8-Bit

Microcontroller

AP08089

XC878 Class B Software Library

Application Note

V 1.3 2009-02

Microcontrollers

Edition 2009-02

Published by
Infineon Technologies AG
81726 Munich, Germany

XC878 Series Application Note

Revision History: V 1.3 2009-02

Previous Versions:

Page	Subjects (major changes since last revision)
–	
–	

Table of Contents

Page

1 Introduction . . . 3
1.1 Purpose . . . 3
1.2 Software Library Certification According to Class B . . . 3
1.3 Acronyms Abbreviations and Special Terms . . . 3
1.4 References . . . 3

2 Overview . . . 4
2.1 IEC60730 Standard Compliance . . . 4
2.1.1 IEC60730 Annex H Standard . . . 4
2.1.1.1 Software Controls . . . 4
2.1.1.2 Components to be Tested For Single-Chip MCU . . . 4
2.2 ClassB Software Library for XC878-16FF Microcontroller . . . 5
2.2.1 XC878 Competitive Advantages . . . 5

3 CPU Registers Test . . . 8
3.1 Test Routine . . . 8

4 SFRs Tests . . . 10
4.1 Timer Tests . . . 10
4.1.1 Test Routines . . . 10
4.1.1.1 Timer0 and Timer1 Test Routines . . . 10
4.1.1.2 Timer2 Test Routine . . . 11
4.2 SSC Test . . . 13
4.2.1 Test Routine . . . 13
4.3 GPIO Test . . . 14
4.3.1 Test Routine . . . 14
4.4 UART0 Test . . . 16

5 CPU Program Counter Test . . . 18
5.1 Test Routines . . . 18
5.1.1 Enable WDT Routine . . . 18
5.1.2 Refresh WDT Routine . . . 19
5.1.3 Forced WDT Reset Routine . . . 20

6 Invariable Memory Test . . . 21
6.1 PFlash ECC Logic Test Routine . . . 21
6.2 DFlash ECC Logic Test Routine . . . 21

7 Variable Memory Test . . . 24
7.1 MarchC Memory Test Routines . . . 24
7.1.1 MarchC Algorithm . . . 24
7.1.2 IRAM Test At Startup . . . 24
7.1.3 IRAM Test At Runtime . . . 25
7.1.4 XRAM Test At Startup . . . 26
7.1.5 XRAM Test At Runtime . . . 26

Table of Contents **Page**

7.2 MarchX Memory Test Routines 27
7.2.1 MarchX Algorithm 27
7.2.2 IRAM Test At Runtime 27
7.2.3 XRAM Test At Runtime 28

8 System Framework 29
8.1 CANscheduler Operation Overview 29
8.2 XC878 Starter Kit Setting and Modification 30
8.3 Resources Requirements 32
8.4 Flowcharts of the CANscheduler 32

1 Introduction

This document describe the Class B Software Library implemented for the XC878-16FF microcontroller chip with 64K Flash. The specification is organised into the following major sections

- Overview
- Descriptions of each component in the Software Library
- An example of a working framework which incorporates the Class B Software Library test routines.

1.1 Purpose

The document forms the basis for the implementation of the Class B software library in a user application.

1.2 Software Library Certification According to Class B

The Software Library test routines described can be used for microcontroller internal supervisory functions and for self-diagnostics. They fulfill the requirements according to the Class B standard and were approved by VDE (reference number 5007865-9999-0001/112626). The implementation has to be tested in each application.

A quick start step by step testing guide on Software Library will be provided to the user upon request.

1.3 Acronyms Abbreviations and Special Terms

List of terms and abbreviations used throughout the document:

• ECC	Error Checking and Correction
• GPIO	General Purpose Input / Output
• MCU	Microcontroller Unit
• SSC	Synchronous Serial Communication
• SFR	Special Function Register
• WDT	Watchdog Timer

1.4 References

1. IEC60730 Annex H -Requirements for Electronic Controls
2. IEC60335-1 Annex R - Software Evaluation
3. XC878 User Manual version 1.0

2 Overview

This document includes the description of the API for each user routine provided in the software library.

2.1 IEC60730 Standard Compliance

From Oct 2007, home appliances to be sold in Europe have to comply with IEC60730 standard. For MCU, the IEC60730 Annex H explains the detail of the tests and diagnostic methods to ensure safe operation of embedded control hardware and software for household appliances.

2.1.1 IEC60730 Annex H Standard

This standard documents the requirements for electronic controls. It contains detailed tests and diagnostic methods to ensure the safe operation of embedded control hardware and software for household appliances.

2.1.1.1 Software Controls

Structure of Control

- Single Channel with functional test structure
- Single Channel with periodic self test - periodically check various critical functions without conflicting with end user application operation.

