Document identifier: IEC60730BIMX8MXEUG
Rev. 3, 07/2021

i.MX8M Safety Example

Contents

Chapter 1
IEC60730B Safety library example user's guide

For easier development of the IEC60730B application, the library also provides the example code. This example is distributed through the MCUXpresso SDK website. This example user's guide describes how to set the hardware correctly and how to use the example code with the IEC60730B Safety library.

The library user's guide is the main documentation for IEC60730B. It is also part of this package and you can download it at www.nxp.com/IEC60730.

Chapter 2
Hardware settings

This chapter describes how to set up the hardware of the evaulation board. The MCU peripherals' setup is described later on.

The IEC60730B library example for the I.MX8mx family supports the following development boards:

- EVKmimx8mm
- EVKmimx8mn

To run the IEC60730B example application, it is neccessary to make some hardware settings. For the default configuration of your development board, see the corresponding board's user manual at www.nxp.com.

2.1 EVK-MIMX8MM (Mini)

The hardware requirements are as follows:

- i.MX8M Mini development board (EVK-MIMX8MM)
- J-Link debug probe
- USB-C cable (12-V power supply)
- USB-C cable (download to ROM memory)
- Micro USB cable (debug print)

Debugger:

The default debugger in the example project is set to J-Link.

FreeMASTER

FreeMASTER communication is used via an external J-Link plugin.

The hardware settings are as follows:

1. Connect the 12-V power supply (J302 USB port) and the J-Link debug probe to the board and switch the SW101 switch to power on the board.
2. Connect a USB-C cable between the host PC and the J301 USB port (this port allows downloading to the ROM memory).
3. If a debug print is needed, connect a micro USB cable between the host PC and the J901 USB port on the target board. Then open the serial terminal with the following settings:
   - 115200 baud rate
   - 8 data bits
   - No parity
   - One stop bit
   - No flow control
4. Set the SW1101 switch to value "01110010" and SW1102 to value "00101010" to boot from the eMMC.

See www.nxp.com/imx8mminievk for more information.

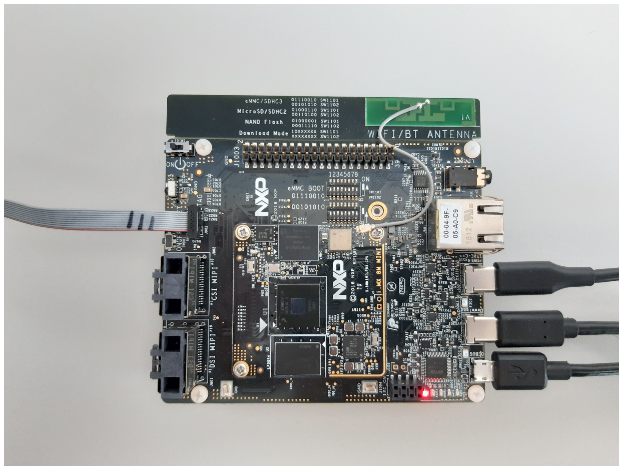

Figure 1. Hardware connection of EVK-MIMX8MM

2.2 EVK-MIMX8MN (Nano)

The hardware requirements are as follows:

- i.MX8M Nano development board (EVK-MIMX8MN)
- J-Link debug probe
- USB-C cable (12-V power supply)
- USB-C cable (download to ROM memory)
- Micro USB cable (debug print)

Debugger:

The default debugger in the example project is set to J-Link.

FreeMASTER

FreeMASTER communication is used via an external J-Link plugin.

The hardware settings are as follows:

1. Connect the 12-V power supply (J302 USB port) and the J-Link debug probe to the board and switch the SW101 switch to power on the board.
2. Connect a USB-C cable between the host PC and the J301 USB port (this port allows downloading to the ROM memory).
3. If a debug print is needed, connect a micro USB cable between the host PC and the J901 USB port on the target board. Then open a serial terminal with the following settings:
    - 115200 baud rate
    - 8 data bits
    - No parity
    - One stop bit
    - No flow control
4. Set switch SW1101 to "0100" to boot from the eMMC.

See www.nxp.com/ imx8mnanoevk for more information.

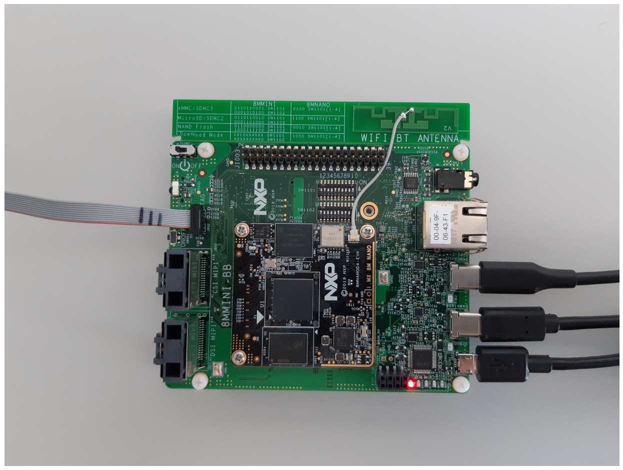

Figure 2. Hardware connection of EVK-MIMX8MN

Chapter 3
File structure

Safety is only a small part of the whole SDK package for your device. The IEC60730 library and examples are located in the middleware and in the board folders. The IEC60730 library is independent of the SDK and can be used stand-alone.

3.1 Library source files location

The library source files are in the *middleware/safety_iec60730b/safety/v4_2* folder in the SDK package.

