NXP Semiconductors
User's Guide

Document identifier: IEC60730BKCM0SEUG
Rev. 0, 06/2020

Kinetis CM0 Safety Example

Contents

Chapter 1
IEC60730B Safety library example user's guide

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

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

Chapter 2
Hardware settings

This chapter describes setting up the hardware of the evaulation board. The MCU peripherals' setup is described later.

The IEC60730B library example for the Kinetis CM0 family supports the following development boards:

- FRDM-KV11z

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

2.1 FRDM-KV11z

To use the on-board debugger on FRDM-KV11z, make sure that jumper J10 is in the default (1-2) position. There are no other hardware settings necessary to run the IEC60730B example code.

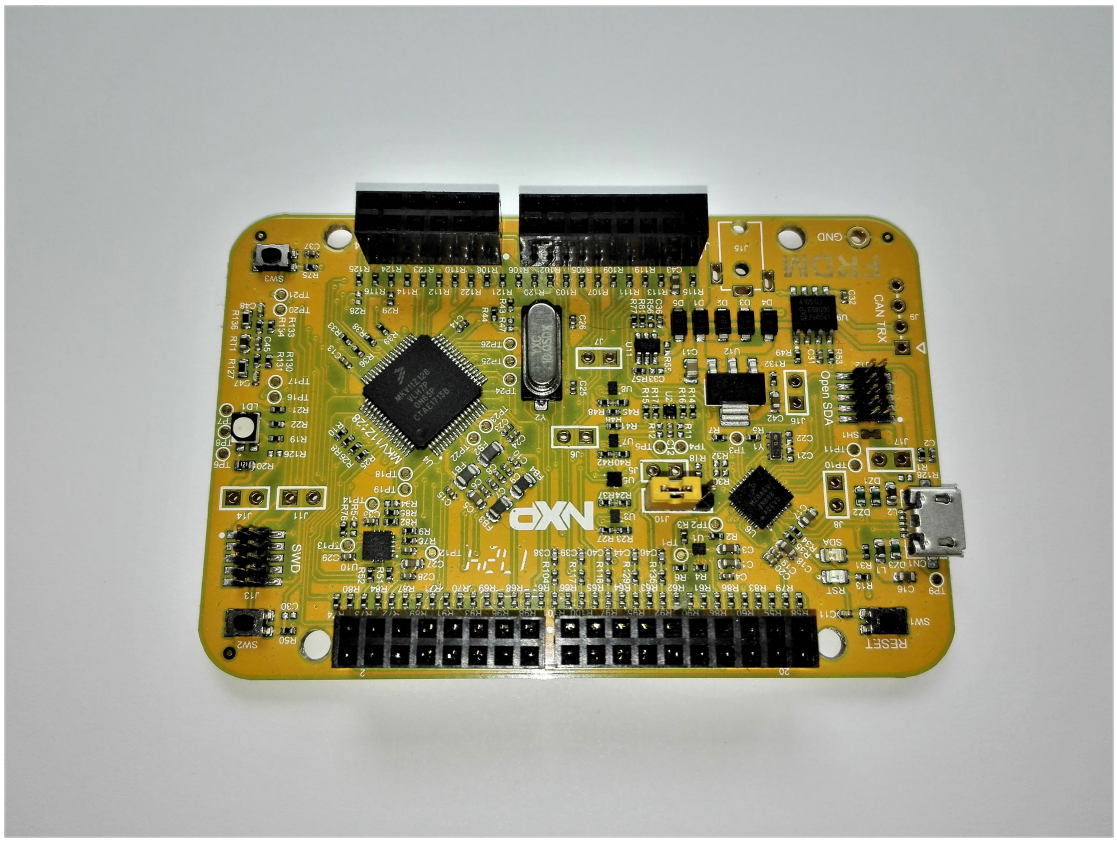


Figure 1. FRDM-KV11z

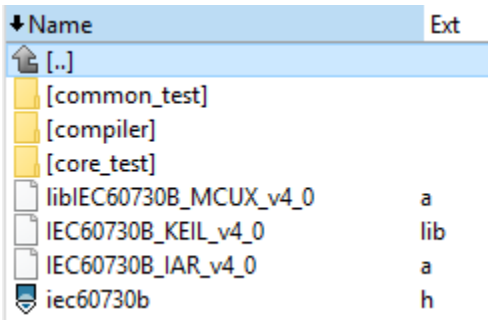
Chapter 3
File structure