Software Classification

IEC60730 Annex H has 3 software classifications for automatic electronic controls:

- Class A - Not intended to be relied upon for the safety of the equipment. Examples: humidity controls, lighting controls, timers.
- Class B - Intended to prevent unsafe operation of the controlled equipment. Examples: thermal cut-offs and door locks for laundry equipment.
- Class C - Intended to prevent special hazards, like explosion of the controlled equipment. Example: automatic burner controls, gas fired controlled dryer.

For our device to be used in home appliances, it has to fulfill Class B requirements.

2.1.1.2 Components to be Tested For Single-Chip MCU

Manufactures of electronics controls are required to test 14 components, but only 10 of those components are relevant to a single-chip MCU, as listed in **Table 2-1**.

The Software Library is developed to cover 6 components. The other 4 components are to be implemented in the application code.

The numbering in the first column of the table makes reference to the components numbered in IEC60730 Annex H table H.11.12.7.

Table 2-1 Components to be tested

	Component	Fault / Error
1.1	CPU registers / SFR registers	Stuck at fault
1.3	Programme counter	Stuck at fault
2	Interrupt handling and execution[1)]	No Interrupt or too frequent Interrupt
3	CPU clock[1)]	Wrong frequency
4.1	Invariable memory	All single bit faults
4.2	Variable memory	DC fault
4.3	Addressing	Stuck at fault
5.1	Internal Data Path	Stuck at fault
6	External Communications[1)]	Hamming distance 3
6.3	Timing[1)]	Wrong point in time/sequence

1) To be implemented in user application code.

2.2 ClassB Software Library for XC878-16FF Microcontroller

The Software Library provides self test routines which the user can call at system startup or periodically at system run time. **Figure 2-1** shows the overview of the Class B Software Library. **Table 2-2** shows the mapping of requirements to the self test routines implemented in the Library.

2.2.1 XC878 Competitive Advantages

Dedicated safety features of the XC878 microcontroller family offer significant competitive advantages. In particular, the embedded flash module with its hardware error correction (ECC), and the invariable memory tests which are done without the need to implement the time consuming CRC-memory checker routines. The ECC can correct single bit error and can inherently signal such events to the application with every flash access. This increases CPU performance, frees-up memory space and makes user software easier and safer.

The XC878 microcontroller comes with a sophisticated clock supervisory feature. The clock control with it’s on-chip oscillator and PLL, can detect clock faults such as the loss of lock, or double and half frequency. If clock failure occurs, the system is automatically brought into a safe-state and a signal is sent to the event application.

The features described make an application safer without additional cost and overhead.

Table 2-2 Requirements Matrix

Description	Self Test Routines
Compliance to IEC60730 ClassB Annex H	CPU_Registers_Test(), Enable_WDT(), Refresh_WDT(), Forced_WDT_Reset(), PFlash_ECC_Logic_Test(), DFlash_ECC_Logic_Test(), IRAM_MarchC_ST_Test(), XRAM_MarchC_ST_Test(), IRAM_MarchC_RT_Test(), XRAM_MarchC_RT_Test(), IRAM_MarchX_RT_Test(), XRAM_MarchX_RT_Test().
CPU SFRs test (Optional to application)	Timer0_Test(), Timer1_Test(), Timer2_Test(), SSC_Test(), GPIO_Test(), UART_Test()

