UG10319

IEC60730B Example User Guide

Rev. 1.0 — 8 September 2025 **User guide**

Document information

Information	Content
Keywords	UG10319, IEC60730B Safety library
Abstract	This example user guide describes how to set the hardware correctly and how to use the example code with the IEC60730B Safety library v5.0.

1 IEC60730B Safety library example user guide

For easier development of the IEC60730B application, the library also provides the example code. This example is distributed through the MCUXpresso SDK website and GitHub. This example user guide describes how to set the hardware correctly and how to use the example code with the IEC60730B Safety library v5.0.

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

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 examples support the following development boards:

- FRDM-MCXA156
- FRDM-MCXA266
- FRDM-MCXA366
- FRDM-MCXA346
- FRDM-MCXC444
- FRDM-MCXE247
- FRDM-MCXE31B
- MCIMX93-EVK

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

2.1 MCIMX93-EVK

The hardware requirements are as follows:

- i.MX93xx Mini development board (MCIMX93-EVK)
- J-Link debug probe
- USB-C cable
- USB-C 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 (J301 USB port) and the J-Link debug probe to the board (J1405 JTAG header) and switch the SW301 switch to power on the board.
2. If a debug print is needed, connect a USB-C cable between the host PC and the J1401 USB port on the target board. Open the serial terminal with the following settings:
   - 9600 baud rate
   - 8 data bits
   - No parity
   - One stop bit

- No flow control

3. Set the SW1301 switch to value "1101" to boot from the FlexSPI Serial NOR.

See www.nxp.com/imx93evk for more information.

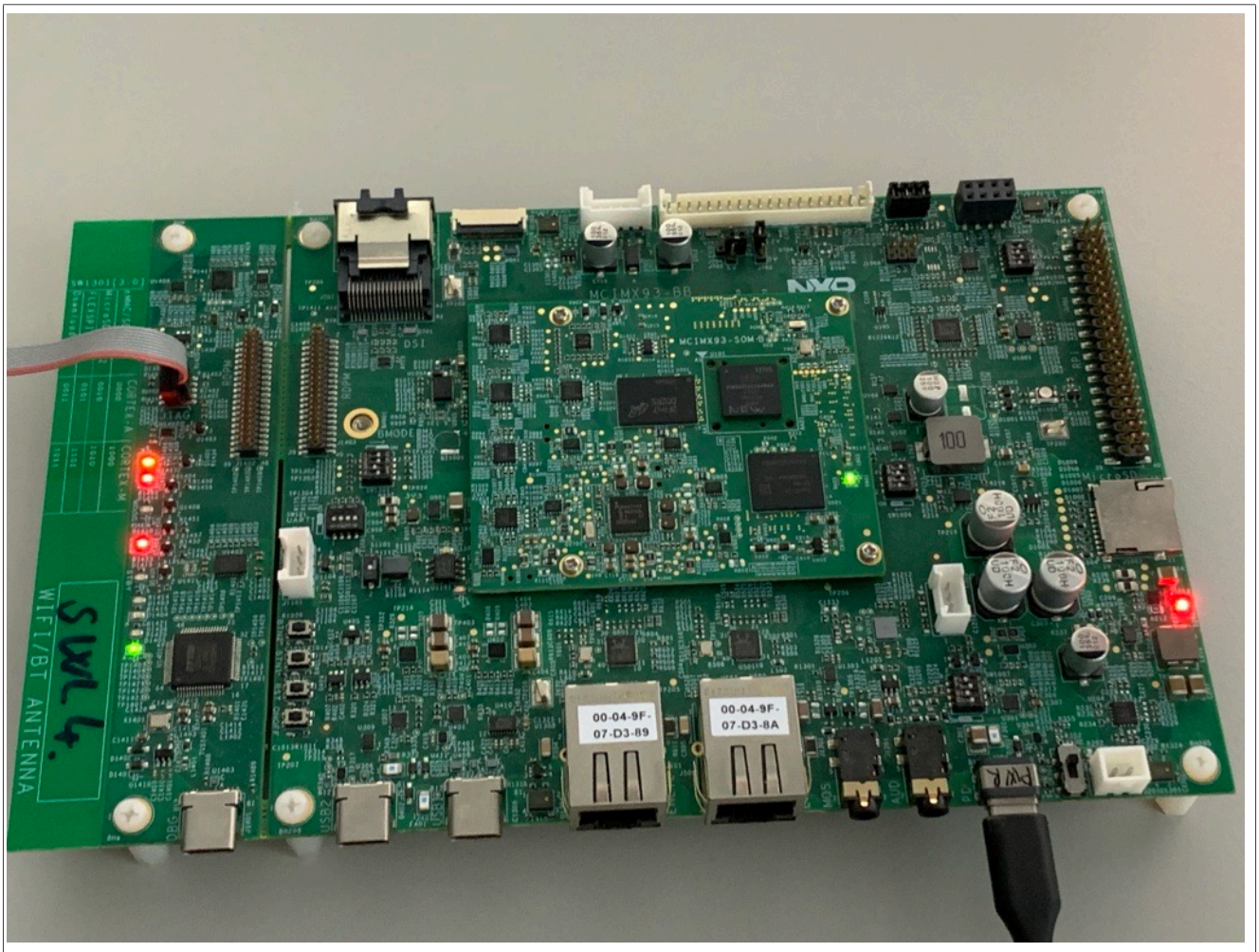

Figure 1. Hardware connection of EVK-MIMX93

2.2 FRDM-MCXA156

Debugger:

To use the on-board debugger and power the board via USB, connect the USB to connector J15.

The default debugger in the example project is set to CMSIS-DAP. Make sure to select SWD as the debugger connection interface.

FreeMASTER

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

See the NXP website for more power and boot options.

The ADC module on the FRDM-MCXA156 does not allow the VrefL, VrefH, and Bandgap to connect internally to the ADC input. Connect these signals (for the Analog I/O test) as follows:

- VrefL - connect GND to Arduino_A0 (J4-2).
- VrefH - connect VCC to Arduino_A1 (J4-4).
- Bandgap - connect a custom reference (for example 1.65 V) to Arduino_A2 (J4-6). The expected value of the custom Bandgap can be set in the *safety_config.h* file (`#define ADC_BANDGAP_LEVEL 1.65`).