The folder has the following structure:

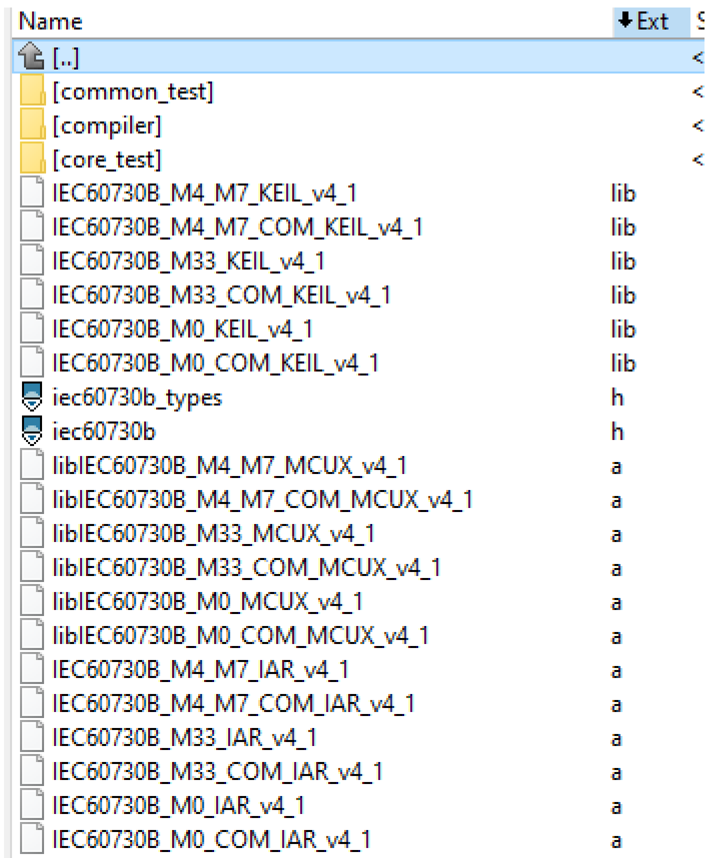

Figure 3. Folder structure

Where:

- The *common_test* folder contains the source files for the peripheral test – this is a common cross core. These tests are compiled to library *libIEC60730B_<core>_COM_<compiler>_<version>.a*.
- The *compiler* folder contains compiler support files.
- The *core_test* folder contains the source files for the core-dependent test. These tests are compiled to library *libIEC60730B_<core>_<compiler>_<version>.a*.
- *iec60730b.h* is the main library header file.
- *iec60730b_types.h* is the header file with the necessary defines for the library.

The folder also contains binary **.lib* files, which are compiled for the IAR, Keil, and MCUXpresso IDEs (see the release notes for details).

3.2 Example of library handling code

The library-handling code and the example aplication are separate from the library file. The example source files and other SDK examples are at this path:

boards/<your board>/demo_apps/safety_iec60730b/

The safety example code is shown in Figure 4.

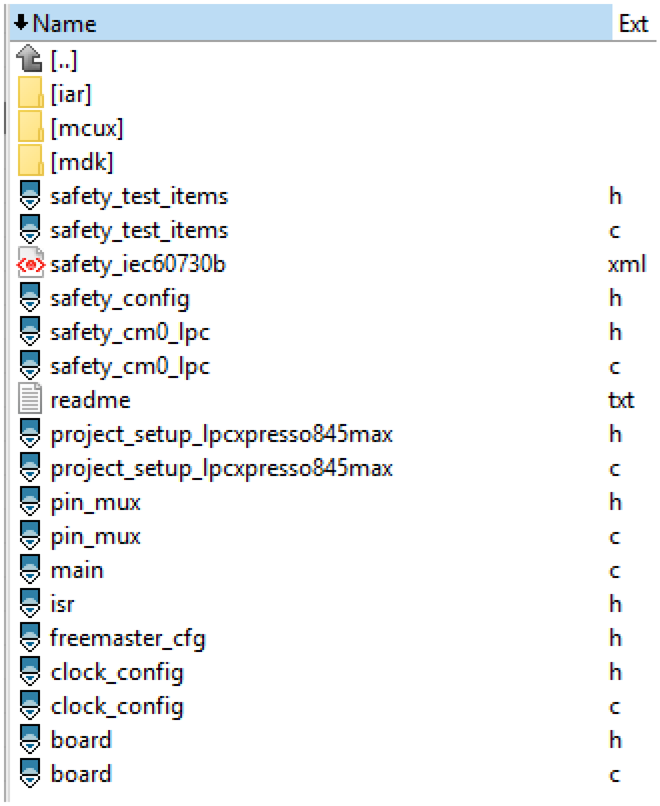

Figure 4. Example of project structure in example folder

This folder contains the example source file and three folders for the IDE project file:

- *iar*
- *mcux*
- *mdk*

The following files are generated by the MCUXpresso configuration tool:

- *clock_config.h*
- *clock_config.c*
- *pin_mux.c*
- *pin_mux.h*

Other files are used only for safety examples and their contents are described in the next chapter.

Chapter 4
Example application

The structure of the example is common in all supported IDEs (IAR, Keil, MCUXpresso).