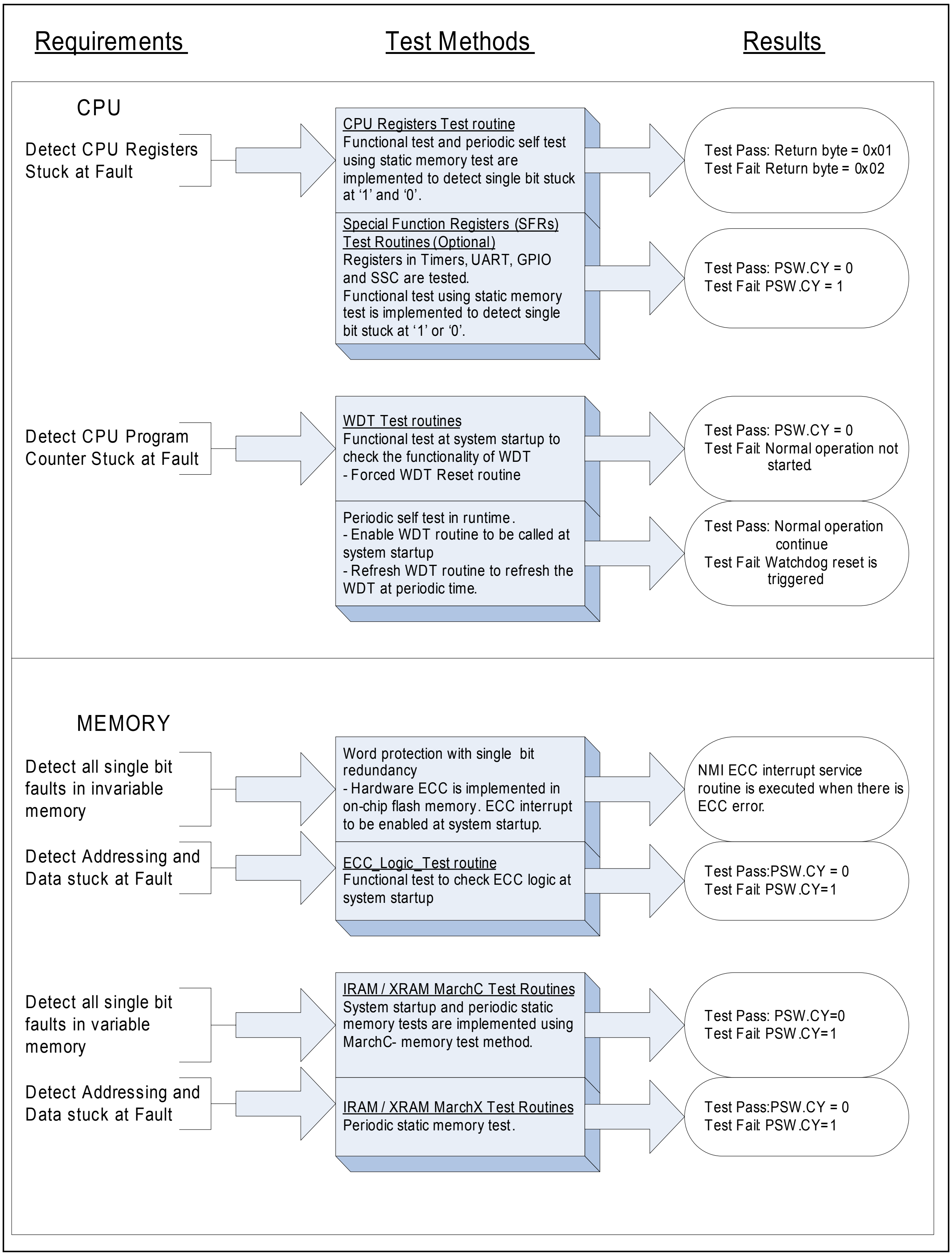
Requirements
Test Methods
Results
CPU
Detect CPU Registers Stuck at Fault
CPU Registers Test routine
Functional test and periodic self test using static memory test are implemented to detect single bit stuck at '1' and '0'.
Test Pass: Return byte = 0x01
Test Fail: Return byte = 0x02
Special Function Registers (SFRs) Test Routines (Optional)
Registers in Timers, UART, GPIO and SSC are tested.
Functional test using static memory test is implemented to detect single bit stuck at '1' or '0'.
Test Pass: PSW.CY = 0
Test Fail: PSW.CY = 1
Detect CPU Program Counter Stuck at Fault
WDT Test routines
Functional test at system startup to check the functionality of WDT
- Forced WDT Reset routine
Test Pass: PSW.CY = 0
Test Fail: Normal operation not started
Periodic self test in runtime.
- Enable WDT routine to be called at system startup
- Refresh WDT routine to refresh the WDT at periodic time.
Test Pass: Normal operation continue
Test Fail: Watchdog reset is triggered
MEMORY
Detect all single bit faults in invariable memory
Word protection with single bit redundancy
- Hardware ECC is implemented in on-chip flash memory. ECC interrupt to be enabled at system startup.
NMI ECC interrupt service routine is executed when there is ECC error.
Detect Addressing and Data stuck at Fault
ECC_Logic_Test routine
Functional test to check ECC logic at system startup
Test Pass:PSW.CY = 0
Test Fail: PSW.CY=1
Detect all single bit faults in variable memory
IRAM / XRAM MarchC Test Routines
System startup and periodic static memory tests are implemented using MarchC- memory test method.
Test Pass: PSW.CY=0
Test Fail: PSW.CY=1
Detect Addressing and Data stuck at Fault
IRAM / XRAM MarchX Test Routines
Periodic static memory test.
Test Pass:PSW.CY = 0
Test Fail: PSW.CY=1

Figure 2-1 Class B Software Library Overview

3 CPU Registers Test

The following CPU core registers are tested:

- Accumulator
- B Register
- Data Pointers, DPTR0 and DPTR1
- Program Status Word

A CPU registers test routine is created to test these core registers. The routine can be called in a startup test and during a periodic test routine.

*Note: Register banks are in IRAM and are therefore not tested in this routine. They are tested in the variable memory test. See **Chapter 7**.*

3.1 Test Routine

The test will check the CPU core registers for stuck at ‘1’ and stuck at ‘0’ faults.

This test is non-destructive. The registers contents are saved into stack before the test is run, and then restored on completion.

Steps to test the registers:

- Storing the register content into stack
- Clearing the register contents.
- Writing 0xAA
- Reading back the contents from register and compare
- Writing the inverse data into the register
- Reading back the contents and compare.
- Restoring the register content from stack
- Testing the next register

This will detect single bit stuck at 1 and stuck at 0 errors.

