

# Kinetis CM4/CM7 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..... 13

Chapter 6 IEC60730B tests..... 23

Chapter 7 Revision history..... 26

# 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](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 Kinetis CM4 and CM7 family supports the following development boards:

- FRDM-KV31f
- FRDM-K22f

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 FRDM-K22F

### Debugger:

The default debugger in the example project is set to CMSIS-DAP.

### FreeMASTER

FreeMASTER communication is used via an onboard debugger with a speed of 9600 bd.

There are no hardware settings necessary to run the IEC60730B example code.

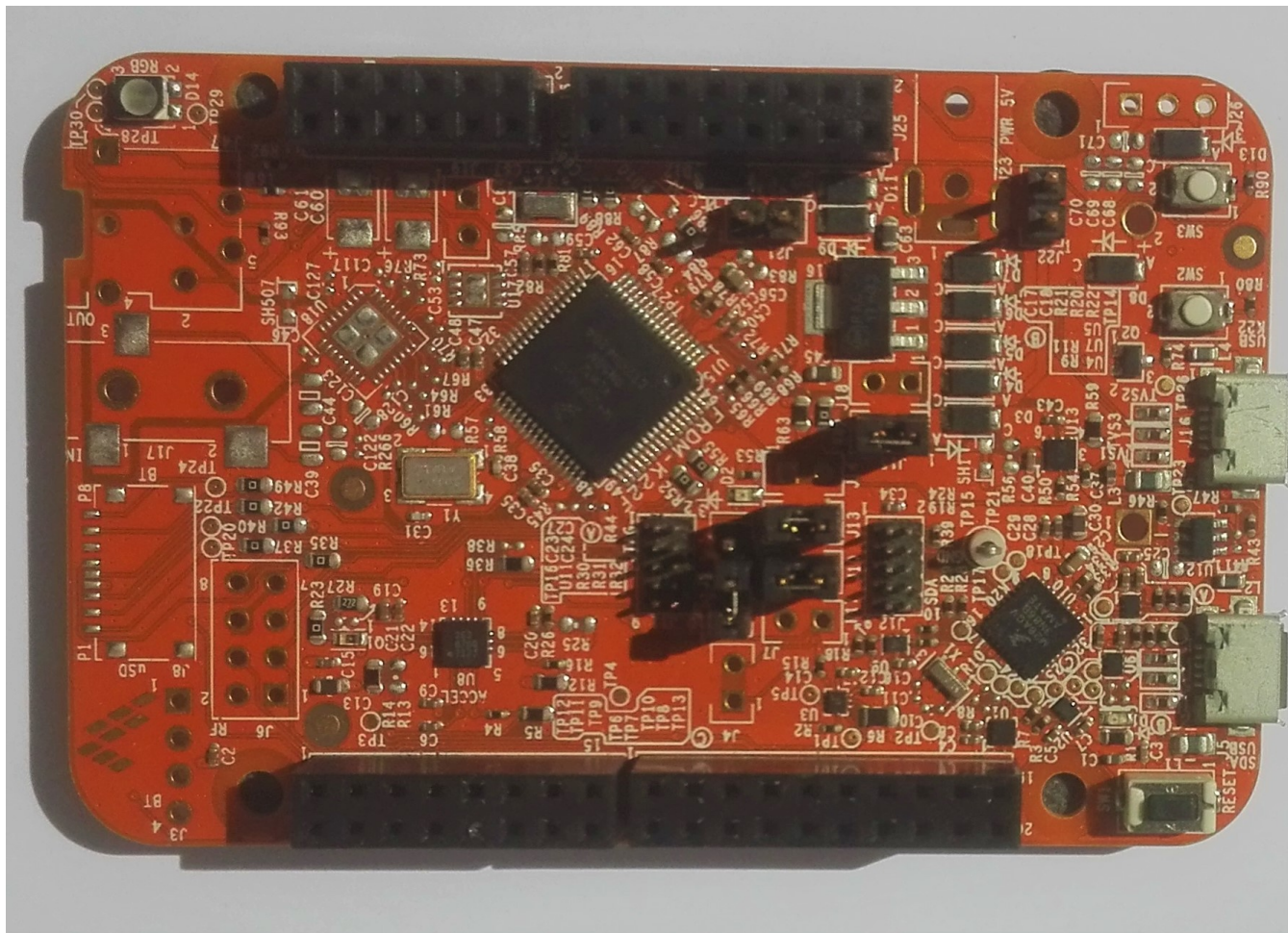


Figure 1. FRDM-K22F

## 2.2 FRDM-KV31f

### **Debugger:**

The default debugger in the example project is set to CMSIS-DAP.

### **FreeMASTER**

FreeMASTER communication is used via an onboard debugger with a speed of 9600 bd.

The IEC60730B example application was tested with the jumper settings shown in [Figure 2](#).