Safety is only a small part of the whole SDK package for your device. The source files of the IEC60730 library and examples are in the middleware and in the board.

3.1 Library source files location

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

The folder has this structure:



Where:

- common_test contains the source files for the “peripheral” test – this is a common cross core.
- compiler - compiler support files.
- core_test contains the source files for the core-dependent test.
- *iec60730b.h* is the main library header file.

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

3.2 Example of library handling code

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

boards/<your board>/demo_apps/safety_iec60730b/

The safety example code is shown in Figure 2.

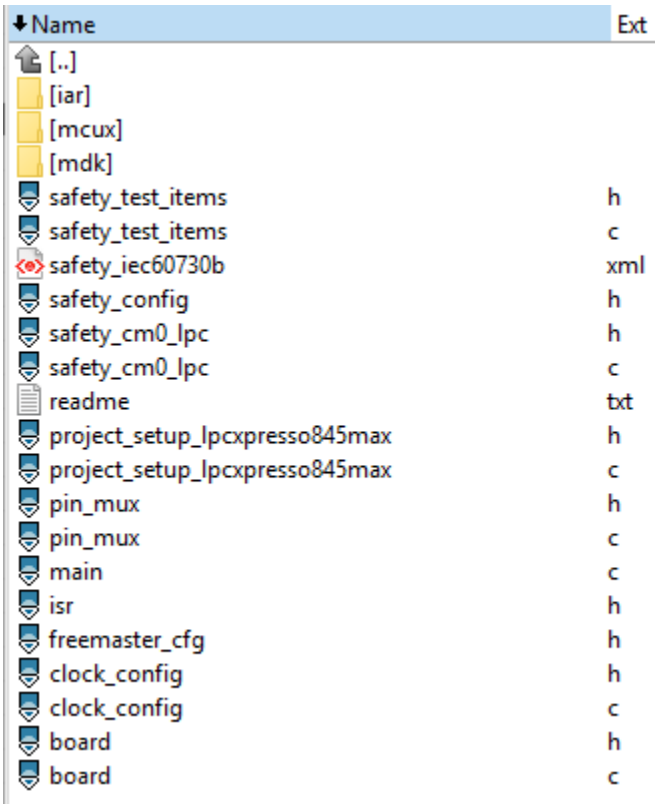


Figure 2. Example of project structure in example folder

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).