Precondition before calling this routine:

- Disabled interrupts

Table 3-1 CPU Core Register Test Routine

Routine	**CPU_registers_Test()**
Inputs	-
Return	R7 of current register bank 0x01 - Test Passed 0x02 - Test Failed
Stack Requirements	5

Table 3-1 CPU Core Register Test Routine (cont'd)

Routine	**CPU_registers_Test()**
Memory destroyed	R7
Execution Time	9.75usec

4 SFRs Tests

In addition to CPU Core registers, the following SFRs are tested:

- Timer registers
- SSC registers
- GPIO registers
- UART0 registers

The recommendation is to run the tests during system startup.

Note that for GPIO test, customisation is required to implement the tests in the user application.

4.1 Timer Tests

List of timer 0, timer 1 and timer 2 registers / flags that are tested.

- Timer registers - THx and TLx (x=0,1), T2L, T2H => Test for Stuck at fault
- Timer run control flags - TCON.TRx (x=0,1), T2CON.TR2 => Functional Test
- Timer overflow flags - TCON.TFx (x=0,1), T2CON.TF2 => Functional Test

4.1.1 Test Routines

Three routines will be provided:

- Timer0_Test()
- Timer1_Test()
- Timer2_Test()

4.1.1.1 Timer0 and Timer1 Test Routines

In the Timer0 and Timer1 test routines, THx and TLx registers are tested for stuck at ‘1’ and stuck at ‘0’ fault. After that, the timer is set to 1usec. A software timeout is set to prevent system hang inside the routine. The timer overflow flag is polled until overflow is detected or until a software timeout.

Testing Methods:

- Test Timer registers, THx and TLx for stuck at ‘1’ and stuck at ‘0’ faults. If error, set timer registers back to reset value and return fail
- Initialise Timer.
- Start the timer
- Keep polling until overflow or software timeout
- If timer overflow, set Timer registers back to reset value and return pass
- If software timeout, set Timer registers back to reset value and return fail

Precondition

- Interrupt for the timer under test is disabled

- Peripheral clock, F_{PCLK} = 24MHz.
- SFR SYSCON0.RMAP = 0, access non-mapped SFR area

Table 4-1 Timer0_Test Routine

Routine	--: Timer0_Test
Input	-
Output	PSW.CY 0 = Test Passed 1 = Test Failed
Stack size required	0
Resource used/ destroyed	Timer0 registers, ACC and R0 set to reset values.
Execution time	4.1usec

Table 4-2 Timer1_Test Routine

Routine	--: Timer1_Test
Input	-
Output	PSW.CY 0 = Test Passed 1 = Test Failed
Stack size required	0
Resource used/ destroyed	Timer1 registers, ACC and R0 set to reset values
Execution time	4.1usec

4.1.1.2 Timer2 Test Routine

In the Timer2 test routine, T2H and T2L registers are tested for stuck at ‘1’ and stuck at ‘0’ fault. After that, the timer is set to 1usec, with F_{PCLK} = 24MHz and SFR bit CF_MISC.T2CCFG=0. A software timeout is set to prevent system hang inside the routine. The timer overflow flag is polled until overflow is detected or until a software timeout.

Precondition

- Interrupt for the timer2 is disabled
- Peripheral clock, F_{PCLK} = 24MHz and SFR bit CF_MISC.T2CCFG=0.
- SFR SYSCON0.RMAP = 0, access non-mapped SFR area

Table 4-3 Timer2_Test Routine

Routine	--: Timer2_Test
Input	-
Output	PSW.CY 0 = Test Passed 1 = Test Failed
Stack size required	0
Resource used/ destroyed	Timer2 registers, ACC and R0 set to reset values
Execution time	4.3usec

4.2 SSC Test

This is to test the functionality of SSC using half duplex mode. No SSC data is being sent out through the GPIO as the SSC are not mapped to the I/O ports.

The registers and flags that are tested:

- RBL, receive buffer register
- TBL, transmit buffer register.
- TIR and RIR flags

4.2.1 Test Routine

Testing Method

- Send data 0xAA
- Poll receive interrupt status flag. If timeout, return error
- Check receive data. Return error if receive data is different from send data
- Clear TIR and RIR flags
- Send data 0x55
- Poll receive interrupt status flag. If timeout return error
- Check receive data. Return error if receive data is different from send data
- Clear TIR and RIR flags
- Set SSC registers back to reset values.
- Return Pass

Precondition

- SSC interrupt and Timer0 interrupt disabled.

- Peripheral clock, F_{PCLK} = 24MHz.

- SFR SYSCON0.RMAP = 0, access non-mapped SFR area

- SSC ports are not mapped to GPIO ports.