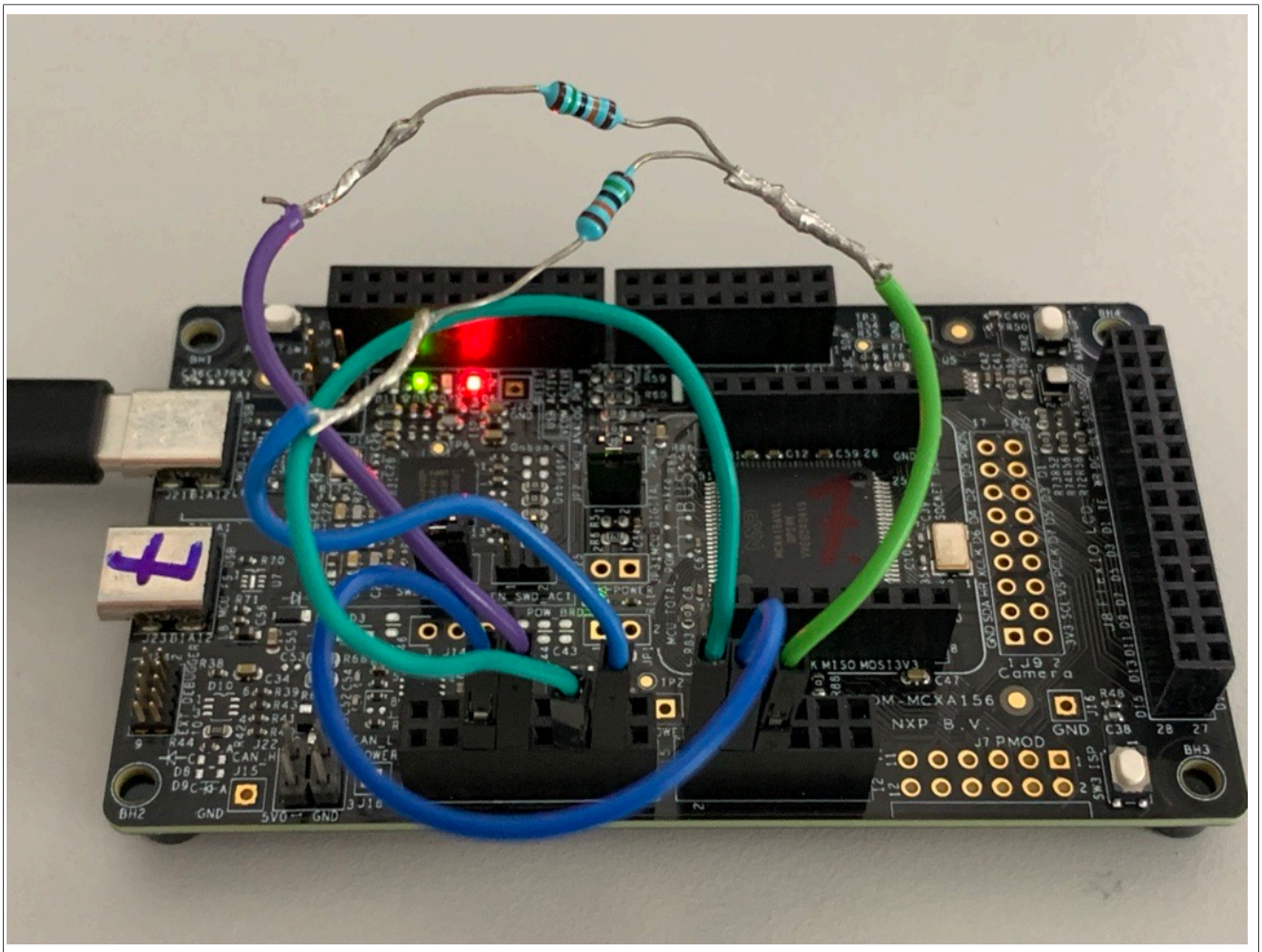

Figure 2. Hardware connection of FRDM-MCXA156

The test voltage of 1.65 V is provided by the resistor voltage divider from the VCC (3.3 V).

2.3 FRDM-MCXA266

Debugger:

To use the onboard debugger and power the board via USB, connect the USB to connector J15.

The default debugger in the example project is set to CMSIS-DAP. Make sure to select SWD as the debugger connection interface.

FreeMASTER

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

See the NXP website for more power and boot options.

The ADC module on the FRDM-MCXA266 does not allow the VrefL, VrefH, and Bandgap to connect internally to the ADC input. Connect these signals (for the Analog I/O test) as follows:

- VrefL - connect GND to Arduino_A0 (J4-2).
- VrefH - connect VCC to Arduino_A1 (J4-4).
- Bandgap - connect a custom reference (for example 1.65 V) to Arduino_A2 (J4-6). The expected value of the custom Bandgap can be set in the *safety_config.h* file (`#define ADC_BANDGAP_LEVEL 1.65`).

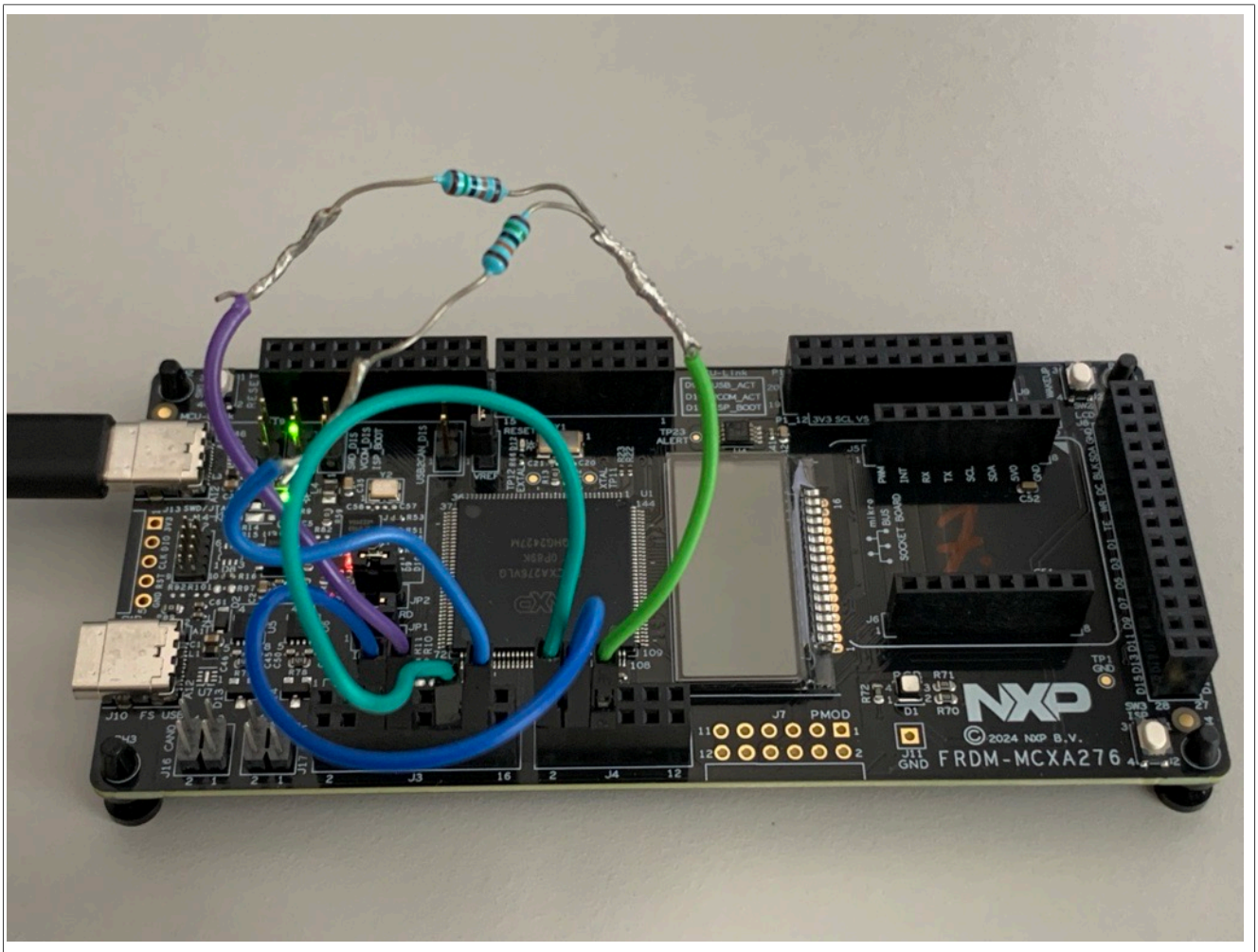

Figure 3. Hardware connection of FRDM-MCXA266

The test voltage of 1.65 V is provided by the resistor voltage divider from the VCC (3.3 V).

2.4 FRDM-MCXA366

Debugger:

To use the onboard debugger and power the board via USB, connect the USB to connector J15.

The default debugger in the example project is set to CMSIS-DAP. Make sure to select SWD as the debugger connection interface.

FreeMASTER

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

See the NXP website for more power and boot options.

The ADC module on the FRDM-MCXA366 does not allow the VrefL, VrefH, and Bandgap to connect internally to the ADC input. Connect these signals (for the Analog I/O test) as follows:

- VrefL - connect GND to Arduino_A0 (J4-2).
- VrefH - connect VCC to Arduino_A1 (J4-4).
- Bandgap - connect a custom reference (for example 1.65 V) to Arduino_A2 (J4-6). The expected value of the custom Bandgap can be set in the *safety_config.h* file (`#define ADC_BANDGAP_LEVEL 1.65`).