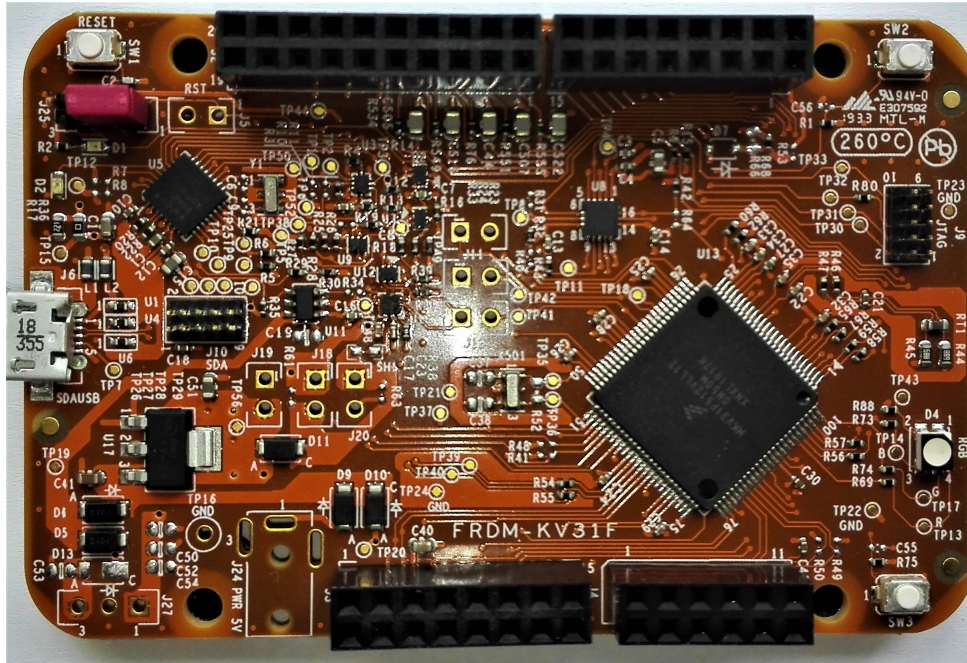


Figure 2. FRDM-KV31F

# 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).

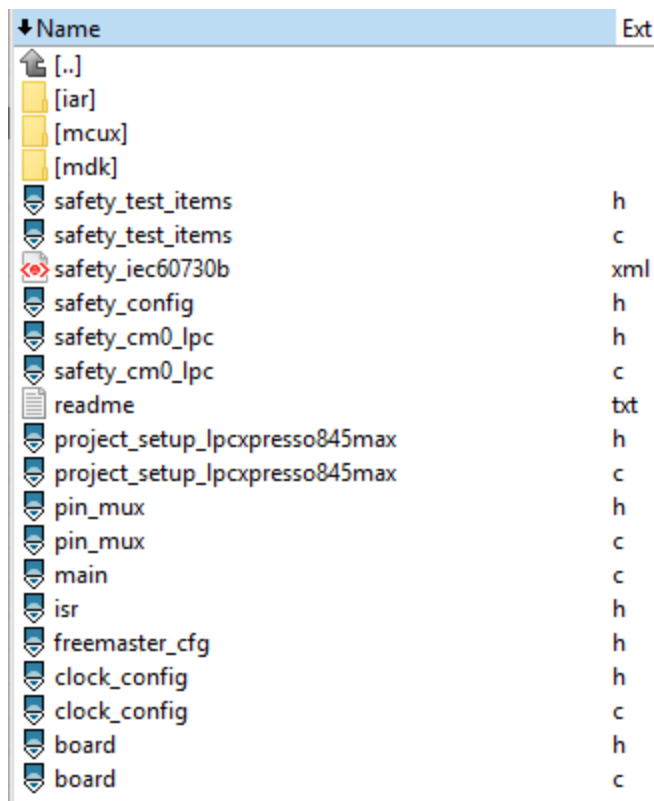


## 3.2 Example of library handling code

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](#).



Name	Ext
[..]	
[iar]	
[mcux]	
[mdk]	
safety_test_items	h
safety_test_items	c
safety_iec60730b	xml
safety_config	h
safety_cm0_lpc	h
safety_cm0_lpc	c
readme	txt
project_setup_lpcxpresso845max	h
project_setup_lpcxpresso845max	c
pin_mux	h
pin_mux	c
main	c
isr	h
freemaster_cfg	h
clock_config	h
clock_config	c
board	h
board	c

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).

