AN1229

Class B Safety Software Library for PIC® MCUs and dsPIC® DSCs

Author: *Veena Kudva*
Microchip Technology Inc.

INTRODUCTION

This application note describes the Class B Safety Software Library routines that detect the occurrence of Faults in a single channel CPU. These routines have been developed in accordance with the IEC 60730 standard to support the Class B certification process. These routines can be directly integrated with the end user's application to test and verify the critical functionalities of a controller without affecting the end user's application.

This application note also describes the Application Programming Interface (API) functions that are available in the Class B Safety Software Library.

The Class B safety software routines can be called periodically at start-up or run time to test the following components:

- CPU Registers
- CPU Program Counter
- Invariable Memory
- Variable Memory
- Clock
- Interrupt Handling and Execution

This application note also outlines various techniques, which are not part of the Class B Safety Software Library, to test components such as external communication, timing, I/O periphery, analog I/O and analog multiplexer.

Note: The term 'IEC 60730 standard' used in this document refers to the *"IEC 60730-1 ed.3.2"* Copyright © 2007 IEC, Geneva, Switzerland. www.iec.ch.

OVERVIEW OF THE IEC 60730 STANDARD

Note: *"The author thanks the International Electrotechnical Commission (IEC) for permission to reproduce information from its International Standard IEC 60730-1ed.3.2 (2007). All such extracts are copyright of IEC, Geneva, Switzerland. All rights reserved. Further information on the IEC is available from www.iec.ch. IEC has no responsibility for the placement and context in which the extracts and contents are reproduced by the author, nor is IEC in any way responsible for the other content or accuracy therein."*

The IEC 60730 standard defines the test and diagnostic methods that ensure the safe operation of the controlled equipment used in household appliances. Annex H of the IEC 60730 standard classifies the software into the following categories (see **Appendix B: "IEC 60730-1 Table H.11.12.7"**):

- Class A
- Class B
- Class C

The Class B Safety Software Library implements the important test and diagnostic methods that fall into the Class B category. These methods use various measures to detect and respond to the software-related Faults and errors.

According to the IEC 60730 standard, the controls with functions that fall into the Class B category should have one of the following structures:

- Single Channel with Functional Test

  In this structure, the Functional test is executed prior to the application firmware execution.

- Single Channel with Periodic Self-Test

  In this structure, the Periodic tests are embedded within the firmware, and the self-test occurs periodically while the firmware is in Execution mode.

- Dual Channel without Comparison

  In this structure, two independent methods execute the specified operations.

SYSTEM REQUIREMENTS

The following system requirements are recommended to run the Class B Safety Software Library:

- For the tests that require the independent time slot monitoring, the system hardware must be provided with at least two independent clock sources (e.g., crystal oscillator and line frequency).
- The user application determines whether the interrupts need to be enabled or disabled during the execution of the Class B Safety Software Library.

  If an interrupt occurs during the execution of the Class B Safety Software Library routine, an unexpected change may occur in any of the registers. Therefore, when the Interrupt Service Routine (ISR) executes, the contents of the register will not match the expected content, and the ISR will return an incorrect result.

CLASS B SAFETY SOFTWARE LIBRARY

The Class B Safety Software Library includes several APIs, which are intended to maximize application reliability through Fault detection. These APIs help meet the IEC 60730 standard compliance. The following tests can be implemented using this library:

- CPU Register Test
- Program Counter Test
- Variable Memory Test
- Invariable Memory (Flash/EEPROM) Test
- Interrupt Test
- Clock Test

In the following sections, the test description and the implementation details are discussed for each test. In addition, each section also lists the APIs that are required to execute the corresponding test.

CPU Register Test

The CPU Register test implements the functional test H.2.16.5 defined by the IEC 60730 standard. It detects stuck-at Faults in the CPU registers. This ensures that the bits in the registers are not stuck at a value '0' or '1'; this is a non-destructive test.

This test performs the following major tasks:

1. The contents of the CPU registers to be tested are saved on the stack before executing the routine.
2. The registers are tested by first successively writing the values 0x5555 and 0xAAAA into the registers, and then reading the values from these registers for verification.
3. The test returns an error code if the returned values do not match.
4. The contents of the register (W0) that return the error code are not preserved. The contents of all other CPU registers are restored upon the completion of the test.

Note:	The interrupts should be disabled during the execution of the CPU Register test so that the data integrity is preserved at all times.

API FUNCTIONS

The following API implements the CPU Register test:

`SSL_16bitsFamily_CPU_RegisterTest`

Program Counter Test

The Program Counter test implements the functional test H.2.16.5 defined by the IEC 60730 standard. The PC holds the address of the next instruction to be executed.

The test performs the following major tasks:

1. The Program Counter test invokes the functions that are located in the Flash memory at different addresses.
2. These functions return a unique value.
3. The returned value is verified using the PC test function.
4. If the values match, the PC has branched to the correct location.

Note 1:	The user application defines the address where the PC branches.
2:	The size of the program memory varies by device. Refer to the specific device data sheet for more details.

The customized linker script defines the addresses where these functions reside in the Flash memory. The functions placed at these addresses return a unique value, which is the starting address of the called function. Example 1 shows how to modify the linker script to place a function in the Flash memory. In this example, the `SSL_TestFunction1` function is placed at address 0x900 in the `SSLTestSection1` section by modifying the linker script.

API FUNCTIONS

The following API implements the Program Counter test:

`SSL_16bitsFamily_PCtest`

Invariable Memory (Flash/EEPROM) Test

The Invariable Memory (Flash/EEPROM) test implements the periodic modified checksum H.2.19.3.1 defined by the IEC 60730 standard. It detects the single bit Faults in the invariable memory. The invariable memory in a system, such as Flash and EEPROM memory, contains data that is not intended to vary during the program execution. The Flash/EEPROM Invariable Memory test computes the periodic checksum using the Cyclic Redundancy Check (CRC). Several standards are used today for the CRC calculation. The characteristics of the CRC divisor vary from 8 to 32 bits depending on the polynomial that is used. The width of a divisor determines its ability to detect the errors. Some commonly used CRC divisors are as follows:

- **CRC-16** = `1 1000 0000 0000 0101` = 8005 (hex)
- **CRC-CCITT** = `1 0001 0000 0010 0001` = 1021 (hex)
- **CRC-32** = `1 0000 0100 1100 0001 0001 1101 1011 0111` = 04C11DB7 (hex)

Figure 1 shows the flowchart for the Invariable Memory test.

The `SSL_16bitsFamily_Flashtest_CRC16` function returns the final CRC value that can be used to perform the following:

1. At the system start-up, the computed CRC checksum can be used as a reference checksum if the `CRC_Flag` is set to 0x00.
2. The reference checksum is stored in the Flash or EEPROM memory and the CRC flag is set to 0xFF.
3. The `SSL_16bitsFamily_Flashtest_CRC16` function can be called periodically if the CRC flag is set to 0xFF.
4. The checksum calculated from step 3 is compared with the reference checksum.
5. If both values match, a status bit can be set by the user application to indicate that the invariable memory has passed the test and no errors were found.