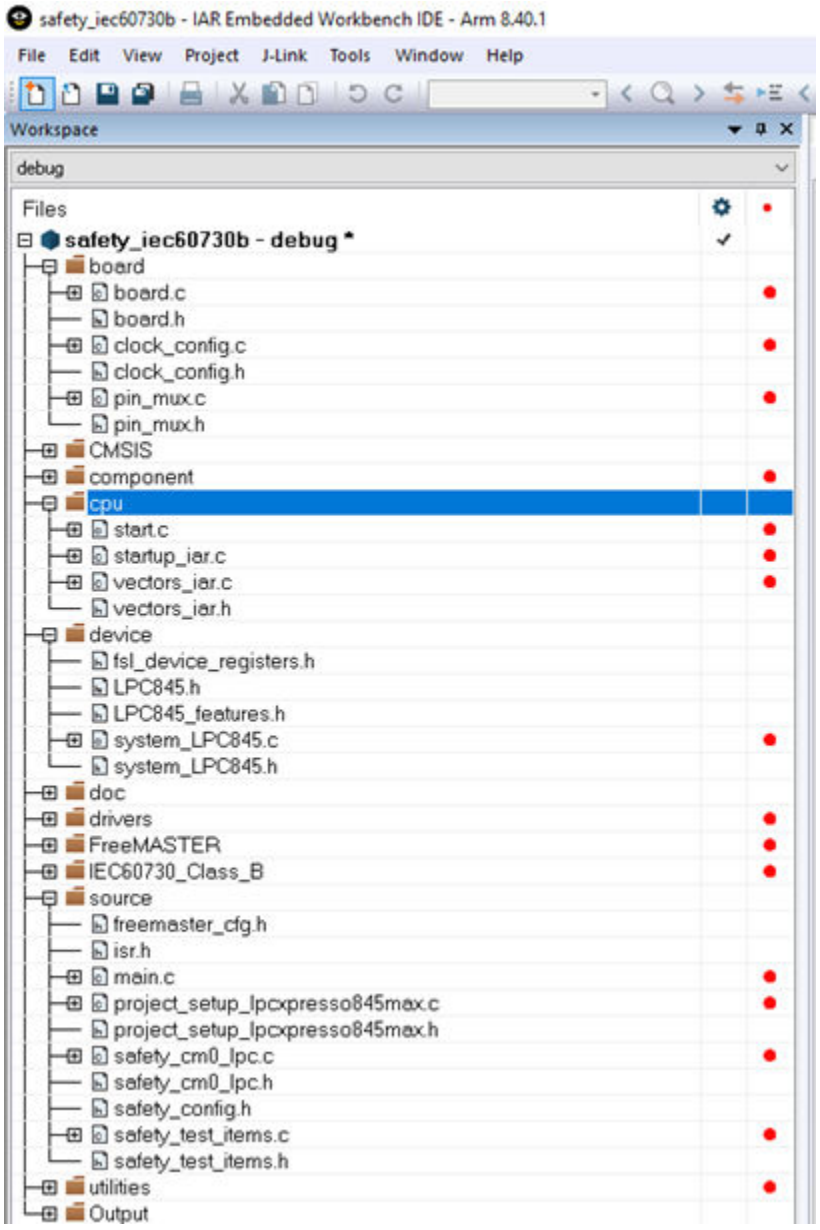


Figure 3. IAR example application structure

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 project hiearchy is shown in Figure 4.

