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

1. 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.

2. 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.

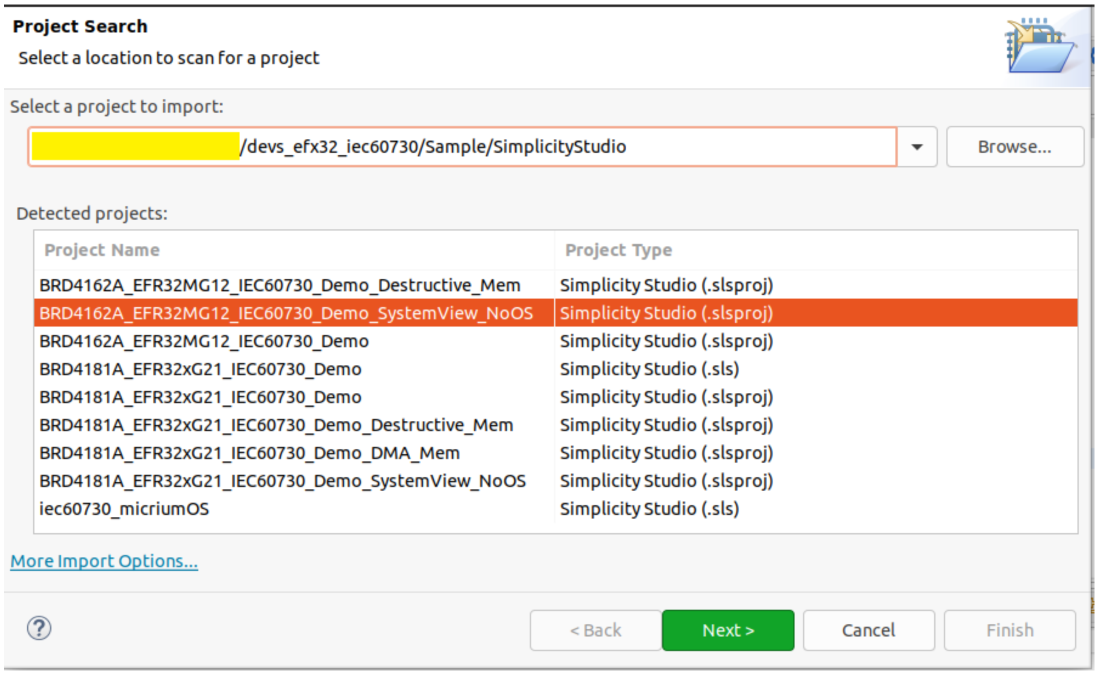
```
$ sudo apt update
$ sudo apt install srecord
```

3. 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.



Project Search

Select a location to scan for a project

Select a project to import:

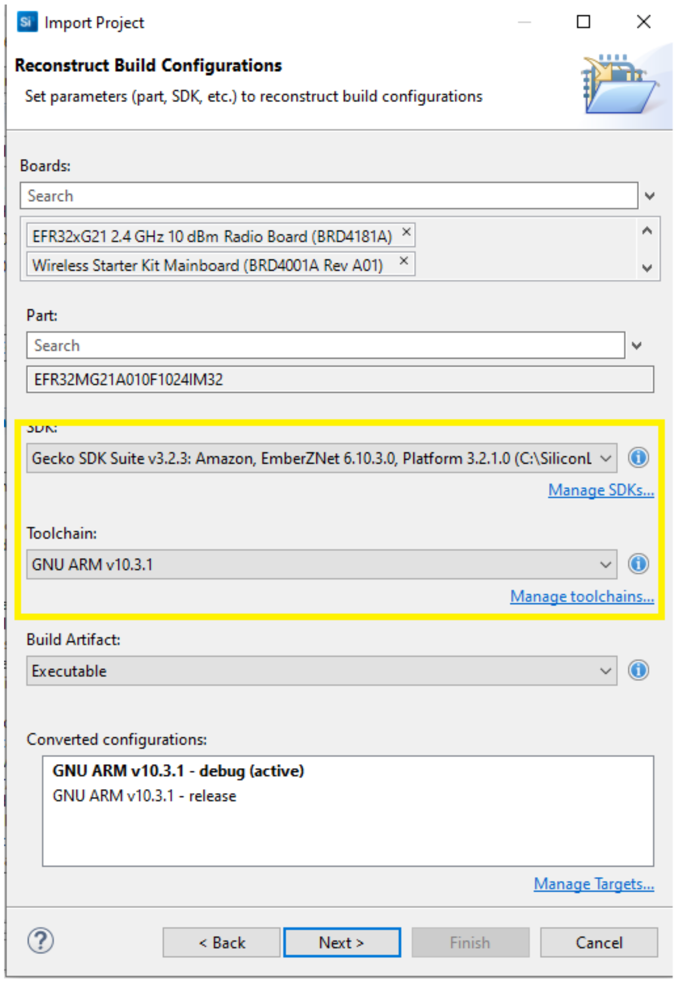
[redacted]/devs_efx32_iec60730/Sample/SimplicityStudio