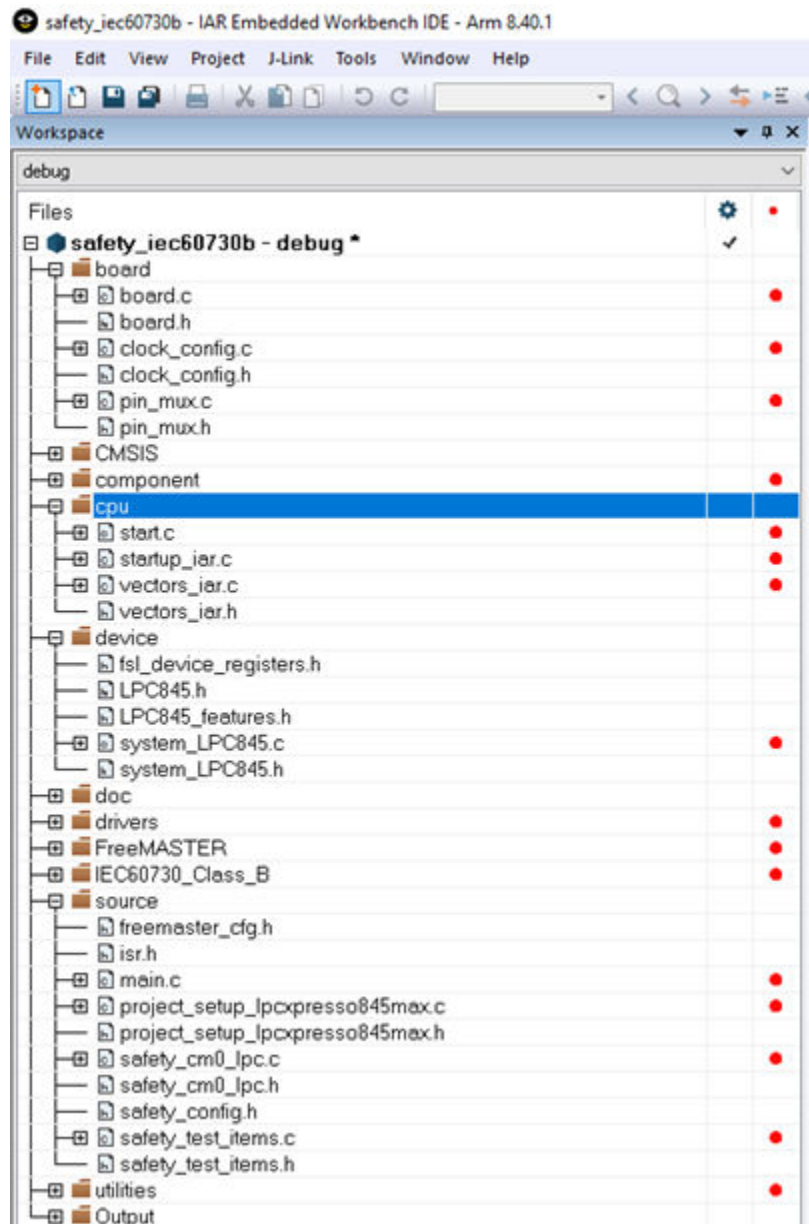


Figure 4. IAR example application structure

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.

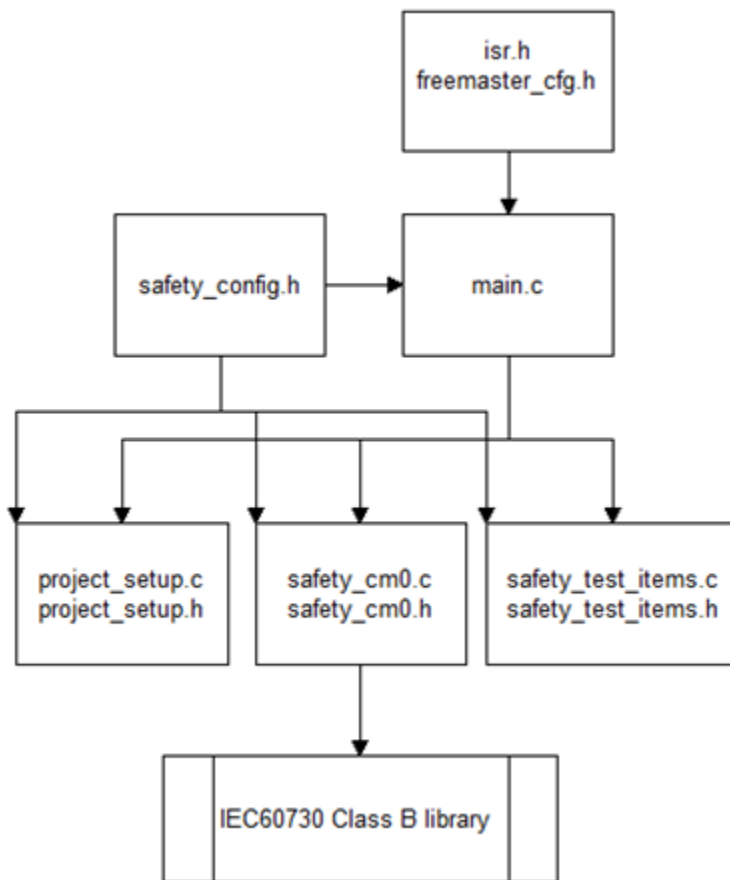


Figure 5. Example of project hierarchy

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 *safety\_cm4\_cm7\_kinetis.c* 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 *safety\_cm4cm7\_kinetis.c* 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 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 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.

## 4.3 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.

```
fs_dio_test_t dio_safety_test_item_0 =
{
    .gpio = GPIOA_BASE,
    .pcr = PORTA_BASE, /* Base address of PCR register */
    .pinNum = 5, .pinDir = PIN_DIRECTION_IN,
    .pinDir = PIN_DIRECTION_IN,
    .pinMux = PIN_MUX_GPIO,
};
/* NULL terminated array of pointers to dio_test_t items for safety DIO test */
fs_dio_test_t *dio_safety_test_items[] = { &dio_safety_test_item_0, &dio_safety_test_item_1, NULL };
```

## 4.4 Source file - safety\_safety\_cm4\_cm7\_<platform>.c/.h

The *safety\_cm4\_cm7\_kinetis.c* 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.

## 5.1 FreeMASTER monitoring

FreeMASTER is used as the external PC tool for real-time monitoring. FreeMASTER is also implemented in the IEC60730B safety examples. For simplicity reasons, the MAP file is the source of the variable address. Before connecting FreeMASTER to your application, make sure that the application is running.

### Running FreeMASTER:

Download and install FreeMASTER from [www.nxp.com/freemaster](http://www.nxp.com/freemaster).

The example project is in the *safety.pmp* file. Open it.

Check the project settings for your application:

- Open "Project->Options ->MAP Files". It must point to your output files.

### IAR IDE and ARM Keil IDE

Navigate to the *boards/<your\_board>/demo\_apps/safety\_iec60730b/<compiler>/<debug or release>/\*.out* file.

### MCUXpresso IDE

Navigate to the *<workspace>/<project\_name>/<Debug or Release>/<project\_name>.axf* file.

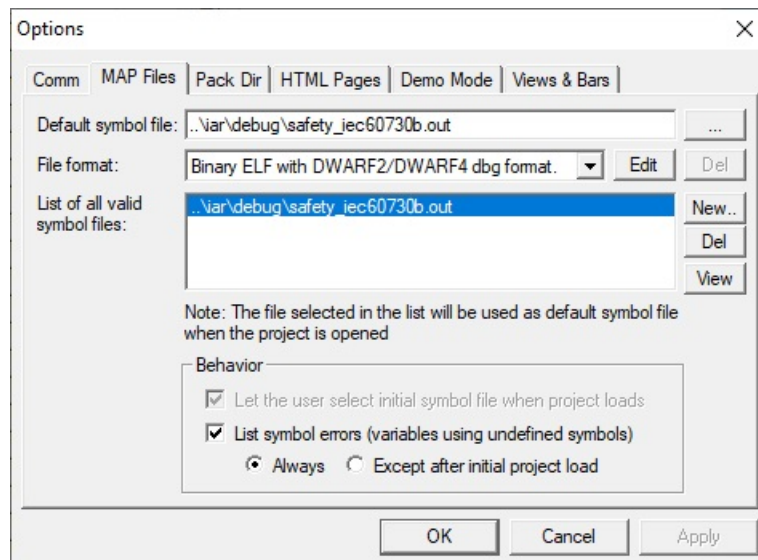


Figure 6. Example of setting MAP files for FRDM-KV11 board

- Open "Project ->Options ->Comm" and select a correct RS-232 connection and speed. The connection speed is in the *safety\_config.h* file's "SERIAL\_BAUD\_RATE" macros. By default, this speed is set to 9600 bd.

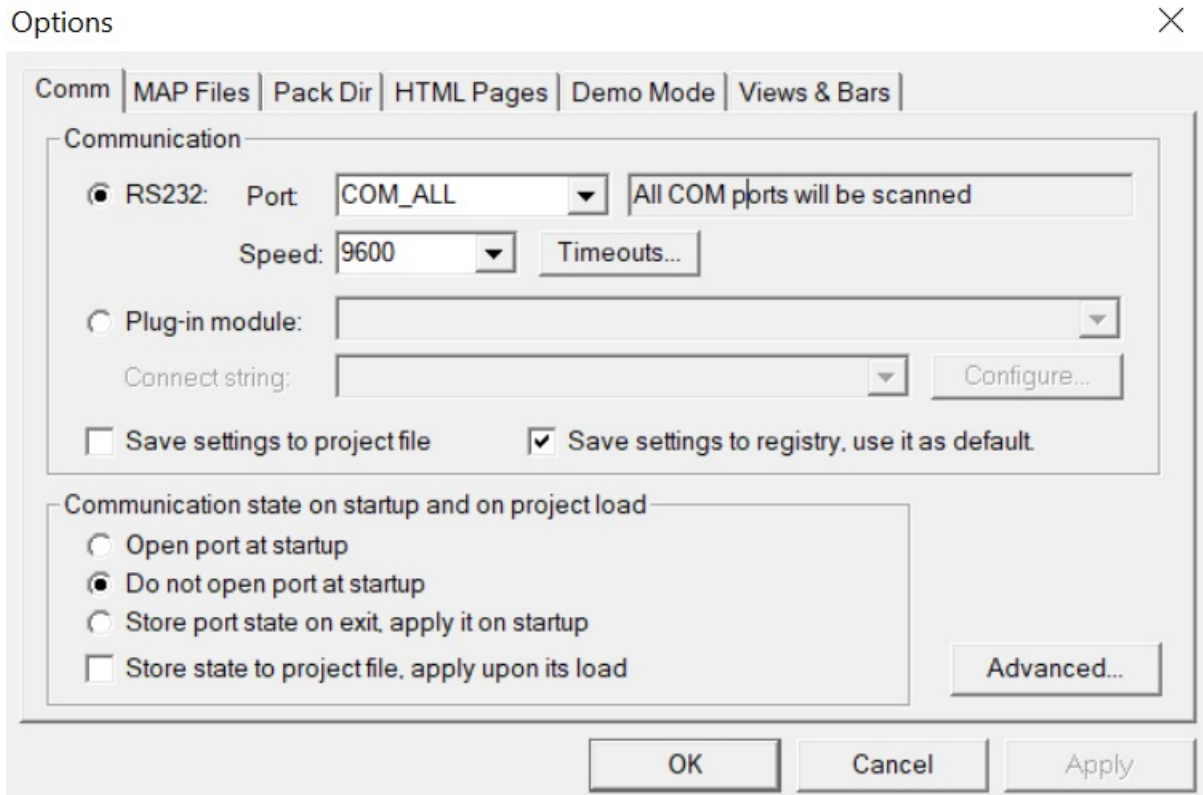


Figure 7. Setting UART speed

Now you can connect to the development board by pressing "CTRL+G" or clicking the "GO" button:

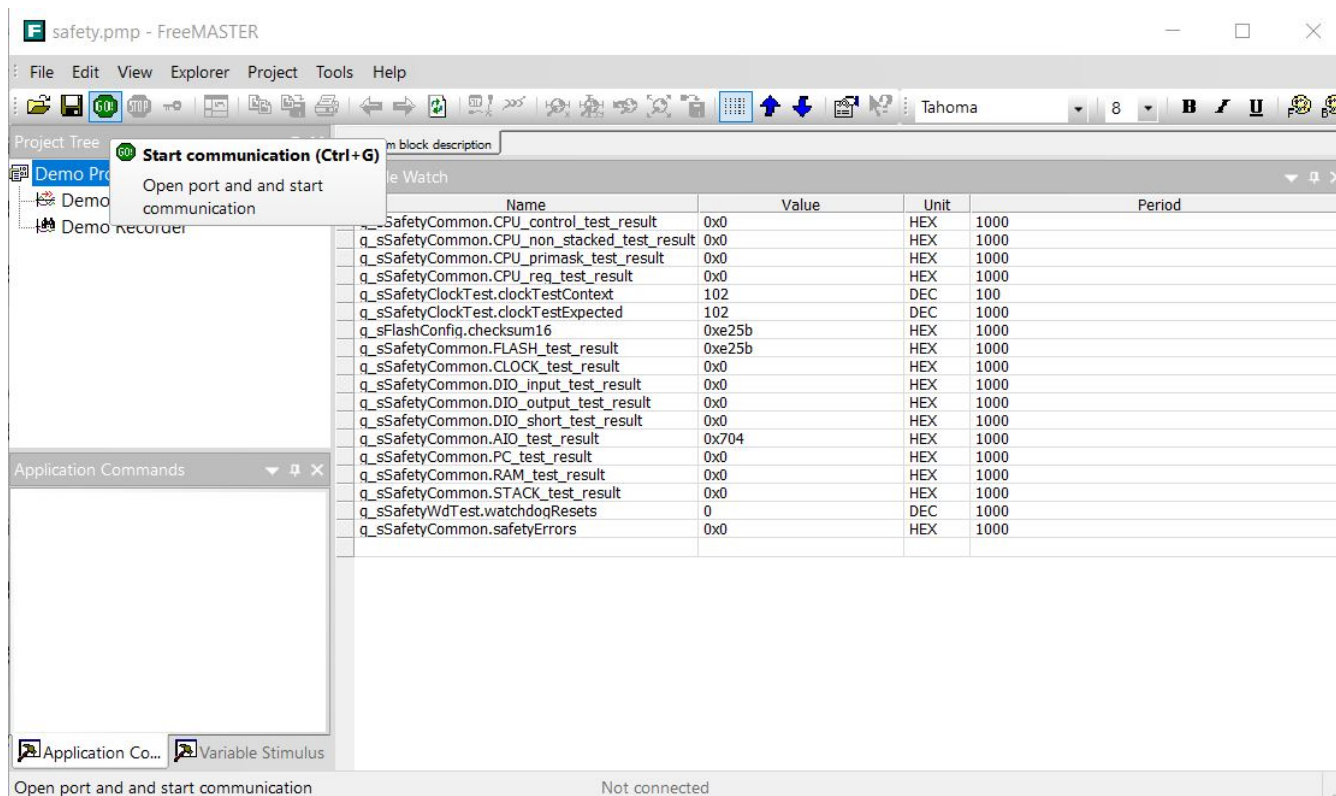


Figure 8. Safety example FMSTR application

Usually, the AIO test is a number of results oscillating between 0x0 and 0x704 (test passed and test in progress).

## 5.2 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.

### 5.2.1 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 parameter 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 parameter 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.

### 5.2.2 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.

### 5.2.2.1 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:

```
..l...l...l...middleware\safety_iec60730b\tools\crlcrc_hex.bat  
..l...l...boards<YOUR_BOARD>\demo_apps\safety_iec60730b\mdk\debug\dev_safety_iec60730b.hex  
..l...l...boards<YOUR_BOARD>\demo_apps\safety_iec60730b\mdk\debug\dev_safety_iec60730b_crc.hex  
..l...l...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.

### 5.2.2.2 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.

### 5.2.2.3 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__)
{
  *(.flashcrc)
}
}

```

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(".flashcrc"))) =
{
  .ui16Start = 0xA55AU,
  .ui32FlashStart = (uint32_t)__ROM_start__,
  .ui32FlashEnd = (uint32_t)&Load$$ER_IROM3$$Limit,
  .ui32CRC = (uint32_t)FS_CFG_FLASH_TST_CRC,
  .ui16End = 0x5AA5U
};

```

## 5.2.3 MCUXpresso postbuild CRC

### NOTE

The invariable memory 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 session (or the GUI Flash tool) configuration correctly.
3. Put the "Information table" at the end of the invariable memory.

### 5.2.3.1 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 "${BuildArtifactFileName}" "${BuildArtifactFileName}.hex"
${ProjDirPath}/crc_hex.bat -${ConfigName}/${BuildArtifactFileName}.hex -${ConfigName}/${
BuildArtifactFileName}_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:

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

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