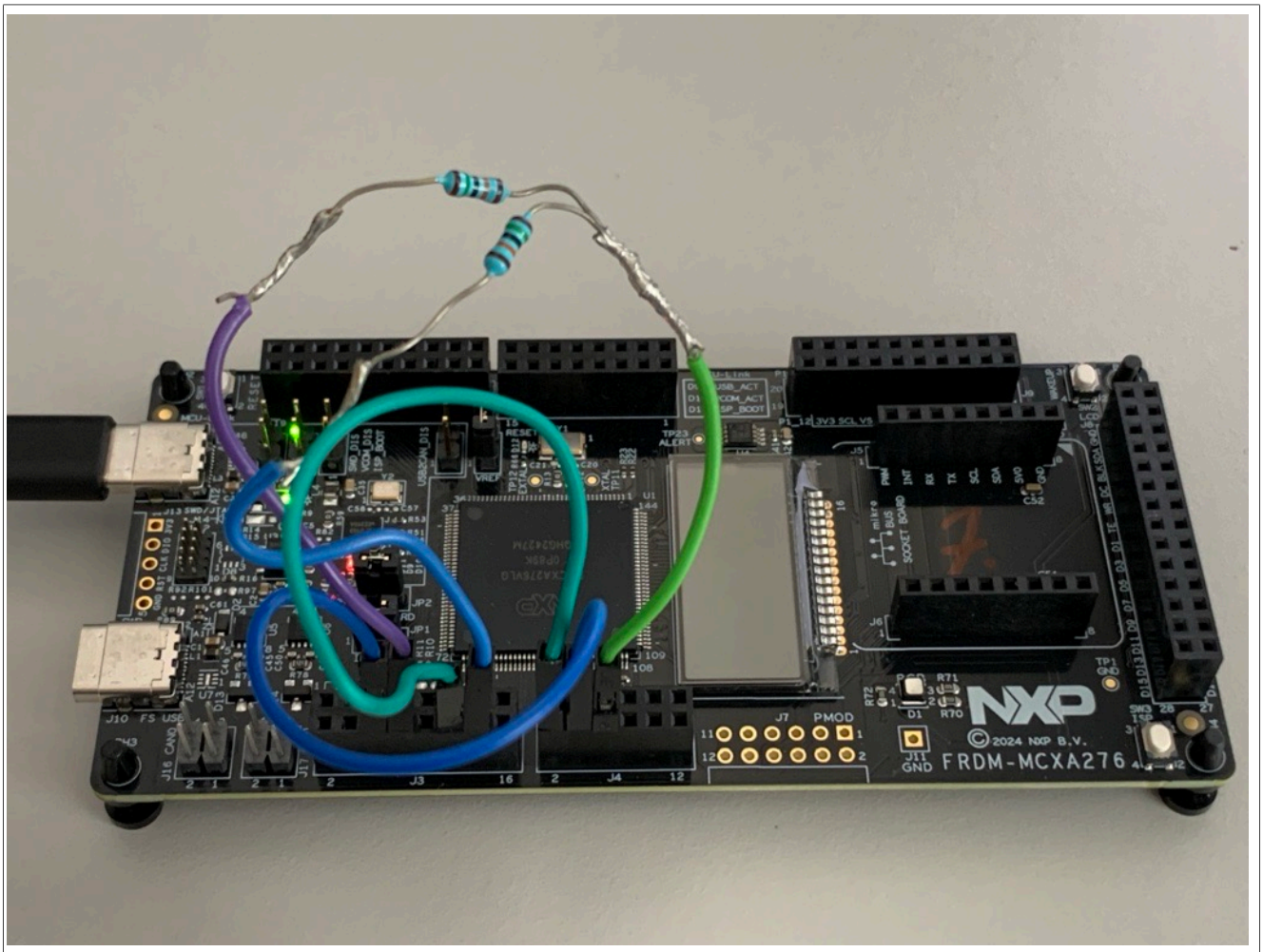

Figure 4. Hardware connection of FRDM-MCXA366

The test voltage of 1.65 V is provided by the resistor voltage divider from the VCC (3.3 V).

2.5 FRDM-MCXA346

Debugger:

To use the onboard debugger and power the board via USB, connect the USB to connector J15.

The default debugger in the example project is set to CMSIS-DAP. Make sure to select SWD as the debugger connection interface.

FreeMASTER

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

See the NXP website for more power and boot options.

The ADC module on the FRDM-MCXA346 does not allow the VrefL, VrefH, and Bandgap to connect internally to the ADC input. Connect these signals (for the Analog I/O test) as follows:

- VrefL - connect GND to Arduino_A0 (J4-2).
- VrefH - connect VCC to Arduino_A1 (J4-4).
- Bandgap - connect a custom reference (for example 1.65 V) to Arduino_A2 (J4-6). The expected value of the custom Bandgap can be set in the *safety_config.h* file (`#define ADC_BANDGAP_LEVEL 1.65`).

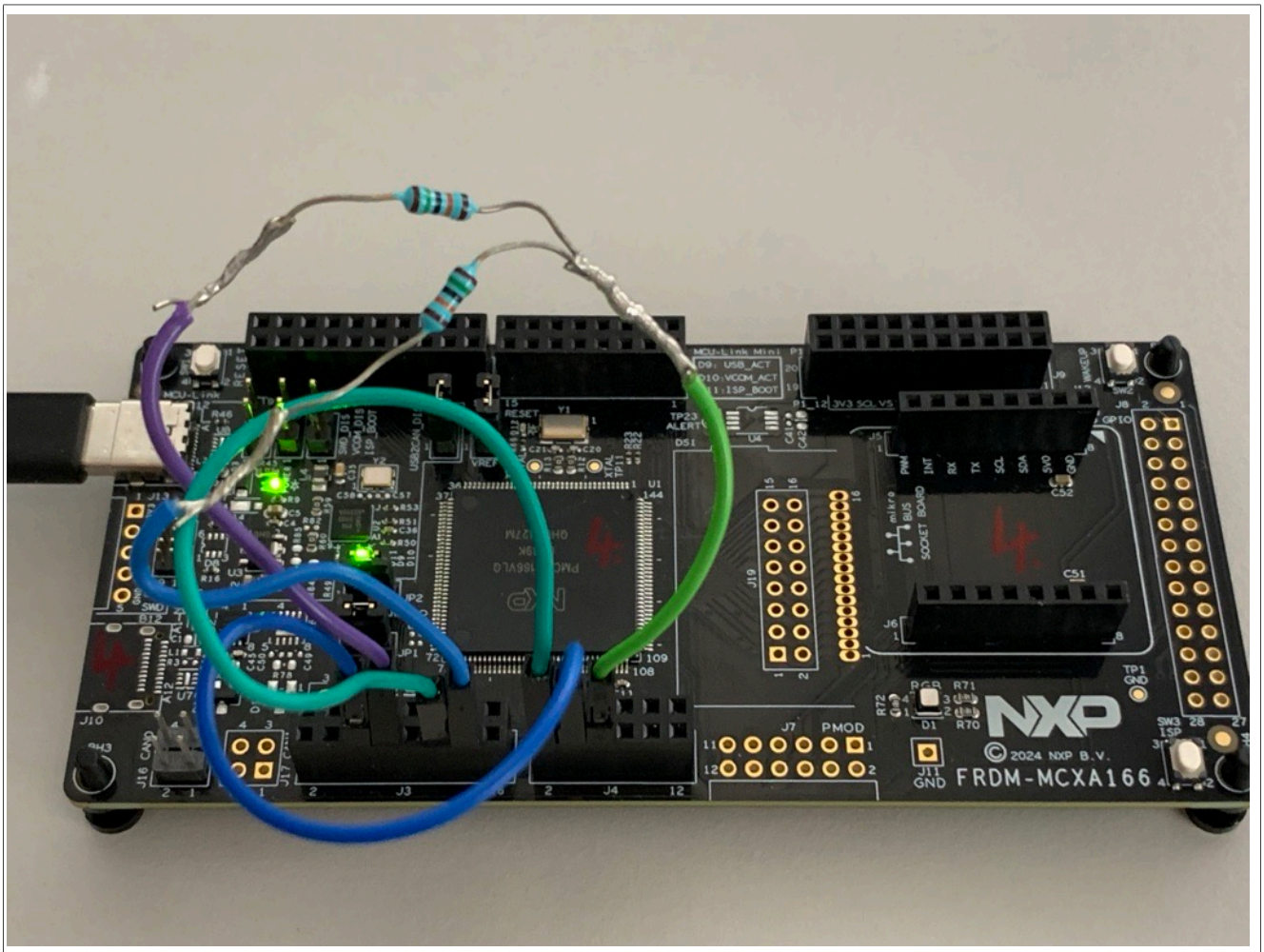

Figure 5. Hardware connection of FRDM-MCXA346

The test voltage of 1.65 V is provided by the resistor voltage divider from the VCC (3.3 V).