Detected projects:

Project Name	Project Type
BRD4162A_EFR32MG12_IEC60730_Demo_Destructive_Mem	Simplicity Studio (.slsproj)
BRD4162A_EFR32MG12_IEC60730_Demo_SystemView_NoOS	Simplicity Studio (.slsproj)
BRD4162A_EFR32MG12_IEC60730_Demo	Simplicity Studio (.slsproj)
BRD4181A_EFR32xG21_IEC60730_Demo	Simplicity Studio (.sls)
BRD4181A_EFR32xG21_IEC60730_Demo	Simplicity Studio (.slsproj)
BRD4181A_EFR32xG21_IEC60730_Demo_Destructive_Mem	Simplicity Studio (.slsproj)
BRD4181A_EFR32xG21_IEC60730_Demo_DMA_Mem	Simplicity Studio (.slsproj)
BRD4181A_EFR32xG21_IEC60730_Demo_SystemView_NoOS	Simplicity Studio (.slsproj)
iec60730_micriumOS	Simplicity Studio (.sls)

Figure 1 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.


Import Project

Reconstruct Build Configurations

Set parameters (part, SDK, etc.) to reconstruct build configurations

Boards:

EFR32xG21 2.4 GHz 10 dBm Radio Board (BRD4181A)

Wireless Starter Kit Mainboard (BRD4001A Rev A01)

Part:

EFR32MG21A010F1024IM32

SDK:

Gecko SDK Suite v3.2.3: Amazon, EmberZNet 6.10.3.0, Platform 3.2.1.0 (C:\SiliconL

Toolchain:

GNU ARM v10.3.1

Build Artifact:

Executable

Converted configurations:

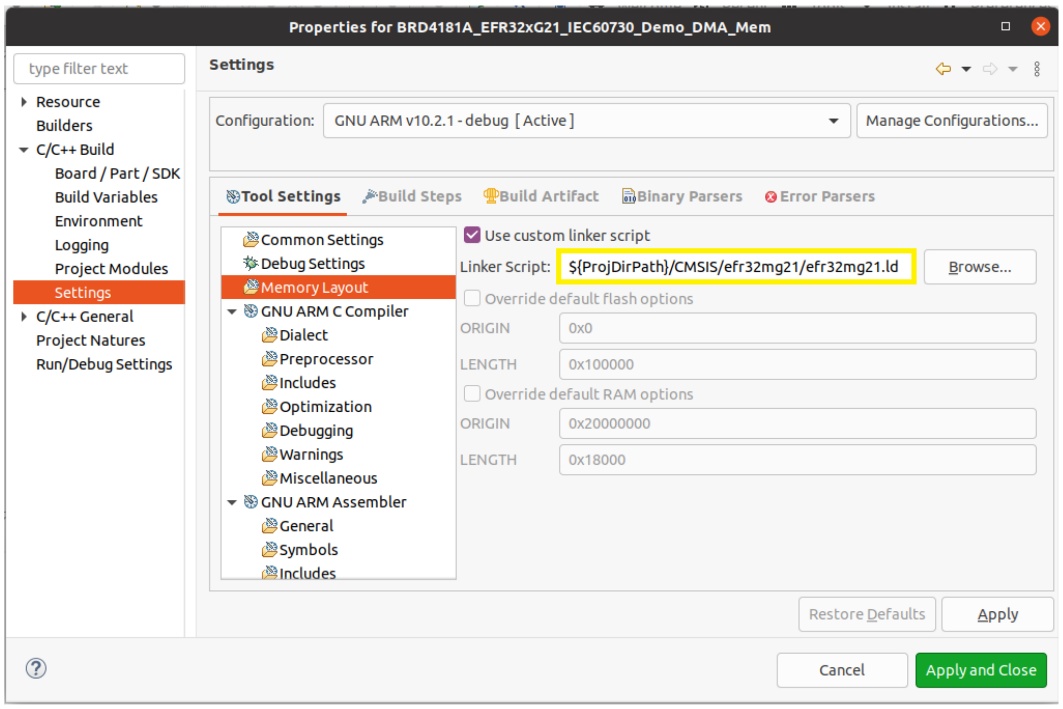
GNU ARM v10.3.1 - debug (active)

GNU ARM v10.3.1 - release

Figure 2 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”.