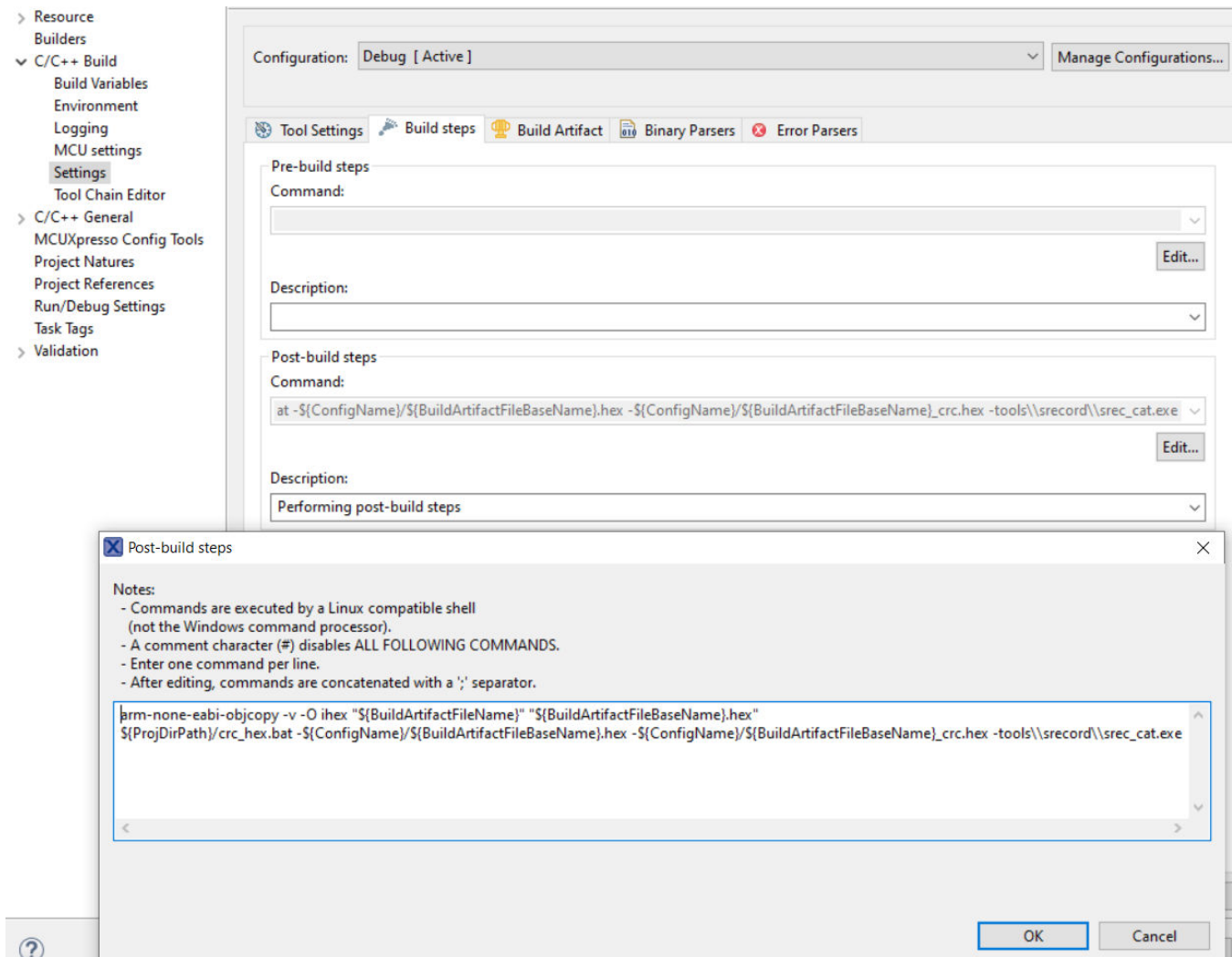


Figure 8. Configuration of post-build steps

### 5.2.3.2 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(".flashcrc"))) =
{
```

```
.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(\".flashcrc\")))`" is a directive for the linker script to place this structure to memory section `flashcrc`.

### MCUXpresso Linker settings

The structure definition in the above example expects memory section `flashcrc` 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(*(.flashcrc*))
} >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.

## 5.2.3.3 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.

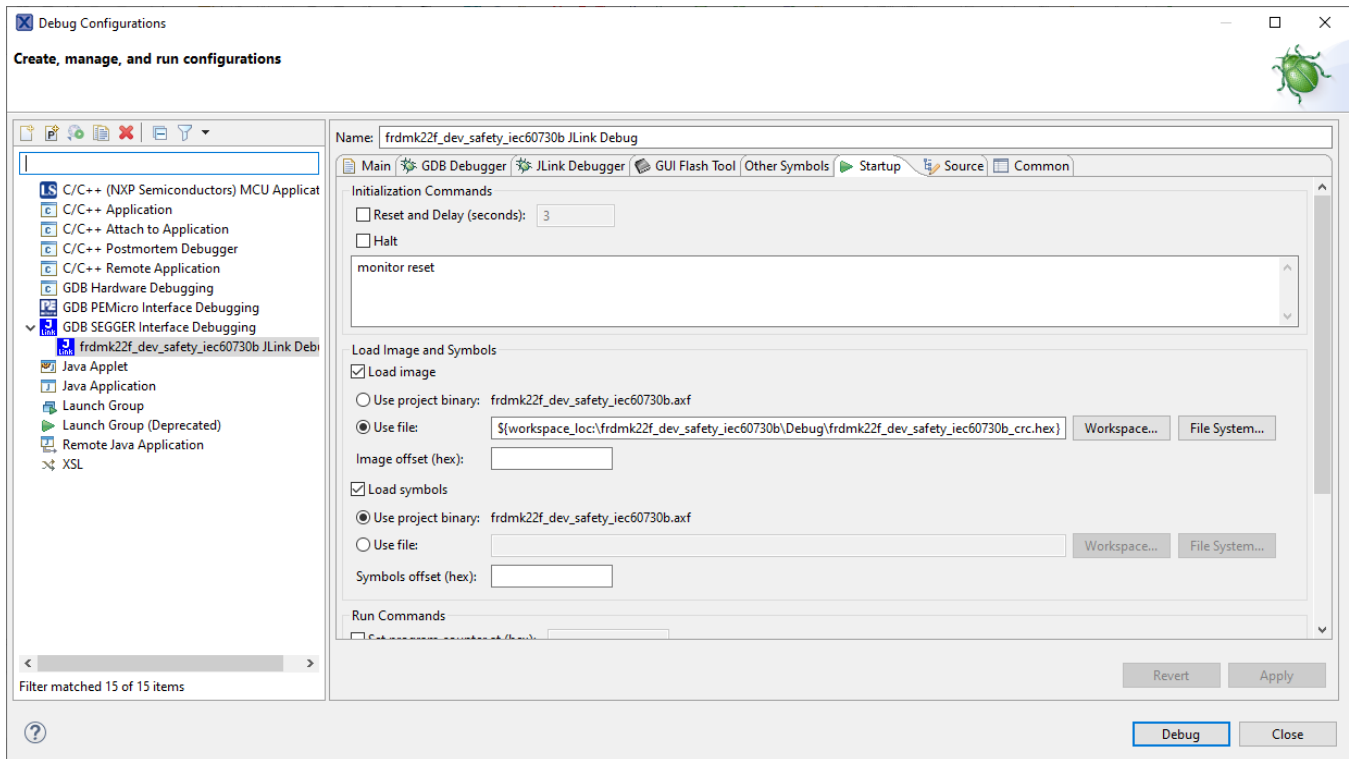


Figure 8. Using output \*.hex file with calculated CRC in MCUXpresso IDE

### 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.

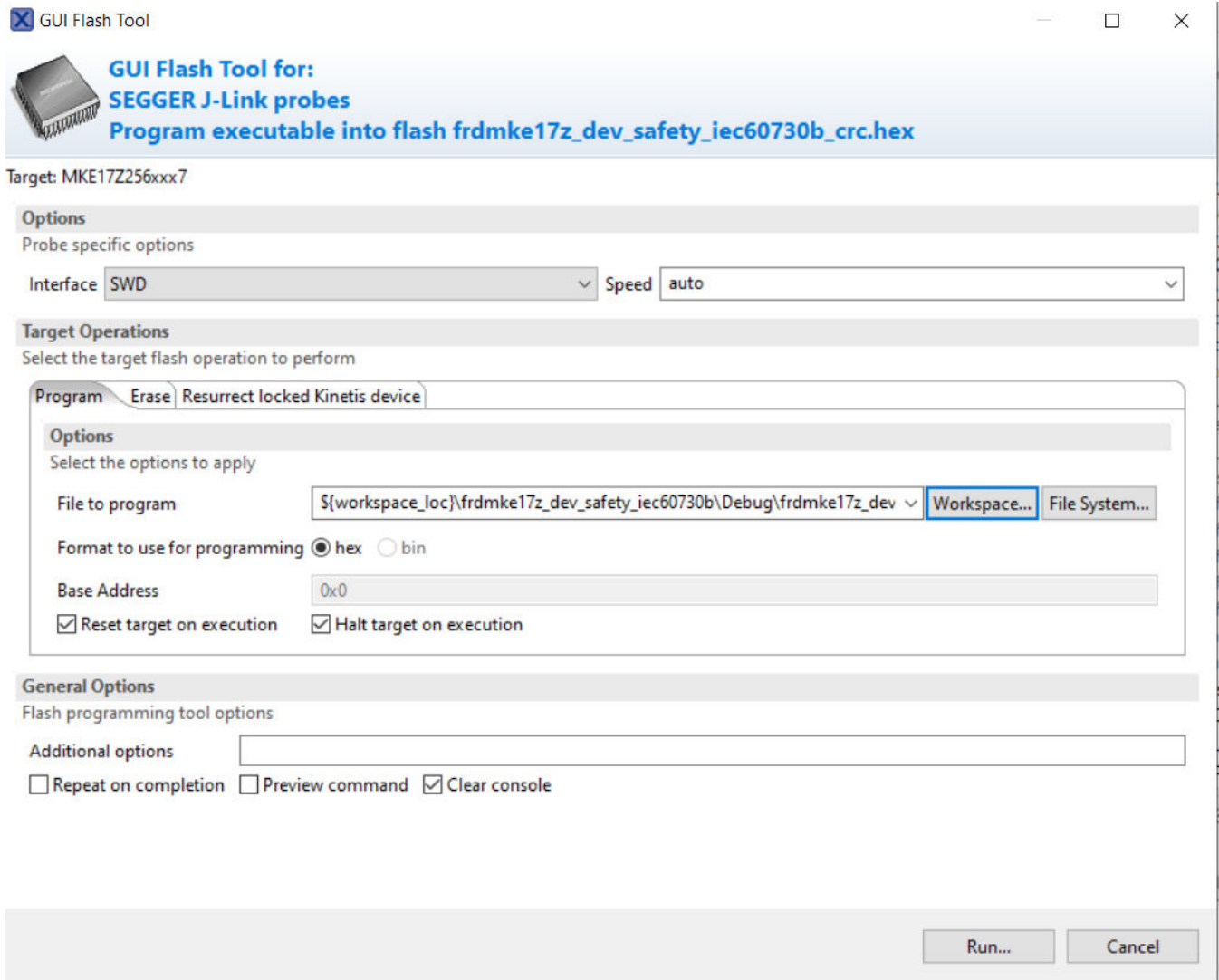


Figure 8. GUI Flash Tool - SEGGER J-Link



# 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 AIO test

The analog IO test procedure performs the plausibility check of the digital IO interface of the processor. The analog IO test can be performed once after the MCU reset and also during runtime.

There are three values tested in the application:

- VrefH
- VrefL
- Bandgap

Ensure that the ADC peripheral is set up correctly before calling the AIO test. In some cases, it is necessary to connect this signal externally (by a wire) to the corresponding pin. The test is performed in a sequence, as defined in the *safety\_config.h* file:

```
/* ADC test */
...
...
{
    {(uint32_t)ADC_MIN_LIMIT(0), (uint32_t)ADC_MAX_LIMIT(60)}, |
    {(uint32_t)ADC_MIN_LIMIT(ADC_MAX), (uint32_t)ADC_MAX_LIMIT(ADC_MAX)}, |
    {(uint32_t)ADC_MIN_LIMIT(ADC_BANDGAP_LEVEL_RAW),
    (uint32_t)ADC_MAX_LIMIT(ADC_BANDGAP_LEVEL_RAW)} | }

#define FS_CFG_AIO_CHANNELS_INIT {6, 5, 4} /* ADC Channels for V_refl, V_refh, bandgap */
```

An example of the setting is shown above. The "FS\_CFG\_AIO\_CHANNELS\_INIT" macro defines that the ADC channel 6 is tested first, with the limits corresponding to VrefL (GND). Channel 5 is tested next, with the limits of VrefH (VCC). Channel 4 is tested next, with the limits for the bandgap.

## 6.2 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.

## 6.3 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.

## 6.4 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.

## 6.5 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 uVision Keil IDE. It is only necessary to turn this test on in the *safety\_config.h* file. In case of any issues, see [Arm uVision 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.

## 6.6 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.

---

## 6.7 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.

---

## 6.8 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.

---

## 6.9 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.

---

## Chapter 7

# Revision history

Table 1. Revision history

Revision number	SDK	Description
0	2.9.0	Initial 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](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: 07/2021

Document identifier: IEC60730BKCM4CM7EUG