2.6 FRDM-MCXC444

Debugger:

To use the onboard debugger and power the board via USB, connect the USB to connector J15.

The default debugger in the example project is set to CMSIS-DAP. Make sure to select SWD as the debugger connection interface.

FreeMASTER

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

See the NXP website for more power and boot options.

The ADC module on the FRDM-MCXC444 does not allow the VrefL, VrefH, and Bandgap to connect internally to the ADC input. Connect these signals (for the Analog I/O test) as follows:

- VrefL - connect GND to Arduino_A0 (J4-2).
- VrefH - connect VCC to Arduino_A1 (J4-4).
- Bandgap - connect a custom reference (for example 1.65 V) to Arduino_A2 (J4-6). The expected value of the custom Bandgap can be set in the *safety_config.h* file (`#define ADC_BANDGAP_LEVEL 1.65`).

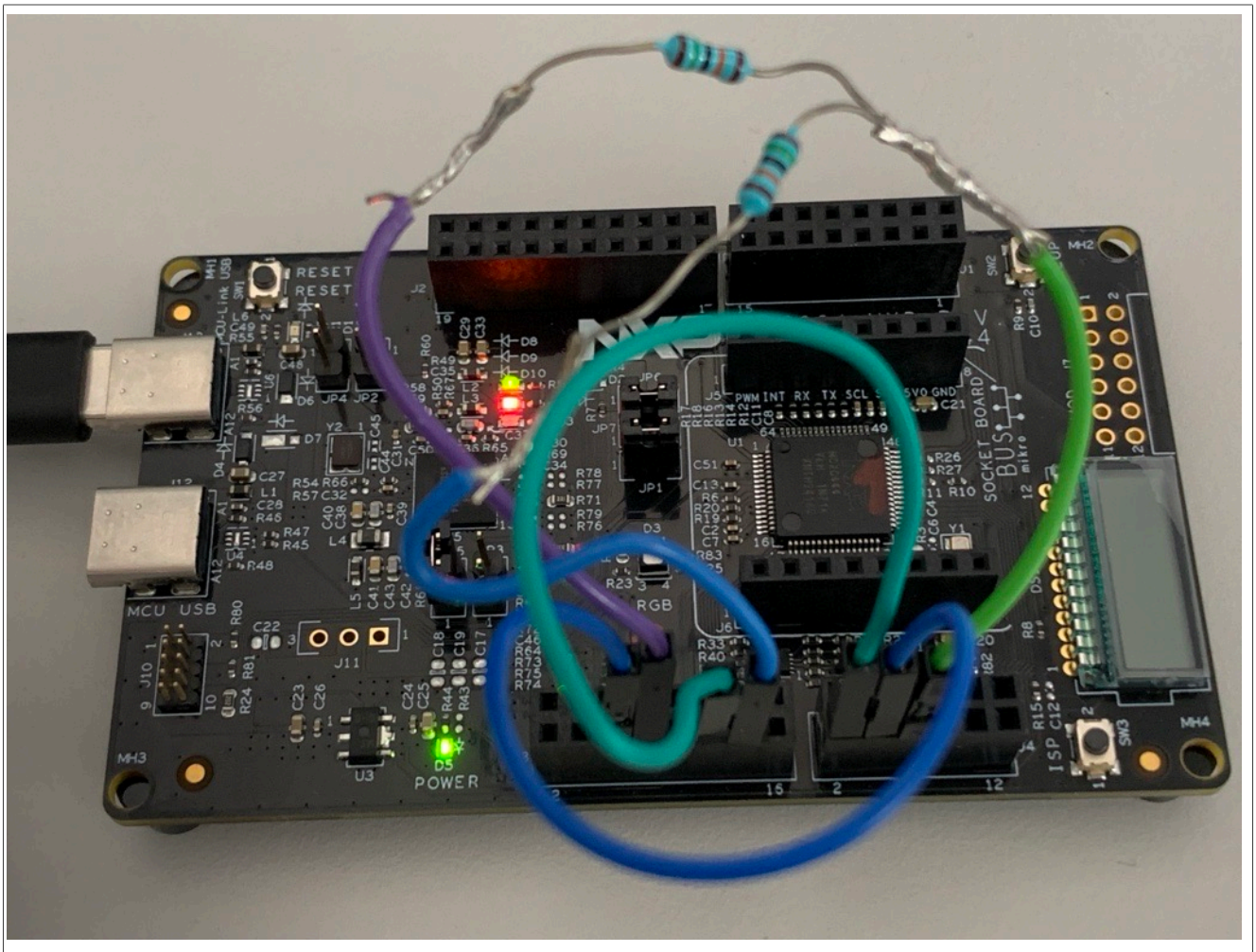

Figure 6. Hardware connection of FRDM-MCXC444

The test voltage of 1.65 V is provided by the resistor voltage divider from the VCC (3.3 V).

2.7 FRDM-MCXE247

Debugger:

To use the onboard debugger and power the board via USB, connect the USB to connector J15.

The default debugger in the example project is set to CMSIS-DAP. Make sure to select SWD as the debugger connection interface.

FreeMASTER

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

See the NXP website for more power and boot options.

The ADC module on the FRDM-MCXE247 does not allow the VrefL, VrefH, and Bandgap to connect internally to the ADC input. Connect these signals (for the Analog I/O test) as follows:

- VrefL - connect GND to Arduino_A0 (J4-2).
- VrefH - connect VCC to Arduino_A1 (J4-4).
- Bandgap - connect a custom reference (for example 1.65 V) to Arduino_A2 (J4-6). The expected value of the custom Bandgap can be set in the *safety_config.h* file (`#define ADC_BANDGAP_LEVEL 1.65`).

