_enable` parameter, endless loop within the function is enabled or disabled, (by setting it to 1 or 0 value). If enabled, the function ends up in endless loop under these conditions:

- entering after non-watchdog or non-POR reset
- counter from watchdog test does not fit the limit values
- watchdog resets exceed the defined limit value

#### **Function prototype:**

```
void IEC60730B_CM4_CM7_watchdog_test_check_RT(unsigned long limit_high, unsigned long limit_low, unsigned long resets_limit, unsigned long endless_loop_enable, WD_Test_Str* pWatchdogBackup);
```

#### **Function inputs:**

`limit_high`: pre-calculated upper limit value of LPTMR counter in Watchdog test

`limit_low`: pre-calculated low limit value of LPTMR counter in Watchdog test

`resets_limit`: defined limit of Watchdog resets

`endless loop enable`: if this is 1, endless loop is activated in the function

`pWatchdogBackup` - pointer to structure of `WD_Test_Str` type, defined in the header file

#### **Function output:**

Void. The function does not return a value.

#### **Function performance:**

Function duration is approximately 22 cycles (0.92  $\mu$ s)<sup>5</sup>. Function size is 80 bytes.<sup>2</sup>

#### **Calling restrictions:**

Watchdog test works only if the application is loaded from Flash memory. The respective setup function must be executed first.

- 
- 1- The number of cycles and the execution time were measured with the use of MKV31 / 80 MHz CPU clock / 20 MHz flash clock
  - 2- Function compiled by IAR v8.22.2
  - 3- The number of cycles and the execution time were measured with the use of MKE18 / 100 MHz CPU clock / 25 MHz flash clock
  - 4- The number of cycles and the execution time were measured with the use of MK32W / 48 MHz CPU clock / 24 MHz flash clock
  - 5- The number of cycles and the execution time were measured with the use of MIMXRT / 600 MHz CPU clock / 75 MHz flash clock

## 4 Watchdog Test Module test concept

---

Content moved to TestReport\_IEC60730B\_Watchdog\_rev3\_0 document

## 6- Watchdog Test Validation

---

Content moved to TestReport\_IEC60730B\_Watchdog\_rev3\_0 document

**Table 2. Validation**

<b>Date</b>	<b>Validated by</b>	<b>Validated Revision of Document</b>	<b>Validated Version of Source Code</b>	<b>Validation Result</b>
11/2015	Jaroslav Lepka	1.0	1.0	P
11/2016	Pavel Sustek	1.1	1.1	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](http://nxp.com)

**Web Support:**

[nxp.com/support](http://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](http://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.

