

# i.MX8M Safety Example



# Contents

Chapter 1 IEC60730B Safety library example user's guide..... 3

Chapter 2 Hardware settings..... 4

Chapter 3 File structure..... 7

Chapter 4 Example application..... 9

Chapter 5 Running example..... 14

Chapter 6 IEC60730B tests..... 15

# 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 this package and accessible at [www.nxp.com/IEC60730](http://www.nxp.com/IEC60730).

# Chapter 2

## Hardware settings

This chapter describes how to set up the hardware of the evaluation 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 necessary to make some hardware settings. For the default configuration of your development board, see the corresponding board's user manual at [www.nxp.com](http://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)

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 SW101 to power on the board.
2. Connect a USB-C cable between the host PC and the J301 USB port (this port allows to download 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](http://www.nxp.com/imx8mminievk) for more information.

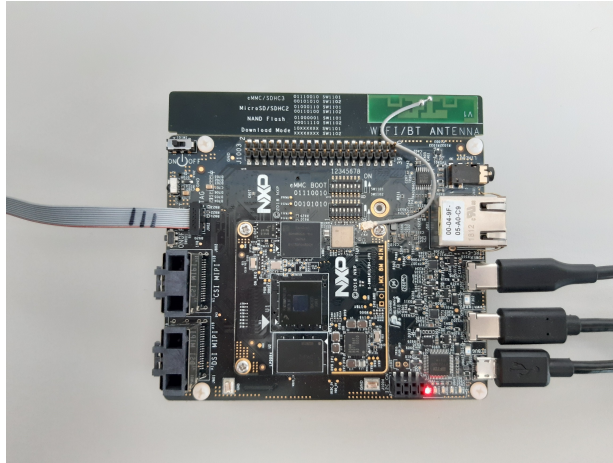


Figure 1. Hardware connection of EVK-MIMX8MM

## 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)

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 SW101 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](http://www.nxp.com/imx8mnanoevk) for more information.

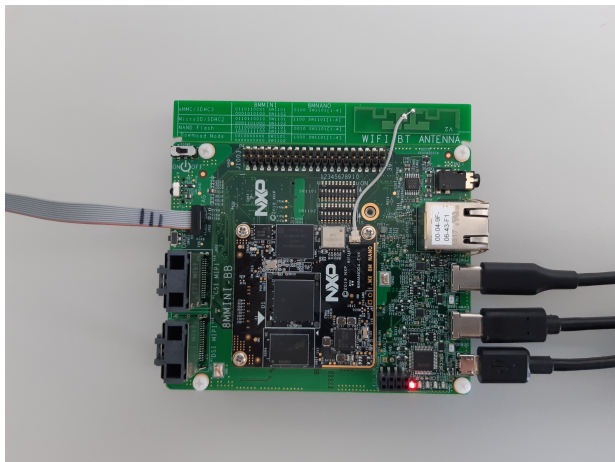


Figure 2. Hardware connection of EVK-MIMX8MN

# 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\_1* folder in the SDK package.

The folder has the following structure:

Name	Ext	S
[..]		<
[common_test]		<
[compiler]		<
[core_test]		<
IEC60730B_M4_M7_KEIL_v4_1	lib	
IEC60730B_M4_M7_COM_KEIL_v4_1	lib	
IEC60730B_M33_KEIL_v4_1	lib	
IEC60730B_M33_COM_KEIL_v4_1	lib	
IEC60730B_M0_KEIL_v4_1	lib	
IEC60730B_M0_COM_KEIL_v4_1	lib	
iec60730b_types	h	
iec60730b	h	
libIEC60730B_M4_M7_MCUX_v4_1	a	
libIEC60730B_M4_M7_COM_MCUX_v4_1	a	
libIEC60730B_M33_MCUX_v4_1	a	
libIEC60730B_M33_COM_MCUX_v4_1	a	
libIEC60730B_M0_MCUX_v4_1	a	
libIEC60730B_M0_COM_MCUX_v4_1	a	
IEC60730B_M4_M7_IAR_v4_1	a	
IEC60730B_M4_M7_COM_IAR_v4_1	a	
IEC60730B_M33_IAR_v4_1	a	
IEC60730B_M33_COM_IAR_v4_1	a	
IEC60730B_M0_IAR_v4_1	a	
IEC60730B_M0_COM_IAR_v4_1	a	

Where:

- common\_test 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*.
- compiler - compiler support files.
- core\_test 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 application 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 3.