Table 4-4 SSC_test Routine

Routine	--: SSC_Test
Input	-
Output	PSW.CY 0 = Test Passed 1 = Test Failed
Stack size required	2
Resource used/ destroyed	SSC registers, Timer0 registers and ACC set to reset values
Execution time	10usec

4.3 GPIO Test

This test will check the general purpose input / output ports registers. It will check for stuck at ‘1’ and stuck at ‘0’ faults. The test is to be done at system startup and before GPIO initialisation.

The user can select which port to be tested by changing the following parameters in the GPIO_Test.h file:

• P0_SELECT	EQU	0xF8	;P0.3-P0.7 to be tested
• P1_SELECT	EQU	0xE4	;P1.2,P1.5-P1.7, to be tested
• P3_SELECT	EQU	0xFF	;ALL to be tested
• P4_SELECT	EQU	0xFF	;ALL to be tested
• P5_SELECT	EQU	0xFF	;ALL to be tested

Note: Set a bit to ‘1’ to indicate the port pin to be tested.

4.3.1 Test Routine

Testing Method

For port direction registers, each port.pin is written with ‘1’ and ‘0’. After each write, the register was read back to check the data.

For PUDSEL and DATA registers, the tested I/O port pins are set to input ports. Each port pin is tested by changing the pull-up/pull down. When pull up is selected, a ‘1’ is expected to be read from the respective bit in the data register. When pull down is selected, a ‘0’ is expected.

- Set port pin to input
- Select pull up
- Wait 43usec
- Read port data register and expect ‘1’ on the port pin
- Select pull down
- Wait 43usec
- Read port data register and expect ‘0’ on the port pin
- Repeat for other ports to be tested.

Precondition

- SFR SYSCON0.RMAP = 0, access non-mapped SFR area

Table 4-5 GPIO_test Routine

Routine	--: GPIO_Test
Input	-

Table 4-5 GPIO_test Routine (cont'd)

Output	PSW.CY 0 = Test Passed 1 = Test Failed
Stack size required	--
Resource used/ destroyed	GPIO registers, PORT_PAGE, ACC and R0 set to reset values
Execution time	105usec

4.4 UART0 Test

At system startup, because communication with an external host is not possible, the only tests made are on the functionality of the transmit flag and whether the SCON register is stuck at faults. No data is being sent out through GPIO as UART are not mapped to the I/O ports.

Timer0 is used as timeout to prevent system hang inside the test routine.

Testing Method:

Test SCON register for stuck at '1' and stuck at '0' fault:

- Write 0x55 to register
- Read back
- Write 0xAA
- Read back

Test UART0 transmit flag:

- Setup UART, refer to UART Registers settings.
- Send data 0xAA
- Start Timer0
- Wait for TX flag to be set and check Timer0 overflow flag. If Timer0 overflow and TX flag not set, return error
- Stop Timer0
- Clear TX flag
- Return pass

Precondition:

- UART interrupt to be disabled.

- Peripheral clock, F_{PCLK} = 24MHz.

- SFR SYSCON0.RMAP=0, access non-mapped SFR area

- UART0 ports are not mapped to GPIO ports.

Table 4-6 UART_test Routine

Routine	--: UART_Test
Input	-
Output	PSW.CY 0 = Test Passed 1 = Test Failed

Table 4-6 UART_test Routine (cont'd)

Stack size required	2
Resource used/ destroyed	UART registers: SCON, BG, TX flag, and timer0 registers, SCU_PAGE and ACC are set to reset values.
Execution time	44.5usec

5 CPU Program Counter Test

The XC878-16FF has a Watchdog Timer (WDT) feature. The WDT provides a reliable and secure way to detect and recover from software or hardware failure. When the WDT is enabled, it will cause the XC878 system to be reset if it is not refresh within a specified time. If the program counter is stuck at one address, then a refresh of the WDT will not occur and result in WDT timer overflow and a reset.

5.1 Test Routines

Three routines are provided in the Software Library to check the functionality of the WDT:

- Enable WDT
- Refresh WDT
- Forced WDT reset

5.1.1 Enable WDT Routine

The Watchdog window time period, P_{WDT}, is calculated from the input frequency and reload value.

- **Input frequency** to the Watchdog Timer can be selected via bit WDTIN in register WDTCON to be either $f_{PCLK}/2$ or $f_{PCLK}/128$.
- **Reload value** WDTREL for the high byte of WDT can be programmed in register WDTREL.

$$P_{WDT} = \frac{2^{(1+WDTIN \times 6)} \times (2^{16} - WDTREL \times 2^{8})}{f_{PCLK}} \tag{5.1}$$

The Watchdog Timer has a ‘programmable window boundary’, it disallows refresh during the Watchdog Timer’s count-up. A Refresh during this window-boundary will cause the Watchdog Timer to activate WDTRST. The window boundary is from 0000_H to (WDTWINB,00H).

In this Enable_WDT routine, the window boundary is set to half of the P_{WDT}. If P_{WDT} is 10msec, the first 5msec is the window boundary where no refresh is allowed. The window boundary is configurable by changing the setting of SFR WDTWINB in this routine.