Properties for BRD4181A_EFR32xG21_IEC60730_Demo_DMA_Mem

Settings

Configuration: GNU ARM v10.2.1 - debug [Active]

Use custom linker script [✓]

Linker Script: ${ProjDirPath}/CMSIS/efr32mg21/efr32mg21.ld

Override default flash options [ ]

ORIGIN 0x0

LENGTH 0x100000

Override default RAM options [ ]

ORIGIN 0x20000000

LENGTH 0x18000

Figure 3 Edit the Linker Script Path

4. Edit the Post-build steps.

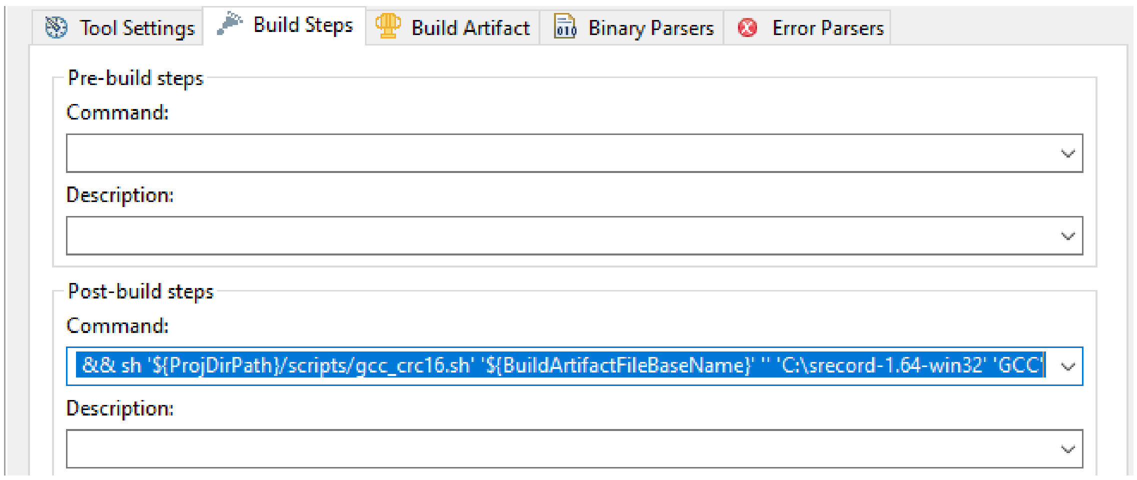
You SHOULD follow these steps: C/C++ Build > Settings > Build Steps > Post-build steps > Command

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'
```



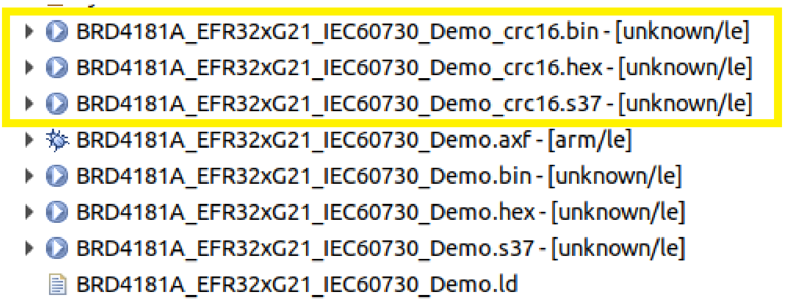
```
&& sh '${ProjDirPath}/scripts/gcc_crc16.sh' '${BuildArtifactFileBaseName}' '' 'C:\srecord-1.64-win32' 'GCC'
```

Figure 4 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.



