User Guide: The Library IEC60730 Integration

This document provides instructions for integrating the Library IEC60730 into a project.

This will guide the developer to install the required software. Then guide the configuration of a project and integrate the source code into the project. Finally, we mention a special case.

KEY FEATURES

IEC60730

Simplicity Studio 5

Contents

[Contents 2](#_Toc88141700)

[1. Background 3](#_Toc88141701)

[2. Install the required software 3](#_Toc88141702)

[3. Edit the Linker file 3](#_Toc88141703)

[4. Edit the Post-build steps 5](#_Toc88141704)

[5. Edit the include path and preprocessor 7](#_Toc88141705)

[6. Add the source code to project 8](#_Toc88141706)

[7. Integrate code into project 9](#_Toc88141707)

[8. Special cases 14](#_Toc88141708)

[9. Revision history 15](#_Toc88141709)

# Background

The IEC60730 is a safety standard used in household applications. It defines the test and diagnostic method that ensure the safe operation of devices. We provide the test of following components: CPU registers, variable memory check, invariable memory check, program counter check, clock check, interrupt check.

At the time of this writing, the library IEC60730 supports EFR32xG21 and EFR32xG23 on Simplicity Studio 5 (SS5) with toolchain GNU ARM v10.3.1 and SDK version is 3.2.3.

# Install the required software.

We use the third-party software (SRecord - <http://srecord.sourceforge.net/>) to calculate CRC value. Firstly, you need to install this software. If you’re using Windows OS, you can go to the link of this software (link above), download the installation, and run the installer to install. In case you’re using Ubuntu OS, please follow the instructions below to install.

$ sudo apt update

$ sudo apt install srecord

# Edit the Linker file.

We will base on the original file of a specific application (for example No OS application, Micrium OS application, etc.) and edit this file to new linker file that meets the requirements of the library IEC60730. You SHOULD copy the linker file to another location in your PC and start editing it.

There are some points to do to edit the linker file as described in IEC 60730’s document.

You can refer to the modified Linker file in our demo example. We currently have demo for No OS application, Micrium OS application.

Graphical user interface, application

Description automatically generated

Figure Import the demo examples.

You SHOULD import the files with extension (\*.slsproj) or (\*.sls) into SS5 and select SDK version 3.2.3, Toolchain GNU ARM v10.3.1 or IAR.

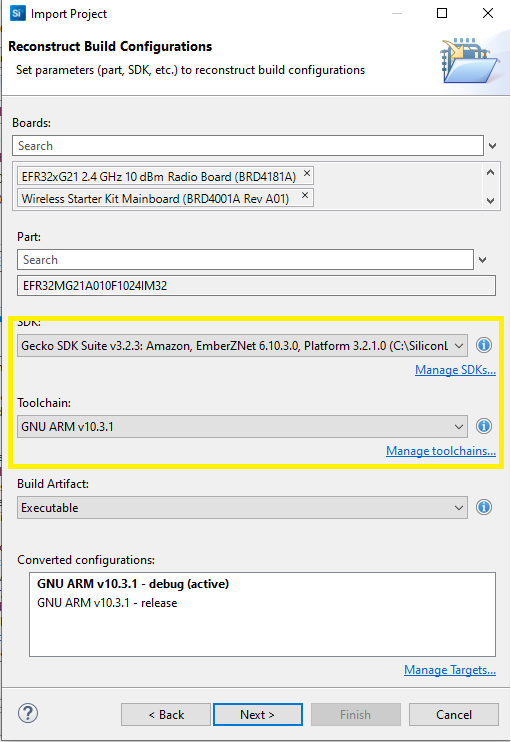


Figure Edit SDK and Toolchain.

After editing the linker file, you need to edit path pointing to this new linker file. Figure 3 show how to edit the Linker Script Path in Simplicity Studio.

You SHOULD follow these steps: C/C++ Build > Settings > Tool Settings > Memory Layout > Linker Script. Remember to check “Use custom linker script”.