The SFR WDTCON.WDTEN bit, which is used to enable or disable the WDT, is a protected bit. This means that when the protection scheme is active, this bit cannot be written directly. Please refer to the XC878 user manual for a detail description of the protection scheme.

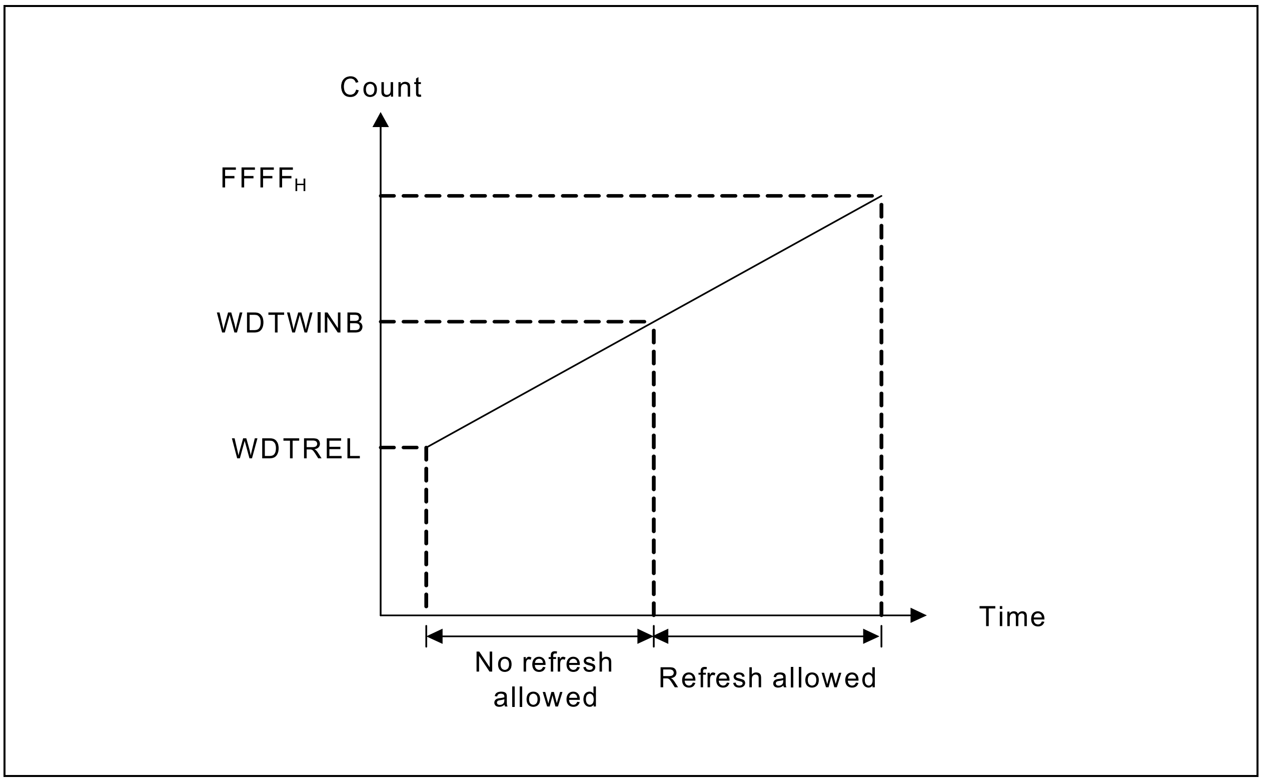

Figure 5-1 Watchdog Timer Timing Diagram

Table 5-1 Enable WDT Routine

Routine	**Enable_WDT()**
Inputs	1. R7 - Input Frequency, R7.bit0 = 0, $F_{PCLK}/2$ R7.bit0 = 1, $F_{PCLK}/128$ other bits of R7 is ignored 2. R5 - Reload value, WDTREL
Return	-
Stack Requirements	0
Memory destroyed	SFR SYSCON0.RMAP
Execution Time	3usec

5.1.2 Refresh WDT Routine

This routine is to be called in the main application programme to refresh the WDT. If the watchdog refresh is performed within the window boundary, a watchdog reset will occur.

Table 5-2 Refresh WDT Routine

Routine	**Refresh_WDT()**
Inputs	-
Return	-
Stack Requirements	0
Memory destroyed	-
Execution Time	1.1usec

5.1.3 Forced WDT Reset Routine

This routine is recommended to be called in the system startup. It is split into two parts. The first part of the routine is executed if system reset is not triggered by WDT reset. It will enable the WDT and hang in an endless loop to force WDT reset to occur.

The second part of the routine is executed if system reset is caused by a WDT reset. It will exit the test routine with the carry flag set to 0.

Note: The WDT reset indication bit, SFR bit PMCON0.WDTRST, is not cleared in this routine.

Precondition:

All interrupts are disabled.

Table 5-3 Forced WDT Reset Routine

Routine	**Forced_WDT_reset()**
Inputs	1. R7 - Input Frequency, R7.bit0 = 0, F_{PCLK}/2 R7.bit0 = 1, F_{PCLK}/128 other bits of R7 is ignored 2. R5 - Reload value, WDTREL
Return	PSW.CY 0 = Watchdog reset is triggered
Stack Requirements	2
Memory destroyed	
Execution Time	-

6 Invariable Memory Test

Infineon XC800 microcontroller on-chip flash memory has hardware ECC. The invariable memory test will check the hardware ECC logic to ensure its functionality. There are 2 test routines, one for PFlash ECC logic and one for DFlash ECC logic

6.1 PFlash ECC Logic Test Routine

Reading an erased Pflash memory will trigger ECC error. This can be used to test the ECC logic for Pflash. The test routine will read the memory location which has erased data. If ECC error is detected, the routine will return 'pass'.

Preconditions:

- Data content at that memory address to be read is erased.
- Disabled ECC interrupt

The ECC Logic Test:

- Read the memory location
- Check flash ECC status, if ECC not triggered, return error
- Return pass

Table 6-1 PFlash ECC Logic Test Routine

Routine	**PFlash_ECC_Logic_Test()**
Inputs	R6 (MSB), R7(LSB) - Memory address where content is erased

Return	PSW.CY, Carry flag
	CY = 0 - Test Passed, ECC error detected.
	CY = 1 - Test Failed, no ECC error detected.

Stack Requirements	1
Memory destroyed	SFR FCS, ACC and DPTR
Execution Time	2.41usec

6.2 DFlash ECC Logic Test Routine

The test routine will read 2 bytes starting from the input memory address, ADDR. If both are '0xFF" and no ECC error is triggered, it is assumed the contents are erased and Dflash programming will be executed twice to generate corrupted data.

The first Dflash programming will program both addresses, ADDR and ADDR+1, with data "0x8A". The second programming will only program 1 byte at address, ADDR+1 with "0x88". With that the content at address, ADDR+1, will be corrupted.

After the second flash programming, a read operation on the memory address, ADDR+1, is executed. If ECC is triggered, the routine will return “pass”, otherwise return an error.

If the content of the input memory address is corrupted by the previous execution of the Dflash ECC Logic Test, no flash programming will be done. If read operation on the memory address, ADDR+1, triggered an ECC error, the routine will return “pass”.

Steps to check DFlash ECC logic:

1. Clear ECC status.
2. Read the memory address, ADDR and check content
3. If ADDR= 0x8A, goto step 6
4. Else if both content= 0xFF, called BootROM flash programming twice. Goto step 6
5. Else return error.
6. Read memory address, ADDR+1 and check ECC status
7. If ECC is triggered, return pass
8. Else return error

Preconditions:

- Data content in memory address, ADDR and ADDR+1, are erased. Or
- Data content in Memory address is corrupted by previous execution of DFlash_ECC_Logic_Test().
- Disabled ECC interrupt

Table 6-2 DFlash ECC Logic Test Routine

Routine	**DFlash_ECC_Logic_Test()**
Inputs	ADDR: R6 (MSB),R7(LSB)-Memory address where content of the first 2 bytes are erased OR R6 (MSB),R7(LSB)-Memory address where content is corrupted by previous execution of DFlash_ECC_Logic_Test().
Return	PSW.CY, Carry flag CY = 0 - Test Passed, ECC error detected CY = 1 - Test Failed, ECC error not detected or input memory not erased or corrupted.
Stack Requirements	12

Table 6-2 DFlash ECC Logic Test Routine (cont'd)

Routine	**DFlash_ECC_Logic_Test()**
Memory destroyed	With inputs memory address contents being erased, BootROM Flash programming is called and the following memories are destroyed: - XRAM memory, address 0xF000 and 0xF001 - IRAM memory, address 0x37 to 0x3E - Current register bank, R0 - R7. - ACC, DPTR0, DPTR1, MEX1, FCS. - Set MEX3 = 0x1F
Execution Time	3.15usec or 200msec with BootROM Dflash programming executed.

7 Variable Memory Test

This chapter described the variable memory tests in software library. The variable memory is referred to the volatile memory. In our system, IRAM and XRAM will be tested. Two types of memory test are provided to check the RAM in the MCU.

- MarchC
- MarchX

In order to detect the bit coupling fault, both the MarchC and MarchX test routines are implemented based on the physical layout of the IRAM and XRAM.

7.1 MarchC Memory Test Routines

This algorithm is based on the MarchC algorithm by Van De Goor, 1991. Four user routines are provided, to test IRAM and XRAM in startup and runtime.

7.1.1 MarchC Algorithm

MarchC test can find stuck-at fault, addressing fault, transition fault and coupling fault. The startup tests are destructive, i.e. all data in the memory under test is destroyed. These tests are to be called at system startup, before the memories are initialised. It will test the complete memory. Running the test in small memory blocks will reduce its capability to detect address decoder faults.