```
BRD4181A_EFR32xG21_IEC60730_Demo_crc16.bin - [unknown/le]
BRD4181A_EFR32xG21_IEC60730_Demo_crc16.hex - [unknown/le]
BRD4181A_EFR32xG21_IEC60730_Demo_crc16.s37 - [unknown/le]
BRD4181A_EFR32xG21_IEC60730_Demo.axf - [arm/le]
BRD4181A_EFR32xG21_IEC60730_Demo.bin - [unknown/le]
BRD4181A_EFR32xG21_IEC60730_Demo.hex - [unknown/le]
BRD4181A_EFR32xG21_IEC60730_Demo.s37 - [unknown/le]
BRD4181A_EFR32xG21_IEC60730_Demo.ld
```

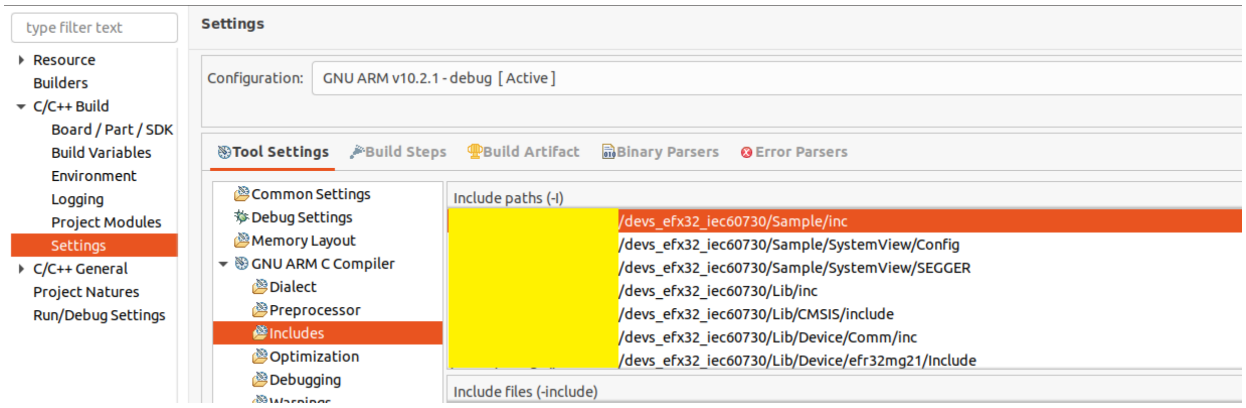
Figure 5 Result after Post-build complete

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.

5. Edit the include path and preprocessor.

Edit the Include paths (-I) 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



Configuration: GNU ARM v10.2.1 - debug [Active]

[redacted]
/devs_efx32_iec60730/Sample/inc
/devs_efx32_iec60730/Sample/SystemView/Config
/devs_efx32_iec60730/Sample/SystemView/SEGGER
/devs_efx32_iec60730/Lib/inc
/devs_efx32_iec60730/Lib/CMSIS/include
/devs_efx32_iec60730/Lib/Device/Comm/inc
/devs_efx32_iec60730/Lib/Device/efr32mg21/Include

Figure 6 Add the include paths.

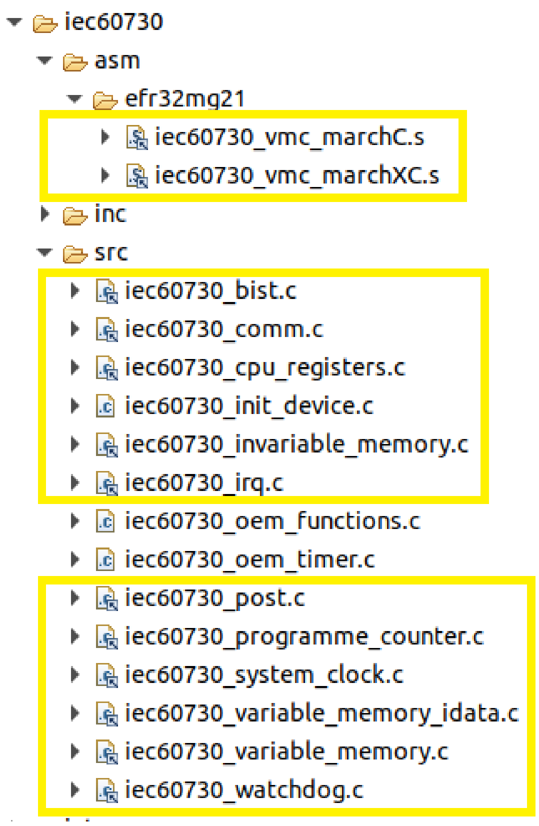
Edit preprocessor, user need definitions as below.

- `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.

6. Add the source code to project.

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



```
iec60730
  asm
    efr32mg21
      iec60730_vmc_marchC.s
      iec60730_vmc_marchXC.s
  inc
  src
    iec60730_bist.c
    iec60730_comm.c
    iec60730_cpu_registers.c
    iec60730_init_device.c
    iec60730_invariable_memory.c
    iec60730_irq.c
    iec60730_oem_functions.c
    iec60730_oem_timer.c
    iec60730_post.c
    iec60730_programme_counter.c
    iec60730_system_clock.c
    iec60730_variable_memory_idata.c
    iec60730_variable_memory.c
    iec60730_watchdog.c
```

Figure 7 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

7. 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.

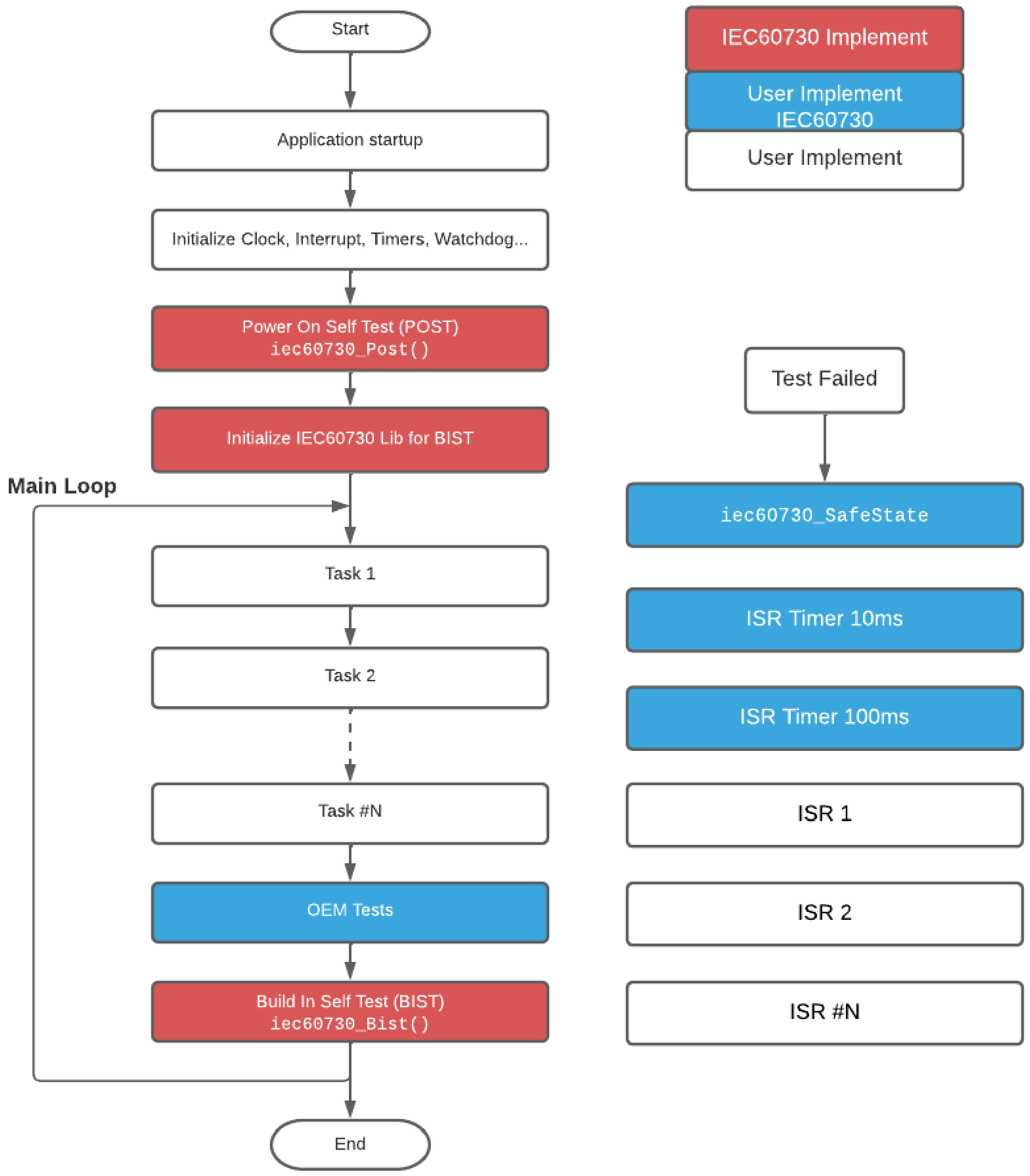


Figure 8 Flow chart of the library IEC60730

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_systemClockToTestClockFrequency`.
- 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.

8. Special cases

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

9. 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