Graphical user interface, text

Description automatically generated

Figure Edit the Linker Script Path

# Edit the Post-build steps.

You SHOULD follow these steps: C/C++ Build > Settings > Build Steps > Post-build steps > Command

Edit the Command in the Post-build steps to generate new firmware files with the CRC value attached at the end of the FLASH area. Currently, we support PC running Windows OS and Ubuntu OS with corresponding scripts for each OS.

The Library IEC60730 supports both CRC-16 and CRC-32, so we have 2 script files to calculate the CRC value for each CRC type. The scripts have suffix of \_crc16 and \_crc32 respectively. We will use suffix \_crcNN for both cases.

Detail about parameter of gcc\_crcNN script can refer to IEC60730’s document. For example, in BRD4181A\_EFR32xG21\_IEC60730\_Demo example.

&& sh '${ProjDirPath}/scripts/gcc\_crc16.sh' '${BuildArtifactFileBaseName}' '' 'C:\srecord-1.64-win32' 'GCC'

Graphical user interface, text, application, email

Description automatically generated

Figure Add the Command to Post-build steps.

**Note**:

* By default, we use gcc\_crc16.sh to calculate CRC value. If you add USE\_CRC\_32 definition, we need to use gcc\_crc32.sh.

**Note**: After the build of a project is complete, call the Command in the Post-build steps to create <Project\_Name>\_crc16 or <Project\_Name>\_crc32 files with the extension \*.bin, \*.hex, and \*. s37.

Text

Description automatically generated

Figure Result after Post-build complete

In case using CRC\_DEBUG definition, users can debug the example program. If not, users SHOULD use the binary file with suffix \_crc16 or \_crc32 to flash.

# Edit the include path and preprocessor.

Edit the Include paths (-l) to the location of the source code of the Library IEC60730.

You SHOULD follow these steps: C/C++ Build > Settings > Tool Settings > GNU ARM C Compiler > Includes

Graphical user interface, text

Description automatically generated

Figure Add the include paths.

Edit preprocessor, user need definitions as below.

* IEC\_BOARD\_HEADER = “iec60730\_board.h”
* Add the CRC\_DEBUG definition if we are running in Debug mode. If you don’t use this definition, you NEED to load files with the suffix \_crc16 or \_crc32 to your target.
* Add the USE\_MARCHX definition if we use March X algorithm (DOES NOT perform STEP 3, 4 of March C algorithm) when testing variable memory in run time self-test or Build In Self-Test (BIST).

You can refer to our demo example of these settings.

# Add the source code to project.

We SHOULD add the source code files (\*.c, \*. s) of Library IEC60730 to the project.

Table

Description automatically generated with low confidence

Figure Add the source code to project.

With IAR compiler, user SHOULD link to assembly files iec60730\_vmc\_marchCIAR.s and iec60730\_vmc\_marchXCIAR.s.

With GCC compiler, use assembly files iec60730\_vmc\_marchC.s and iec60730\_vmc\_marchXC.s for VMC module

# Integrate code into project.

The library IEC60730 is divided into run 2 main test phases: Power On Self-Test (POST) and Build In Self-Test (BIST). Figure 8 shows the basic of the library IEC60730 integration into user software solution.

Diagram, schematic

Description automatically generated

Figure Flow chart of the library IEC60730

**Note**: Destructive Memory Test is a special case of testing. This test will be mentioned in section 8.

To integrate the library IEC60730 to fully test steps such as flow charts, You SHOULD follow these steps:

1. Create a new file named iec60730\_oem.h, you SHOULD refer to the content of this file in the demo examples we have created.

2. Create a file \*.h any example iec60730\_board.h and define symbol IEC\_BOARD\_HEADER = “iec60730\_board.h”. Please see 0 for more details. You also SHOULD refer to the content of this file in the demo examples we have created.

3. Requires declaration of the following variables

imcParams\_t imcTest \_\_CLASSB\_RAM;

vmcParams\_t vmcTest \_\_CLASSB\_RAM;

These two variables used for invariable memory check and variable memory check.

SI\_SEGMENT\_VARIABLE(iec60730\_IRQFreqBoundsSize, uint8\_t, SI\_SEG\_IDATA) = 1;

SI\_SEGMENT\_VARIABLE(iec60730\_IRQExecCount[IEC60730\_IRQ\_SIZE], uint8\_t, SI\_SEG\_IDATA) = {0};

SI\_SEGMENT\_VARIABLE(iec60730\_IRQFreqBounds[IEC60730\_IRQ\_SIZE], iec60730\_IRQExecutionBounds\_t, SI\_SEG\_CODE) = {

{ .min = 9, .max = 11 },

{ .min = 9, .max = 11 },

{ .min = 9, .max = 11 },

{ .min = 9, .max = 11 },

{ .min = 9, .max = 11 },

{ .min = 9, .max = 11 },

{ .min = 9, .max = 11 },

{ .min = 9, .max = 11 }

};

These variables are used for IRQ test configuration and definition IEC60730\_IRQ\_SIZE SHOULD be defined in the file iec60730\_oem.h.

4. Configure Watchdog Test: this configuration determines which watchdog unit will be checked. The library does not initialize the watchdog units and user should do the initialization.

* The number of Watchdog units which are tested by lib should be defined by user code by defining the macros "IEC60730\_WDOGINST\_NUMB" and IEC60730\_ENABLE\_WDOGx (x = 0, 1). The number of Watchdog unit depends on target EFR32 series. Example for series 1:

#define IEC60730\_WDOGINST\_NUMB 1

#define IEC60730\_ENABLE\_WDOG0

If these macros are not defined, then the default configuration will be used. For detail implementation, refer iec60730\_watchdog.h file.

* To clear reset cause flags in the RSTCASUES register after watchdog testing completed -> Add the definition of macro "#define IEC60730\_RSTCAUSES\_CLEAR\_ENABLE" to user code. By default, this feature is disabled.
* The static variable iec60730\_WatchdogCount must be located at memory location that is not cleared when system startup (section ".ram\_no\_clear").
* The global variable iec60730\_WatchdogState must be located at memory location that is not cleared when system startup. And it should be located at retention RAM block that is available on EFR32 Series 1 devices (section ".ram\_ret\_no\_clear"). This will avoid the missing contain of the variable when device returns from the power saving mode EM4.
* There's no retention RAM on EFR32 Series 2 devices but they have backup RAM (BURAM) that could be used to save value of the variable. To enable saving iec60730\_WatchdogState to backup RAM on Series 2, add the macro "#define IEC60730\_SAVE\_STAGE\_ENABLE" to user code. By default, it will be disabled. Define macro "IEC60730\_BURAM\_IDX" to select which register of the BURAM will be used. The default value is 0x0.

5. Before calling iec60730\_Post function, we need to do the following steps:

* Configure the clock for the timers. You can refer to these configurations in our demo examples, file iec60730\_init\_device.c.
* Create two timers with 10 milliseconds (ms) and 100ms interrupt periods (parameters 10ms and 100ms are recommend values) to test the clock and the clock switch. You can refer to our demo example, file iec6070\_oem\_timer.c for more details. Note that adjusting the 10ms and 100ms values will require adjusting other parameters such as iec60730\_clockTestTolerance, iec60730\_systemClockToTestClockFre quency.
* Configuration for variables

imcTest.gpcrc = DEFAULT\_GPRC;

Hardware configuration for CRC calculation.

vmcTest.start = RAMTEST\_START;

vmcTest.end = RAMTEST\_END;

Configure the start and end address of the RAM area under check.

iec60730\_clockTestTolerance = 1;

iec60730\_systemClockToTestClockFrequency = 10;

Configuration for clock test.