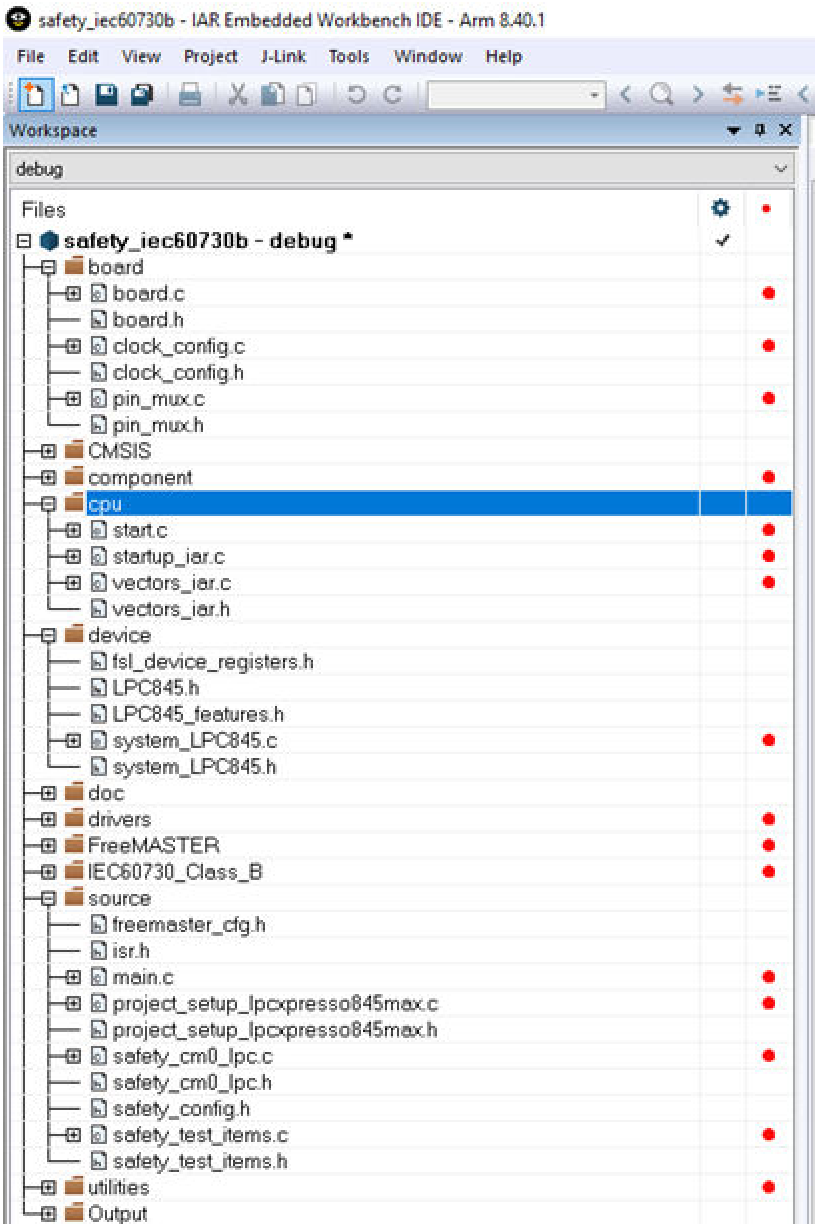

Figure 5. IAR example application structure

The project contains the CMSIS, SDK, library, and safety example-related folders.

The safety-related folders are the following:

- *Board* – this folder contains the files related to the board used (*clock_config.h*, *pin_config.h*, *board.h*, and so on).
- *CPU* – this folder contains the startup code and vectors table.
- *IEC60730_Class_B* – files for the IEC60730B Safety library.

- *Source* – source file for the safety example (see the next explanation).

The example of project hiearchy is shown in Figure 6.

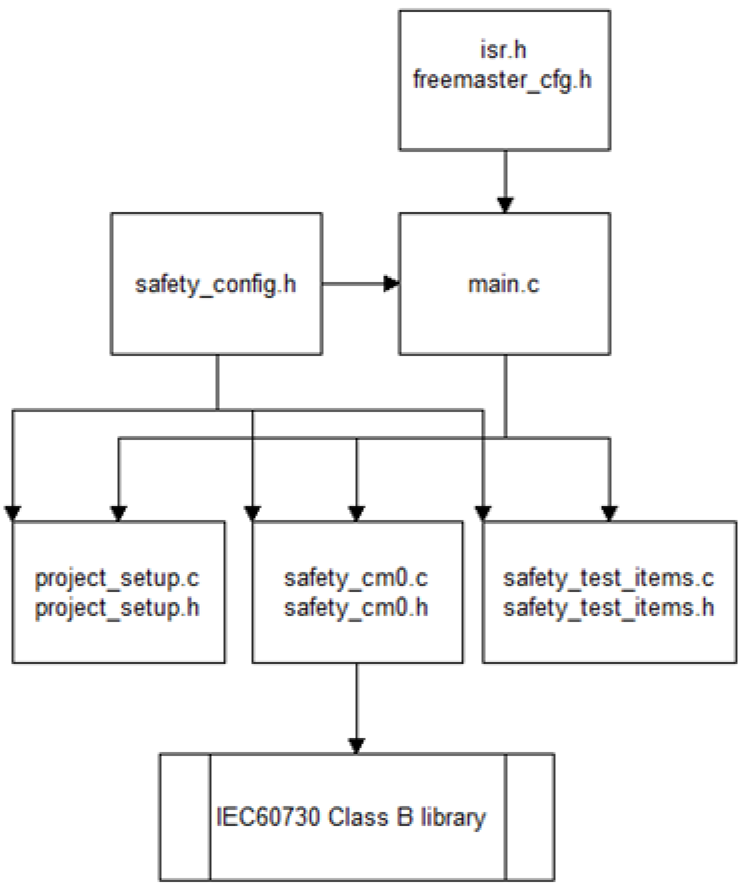

Figure 6. Example of project hiearchy

Figure 6 shows that the functions in the *project_setup.c* file are called from the *main.c* file. The library-handling functions are located in the file and also called from the *main.c* file.

The main example application header file *safety_config.h* contains all definitions for running the safety test in examples. The *safety_test_items.c* file declares the structures for the DIO (or TSI) safety test. The *project_setup_<your_board>.c* file contains the setup functions (clock, port, UART, and so on). The file contains the handling function for safety routines from the IEC60730B library and also the test-initialization function for safety.

4.1 How to open the project

IAR IDE

Open the project file located at *boards/<your_board>/demo_apps/safety_iec60730b/iar/safety_iec60730b.eww*.

Arm Keil IDE

Open the project file located at *boards/<your_board>/demo_apps/safety_iec60730b/mdk/safety_iec60730b.uvprojx*.

MCUXpresso IDE

Firstly, drag and drop the *<name_of_the_package>.zip* package into the MCUXpresso IDE (into the "Installed SDKs" tab). Secondly, import the SDK example (safety_iec60730b).

If you are not familiar with the MCUXpresso IDE yet, see *docs/Getting Started with MCUXpresso SDK for <your_board>.pdf* ("Build an example application" section).

4.2 Build configurations

debug_ram

This configuration is targeted to be executed from the RAM memory. Thus, some safety tests must be turned off because the ROM memory is not used (turn on/off a particular test in the beginning of the "*safety_config.h*" file).

1. Turn off the flash test, program counter test, and watchdog test.
2. Build the project (F7).
3. Download and debug (Ctrl+D).
4. Press F5 to run.
5. Check the terminal output: "Hello safety world!".

debug_flash

This configuration is targeted to be executed from the ROM memory. All safety tests can be turned on at the beginning of the *safety_config.h* file.

