AN205411

FM0+ IEC60730 Class B Self-Test Library

Associated Part Family:	Series Name	Product Number
	S6E1A1	S6E1A11B0A S6E1A12B0A S6E1A11C0A S6E1A12C0A

This application notes describes how to use and implement the library functions provided. It will first show the requirement of IEC60730 Class B, and then explain how it can be implemented. At last an example is given to show how to integrate test functions into a real system.

Contents

1 Introduction....1
1.1 About Document....1
1.2 About IEC60730....1
1.3 About S6E1A Series MCU....2
1.4 About FM0+ IEC60730 STL Demo Project....2
2 IEC60730 Class B Requirement....2
3 IEC60730 Class B STL Overview....4
4 IEC60730 Class B STL API....5
4.1 CPU Register Test....5
4.2 CPU PC Test....8
4.3 Interrupt Test....9
4.4 Clock Test....11
4.5 Invariable Memory Test....20
4.6 Variable Memory Test....27
4.7 IO Test....29
4.8 AD Test....31
5 Example project....33
5.1 User Configuration....33
5.2 Project Structure....33
5.3 Sample Code....35
6 STL API Performance....40
7 Reference Documents....40
8 Appendix....41
8.1 CRC code making method....41
Document History....45

1 Introduction

1.1 About Document

This application notes describes how to use and implement the library functions provided. It will first show the requirement of IEC60730 Class B, and then explain how it can be implemented. At last an example is given to show how to integrate test functions into a real system.

1.2 About IEC60730

The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). International Standard IEC60730-1 has been prepared by IEC technical committee for automatic controls in household use. From 2007 onwards, home appliances have to comply with Standard IEC60730 to make system more safety.

The Annex H of IEC60730 applies to electronic controls and embedded systems implemented by both hardware and software. Therefore the system using a microcontroller is typically the case in modern appliances. Especially, Annex H of IEC60730 explains detailed test and diagnostic methods for microcontrollers.

In Annex H, the software-related Standard items are classified by Class A, B or C.

Class A:	control functions which are not intended to be relied upon for the safety of the equipment, such as humidity controls, lighting controls and timers.

Class B:	software that includes code intended to prevent hazards if a fault, other than a software fault, occurs in the appliance, such as thermal cut-outs and door locks for laundry equipment.
Class C:	software that includes code intended to prevent hazards without the use of other protective devices, such as thermal cut-outs for closed water heater systems.

1.3 About S6E1A Series MCU

S6E1A1 series MCU is 32-bit general purpose MCU of FM0+ family that features the industry's leading-edge ARM Cortex-M0+ CPU and integrates Spansion's highly reliable and high-speed secure embedded flash technology. With a maximum CPU frequency of 40MHz, a high speed flash memory, FM0+ covers the high end of the line-up. The wide operation supply voltage range (2.7~ 5.5V) improves the signal to noise ratio, results in a robust design. All products are based on the same architecture (software compatible), use the same peripherals and are pin compatible in most cases.

It includes a host of robust peripheral features, including motor control timers (MFT), base timer (can be configured to PWM, PPG, Reload, PWC timer), ADCs, on-chip memory (up to 88K Flash, up to 6K SRAM) and a wide range of communication interfaces (I2C, UART, CSIO, LIN, CAN).

The size of on-chip memory can be configured according to different part number and the package is available in LQFP and QFN, shown in table 1-1.

Table 1. FM0+ Product List

Product	Flash	SRAM	Package
S6E1A11B0A	FLASH: 56KB	6kB	LQFP-32 QFN-32
S6E1A12B0A	FLASH: 88KB	6kB	
S6E1A11C0A	FLASH: 56KB	6kB	LQFP-48 QFN-48 LQFP-52
S6E1A12C0A	FLASH: 88KB	6kB	

1.4 About FM0+ IEC60730 STL Demo Project

This is a sample project to demonstrate how to use FM0+ IEC60730 Self-Test Library. It is developed in IAR EWARM Workbench V6.50 and Keil µVision V5.10 IDE, and evaluated on Spansion's SK-FM0P-48LQFP-9AF160K V1.0.0 start-kit board.

Note: If the different version of IAR EWARM Workbench V6.50 and Keil µVision V5.10 are used to open this example project, MCU type information in project setting may lose, please check it.

Note: If the former version of IAR EWARM Workbench V6.50 is used to open this example project, MCU type, pre-included files (in preprocess table of C/C++ compiler), icf file (in link table of debug option), flash loader file (down table of debugger option) may lose, please check these settings.

Note: If the former version of Keil µVision V5.10 is used to open this example project, MCU type, pre-included files (in C/C++ table of project option), debug setting (in debug table of project setting) may lose, please check these settings.

2 IEC60730 Class B Requirement

The specification defined in IEC60730 requires controls with functions classified as software class B or C shall use measures to avoid and control software-related faults/errors in safety-related data and safety-related segments of the software. This means the software must use test method to detect faults internal and external of the microcontroller.

FM0+ IEC60730 self-test library (STL) focuses on software Class B requirement for S6E1A1 series MCU, which covers most IEC60730 requirements listed in the standard. For Class B controllers, below table lists elements that must be tested, method to be adapted and definitions to be implemented as summary of Annex H table H.11.12.7.

Table 2. FM0+ IEC60730 STL Test Items

Component	Fault/Error	Method used in STL	Definitions	In STL
1. CPU 1.1 Register	Stuck at	static memory test	H. 2.19.6	YES
1.2 Program counter	Stuck at	logical monitoring of the program sequence	H.2.18.10.2	YES
2. Interrupt	No interrupt or too frequency interrupt	Time-slot monitoring	H.2.18.10.4	YES
3. Clock	Wrong frequency	Frequency monitor	H.2.18.10.1	YES
4. Memory 4.1. Invariable memory	All single bit faults	Redundancy check	H.2.19.3.2	YES
4.2. Variable memory	DC fault	static memory test	H.2.19.6	YES
4.3. Address[1]	Stuck at	Redundancy check	-	YES
5. Internal data path[2]	Stuck at	-	-	NO
6.External communication 6.1 Data[3]	Hamming distance 3	-	-	NO
6.3 Timing	Wrong point in time	-	-	NO
7. Input/output periphery 7.1 Digital I/O	Function error	Output verification	H.2.18.12	YES
7.2 A/D	Function error	Input comparison	H.2.18.8	YES

Note: The address test can be partly covered by test method of invariable and variable memory test. E.g. the error that two cells are mapped to a same address can be identified when doing invariable memory test with CRC test.

Note: Internal data path is only tested when using external memory.

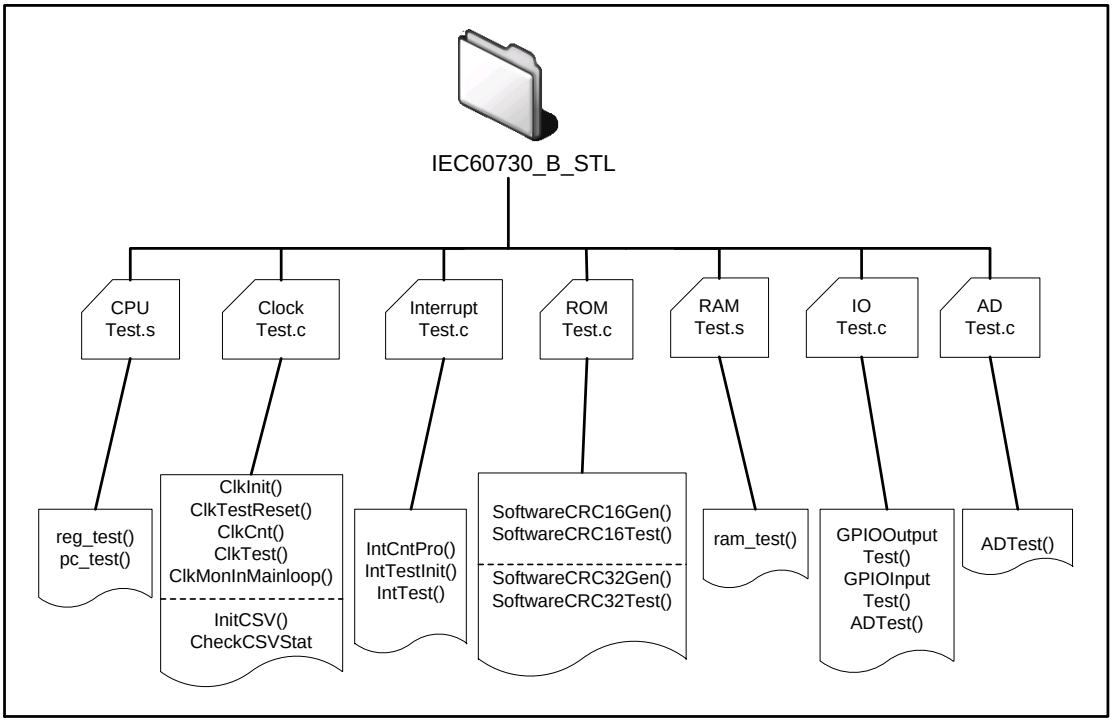
Note: The external communication test is not involved in this STL. But external communication data can be tested with similar method of invariable memory test.

3 IEC60730 Class B STL Overview

As shown in following figure, the STL block diagram includes CPU, Interrupt, Clock, Memory, and Input/output periphery module. It shows file structure and software APIs in the STL. The STL is coded by mixed C and assembly language.

FM0P IEC60730 STL should be compatible with ARM, IAR compiler. So STL supplies two kinds of CPU test.s and ram test.s files according to different compilers.

Figure 1. FM0+ IEC60730 Class B STL Block Diagram



The STL consists of several independent function modules, which have to be executed once or cyclically as required by the application.

The test function implemented once is called Power-On Self-Test (POST), which should be implemented in system initialization, this test is always complete but destructive(need Initialize), which means it covers full test area but the data is not restored after executing test. PC, register, ROM/RAM, IO, AD test are all POST.

The test function implemented cyclically is called Build-In Self-Test (BIST), which should be implemented in main loop or timer interrupt service routine in a certain interval, this test will not change test data and act as a monitor when program is running. Interrupt and clock are BIST.

Note: The library should be used as explained, if any part is changed, a new validation is needed for these parts.

Note: This library is usable, as-is, for all Spansion Cortex-M0+ MCU, including those not especially mentioned in this application notes.

Note: The prefix of file and function name is omitted for easy description.

Note: The STL provides two types of assembly files for CPU and RAM test for IAR and KEIL IDE.

Note: User has alternative test method in clock and Flash test.

4 IEC60730 Class B STL API

4.1 CPU Register Test

ARM Cortex-M0+ has 19 core registers, which can be read and written. These registers below need to be tested.

Register Name	Bits tested
R0-R12	[31:0]
R13 (SP_main, SP_process)[1]	[31:4]
R14 (LR)	[31:0]
APSR[2]	[31:28]
PRIMASK[3]	0

Table 4-1: Cotex-M0+ Register List

Note: ARM Cortex-M0+ kernel has two stack pointers: main stack pointer (MSP) and process stack pointer (PSP). Handler mode uses MSP and process mode uses MSP or PSP. R13 indicates current SP.

Note: Only high 4 bits of APSR is valid.

Note: Only bit 0 of PRIMASK is valid.

4.1.1 Test Description

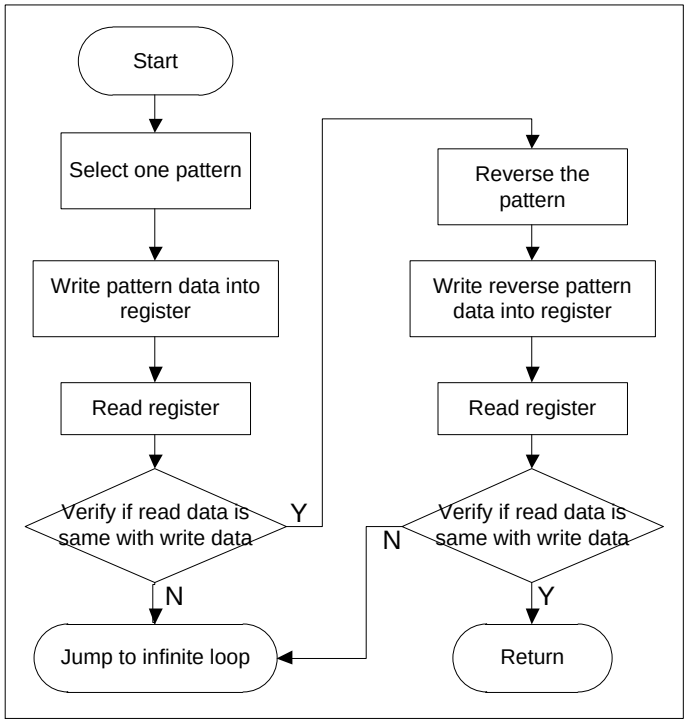
As shown at table H.11.12.7, registers must be checked for “stuck-at error”, a simple checker board method is used to implement register test, which is an effective method to detect stuck-at error.