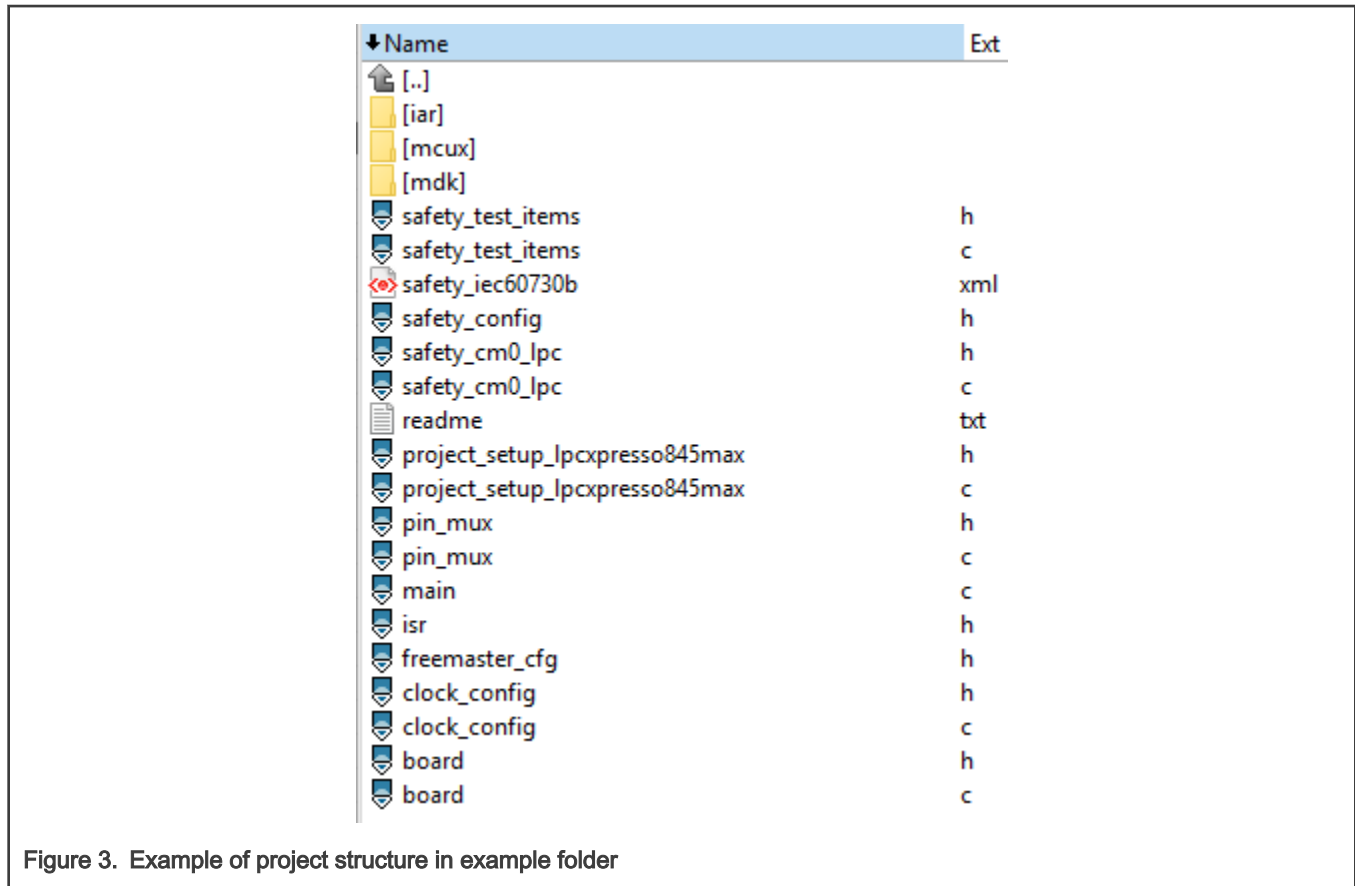


Figure 3. Example of project structure in example folder

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 4. 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 hierarchy is shown in [Figure 5](#).