The flash loader is not supported for the Nano device in the IAR IDE. Thus, the UUU tool must be used to download the binary image into the ROM QSPI memory. The following steps are required to download the image to the ROM memory:

- Set the device to the download mode: switch SW1101 to value "1000".
- Setup the bootloader: check the Device Manager. Two new COM ports should appear. Open both COM ports in the serial terminal (with settings mentioned above). The COM port with the higher number shows the output of the bootloader. The other COM port shows the output of the application.
- Open the Windows OS command prompt and change the directory to *<PACKAGE_DIR>\example\boards\<your_board>\demo_apps\safety_iec60730b\iar*.
- Run the "uuu -b emmc uboot_emmc.bin" command and wait a few seconds. The output of the bootloader terminal should look as follows:

```
MMC write: dev # 1, block # 0, count 2764 ... 2764 blocks written: OK
Writing 'bootloader' DONE!
Detect USB boot. Will enter fastboot mode!
Detect USB boot. Will enter fastboot mode!
```

- Turn off the board (SW101).
- Set the device to boot from eMMC:
  — i.MX8M Mini: switch SW1101 to value "01110010" and SW1102 to value "00101010".
  — i.MX8M Nano: switch SW1101 to value "0100".
- Turn on the board (SW101).
- Open the bootloader terminal and wait a few seconds. When "u-boot=> " appears, run the following commands in a sequence:

```
env set -f bootcmd "mmc dev ${mmcdev}; if mmc rescan; then if run loadbootscript;
then run bootscript; else if run loadimage; then run mmcboot; else sf probe; bootaux
0x8000000;; fi; fi; else booti ${loadaddr} - ${fdt_addr}; fi"
```

```
env save
reset
```

- The bootlader is now located in the eMMC memory and set to boot automatically from the QSPI flash after the reset.
- Run the "fastboot usb" command (the device will wait for an input on the USB port).
- Build the project in the IAR IDE (press the "F7" key). The output is the *example\boards\<your_board>\demo_apps\safety_iec60730b\iar\debug_flash\Exe\dev_safety_iec60730b.bin* file.
- Download the built image into the QSPI flash: Navigate to the *<PACKAGE_DIR>\example\boards\<your_board>\demo_apps\safety_iec60730b\iar* directory in the Windows OS command prompt.
- Run the "uuu -b qspi uboot_qspi.bin .\debug_flash\Exe\dev_safety_iec60730b.bin" command. Wait a few seconds. The output of the Windows OS command prompt should look as follows:

```
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.2.135-0-gacaf035
Success 1 Failure 0
1:11 6/ 6 [Done ] FB: done
```

- The output of the bootloader terminal should look as follows:

```
downloading of 18532 bytes finished
SF: Detected n25q256a with page size 256 Bytes, erase size 4 KiB, total 32 MiB
SF: 20480 bytes @ 0x0 Erased: OK
device 0 offset 0x0, size 0x4864
SF: 18532 bytes @ 0x0 Written: OK
```

- The image is now downloaded into the QSPI flash.
- Turn the board off and back on again.
- Check whether "Hello safety world!" is printed in the serial terminal.
- Debugging from the IAR IDE is possible only in the Attach mode (debugging without downloading).

4.3 Example settings - safety_config.h

The main example settings header file is *safety_config.h*. The neccessary macros for the safety example are defined in this file.

The "switch macros", which enable the user to turn off the calling of the safety test, are defined in the beginning. When starting, turn off the FLASH test and the WDOG test. On LPC devices, turn off also the Clock test.

```
/* This macro enables infinity while loop in the SafetyErrorHandling() function */
#define SAFETY_ERROR_ACTION 1
/* TEST SWITCHES - for debugging, it is better to turn the FLASH and WDOG tests OFF. */
#define ADC_TEST_ENABLED 1
#define CLOCK_TEST_ENABLED 1
#define DIO_TEST_ENABLED 1
#define FLASH_TEST_ENABLED 1
#define RAM_TEST_ENABLED 1
```

```
#define PC_TEST_ENABLED 1
#define WATCHDOG_ENABLED 1
#define FMSTR_SERIAL_ENABLE 1
```

Other defines are used to configure the safety test as a parameter to a function or to fill structures.

4.4 safety_test_items.c file

The *safety_test_items.c* and *.h* files are the configuration files for the DIO test.

The file contains the *fs_dio_test_<platform>_t* list of structures. The pointers to these structures are collected in the *dio_safety_test_items[]* array, which is used in the example application.

4.5 Source file - safety_.c/.h

The source file and the corresponding **.h* file contain a library handling function. Each function contains a detection. If a safety error ocurrs, the *SafetyErrorHandling()* function is called.

Chapter 5
Running example

For the first run of the example on your hardware, it is recomended to turn off Flash, WDOG, Clock, AIO, and DIO test. In the next step, turn on step by step.

When the WDOG is turned off and a safety error happens, the example stays in an endless loop.

5.1 Post-build CRC calculation

The post-build CRC calculation can be used in several ways, depending on the IDE's built-in options. In IDEs that do not have the built-in options, use the SRecord tool.

SRecord is a standalone utility for memory manipulation. This utility and all information about it are available at Peter Miller's http://srecord.sourceforge.net/ webpage.

In the SDK package, the SRecord tool is in the *<sdk_pack>/tools/srecord* folder.

In the IEC60730B Safety example, the SRecord tool is used for the post-build CRC calculations in the MCUXpresso and uVision Keil IDEs.

In the IAR IDE, use the "ielftool" integrated feature.

The SRecord utility is used to calculate the post-build CRC without any changes. In the postbuild, an additional **.bat* file that uses the SRecord tool is called.

NOTE

The invariable memory test can be turned off/on in file *safety_config.h* file.

5.1.1 Postbuild in IEC60730B safety example

The approach with SRecord is used in the safety examples for the MCUXpresso and uVision Keil IDEs, when the post-build command calls the *crc-hex.bat* file, which supports the CRC16 and CRC32 calculations.

The *crc-hex.bat* file is in your SDK package, in the *<sdk_package>/middleware/safety_iec60730b/tools/crc* folder.

The complete post-build command, which is used in the safety example to calculate CRC32 in the uVision Keil IDE is as follows:

```
..\..\..\..\..\middleware\safety_iec60730b\tools\crc\crc_hex.bat
-..\..\..\..\boards\<YOUR_BOARD>\demo_apps\safety_iec60730b\mdk\debug\safety_iec60730b.hex
-..\..\..\..\boards\<YOUR_BOARD>\demo_apps\safety_iec60730b\mdk\debug\safety_iec60730b_crc.hex
-..\..\..\..\tools\srecord\srec_cat.exe -CRC32
```

"<YOUR_BOARD>" is the name of your SDK development board, e.g. "frdmk22f".

The first line is the path from the project root path (IDE project file) to the *crc_hex.bat* file. The other lines are the parameters for the *crc_hex.bat* file.