This test should be called at startup file when system resets in Privileged mode, as kernel registers needs to be accessed. This test does not disable interrupts during the register test. It is the responsibility of the application to disable interrupts when this function is called to ensure that the register test is not interrupted.

Assembly is used to implement register test due to access to registers directly. And as it is highly critical, it is designed that once register test error is detected, program will run into an infinite loop.

The flow chart to test 1 register is shown as following figure.

Figure 2. Test 1 Register



4.1.2 API Definition

Name	iec60730_reg_test
Parameter	None
Return	None

Description:

This function tests all registers including R0-R12 (low: R0-R7, high: R8-R12) special registers (SP, LR, APSR, PRIMASK) with checker board method. This function should be called at reset handler.

4.2 CPU PC Test

4.2.1 Test Description

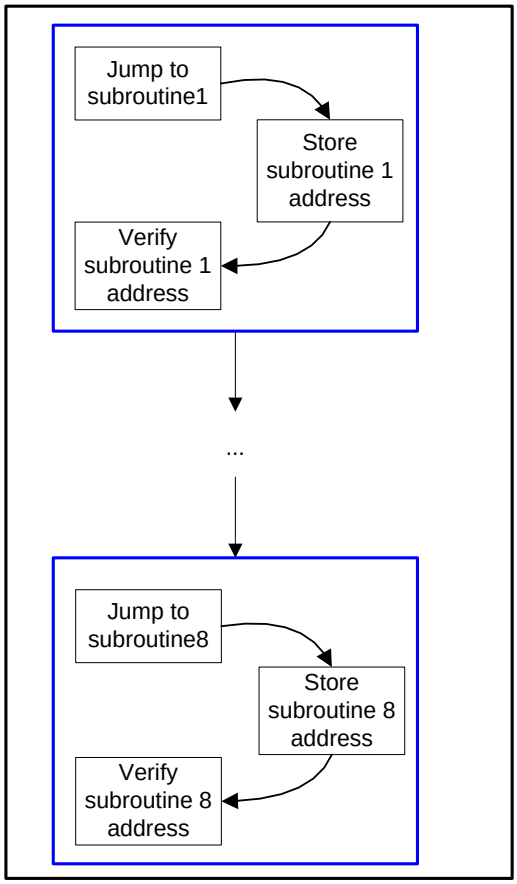
As shown at table H.11.12.7, PC must be checked for “stuck-at error”. PC test makes use of 8 subroutines and validates if PC value gotten from each subroutines is same with pre-define value.

This test should be called at startup file when system resets in Privileged mode. This test does not disable interrupts during the register test. It is the responsibility of the application to disable interrupts when this function is called to ensure that the register test is not interrupted.

Assembly is used to implement PC test due to access to PC register directly. As it is highly critical, it is designed that once PC test error is detected, program will run into an infinite loop.

The PC test flow is shown as following figure.

Figure 3. PC Test Flow Chart



4.2.2 API Definition

Name	iec60730_pc_test
Parameter	None
Return	None

Description:

This function jumps to subroutines at different areas and gets the subroutine address, then verifies whether address gotten is correct. It should be called at reset handler.

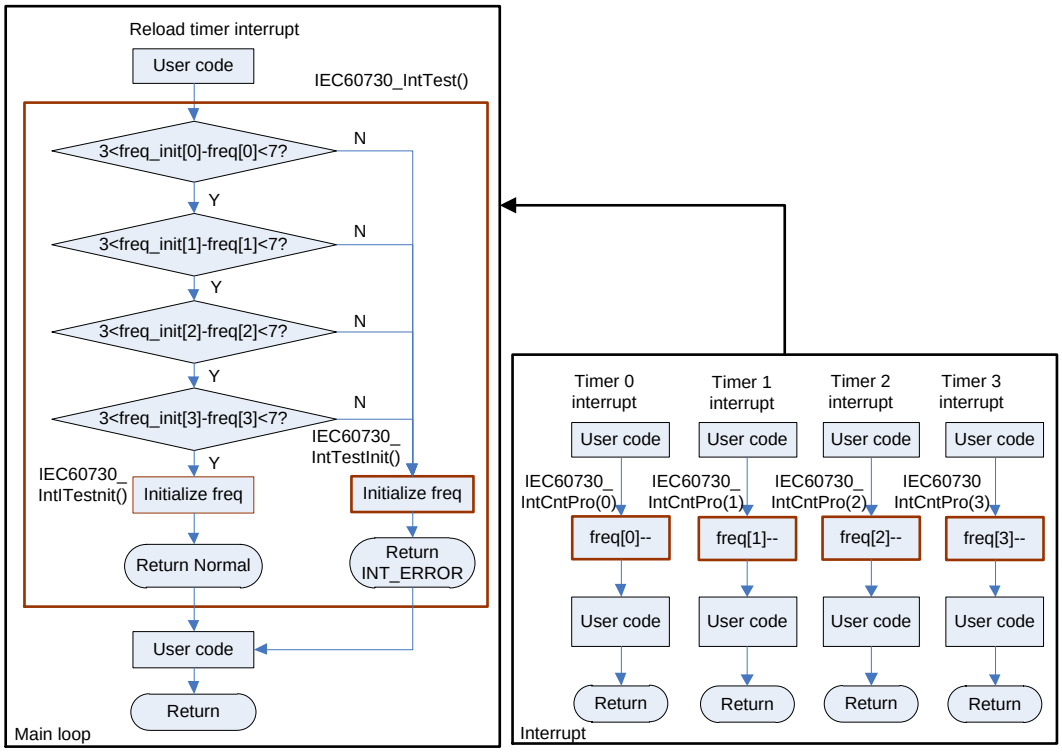
4.3 Interrupt Test

4.3.1 Test Description

To meet Class B requirement, interrupt must be checked for “incorrect frequency”. This test is a task which is highly system dependent and therefore the STL can only contribute the wrap up handle, which checks that a number of specific interrupts occurred at least and at most a predefined number of times. It is assumed that **IEC60730_IntTest** (interrupt test function) is called in specified intervals, e.g. triggered by a timer or line frequency interrupt. Each specific interrupt handler which is to be supervised, must decrement a dedicated global variable (**Freq**) by calling **IEC60730_IntCnt**, **IEC60730_IntTest** compares that variable to predefined upper and lower bounds, sets it to its preset value and returns an error, if the limits are exceeded.

For example, measure if timer 0-3 interrupts happen 5 times in 10 second, assume 10 second timing can be gotten by a reload timer and set the range of interrupt frequency of timer 0-3 at [3, 7].

Figure 4. Interrupt Test Block Diagram



The interrupt test is independent from user application. User just needs to add the interrupt test API into his interrupt which he wants to test.

4.3.2 API Definition

Name	IEC60730_IntTestInit
Parameter	pFreq: pointer to frequency counters pFreqLower: pointer to lower frequencies pFreqUpper: pointer to upper frequencies pFreqInitial: pointer to frequency initial value ArraySize: pointer to interrupt num
Return	None

Description:

This function Initializes str_int_test_par_t structure for interrupt test, which includes pre-defined frequency ranges and frequency initial values. It should be called at system initialization, before interrupt test starts.

Name	IEC60730_IntCntPro
Parameter	IntNum: interrupt number
Return	None

Description:

This function decreases frequency counter of the interrupt specified by the interrupt number, and should be called in the interrupt which to be supervised.

Name	IEC60730_IntTest
Parameter	None
Return	0: IEC60730_TEST_NORMAL 1: IEC60730_TEST_FUNC_ERROR

Description:

This is interrupt test main function, which verifies whether the interrupts are handled in time. It should be called at a timer interrupt or main loop in a certain interval.

4.4 Clock Test

4.4.1 Test Description

To meet Class B requirement, CPU clock must be checked for “wrong frequency”. This requires a second independent clock as a standard clock for clock test. This library provides two ways to implement clock test. First, FM0+ MCU has integrated a watch counter which can be sourced by an external sub clock (32.768kHz oscillator). The sub clock can be treated as the standard clock. For the second, FM0+ MCU has integrated a **Clock Supervisor** (in following called **CSV**), which includes the functions: Clock failure detection and Anomalous frequency detection. The CSV can also be used for clock test.

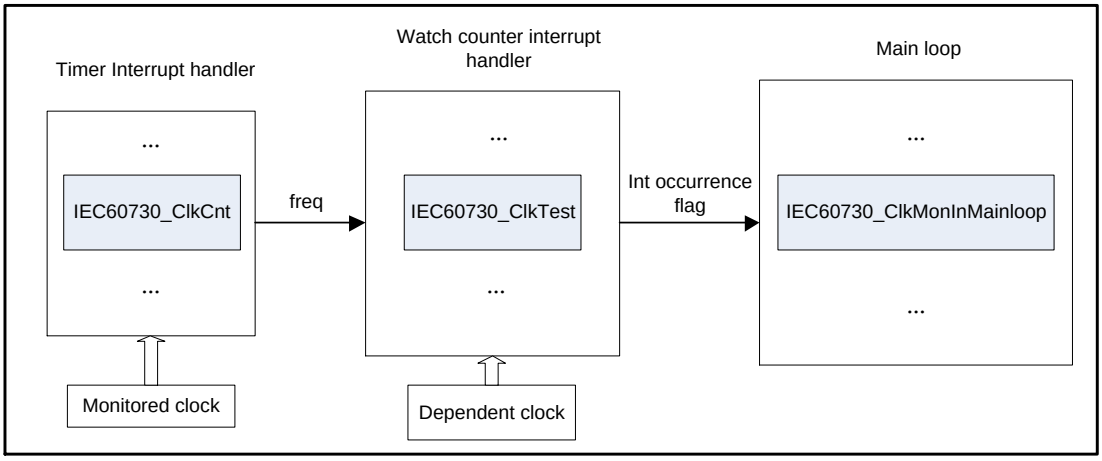
User should enable the definition “IEC60730_CLKTEST_USE_CSV” in IEC60730_user.h file if he wants to use CSV to perform clock test.

Use watch counter to do clock test

This test takes watch counter as standard clock, and tests whether the frequency of CPU clock is within acceptable bound by verifying a time tick which is counted in a timer interrupt. The source clock of timer interrupt should be same with CPU clock. The case that CPU clock is sourced by sub clock can not be tested, as 32.768kHz oscillator is assumed accurate.

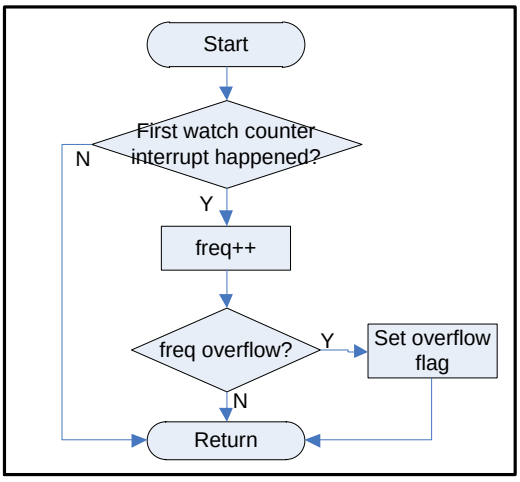
These test functions are implemented: **IEC60730_ClkCnt**, **IEC60730_ClkTest**, and **IEC60730_ClkMonMainloop**, shown as following figure. The timer interrupt occurrence frequency is monitored by watch counter and the watch counter interrupt occurrence is checked in main loop.

Figure 5. Clock Test Block Diagram



The API **IEC60730_ClkCnt** is used to count a global variable “freq”, which is called in a timer interrupt handler, the source clock of timer should be same with CPU clock. The flowchart of **IEC60730_ClkCnt** is shown as following figure.