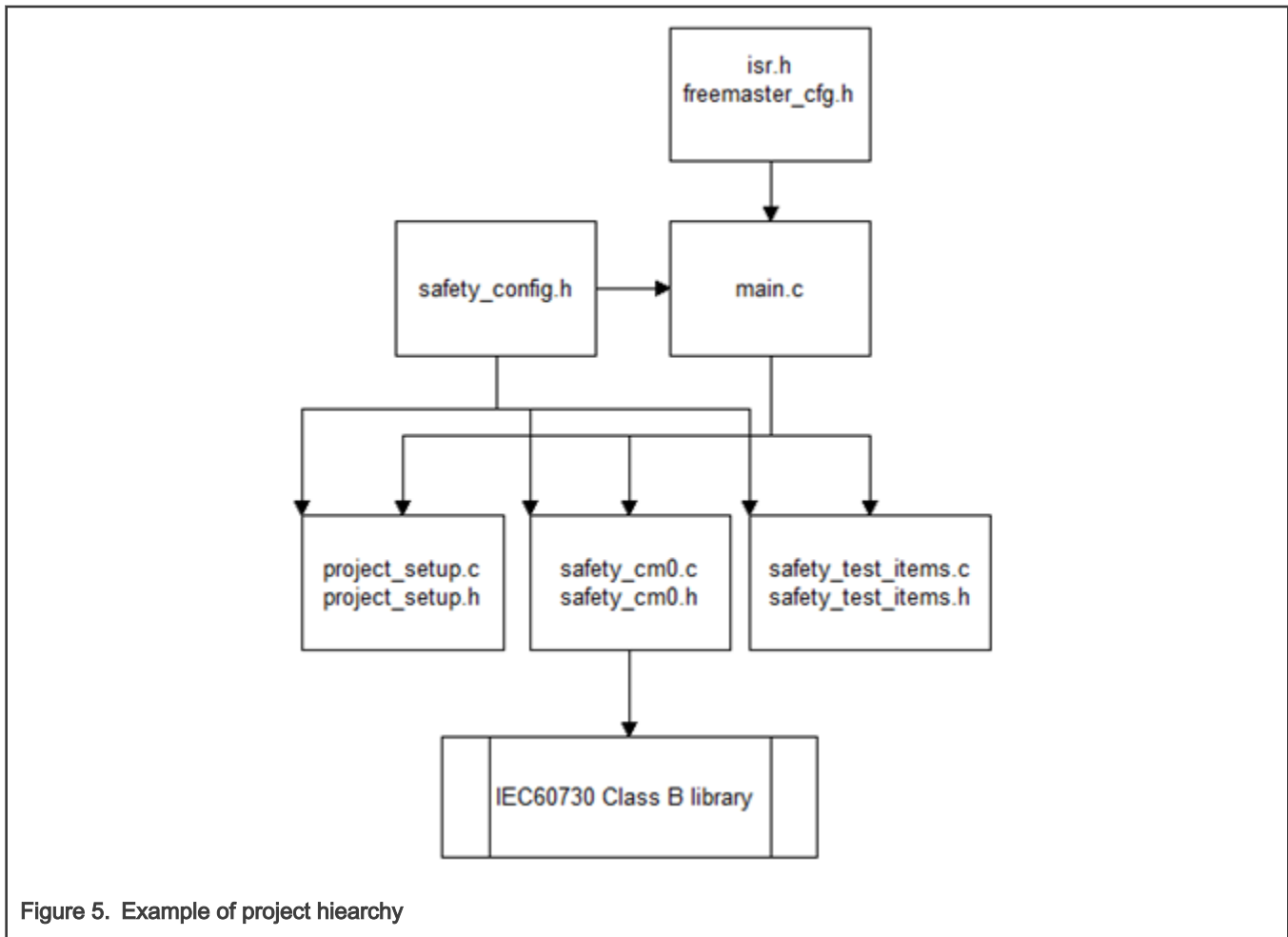


Figure 5. Example of project hierarchy

[Figure 5](#) 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
0x80000000;; 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 necessary macros for the safety example are defined in this file.

The "switch macros", which enable the user to turn off 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 SafetyErrorHandling() function */
#define SAFETY_ERROR_ACTION 1

/* TEST SWITCHES - for debugging it is better to turn the flash test and watchdog OFF */
#define ADC_TEST_ENABLED 1
#define CLOCK_TEST_ENABLED 1
#define DIO_TEST_ENABLED 1
#define FLASH_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 occurs, the *SafetyErrorHandling()* function is called.

# Chapter 5

## Running example

For the first run of the example on your hardware, it is recommended 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.

# 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 TSLv5 test

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

### 6.1 Clock test

The clock test procedure tests the oscillator 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 is turned off by default 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 Arm Keil IDE, it is necessary to use a third-party tool (Srecord):