API FUNCTIONS

The following APIs implement the Invariable Memory test:

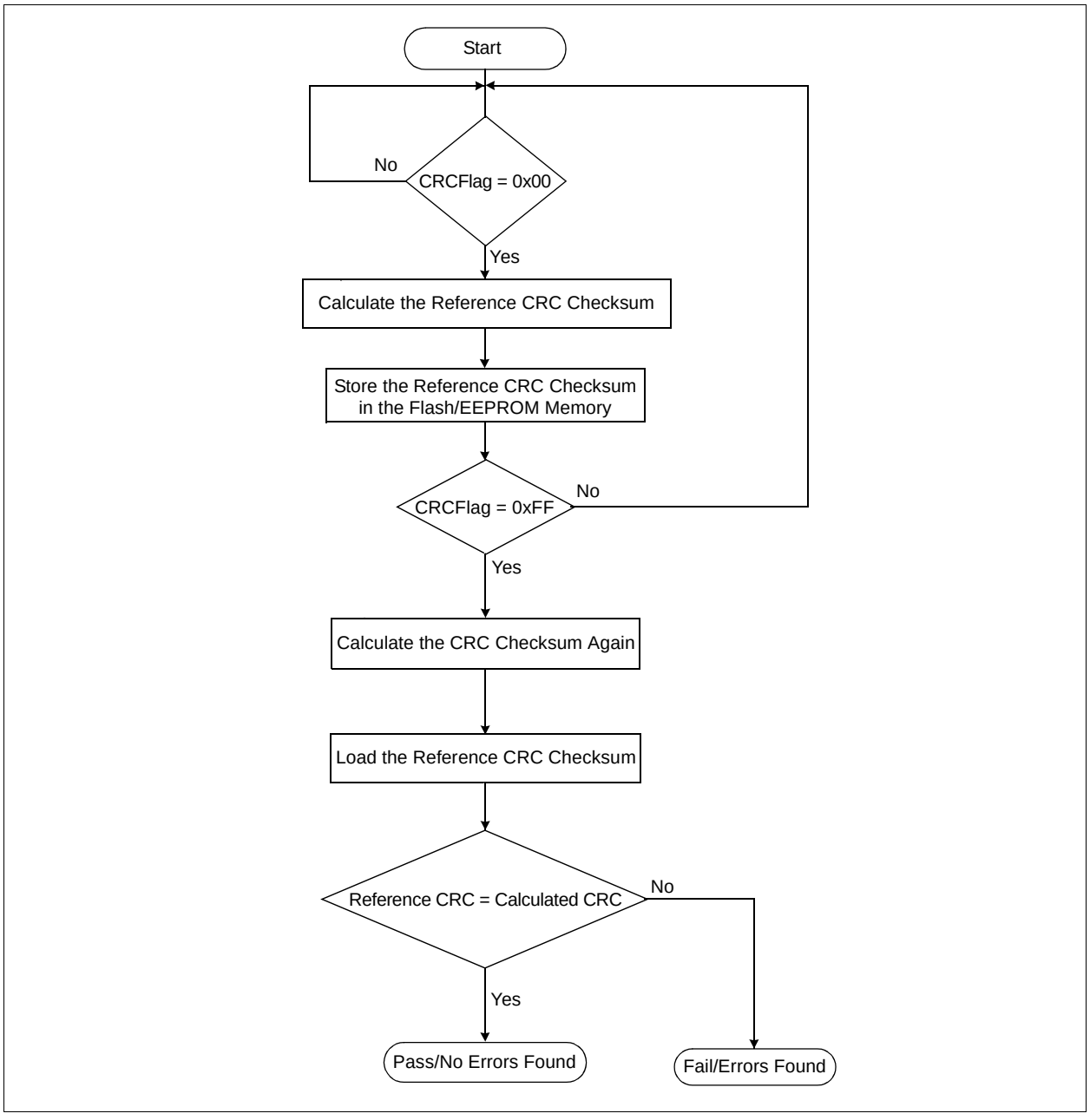
- `SSL_16bitsFamily_Flashtest_CRC16`
- `SSL_16bitsFamily_EEPROMtest_CRC16`

Note:	The EEPROM test applies only to dsPIC30F devices.

EXAMPLE 1: LINKER SCRIPT MODIFICATION

```
/* The modified linker script */
SslTestSection1 0x900:
{
  *(.SslTestSection1);
} program
/*The SSL_TestFunction1 function*/
long __attribute__((__section__(".SslTestSection1"))) SSL_TestFunction1()
{
  return((long)&SSL_TestFunction1);
}
```

FIGURE 1: FLOWCHART FOR THE INVARIABLE MEMORY TEST



Variable Memory Test

The Variable Memory test implements the Periodic Static Memory test H.2.19.6 defined by the IEC 60730 standard. It detects single bit Faults in variable memory. The variable memory contains data, which is intended to vary during program execution. The RAM Memory test is used to determine if any bit of the RAM memory is stuck at '1' or '0'. The March Memory test and Checkerboard test are some of the widely used static memory algorithms for checking the DC Faults.

The following tests can be implemented using the Class B Safety Software Library:

- March Test
  - March C Test
  - March C Minus Test
  - March B Test

MARCH TEST

A March test performs a finite set of operations on every memory cell in a memory array. Each operation performs the following tasks:

1. Writes '0' to a memory cell (w0).
2. Writes '1' to a memory cell (w1).
3. Reads the expected value '0' from a memory cell (r0).
4. Reads the expected value '1' from a memory cell (r1).

March Test Notations

Example 2 illustrates the notations that are used in the March test:

EXAMPLE 2: MARCH TEST NOTATIONS

- ⇑: Arranges the address sequence in ascending order.
- ⇓: Arranges the address sequence in descending order.
- ⇕: Arranges the address sequence in either ascending or descending order.
- *r*0: Indicates a read operation (reads '0' from a memory cell).
- *r*1: Indicates a read operation (reads '1' from a memory cell).
- *w*0: Indicates a write operation (writes '0' to a memory cell).
- *w*1: Indicates a write operation (writes '1' to a memory cell).

Note: Except for the "SSL_16bitsFamily_RAM_STACKtest_MarchC" function, remaining March memory test functions do not test the Stack area of the RAM.

MARCH C TEST

The March C test is used to detect the following types of Fault in the variable memory:

- Stuck-at Fault
- Addressing Fault
- Transition Fault
- Coupling Fault

The complexity of this test is 11n, where n indicates the number of bits in the memory. This test is a destructive test (i.e., memory contents are not preserved). Therefore, it is designed to run at the system start-up before initializing the memory and the run-time libraries.

EXAMPLE 3: MARCH C ALGORITHM

```
MarchC
{
  ⇕(w0); ⇑(r0, w1); ⇑(r1, w0);

  ⇕(r0); ⇓(r0, w1); ⇓(r1, w0); ⇑(r0)
}
```

Example 4 shows the pseudocode that demonstrates the implementation of the March C test.

API FUNCTIONS

The following APIs implement the March C test:

- `SSL_16bitsFamily_RAMtest_MarchC`
- `SSL_16bitsFamily_RAM_STACKtest_MarchC`

EXAMPLE 4: PSEUDOCODE FOR MARCH C TEST

```
for(i=0;i<=(n-1);i++)
    x(i)=0;                         /*write background to zero*/

for(i=0;i<=(n-1);i++)
{
    if (x(i)==0)
        x(i) =1;
    else
        return fail;
}
for(i=0;i<=(n-1);i++)
{
    if(x(i)==1)
        x(i)=0;
    else
        return fail;
}
for(i=(n-1);i>=0;i--)
{
    if(x(i)==0)
        x(i)=1;
    else
        return fail;
}
for(i=(n-1);i>=0;i--)
{
    if(x(i)==1)
        x(i)=0;
    else
        return fail;
for(i=(n-1);i>=0;i--)
{
    if(x(i)==0) {}
    else
        return fail
}
return pass;
```

MARCH C MINUS TEST

The March C Minus test performs a finite set of operations on every memory location in RAM. The memory locations are first tested in ascending order and then in descending order.

When the memory locations are tested in ascending order, each operation performs the following tasks on every memory location:

1. Writes 0xAAAA to the memory location.
2. Reads 0xAAAA from the memory location.
3. Writes 0x5555 to the memory location.
4. Reads 0x5555 from the memory location.
5. Writes 0xAAAA to the memory location

When the memory locations are tested in descending order, each operation performs the following tasks on every memory location:

1. Reads 0xAAAA from the memory location.
2. Writes 0x5555 to the memory location.
3. Reads 0x5555 from the memory location.
4. Writes 0xAAAA to the memory location.

This test is a destructive test. Therefore, it is designed to run at the system start-up before initializing the memory and the run-time libraries.

Example 5 shows the pseudocode that demonstrates the implementation of the March C Minus test.

API FUNCTIONS

The following API implements the March C Minus test:

`SSL_16bitsFamily_RAMtest_MarchC_Minus`

EXAMPLE 5: PSEUDOCODE FOR MARCH C MINUS TEST

```
for(ptr=ramStartAddress; ptr<ramEndAddress; ptr++)
ptr = 0xAAAA;
for(ptr=ramStartAddress; ptr<ramEndAddress; ptr++)
{
    tempData=*ptr;                                      //read 0xAAAA
    if(tempData!=0xAAAA)                                //Check if 0xAAAA
        return TEST_FAIL;
    else
        *ptr=0x5555;                                    //write 0X5555
}
for(ptr=ramStartAddress; ptr<ramEndAddress; ptr++)
{
    tempData=*ptr;                                      //read 0x5555
    if(tempData!=0x5555)                                //Check if 0x5555
        return  TEST_FAIL;
    else
        *ptr=0xAAAA;                                    //write 0AAAA
}
for(ptr= ramEndAddress ptr>=(ramStartAddress);ptr--)
{
    tempData=*ptr;                                      //read 0xAAAA
    if(tempData!=0xAAAA)                                //Check if 0xAAAA
        return TEST_FAIL;
    else
        *ptr=0x5555;                                    //write 0x5555
}
for(ptr=ramEndAddress ptr>=(ramStartAddress);ptr--)
{
    tempData=*ptr;                                      //read 0x5555
    if(tempData!=0x5555)                                //Check if 0x5555
        return  TEST_FAIL;
    else
        *ptr=0xAAAA;                                    //write 0xAAAA
}
return TEST_PASS;
```

MARCH B TEST

The March B is a non-redundant test that can detect the following types of Fault:

- Stuck-at
- Linked Idempotent Coupling
- Inversion Coupling

This test is of complexity 17n, where n indicates the number of bits in the memory.

EXAMPLE 6: MARCH B ALGORITHM

```
MarchB
{
  ⇕(w0);⇑(r0,w1, r1, w0,r0,w1);⇑(r1, w0,w1 );

  ⇓(r1, w0,w1, w0);⇓(r0,w1,w0);
}
```

Example 7 shows the pseudocode that demonstrates the implementation of the March B test.

API FUNCTIONS

The following API implements the March B test:

`SSL_16bitsFamily_RAMtest_MarchB`

Note 1: The user application should allocate 0x50 bytes for the stack before executing any of the March tests. The stack must be allocated at an appropriate address so that it does not get overwritten during the test execution.

2: It is recommended that the stack should be placed at the beginning or at the end of the data memory. The user application should specify an address such that it does not overlap other statically allocated resources (e.g., the MPLAB® ICD 2 RAM space which starts from the address, 0x800).

3: The following changes are made to the `.gld` file before executing the March B or March C test:

```
.stack 0x850: /*Stack Starting
Address\*
{
  __SP_init = .;
  . += 0x50; /* Stack length*/
  __SPLIM_init = .;
  . += 8;
 } >data
```

CHECKERBOARD RAM TEST

The Checkerboard RAM test writes the checkerboard pattern to a memory location, ‘N’, and the inverted pattern to the memory location, ‘N+1’. This is a non-destructive memory test.

This test performs the following major tasks:

1. Saves the contents of the memory locations to be tested in the CPU registers.
2. Writes the value, 0xAAAA, to the memory location, ‘N’, and the inverted value, 0x5555, to the memory location, ‘N+1’.
3. Reads the memory locations (‘N’ and ‘N+1’) and verifies its contents. If the values match, the function returns ‘1’; otherwise it returns ‘0’.
4. Step 2 and 3 are repeated by writing the inverted pattern to the same locations.

API FUNCTIONS

The following API implements the Checkerboard RAM test:

`SSL_16bitsFamily_RAMtest_CheckerBoard`

EXAMPLE 7: PSEUDOCODE FOR MARCH B TEST

```
for(i=0;i<=(n-1);i++)
    x(i)=0;                /*write background to zero*/
for(i=0;i<=(n-1);i++)
{
    if(x(i)=0)
        x(i)=1;
    else
        return fail;
    if(x(i)==1)
        x(i)=0;
    else
        return fail;
    if(x(i)==0)
        x(i)=1;
    else
        return fail;
}
for(i=0;i<=(n-1);i++)
{
    if(x(i)==1)
    {
        x(i)=0;
        x(i)=1;
    }
    else
        return fail;
}
for(i=(n-1);i>=0;i--)
 {
    if(x(i)=1)
    {
        x(i)=0;
        x(i)=1;
        x(i)=0;
    }
    else
       return fail;
}
for(i=(n-1);i>=0;i--)
 {
    if(x(i)==0)
    {
        x(i)=1;
        x(i)=0;
    }
    else
        return fail;
}
return pass;
```

Interrupt Test

The Interrupt test implements the independent time slot monitoring H.2.18.10.4 defined by the IEC 60730 standard. It checks whether the number of interrupts that occurred is within the predefined range.

The goal of the Interrupt test is to verify that interrupts occur regularly. The Interrupt test function can be invoked at specified time intervals. It is triggered by a timer or line frequency interrupt to monitor and verify the interrupt operation.

To keep track of the interrupts that occur frequently, a dedicated counter in each ISR can be decremented when an interrupt occurs. For example, if the Serial Peripheral Interface (SPI) is configured to generate an interrupt every 2 ms, the SPI will generate at least five interrupts in 10 ms. When a SPI interrupt occurs, the counter dedicated to keep track of the SPI interrupt is decremented. Thus, if the counter is initialized to five, the counter is decremented to zero in 10 ms. This is verified by the Interrupt test function that is triggered after every 10 ms.

To keep track of interrupts that occur rarely, a dedicated counter within the Interrupt test function is decremented if the specific interrupt did not occur during the last time interval. Refer to the example code, which is available for download from the Microchip web site (see **Appendix A: "Source Code"** for details.).

Clock Test

According to the IEC 60730 standard, only harmonics and subharmonics of the clock need to be tested. The Clock test implements the independent time slot monitoring H.2.18.10.4 defined by the IEC 60730 standard. It verifies the reliability of the system clock (i.e., the system clock should be neither too fast nor too slow):

Depending on the choice of the reference clock, one of the following Clock tests can be used:

- Clock Test Using the Secondary Oscillator
- Clock Test Using the Line Frequency (50 Hz, 60 Hz)

CLOCK TEST USING THE SECONDARY OSCILLATOR

The `SSL_16bitsFamily_CLOCKtest` function is used to verify the proper operation of the CPU clock when the secondary oscillator is used as a reference clock.

This test performs the following major tasks:

1. The LP secondary oscillator is used as an independent clock source or a reference clock source. This 32 kHz oscillator is used to clock Timer1.
2. The primary oscillator with Phased-Lock Loop (PLL) is the clock source to the CPU. The Timer2 runs at the CPU clock frequency.
3. Timer1 is configured to generate an interrupt at specified time intervals (e.g., 1 ms).
4. The PR2 register in the Timer2 module holds the time period value. It must be initialized to a value greater than 1 ms so that Timer2 does not time-out before the occurrence of a Timer1 interrupt.
5. The TMR2 value of Timer2 is saved within the Timer1 interrupt handler. This value represents the number of CPU clock cycles elapsed in the 1 ms time period of the secondary oscillator. If the number of clock cycles is beyond the defined boundary, the function sets an error flag.

For example, the following parameters are used to calculate the `CLK_MIN_TIME` and `CLK_MAX_TIME` values for a dsPIC30F device:

- Primary Oscillator: XT_PLL8
- FOSC: 7.37 MHz * 8
- FCY: FOSC/4: (7.37 * 10^6 * 8)/4
- FCY: 14740000
- Secondary Oscillator: 32 kHz
- Timer1 Period Register (PR1): 31

Therefore, with 4% tolerance, the number of CPU clock cycles in 1 ms (14740 cycles) are:

- `CLK_MIN_TIME:` 14150
- `CLK_MAX_TIME`: 15330

API FUNCTIONS

The following API implements the Clock test:

`SSL_16bitsFamily_CLOCKtest`

CLOCK TEST USING THE LINE FREQUENCY (50 Hz, 60 Hz)

The `SSL_16bitsFamily_CLOCKtest_LineFreq` function is used to verify the proper operation of the CPU clock. The 50 Hz/60 Hz line frequency is used as an independent clock source or a reference clock source. The input capture module is used for the period measurement. The 50 Hz/60 Hz line frequency is fed to the Input Capture pin (IC1) of the respective device.

This test performs the following major tasks:

1. The IC1CON register is configured as follows:
   a) Timer2 is selected as the IC1 time base.
   b) An interrupt is generated on every second capture event.
   c) The capture event is generated on every rising edge of the line frequency.
2. The Timer2 prescaler is configured to operate in 1:8 mode so that the timer count does not time-out within 20 ms/16.66 ms.
3. The capture event is generated on every rising edge of line frequency. For period measurement, the capture interrupt is generated after taking two time-stamps (see Figure 2).
4. The difference between the two time-stamps (V1 and V2) provides the timer period value. The number of CPU cycles in 20 ms/16.66 ms of the line frequency is computed as follows:

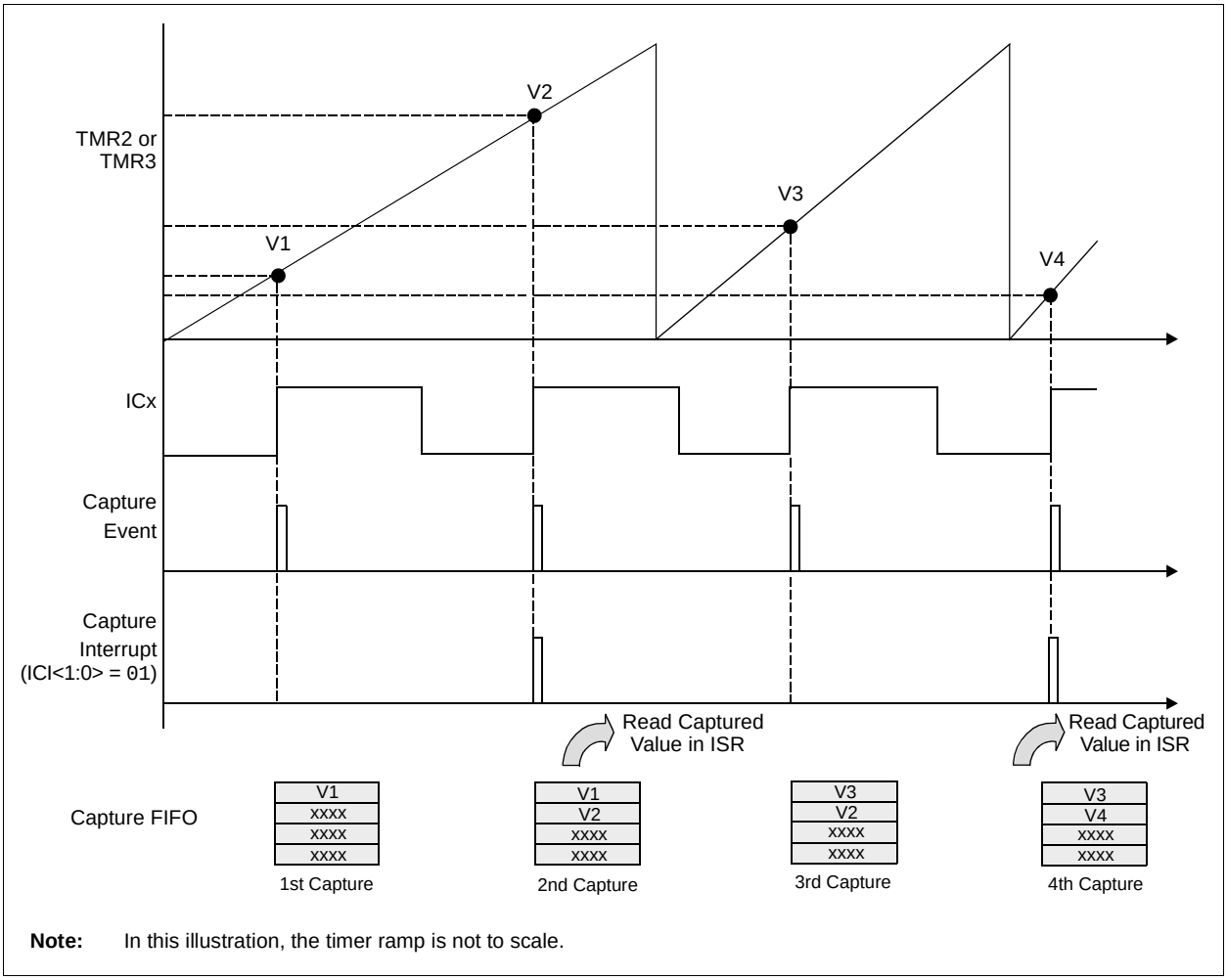
   Number of Clock Cycles = ((V1 – V2) * Timer2 Prescaler)

API FUNCTIONS

The following API implements the Clock test:

`SSL_16bitsFamily_CLOCKtest_LineFreq`

FIGURE 2: TIMER VALUE CAPTURE



Note: In this illustration, the timer ramp is not to scale.

Addressing of Variable and Invariable Memory and Internal Data Path

For single chip microcontrollers or digital signal controllers, such as PIC MCUs and dsPIC DSCs, the Periodic Static Memory test is used to test the variable memory, and the periodic checksum is used to test the invariable memory. These tests detect any stuck-at Fault in the internal address bus and internal data path.

Addressing Wrong Address

This test is required only for microcontrollers with an external memory device.

External Communication

The IEC 60730 Class B specifications suggest the following measures to ensure reliable communication between components:

TRANSFER REDUNDANCY

The transfer redundancy is a Fault/error control technique that protects against coincidental and/or systematic errors in the input and output information. It is achieved by transferring the data between the transmitter and receiver. The data is transferred at least twice in succession and then compared.

PROTOCOL TEST

The Protocol test is a Fault/error control technique in which the data is transferred to and from the computer components to detect errors in the internal communication protocol.

CRC SINGLE WORD

A CRC polynomial is used to calculate the CRC checksum of the transmitted message. At the transmitting end, this CRC checksum is appended to the message before transmitting it. At the receiving end, the receiver uses the same CRC polynomial to compute the CRC checksum, and compares the computed value with the received value.

Timing

The PIC MCUs and dsPIC DSCs have several dedicated communication interfaces, such as UART, I^2C™ and SPI modules. The IEC 60730 Class B specifications suggest that these modules should use time slot monitoring to ensure that the communication occurs at the correct point in time.

Plausibility Check

The plausibility checks on the I/O periphery, analog multiplexer and A/D convertor can be performed as follows:

I/O PERIPHERY

The plausibility check on an I/O pin can be performed by toggling the I/O and checking the state of the pin.

ANALOG MULTIPLEXER

To verify the operation of the analog multiplexer, known voltage values are applied to all channels. These values are read and compared with the applied voltage for verification.

A/D CONVERTER

To test the analog functions of the A/D converter, a known external voltage is applied to the analog inputs. The conversion results are then compared with the applied voltage.

API Functions

This section lists and describes the Application Programming Interface (API) functions that are available in the Class B Safety Software Library. The functions are listed below followed by their individual detailed descriptions:

- `SSL_16bitsFamily_CPU_RegisterTest`
- `SSL_16bitsFamily_PCtest`
- `SSL_16bitsFamily_Flashtest_CRC16`
- `SSL_16bitsFamily_EEPROMtest_CRC16`
- `SSL_16bitsFamily_RAMtest_MarchC`
- `SSL_16bitsFamily_RAM_STACKtest_MarchC`
- `SSL_16bitsFamily_RAMtest_MarchC_Minus`
- `SSL_16bitsFamily_RAMtest_MarchB`
- `SSL_16bitsFamily_RAMtest_CheckerBoard`
- `SSL_16bitsFamily_CLOCKtest`
- `SSL_16bitsFamily_CLOCKtest_LineFreq`

SSL_16bitsFamily_CPU_RegisterTest

Description

This function implements the CPU Register test. The test successively writes the values 0x5555 and 0xAAAA into the CPU registers and then reads the values from these registers for verification. The function returns an error code if the values do not match. The contents of the register (W0) that returns the error code are not preserved. The contents of the CPU registers to be tested are saved on the stack before executing the routine and are restored upon the completion of the test.

Include

None.

Prototype

```
int SSL_16bitsFamily_CPU_RegisterTest();
```

Arguments

None.

Return Value

```
CPU_REGISTER_TEST_FAIL: return value = 0
CPU_REGISTER_TEST_PASS: return value = 0
```

Remarks

None.

Source File

None.

TABLE 1: RESOURCE REQUIREMENTS

Parameter	Requirements
Program Memory	1308 bytes (dsPIC30F/dsPIC33F) 747 bytes (PIC24H/PIC24F)
Stack	62 bytes (dsPIC30F/dsPIC33F) 30 bytes (PIC24H/PIC24F)
Execution Time	351 cycles (dsPIC30F/dsPIC33F) 181 cycles (PIC24H/PIC24F)

SSL_16bitsFamily_PCtest

Description

This function implements the Program Counter test, which is a functional test of the Program Counter (PC). The test invokes the functions that are located in the Flash memory at different addresses. The customized linker script defines the addresses, where these functions reside in the Flash memory. The functions placed at these addresses return a unique value, which is the starting address of the called function. This returned value is verified using the `SSL_16bitsFamily_PCtest` function.

Include

`SSL_PcTest.h`

Prototype

```
SSL_16bitsFamily_PCtest();
```

Arguments

None.

Return Value

`PC_TEST_FAIL: return value = 0`

`PC_TEST_PASS: return value = 0`

Remarks

None.

TABLE 2: RESOURCE REQUIREMENTS

Parameter	Requirements
Program Memory	258 bytes
Stack	28 bytes
Execution Time	32 cycles

SSL_16bitsFamily_Flashtest_CRC16

Description

This function implements the Invariable Memory test. It computes the CRC of the data located between the address FLASH_STARTADDRESS and the address FLASH_ENDADDRESS. This function returns the final CRC value.

Include

SSL_Flash_CRC.h

Prototype

```
unsigned int SSL_16bitsFamily_Flashtest_CRC16
(uReg32 startAddress,uReg32 endAddress, unsigned int init_CrcValue);
```

ARGUMENTS

startAddress	Indicates the starting address of the data to be tested
endAddress	Indicates the ending address of the data to be tested
init_CrcValue	Indicates the initial value of the CRC

Return Value

crc_Result	Holds the CRC result

Remarks

None.

TABLE 3: RESOURCE REQUIREMENTS

Parameter	Requirements
Program Memory	489 bytes
Stack	70 bytes
Execution Time	446 cycles(1)

Note 1: The execution time specified here is for a single Flash memory location.

SSL_16bitsFamily_EEPROMtest_CRC16

Description

This function implements the Invariable Memory test. It computes the CRC of the data located between the address, EEPROM_STARTADDRESS, and the address, EEPROM_ENDADDRESS. This function returns the final CRC value.

Include

SSL_EEPROM_CRC.h

Prototype

```
unsigned int SSL_16bitsFamily_EEPROMtest_CRC16
(uReg32 startAddress,uReg32 endAddress ,unsigned int  init_CrcValue);
```

Arguments

startAddress	Indicates the starting address of the data to be tested
endAddress	Indicates the ending address of the data to be tested
init_CrcValue	Indicates the initial value of the CRC

Return Value

crc_Result	Holds the CRC result

Remarks

None.

Source File

None.

TABLE 4: RESOURCE REQUIREMENTS

Parameter	Requirements
Program Memory	492 bytes
Stack	70 bytes
Execution Time	348 cycles(1)

Note 1: The execution time specified here is for a single EEPROM location.

SSL_16bitsFamily_RAMtest_MarchC

Description

This function implements the March C test. This test accesses a 16-bit word from the RAM memory. The address must be aligned to the data type and the length must be an integral multiple of the data width. This is a destructive test; therefore, this test can be executed at the system start-up before initializing the memory and the run-time libraries. The memory will be cleared when the control returns from the `SSL_16bitsFamily_RAMtest_MarchC` function.

Include

`SSL_MarchC.h`

Prototype

```
int SSL_16bitsFamily_RAMtest_MarchC(int * ramStartAddress,int ramSize);
```

Arguments

`ramStartAddress`	Indicates the starting address from where the March C algorithm starts reading the data
`ramSize`	Indicates the number of bytes that are tested; the size must be an even number

Return Value

`MARCHC_RAM_TEST_FAIL: return value = 0`

`MARCHC_RAM_TEST_PASS: return value = 1`

Remarks

None.

Source File

`SSL_MarchCRamTest.c`

TABLE 5: RESOURCE REQUIREMENTS

Parameter	Requirements
Program Memory	585 bytes
Stack	88 bytes
Execution Time	1254 cycles(1)

Note 1: The execution time specified here is for a single RAM location.

SSL_16bitsFamily_RAM_STACKtest_MarchC

Description

This function implements the March C test on the RAM memory and stack. This test accesses a 16-bit word from the RAM memory. The address must be aligned to the data type and the length must be an integral multiple of the data width. It first tests the RAM memory and then the stack area by transferring the stack contents into the tested RAM area. After the stack is tested, it restores the contents of the stack. This is a destructive test; therefore, this test can be executed at system start-up before initializing the memory and the run-time libraries. The memory will be cleared when the control returns from the `SSL_16bitsFamily_RAM_STACKtest_MarchC` function.

Include

`SSL_MarchC.h`

Prototype

```
int SSL_16bitsFamily_RAM_STACKtest_MarchC(int * ramStartAddress,int ramSize);
```

Arguments

`ramStartAddress`	Indicates the starting address from where the March C algorithm starts reading the data
`ramSize`	Indicates the number of bytes that are tested; the size must be an even number

Return Value

```
MARCHC_RAM_STACK_TEST_FAIL:  return value = 0
MARCHC_RAM_STACK_TEST_PASS:  return value = 1
```

Remarks

None.

Source File

`SSL_MarchCRamAndStackTest.c`

TABLE 6: RESOURCE REQUIREMENTS

Parameter	Requirements
Program Memory	890 bytes
Stack	88 bytes
Execution Time	1576 cycles(1)

Note 1: The execution time specified here is for a single RAM location.

SSL_16bitsFamily_RAMtest_MarchC_Minus

Description

This function implements the March C Minus test. This test accesses a 16-bit word from the RAM memory. The address must be aligned to the data type and the length must be an integral multiple of the data width. This is a destructive test (i.e., the memory contents are not preserved); therefore, this test can be executed at the system start-up before initializing the memory and the run-time libraries. The memory will contain 0xAAAA when the control returns from the `SSL_16bitsFamily_RAMtest_MarchC_Minus` function.

Include

`SSL_MarchC_Minus.h`

Prototype

```
int SSL_16bitsFamily_RAMtest_MarchC_Minus(int * ramStartAddress,int ramSize);
```

Arguments

`ramStartAddress`	Indicates the starting address from where the March C algorithm starts reading the data
`ramSize`	Indicates the number of bytes that are tested; the size must be an even number

Return Value

```
MARCHC_RAM_TEST_FAIL:  return value = 0
MARCHC_RAM_TEST_PASS:  return value = 1
```

Remarks

None.

Source File

`SSL_MarchC_MinusRamTest.c`

TABLE 7: RESOURCE REQUIREMENTS

Parameter	Requirements
Program Memory	381 bytes
Stack	30 bytes
Execution Time	122 cycles(1)

Note 1: The execution time specified here is for a single RAM location.

SSL_16bitsFamily_RAMtest_MarchB

Description

This function implements the March B test. This test accesses a 16-bit word from the RAM memory. The address must be properly aligned to the data type and the length must be an integral multiple of the data width. This is a destructive test; therefore, this test can be executed at system start-up before initializing the memory and the run-time library. The memory will be cleared when the control returns from the `SSL_16bitsFamily_RAMtest_MarchB` function.

Include

`SSL_MarchB.h`

Prototype

```
int SSL_16bitsFamily_RAMtest_MarchB(int * ramStartAddress,int ramSize);
```

Arguments

`ramStartAddress`	Indicates the starting address from where the March B algorithm starts reading the data
`ramSize`	Indicates the number of bytes that are tested; the size must be an even number

Return Value

`MARCHB_TEST_FAIL: return value = 0`

`MARCHB_TEST_PASS: return value = 1`

Remarks

None.

Source File

`SSL_MarchBRamTest.c`

TABLE 8: RESOURCE REQUIREMENTS

Parameter	Requirements
Program Memory	630 bytes
Stack	88 bytes
Execution Time	1183 cycles(1)

Note 1: The execution time specified here is for a single RAM location.

SSL_16bitsFamily_RAMtest_CheckerBoard

Description

This function implements the Checkerboard test on the RAM memory. The test is performed on the memory space specified by the variable, RamSize. The execution begins from the address defined by the variable, RAMSTARTADDRESS. The number of specified locations must be even.

Include

SSL_CBram.h

Prototype

```
int SSL_16bitsFamily_RAMtest_CheckerBoard(int *ramStartAddress,int RamSize);
```

Arguments

RamStartAddress	Indicates the starting address from where the Checkerboard test is to be performed
RamSize	Indicates the number of locations that are tested; the size must be an even number

Return Value

CB_TEST_FAIL: return value = 0

CB_TEST_PASS: return value = 1

Remarks

None.

Source File

SSL_CheckerBoard.s

TABLE 9: RESOURCE REQUIREMENTS

Parameter	Requirements
Program Memory	321 bytes
Stack	68 bytes
Execution Time	43 cycles(1)

Note 1: The execution time specified here is for a single RAM location.

SSL_16bitsFamily_CLOCKtest

Description

This function implements the Clock test. It is used to verify the proper operation of the CPU clock. The TMR2 value of Timer2 is saved within the Timer1 interrupt handler. This value represents the number of CPU clock cycles elapsed in 1 ms time period of the secondary oscillator. If the number of clock cycles is beyond the defined boundary, the function sets an error flag.

Include

`SSL_ClockTest.h`

`SSL_ClockSwitch.h` This file is required only when a PIC24F device is used

Prototype

```
unsigned int SSL_16bitsFamily_CLOCKtest(void);
```

Arguments

None.

Return Value

`CLOCK_NO_ERROR` The CPU clock is operating within the specified range

`CLOCK_ERROR` The CPU clock is not operating within the specified range

Remarks

None.

Source File

`SSL_ClockTest.c`

TABLE 10: RESOURCE REQUIREMENTS

Parameter	Requirements
Program Memory	387 bytes
Stack	8 bytes
Execution Time	30 cycles

SSL_16bitsFamily_CLOCKtest_LineFreq

Description

This function implements the line frequency Clock test. It is used to verify the proper operation of the CPU clock. It uses the following procedure to configure the IC1CON register:

1. The Timer2 module is selected as the IC1 time base.
2. An interrupt is generated on every second capture event.
3. The capture event is generated on every rising edge of the line frequency.

The IC1 pin generates an interrupt after every 20 ms if the line frequency is 50 Hz and after every 16.66 ms if the line frequency is 60 Hz. The Timer2 prescaler is configured to operate in 1:8 mode so that the timer count does not time-out within 20 ms/16.66 ms. The capture event is generated on every rising edge of the line frequency. For period measurement, the capture interrupt is generated after taking two time-stamps, V1 and V2 (see Figure 2). The total number of clock cycles is calculated using the following formula:

Total Number of Clock Cycles = Timer Count * Timer Prescaler.

If the number of clock cycles is beyond the defined boundary, the function sets an error flag.

Include

`SSL_ClockTest_LineFreq.h`

`SSL_ClockSwitch.h` This file is required only when a PIC24F device is used

Prototype

```
int  SSL_16bitsFamily_CLOCKtest_LineFreq();
```

Arguments

None.

Return Value

`CLOCK_NO_ERROR`	The CPU clock is operating within the specified range
`CLOCK_ERROR`	The CPU clock is not operating within the specified range

Remarks

None.

Source File

`SSL_ClockTest_LineFreq.c`

TABLE 11: RESOURCE REQUIREMENTS

Parameter	Requirements
Program Memory	447 bytes
Stack	12 bytes
Execution Time	25 cycles

SUMMARY

This application note describes how to implement various diagnostic measures proposed by the IEC 60730 standard. These measures ensure the safe operation of controlled equipment that falls under the Class B category. In addition, this application note also describes the different APIs that are available in the Class B Safety Software Library. These APIs can be directly integrated with the end user's application to test and verify the critical functionalities of a controller and are intended to maximize the application reliability through Fault detection. When implemented on a dsPIC DSC or PIC MCU microcontroller, these APIs help meet the IEC 60730 standard's requirements.

Microchip has developed the Class B Safety Software Library to assist you in implementing the safety software routines. Contact your Microchip sales or application engineer if you would like further support.

REFERENCES

- IEC 60730 Standard, *"Automatic Electrical Controls for Household and Similar Use",* IEC 60730-1 Edition 3.2, 2007-03
- Bushnell, M., Agarwal, V. *"Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits"* New York: Springer, 1st ed. 2000. Corr. 2nd printing, 2005
- Wu, C. *"Memory Testing"*
- Wu, C. "*RAM Fault Models and Memory Testing*"
- Suk, D.S. and Reddy, S.M. *"A March Test for Functional Faults in Semiconductor Random-Access Memories"*, IEEE Trans. Computers, Vol. C-30, No. 12, 1981, pp. 982-985

APPENDIX A: SOURCE CODE

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the “Company”) is intended and supplied to you, the Company’s customer, for use solely and exclusively with products manufactured by the Company.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

All of the software covered in this application note is available as a single WinZip archive file. This archive can be downloaded from the Microchip corporate web site at:

www.microchip.com

APPENDIX B: IEC 60730-1 TABLE H.11.12.7

The following table is reproduced with the permission of the International Electrotechnical Commission (IEC). IEC 60730-1 ed.3.2 “Copyright © 2007 IEC, Geneva, Switzerland. www.iec.ch”.

TABLE B-1: H.11.12.7

Component [1]	Fault/error	Software class		Acceptable measures [2] [3] [4]	Definitions
		B	C		
1. CPU					
1.1					
Registers	Stuck at	rq		Functional test, or	H.2.16.5
				periodic self-test using either:	H.2.16.6
				– static memory test, or	H.2.19.6
				– word protection with single bit redundancy	H.2.19.8.2
	DC fault		rq	Comparison of redundant CPUs by either:	
				– reciprocal comparison	H.2.18.15
				– independent hardware comparator, or	H.2.18.3
				Internal error detection, or	H.2.18.9
				redundant memory with comparison, or	H.2.19.5
				periodic self-tests using either	
				– walkpat memory test	H.2.19.7
				– Abraham test	H.2.19.1
				– transparent GALPAT test; or	H.2.19.2.1
				word protection with multi-bit redundancy, or	H.2.19.8.1
				static memory test and word protection	H.2.19.6
				with single bit redundancy	H.2.20.8.2
1.2					
Instruction	Wrong		rq	Comparison of redundant CPUs by either:	
decoding and	decoding			– reciprocal comparison	H.2.18.15
execution	and execution			– independent hardware comparator, or	H.2.18.3
				internal error defection, or	H.2.18.9
				periodic self-test using equivalence class test	H.2.18.5
1.3				Functional test, or	H.2.16.5
Programme	Stuck at	rq		periodic self-test, or	H.2.16.6
counter				independent time-slot monitoring, or	H.2.18.10.4
				logical monitoring of the programme sequence	H.2.18.10.2
				Periodic self-test and monitoring using either:	H.2.16.7
	DC fault		rq	– independent time-slot and logical monitoring	H.2.18.10.3
				– internal error detection, or	H.2.18.9
				comparison of redundant functional channels by either:	
				– reciprocal comparison	H.2.18.15
				– independent hardware comparator	H.2.18.3
1.4					
Addressing	DC fault		rq	Comparison of redundant CPUs by either:	
				– reciprocal comparison	H.2.18.15
				– independent hardware comparator; or	H.2.18.3
				Internal error detection; or	H.2.18.9
				periodic self-test using a testing pattern of	H.2.16.7
				the address lines; or	H.2.18.22
				full bit bus parity including the address	H.2.18.1.1
					H.2.18.1.2

TABLE B-1: H.11.12.7 (CONTINUED)

Component [1]	Fault/error	Software class		Acceptable measures [2] [3] [4]	Definitions
		B	C		
1.5 Data paths instruction decoding	DC fault and execution		rq	Comparison of redundant CPUs by either: reciprocal comparison, or independent hardware comparator, or Internal error detection, or periodic self-test using a testing pattern, or data redundancy, or multi-bit bus parity	 H.2.18.15 H.2.18.3 H.2.18.9 H.2.16.7 H.2.18.22 H.2.18.1.2
2. Interrupt handling and execution	No interrupt or too frequent interrupt	rq		Functional test; or time-slot monitoring	 H.2.16.5 H.2.18.10.4
	No interrupt or too frequent interrupt related to different sources		rq	Comparison of redundant functional channels by either reciprocal comparison, independent hardware comparator, or Independent time-slot and logical monitoring	 H.2.18.15 H.2.18.3 H.2.18.10.3
3. Clock		rq		Frequency monitoring, or time slot monitoring	H.2.18.10.1 H.2.18.10.4
	Wrong frequency (for quartz synchronized clock: harmonics/ subharmonics only)		rq	Frequency monitoring, or time-slot monitoring, or comparison of redundant functional channels by either: – reciprocal comparison – independent hardware comparator	H.2.18.10.1 H.2.18.10.4 H.2.18.15 H.2.18.3
4. Memory 4.1 Invariable memory	All single bit faults	rq		Periodic modified checksum; or multiple checksum, or word protection with single bit redundancy	 H.2.19.3.1 H.2.19.3.2 H.2.19.8.2
	99,6 % coverage of all information errors		rq	Comparison of redundant CPUs by either: – reciprocal comparison – independent hardware comparator, or redundant memory with comparison, or periodic cyclic redundancy check, either – single word – double word, or word protection with multi-bit redundancy	 H.2.18.15 H.2.18.3 H.2.19.5 H.2.19.4.1 H.2.19.4.2 H.2.19.8.1

TABLE B-1: H.11.12.7 (CONTINUED)

Component [1]	Fault/error	Software class		Acceptable measures [2] [3] [4]	Definitions
		B	C		
4.2 Variable memory	DC fault	rq		Periodic static memory test, or	H.2.19.6
				word protection with single bit redundancy	H.2.19.8.2
	DC fault and dynamic cross links		rq	Comparison of redundant CPUs by either:	
				– reciprocal comparison	H.2.18.15
				– independent hardware comparator, or	H.2.18.3
				redundant memory with comparison, or	H.2.19.5
				periodic self tests using either:	
				– walkpat memory test	H.2.19.7
				– Abraham test	H.2.19.1
				– transparent GALPAT test, or	H.2.19.2.1
				word protection with multi-bit redundancy	H.2.19.8.1
4.3 Addressing (relevant to variable and invariable memory)	Stuck at	rq		Word protection with single bit parity including the address, or	H.2.19.18.2
	DC fault		rq	comparison of redundant CPUs by either:	
				– reciprocal comparison, or	H.2.18.15
				– independent hardware comparator, or	H.2.18.3
				full bus redundancy	H.2.18.1.1
				Testing pattern, or	
				periodic cyclic redundancy check, either:	H.2.18.22
				– single word	H.2.19.4.1
				– double word, or	H.2.19.4.2
				word protection with multi-bit redundancy including the address	H.2.19.8.1
5. Internal data path 5.1 Data	Stuck at	rq		Word protection with single bit redundancy	H.2.19.8.2
	DC fault		rq	Comparison of redundant CPUs by either:	
				– reciprocal comparison	H.2.18.15
				– independent hardware comparator, or	H.2.18.3
				word protection with multi-bit redundancy	H.2.19.8.1
				including the address, or data redundancy, or	H.2.18.2.1
				testing pattern, or	H.2.18.22
				protocol test	H.2.18.14
5.2 Addressing	Wrong address	rq		Word protection with single bit redundancy including the address	H.2.19.8.2
	Wrong address and multiple addressing		rq	Comparison of redundant CPUs by:	
				– reciprocal comparison	H.2.18.15
				– independent hardware comparator, or	H.2.18.3
				word protection with multi-bit redundancy, including the address, or full bus redundancy; or testing pattern including the address	H.2.19.8.1 H.2.18.1.1 H.2.18.22

TABLE B-1: H.11.12.7 (CONTINUED)

Component [1]	Fault/error	Software class		Acceptable measures [2] [3] [4]	Definitions
		B	C		
6 External communication	Hamming distance 3	rq		Word protection with multi-bit redundancy, or CRC – single word , or	H.2.19.8.1 H.2.19.4.1
				transfer redundancy, or	H.2.18.2.2
				protocol test	H.2.18.14
6.1 Data	Hamming distance 4		rq	CRC – double word, or	H.2.19.4.2
				data redundancy or comparison of redundant functional channels by either:	H.2.18.2.1
				– reciprocal comparison	H.2.18.15
				– independent hardware comparator	H.2.18.3
6.2 Addressing	Wrong address		rq	Word protection with multi-bit redundancy, including the address, or CRC single word	H.2.19.8.1 H.2.19.4.1
				including the addresses, or	
				transfer redundancy or	H.2.18.2.2
				protocol test	H.2.18.14
	Wrong and multiple addressing		rq	CRC – double word, including the address, or full bus redundancy of data and address, or comparison of redundant communication channels by either:	H.2.19.4.2 H.2.18.1.1
				– reciprocal comparison	H.2.18.15
				– independent hardware comparator	H.2.18.3
6.3 Timing	Wrong point in time	rq		Time-slot monitoring, or scheduled transmission	H.2.18.10.4 H.2.18.18
		rq		Time-slot and logical monitoring, or	H.2.18.10.3
				comparison of redundant communication channels by either:	
				– reciprocal comparison	H.2.18.15
				– independent hardware comparator	H.2.18.3
	Wrong sequence	rq		Logical monitoring, or time-slot monitoring, or scheduled transmission	H.2.18.10.2 H.2.18.10.4 H.2.18.18
			rq	(same options as for wrong point in time)	
7. Input/output periphery	Fault conditions specified in H.27	rq		Plausibility check	H.2.18.13
			rq	Comparison of redundant CPUs by either:	
				– reciprocal comparison	H.2.18.15
				– independent hardware comparator, or	H.2.18.3
7.1 Digital I/O				input comparison, or	H.2.18.8
				multiple parallel outputs; or	H.2.18.11
				output verification, or	H.2.18.12
				testing pattern, or	H.2.18.22
				code safety	H.2.18.2

TABLE B-1: H.11.12.7 (CONTINUED)

Component [1]	Fault/error	Software class		Acceptable measures [2) 3) 4)]	Definitions
		B	C		
7.2 Analog I/O					
7.2.1 A/D- and D/A- convertor	Fault conditions specified in H.27	rq		Plausibility check	H.2.18.13
			rq	Comparison of redundant CPUs by either:	
				– reciprocal comparison	H.2.18.15
				– independent hardware comparator, or	H.2.18.3
				input comparison, or	H.2.18.8
				multiple parallel outputs, or	H.2.18.11
				output verification, or	H.2.18.12
				testing pattern	H.2.18.22
7.2.2 Analog multiplexer	Wrong addressing	rq		Plausibility check	H.2.18.13
			rq	Comparison of redundant CPUs by either:	
				– reciprocal comparison	H.2.18.15
				– independent hardware comparator, or	H.2.18.3
				input comparison or	H.2.18.8
				testing pattern	H.2.18.22
8. Monitoring devices and comparators	Any output outside the static and dynamic functional specification		rq	Tested monitoring, or	H.2.18.21
				redundant monitoring and comparison, or	H.2.18.17
				error recognizing means	H.2.18.6
9. Custom chips [5)] e.g. ASIC, GAL, Gate array	Any output outside the static and dynamic functional specification	rq		Periodic self test	H.2.16.6
			rq	Periodic self-test and monitoring, or	H.2.16.7
				dual channel (diverse) with comparison, or	H.2.16.2
				error recognizing means	H.2.18.6

CPU: Central Programmation Unit

rq: Coverage of the fault is required for the indicated software class.

1) For fault/error assessment, some components are divided into their subfunctions.

2) For each subfunction in the table, the software class C measure will cover the software class B fault/error.

3) It is recognized that some of the acceptable measures provide a higher level of assurance than is required by this standard.

4) Where more than one measure is given for a subfunction, these are alternatives.

5) To be divided as necessary by the manufacturer into subfunctions.

6) Table H.11.12.7 is applied according to the requirements of H.11.12 to H.11.12.13 inclusive.

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Trademarks

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

01/02/08