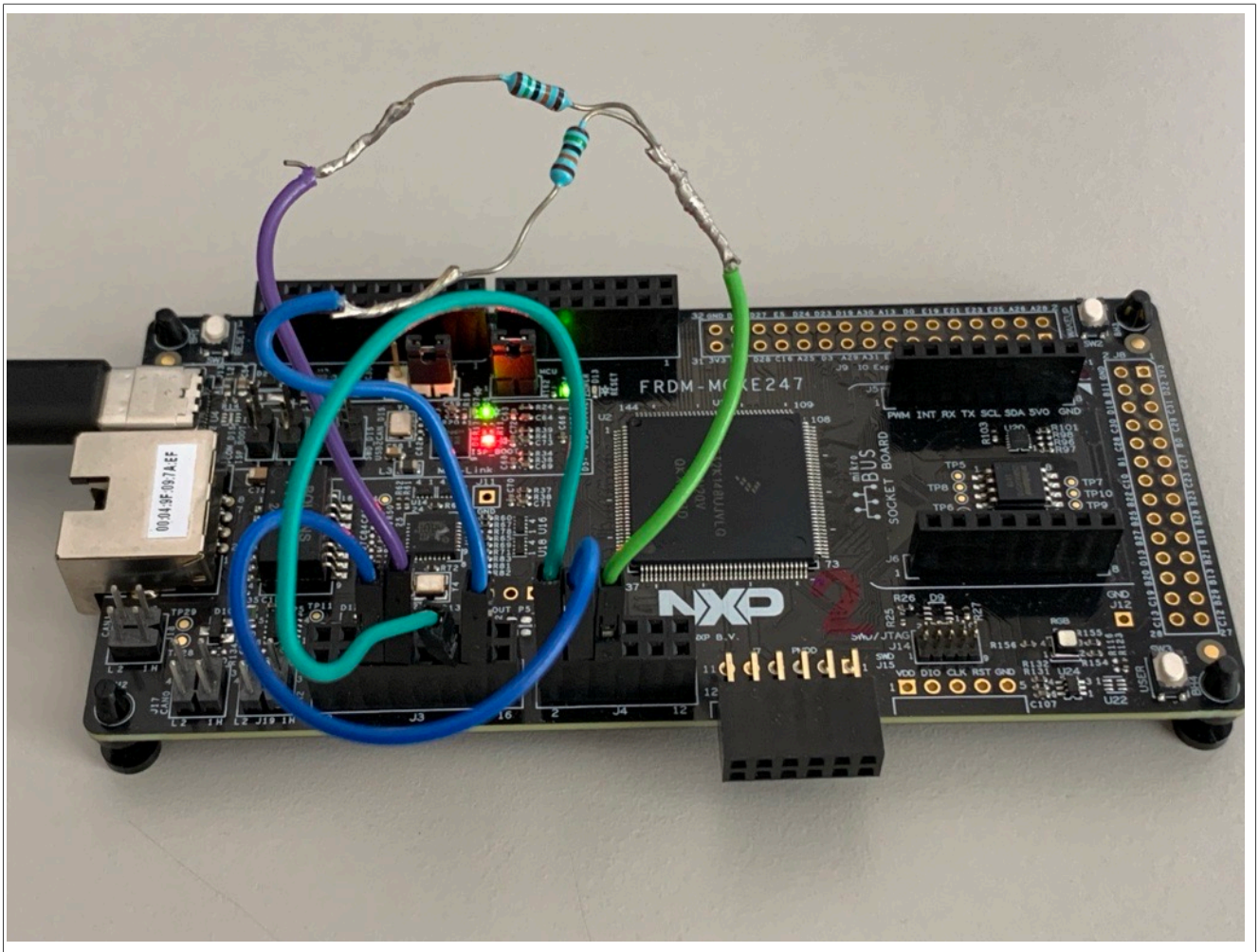

Figure 7. Hardware connection of FRDM-MCXE247

The test voltage of 1.65 V is provided by the resistor voltage divider from the VCC (3.3 V).

2.8 FRDM-MCXE31B

Debugger:

To use the onboard debugger and power the board via USB, connect the USB to connector J15.

The default debugger in the example project is set to CMSIS-DAP. Make sure to select SWD as the debugger connection interface.

FreeMASTER

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

See the NXP website for more power and boot options.

The ADC module on the FRDM-MCXE31B does not allow the VrefL, VrefH, and Bandgap to connect internally to the ADC input. Connect these signals (for the Analog I/O test) as follows:

- VrefL - connect GND to Arduino_A0 (J4-2).
- VrefH - connect VCC to Arduino_A1 (J4-4).
- Bandgap - connect a custom reference (for example 1.65 V) to Arduino_A2 (J4-6). The expected value of the custom Bandgap can be set in the *safety_config.h* file (`#define ADC_BANDGAP_LEVEL 1.65`).

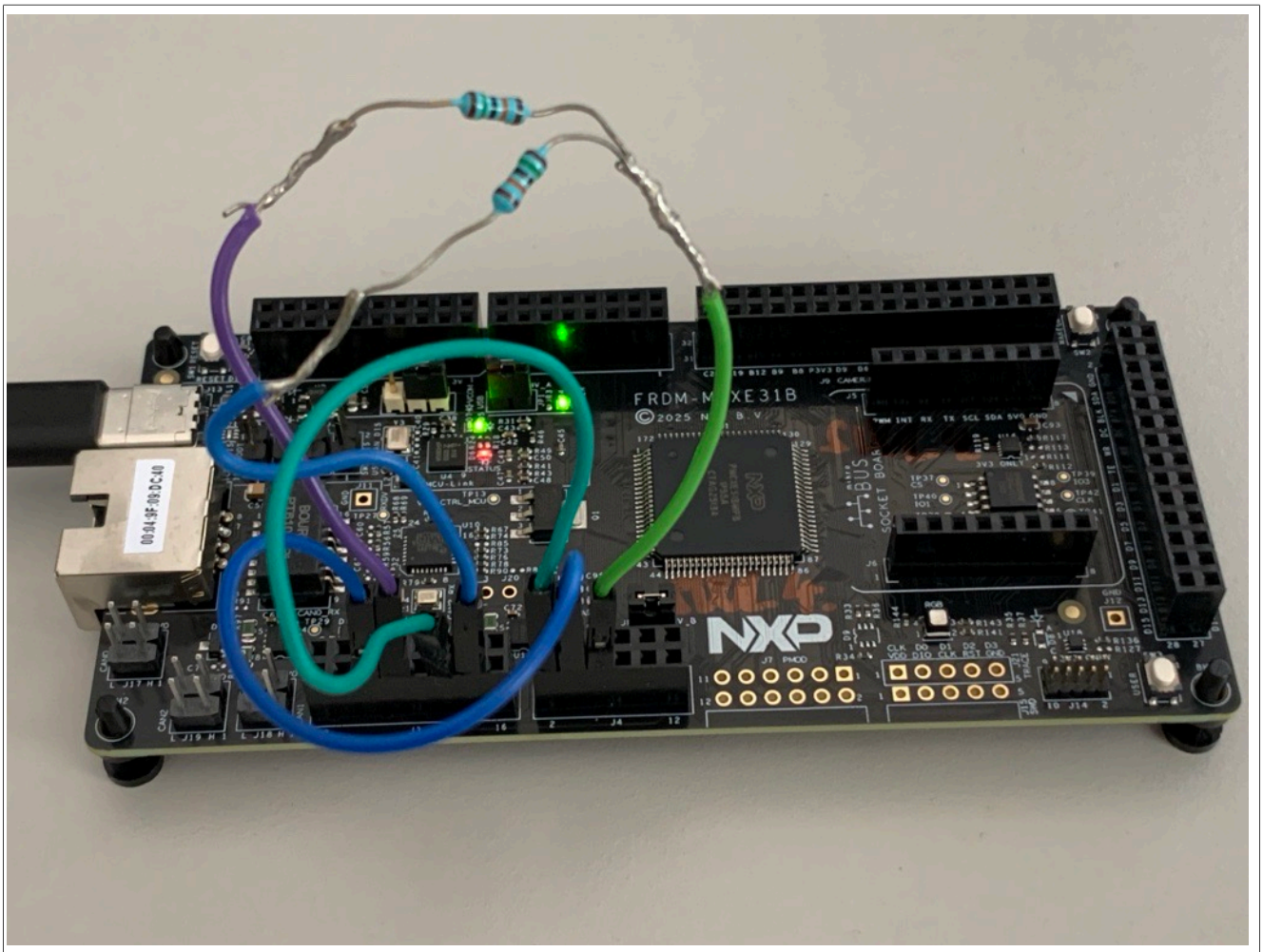

Figure 8. Hardware connection of FRDM-MCXE31B

The test voltage of 1.65 V is provided by the resistor voltage divider from the VCC (3.3 V).

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/source* folder in the SDK package.

The folder has the following structure:

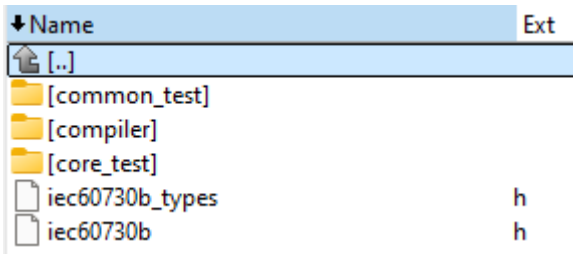

Figure 9. Folder structure

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 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 10.