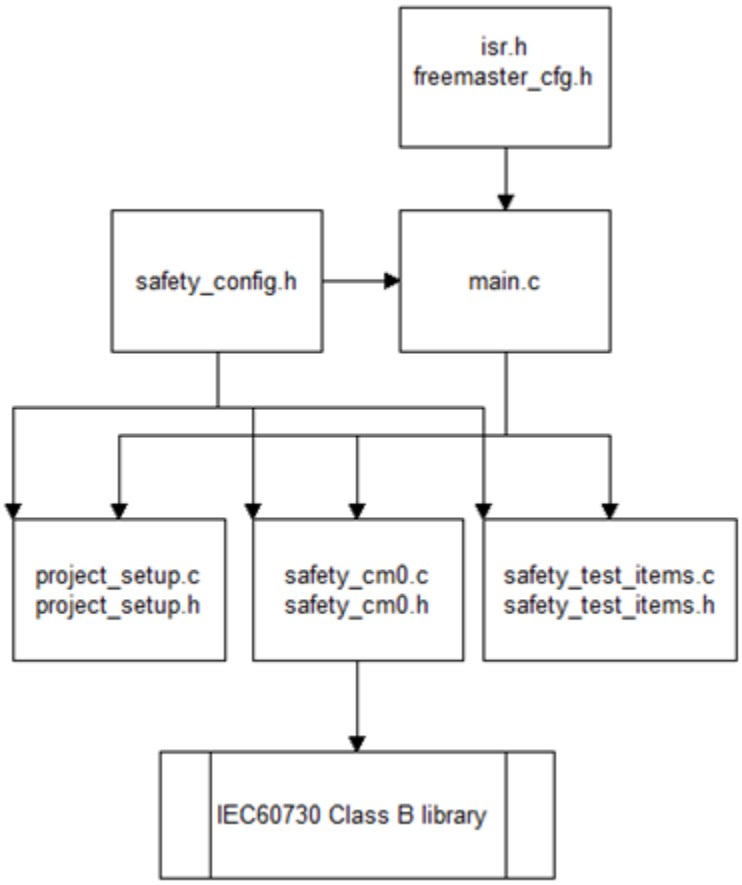


Figure 4. Example of project hiearchy

Figure 4 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_kinetis.c* file and also called from the *main.c* file.

The main example aplication 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_cm0_kinetis.c file contains the handling function for safety routines from the IEC60730B library and also the test-initialization function for safety.

4.1 Example settings - safety_config.h

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

The "switch macros", which allow the user to turn off calling 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 0
#define CLOCK_TEST_ENABLED 1
```

```
#define DIO_TEST_ENABLED 1
#define FLASH_TEST_ENABLED 1
#define PC_TEST_ENABLED 1
#define WATCHDOG_ENABLED 1
```

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

4.2 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 colected 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,
.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 };
```

4.3 Source file - safety_cm0_kinetis.c/.h

The *safety_cm0_kinetis.c* source file and the correspond **.h* file contain a library handling function. Each function contains a detection. If a safety error ocurrs, the *SafetyErrorHandling()* function is called.

Chapter 5
Running example

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

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

5.1 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 - this 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.

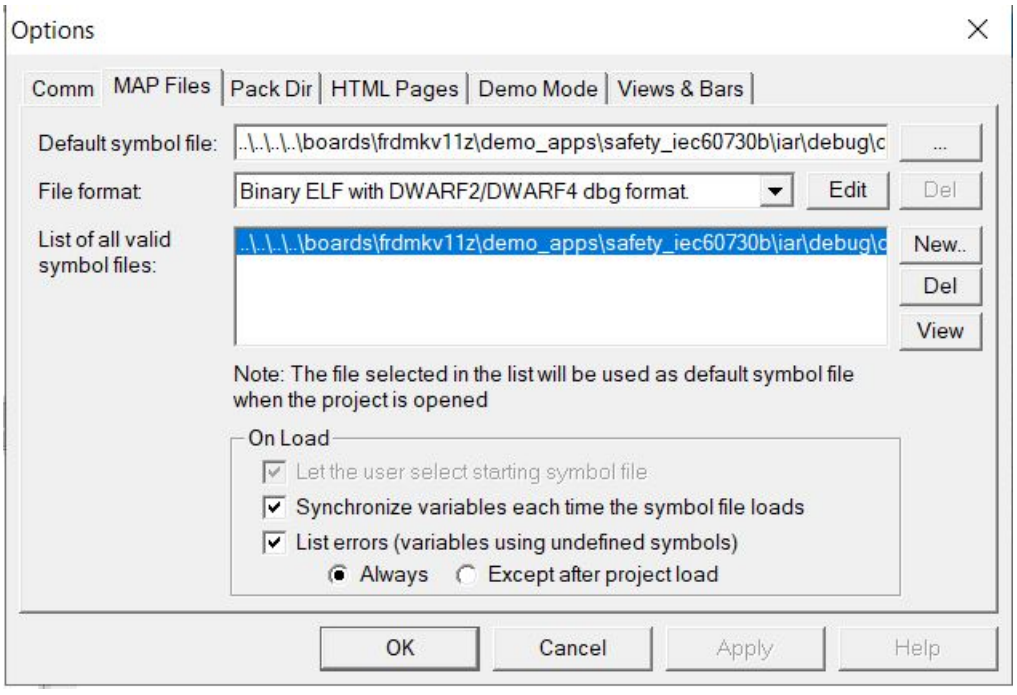


Figure 5. Example of setting MAP files for FRDM-KV11 board

- Open Project ->Options ->Comm and select a correct RS-232 connection and speed. The speed of conncetion is in the *safety_config.h* file Macros ***"SERIAL_BAUD_RATE"***.

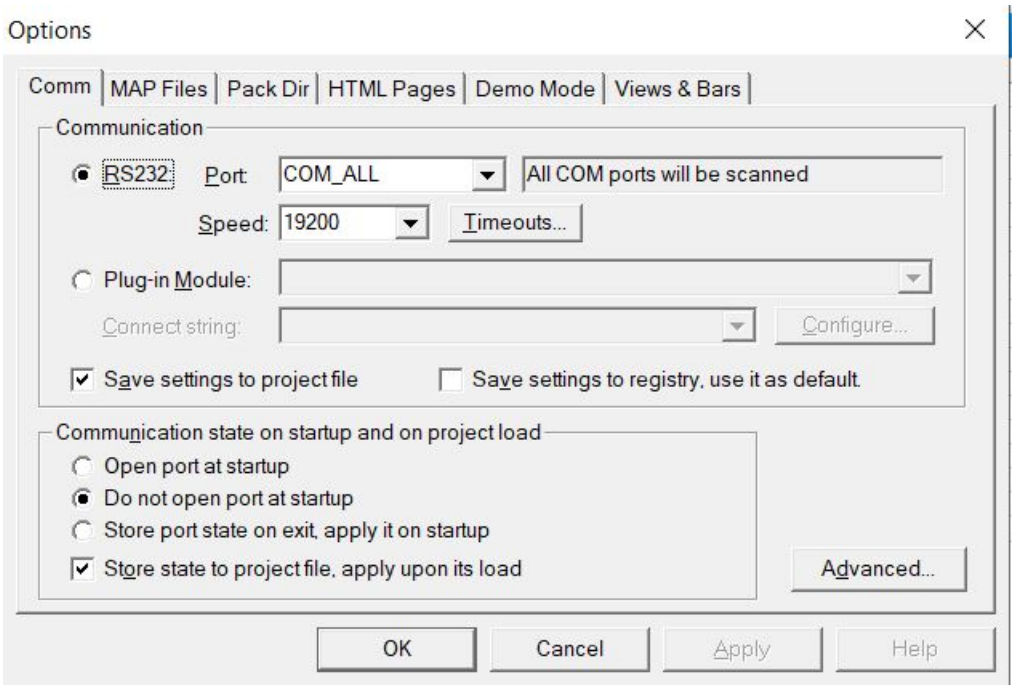


Figure 6. Setting UART speed

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

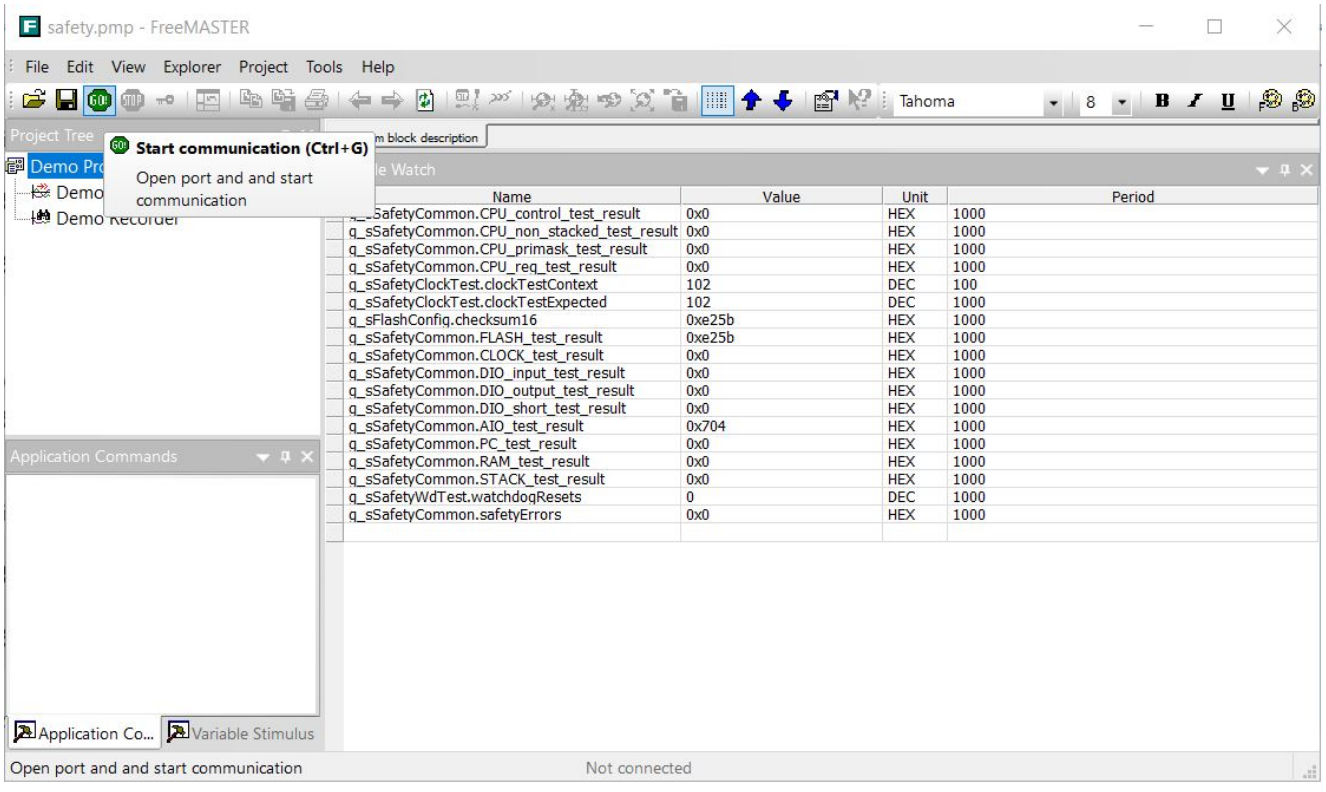


Figure 7. Safety example FMSTR application

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

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 couter test
- Stack test
- Watchdog 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 setup correctly **before** calling the AIO test. In some cases, it is necessary to connect this signal externally by wire to the corresponding pin. The test is perfomed 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 ADC channel 6 is tested first, with limits corresponding to VrefL (GND). Channel 5 is tested next, with limits of VrefH (VCC). Channel 4 is tested next, with limits for the bandgap.

6.2 Clock test

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

NOTE

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

6.3 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 is turned off by default in the *safety_config.h* file.

The test consists of 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 is added to the presented example by default.
- Check if Option -> User -> AfterBuild contains *"./crc_keil.bat"* and the checkbox is selected (it should be filled and checked 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 the **MCUXpresso** IDE:

- Check if the post-build actions are set: *"${ProjDirPath}"/linker/crc_mcux.bat "${BuildArtifactFileBaseName}"*. It should be filled by default.
- The final post-processed image is downloaded to the ROM memory using the GUI Flash Tool (choose the *<board>_safety_iec60730b_CRC.bin* file).
- The "Attach only" option must be set if you want to debug the application.
- Turn on the Flash test in the *safety_config.h* file.

NOTE

When you debug your application with the Flash test turned on, be careful 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 interupted.

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 corrupeted by the WDOG reset.

NOTE

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

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

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

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

Date of release: 06/2020

Document identifier: IEC60730BKCM0SEUG