6. The function iec60730\_Post SHOULD be called at the initialization step.

iec60730\_ImcInit(&imcTest);

iec60730\_VmcInit(&vmcTest);

iec60730\_clockTicksReset();

7. Before calling iec60730\_Bist, we SHOULD initialize the parameters. You can refer to the code below.

8. Before calling the iec60730\_Bist function, we SHOULD set the flag for the iec60730\_programeCounterCheck variable. Some of the following flags are set by the Library IEC60730: IEC60730\_VMC\_COMPLETE, IEC60730\_IMC\_COMPLETE, IEC60730\_CPU\_CLOCKS\_COM PLETE, and IEC60730\_INTERRUPT\_COMPLETE.

Other flags (IEC60730\_GPIO\_COMPLETE, IEC60730\_ANALOG\_COMPLETE, etc.) are up to you to develop additional test functions. The iec60730\_programmeCounterCheck variable SHOULD set the flags corresponding to the unavailable test to ensure that the Program Counter Check is guaranteed.

In the demo examples, you will often see the following code.

iec60730\_programmeCounterCheck |= IEC60730\_GPIO\_COMPLETE \

| IEC60730\_COMMS\_COMPLETE \

| IEC60730\_ANALOG\_COMPLETE \

| IEC60730\_OEM0\_COMPLETE \

| IEC60730\_OEM1\_COMPLETE \

| IEC60730\_OEM2\_COMPLETE \

| IEC60730\_OEM3\_COMPLETE \

| IEC60730\_OEM4\_COMPLETE \

| IEC60730\_OEM5\_COMPLETE \

| IEC60730\_OEM6\_COMPLETE \

| IEC60730\_OEM7\_COMPLETE;

Or

iec60730\_programmeCounterCheck |= IEC60730\_GPIO\_COMPLETE \

| IEC60730\_ANALOG\_COMPLETE \

| IEC60730\_OEM0\_COMPLETE \

| IEC60730\_OEM1\_COMPLETE \

| IEC60730\_OEM2\_COMPLETE \

| IEC60730\_OEM3\_COMPLETE \

| IEC60730\_OEM4\_COMPLETE \

| IEC60730\_OEM5\_COMPLETE \

| IEC60730\_OEM6\_COMPLETE \

| IEC60730\_OEM7\_COMPLETE;

There is a difference between these two examples. The IEC60730\_COMMS\_COMPLETE flag is set in the first example, but not in the second example. The reason is the second example executes the external communication test that set IEC60730\_COMMS\_COMPLETE flag by itself, so this flag is not set here.

9. The function iec60730\_Bist SHOULD be called in a periodical task or in supper loop while(1).

10. Remember to increment the IRQ counter variable every time the interrupt to test occurs. You can refer to the IrqTick function in our demo examples.

void TIMER0\_IRQHandler(void) {

...

IrqTick();

...

}

void IrqTick(void) {

iec60730\_IRQExecCount[0]++;

}

11. Create the function iec60730\_SafeState. The purpose of this function is to handle when an error occurs. An example of handling of this function

void iec60730\_SafeState(iec60730\_TestFailure\_t failure) {

iec60730\_Timers\_Disable();

currFailure = failure; // Use for debug purpose

while (1) {

iec60730\_RestartWatchdog(iec60730\_wdogInstance);

}

}

**Note**: After importing our demo example successfully, you SHOULD do some steps as below to complete importing.

* Create folder name scripts in your project and copy scripts files (gcc\_crcNN.bat and gcc\_crcNN.sh) to this folder.
* Edit linker scripts section as shown in section 3.

# Special cases

<This page left blank for the update in the future>

# Revision history

|  |  |  |
| --- | --- | --- |
| **Revision** | **Date** | **Description** |
| 0.1 | Oct 2021 | Initial Revision |
| 0.2 | Nov 2021 | Section 7: Added description for Watchdog |
| 0.3 | Mar 2023 | Remove Section 8. |
| 0.4 | Apr 2023 | Update document |