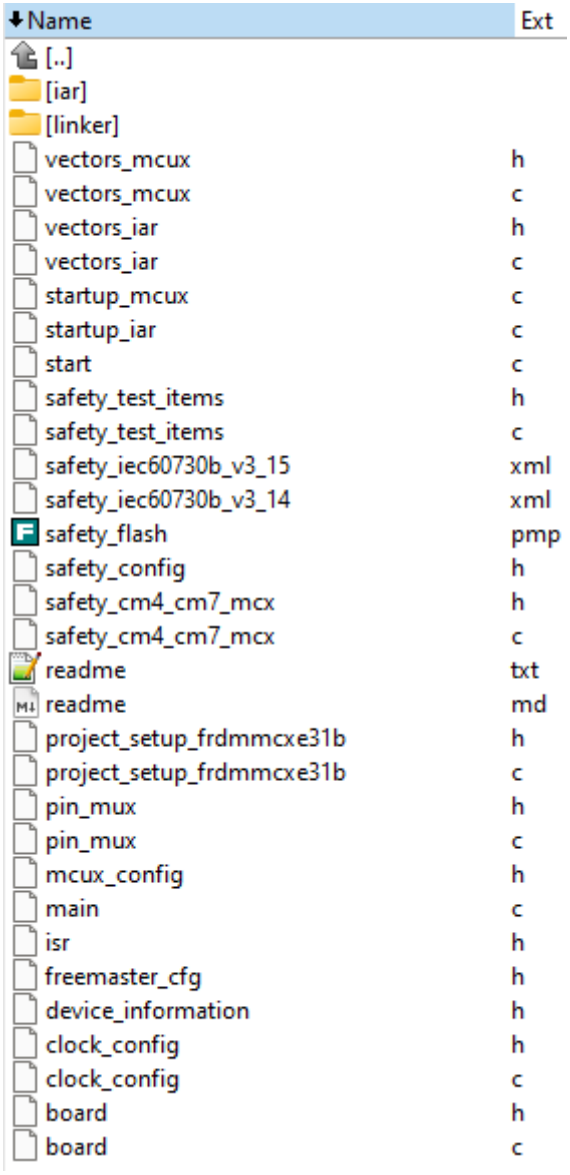

Figure 10. Example of project structure in example folder

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

- *iar*

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.

4 Example application

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

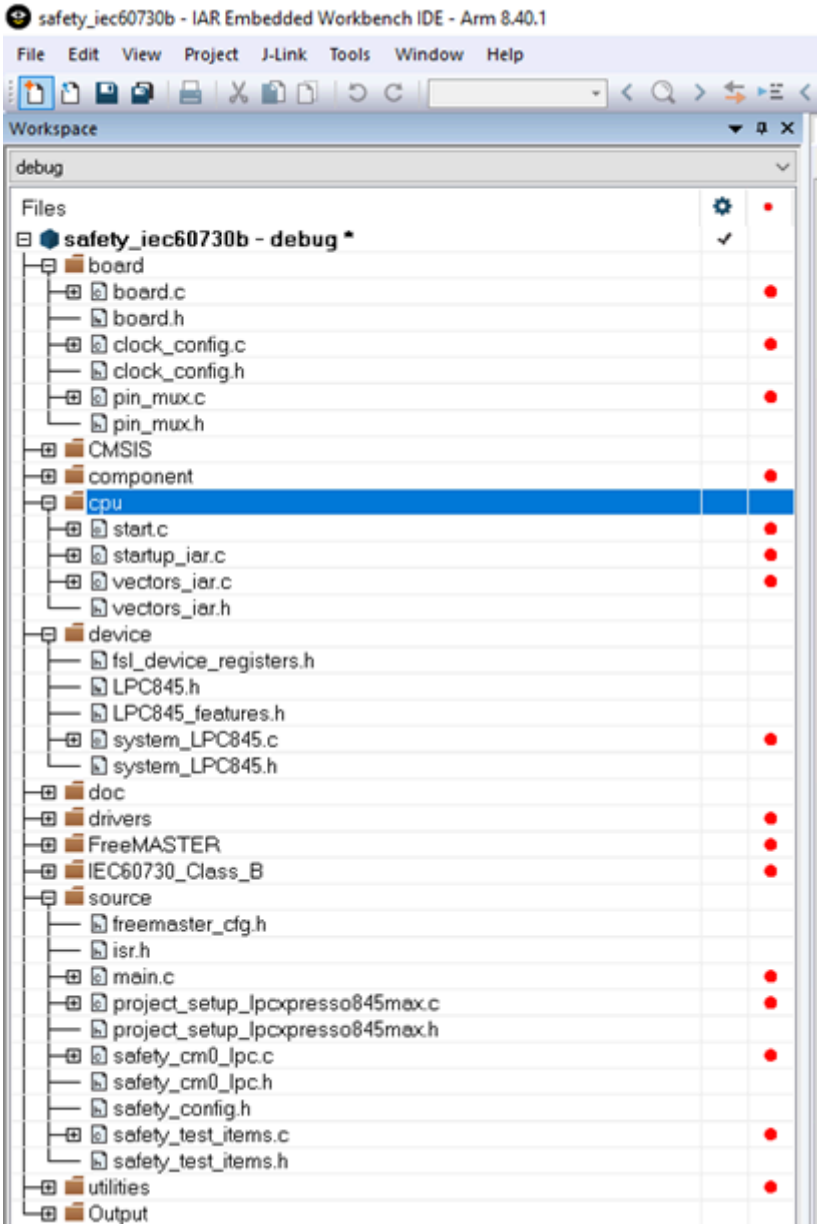

Figure 11. IAR example application structure

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 12.

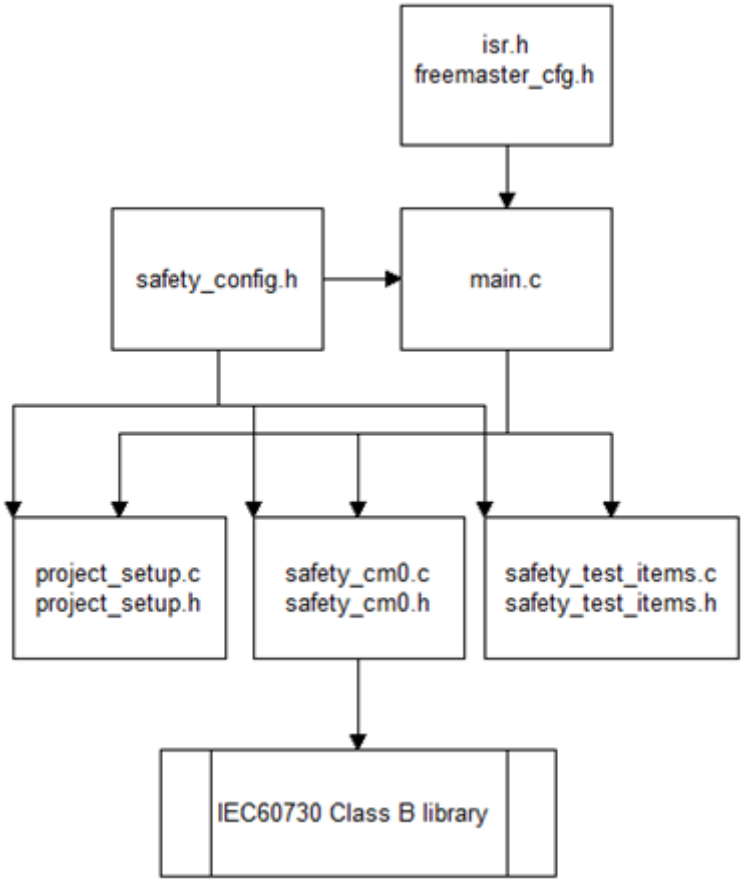

Figure 12. Example of project hierarchy

Figure 12 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_cm0_mcx.c*, *safety_cm4_cm7_mcx.c*, *safety_cm33_mcx.c*, *safety_cm33_mcx.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_cm33_imx.c*, *safety_cm0_mcx.c*, *safety_cm4_cm7_mcx.c*, *safety_cm33_mcx.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