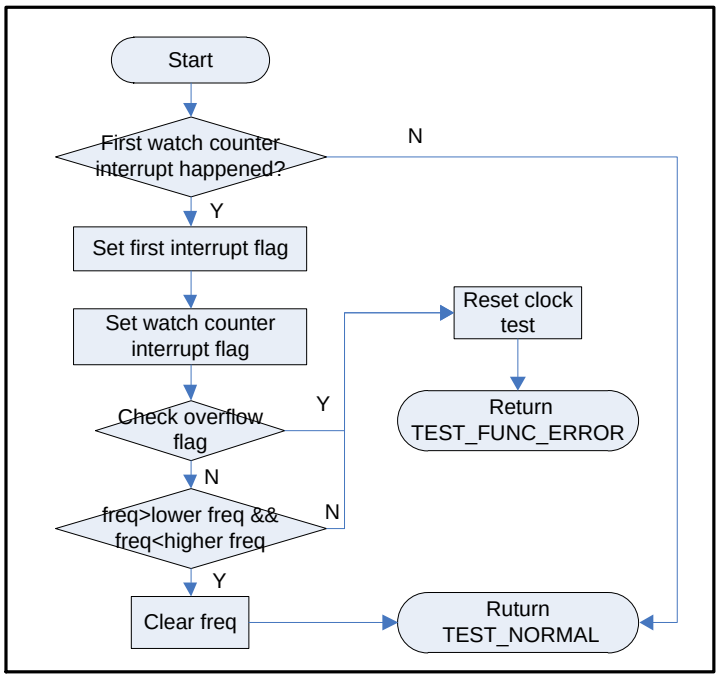
Figure 6. Clock Counter Flowchart



Note: The global variable “freq” starts to count until first watch counter interrupt occurred, because it is a limitation of watch counter in FM0+ MCU that the first count cycle is 2 times of normal cycle. So the first watch counter interrupt should be ignored.

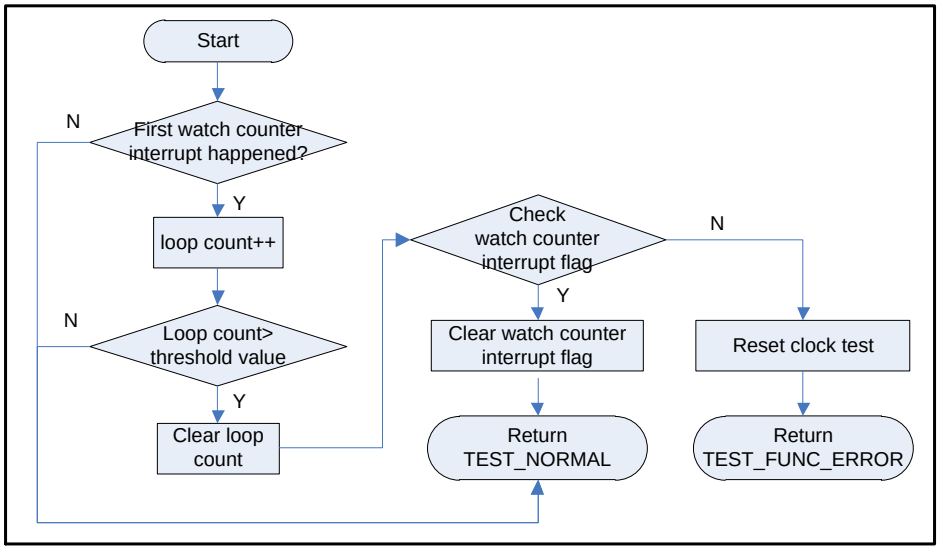
API **IEC60730_ClkTest** is to check whether “freq” is in pre-defined range, which is called in watch counter interrupt handler.

Figure 7. Clock Test Flowchart



API **IEC60730_ClkMonInMainloop** guarantees the occurrence of watch counter interrupt in a certain period, this period depends on the threshold value set by user according to a real application. The flowchart of **IEC60730_ClkMonMainInloop** is shown as following figure.

Figure 8. Clock Main Loop Monitor Flowchart

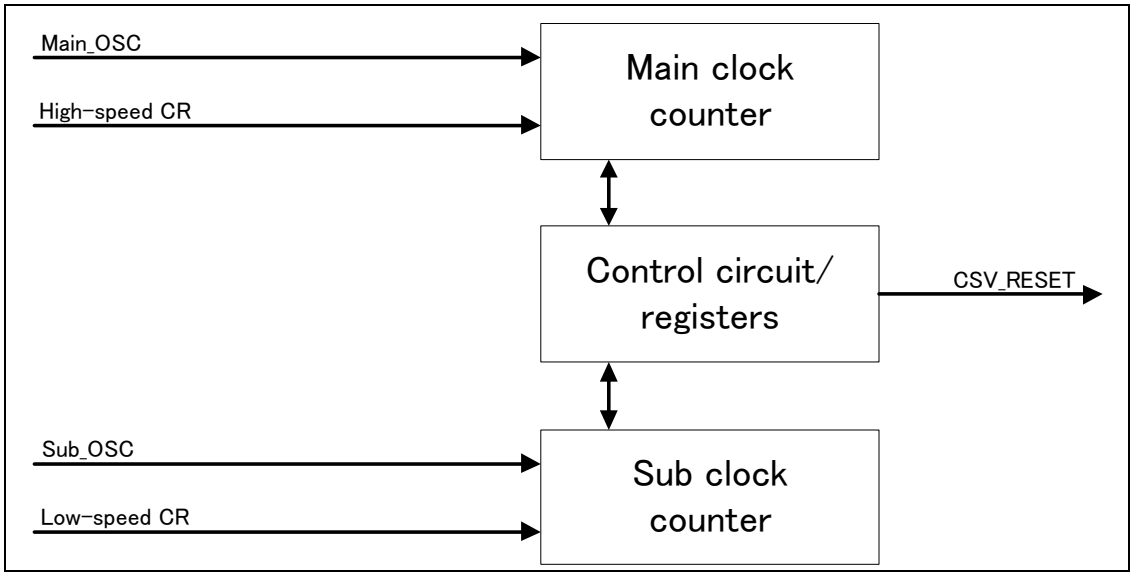


Use CSV to do clock test

The CSV has two types of functions: **Clock failure detection** (**CSV**: Clock failure detection by clock Super Visor) and **Anomalous frequency detection** (**FCS**: anomalous Frequency detection by Clock Super visor).

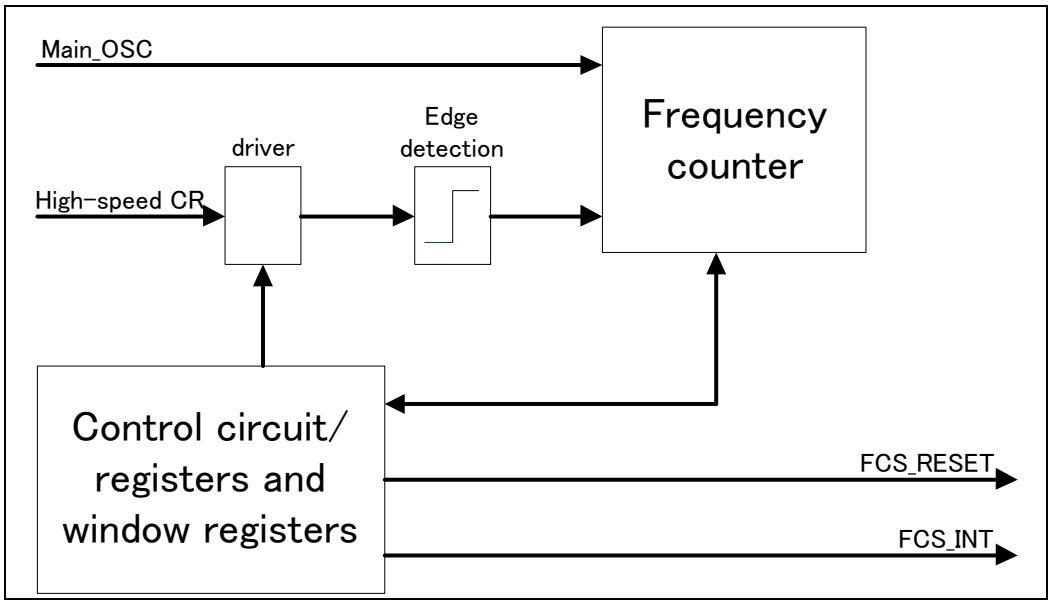
The clock failure detection monitors the main and sub clocks. If a rising edge of the monitored clock is not detected within the specified period, this function determines that the oscillator has failed, and outputs a system reset request. The main clock is monitored with the high-speed CR clock, and the sub clock is monitored with the low-speed CR clock. When a rising edge is not detected within 32 clocks of high-speed CR for the main clock, or within 32 clocks of low-speed CR for the sub clock, this function determines that the oscillator has failed. Figure 4-8 shows the block diagram of the clock failure detection.

Figure 9. Clock Failure Detection Block Diagram



The Anomalous frequency detection monitors the main clock. Within the specified period between an edge and the next edge of the divided clock of high-speed CR, this function counts up the internal counter using the main clock. If the count value reaches out of the set window range, the function determines that the main clock frequency is anomalous, and outputs an interrupt request or a system reset request to the CPU. Figure 4-9 shows the block diagram of the anomalous frequency detection.

Figure 10. Anomalous Frequency Detection Block Diagram

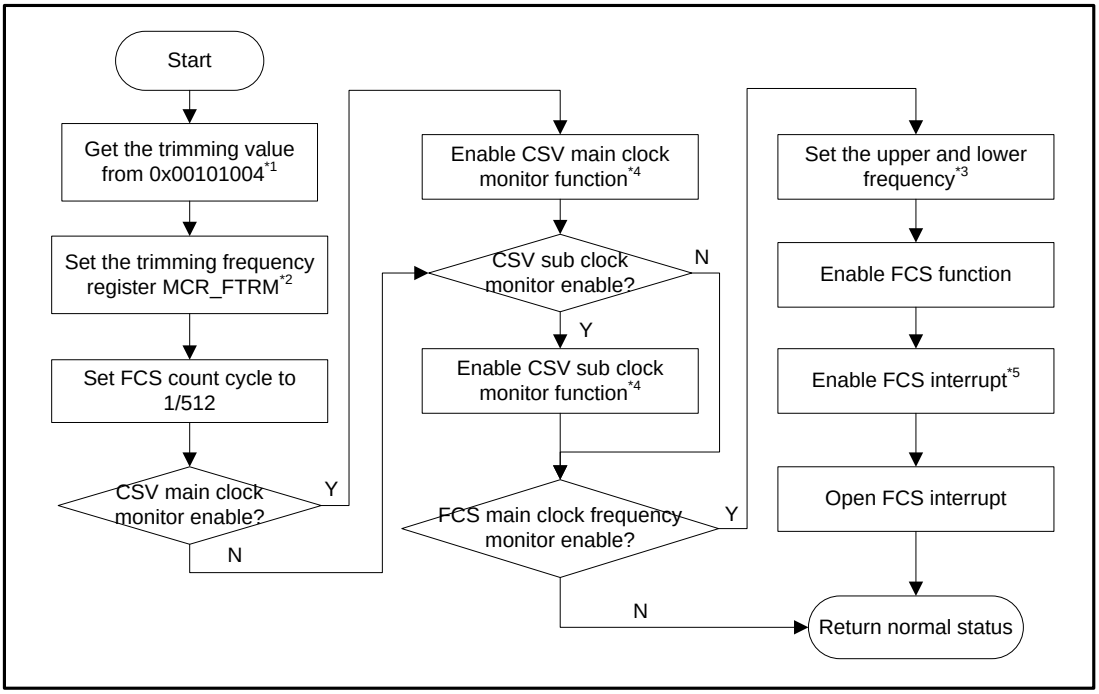


Two test functions are implemented:

IEC60730_InitCSV and **IEC60730_CheckCSVStat.**

The API **IEC60730_InitCSV** provides a selection for user to disable/enable Clock failure detection and Anomalous frequency detection functions. It should be called before system clock initialization. Figure 4-10 shows the flow chart of it.

Figure 11. IEC60730_InitCSV Flow Chart



Note: The default high-speed CR trimming value is stored in the address 0x00100004 when leaving factory.

Note: If the CR trimming value in the address 0x00100004 is destroyed, a typical value must be written into the trimming register MCR_FTRM.

Note: When setting the expected accuracy of main clock, high-speed CR frequency should also be considered. Consider the high-speed CR oscillator precision is 4M±3% (As found in data sheet, for S6E1A1, the high-speed CR oscillator precision is 4M±2% in 25°C, so 4M±3% is used for a little margin). The base upper and lower count can be calculated by following formula.

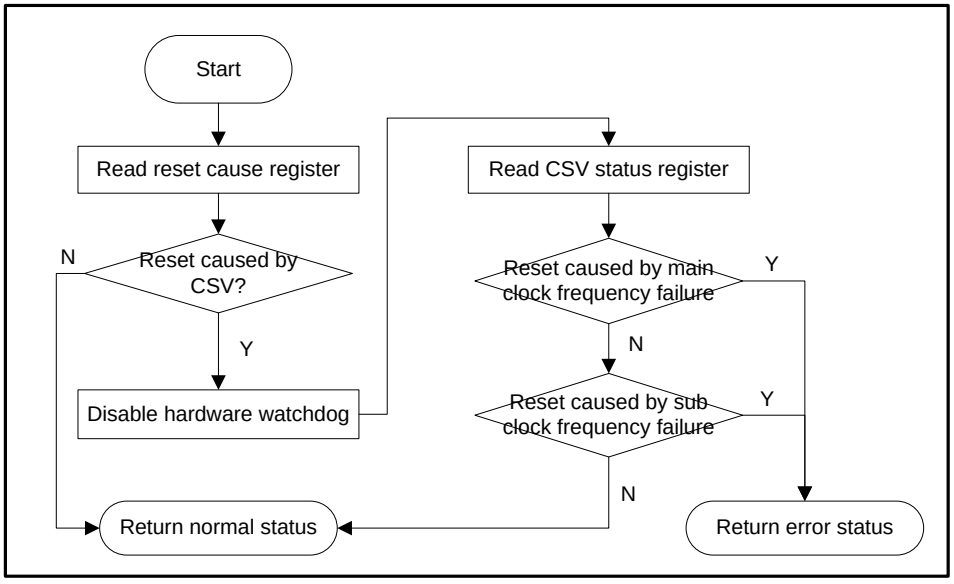
1. base lower count (operating in +3.0%) = 1/[(freq/512*) × (1 + 0.03)] × freq = 512/1.03 = 497
2. base upper count (operating in -3.0%) = 1/[(freq/512*) × (1 - 0.03)] × freq = 512/0.97 = 528
3. If 5% accuracy is set,
4. lower count = 497 × 0.95 = 472
5. upper count = 528 × 1.05 = 554

Note: After enable CSV function, a reset will occurred when a rising edge is not detected within 32 clocks of high-speed CR for the main clock, or within 32 clocks of low-speed CR for the sub clock.

Note: After enable FCS function and FCS interrupt, a FCS interrupt will occur if main clock frequency is detected not in the setting range, but FCS reset is set not to output.

The API **IEC60730_CheckCSVStat** is used to check if Clock failure detection or Anomalous frequency detection happens. This API should be called before **IEC60730_InitCSV.** Figure 4-11 shows the flow chart of it.

Figure 12. IEC60730_CheckCSVStat Flow Chart



4.4.2 API Definition

Use watch counter to do clock test

Name	IEC60730_ClkCnt
Parameter	None
Return	None

Description:

This API is used to count clock frequency, which should be called in the timer interrupt.

Name	IEC60730_ClkTest
Parameter	None
Return	0: IEC60730_TEST_NORMAL 1: IEC60730_TEST_FUNC_ERROR

Description:

This API tests if the frequency of CPU clock is within acceptable bound by verifying a time tick which is counted in a timer interrupt. It should be called in the watch counter interrupt, which is sourced by an independent 32.768kHz clock (sub-clock of FM4 MCU).

Name	IEC60730_ClkMonInMainloop
Parameter	None
Return	0: IEC60730_TEST_NORMAL 1: IEC60730_TEST_FUNC_ERROR

Description:

This API is used to monitor watch counter interrupt occurrence, it should be called in main loop.

Name	IEC60730_ClkTestReset
Parameter	None
Return	None

Description:

This API resets interrupt test variables.

Name	IEC60730_ClkInit
Parameter	FreqLower: indicate timer interrupt minimum occur frequency FreqUpper: indicate timer interrupt maximum occur frequency ClkTestThreshold: indicate threshold value
Return	None

Description:

This API should be called at system initialization before clock test starts.

The parameter FreqLower and FreqUpper should be set according to actual example. For example, if user uses 1s interval for watch counter to monitor a 50ms timer interrupt. The value FreqLower =18, FreqUpper =22 can be set as bound of timer clock frequency, the standard of which is 20.

It is important to estimate threshold value, which should be at least 1s/mainloop execution time.

Use CSV to do clock test

Name	IEC60730_CheckCSVStat
Parameter	pRegRSTStat: get the data from reset cause register
Return	None

Description:

This API is used to check if Clock failure detection or anomalous frequency detection happens. The parameter “pRegRSTStat” store the address of data read from reset cause register. This API only handles the reset caused by CSV, otherwise it will return normal status. It should be called before IEC60730_InitCSV.

Name	IEC60730_InitCSV
Parameter	CSV_MCLKMonEn: 0: disable CSV main clock monitor 1:enable CSV main clock monitor CSV_SCLKMonEn: 0: disable CSV sub clock monitor 1:enable CSV sub clock monitor FCS_MONInfo: a fcs_mon_info_t structure typedef struct fcs_mon_info { stl_uint8_t FCSMonEn; /* 0: disable FCS function, 1: enable FCS function */ stl_uint8_t MCLKFreqAccuracy; /* input the excepted accuracy of main clock, 5->5%*/ } fcs_mon_info_t;
Return	0: IEC60730_TEST_NORMAL 2: IEC60730_TEST_PARA_ERROR

Description:

This API can enable/disable CSV main/sub clock function, and input the expected accuracy of main clock frequency. It should be called before system clock initialization.

4.5 Invariable Memory Test

Invariable memory in FM0+ MCU means On-Chip Flash. The Flash size can be configured according to different product shown as table 1-1.

The CRC (Cyclic Redundancy Check) module is an error detection system. The CRC code is a remainder after an input data string is divided by the pre-defined generator polynomial, assuming the input data string is a high order polynomial. Ordinarily, a data string is suffixed by a CRC code when being sent, and the received data is divided by a generator polynomial as described above. If the received data is dividable, it is judged to be correct. On-Chip Flash Test confirms with CRC that data and program is correct.

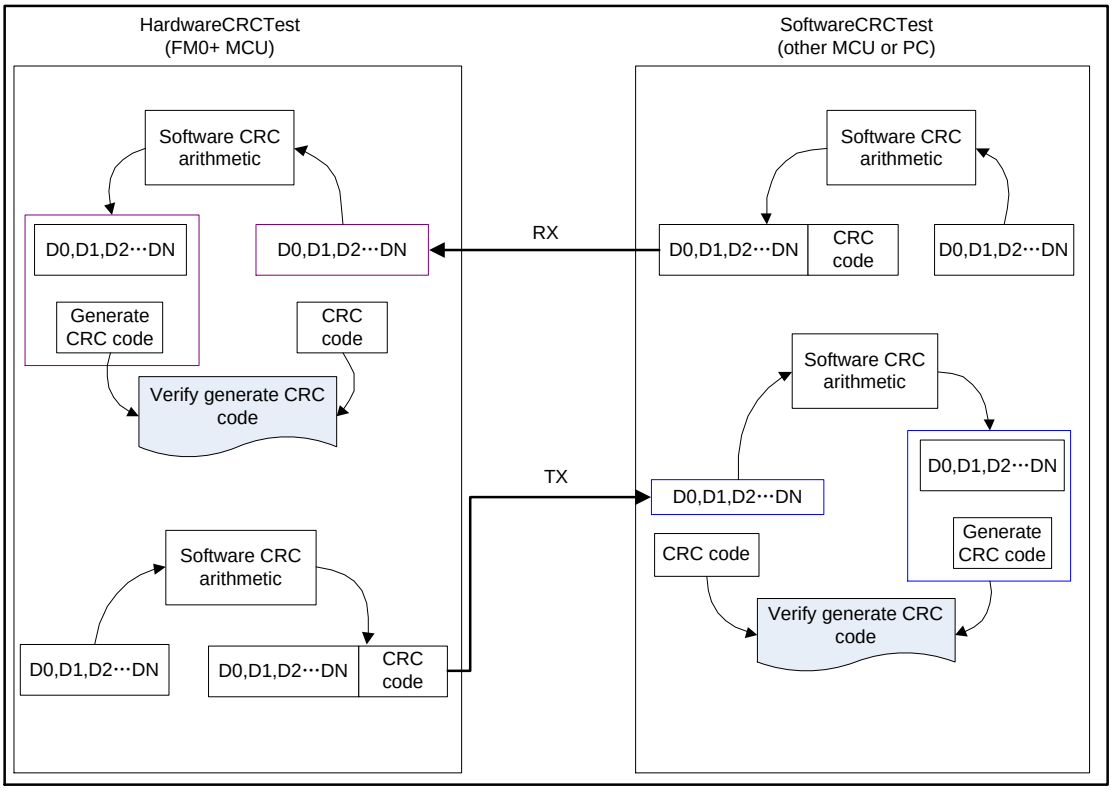
This module can either use CCITT CRC16 or IEEE-802.3 CRC32, which can be configured by CRCCR: CRC32 bit. In this module, the generator polynomials are fixed to the numeric values for those two modes.

- CCITT CRC16 generator polynomial: 0x1021(Omitted most significant bit of 0x11021)
- IEEE-802.3 CRC32 generator polynomial: 0x04C11DB7

Due to the lack of CRC hardware module, we use software CRC16/32 to do the testing.

Following figure shows an application of CRC test when FM0+ MCU communicates with other devices.

Figure 13. CRC test by communication



4.5.1 Test Description

To meet Class B requirement, Flash test must be checked for “single bit fault”. This test can be implemented as CRC16/32 test.

Enable the definition “FLASH_TEST_USE_CRC16” in IEC60730_user.h file if user wants to use CRC16 arithmetic for Flash test, otherwise CRC32 arithmetic will be implemented.

This test can be implemented at startup procedure to test whole code area, or it can also be called periodically to test sub blocks. Flash Test compares the generated CRC code at the time of test with the stored CRC code when build by a workbench tool. See, 8.1 **CRC code making method** for generating CRC code with a workbench tool.

Note: The CRC can also be used to test external communication data, which fulfills H.2.19.4.1 to detect hamming distance 3 errors.

- **Software CRC**

  - **Software CRC16 Arithmetic**

The CRC table enquiry method is used. The software CRC16 arithmetic should implement 6 steps to generate a new CRC code.

(1) Initialize CRC code in 0xFFFF.

(2) Store CRC code in “temp” after having divided it by 256.

(3) Left shift 8 bits of the CRC code.

(4) Store the CRC code by XOR CRC code with the data gotten from CRC table (use the data which calculated by XOR “temp” with the target data for a table index).

(5) Increment the target data for 1 byte.

(6) Repeat processes of (2) to (5) until byte size of target data.

The software CRC16 generation code and CRC16 table is shown as following figure.

Figure 14. Software CRC16 Generation Source Code

```
stl uint16 t IEC60730 SoftwareCRC16Gen(stl uint8 t *pData, stl uint32 t Size)
{
    stl uint8 t temp;
    stl uint8 t *p temp data = pData;
    stl uint16 t crc = 0xFFFF;
    while(Size-- != 0)
    {
        temp = crc/256;
        crc <<=8;
        crc ^= CRCTable[temp^*p temp data];
        p temp data++;
    }
    return crc;
}
```

Figure 15. CRC16 table

```
const stl uint16 t crc table[256]={
    0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
    0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,
    0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,
    0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,
    0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,
    0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,
    0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,
    0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,
    0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,
    0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,
    0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,
    0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,
    0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,
    0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,
    0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,
    0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,
    0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,
    0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,
    0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,
    0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,
    0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,
    0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
    0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,
    0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,
    0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,
    0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,
    0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,
    0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,
    0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,
    0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,
    0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,
    0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0
};
```

Software CRC32 Arithmetic

The CRC table enquiry method is used. The software CRC32 arithmetic should implement 6 steps to generate a new CRC code.

(1) Initialize CRC code in 0xFFFFFFFF.

(2) Store CRC code in “temp” after having 24 bits shifted it.

(3) Store the CRC code by XOR left 8 bits shifted CRC code with the data gotten from CRC table (use the data which calculated by XOR “temp” with the target data for a table index).

(4) Increment the target data for 1 byte.

(5) Repeat processes of (2) to (4) until byte size of target data.

(6) Finally, return reversed bit of CRC code.

The software CRC32 generation code and CRC32 table is shown as following figure.

Figure 16. Software CRC32 Generation Source Code

```
stl uint32 t IEC60730 SoftwareCRC32Gen(stl uint8 t *pData, stl uint32 t Size)
{
    stl uint8 t temp;
    stl uint8 t *p temp data = pData;
    stl_uint32_t crc = 0xFFFFFFFF;
    while(Size--)
    {
        temp=( crc >> 24 );
        crc = ( crc << 8 ) ^ CRCTable[temp^*p temp data];
        p temp data++;
    }
    return ~crc;
}
```

Figure 17. CRC32 table

```
const stl uint32 t CRCTable[256]={
    0x00000000L,  0x04c11db7L,  0x09823b6eL,  0x0d4326d9L,
    0x130476dcL,  0x17c56b6bL,  0x1a864db2L,  0x1e475005L,
    0x2608edb8L,  0x22c9f00fL,  0x2f8ad6d6L,  0x2b4bcb61L,
    0x350c9b64L,  0x31cd86d3L,  0x3c8ea00aL,  0x384fbdbdL,
    0x4c11db70L,  0x48d0c6c7L,  0x4593e01eL,  0x4152fda9L,
    0x5f15adacL,  0x5bd4b01bL,  0x569796c2L,  0x52568b75L,
    0x6a1936c8L,  0x6ed82b7fL,  0x639b0da6L,  0x675a1011L,
    0x791d4014L,  0x7ddc5da3L,  0x709f7b7aL,  0x745e66cdL,
    0x9823b6e0L,  0x9ce2ab57L,  0x91a18d8eL,  0x95609039L,
    0x8b27c03cL,  0x8fe6dd8bL,  0x82a5fb52L,  0x8664e6e5L,
    0xbe2b5b58L,  0xbaea46efL,  0xb7a96036L,  0xb3687d81L,
    0xad2f2d84L,  0xa9ee3033L,  0xa4ad16eaL,  0xa06c0b5dL,
    0xd4326d90L,  0xd0f37027L,  0xddb056feL,  0xd9714b49L,
    0xc7361b4cL,  0xc3f706fbL,  0xceb42022L,  0xca753d95L,
    0xf23a8028L,  0xf6fb9d9fL,  0xfbb8bb46L,  0xff79a6f1L,
    0xe13ef6f4L,  0xe5ffeb43L,  0xe8bccd9aL,  0xec7dd02dL,
    0x34867077L,  0x30476dc0L,  0x3d044b19L,  0x39c556aeL,
    0x278206abL,  0x23431b1cL,  0x2e003dc5L,  0x2ac12072L,
    0x128e9dcfL,  0x164f8078L,  0x1b0ca6a1L,  0x1fcdbb16L,
    0x018aeb13L,  0x054bf6a4L,  0x0808d07dL,  0x0cc9cdcaL,
    0x7897ab07L,  0x7c56b6b0L,  0x71159069L,  0x75d48ddeL,
    0x6b93dddbL,  0x6f52c06cL,  0x6211e6b5L,  0x66d0fb02L,
    0x5e9f46bfL,  0x5a5e5b08L,  0x571d7dd1L,  0x53dc6066L,
    0x4d9b3063L,  0x495a2dd4L,  0x44190b0dL,  0x40d816baL,
    0xaca5c697L,  0xa864db20L,  0xa527fdf9L,  0xa1e6e04eL,
    0xbfa1b04bL,  0xbb60adfcL,  0xb6238b25L,  0xb2e29692L,
    0x8aad2b2fL,  0x8e6c3698L,  0x832f1041L,  0x87ee0df6L,
    0x99a95df3L,  0x9d684044L,  0x902b669dL,  0x94ea7b2aL,
    0xe0b41de7L,  0xe4750050L,  0xe9362689L,  0xedf73b3eL,
    0xf3b06b3bL,  0xf771768cL,  0xfa325055L,  0xfef34de2L,
    0xc6bcf05fL,  0xc27dede8L,  0xcf3ecb31L,  0xcbffd686L,
    0xd5b88683L,  0xd1799b34L,  0xdc3abdedL,  0xd8fba05aL,
    0x690ce0eeL,  0x6dcdfd59L,  0x608edb80L,  0x644fc637L,
    0x7a089632L,  0x7ec98b85L,  0x738aad5cL,  0x774bb0ebL,
    0x4f040d56L,  0x4bc510e1L,  0x46863638L,  0x42472b8fL,
    0x5c007b8aL,  0x58c1663dL,  0x558240e4L,  0x51435d53L,
    0x251d3b9eL,  0x21dc2629L,  0x2c9f00f0L,  0x285e1d47L,
    0x36194d42L,  0x32d850f5L,  0x3f9b762cL,  0x3b5a6b9bL,
    0x0315d626L,  0x07d4cb91L,  0x0a97ed48L,  0x0e56f0ffL,
    0x1011a0faL,  0x14d0bd4dL,  0x19939b94L,  0x1d528623L,
    0xf12f560eL,  0xf5ee4bb9L,  0xf8ad6d60L,  0xfc6c70d7L,
    0xe22b20d2L,  0xe6ea3d65L,  0xeba91bbcL,  0xef68060bL,
    0xd727bbb6L,  0xd3e6a601L,  0xdea580d8L,  0xda649d6fL,
    0xc423cd6aL,  0xc0e2d0ddL,  0xcda1f604L,  0xc960ebb3L,
    0xbd3e8d7eL,  0xb9ff90c9L,  0xb4bcb610L,  0xb07daba7L,
    0xae3afba2L,  0xaafbe615L,  0xa7b8c0ccL,  0xa379dd7bL,
    0x9b3660c6L,  0x9ff77d71L,  0x92b45ba8L,  0x9675461fL,
    0x8832161aL,  0x8cf30badL,  0x81b02d74L,  0x857130c3L,
    0x5d8a9099L,  0x594b8d2eL,  0x5408abf7L,  0x50c9b640L,
    0x4e8ee645L,  0x4a4ffbf2L,  0x470cdd2bL,  0x43cdc09cL,
    0x7b827d21L,  0x7f436096L,  0x7200464fL,  0x76c15bf8L,
    0x68860bfdL,  0x6c47164aL,  0x61043093L,  0x65c52d24L,
    0x119b4be9L,  0x155a565eL,  0x18197087L,  0x1cd86d30L,
    0x029f3d35L,  0x065e2082L,  0x0b1d065bL,  0x0fdc1becL,
    0x3793a651L,  0x3352bbe6L,  0x3e119d3fL,  0x3ad08088L,
    0x2497d08dL,  0x2056cd3aL,  0x2d15ebe3L,  0x29d4f654L,
    0xc5a92679L,  0xc1683bceL,  0xcc2b1d17L,  0xc8ea00a0L,
    0xd6ad50a5L,  0xd26c4d12L,  0xdf2f6bcbL,  0xdbee767cL,
    0xe3a1cbc1L,  0xe760d676L,  0xea23f0afL,  0xeee2ed18L,
    0xf0a5bd1dL,  0xf464a0aaL,  0xf9278673L,  0xfde69bc4L,
    0x89b8fd09L,  0x8d79e0beL,  0x803ac667L,  0x84fbdbd0L,
    0x9abc8bd5L,  0x9e7d9662L,  0x933eb0bbL,  0x97ffad0cL,
    0xafb010b1L,  0xab710d06L,  0xa6322bdfL,  0xa2f33668L,
    0xbcb4666dL,  0xb8757bdaL,  0xb5365d03L,  0xb1f740b4L
};
```

4.5.2 API Definition

Use CRC16 arithmetic to implement Flash test

Name	IEC60730_SoftwareCRC16Gen
Parameter	pData: test data address Size: data size
Return	CRC value

Description:

This API implements CRC16 generation by software CRC arithmetic. The CRC table enquiry method is used.

Name	IEC60730_SoftwareCRC16Test
Parameter	pData: test data address Size: data size Crc: expected CRC code
Return	0: IEC60730_TEST_NORMAL 1: IEC60730_TEST_FUNC_ERROR

Description:

This API implements software CRC16 test.

Use CRC32 arithmetic to implement Flash test

Name	IEC60730_SoftwareCRC32Gen
Parameter	pData: test data address Size: data size
Return	CRC value

Description:

This API implements CRC32 generation by software CRC arithmetic. The CRC table enquiry method is used.

Name	IEC60730_SoftwareCRC32Test
Parameter	pData: test data address Size: data size Crc: expected CRC code
Return	0: IEC60730_TEST_NORMAL 1: IEC60730_TEST_FUNC_ERROR

Description:

This API implements software CRC32 test.

4.6 Variable Memory Test

Variable memory test in FM0+ MCU means SRAM test, the SRAM size can be configured according to different product, shown as table 1-1.

4.6.1 Test Description

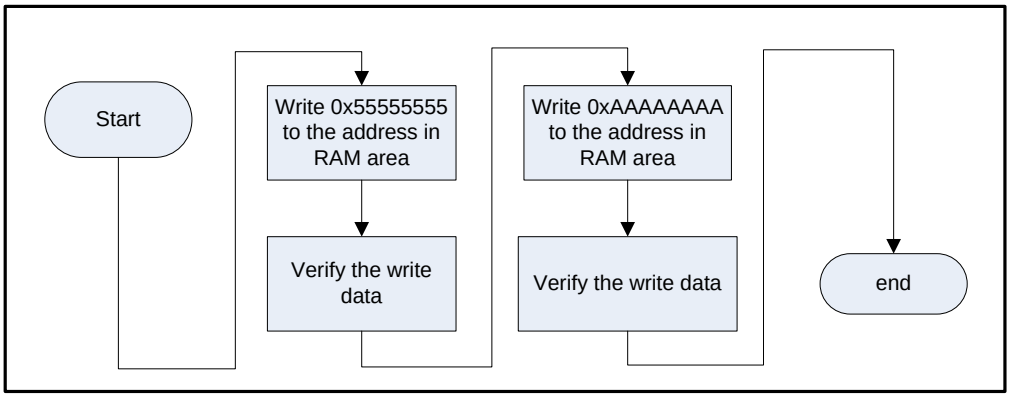
To meet Class B requirement, SRAM test must be checked for “DC fault”. A simple checkerboard method is used to implement this SRAM.

This test can be implemented at startup procedure to test entire SRAM area. And it can also test sub blocks periodically when code is running, however user should pay attention that the data will be destroyed after test.

As all RAM area is involved in this test, it is better not to use variable in this test, so assembly is used to implement register test. And as it is highly critical, it is designed that once RAM test error is detected, program will run into an infinite loop.

The procedure to test 1 word data is shown as following figure.

Figure 18. Test 1 Word with Checkerboard Method



4.6.2 API Definition

Name	iec60730_ram_test
Parameter	StartAddr(R0): start RAM address EndAddr(R1): end RAM address
Return	None

Description:

This API tests SRAM area with Checkerboard arithmetic which writes alternate “0” and “1” to memory, and verifies if the write data is right by reading back the data written. It can detect stuck-at faults and direct coupling faults.

This test should be called in startup procedure, and it can also be called in cycle, but the data is not saved after test.

4.7 IO Test

FM4 MCU has up to 8 IO ports: Port0-Port8, each port has up to 16 channels. These ports can be configured according to different package.

4.7.1 Test Description

To meet Class B requirement, GPIO must be check for “Function error”. So function test is implemented for both input and output function. The IO direction can be configured by IO register shown in figure 4-14. Please refer to the peripheral manual for detail of GPIO.