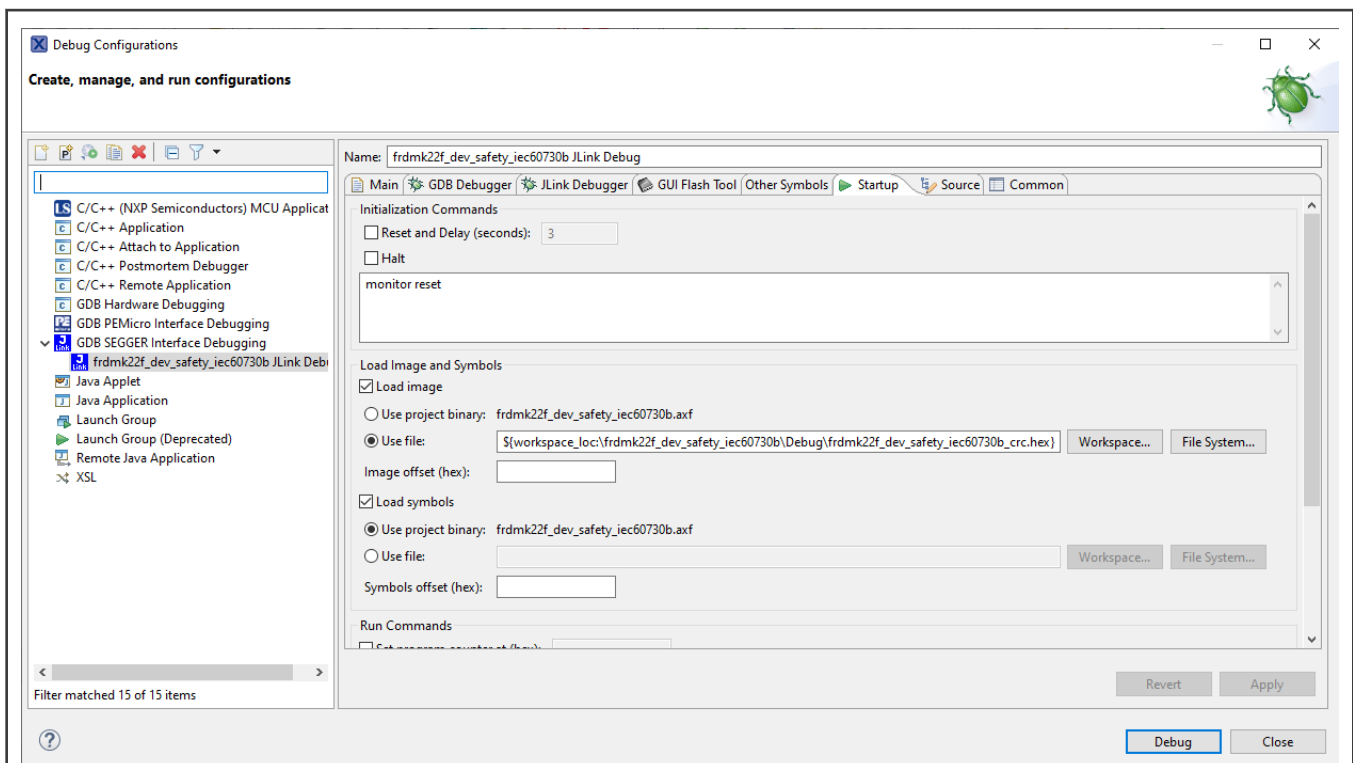
- Srecord and all necessary settings are added to the presented example by default.
- Turn on the Flash test in the *safety\_config.h* file.
- The final post-processed image is downloaded to the ROM memory using the "Download" button.
- For more information on using Srecord in Arm Keil IDE, see *Calculating Post-Build CRC in Arm® Keil®* (document [AN12520](#)).

In case of an issue, check the following settings:

- Check if the "Option -> User -> AfterBuild" tab contains the `"../..../..../middleware/safety_iec60730b/tools/crc/crc_hex.bat  
../..../boards/<YOUR_BOARD>demo_app/safety_iec60730b/mdk/debug/safety_iec60730b.hex  
../..../boards/<YOUR_BOARD>demo_app/safety_iec60730b/mdk/debug/safety_iec60730b_crc.hex  
../..../tools/records/rec_cat"` path and if the checkbox is selected (it should be filled and checked by default).
- Check if the "Option->Debug ->Initialization File" tab contains the `"../..../..../middleware/safety_iec60730b/boards/  
common/debug.in"` path.

In the MCUXpresso IDE:

- Srecord and all necessary settings are added to the presented example by default.
- Turn on the Flash test in the *safety\_config.h* file.
- Create the "debug configuration" for your debbuger.
- Open the "Debug Configurations" menu ("Run->Debug configuration") and select the "Startup" tab. In this tab, select "Load Image -> Use File -> <YOUR\_BOARD>\_safety\_iec60730b\_crc.hex".
- This edited HEX file is located in the <workspace>/<your\_project>/Debug/<your\_project>\_crc.hex folder.



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



In case of an issue, check also the following settings:

- Check if the post-build actions are set: `"arm-none-eabi-objcopy -v -O ihex "${BuildArtifactFileName}" "${BuildArtifactFileName}.hex" ${ProjDirPath}/linker/crc_hex.bat -..||||${ConfigName}|\frdmk22f_dev_safety_iec60730b.hex -..||||${ConfigName}|\frdmk22f_dev_safety_iec60730b_crc.hex -..||safety_iec60730b|ltools|lsrec|srec_cat"`. It should be filled by default.

#### 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 interrupted.

## 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 corrupted 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.

### ***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).

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, Altivec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© NXP B.V. 2020.

All rights reserved.

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

For sales office addresses, please send an email to: [salesaddresses@nxp.com](mailto:salesaddresses@nxp.com)

Date of release: 12/2020

Document identifier: IEC60730BIMX8MXEUG