Introduction to Nuvoton IEC60730-1 Class B STL

Application Note for NuMicro® M451 Series

Document Information

Abstract	This application note introduces Nuvoton IEC60730-1 Class B Software Test Library (STL) consisting of low-level software routines. These routines implement basic requirements specified in IEC60730-1 Annex H MCU part of the standard. The user can add STL into existing application to meet the requirements from IEC60730-1.
Apply to	NuMicro® M451 Series.

Table of Contents

1 Overview

IEC60730-1 is a safety standard for all home appliances sold in Europe. To accelerate certification process in products, Nuvoton provides users with the sample code consisting of low-level software routines. These routines implement basic requirements specified in Annex H of the IEC60730-1 standard. There are three parts of the sample code, including test items for startup, runtime and application. The user can add and modify this code into existing application to meet the requirements from IEC60730-1.

The main purpose of this application note and the associated software is to facilitate and accelerate user software development and certification processes for applications based on M451 series microcontrollers. The Software Test Library (STL) collects common sets of tests dedicated mainly to generic blocks of M451 microcontrollers. The common part of STL package can also be reused for any other microcontrollers of M4 family. The user can include the STL into a final customer project, together with additional product specific tests and settings, for example, ADC and DAC tests.

With this application note, the user can better understand the methodology and techniques used in the STL. The provided examples show how to integrate the STL and the associated firmware in the application. The final implementation and functionality always has to be verified by the certification body at the application level.

2 STL Test Items

The STL consists of several kinds of test items, including CPU register, Program Counter, WDT, Stack, Interrupt, RAM and Flash tests. These tests are executed during startup or runtime stages or both. All these test functions have been released with this application note.

The user can download the STL from https://github.com/opennuvoton and add them into an existing project to implement the self-test feature.

2.1 Test Items for Startup

When the system is powered on, test items can confirm whether the function of the MCU is correct through the test items for startup. When the test is completed, system can execute the application. If the MCU is detected abnormally, it will stop the system and show an error message.

The following table lists the summary of test items for startup.

Test Items	Description	IEC60730-1 Annex H Items
WDT and WWDT Reset Test	Test the reset function of WDT and WWDT.	1.3 Program Counter Test
CPU Register Test	Read and write specific test patterns on CPU registers.	1.1 CPU Register Test
Program Counter Test	Test the branch address location for program counter.	1.3 Program Counter Test
WDT Test	Test WDT time period by Timer0.	1.3 Program Counter Test
Interrupt Test	Test Interrupt count using Systick and Timer0.	2.0 Interrupt Test
RAM Test – March C	Read and write specific test patterns on RAM by March C algorithm.	4.2 RAM Test
RAM Test – March X	Read and write specific test patterns on RAM by March X algorithm.	4.2 RAM Test
Flash Test	Read Flash data and get CRC32 checksum for check content of Flash.	4.1 ROM Test

Table 2-1 Startup Test Items

2.2 Test Items for Runtime

In the application execution, the test Items for runtime detect the MCU function continuously. In order not to affect the efficiency of the application, these test items are part of whole function. It focuses on the key functions of the system. If the MCU is detected abnormally, it will stop the system and show an error message.

The following table lists the summary of test items for runtime.

Test Items	Description	IEC60730-1 Annex H Items
CPU Register Test	Read and write specific test patterns on CPU registers.	1.1 CPU Register Test
Program Counter Test	Test the branch address location for program counter.	1.3 Program Counter Test
Stack Test	Test whether Stack boundary being overflowed or not.	4.2 RAM Test
Interrupt Test	Test Interrupt count using Systick and Timer0.	2.0 Interrupt Test
RAM Test – March X	Read and write specific test patterns on RAM by March X algorithm.	4.2 RAM Test
Flash Test	Read Flash data and get CRC32 checksum for check content of Flash.	4.1 ROM Test

Table 2-2 Runtime Test Items

2.3 Application Test at Runtime

This application test confirms the peripheral functions of the MCU. It provides test program for common peripheral functions. MCU could verify the correctness of peripheral functions while the system is executing the application. If the MCU is detected abnormally, it will stop the system and show an error message.

The following table lists the summary of application tests for runtime.

Test Items	Description	IEC60730-1 Annex H Items
GPIO Test	Test GPIO input and output signal.	7.1 Digital I/O Test
SPI Test	Test SPI Master loop back transfer. It compares the received data with transmitted data.	7.1 Digital I/O Test
PWM Test	Test PWM period and duty by PWM capture function.	7.1 Digital I/O Test
ADC Test	Sample LDO voltage by ADC function. it compares the ADC result with LDO spec.	7.2 Analog I/O Test

Table 2-3 Runtime Application Tests

2.4 Program Flow Chart

The basic structure of control flow in the system level is shown in Figure 2-1.

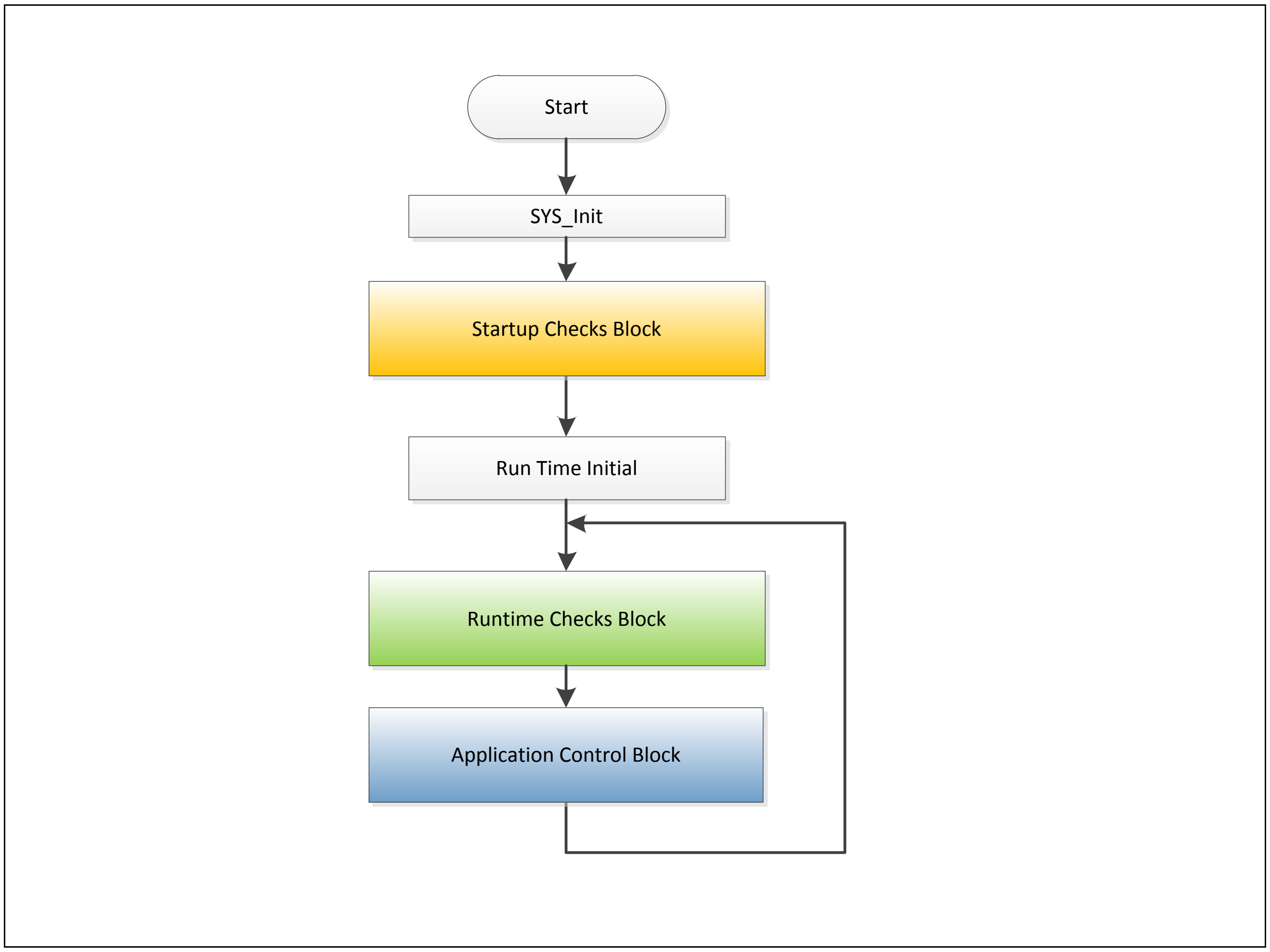

Figure 2-1 Program Flow Chart

There are three parts of the sample code, including startup checks block, runtime checks block and application control block. The user can add and modify these programs into the existing application to meet the requirements from IEC60730-1. The following sections show the details of these three parts.

2.4.1 Startup Checks Block

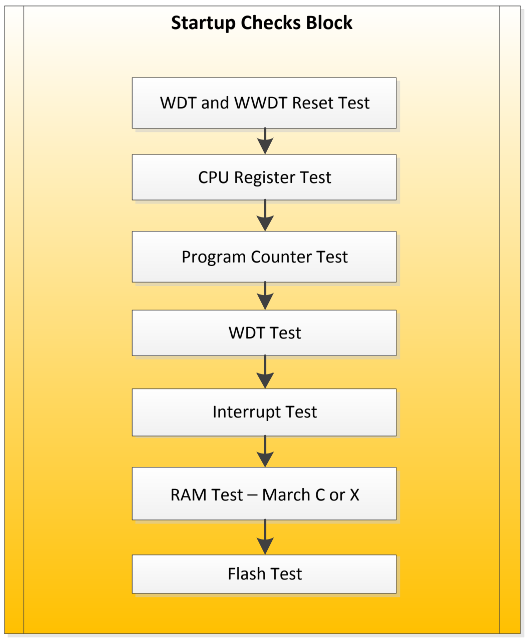

Figure 2-2 Startup Checks Block

When the system is powered on, startup checks block confirms whether the MCU is correct through the test items for MCU startup. After the test is completed, system could execute the application.

2.4.2 Runtime Checks Block

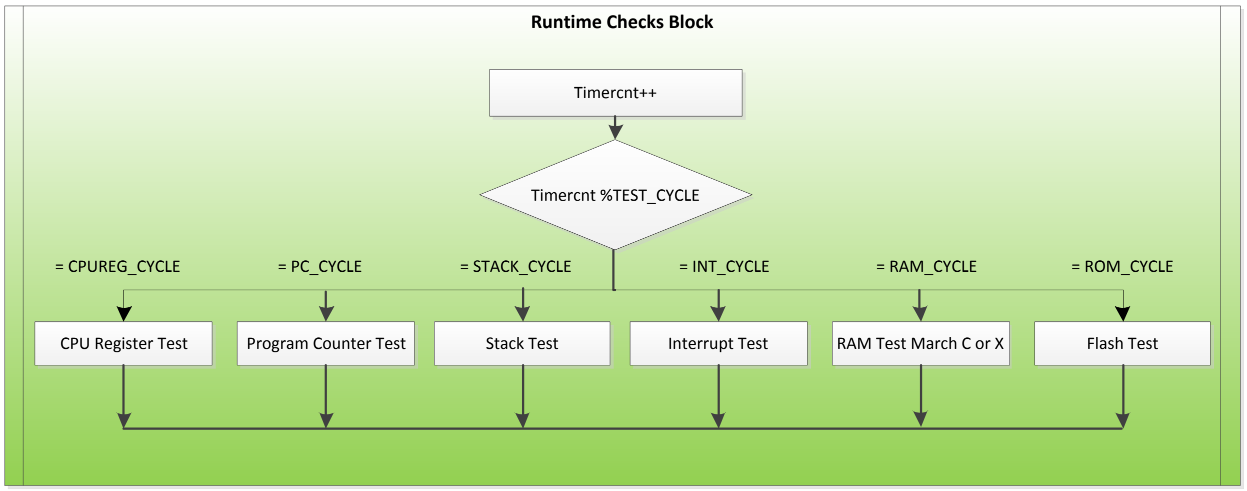

Figure 2-3 Runtime Checks Block

Each time MCU enters the runtime checks block, MCU will check that the value of Timercnt to determine the test item to be executed. The user can set it in the iec60730_test_runtime.h file, and adjust the execution times of each test item. According to this setting keeps the application performance.

2.4.3 Application Control Block

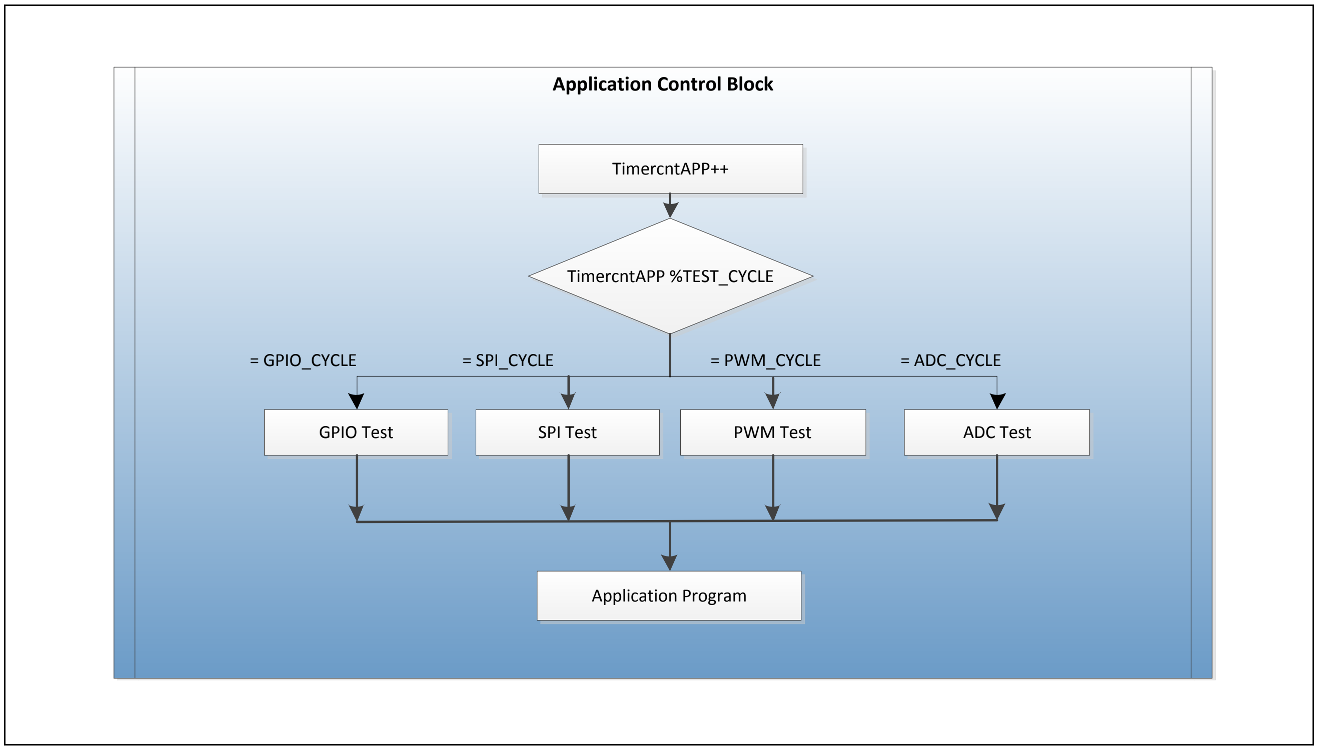

Figure 2-4 Application Control Block

Each time MCU enters the application control block, MCU will check that the value of TimercntAPP to determine the test item to be executed. The user can set it in the iec60730_test_app.h file, and adjust the execution times of each test item. According to this setting keeps the application performance. After the test item has finished, the MCU could run application of the system.

2.5 Program Performance Benchmarking

This section shows the performance of test items, including startup test and runtime test. User could define and normalize test time for system performance to keep CPU resource for application.

- Test Items for Startup

Test Items	Execution time		Peripherals	Description
	Clock: 22 MHz	Clock: 72 MHz		
CPU Register Test	18.66 us	5.76 us		
Program Counter Test	3.64 us	1.10 us		
WDT Test	31.30 ms	31.30 ms	Timer0, WDT	
Interrupt Test	10.44 ms	10.44 ms	Timer0, Systick	
RAM Test – March C	15.86 ms	4.89 ms		Test Size: 32 KB
RAM Test – March X	10.68 ms	3.3 ms		Test Size: 32 KB
Flash Test (SW)	489 ms	151 ms		Test Size: 63 KB
Flash Test (HW)	7.38 ms	2.28 ms	CRC	Test Size: 63 KB

Table 2-4 Performance of Startup Test Items

- Test Items for Runtime

Test Items	Execution time		Peripherals	Description
	Clock: 22 MHz	Clock: 72 MHz		
CPU Register Test	6.03 us	1.69 us		
Program Counter Test	2.03 us	0.63 us		
Stack Test	2.88 us	14.06 us		
Interrupt Test	3.92 us	1.11 us	Timer0, Systick	
RAM Test – March X	25.72 us	8.05 us		Test Size: 32 B
Flash Test (SW)	316 us	96.40 us		Test Size: 32 B
Flash Test (HW)	6.52 us	2.02 us	CRC	Test Size: 32 B

Table 2-5 Performance of Runtime Test Items

3 Function Description of Test Items for Startup

3.1 WDT and WWDT Reset Test

This test item is used to test WDT and WWDT Reset function and check the results.

Format	void IEC60730_WDT_WWDT_Reset_Test (uint32_t* WDTTestPass)
Arguments	uint32_t* WDTTestPass Test result will be stored in this value. 0 = Fail 1 = Pass
Global Values	None
Return Values	0
Related Files	wdt_wwdt_reset_test_startup.s

This test item is to test the reset function of WDT and WWDT individually. The system will reset twice by WDT and WWDT when the system is powered on.

3.2 CPU Registers Test

This test item is used to read and write specific test patterns on CPU registers and check the result.

Format	void **IEC60730_CPU_Reg_Test** (void)
Arguments	None
Global Values	*uint32_t CPUTestPass;* Test result will be stored in this value and main program will check its value to detect pass or fail. 0 = Fail 1 = Pass
Return Values	None
Related Files	cpureg_test_startup.s

This test item uses the test patterns, 0xAAAAAAAA and 0x55555555, to verify the following registers:

- General purpose registers(R0 ~ R12)
- PRIMASK register
- CONTROL register
- SP register
- LR register
- APSR register

If any errors occur, this test will abort immediately and show an error message.

3.3 Program Counter Test

This test item is used to test whether MCU Program Counter can branch to the pre-defined address location or not.

Format	uint32_t **IEC60730_PC_Test** (uint32_t* PCTestPass)
Arguments	*uint32_t* PCTestPass* Test result will be stored in this value. 0 = Fail 1 = Pass
Global Values	None
Return Values	0
Related Files	pc_test_startup.c iec60730.sct

This test requires a scatter file, “iec60730.sct”, to arrange the code layout.

In this file, there are two sections, pc_test_1 and pc_test_2, located in different address.

User can modify this address to fit the test environment.

```
LR_IROM1 0x00000000         {    ; load region
  ER_IROM1 0x00000000 0x00040000  {  ; load address = execution address
*.o (RESET, +First)
*(InRoot$$Sections)
  }
```

```
ABS_ADDRESS1 0x1554 FIXED 36 {
   pc_test_startup.o (pc_test_1)
 }

 ABS_ADDRESS2 0x2AA8 FIXED 36 {
   pc_test_startup.o (pc_test_2)
 }

 ER_IROM2 +0 0x00040000  {  ; load address = execution address
  .ANY (+RO)
```

The pc_test_1 function returns the known value defined by user and pc_test_2 returns its function address. This test will check those two return values to make sure the program counter branches correctly.

3.4 WDT Test

This test item is used to compare Timer0 interrupt times with WDT.

Format	uint32_t **IEC60730_WDT_Test** (uint32_t* WDTTestPass)
Arguments	*uint32_t* WDTTestPass* Test result will be stored in this value. 0 = Fail 1 = Pass
Global Values	*uint32_t u32Timer0Cnt* It is counted by Timer0 interrupt. *uint32_t u32WdtInt* It is invoked once by WDT interrupt.
Return Values	0
Related Files	wdt_test_startup.c

The WDT test uses WDT and Timer0 to check if the WDT can work properly. The WDT is set to use LIRC as clock source. While the WDT interrupt is invoked, the interrupt count of Timer0 is checked whether it is located in the range of deviation, and then determines whether the test is passed or failed.

3.5 Interrupt Test

This test item is used to compare Timer0 interrupt times with Systick Timer.

Format	uint32_t **IEC60730_Interrupt_Test** (uint32_t* IntTestPass)
Arguments	*uint32_t* IntTestPass* Test result will be stored in this value. 0 = Fail 1 = Pass
Global Values	*uint32_t u32Timer0Cnt* It is counted by Timer0 interrupt. *uint32_t u32SysTickCnt* It is counted by Systick Timer interrupt.
Return Values	0
Related Files	interrupt_test_startup.c

The interrupt test uses Systick and Timer0 that have different clock sources. By comparing the invoked count by different timers, users can determine whether the test is passed or failed.

3.6 RAM Test – March C

The RAM memory is tested by March C algorithm.

Format	void **IEC60730_RamMarchC** (void)
Arguments	None
Global Values	*uint32_t RAMStartAddr* The starting address of RAM. It is set before user call the function. *uint32_t RAMEndAddr* The end address of RAM. It is set before user call the function. *uint32_t RAMTestPass* Test result will be stored in this value and main program will check its value to detect pass or fail.

	0 = Fail 1 = Pass
Return Values	None
Related Files	ram_marchc_test_startup.c

This test is enabled by the defined RAM_STARTUP_MARCHC_WOM of iec60730_test.h file. The data in the tested area will be lost after test. The user can back up the data before test and then restore it after test. March X is a subset of March C. The user can select one of these two algorithms to test the RAM.

The test procedure of March C algorithm:

1. Clear all memory content to be zero.
2. Scan memory content from low to high address. For each memory location, check the read bit is 0 or not. If not, the test is failed otherwise write 1 back to the memory location.
3. Scan memory content from low to high address. For each memory location, check the read bit is 1 or not. If not, the test is failed otherwise write 0 back to the memory location
4. Scan memory content from high to low address. For each memory location, check the read bit is 0 or not. If not, the test is failed otherwise write 1 back to the memory location
5. Scan memory content from high to low address. For each memory location, check the read bit is 1 or not. If not, the test is failed otherwise write 0 back to the memory location
6. Check whether all bits are zero or not. The scanning direction can be either from high to low, or low to high address.

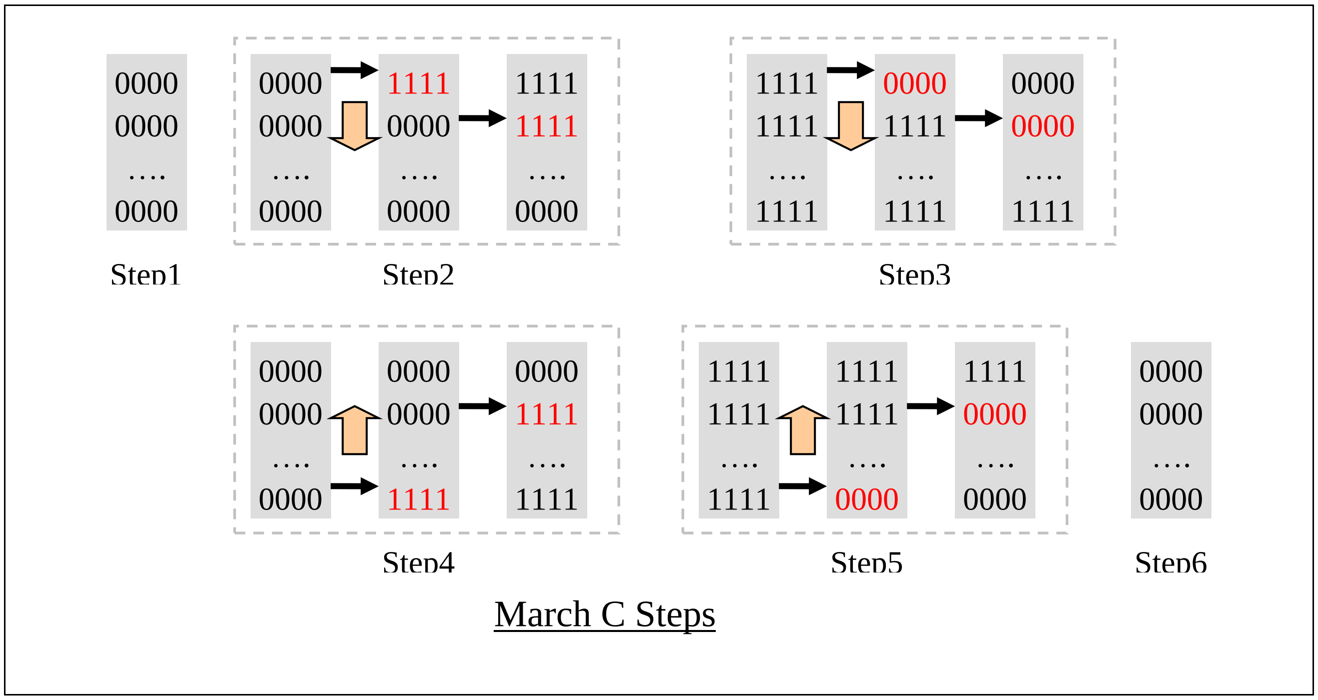

Figure 3-1 March C Algorithm

3.7 RAM Test – March X

The RAM memory is tested by March X algorithm.

Format	void **IEC60730_RamMarchX** (void)
Arguments	None
Global Values	*uint32_t RAMStartAddr* The starting address of RAM. It is set before user call the function. *uint32_t RAMEndAddr* The end address of RAM. It is set before user call the function. *uint32_t RAMTestPass* Test result will be stored in this value and main program will check its value to detect pass or fail. 0 = Fail 1 = Pass
Return Values	None
Related Files	ram_marchx_test_startup.c

This test is enabled by the define RAM_STARTUP_MARCHX_WOM of iec60730_test.h file. The data in the tested area will be lost after test. The user can back up the data before test and then restore it after test.

The test procedure of March X algorithm:

1. Clear all memory content to be zero.
2. Scan memory content from low to high address. For each memory location, check the read bit is 0 or not. If not, the test is failed otherwise write 1 back to the memory location.
3. Scan memory content from high to low address. For each memory location, check the read bit is 1 or not. If not, the test is failed otherwise write 0 back to the memory location
4. Check whether all bits are zero or not. The scanning direction can be either from high to low, or low to high address.

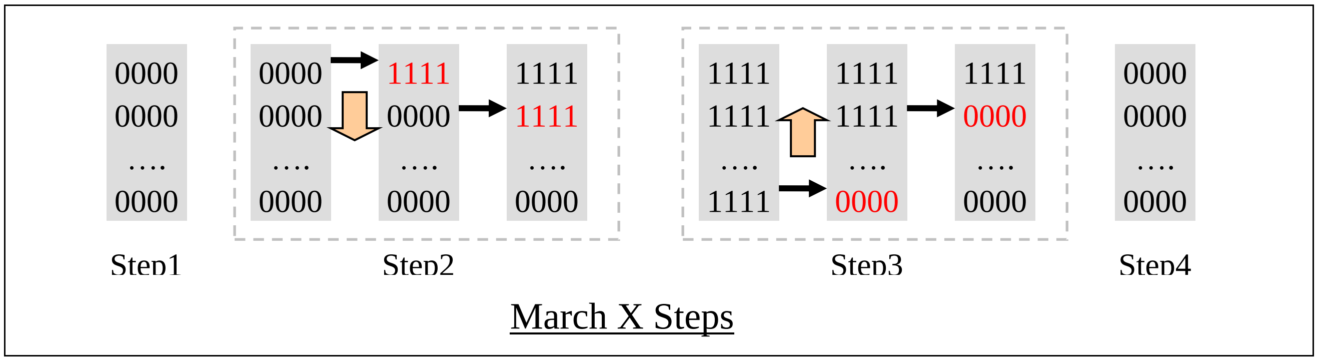

Figure 3-2 March X Algorithm

3.8 Flash Test

The Flash memory is tested by checking CRC checksum value.

Format	uint32_t **IEC60730_Flash_Test** (uint32_t* ROMTestPass)
Arguments	*uint32_t* ROMTestPass;* Test result will be stored in this value and main program will check its value to detect pass or fail. 0 = Fail 1 = Pass
Global Values	None
Return Values	0
Related Files	flash_test_startup.c

The test function calculates the CRC32 checksum for the Flash program area, and then compares the checksum with prior calculated value __Check_Sum in startup_M451Series.s file to determine whether the test is passed or not. The MCU can get CRC32 checksum by hardware interface if the user defines CRC_COMAPRE_WITH_HW in iec60730_test.h file.

This test requires a scatter file, “iec60730.sct”, to arrange the code layout. In this file, there is an absolute address to define checksum value. The Flash program area is from address 0 to this checksum value address (0x10000-0x100).

User can modify this address to fit the test environment.

```
ABS_ADDRESS3 0x10000-0x100 FIXED 36 {
    *.o (CHECKSUM +Last)
}
```

4 Function Description of Test Items for Runtime

4.1 CPU Registers Test

This test item is used to read and write specific test patterns on CPU registers and check the results.

Format	void **IEC60730_CPU_Reg_Test_RunTime** (void)
Arguments	None
Global Values	*uint32_t CPUTestPass* Test result will be stored in this value. 0 = Fail 1 = Pass
Return Values	None
Related Files	cpureg_test_runtime.s

This test item uses the test patterns, 0xAAAAAAAA and 0x55555555, to verify the following registers:

- General purpose registers(R0 ~ R7)

If any errors occur, this test will abort immediately and show an error message.

4.2 Program Counter Test

This test item is used to test whether MCU Program Counter can branch to the pre-defined address location or not.

Format	uint32_t **IEC60730_PC_Test_RunTime** (uint32_t* PCTestPass)
Arguments	*uint32_t* PCTestPass* Test result will be stored in this value. 0 = Fail 1 = Pass
Global Values	None

Return Values	0
Related Files	pc_test_runtime.c

This test is the same with **IEC60730_PC_Test** (uint32_t* PCTestPass) function. The pc_test_1 function returns the known value defined by user and pc_test_2 returns its function address. This test will check those two return values to make sure the program counter branches correctly.

4.3 Stack Test

This test item is used to test whether Stack boundary is overflowed or not.

Format	uint32_t **IEC60730_Stack_Test_RunTime** (uint32_t* StackTestPass)
Arguments	*uint32_t* StackTestPass* Test result will be stored in this value. 0 = Fail 1 = Pass
Global Values	*STACK_TEST_PTRN[]* It stores the pre-defined pattern.
Return Values	0
Related Files	stack_test_runtime.c, stack_overrun_ptrn.c

The Stack test checks whether the specific pattern in the beginning of stack area is destroyed or not. If the pattern has not been changed, it means the Stack overrun condition never met.

4.4 Interrupt Test

This test item is used to compare Timer0 interrupt times with Systick Timer.

Format	uint32_t **IEC60730_Interrupt_Test_Runtime** (uint32_t* IntTestPass)
Arguments	*uint32_t* IntTestPass* Test result will be stored in this value.

	0 = Fail 1 = Pass
Global Values	*uint32_t u32Timer0Cnt* It is counted by Timer0 interrupt. *uint32_t u32SysTickCnt* It is counted by Systick Timer interrupt.
Return Values	0
Related Files	interrupt_test_runtime.c

The interrupt test uses Systick and Timer0 that have different clock sources. By comparing the invoked count by different timers, the user can determine whether the test is passed or failed.

4.5 RAM Test – March X

The RAM memory is tested by March X algorithm.

Format	uint32_t **IEC60730_RAM_MarchX_Test_RunTime** (uint32_t* RAMTestPass)
Arguments	*uint32_t* RAMTestPass;* Test result will be stored in this value and main program will check its value to detect pass or fail. 0 = Fail 1 = Pass

Global Values	*uint32_t RAMStartAddr;*
	The starting address of RAM. It is set before user call the function.
	uint32_t RAMEndAddr;
	The end address of RAM. It is set before user call the function.

Return Values	0
Related Files	ram_marchcx_test_runtime.c

This test is enabled by the defined RAM_RUNTIME_MARCHX_WOM of iec60730_test_runtime.h file. All RAM area excluding stack and stack pattern will be tested. After tested, all data of verified area can be kept no change.

4.6 Flash Test

The Flash memory is tested by checking CRC checksum value.

Format	uint32_t **IEC60730_Flash_Test_RunTime** (uint32_t* ROMTestPass)
Arguments	*uint32_t* ROMTestPass;* Test result will be stored in this value and main program will check its value to detect pass or fail. 0 = Fail 1 = Pass
Global Values	None
Return Values	0
Related Files	flash_test_runtime.c

The test function calculates the CRC32 checksum for the flash area specified by ROM_START and ROM_LENGTH defined in iec60730_test_runtime.h file. Then, it compares the checksum with prior calculated value __Check_Sum in the startup_M451Series.s file to determine whether the test is passed or not. The MCU can get CRC32 checksum by hardware interface if the user defines CRC_COMAPRE_WITH_HW in the iec60730_test.h file.

5 Function Description of Application Test at Runtime

5.1 GPIO Test

GPIO is tested by three GPIO pin: PB.5, PB.6 and PB.7.

Format	uint8_t **IEC60730_IO_Test** (void);
Arguments	None
Global Values	None
Return Values	0 = Fail 1 = Pass
Related Files	io_test.c

The test function has two parts. The first part which needs to connect PB.6 with PB.7. PB.6 will output a signal to PB.7. Then it compares the signal to check the I/O output and input function.

As to the second part, the MCU checks the I/O status by register directly. The PIN register will show the actual level of PB.5.

5.2 SPI Test

SPI is tested by SPI0_MISO and SPI0_MOSI.

Format	uint8_t **IEC60730_SPI_Test** (void);
Arguments	None
Global Values	None
Return Values	0 = Fail 1 = Pass
Related Files	SPI_test.c

The SPI test needs to connect SPI0_MISO with SPI0_MOSI. SPI0 will transfer 256 bytes data and get feedback data. If the data is the same, the SPI test is passed.

5.3 PWM Test

PWM is tested by PWM1_CH0 and PWM1_CH2

Format	uint8_t **IEC60730_PWM_Test** (void);
Arguments	None
Global Values	None
Return Values	0 = Fail 1 = Pass
Related Files	PWM_test.c

The PWM test needs to connect PWM1_CH0 with PWM1_CH2. It will set PWM1_CH0 to output a 10 kHz waveform and set PWM1_CH2 as capture mode. It calculates the capture data to get the waveform period. If the captured period is 10 kHz, the PWM test is passed.

5.4 ADC Test

ADC is tested by ADC_CH1 with V_{LDO}.

Format	uint8_t **IEC60730_AD_Test** (void);
Arguments	None.
Global Values	uint8_t u8ConvertTimes – ADC sample times. uint8_t u8ADCStop – Flag to indicate whether ADC sample is finished or not. uint32_t u32ConvertResVgap – the ADC result of V_{BG} uint32_t u32ConvertResVldo – the ADC result of V_{LDO}
Return Values	0 = Fail 1 = Pass
Related Files	ad_test.c

The ADC test calculates the V_{REF} value by V_{BG} value. Then it can use the V_{REF} to calculate the voltage of the MCU V_{LDO} pin by connect it with ADC_CH1. If the difference of these V_{LDO} value is larger than 1.98V or smaller than 1.62V, the test is failed.

6 STL Evaluation Board

6.1 STL Evaluation Board Overview

The Nuvoton STL evaluation board is based on the Cortex®-M4 M452RE6AE microcontroller. It implements a test setting interface to enable test items on the board. In addition, it has an external interface included with 6-ch PWM, 2-ch ADC and 2-ch ACMP. It could be used to connect external application board. User can develop their own application board to extend the test items by themselves.

The Nuvoton STL evaluation board is shown in Figure 6-1.

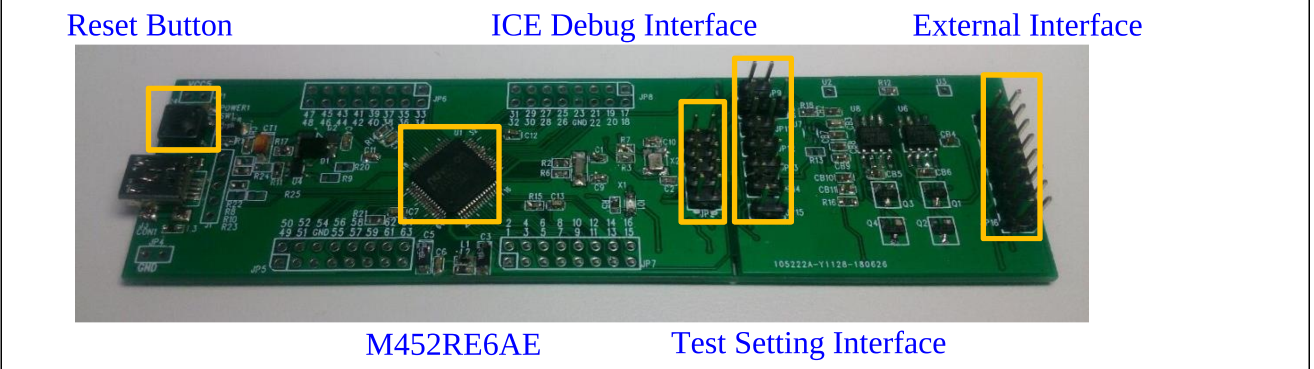

Figure 6-1 Nuvoton STL Evaluation Board

6.2 Test Setting Table

The STL EVB is set by these connectors. The connectors need to be short or open before the MCU executes the test item. The GPIO test setting is different with the SPI test so it can only be enabled individually.

Test Items	JP9	JP11	JP12	JP13	JP15
ADC Test	Short(1,2)				
PWM Test		Short(1,2)			
GPIO Test			Short(1,2)	Open(1,2)	
SPI Test				Short(1,2)	

Table 6-1 Test Setting Table

6.3 Schematics

6.3.1 MCU Schematic

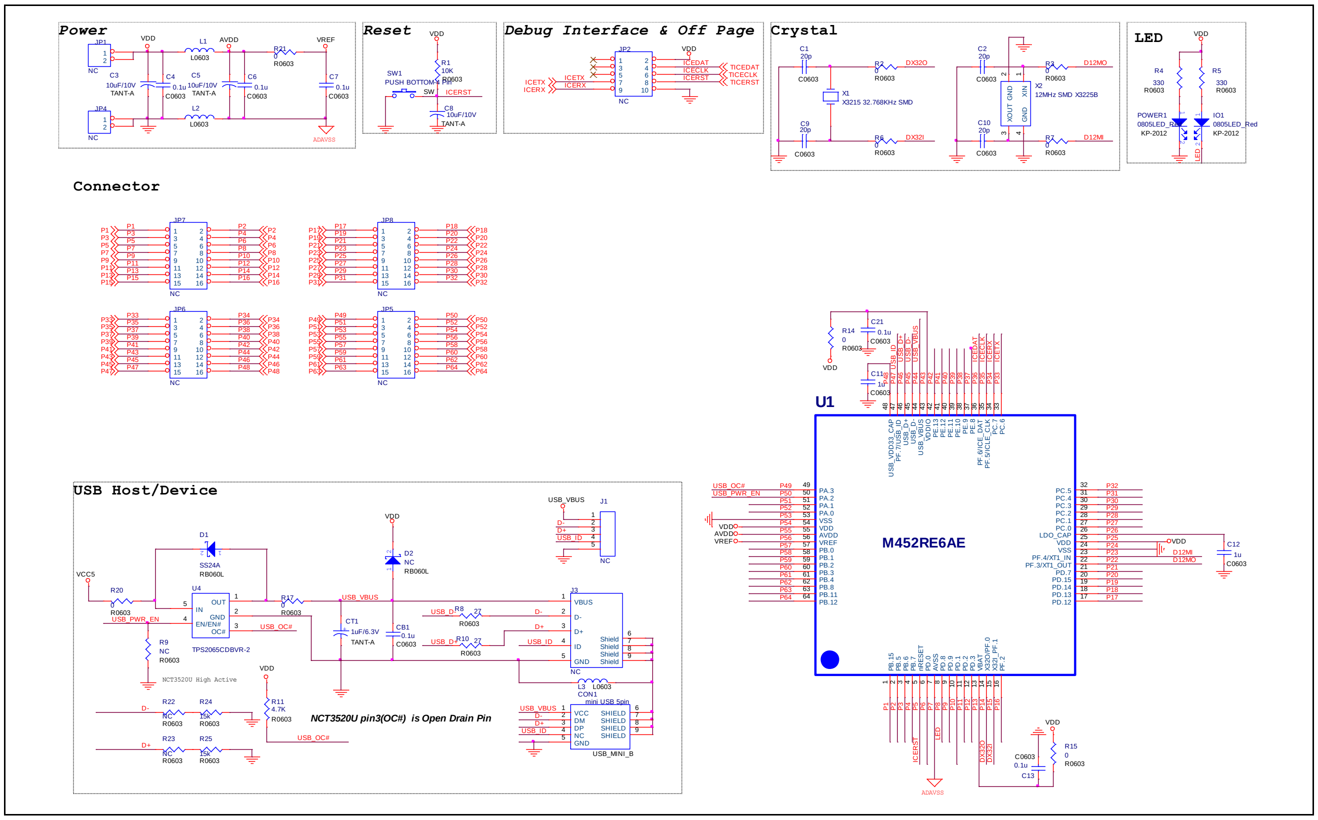

Figure 6-2 MCU Schematic

6.3.2 Application Schematic

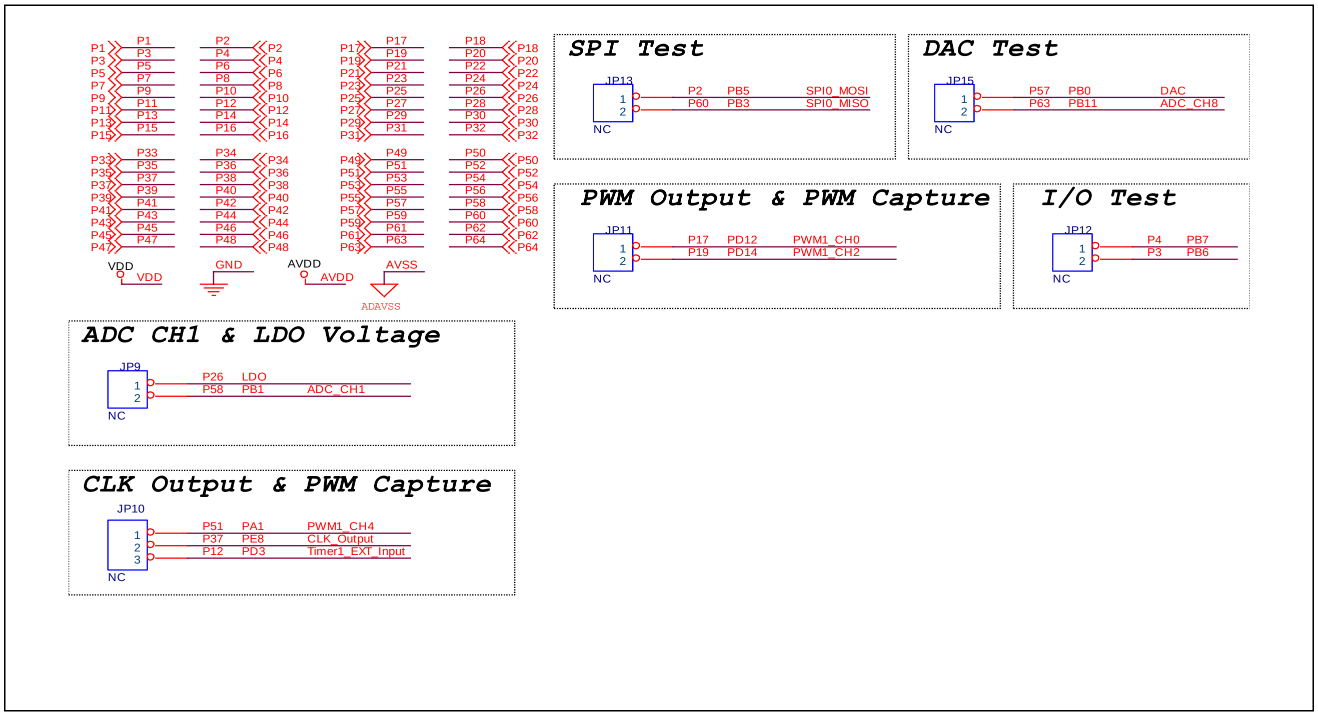

Figure 6-3 Application Schematic

6.3.3 External Interface

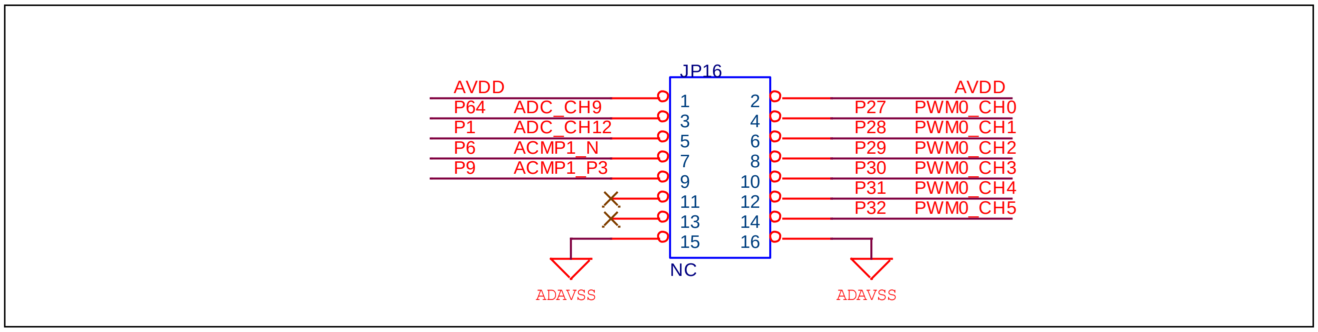

Figure 6-4 External Interface

This external interface is included with 6-ch PWM, 2-ch ADC and 2-ch ACMP. It could be used to connect external application board. User can develop their own application board to extend the test items by themselves.

7 Conclusion

When home appliances are under development, home appliances should follow the requirements specified in the IEC60730-1 standard for safety. Nuvoton provide an IEC60730-1 class B Software Test Library (STL) to implement MCU basic requirements specified in Annex H MCU part of the IEC60730-1 standard. It consists of several kinds of test items, including CPU register, Program Counter, WDT, Stack, Interrupt, RAM and Flash tests to detect the system fault/error. User can implement safety function based on the fault/error event to protect the system from dangers like fire hazard or motor rotor fault.

This STL collects common sets of tests dedicated mainly to generic blocks of M4 microcontroller family. User could take this application note to know the test items. According to the description of these test items, the user can include the STL package into a final project. In addition, the STL can be modified easily based on the application function of the product to achieve rapid development flow for IEC60730-1.

Finally, the user can implement a reliability and safety system compliant with IEC60730-1 safety standards.

Revision History

Date	Revision	Description
2019.09.30	1.00	1. Initially issued.