The *crc-hex.bat* file has three mandatory parameters and one optional parameter:

- The first paramater is the path from the *crc-hex.bat* file to your application's **.hex* file (*safety_iec60730b.hex*). It is the input for the calculation.
- The second parameter is the path for the generated output file. This file (with the specified name) is stored as a result of the script (*safety_iec60730b_crc.hex*) with the calculated CRC.
- The third parameter is the path from the *crc-hex.bat* file to the *srec_cat.exe* file.
- The fourth parameters is optional. When it is filled with"-CRC32", the result will be CRC32. Otherwise, the CRC16 calculation happens.

A dedicated structure in the input **.hex* file is used to define the area where the CRC will be calculated. All necessary information for the CRC will be read by the *crc-hex.bat* file from this structure.

Information table in the *.hex file

It is necessary to add a dedicated marker structure to the memory **.hex* file to use the presented *crc-hex.bat* file.

The presented *crc-hex.bat* file parses the last 16 bytes from the input **.hex* file to the found information table.

This information table must have a dedicated structure and it must be placed at the end of the input **.hex* file.

The structure of the information table is as follows:

```
/* The safety-related FLASH CRC value. */
fs_crc_t c_sfsCRC =
{
.ui16Start = 0xA55AU,
.ui32FlashStart = (uint32_t)__ROM_start__,
.ui32FlashEnd = (uint32_t)&Load$$ER_IROM3$$Limit,
.ui32CRC = (uint32_t)FS_CFG_FLASH_TST_CRC,
.ui16End = 0x5AA5U
};
```

- 0x5AA5 - the start/end marker for the information table
- ui32FlashStart - the start address for the CRC calculation
- ui32FlashEnd - the end address for the CRC calculation
- ui32CRC - the seed value

This table must be placed at the end of the **.hex* file. This can be assured by a linker script. The linker script depends on the IDE used. The exact description for the supported IDE is in the following chapter.

5.1.2 Arm uVison Keil IDE postbuild CRC

The safety example in the uVision Keil used Srecord to generate the postbuild for the invariable memory test.

To use the presented *crc-hex.bat* file, it is necessary to have correct settings in the IDE.

From the start, all necessary settings are added in the example project by default:

- The Flash test is turned on in the *safety_config.h* file.
- The output **.hex* file is turned on and the postbuild CRC is calculated by the *crc-hex.bat* file with the Srecord.
- The final post-processed image is downloaded to the ROM memory using the "Download" button.

5.1.2.1 Postbuild CRC settings

As mentioned in Postbuild in IEC60730B safety example , for the presented *crc-hex.bat* file, it is necessary to do some settings also in the IDE.

1. Set the IDE to generate the output **.hex* file. Go to "Options → Output" and check the "Create HEX File" box.

2. Enable the afterbuild options in "Options->User → After Build/Rebuild", check "Run #1", and fill it with the following command:

..\..\..\..\..\middleware\safety_iec60730b\tools\crc\crc_hex.bat
-..\..\..\..\boards\<YOUR_BOARD>\demo_apps\safety_iec60730b\mdk\debug\dev_safety_iec60730b.hex

-..\..\..\..\boards\<YOUR_BOARD>\demo_apps\safety_iec60730b\mdk\debug\dev_safety_iec60730b_crc.hex
-..\..\..\..\tools\srecord\srec_cat.exe

The meaning of this afterbuild command is described in Postbuild in IEC60730B safety example .

The product of the postbuild operation with the *crc-hex.bat* file is the *<your_project_name>_crc.hex* edited file, which must be loaded to the target. The best way to do this is to create a debug initialization file.

5.1.2.2 Debug initialization settings

By default, the uVision Keil IDE downloads the output file specified in "Options->output". Due to this, it is necessary to create an alternative debug initialization file. In our case, a **.hex* file with an added CRC is dedicated for the download to the target.

In the uVision Keil IDE, it is necessary to select the following options:

- "Options ->Debug->Initialization file" - fill it with the "safety_debug.ini" pattern.
- "Options->Utilities->Init File" - fill it with the "safety_debug.ini" pattern.

Use a text editor to create the *safety_debug.ini* file. Create an empty file, save it with the **.ini* extension, and copy the following command into the file: "LOAD .\debug\<YOUR_PROJECT>_crc.hex INCREMENTAL".

This command loads the *<YOUR_PROJECT>_crc.hex* file from the *.\debug* relative path and this address is relative to the project file (*<YOUR_PROJECT>.uvprojx* in the presented case). It means that the file is in the *debug* folder.

It is necessary to save this file to the project root path (to the folder with *<YOUR_PROJECT>.uvprojx* in the presented case).

After these IDE settings, the IDE calls the *crc-hex.bat* file after the build and it uses the alternative hex file *<YOUR_PROJECT>_crc.hex* as the source for programming during the download.

5.1.2.3 Linker settings for information table

The *crc-hex.bat* postbuild file expects the information table at the end of the **.hex* file. For this purpose, it is good to define your own section in the linker. In the uVision Keil IDE, it can be the following:

```
LR_IROM3 m_fs_flash_crc_start __size_flash_crc__{
; Safety-flash CRC region
ER_CRC (m_fs_flash_crc_start) FIXED (__size_flash_crc__)
{
*(.flshcrc)
}
}
```

Where "m_fs_flash_crc_start" and "__size_flash_crc__" are the user-defined address. This address must be at the end of the flash.

After defining this section in the ROM, a correct structure must be defined in the C language:

```
/* The safety-related FLASH CRC value. */
fs_crc_t c_sfsCRC __attribute__((used, section(".flshcrc"))) =
{
.ui16Start = 0xA55AU,
```

```
.ui32FlashStart = (uint32_t)__ROM_start__,

.ui32FlashEnd = (uint32_t)&Load$$ER_IROM3$$Limit,

.ui32CRC = (uint32_t)FS_CFG_FLASH_TST_CRC,

.ui16End = 0x5AA5U

};
```

5.1.3 MCUxpresso postbuild CRC

NOTE

The invariable memoty test example uses the *crc-hex.bat* file for the post-build calculation, so this example does not work on Unix/Mac operating systems.

To use the *crc-hec.bat* file in the MCUXpresso IDE, do some settings in the IDE.