4.1.1 IAR IDE

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

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

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 = GPIOD_BASE,
 .pcr = PORTD_BASE, /* Base address of PCR register */
 .pinNum = 6,
 .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 *g_dio_safety_test_items[] = {&dio_safety_test_item_0,
 &dio_safety_test_item_1, NULL};
```

4.4 Source file - safety_cmXX_YYY.c/.h (safety_cm0_mcx, safety_cm4_cm7_mcx, safety_cm33_mcx, *safety_cm33_imx.c*)

The *safety_cm0_mcx.c*, *safety_cm4_cm7_mcx.c*, *safety_cm33_mcx.c*, *safety_cm33_imx.c* source files and the corresponding **.h* files contain a library-handling function. Each function contains a detection mechanism. If a safety error occurs, the *SafetyErrorHandling()* function is called.

5 Running example

For the first run of the example on your hardware, it is recommended to turn off the Flash, WDOG, Clock, AIO, and DIO tests. In the next step, turn them 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.
- 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.

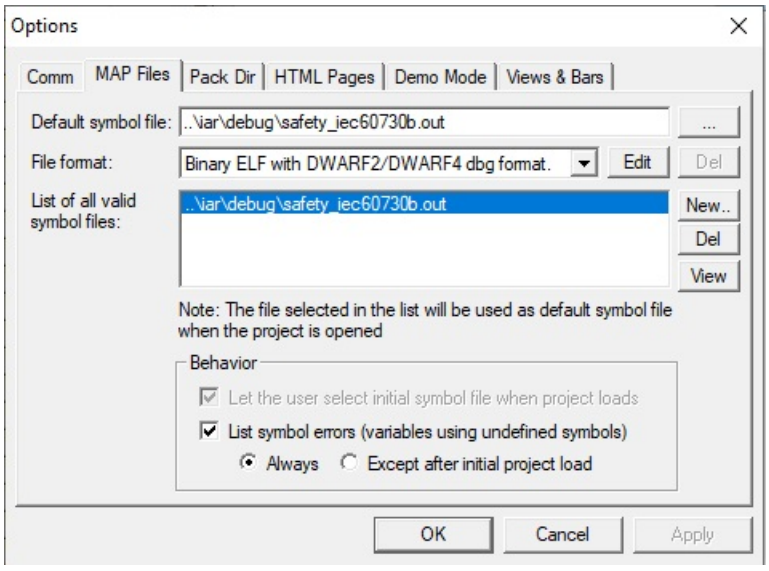

Figure 13. Example of setting MAP files for FRDM-KV11 board

  - 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.

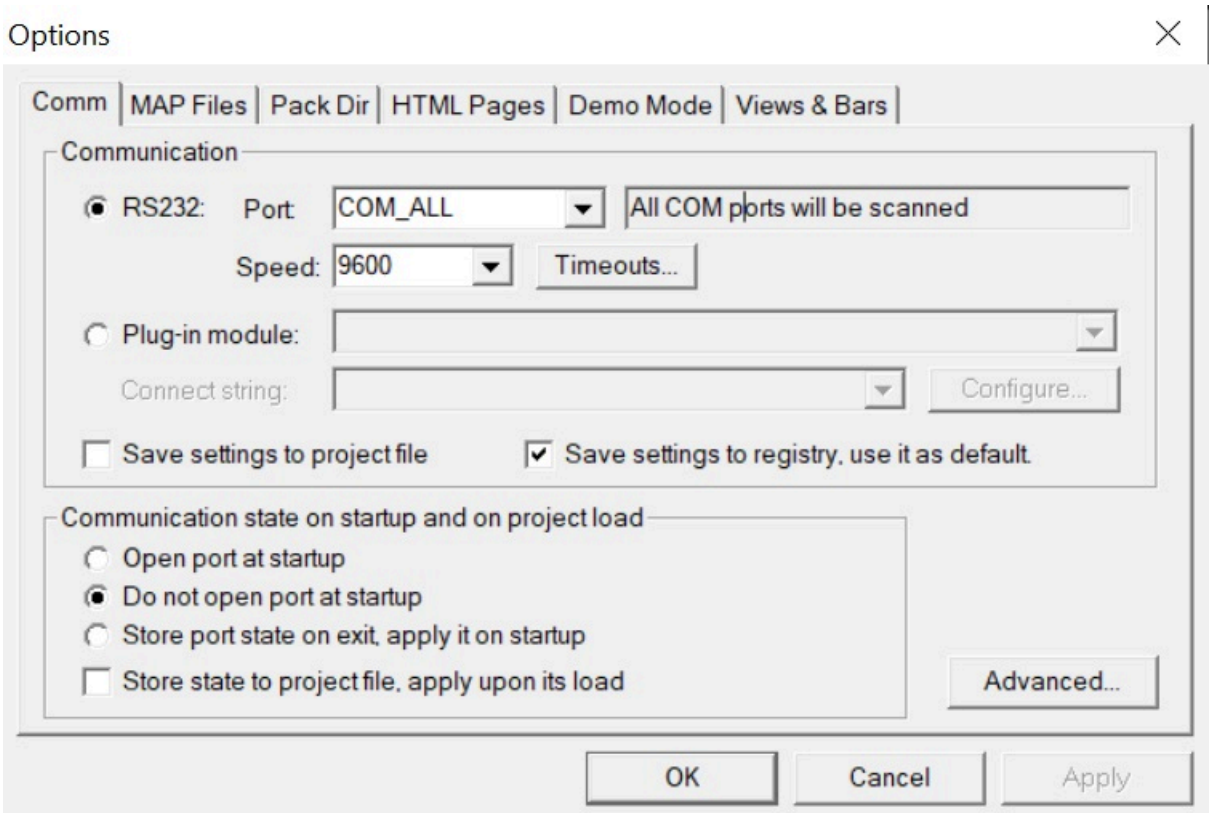

Figure 14. Setting UART speed

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

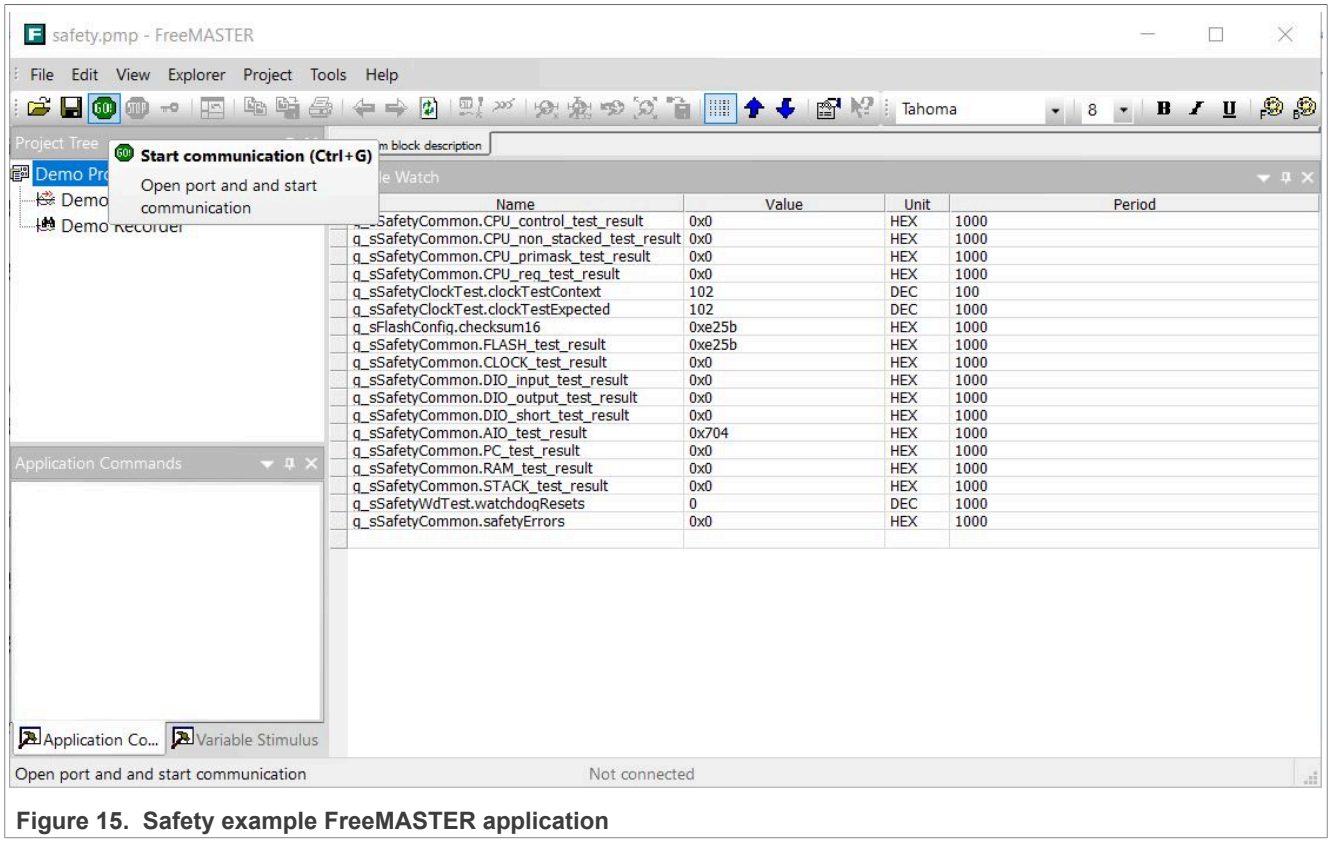

Figure 15. Safety example FreeMASTER application

- 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 (like MCUXpresso and uVision Keil), use the SRecord tool.

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

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

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

5.2.1 IAR IDE post-build CRC

The IAR Embedded Workbench IDE has a built-in feature (*Build Actions*) for executing actions from files or custom commands.

The IDE provides several options for when these actions are executed relative to the build stages:

- *Before compilation*
- *Before linking*
- *After linking*
- *Automatically*

5.2.1.1 Post-build configuration

To view or modify the post-build command go to the "Options → Build Actions → Build Action Configuration" menu.

The following post-build string is set in the project:

For devices supporting CRC16:

```
ielftool --fill 0xFF;c_checksumStart-c_checksumEnd+3 --checksum
 __checksum:2,crc16,0x0; c_checksumStart-c_checksumEnd+3 --verbose "$TARGET_PATH
$" "$TARGET_PATH$"
```

For devices supporting CRC32:

```
ielftool --fill 0xFF;c_checksumStart-c_checksumEnd+3 --checksum
 __checksum:4,crc32:i,0xFFFFFFFF;c_checksumStart-c_checksumEnd+3 --verbose
 $TARGET_PATH$ $TARGET_PATH$
```

This command performs two main operations on the **.elf* file after the build:

- `--fill 0xFF+c_checksumStart-c_checksumEnd+3` fills the memory region from `c_checksumStart` to `c_checksumEnd+3` with the value of "0xFF".
- `--checksum__checksum:2,crc16,0x0;c_checksumStart-c_checksumEnd+3` calculates a CRC16 checksum over the same region and inserts the result at the "__checksum" symbol.

The rest of the command (`--verbose "$TARGET_PATH$" "$TARGET_PATH$`) enables a detailed output for debugging and specifies the input and output file paths.

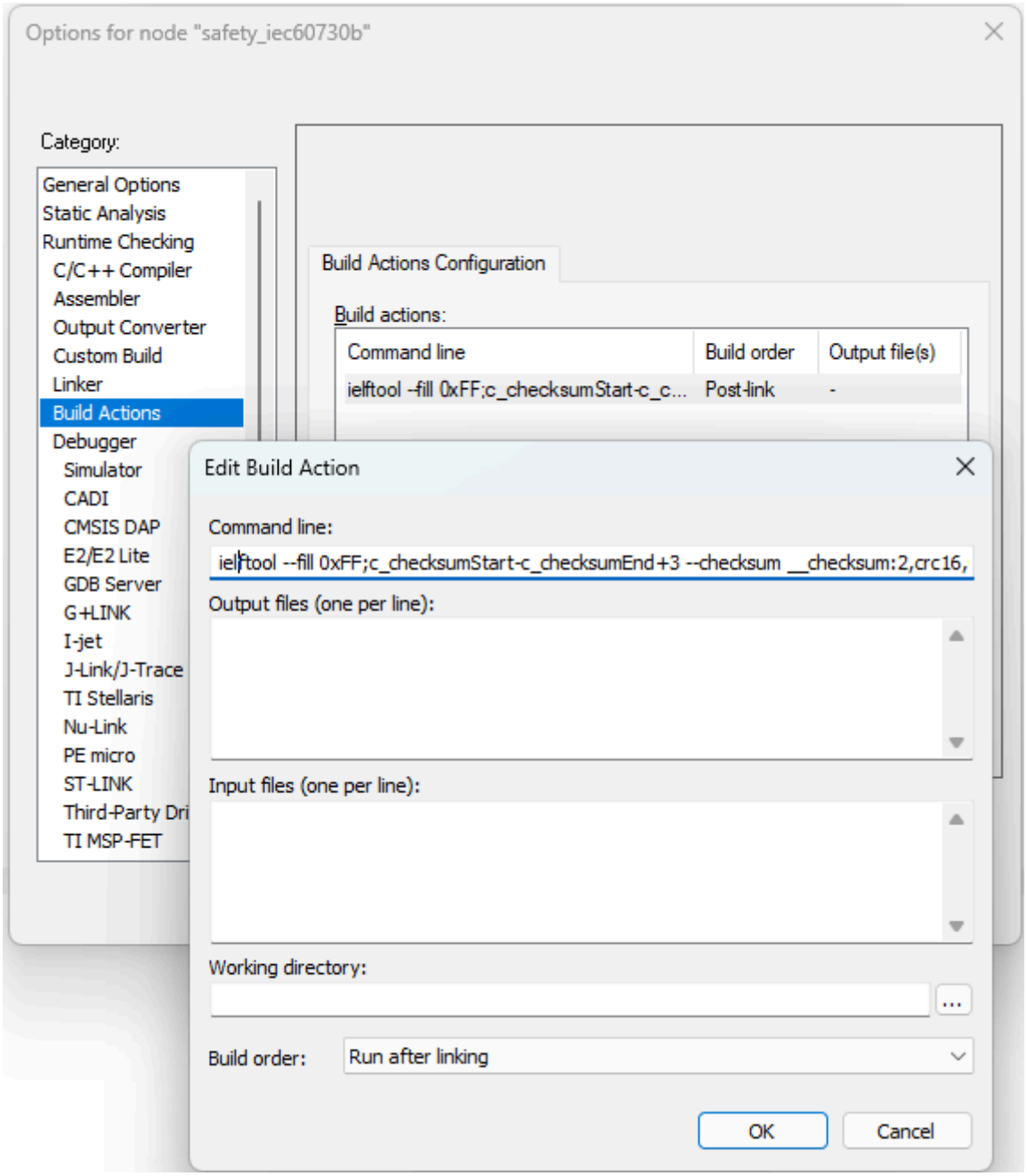

Figure 16. Configuration of post-build steps

5.2.1.2 IAR linker settings

The command in the above example expects the "**checksum_start_mark**" and "**checksum_end_mark**" memory sections to be defined in the linker.

This can be set as follows:

```
define block SAFETY_FLASH_BLOCK with alignment = 8, fixed order
{
 readonly section checksum_start_mark,
```

```
 section .text object main.o,
 section .text object safety_cm33_mcx.o,
 section .rodata object safety_cm33_mcx.o,
 readonly section checksum_end_mark
};
```

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 (if supported for the device) 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 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 necessary to turn this test on in the *safety_config.h* file. In case of any issues, see Arm uVison Keil IDE post-build CRC.

In the MCUXpresso IDE, the CRC is calculated by the Srecord third-party tool. You must perform some additional steps. For more information, see MCUxpresso post-build CRC.

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

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 the 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 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 runs 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, keep the WDOG_backup variable in a part of memory that 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.*

7 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

8 Revision history

Table 1. Revision history

Document ID	Release date	Description
UG10319 v. 1.0	8 September 2025	• Initial version

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents