

## Class B Safety Software Library for PIC<sup>®</sup> MCUs and dsPIC<sup>®</sup> DSCs

Authors: Veena Kudva & Adrian Aur  
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](http://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](http://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, which applies to 16-bit and 32-bit devices, 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 for supported architectures.

## 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 binary sequences (length is dependant upon architecture), 010101... followed by 101010... 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.

<b>Note:</b> The interrupts should be disabled during the execution of the CPU Register test so that the register integrity is preserved at all times.
--

## API FUNCTIONS

The following API functions implement the CPU Register test:

- `SSL_16bitsFamily_CPU_RegisterTest`
- `SSL_32bitsFamily_CPU_RegisterTest`

## Program Counter Test

The Program Counter (PC) 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 PC 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 branches 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. The actual Flash address space is processor dependent. Please refer to the processor-specific linker script example provided.

### API FUNCTIONS

The following API functions implement the PC test:

- SSL\_16bitsFamily\_PCtest
- SSL\_32bitsFamily\_PCtest

### 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);
}
```

## 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 illustrates the flowchart for the Invariable Memory test.

The CRC16 calculation 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 CRC16 calculation 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.

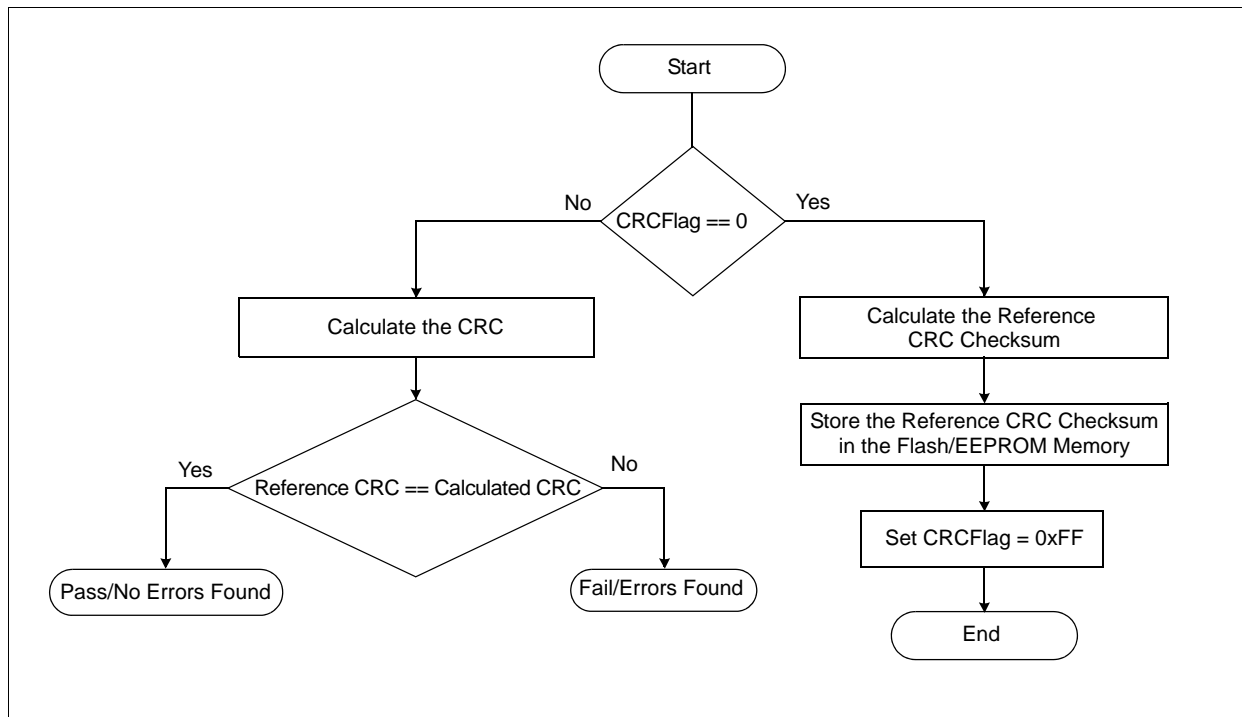
## API FUNCTIONS

The following API functions implement the Invariable Memory test:

- `SSL_16bitsFamily_Flashtest_CRC16`
- `SSL_16bitsFamily_EEPROMtest_CRC16`
- `SSL_32bitsFamily_Flashtest_CRC16`

**Note:** The EEPROM test applies only to dsPIC30F devices.

**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

Figure 2 illustrates the notations that are used in the March test.

**FIGURE 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.
r0	Indicates a read operation (reads '0' from a memory cell).
r1	Indicates a read operation (reads '1' from a memory cell).
w0	Indicates a write operation (writes '0' to a memory cell).
w1	Indicates a write operation (writes '1' to a memory cell).

**Note:** The March memory functions do not test the Stack area of the RAM. The following special functions are provided for the Stack area test:

```
SSL_16bitsFamily_RAM_STACKtest_MarchC
SSL_32bitsFamily_RAM_STACKtest_MarchC
```

## 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 2 shows the pseudocode that demonstrates the implementation of the March C test.

## API FUNCTIONS

The following API functions implement the March C test:

- SSL\_16bitsFamily\_RAMtest\_MarchC
- SSL\_16bitsFamily\_RAM\_STACKtest\_MarchC
- SSL\_32bitsFamily\_RAMtest\_MarchC
- SSL\_32bitsFamily\_RAM\_STACKtest\_MarchC

Figure 3 illustrates a March C algorithm.

**FIGURE 3: MARCH C ALGORITHM**

```
MarchC
{
    ⤴(w0); ⤴(r0, w1); ⤴(r1, w0);
    ⤴(r0); ⤴(r0, w1); ⤴(r1, w0); ⤴(r0)
}
```

## EXAMPLE 2: 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 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  $10n$ , where  $n$  indicates the number of bits in the memory.

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.

## API FUNCTIONS

The following API functions implement the March C Minus test:

- `SSL_16bitsFamily_RAMtest_MarchC_Minus`
- `SSL_32bitsFamily_RAMtest_MarchC_Minus`

Figure 4 illustrates a March C Minus algorithm.

**FIGURE 4: MARCH C MINUS ALGORITHM**

```
MarchC
{
     $\Downarrow(w0); \Uparrow(r0, w1); \Uparrow(r1, w0);$ 
     $\Downarrow(r0); \Downarrow(r0, w1); \Downarrow(r1, w0);$ 
}
```

## 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.

Figure 5 illustrates a March B algorithm.

**FIGURE 5: MARCH B ALGORITHM**

```
MarchB
{
    ⤵(w0); ⤴(r0, w1, r1, w0, r0, w1); ⤴(r1, w0, w1);
    ⤵(r1, w0, w1, w0); ⤵(r0, w1, w0);
}
```

Example 3 shows the pseudocode that demonstrates the implementation of the March B test.

## API FUNCTIONS

The following API functions implement the March B test:

- SSL\_16bitsFamily\_RAMtest\_MarchB
- SSL\_32bitsFamily\_RAMtest\_MarchB

**Note 1:** The user application should allocate appropriate space for the stack before executing any of the March tests (see the details in the specific API function description). The stack must be allocated at an appropriate address so that it does not get overwritten during the test execution.

- 2: Depending on the architecture, it is recommended that the stack 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<sup>®</sup> ICD 2 RAM space or other debugger used RAM space).

**EXAMPLE 3: 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;
```



## CHECKERBOARD RAM TEST

The Checkerboard RAM test writes the checkerboard patterns to a sequence of adjacent memory locations. This test is performed in units (memory chunks) of architecture-specific sizes (4 bytes for 16-bit architecture, 64 bytes for 32-bit architecture). 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 binary value (length is dependant upon architecture) 101010... to the memory location, 'N', and the inverted binary value, 010101..., to the memory location, 'N+1', and so on, until the whole memory chunk is filled.
3. Reads the contents of all the memory locations in the current chunk and verifies its contents. If the values match, the function continues; otherwise it stops and returns an error.
4. Step 2 and 3 are repeated by writing the inverted pattern to the same locations.
5. Once a memory chunk is completed the test of the next chunk is started until all of the requested memory area is tested.

## API FUNCTIONS

The following API functions implement the Checkerboard RAM test:

- `SSL_16bitsFamily_RAMtest_CheckerBoard`
- `SSL_32bitsFamily_RAMtest_CheckerBoard`

## 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 (Sosc)
- Clock Test Using the Line Frequency (50 Hz, 60 Hz)

## CLOCK TEST USING THE Sosc

The Clock Test function is used to verify the proper operation of the CPU clock when the Sosc 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 the hardware Timer1.
2. Usually, the Primary Oscillator (Posc) with Phase-Locked Loop (PLL) is the clock source to the CPU. The test uses a hardware timer that runs at the CPU clock frequency (Timer2 for 16-bit architecture and the CPU Core timer for 32-bit architecture).
3. Timer1 is configured to overflow and generate an interrupt at specified time intervals (e.g., 1 ms).
4. The value of the hardware timer used to count the CPU clock counts is saved when Timer1 overflows. This value represents the number of CPU clock cycles elapsed in the 1 ms time period of the Sosc. If the number of clock cycles is outside a specified range, the function returns an error code.

## API FUNCTIONS

The following API functions implement the Clock test:

- `SSL_16bitsFamily_CLOCKtest`
- `SSL_32bitsFamily_CLOCKtest`

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

The Clock Test 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) Hardware Timer2 is selected as the IC1 time base.
  - b) The capture operation is programmed to occur on every rising edge of the line frequency.
  - c) A capture done event (interrupt) is programmed to occur on every second capture event

2. The Timer2 prescaler is configured so that the timer count does not time-out within 20 ms/ 16.66 ms.
3. The capture is performed on every rising edge of line frequency. For period measurement, the capture done event (interrupt) is generated after taking two time-stamps (see Figure 6).
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:  

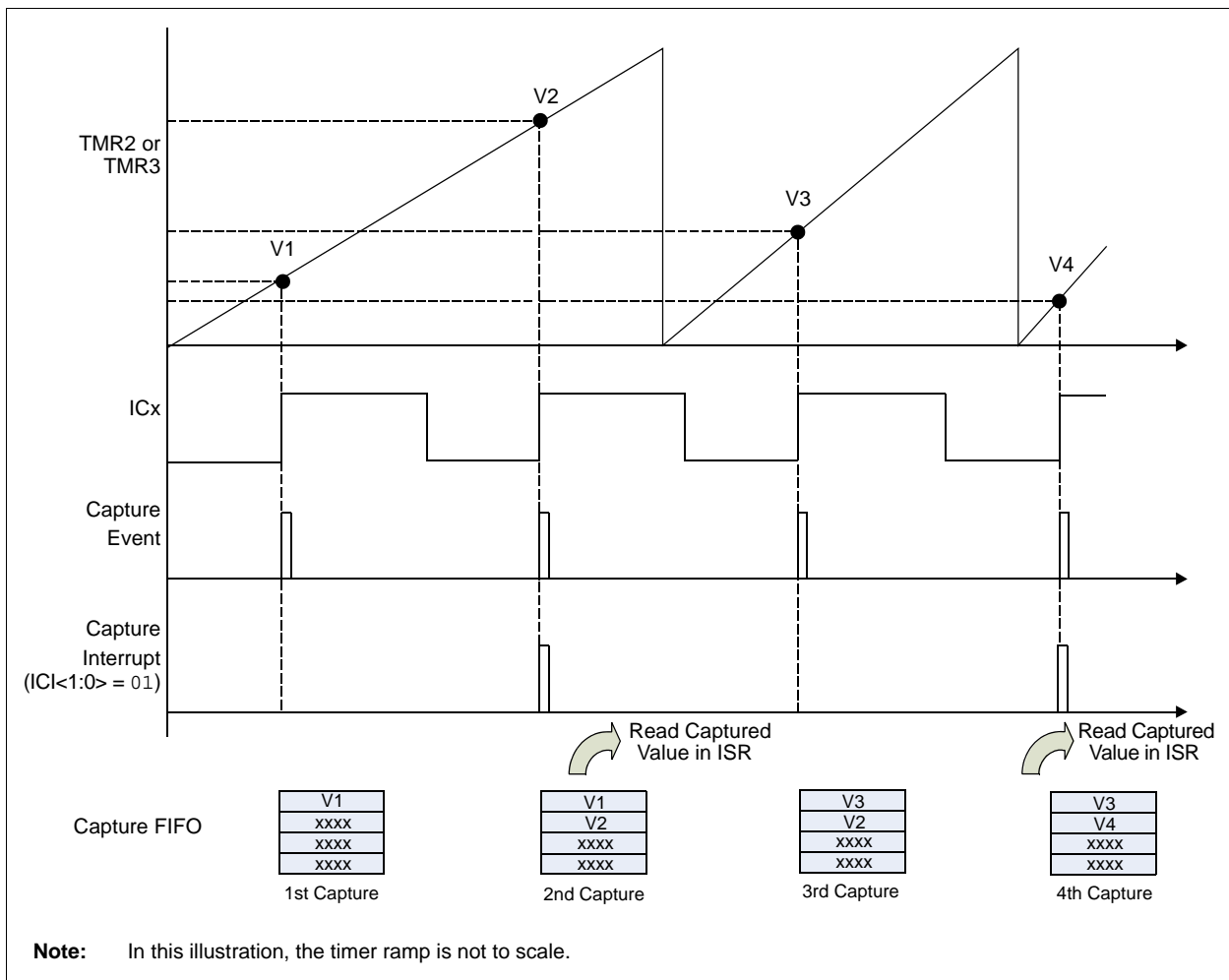
$$\text{Number of Clock Cycles} = ((V1 - V2) * \text{Timer2 Prescaler})$$

## API FUNCTIONS

The following API functions implement the Clock test:

- SSL\_16bitsFamily\_CLOCKtest\_LineFreq
- SSL\_32bitsFamily\_CLOCKtest\_LineFreq

**FIGURE 6: TIMER VALUE CAPTURE**



## 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<sup>2</sup>C™ 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 for 16-Bit PIC MCUs and dsPIC DSCs

This section lists and describes the API functions that are available in the Class B Safety Software Library for 16-bit architecture. The API 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 = 1

### 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 PC test, which is a functional test of the 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 = 1

**Remarks**

None.

**TABLE 2:      RESOURCE REQUIREMENTS**

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

# AN1229

---

## 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

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

### Return Value

<code>crc_Result</code>	Holds the CRC result
-------------------------	----------------------

### Remarks

None.

**TABLE 3: RESOURCE REQUIREMENTS**

Parameter	Requirements
Program Memory	489 bytes
Stack	70 bytes
Execution Time	446 cycles <sup>(1)</sup>

**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 <sup>(1)</sup>

**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.

**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 test execution.

**2:** It is recommended that the stack be placed at the beginning or at the end of the data memory. The user application should specify an address that 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
```

### Source File

SSL\_MarchCRamTest.c

**TABLE 5: RESOURCE REQUIREMENTS**

Parameter	Requirements
Program Memory	585 bytes
Stack	88 bytes
Execution Time	1254 cycles <sup>(1)</sup>

**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.

**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 test execution.

**2:** It is recommended that the stack be placed at the beginning or at the end of the data memory. The user application should specify an address that 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*/
    __SP LIM_init = .;
    . += 8;
} >data
```

### Source File

SSL\_MarchCRamAndStackTest.c

**TABLE 6: RESOURCE REQUIREMENTS**

Parameter	Requirements
Program Memory	890 bytes
Stack	88 bytes
Execution Time	1576 cycles <sup>(1)</sup>

**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.

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.

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

### EXAMPLE 4: 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 0xAAAA
}
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;
```

---

**SSL\_16bitsFamily\_RAMtest\_MarchC\_Minus(Continued)**

---

**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 <sup>(1)</sup>

**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.

**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 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
```

### Source File

SSL\_MarchBRamTest.c

**TABLE 8: RESOURCE REQUIREMENTS**

Parameter	Requirements
Program Memory	630 bytes
Stack	88 bytes
Execution Time	1183 cycles <sup>(1)</sup>

**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

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

### Return Value

<code>CB_TEST_FAIL</code>	Return value = 0
<code>CB_TEST_PASS</code>	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 <sup>(1)</sup>

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

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 Posc with Phase-Locked Loop (PLL) is the clock source to the CPU. 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 Sosc. 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:

- Posc: XT\_PLL8
- Fosc: 7.37 MHz \* 8
- Fcy: FOSC/4:  $(7.37 * 10^6 * 8)/4$
- Fcy: 14740000
- Sosc: 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

### 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.

---

**SSL\_16bitsFamily\_CLOCKtest(Continued)**

---

**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 6). 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



## 32-Bit API Functions for 32-Bit PIC MCUs

This section lists and describes the API functions that are available in the PIC32MX Class B Safety Software Library for the 32-bit architecture. The functions are listed below followed by their individual detailed descriptions:

- `SSL_32bitsFamily_CPU_RegisterTest`
- `SSL_32bitsFamily_PCtest`
- `SSL_32bitsFamily_Flashtest_CRC16`
- `SSL_32bitsFamily_RAMtest_MarchC`
- `SSL_32bitsFamily_RAM_STACKtest_MarchC`
- `SSL_32bitsFamily_RAMtest_MarchC_Minus`
- `SSL_32bitsFamily_RAMtest_MarchB`
- `SSL_32bitsFamily_RAMtest_CheckerBoard`
- `SSL_32bitsFamily_CLOCKtest`
- `SSL_32bitsFamily_CLOCKtest_LineFreq`

## SSL\_32bitsFamily\_CPU\_RegisterTest

### Description

This function implements the CPU Register test. First it tests the register 0 to have all bits cleared. For all the other registers (1 to 31) the test successively writes the values 0x55555555 and 0xAAAAAAAA into the corresponding CPU register, and then reads the values from the register for verification. The function returns an error code if the values do not match.

### Include

SSL\_CpuRegisterTest.h

### Prototype

```
int SSL_32bitsFamily_CPU_RegisterTest(void);
```

### Arguments

None.

### Return Value

CPU\_REGISTER\_TEST\_FAIL      The test failed. Some CPU register(s) has been detected to have stuck bits  
CPU\_REGISTER\_TEST\_PASS      The test passed. CPU registers have not been detected to have stuck bits

### Remarks

This is a non-destructive test. Interrupts should be disabled when calling this test function.

The standard C language call convention applies regarding the registers usage (refer to **Section 5.6 “Function Calling Convention”** in the MPLAB® C Compiler For PIC32 MCUs User's Guide (DS51686)).

Upon the function entry, all registers that need to be preserved are saved into the stack. These registers are restored upon completion of the test. Upon the function return, the register v0 will contain the return code.

### Source File

SSL\_CPURegisterTest.S

**TABLE 12: RESOURCE REQUIREMENTS**

Parameter	Requirements
Program Memory	2156 bytes
Stack	44 bytes
Execution Time	922 cycles (11.5 $\mu$ s) <sup>(1)</sup>

- Note 1:** The value for the CPU cycles can vary greatly depending on the system settings and the build parameters. All execution time measurements are done with the following settings: 80 MHz, 2 Flash wait States, 0 RAM wait states, Prefetch and Cache enabled, PBDIV = 1:1 interrupts and DMA disabled.
- 2:** The co-processor 0 (CP0) registers are not tested by this function. These registers are configuration and status registers and modifying their values could adversely affect the system behaviour at run time.

---

## SSL\_32bitsFamily\_PCtest

---

### Description

This function implements the PC test, which is a functional test of the PC. It checks that the PC register is not stuck and it properly holds the address of the next instruction to be executed.

The test invokes the functions that are located in Flash memory at different addresses. 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_32bitsFamily_PCtest` function.

The provided customized linker script, `elf32pic32mx.ld`, defines the addresses where these functions reside in Flash memory.

### Include

`SSL_PcTest.h`

### Prototype

```
int SSL_32bitsFamily_PCtest(void);
```

### Arguments

None.

### Return Value

`PC_TEST_FAIL`      The test failed. The PC register has been detected to hold an incorrect address.  
`PC_TEST_PASS`      The test passed. The PC register holds the correct address.

### Remarks

The test uses three different functions:

- `SSL_TestPCFunction1()`
- `SSL_TestPCFunction2()`
- `SSL_TestPCFunction3()`

The ROM location of these functions that are used for PC test at run time can be changed by modifying the provided `elf32pic32mx.ld` file.

The `elf32pic32mx.ld` linker script file should be added to the project.

### Source File

**TABLE 13: RESOURCE REQUIREMENTS**

Parameter	Requirements
Program Memory	332 bytes (debug build) 212 bytes (-O3 -fomit-frame-pointer)
Stack	48 bytes (debug build)
Execution Time	90 cycles (1.125 $\mu$ s) <sup>(1)</sup>

**Note 1:** The value for the CPU cycles can vary greatly depending on the system settings and the build parameters. All execution time measurements are done with the following settings: 80 MHz, 2 Flash wait States, 0 RAM wait states, Prefetch and Cache enabled, PBDIV = 1:1 interrupts and DMA disabled. Build: MPLAB C32 V1.10(b) Release -O3 -fomit-frame-pointer.

## SSL\_32bitsFamily\_Flashtest\_CRC16

### Description

This function calculates the 16-bit CRC of the supplied memory area using the standard Linear Feedback Shift Register (LFSR) implementation.

It calculates the CRC over the memory area between the start address and end address and returns the CRC Value.

The 16-bit CRC is calculated using the supplied generator polynomial and initial seed. Different generator polynomials can be selected.

### Include

SSL\_Flash\_CRC.h

### Prototype

```
unsigned int SSL_32bitsFamily_Flashtest_CRC16(char* startAddress, char* endAddress,
unsigned int crcPoly, unsigned int crcSeed);
```

### Arguments

startAddress	Start address of the memory area to calculate the CRC over
endAddress	Final address for which the CRC is calculated
crcPoly	The generator polynomial to be used. One of the standard supplied polynomials can be used as well as other user-defined ones.
crcSeed	The initial value in the CRC LFSR. The usual recommended value is 0xFFFF.

### Return Value

The value of the calculated CRC over the specified memory area.

### Remarks

This test is non-destructive for the memory area to which it is applied.

The start address and end address over which the CRC value is calculated are PIC32 variant and application-dependent.

They are run-time parameters.

### Source File

SSL\_FlashTest\_CRC16.c

**TABLE 14: RESOURCE REQUIREMENTS**

Parameter	Requirements
Program Memory	348 bytes (debug build) 196 bytes (-O3 -fomit-frame-pointer)
Stack	48 bytes (debug build)
Execution Time	99264 cycles (1240.8 us) for 1024 bytes Flash CRC <sup>(1)</sup>

**Note 1:** The value for the CPU cycles can vary greatly depending on the system settings and the build parameters. All execution time measurements are done with the following settings: 80 MHz, 2 Flash wait States, 0 RAM wait states, Prefetch and Cache enabled, PBDIV = 1:1 interrupts and DMA disabled. Build: MPLAB C32 V1.10(b) Release -O3 -fomit-frame-pointer.

---

**SSL\_32bitsFamily\_RAMtest\_MarchC**


---

**Description**

This function implements the March C test. This test performs 32-bit word RAM accesses. The address of the RAM area to be tested must be 32-bit aligned and the size of the tested RAM area must be an integral multiple of 4.

The tested RAM memory will be cleared when the control returns from the `SSL_32bitsFamily_RAMtest_MarchC` function.

**Include**

`SSL_MarchC.h`

**Prototype**

```
int SSL_32bitsFamily_RAMtest_MarchC(int* ramStartAddress, int ramSize);
```

**Arguments**

<code>ramStartAddress</code>	Start address from which the March C test is to be performed. Must be properly 32-bit aligned.
<code>ramSize</code>	Number of consecutive byte locations for which the test is to be performed. The size must be a number multiple of 4.

**Return Value**

<code>MARCHC_TEST_PASS</code>	The test passed. RAM area tested has not been detected to have faults
<code>MARCHC_TEST_FAIL</code>	The test failed. Some RAM area location has been detected to have faults.

**Remarks**

This is a destructive memory test. The test must not be performed over the RAM areas that have to be preserved; otherwise, these RAM areas must be saved/restored before/after running the test.

Alternatively the test could be run at system startup before the memory and the run time library is initialized; however, the stack must be initialized.

At least 100 bytes should be available for the stack for executing the March C test. The tested RAM area must not overlap the stack.

Other statically allocated resources such as the MPLAB ICD or REAL ICE allocated RAM buffers should be excluded from this test.

The start address from which the March C test is to be performed and the size of the RAM area are PIC32 variant and application-dependent. They are run-time parameters.

**Source File**

`SSL_MarchCRamTest.c`

**TABLE 15: RESOURCE REQUIREMENTS**

Parameter	Requirements
Program Memory	1304 bytes (debug build) 724 bytes (-O3 -fomit-frame-pointer)
Stack	100 bytes (debug build)
Execution Time	602228 cycles (7.52785 ms) for 1024 bytes of RAM test <sup>(1)</sup>

**Note 1:** The value for the CPU cycles can vary greatly depending on the system settings and the build parameters. All execution time measurements are done with the following settings: 80 MHz, 2 Flash wait States, 0 RAM wait states, Prefetch and Cache enabled, PBDIV = 1:1 interrupts and DMA disabled. Build: MPLAB C32 V1.10(b) Release -O3 -fomit-frame-pointer.

---

## SSL\_32bitsFamily\_RAM\_STACKtest\_MarchC

---

### Description

This function implements the March C test on both a RAM and a stack area.

First the RAM area is tested using the standard March C test. If the test succeeded the requested Stack area is copied into the RAM area that has just been tested and then the March C test is run over the Stack area as if it were a regular RAM area. The saved Stack area is restored and the result of the test is returned to the user.

This test performs 32-bit word RAM accesses. The address of both the RAM and stack areas to be tested have to be 32-bit aligned and the size of the tested areas must be an integral multiple of 4.

The tested RAM memory will be cleared when the control returns from the `SSL_32bitsFamily_RAM_STACKtest_MarchC` function.

### Include

`SSL_MarchC.h`

### Prototype

```
int SSL_32bitsFamily_RAM_STACKtest_MarchC(int* ramStartAddress, int ramSize, int* stackTopAddress, int stackSize);
```

### Arguments

<code>ramStartAddress</code>	Start address of RAM area for which the March C test is to be performed. Must not overlap the Stack area! Must be properly 32-bit aligned.
<code>ramSize</code>	Number of consecutive byte locations for which the test is to be performed. The size must be a number multiple of 4. The size of the RAM area tested must be > 100 bytes.
<code>stackTopAddress</code>	Address of the top of the Stack area for which the March C test is to be performed. Note that the stack is supposed to grow downward. This must not overlap the RAM area. Must be properly 32-bit aligned.
<code>stackSize</code>	Number of consecutive byte locations in the Stack area for which the test is to be performed. The size must be a number multiple of 4 and must be < RAM size.

### Return Value

<code>MARCHC_TEST_PASS</code>	The test passed. RAM and Stack area tested have not been detected to have faults.
<code>MARCHC_TEST_FAIL</code>	The test failed. Either some RAM or Stack area location has been detected to have faults.
<code>MARCHC_TEST_FAIL_SIZE</code>	The test failed. There was not enough space in the RAM area to save the Stack area.
<code>MARCHC_TEST_FAIL_STACK</code>	The test failed. The requested Stack area does not actually contain the current hardware SP register.

### Remarks

The RAM and Stack areas must not overlap.

The Stack grows downward so the tested area is: `[stackTopAddress-stackSize, stackTopAddress]`

The processor SP register is changed to the point to the RAM area while the Stack area is tested.

The size of the Stack area to be tested must be less than the size of the RAM area.

Since running the March C RAM and Stack test requires at least 128 bytes of stack, it implies that the size of the tested RAM area should be at least 128 bytes long.

Once the Stack area is tested, the SP register is restored.

This is a destructive memory test.

---

**SSL\_32bitsFamily\_RAM\_STACKtest\_MarchC(Continued)**


---

The test must not be performed over the RAM areas that have to be preserved; otherwise, these RAM areas must be saved/restored before/after running the test.

Alternately, the test could be run at system startup before the memory and the run time library is initialized; however, the stack needs to be initialized.

At least 128 bytes should be available to the stack for executing the March C test.

The tested RAM area must not overlap the stack.

Other statically allocated resources, such as the MPLAB ICD or REAL ICE allocated RAM buffers should be excluded from this test.

The start address for both RAM and the stack areas to be tested and the size of these areas are PIC32MX variant and application-dependent. They are run-time parameters.

The standard C language call convention applies regarding the registers usage (refer to **Section 5.6 “Function Calling Convention”** in the MPLAB® C Compiler For PIC32 MCUs User's Guide (DS51686).

Upon the function return the register v0 will contain the return code.

### Source File

SSL\_MarchCStackTest.S

**TABLE 16: RESOURCE REQUIREMENTS**

Parameter	Requirements
Program Memory	1572 bytes (debug build) 992 bytes (-O3 -fomit-frame-pointer)
Stack	128 bytes (debug build)
Execution Time	1208616 cycles (15.1077 ms) for 1024 bytes RAM + 1024 bytes stack test <sup>(1)</sup>

**Note 1:** The value for the CPU cycles can vary greatly depending on the system settings and the build parameters. All execution time measurements are done with the following settings: 80 MHz, 2 Flash wait States, 0 RAM wait states, Prefetch and Cache enabled, PBDIV = 1:1 interrupts and DMA disabled. Build: MPLAB C32 V1.10(b) Release -O3 -fomit-frame-pointer.

---

## SSL\_32bitsFamily\_RAMtest\_MarchC\_Minus

---

### Description

This function implements the March C Minus test. This test performs 32-bit word RAM accesses. The address of the RAM area to be tested must be 32-bit aligned and the size of the tested RAM area must be an integral multiple of 4.

The tested RAM memory will be cleared when the control returns from the `SSL_32bitsFamily_RAMtest_MarchC_Minus` function.

### Include

`SSL_MarchC.h`

### Prototype

```
int SSL_32bitsFamily_RAMtest_MarchC_Minus(int* ramStartAddress, int ramSize);
```

### Arguments

<code>ramStartAddress</code>	Start address from which the March C Minus test is to be performed. Must be properly 32-bit aligned.
<code>ramSize</code>	Number of consecutive byte locations for which the test is to be performed. The size must be a number multiple of 4.

### Return Value

<code>MARCHC_TEST_PASS</code>	The test passed. RAM area tested has not been detected to have faults.
<code>MARCHC_TEST_FAIL</code>	The test failed. Some RAM area location has been detected to have faults.

### Remarks

This is a destructive memory test.

The test must not be performed over the RAM areas that have to be preserved; otherwise, those RAM areas must be saved/restored before/after running the test.

Alternatively, the test could be run at system startup before the memory and the run time library is initialized; however, the stack needs to be initialized.

At least 100 bytes should be available to the stack for executing the March C Minus test.

The tested RAM area must not overlap the stack.

Other statically allocated resources such as the MPLAB ICD or REAL ICE allocated RAM buffers should be excluded from this test.

The start address from which the March C Minus test is to be performed and the size of the RAM area are PIC32 variant and application dependent. They are run-time parameters.



---

**SSL\_32bitsFamily\_RAMtest\_MarchC\_Minus(Continued)**

---

**Source File**

SSL\_MarchCRamTest.c

**TABLE 17: RESOURCE REQUIREMENTS**

Parameter	Requirements
Program Memory	1304 bytes (debug build) 724 bytes (-O3 -fomit-frame-pointer)
Stack	100 bytes (debug build)
Execution Time	526956 cycles (6.58695 ms) for 1024 bytes of RAM test. <sup>(1)</sup>

**Note 1:** The value for the CPU cycles can vary greatly depending on the system settings and the build parameters. All execution time measurements are done with the following settings: 80 MHz, 2 Flash wait States, 0 RAM wait states, Prefetch and Cache enabled, PBDIV = 1:1 interrupts and DMA disabled. Build: MPLAB C32 V1.10(b) Release -O3 -fomit-frame-pointer.

## SSL\_32bitsFamily\_RAMtest\_MarchB

### Description

This function implements the March B test. This test performs 32-bit word RAM accesses. The address of the RAM area to be tested must be 32-bit aligned and the size of the tested RAM area must be an integral multiple of 4.

The tested RAM memory will be cleared when the control returns from the SSL\_32bitsFamily\_RAMtest\_MarchB function.

### Include

SSL\_MarchB.h

### Prototype

```
int SSL_32bitsFamily_RAMtest_MarchB(int* ramStartAddress, int ramSize);
```

### Arguments

ramStartAddress	Start address from which the March B test is to be performed. Must be 32-bit aligned.
ramSize	Number of consecutive byte locations for which the test is to be performed. The size must be a number multiple of 4.

### Return Value

MARCHB_TEST_PASS	The test passed. RAM area tested has not been detected to have faults.
MARCHB_TEST_FAIL	The test failed. Some RAM area location has been detected to have faults.

### Remarks

This is a destructive memory test.

The test must not be performed over the RAM areas that have to be preserved; otherwise, these RAM areas must be saved/restored before/after running the test.

Alternately, the test could be run at system startup before the memory and the run time library is initialized; however, the stack needs to be initialized.

At least 100 bytes should be available to the stack for executing the March B test.

The tested RAM area must not overlap the stack.

Other statically allocated resources such as the MPLAB ICD or REAL ICE allocated RAM buffers should be excluded from this test.

The start address from which the March B test is to be performed and the size of the RAM area are PIC32 variant and application-dependent. They are run-time parameters.

### Source File

SSL\_MarchBRamTest.c

**TABLE 18: RESOURCE REQUIREMENTS**

Parameter	Requirements
Program Memory	1540 bytes (debug build) 612 bytes (-O3 -fomit-frame-pointer)
Stack	100 bytes (debug build)
Execution Time	737622 cycles (9.220275 ms) for 1024 bytes RAM test. <sup>(1)</sup>

**Note 1:** The value for the CPU cycles can vary greatly depending on the system settings and the build parameters. All execution time measurements are done with the following settings: 80 MHz, 2 Flash wait States, 0 RAM wait states, Pre-fetch and Cache enabled, PBDIV=1:1 interrupts and DMA disabled. Build: MPLAB C32 V1.10(b) Release -O3 -fomit-frame-pointer.

---

## SSL\_32bitsFamily\_RAMtest\_CheckerBoard

---

### Description

This function implements the Checkerboard test on the RAM memory.

It writes the checkerboard pattern (0x55555555 followed by 0xAAAAAAAA) to adjacent memory locations starting at ramStartAddress.

It performs the following steps:

1. The content of a 64 bytes memory chunk to be tested is saved in temporary CPU registers.
2. Writes the pattern 0x55555555 followed by 0xAAAAAAAA to adjacent memory locations filling up the 64 bytes memory chunk.
3. It reads the memory chunk adjacent locations and checks that the read-back values match the written pattern. If the values match set the success result and go to step 4; otherwise, set the error result and go to step 6.
4. Writes the inverted pattern 0xAAAAAAAA followed by 0x55555555 to adjacent memory locations filling up the 64 bytes memory chunk.
5. It reads the memory chunk adjacent locations and checks that the read-back values match the written pattern. If the values match set the success result; otherwise, set the error result.
6. The content of the tested 64 bytes memory chunk is restored from the temporary CPU registers.
7. If the result shows error, the test is done and returns.
8. The address pointer is incremented to point to the next sequential 64 bytes memory chunk and the test is repeated from step 1 until all the number of requested memory locations is tested.

### Include

SSL\_CBram.h

### Prototype

```
int SSL_32bitsFamily_RAMtest_CheckerBoard(int* ramStartAddress, int ramSize);
```

### Arguments

ramStartAddress	Start address from which the Checkerboard test is to be performed. Must be properly 32-bit aligned.
ramSize	Number of consecutive byte locations for which the test is to be performed. The size must be a number multiple of 64.

### Return Value

CB_TEST_PASS	The test passed. RAM area tested has not been detected to have stuck bits.
CB_TEST_FAIL	The test failed. Some RAM area location has been detected to have stuck bits.

### Remarks

This is a non-destructive memory test. The content of the tested memory area is saved and restored. The test operates in 64 bytes long memory chunks at a time.

At least 32 bytes should be available to the stack for executing the RAM Checker Board test.

The tested RAM area must not overlap the stack.

The start address from which the Checker Board test is to be performed and the size of the RAM area are PIC32 variant and application-dependent. They are run-time parameters.

The routine accesses one 4 byte RAM word at a time.

The standard C language call convention applies regarding the registers usage (refer to **Section 5.6 “Function Calling Convention”** in the MPLAB® C Compiler For PIC32 MCUs User's Guide (DS51686).

---

## SSL\_32bitsFamily\_RAMtest\_CheckerBoard(Continued)

---

Upon the function entry all registers that need to be preserved are save into the stack. These registers are restored upon the completion of the test.

Upon the function return the register v0 will contain the return code.

### Source File

SSL\_CheckerBoardTest.S

**TABLE 19: RESOURCE REQUIREMENTS**

Parameter	Requirements
Program Memory	776 bytes
Stack	32 bytes
Execution Time	3412 cycles (42.65 $\mu$ s) for 1024 bytes RAM test <sup>(1)</sup>

**Note 1:** The value for the CPU cycles can vary greatly depending on the system settings and the build parameters. All execution time measurements are done with the following settings: 80 MHz, 2 Flash wait States, 0 RAM wait states, Prefetch and Cache enabled, PBDIV = 1:1 interrupts and DMA disabled. Build: MPLAB C32 V1.10(b) Release -O3 -fomit-frame-pointer.

---

## SSL\_32bitsFamily\_CLOCKtest

---

### Description

The CPU Clock test verifies that the system clock is within specified limits.

The SOSC is used as the reference clock.

The function monitors the CPU Core Timer that runs on the CPU system clock.

The test performs the following major steps:

1. The Sosc is used as the independent clock source/reference clock source connected to hardware Timer1.
2. The CPU Core Timer monitored in this measurement is incremented every other CPU system clock.
3. Usually the system runs on the POSC with PLL as the clock source to the CPU. However, any CPU clock source except the SOSC itself which is used as a reference is valid for this test.
4. Timer1 is configured to time out after the specified interval of time elapsed (e.g., 10 ms).
5. The content of the Core Timer is saved at the beginning of the measurement, once Timer1 is started.
6. When the hardware Timer1 times out another reading of the Core Timer is taken and the difference is made with the start value. This difference value represents the number of CPU clock cycles counted by the Core Timer during the SOSC period of time.
7. If this value crosses the defined boundary limits the function returns an appropriate error value, specifying which exactly limit (upper/lower) was violated.

### Include

SSL\_ClockTest.h

### Prototype

```
int SSL_32bitsFamily_CLOCKtest(unsigned int sysClk, int nMs, int hiClkErr, int loClkErr);
```

### Arguments

sysClk	The current system running frequency, Hz
nMs	Number of milliseconds to be used for the CPU Clock monitoring ( $1 \leq nMs \leq 1000$ )
hiClkErr	The upper error limit for the system clock, Hz. A monitored value greater than (sysClk+hiClkErr) will trigger the return of the CLOCK_TEST_FAIL_HI error code.
loClkErr	The lower error limit for the system clock, Hz. A monitored value less than (sysClk-loClkErr) will trigger the return of the CLOCK_TEST_FAIL_LOW error code.

### Return Value

CLOCK_TEST_PASS	The test passed. The monitored CPU clock is within the requested limits.
CLOCK_TEST_FAIL_HI	The test failed. The monitored CPU clock is greater than the specified upper limit.
CLOCK_TEST_FAIL_LOW	The test failed. The monitored CPU clock is less than the specified lower limit.

### Remarks

The test uses the hardware Timer1. It initializes the timer as needed and, after the test is done, shuts off the timer. The previous state of Timer1 is not preserved/restored.

The test assumes that the Core Timer is enabled and linearly counting up as it should do during normal system operation. If the application code specifically disables the Core Timer, it should enable it before this test is called. Also if the value in the Core Timer is updated/changed as part of an ISR, this ISR should be disabled.

The interrupts should be disabled when executing this test as the time spent in ISRs will affect the accurate timing of this routine.

---

## SSL\_32bitsFamily\_CLOCKtest(Continued)

---

The SOSC is used as a reference. The frequency of the signal/crystal connected to this input is defined using the `CLOCK_TEST_SOSC_FREQ` symbol.

The value of the CPU clock monitoring time, nMs, is limited because of the use of the hardware Timer1, which is a 16-bit timer. Therefore, the value loaded into this Timer1 register should not exceed  $2^{16-1}$ .

### Source File

`SSL_ClockTest.c`

**TABLE 20: RESOURCE REQUIREMENTS**

Parameter	Requirements
Program Memory	768 bytes (debug build) 528 bytes (-O3 -fomit-frame-pointer)
Stack	100 bytes (debug build)
Execution Time	16012192 cycles (200.1524 ms) for 100 ms measurement. <sup>(1,2)</sup>

- Note 1:** The maximum execution time is routine execution + 2 SOSC signal periods (1 period for measurement + synchronization time).
- 2:** The value for the CPU cycles can vary greatly depending on the system settings and the build parameters. All execution time measurements are done with the following settings: 80 MHz, 2 Flash wait States, 0 RAM wait states, Prefetch and Cache enabled, PBDIV=1:1 interrupts and DMA disabled. Build: MPLAB C32 V1.10(b) Release -O3 -fomit-frame-pointer.

---

## SSL\_32bitsFamily\_CLOCKtest\_LineFreq

---

### Description

The CPU Clock Line Test verifies that the system clock is within specified limits.

An external frequency applied on the Input Capture 1 pin (IC1) is used as the reference clock.

The hardware Timer2 that runs on the system Peripheral Bus (PB) clock is used to monitor the CPU clock and Peripheral Bus divider.

The test performs the following major steps:

1. The IC1 input is used as the independent clock source/reference clock source to capture the hardware Timer2. An external reference frequency, usually the line frequency, must be applied to the IC1 input pin.
2. The Input Capture 1 is configured as follows:
  - Timer2 is selected as the IC1 time base
  - Capture is performed on every rising edge
  - Capture done event is generated on every second capture
3. The hardware Timer2 prescaler is calculated (based on the input reference frequency and the current PB frequency) as being the smallest divider possible such that the 16 bit Timer2 does not overflow within a period time of the input reference signal.

This way, the best resolution is achieved for the given conditions. If no valid prescaler value can be obtained an error value is returned (the maximum divider for Timer2 is 256).

4. The IC1 performs the capture on every rising edge of the input reference frequency. For period measurement, the capture done event is generated after the IC1 module takes two time stamps (i.e., after every period of the input reference (20 ms if reference frequency is 50 Hz, 16.66ms if the reference frequency is 60 Hz)).
5. Once the capture done event is signaled, the 2 Timer2 captured readings are extracted and the number of elapsed PB clocks is calculated as being the difference between the two readings multiplied by the Timer2 prescaler.

If this value crosses the defined boundary limits the function returns an appropriate error value, specifying which exactly limit (upper/lower) was violated.

#### CALCULATION EXAMPLE 1:

System Clock = 80 MHz

PB Clock = 80 MHz (PB divider = 1:1)

Input Reference = 50 Hz

T2 Min Divider =  $\text{floor}(\text{PBclk}/(65536*\text{RefClk})) + 1 = 25$

Actual T2 Divider = 32

The number of cycles counted by the Timer2 in the Reference clock period is =  $(80,000,000/32)/50 = 50,000$ .

#### CALCULATION EXAMPLE 2:

System Clock = 80 MHz

PB Clock = 10 MHz (PB divider = 1:8)

Input Reference = 60 Hz

T2 Min Divider =  $\text{floor}(\text{PBclk}/(65536*\text{RefClk})) + 1 = 3$

Actual T2 Divider = 4

The number of cycles counted by Timer2 in the Reference clock period is =  $(10,000,000/4)/60 = 41,666$ .

### Include

SSL\_ClockTest\_LineFreq.h

### Prototype

```
int SSL_32bitsFamily_CLOCKtest_LineFreq(unsigned int sysClk, unsigned int lineRefClk,
int hiClkErr, int loClkErr);
```

## SSL\_32bitsFamily\_CLOCKtest\_LineFreq(Continued)

### Arguments

sysClk	The current system running frequency, Hz
lineRefClk	The frequency of the reference applied to the IC1 input pin, Hz. Usual values are 50/60 Hz
hiClkErr	The upper error limit for the system clock, Hz. A monitored value greater than (sysClk+hiClkErr) will trigger the return of the CLOCK_TEST_FAIL_HI error code.
loClkErr	The lower error limit for the system clock, Hz. A monitored value less than (sysClk-loClkErr) will trigger the return of the CLOCK_TEST_FAIL_LOW error code.

### Return Value

CLOCK_TEST_PASS	The test passed. The monitored CPU clock is within the requested limits.
CLOCK_TEST_FAIL_HI	The test failed. The monitored CPU clock is greater than the specified upper limit.
CLOCK_TEST_FAIL_LOW	The test failed. The monitored CPU clock is less than the specified lower limit.
CLOCK_TEST_FAIL_LOW_REF	The test failed. The frequency of the provided reference was too low and could not be used.

### Remarks

The test uses the hardware Input Capture 1 module. It initializes the module as needed and, after the test is done, it shuts it off. The previous state of IC1 is not preserved/restored.

The test uses the hardware Timer2. It initializes the timer as needed and, after the test is done, shuts off the timer. The previous state of Timer1 is not preserved/restored.

The value of the PB frequency which is used as input by the Timer2 is derived from the system CPU clock by dividing it with the PB divider. The test does not change the value of the PB divider, it uses the current value.

The interrupts should be disabled when executing this test as the time spent in ISRs affects the accurate timing of this routine.

The frequency of the signal used as a reference on IC1 input should normally be the line frequency.

However, any frequency can be used as long as a valid Timer2 divider can be obtained (see the example calculation below for the 16-bit Timer2).

If the reference frequency is too low, a valid divider for the Timer2 won't be possible (the maximum divider for Timer2 is 256). A greater PB divider must be selected in this case.

If the selected reference frequency is too high the number of Timer2 captured counts will be too small and the measurement won't have enough resolution.

Source File

SSL\_ClockTest\_LineFreq.c

**TABLE 21: RESOURCE REQUIREMENTS**

Parameter	Requirements
Program Memory	796 bytes (debug build) 488 bytes (-O3 -fomit-frame-pointer)
Stack	48 bytes (debug build)

- Note 1:** The maximum execution time is routine execution + 3 reference signal periods (2 periods for measurement + synchronization time).
- 2:** The value for the CPU cycles can vary greatly depending on the system settings and the build parameters. All execution time measurements are done with the following settings: 80 MHz, 2 Flash wait States, 0 RAM wait states, Prefetch and Cache enabled, PBDIV = 1:1 interrupts and DMA disabled. Build: MPLAB C32 V1.10(b) Release -O3 -fomit-frame-pointer.



---

---

Execution Time	4198140 cycles (52.47675 ms) for 50 Hz reference <sup>(1,2)</sup> .
----------------	---

- Note 1:** The maximum execution time is routine execution + 3 reference signal periods (2 periods for measurement + synchronization time).
- 2:** The value for the CPU cycles can vary greatly depending on the system settings and the build parameters. All execution time measurements are done with the following settings: 80 MHz, 2 Flash wait States, 0 RAM wait states, Prefetch and Cache enabled, PBDIV = 1:1 interrupts and DMA disabled. Build: MPLAB C32 V1.10(b) Release -O3 -fomit-frame-pointer.

## 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](http://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 <sup>1)</sup>	Fault/error	Software class		Acceptable measures <sup>2) 3) 4)</sup>	Definitions
		B	C		
1. CPU 1.1 Registers	Stuck at  DC fault	rq	rq	Functional test, or periodic self-test using either: – static memory test, or – word protection with single bit redundancy Comparison of redundant CPUs by either: – reciprocal comparison – independent hardware comparator, or Internal error detection, or redundant memory with comparison, or periodic self-tests using either – walkpat memory test – Abraham test – transparent GALPAT test; or word protection with multi-bit redundancy, or static memory test and word protection with single bit redundancy	H.2.18.5 H.2.18.6 H.2.19.6 H.2.19.8.2 H.2.18.15 H.2.18.3 H.2.18.9 H.2.19.5 H.2.19.7 H.2.19.1 H.2.19.2.1 H.2.19.8.1 H.2.19.6 H.2.20.8.2
1.2 Instruction decoding and execution	Wrong decoding and execution		rq	Comparison of redundant CPUs by either: – reciprocal comparison – independent hardware comparator, or internal error detection, or periodic self-test using equivalence class test	H.2.18.15 H.2.18.3 H.2.18.9 H.2.18.5
1.3 Programme counter	Stuck at  DC fault	rq	rq	Functional test, or periodic self-test, or independent time-slot monitoring, or logical monitoring of the programme sequence Periodic self-test and monitoring using either: – independent time-slot and logical monitoring – internal error detection, or comparison of redundant functional channels by either: – reciprocal comparison – independent hardware comparator	H.2.18.5 H.2.18.6 H.2.18.10.4 H.2.18.10.2 H.2.18.7 H.2.18.10.3 H.2.18.9 H.2.18.15 H.2.18.3
1.4 Addressing	DC fault		rq	Comparison of redundant CPUs by either: – reciprocal comparison – independent hardware comparator; or Internal error detection; or periodic self-test using a testing pattern of the address lines; or full bit bus parity including the address	H.2.18.15 H.2.18.3 H.2.18.9 H.2.18.7 H.2.18.22 H.2.18.1.1 H.2.18.1.2

TABLE B-1: H.11.12.7 (CONTINUED)

Component <sup>1)</sup>	Fault/error	Software class		Acceptable measures <sup>2) 3) 4)</sup>	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.2.18.7 H.2.18.22 H.2.18.1.2
2. Interrupt handling and execution	No interrupt or too frequent interrupt No interrupt or too frequent interrupt related to different sources	rq	rq	Functional test; or time-slot monitoring  Comparison of redundant functional channels by either reciprocal comparison, independent hardware comparator, or Independent time-slot and logical monitoring	H.2.18.5 H.2.18.10.4  H.2.18.15 H.2.18.3 H.2.18.10.3
3. Clock	Wrong frequency (for quartz synchronized clock: harmonics/ subharmonics only)	rq	rq	Frequency monitoring, or time slot monitoring 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.2.18.10.1 H.2.18.10.4  H.2.18.15 H.2.18.3
4. Memory 4.1 Invariable memory	All single bit faults  99,6 % coverage of all information errors	rq	rq	Periodic modified checksum; or multiple checksum, or word protection with single bit redundancy 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.2.19.3.1 H.2.19.3.2 H.2.19.8.2  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 <sup>1)</sup>	Fault/error	Software class		Acceptable measures <sup>2) 3) 4)</sup>	Definitions
		B	C		
4.2 Variable memory	DC fault  DC fault and dynamic cross links	rq	rq	Periodic static memory test, or word protection with single bit redundancy Comparison of redundant CPUs by either: – reciprocal comparison – independent hardware comparator, or redundant memory with comparison, or periodic self tests using either: – walkpat memory test – Abraham test – transparent GALPAT test, or word protection with multi-bit redundancy	H.2.19.8 H.2.19.8.2  H.2.18.15 H.2.18.3 H.2.19.5  H.2.19.7 H.2.19.1 H.2.19.2.1 H.2.19.8.1
4.3 Addressing (relevant to variable and invariable memory)	Stuck at  DC fault	rq	rq	Word protection with single bit parity including the address, or comparison of redundant CPUs by either: – reciprocal comparison, or – independent hardware comparator, or full bus redundancy Testing pattern, or periodic cyclic redundancy check, either: – single word – double word, or word protection with multi-bit redundancy including the address	H.2.19.18.2   H.2.18.15 H.2.18.3 H.2.18.1.1  H.2.18.22 H.2.19.4.1 H.2.19.4.2 H.2.19.8.1
5. Internal data path  5.1 Data	Stuck at DC fault	rq	rq	Word protection with single bit redundancy Comparison of redundant CPUs by either: – reciprocal comparison – independent hardware comparator, or word protection with multi-bit redundancy including the address, or data redundancy, or testing pattern, or protocol test	H.2.19.8.2  H.2.18.15 H.2.18.3 H.2.19.8.1 H.2.18.2.1 H.2.18.22 H.2.18.14
5.2 Addressing	Wrong address  Wrong address and multiple addressing	rq	rq	Word protection with single bit redundancy including the address Comparison of redundant CPUs by: – reciprocal comparison – independent hardware comparator, or word protection with multi-bit redundancy, including the address, or full bus redundancy; or testing pattern including the address	H.2.19.8.2  H.2.18.15 H.2.18.3 H.2.19.8.1 H.2.18.1.1 H.2.18.22

TABLE B-1: H.11.12.7 (CONTINUED)

Component <sup>1)</sup>	Fault/error	Software class		Acceptable measures <sup>2) 3) 4)</sup>	Definitions
		B	C		
6 External communication	Hamming distance 3	rq		Word protection with multi-bit redundancy, or CRC – single word, or transfer redundancy, or protocol test	H.2.19.8.1 H.2.19.4.1 H.2.18.2.2 H.2.18.14
6.1 Data	Hamming distance 4		rq	CRC – double word, or data redundancy or comparison of redundant functional channels by either: – reciprocal comparison – independent hardware comparator	H.2.19.4.2 H.2.18.2.1 H.2.18.15 H.2.18.3
6.2 Addressing	Wrong address		rq	Word protection with multi-bit redundancy, including the address, or CRC single word including the addresses, or transfer redundancy or protocol test	H.2.19.8.1 H.2.19.4.1 H.2.18.2.2 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: – reciprocal comparison – independent hardware comparator	H.2.19.4.2 H.2.18.1.1 H.2.18.15 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 comparison of redundant communication channels by either: – reciprocal comparison – independent hardware comparator	H.2.18.10.3 H.2.18.15 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 – independent hardware comparator, or	H.2.18.15 H.2.18.3
7.1 Digital I/O				input comparison, or multiple parallel outputs; or output verification, or testing pattern, or code safety	H.2.18.8 H.2.18.11 H.2.18.12 H.2.18.22 H.2.18.2

**TABLE B-1: H.11.12.7 (CONTINUED)**

Component <sup>1)</sup>	Fault/error	Software class		Acceptable measures <sup>2) 3) 4)</sup>	Definitions
		B	C		
7.2 Analog I/O	Fault conditions specified in H.27	rq	rq	Plausibility check	H.2.18.13
7.2.1 A/D- and D/A- convertor				Comparison of redundant CPUs by either: – reciprocal comparison – independent hardware comparator, or input comparison, or multiple parallel outputs, or output verification, or testing pattern	H.2.18.15 H.2.18.3 H.2.18.8 H.2.18.11 H.2.18.12 H.2.18.22
7.2.2 Analog multiplexer	Wrong addressing	rq	rq	Plausibility check  Comparison of redundant CPUs by either: – reciprocal comparison – independent hardware comparator, or input comparison or testing pattern	H.2.18.13  H.2.18.15 H.2.18.3 H.2.18.8 H.2.18.22
8. Monitoring devices and comparators	Any output outside the static and dynamic functional specification		rq	Tested monitoring, or redundant monitoring and comparison, or error recognizing means	H.2.18.21 H.2.18.17 H.2.18.6
9. Custom chips <sup>5)</sup> e.g. ASIC, GAL, Gate array	Any output outside the static and dynamic functional specification	rq	rq	Periodic self test  Periodic self-test and monitoring, or dual channel (diverse) with comparison, or error recognizing means	H.2.16.6  H.2.16.7 H.2.16.2 H.2.18.6
<p>CPU: Central Programming Unit</p> <p>rq: Coverage of the fault is required for the indicated software class.</p> <p><sup>1)</sup> For fault/error assessment, some components are divided into their subfunctions.</p> <p><sup>2)</sup> For each subfunction in the table, the software class C measure will cover the software class B fault/error.</p> <p><sup>3)</sup> It is recognized that some of the acceptable measures provide a higher level of assurance than is required by this standard.</p> <p><sup>4)</sup> Where more than one measure is given for a subfunction, these are alternatives.</p> <p><sup>5)</sup> To be divided as necessary by the manufacturer into subfunctions.</p> <p><sup>6)</sup> Table H.11.12.7 is applied according to the requirements of H.11.12 to H.11.12.13 inclusive.</p>					



## APPENDIX C: REVISION HISTORY

### Revision A (September 2008)

This is the initial release of this application note.

### Revision B (May 2010)

Information related to 32-bit PIC microcontrollers has been added throughout the document, which includes the following new functions:

- `SSL_32bitsFamily_CPU_RegisterTest`
- `SSL_32bitsFamily_PCtest`
- `SSL_32bitsFamily_Flashtest_CRC16`
- `SSL_32bitsFamily_RAMtest_MarchC`
- `SSL_32bitsFamily_RAM_STACKtest_MarchC`
- `SSL_32bitsFamily_RAMtest_MarchC_Minus`
- `SSL_32bitsFamily_RAMtest_MarchB`
- `SSL_32bitsFamily_RAMtest_CheckerBoard`
- `SSL_32bitsFamily_CLOCKtest`
- `SSL_32bitsFamily_CLOCKtest_LineFreq`

NOTES:

---

**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.

---

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

#### **Trademarks**

The Microchip name and logo, the Microchip logo, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, PIC<sup>32</sup> logo, rfPIC and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Octopus, Omniscient Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICkit, PICTail, REAL ICE, rLAB, Select Mode, Total Endurance, TSHARC, UniWinDriver, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2010, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

ISBN: 978-1-60932-172-7

*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.*

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



---

## 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](http://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

#### Cleveland

Independence, OH  
Tel: 216-447-0464  
Fax: 216-447-0643

#### 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 - Chongqing

Tel: 86-23-8980-9588  
Fax: 86-23-8980-9500

#### 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 - Xian

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

#### China - Xiamen

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

#### China - Zhuhai

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

### ASIA/PACIFIC

#### India - Bangalore

Tel: 91-80-3090-4444  
Fax: 91-80-3090-4123

#### 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: 886-3-6578-300  
Fax: 886-3-6578-370

#### 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

01/05/10