1. Set the "Options → C/C++ Build → Settings → Build steps → Post-build steps" options correctly.
2. Set the debug sesion (or the GUI Flash tool) configuration correctly.
3. Put the "Information table" at the end of the invariable memory.

5.1.3.1 Post-build configuration

It is necessary to set the post-build string, so go to the "Options → C/C++ Build → Settings → Build steps → Post-build steps" menu.

Copy and paste the following post-build string into it:

```
arm-none-eabi-objcopy -v -O ihex "${BuildArtifactFileName}" "${BuildArtifactFileBaseName}.hex"

${ProjDirPath}/crc_hex.bat -${ConfigName}/${BuildArtifactFileBaseName}.hex -${ConfigName}/$
{BuildArtifactFileBaseName}_crc.hex -tools\\srecord\\srec_cat.exe
```

This string ensures that the MCUxpresso IDE generates a **.hex* file with the same name as your project. After this, call the *crc_hex.bat* file with the correct parameters as follows:

- *-${ConfigName}/${BuildArtifactFileBaseName}.hex* - the path to your application **.hex* file.
- *-${ConfigName}/${BuildArtifactFileBaseName}_crc.hex* - the path to the generated **.hex* file with the CRC added.
- *-tools\\srecord\\srec_cat.exe* - the path to the *screcat.exe* utility.

Because the name of your poject is set as the "${BuildArtifactFileBaseName}" variable, this postbuild is independent on your project name.

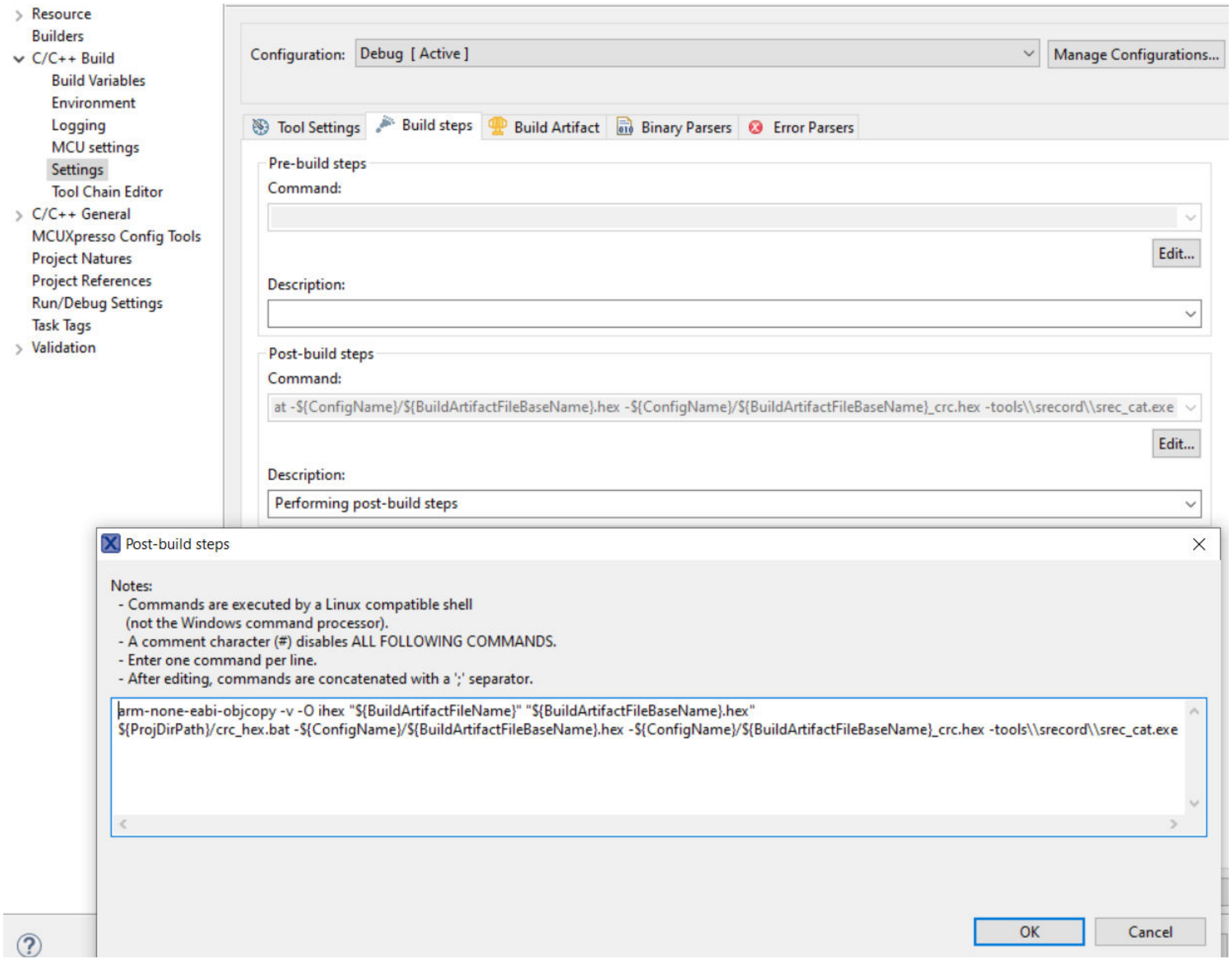

Figure 6. Configuration of post-build steps

5.1.3.2 Place information table

The *crc-hex.bat* file expects the information table in the last 16 bytes of the input **.hex* file. This table can be defined as the following structure:

```
/* The safety-related FLASH CRC value. */
fs_crc_t c_sfsCRC __attribute__((used, section(".flshcrc"))) =
{
.ui16Start = 0xA55AU,
.ui32FlashStart = (uint32_t)&__ROM_start__,
.ui32FlashEnd = (uint32_t)&m_safety_flash_end,
.ui32CRC = (uint32_t)FS_CFG_FLASH_TST_CRC,
.ui16End = 0x5AA5U
};
```

Where "__attribute__((used, section(".flshcrc")))" is a directive for the linker script to place this strucuture to memory section "flshcrc".

MCUXpresso Linker settings

The structure definition in the above example expects memory section "flscrc" to be defined in the linker. This can be set as follows:

```
/* The safety FLASH CRC. */
.SEC_CRC m_fs_flash_crc_start : ALIGN(4)
{
FILL(0xff)
KEEP(*(.flshcrc*))
} >MEM_FLASH
```

Where "m_fs_flash_crc_start" is the user-defined address, but this section must be placed at the end of the output **.hex* file.

5.1.3.3 Flash loader configuration

It is necessary to set a correct output file for the download to the target. There are the following two ways to do this in the MCUXpresso IDE:

1. Using the "Debug configuration".
2. Using the "GUI Flash Tool".

Debug configuration

- Create the debug configuration for your debugger.
- Open the "Debug Configurations" menu ("Run → Debug configuration") and select the "Startup" tab. In this tab, select "Load Image -> Use File -> *<YOUR_PROJECT_NAME_crc.hex*".
- This edited **.hex* file is in the *<workspace>/<your_project>/Debug/<your_project>_crc.hex* folder.

This can be set in the OpenSDA, CMSIS-DAP, or J-Link debuggers.

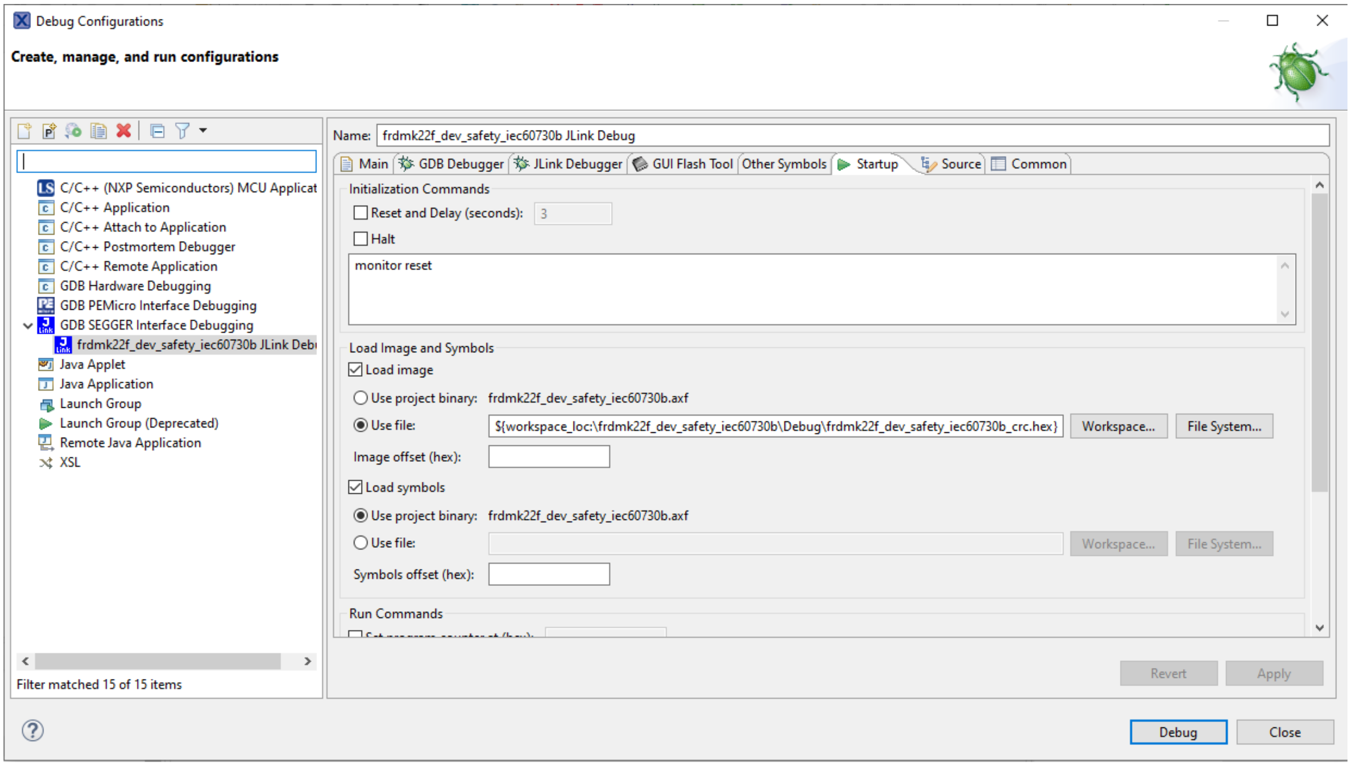

Figure 6. Using output *.hex file with calculated CRC in MCUXpresso IDE

Using GUI Flash Tool

Only the SEGGER J-Link probes in the GUI Flash Tool support **.hex* files.

In the GUI Flash Tool settings, select "Workspace → <Configuration> → *<PROJECT_NAME>_crc.hex*" file for download.

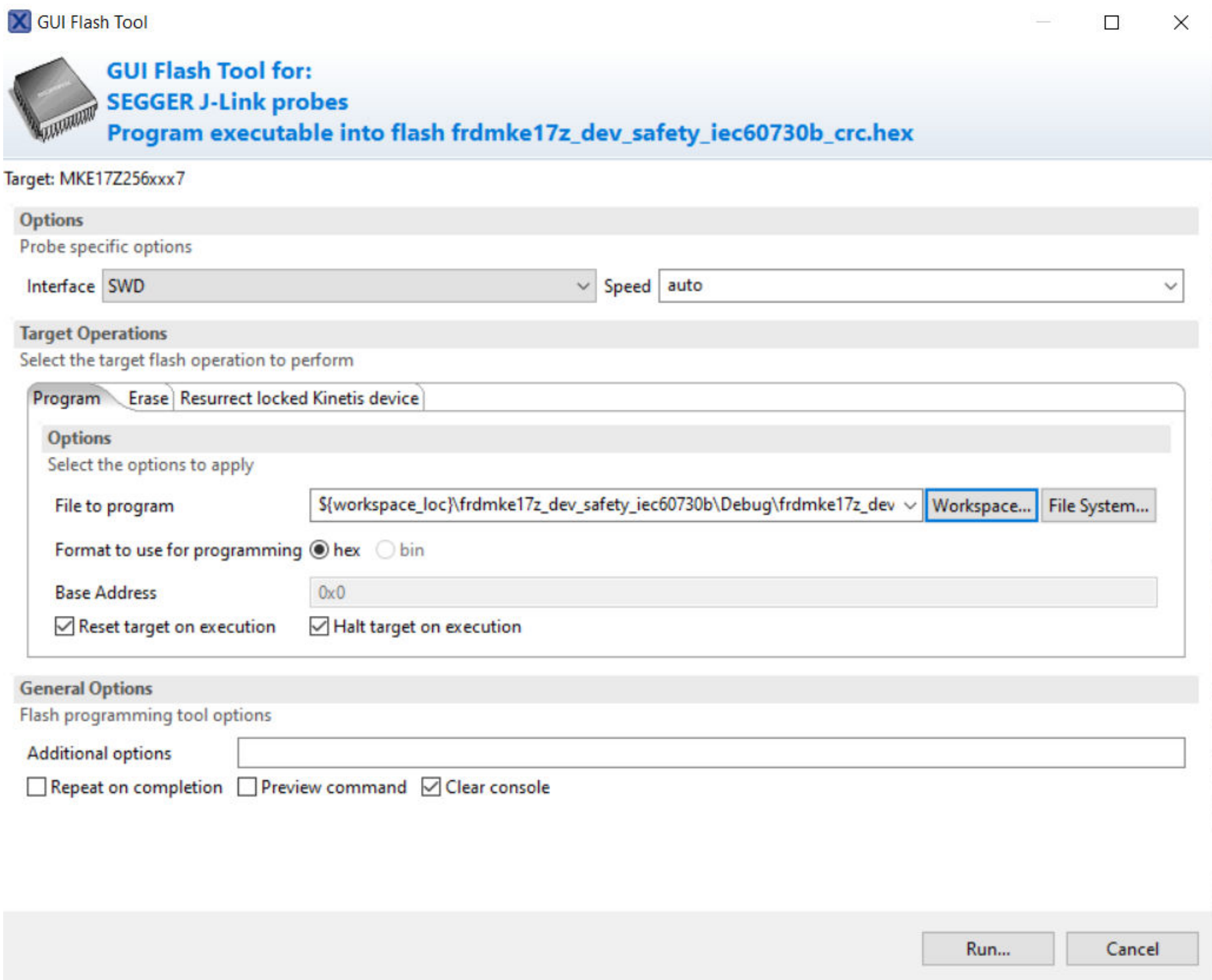

Figure 6. GUI Flash Tool - SEGGER J-Link

Chapter 6
IEC60730B tests

The library contains the following tests:

- Analog I/O test
- Clock test
- CPU register test
- Digital I/O test
- Invariable memory (flash) test
- Variable memory (RAM) test
- Program counter test
- Stack test
- Watchdog test
- Touch-sensing peripheral TSIv5 test

The following chapters describe each test with focus on the example application (debugging).

6.1 Clock test

The clock test procedure tests the oscilator frequency for the CPU core in the wrong frequency condition.

NOTE

The default clock setting from the SDK library is used in the example. For a real application, ensure that the reference clock source is not dependent on the primary (tested) clock.

6.2 CPU register

The CPU register test procedure tests all CPU registers for the stuck-at condition (except for the program counter register). The program counter test is implemented as a stand-alone safety routine.

Some tests stay in an endless loop in case of an error, others return a corresponding error message.

6.3 DIO test

The Digital Input/Output (DIO) test procedure performs the plausibility check of the processor's digital IO interface.

NOTE

Make sure that the time between the "set" and "get" functions is sufficient for the GPIO peripheral speed.

6.4 Invariable memory test

The invariable (Flash) memory test provides a CRC check of a dedicated part of memory. This test can be turned off in the *safety_config.h* file.

The test consists of the following two parts:

- Post-build CRC calculation of the dedicated memory.
- Runtime CRC calculation and comparison with the post-build result.

The post-build calculation is different for each IDE:

In the IAR IDE, the CRC is calculated by the IDE directly using the linker (see Options->Build Action). The Flash test is fully integrated to the example project in the IAR IDE. It is necessary only to turn this test on in the *safety_config.h* file.

In the uVision Keil IDE, the CRC is calculated by the Srecord third-party tool, which is called from the IDE (see Options → User → After Build) The Flash test is fully integrated to the example project in the uVison Keil IDE. It is only necessary to turn this test on in the *safety_config.h* file. In case of any issues, see Arm uVison Keil IDE postbuild CRC

In the MCUXpresso IDE, the CRC is calculated by the Srecord third-party tool. The user must do some additional steps. For more information, see MCUxpresso postbuild CRC.

NOTE

The invariable memory test example uses the *crc.bat* file for post-build calculation, so this example does not work on a Unix/Mac operating system.

NOTE

When you debug your application with the Flash test turned on, be careful when using the breakpoint. The software breakpoint usually changes the CRC result and causes a safety error.

6.5 Variable memory test

The variable memory on the supported MCU is an on-chip RAM.

The RAM memory test is provided by the MarchC or MarchX tests.

The test copies a block of memory to the backup area defined by the linker. Be sure that the BLOCK_SIZE parameter is smaller than the backup area defined by the linker.

NOTE

This test cannot be interupted.

6.6 Program counter test

The CPU program counter register test procedure tests the CPU program counter register for the stuck-at condition. The program counter register test can be performed once after the MCU reset and also during runtime.

NOTE

The program counter test cannot be interrupted.

6.7 Stack test

This test routine is used to test the overflow and underflow conditions of the application stack. The testing of the stuck-at faults in the memory area occupied by the stack is covered by the variable memory test. The overflow or underflow of the stack can occur if the stack is incorrectly controlled or by defining the "too-low" stack area for the given application.

NOTE

Choose a correct pattern to fill the tested area. This pattern must be unique to the application.

6.8 Watchdog test

The watchdog test provides the testing of the watchdog timer functionality. The test is run only once after the reset. The test causes the WDOG reset and compares the preset time for the WDOG reset to the real time.

For this test to run correctly, it is necessary to keep the WDOG_backup variable in a part of memory which is not corrupeted by the WDOG reset.

NOTE

Some debuggers do not allow the WDOG reset. Due to this, it is necessary to turn off the WDOG when debugging the application.

Chapter 7
Revision history

Table 1. Revision history

Revision number	SDK	Description
0	2.9.0	Intial release.
1	2.10.0	Change devices supported in SDK rel. 2.10.
2	2.10.0	Post-build description added.
3	-	Version cover SDK 2.9 and SDK 2.10 release - document for web

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 07/2021

Document identifier: IEC60730BIMX8MXEUG