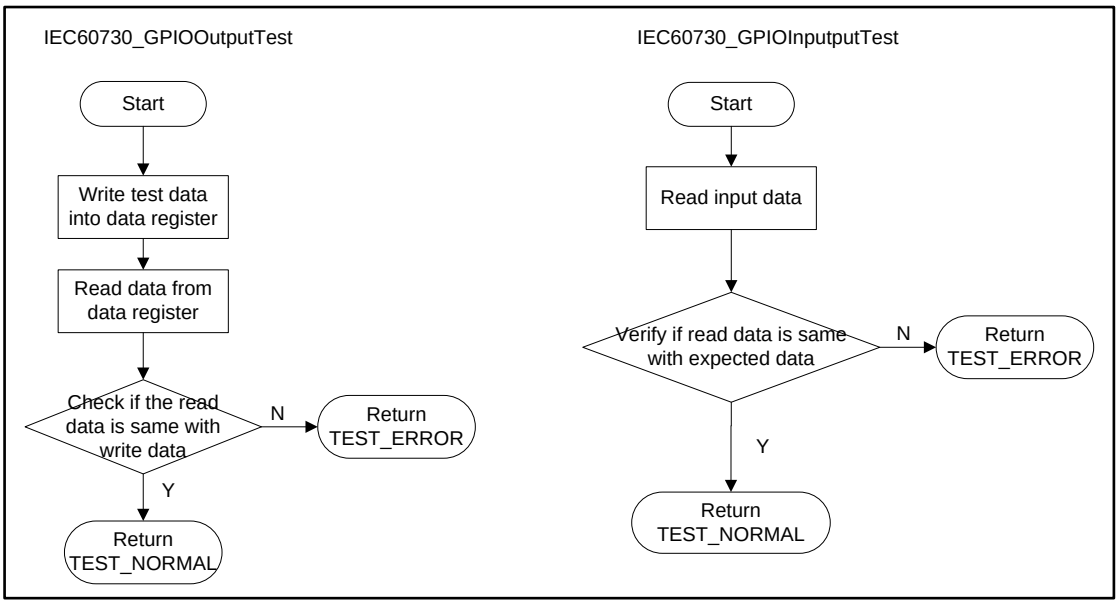
- Input IO configuration: ADE=0,PFR=0,DDR=0
- Output IO configuration: ADE=0,PFR=0,DDR=1

Table 2. IO Function Configuration

I/O Port Function		ADE/ SPSR	PFR	DDR	PCR
Available main function	**Available sub function**				
Special pin Analog Input Oscillation	N/A	1	-	-	Disconnect
GPIO function input pin	Peripheral function input pin	0	0	0	Valid
GPIO function output pin	GPIO function input pin (FB) Peripheral function input pin (FB)			1	Disconnect
Peripheral function output pin	GPIO function input pin (FB) Peripheral function input pin (FB)		1	-	Disconnect
Peripheral function bidirectional pin	GPIO function input pin (FB) Peripheral function input pin (FB)				Valid
Peripheral function input pin	GPIO function input pin				Valid

The IO input test checks if selected IO input value which stores in PDIR is same with expected value, . And IO output can check if output value by which stores in PDOR is correct. These tests should be tested in startup procedure as function test.

Figure 19. Input /Output Test Flowchart



4.7.2 API Definition

Name	IEC60730_GPIOOutputTest
Parameter	Port: port number Bit: bit number Value: output level
Return	0: IEC60730_TEST_NORMAL 1: IEC60730_TEST_FUNC_ERROR 2: IEC60730_TEST_PARA_ERROR

Description:

This API implements GPIO output test by setting a level for output pin and check if read back value is the expected value.

Name	IEC60730_GPIOInputTest
Parameter	Port: port number Bit: bit number Value: expected pin level
Return	0: IEC60730_TEST_NORMAL 1: IEC60730_TEST_FUNC_ERROR 2: IEC60730_TEST_PARA_ERROR

Description:

This API implements GPIO input test by reading the value from input pin and check if read value is the expected value.

4.8 AD Test

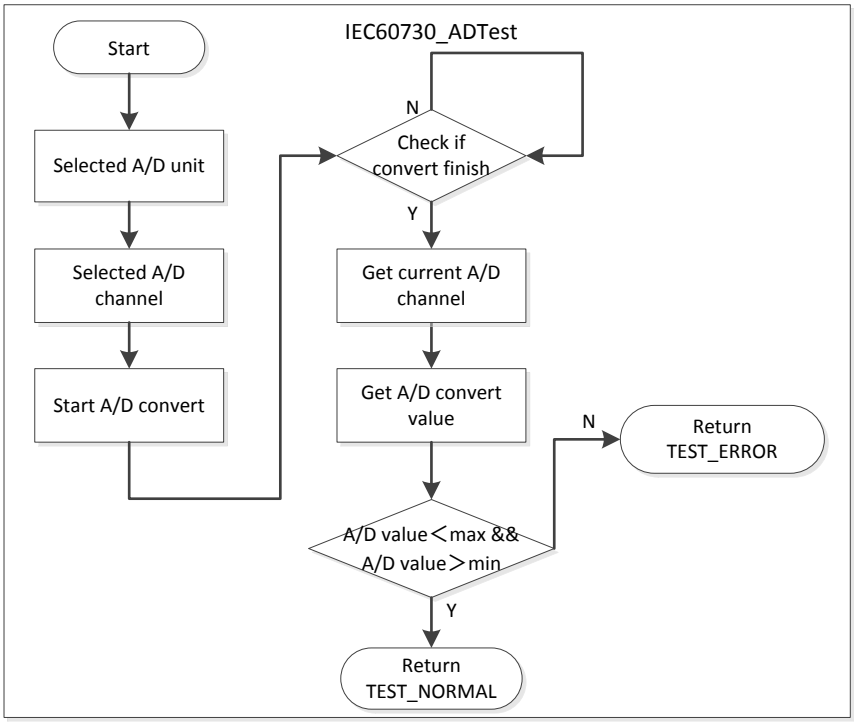
FM0+ MCU integrates a 12bit AD module. It has 1 unit with totally 8 channels.

4.8.1 Test Description

To meet Class B requirement, AD must be check for “Function error”. This test samples AD signal from selected AD channels and check if the AD convert values are in the expected ranges.

Scan mode is used, multi-channel can be tested at the same time. The AD test flowchart of checking single-channel is shown as following figure.

Figure 20. AD Test Flowchart



4.8.2 API Definition

Name	IEC60730_ADTest
Parameter	ADTest_Info: a ad_test_info_t structure typedef struct ad_test_info { uint8_t ADUnit; /* unit num, 8/10 bit A/D -> 0/1/2 */ uint8_t *Ch; /* pointer to AD channel num */ uint8_t ChSize; /* channel size */ uint16_t *ExpLowerValue; /* pointer to expected lower value */ uint16_t *ExpUpperValue; /* pointer to expected upper value */ } ad_test_info_t;
Return	0: IEC60730_TEST_NORMAL 1: IEC60730_TEST_FUNC_ERROR 2: IEC60730_TEST_PARA_ERROR

Description:

This API implements AD test by checking if AD convert result is in expected range. It should be implemented in startup procedure.

5 Example project

Two demo projects are provided according to IAR and keil IDE. This chapter introduces IAR demo project based on Spansion SK-FM0P-48LQFP-9AF160K V1.0.0 and shows how to integrate the IEC60730 STL into a real system.

5.1 User Configuration

User should first configure some definitions in IEC60730_user.h file.

5.1.1 The definition “IEC60730_FLASHTEST_USE_CRC16”

If user wants to use CRC16 arithmetic for Flash test, enable this definition, if user wants to use CRC32 arithmetic for Flash test, disable this definition.

In this demo program, CRC32 arithmetic is used.

5.1.2 The definition “IEC60730_CLKTEST_USE_CSV”

If user wants to use CSV to implement clock test, enable this definition, or clock test will be done with watch counter as standard timer, which is sourced by sub clock.

In the demo program, the latter method is demonstrated.

5.2 Project Structure

Class B STL routines are divided into two main processes: startup and periodic self-tests. The periodic test must be initialized by a set-up block before it is applied.

5.2.1 Startup Self-Test

PC, register, SRAM test are all startup self-tests, and they should be called in reset handler.

And Flash, AD, IO can be tested after system clock initialization after program jumps into main function.

For AD test, channel 0/3 (Analog input) are used for test. P10 should connect to VCC and P13 need to connect to GND.

For IO input test, key input pin P04/ P0F are used for test.

5.2.2 Periodic Test Initialization

Interrupt and clock test should be initialized before tests start.

- Interrupt Test Initialization

It is designed that a dual time interrupt is used to monitor reload timer 0-3. The initialization setting parameter is shown as following table.

Interrupt Name	Interrupt Interval	interrupt of dual timer	Standard Frequency	Pre-defined Range
Reload timer 0	2.5ms	25ms	10	[8,12]
Reload timer 1	1ms	25ms	25	[22, 28]
Reload timer 2	500us	25ms	50	[45,55]
Reload timer 3	250us	25ms	100	[95,105]

Table 5-1: Input /Output Test Flowchart

- Clock Test Initialization

The CPU clock is HCLK, and the source clock of dual timer in this system is set to PCLK0 (HCLK/2). So the source clock of dual timer can be tested indirectly instead of CPU clock by watch counter.

It is designed that the interrupt Interval of watch counter is 0.5s and interrupt Interval of dual timer is 25ms, so the Standard Frequency of dual timer is 20 and the accepted range is set between 18 and 22. Assume it takes 10 cycles to implement main loop. So the minimum execution time of main loop is 1/4000000 s, so set the threshold value to 10000000.

5.2.3 Periodic Test

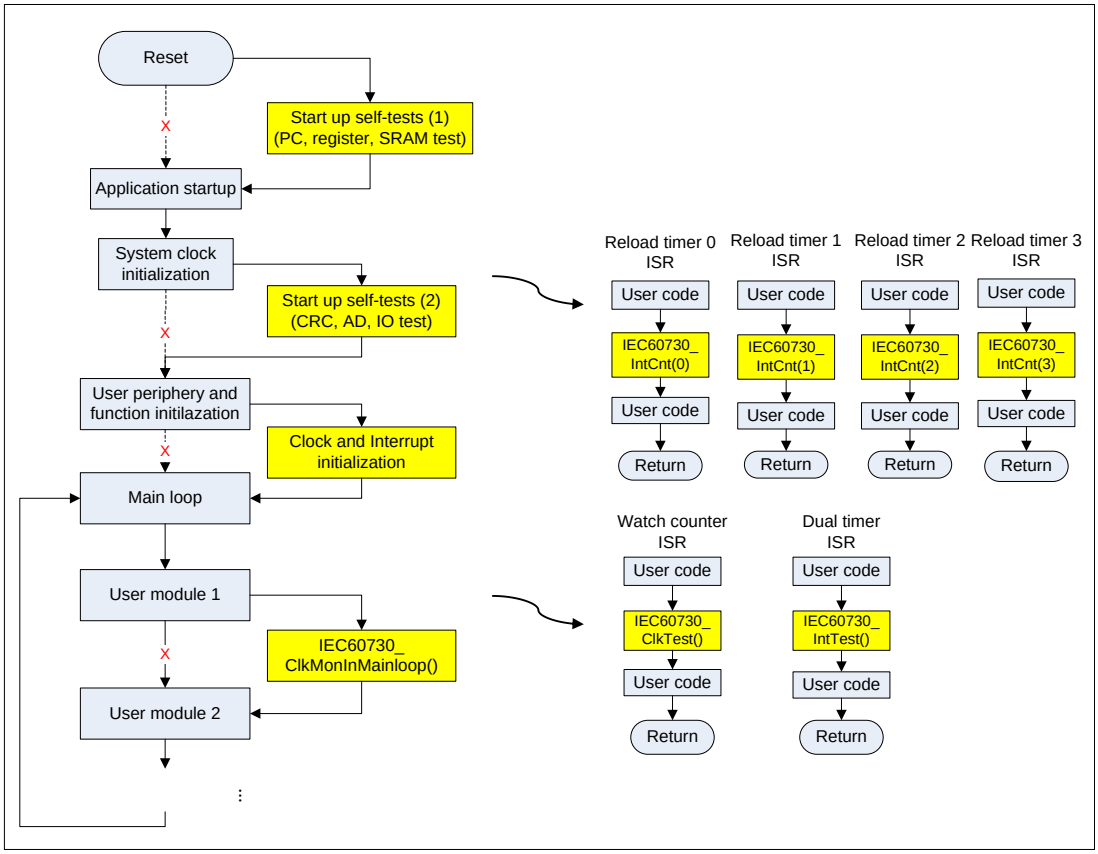
The interrupt and clock test should be tested in period when code is running.

Integrate **IEC60730_IntTest** into dual timer interrupt and **IEC60730_IntCnt** into each reload timer interrupts.

Integrate **IEC60730_ClkTest** into watch counter interrupt, **IEC60730_ClkCnt** into dual timer interrupt, and **IEC60730_ClkMonInMainloop** into main loop.

The Figure 5-1 shows the basic principle of how to integrate the Class B software package into this application software.

Figure 21. Project Structure



5.3 Sample Code

5.3.1 Startup File

- **Reset handler**

Figure 22. Reset Handler Sample Code

```
Reset_Handler
                BL      iec60730_reg_test  ; after reset, test register first
                BL      iec60730_pc_test   ; test pc
                LDR     R0, =0x20000000    ; set RAM start address
                LDR     R1, =0x200017FF    ; set RAM end address
                BL      iec60730_ram_test  ; test all D-RAM area
```

5.3.2 Main File

- **Main function**

Figure 23. Main Function Sample Code

```
uint32 t main(void)
{
    uint32 t sw crc;
    uint8 t a[10] = {0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,0x99};

    /* Use CSV to implement clock test */
#ifdef IEC60730 CLKTEST USE CSV
    uint16 t reg rst str;
    fcs mon info t fcs mon info = {FCS MON DISABLE, 5};
    if(IEC60730 TEST NORMAL != IEC60730 CheckCSVStat(&reg rst str))
    {
        while(1);
    }
    IEC60730 InitCSV(CSV MCLK MON ENABLE, CSV SCLK MON ENABLE, fcs mon info);
#endif

    SystemInit();
#ifdef IEC60730 FLASHTEST USE CRC16
    /* use software CRC16 to calculate expected crc first,
       then verify if the CRC code calculated by hardware is same with expected crc */
    sw crc = IEC60730 SoftwareCRC16Gen(a, sizeof(a));
    if(IEC60730 TEST NORMAL != IEC60730 SoftwareCRC16Test(a, sizeof(a), sw crc))
    {
        while(1);
    }
#else
    /* use software CRC32 to calculate expected crc first,
       then verify if the CRC code calculated by hardware is same with expected crc */
    sw crc = IEC60730 SoftwareCRC32Gen(a, sizeof(a));
    if(IEC60730 TEST NORMAL != IEC60730 SoftwareCRC32Test(a, sizeof(a), sw crc))
    {
        while(1);
    }
#endif
    /* GPIO output test
    *  test P61 (control LED)
    */
#ifdef SK FM0P 48LQFP 9AF160K V1 0 0
    /* Test LED PORT Output */
    if(IEC60730 TEST NORMAL != IEC60730 GPIOOutputTest(LED PORT,LED PIN,TEST PIN LOW))
    {
        while(1);
    }
    if(IEC60730 TEST NORMAL != IEC60730 GPIOOutputTest(LED PORT,LED PIN,TEST PIN HIGH))
    {
        while(1);
    }
#endif
    /* GPIO input test
    *  test P04
    *  test P0F
    */
#ifdef SK FM0P 48LQFP 9AF160K V1 0 0
    /* Test GPIO P04 input */
    if(IEC60730 TEST NORMAL != IEC60730 GPIOInputTest(PORT NUM 0, BIT NUM 4, TEST PIN HIGH))
    {
        while(1);
    }
    /* Test GPIO P0F input */
    if(IEC60730 TEST NORMAL != IEC60730 GPIOInputTest(PORT NUM 0, BIT NUM F, TEST PIN HIGH))
    {
        while(1);
```

```
    }
#endif

    /* Init LEDs  */
    LED Init();

    /* Init Buttons */
    Button Init();

#ifdef SK FM0P 48LQFP 9AF160K V1 0 0
   /* AD test  - Check if input is in expected range.
    *    Steps - 1.Connect AN00 test pin (P10-NO.25)to VCC
    *          - 2.Connect AN03 test pin (P13-NO.28)to GND
    *          - This ADTest function will sample the analog value and compare
    *            with the set range
    */
    if(IEC60730 TEST NORMAL != IEC60730 ADTest(ADC UNIT0, CH00, 0xFFA, 0x1000))
    {
        while(1);
    }
    if(IEC60730 TEST NORMAL != IEC60730 ADTest(ADC UNIT0, CH03, 0x000, 0x010))
    {
        while(1);
    }
#endif
    /* Interrupt test initialization */
    IEC60730 IntTestInit(IntTest Freq,     \
                         IntTest FreqLower,\
                         IntTest FreqUpper,\
                         IntTest FreqInit, \
                         sizeof(IntTest Freq)/sizeof(uint32 t));

#ifndef IEC60730 CLKTEST USE CSV
    /* clock test initialization
     *  test CPU clock by checking if the 25ms interval time is set for dual timer,
     *  the occurrence frequency of dual-time is about 20 per 500ms(produced by watch counter)
     *  1 cycle time = (1/40MHz). Assume it takes 10 cycles to implement main loop.
     */
    IEC60730 ClkInit(18, 22, 10000000);

    /* Initialize watch-counter */
    WTC Init();
#endif
    /* Initialize dual-timer */
    DT Init();

    /* Initialize 4 base-timers */
    BT Init();

    /* Main Loop */
    while(1)
    {
        /* Wait for timer tick- LED will keep blinking */
        /* Dual-Timer Tmr1Tick-25ms */
        if(40 == Tmr1Tick)
        {
             LED Twinkle();
             Tmr1Tick = 0;                    // clear timer tick
        }

#ifndef IEC60730 CLKTEST USE CSV
        /* monitor watch counter interrupt */
        if(IEC60730 TEST NORMAL != IEC60730 ClkMonInMainloop())
        {
            while(1);
        }
#endif
    }
```

- **Dual Timer ISR**

Figure 24. Dual Timer ISR

```
void DT QDU IRQHandler(void)
{
    if(1 == FM0P DTIM->TIMER1RIS&0x01)
    {
        FM0P_DTIM->TIMER1INTCLR = 1;
#ifndef IEC60730 CLKTEST USE CSV
        /* count the clock tick     */
        IEC60730_ClkCnt();
#endif
        /* Set timer tick for LEDs  */
        Tmr1Tick++;
        /* implement interrupt test */
        if(IEC60730 TEST NORMAL != IEC60730 IntTest())
        {
            while(1);
        }
    }
}
```

- **Watch Counter ISR**

Figure 25. Watch Counter ISR

```
void TIM IRQHandler(void)
{
    if(1 == bFM0P INTREQ IRQ24MON WCINT)
    {
        /* Clear interrupt flag    */
        FM0P WC->WCCR &= 0xFE;

        /* implement clock test    */
        if(IEC60730 TEST NORMAL != IEC60730 ClkTest())
        {
            while(1);
        }
    }
}
```

- **Reload Timer ISR**

Figure 26. Reload Timer ISR

```
void BT0 3 FLASH IRQHandler(void)
{
    if(FM0P BT0 RT->STC&0x01)
    {
        FM0P BT0 RT->STC = FM0P BT0 RT->STC & 0xFE;
        IEC60730 IntCntPro(0);       /* count frequency value for interrupt 0 */
    }
    else if(FM0P BT1 RT->STC&0x01)
    {
        FM0P BT1 RT->STC = FM0P BT1 RT->STC & 0xFE;
        IEC60730 IntCntPro(1);       /* count frequency value for interrupt 1 */
    }
    else if(FM0P BT2 RT->STC&0x01)
    {
        FM0P BT2 RT->STC = FM0P BT2 RT->STC & 0xFE;
        IEC60730_IntCntPro(2);       /* count frequency value for interrupt 2 */
```

```
    }
    else if(FM0P_BT3_RT->STC&0x01)
    {
        FM0P_BT3_RT->STC = FM0P_BT3_RT->STC & 0xFE;
        IEC60730_IntCntPro(3);       /* count frequency value for interrupt 3 */
    }
}
```

6 STL API Performance

Table 2. STL API Performance

API Name	Execution time (Cycles)	Stack usage (Bytes)	ROM Usage (Bytes)	RAM usage (Bytes) (Global variable)
iec60730_pc_test	48	0	200	0
iec60730_reg_test	222	0	604	0
IEC60730_IntTestInit	81 (4 interrupts)	8	62	0
IEC60730_IntCntPro	22	4	46	0
IEC60730_IntTest	168 (4 interrupts)	4	98	20
IEC60730_ClkCnt	14	4	36	0
IEC60730_ClkTest	43	8	84	32
IEC60730_ClkMonInMainloop	43	8	76	0
IEC60730_ClkTestReset	15	0	32	0
IEC60730_InitCSV	62	16	264	0
IEC60730_CheckCSVStat	11	4	80	0
IEC60730_SoftwareCRC16Gen	191 (10 bytes data)	16	66+ 512 (CRC table)	0
IEC60730_SoftwareCRC16Test	199 (10 bytes data)	8	22	0
IEC60730_SoftwareCRC32Gen	169 (10 bytes data)	8	48+ 1024 (CRC table)	0
IEC60730_SoftwareCRC32Test	176 (10 bytes data)	8	20	0
iec60730_ram_test	104 (16 bytes data)	0	80	0
IEC60730_GPIOOutputTest	127	28	288	0
IEC60730_GPIOInputTest	136	36	320	0
IEC60730_ADTest	1014	76	876	0

Note: The code execution cycle is tested in normal run status.

Note: The ROM size of this STL is 3468 bytes. (Use CRC16 for Flash test, and watch counter for clock test)

Note: The code is compiled by IAR Embedded Workbench IDE V6.50 and optimization level was set to “Low”.

7 Reference Documents

[1]. IEC 60730-1 Reference Manual Edition3.2, 2007

[2]. ARMv7-M Architecture Application Level Reference Manual, 2008

[3]. Cortex-M0P r0p1 Technical Reference Manual, 2012

[4]. S6E1A1-DS710-00001 (FM0+ Family - S6E1A1 Series Data Sheet)

[5]. Spansion 32-bit Microcontroller FM0+ Peripheral Manual, 2013

[6]. IAR SYSTEM Technical Note 65473 – IELFTOOL Checksum – Basic actions

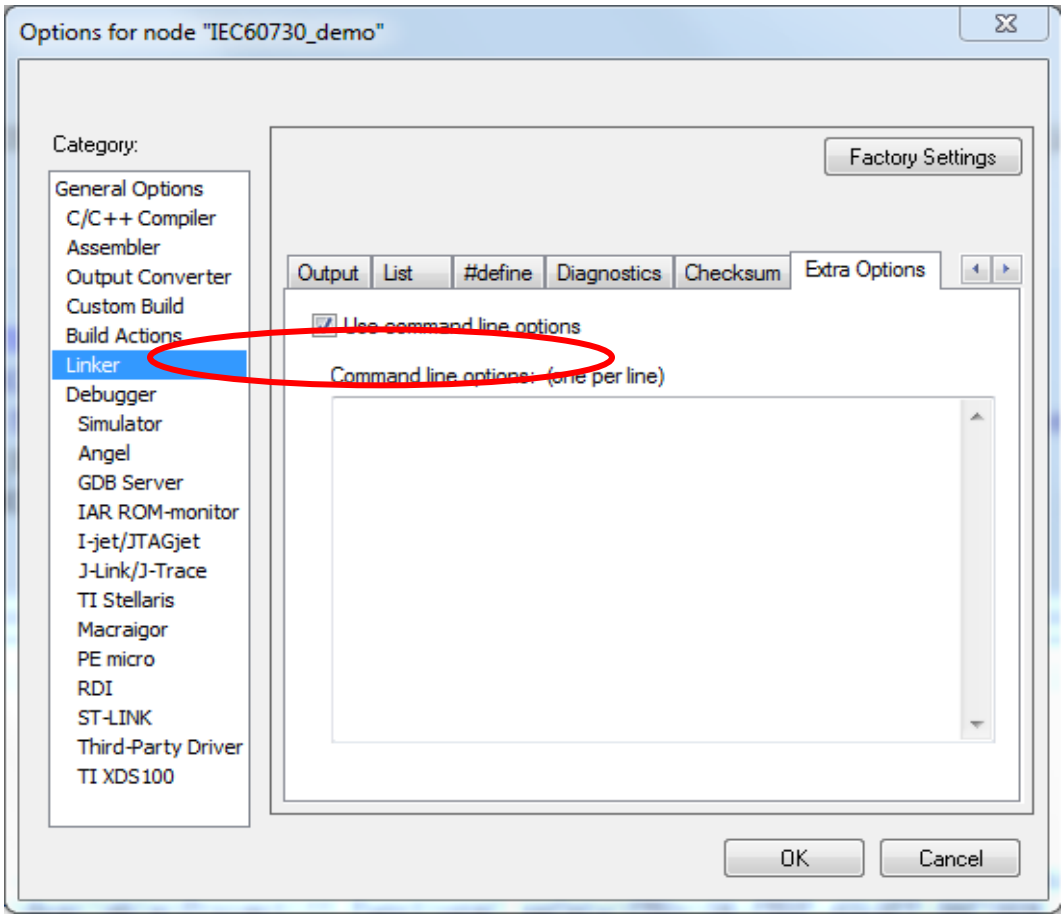
8 Appendix

8.1 CRC code making method

The method to make CRC code to use in 4.1 CPU Register Test, follows is example of IAR Embedded Workbench. Please refer to IAR's manual for details (IAR Technical Note 65473 – IELFTOOL Checksum – Basic actions).

8.1.1 Start of the Command-Line

Click "Project"→"Options"→"Linker"→"Extra options" tabs, then check the "Use command line options".



8.1.2 Input the command

"--place_holder" command

"--place_holder" is used that make CRC code and a section in ROM. If input the following command, to set the size of section in 4byte and the alignment in 1.

--place_holder __checksum,4,.checksum,1

"--fill" command

The unused area of the target area needs to fill with optional value making the CRC code. Therefore, use “--fill” command. If input the following command, 0x00000000-0x00003FFF is filled with 0xFF.

--fill 0xFF;0x0000-0x3FFF

If input the following command, 0x00000000-0x00003FFF, 0x5000-0x5FFF and 0x6500-0x6FFF are filled with 0xFF.

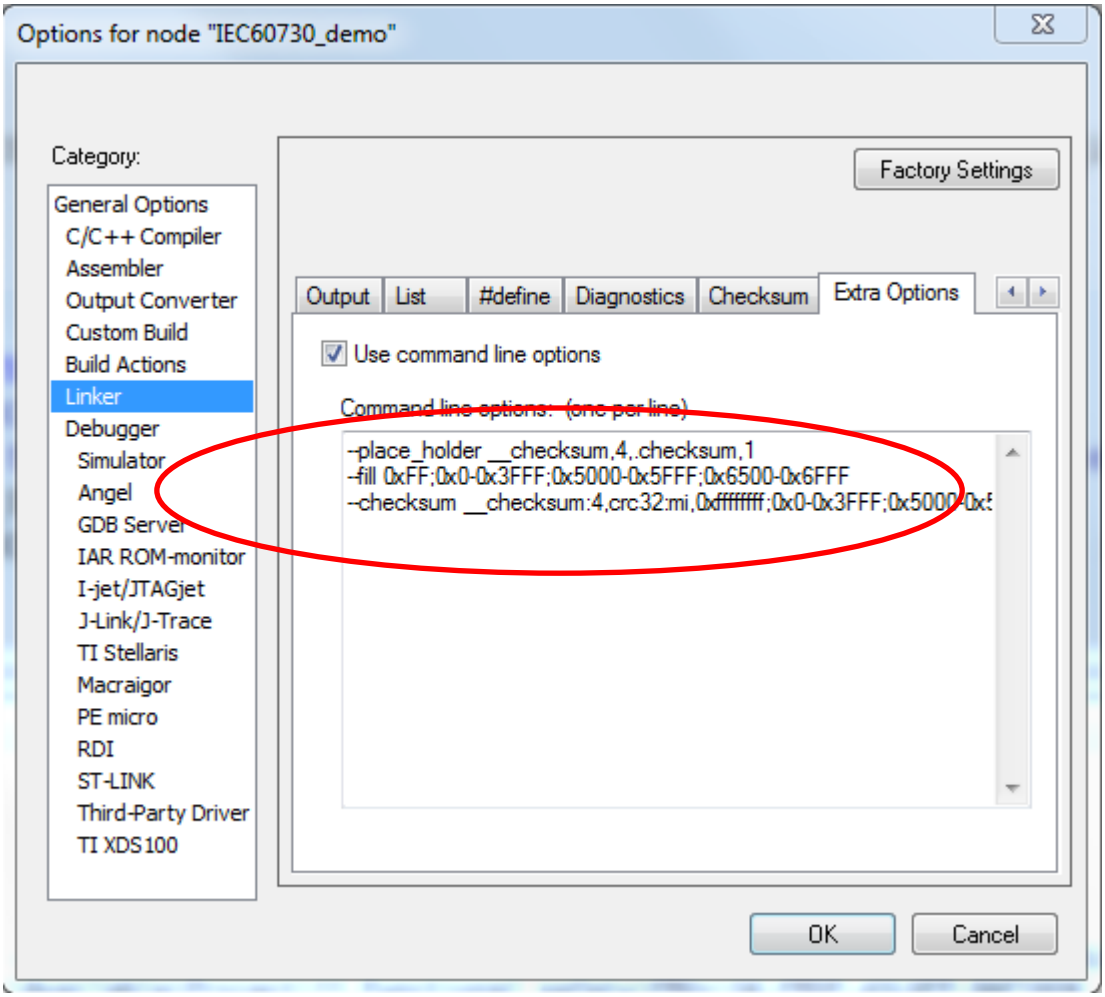
--fill 0xFF;0x0-0x3FFF;0x5000-0x5FFF;0x6500-0x6FFF

“--checksum” command

Set algorithm of CRC. If input the following command, you can set items as follow. The CRC code is stored in the symbol name “__checksum”, the CRC code size is 4byte, the algorithm is CRC32, calculation is LSB first, CRC code is initialized by 0xFFFFFFFF, 0x00000000-0x00003FFF, 0x5000-0x5FFF and 0x6500-x6FFF are filled with 0xFF.

--checksum __checksum:4,crc32:mi,0xffffffff;0x0-0x3FFF;0x5000-0x5FFF;0x6500-0x6FFF

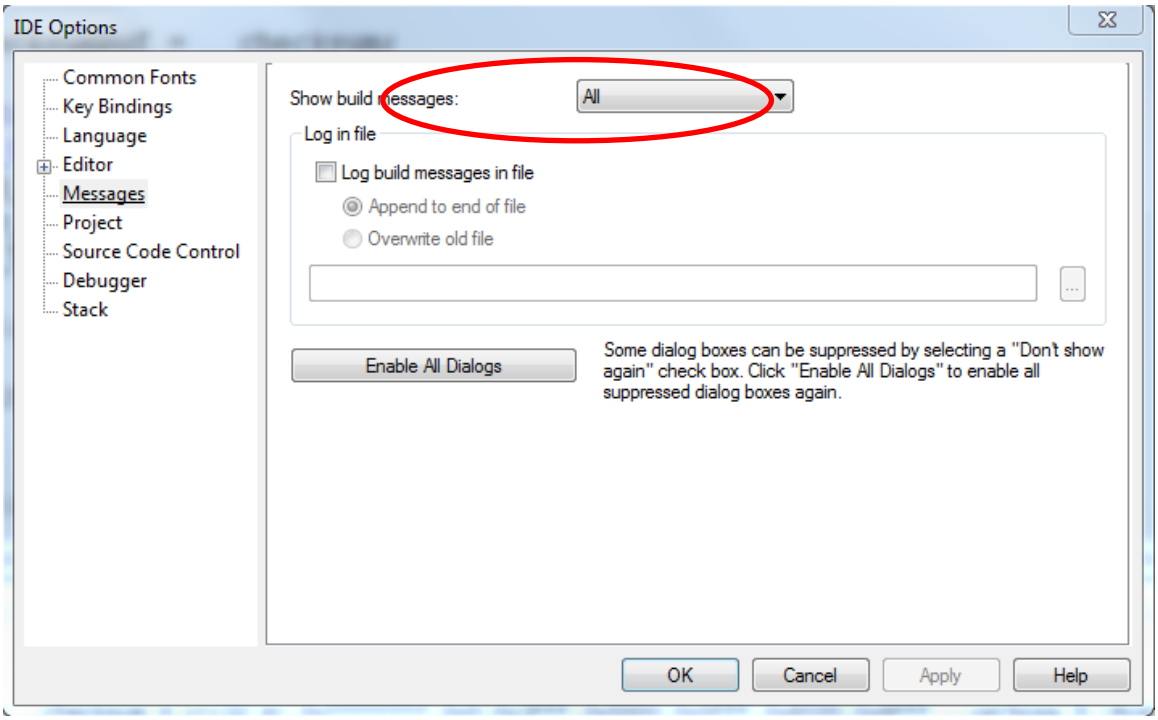
If input the command mentioned above (1, 2, and 3), close the window by clicking the “OK”.



8.1.3 Setting of build messages to display in the message window

If set the following contents, you can display build messages at the time of make to the message window.

Click “Tools”→”Options”→”Messages” tabs, then select the “All” from the combo box of “Show build messages”. Finally, close the window by clicking the “OK”.



8.1.4 Setting of the Linker configuration file

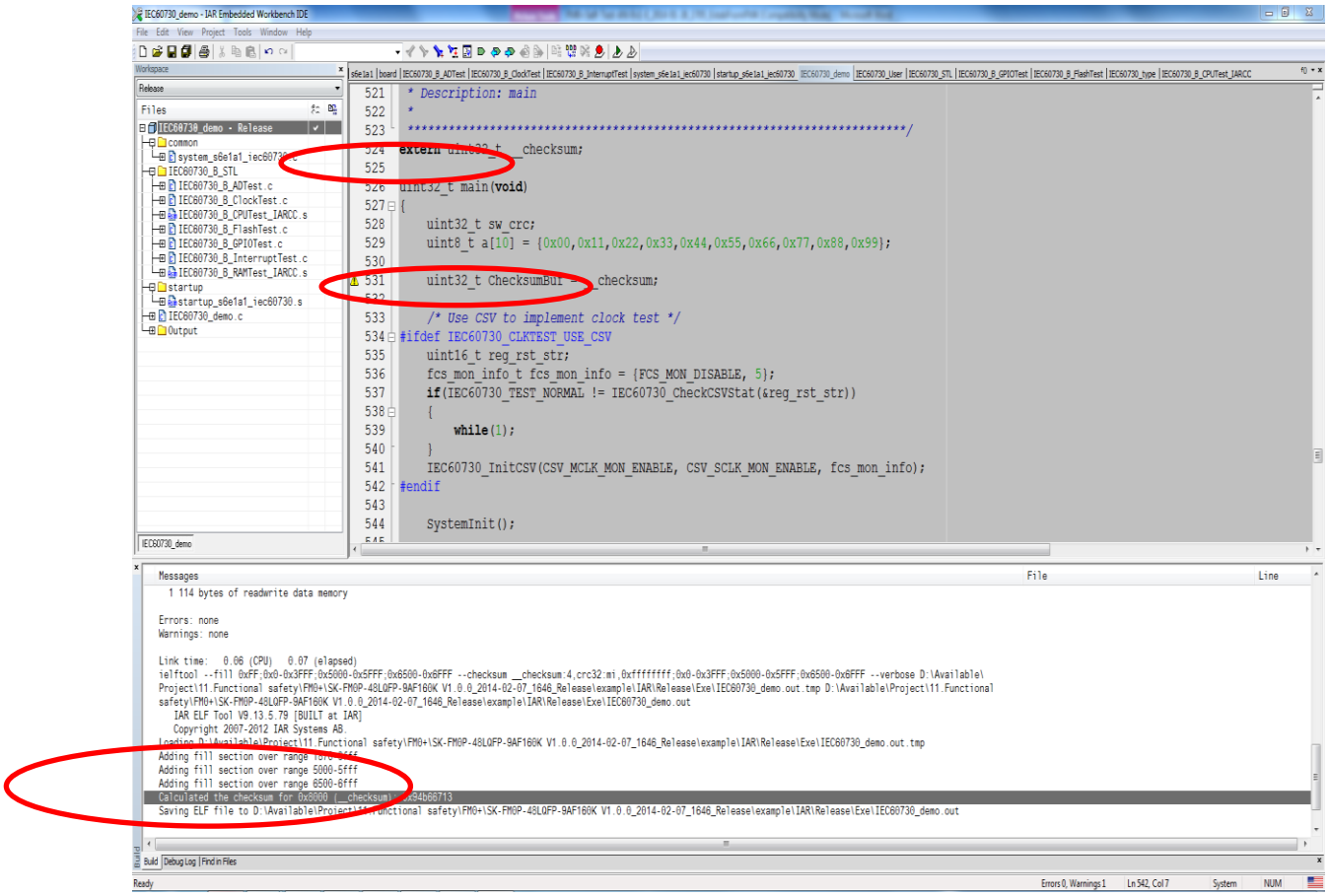
Add settings to the Linker configuration file to store CRC code in Flash. In the case of debug mode, you must use “S6E1A11x0A_ram.icf” file. In the case of release mode, you must use “S6E1A11x0A.icf” file.

Configuring the CRC code to store at address 0x8000, please add the commands below:

```
define symbol __ICFEDIT_checksum_start__ = 0x00008000;
place at address mem: __ICFEDIT_checksum_start__ { readonly section .checksum };
```

8.1.5 Making CRC code

Confirm that the CRC code was made after make.



Document History

Document Title: AN205411 - FM0+ IEC60730 Class B Self-Test Library

Document Number: 002-05411

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	_	XLZH	02/07/2014	Initial Release
*A	5029006	XLZH	12/03/2015	Migrated Spansion Application Note “MCU-AN-510126-E-10” to Cypress format
*B	5873207	AESATP12	09/06/2017	Updated logo and copyright.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2014-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.