The runtime tests are run in blocks. The data of the memory under test are preserved by storing the contents in XRAM area.

The following is a list of steps in the MarchC memory test:

1. Write all zeros to memory under test.
2. Starting at lowest address, read zeros, write ones, increment address.
3. Starting at lowest address, read ones, write zeros, increment address.
4. Starting at highest address, read zeros, write ones, decrement address.
5. Starting at highest address, read ones, write zeros, decrement address.
6. Read all zeros from memory.

7.1.2 IRAM Test At Startup

This test routine runs the MarchC test on the complete IRAM; i.e. from address 0x00 to 0xFF. Because the test is destructive, the return address is stored in the data address pointer, DPTR, at the start of the routine. Before exiting the test routine, the return address is pushed back to the stack area.

The content of PSW register is changed and register bank 0 is selected.

Precondition:

- All interrupts are disabled.

- IRAM not initialised.

Table 7-1 IRAM MarchC Startup Test Routine

Routine	--: IRAM_MarchC_ST_Test
Input	-
Output	PSW.CY 0 = Test Passed 1 = Test Failed
Stack size required	0
Resource used/destroyed	PSW, EO, DPTR1, ACC All IRAM under test is cleared to '0'
Execution time	620usec

7.1.3 IRAM Test At Runtime

This routine tests the memory in block, where the block length is to be specified by the user. This test is non-destructive; i.e. The IRAM data under test is stored into XRAM before the MarchC test is run, and then the IRAM data is restored before the return to user code.

Attention: Current register bank (R0-R7) address and stack area should not be tested.

Precondition:

All interrupts are disabled.

Table 7-2 IRAM MarchC Runtime Test Routine

Routine	--: IRAM_MarchC_RT_Test
Input	1. R7 - Start address of IRAM to be tested 2. R4(MSB), R5(LSB) - XRAM start address to store the IRAM data 3. R3 - Number of bytes to be tested, range 1 to X, where X = 256 - stack area - register banks
Output	PSW.CY 0 = Test Passed 1 = Test Failed
Stack size required	2
Resource used/destroyed	PSW, ACC, DPTR, R0-R7 of current register bank, XRAM memory area where contents of memory under test are stored MEX3 set to 0x1F
Execution time	530usec with R3=128bytes

7.1.4 XRAM Test At Startup

Destructive test. The contents of the complete XRAM will be destroyed.

Table 7-3 XRAM MarchC Startup Test Routine

Routine	--: XRAM_MarchC_ST_Test
Input	-
Output	PSW.CY 0 = Test Passed 1 = Test Failed
Stack size required	--
Resource used/destroyed	PSW, ACC, DPTR0, SCU_PAGE, XADDRH All XRAM under test is cleared to ‘0’ MEX3 set to 0x1F
Execution time	14.5msec

7.1.5 XRAM Test At Runtime

This test is non-destructive, and so the test can be executed when the application is running. It is tested in block, where the block length is to be specified by the user.

The XRAM data under test is stored into another user specified location in XRAM. The content is restored before it returns to user code.

Precondition:

All interrupts are disabled.

Table 7-4 XRAM MarchC Runtime Test Routine

Routine	--: XRAM_MarchC_RT_Test
Input	1. R6(MSB),R7(LSB) - Start address of XRAM to be tested 2. R4(MSB),R5(LSB) - Start address of XRAM to store data, 3. R3 - Number of bytes to be tested. If R3 = 0, 256 bytes of XRAM will be tested.
Output	PSW.CY 0 = Test Passed 1 = Test Failed
Stack size required	2
Resource used/destroyed	PSW, DPTR0, DPTR1, ACC R0-R7 of current register bank, XRAM memory area where contents of memory under test are stored
Execution time	1.0msec with R3=128bytes

7.2 MarchX Memory Test Routines

The MarchX test algorithm is less complex than MarchC algorithm. However the coupling fault coverage is reduced in MarchX test. It cannot detect:

- Idempotent coupling fault, CFin
- Dynamic coupling fault, CFdyn.

Two routines are provided to perform IRAM and XRAM memory tests at runtime.

7.2.1 MarchX Algorithm

The following list the steps in MarchX memory test:

1. Write all zeros to memory under test.
2. Starting at lowest address, read zeros, write ones, increment address.
3. Starting at highest address, read ones, write zeros, decrement address.
4. Read all zeros from memory.

7.2.2 IRAM Test At Runtime

The IRAM test is non-destructive and therefore can be executed when the application is running. It is tested in block, where the block length is specified by the user.

The IRAM data under test is stored in an XRAM location before the MarchX test starts. The data is restored before the test returns control to user code.

Attention: Current register bank (R0-R7) address and stack area should not be tested.

Precondition:

All interrupts are disabled.

Table 7-5 IRAM MarchX Runtime Test Routine

Routine	--: IRAM_MarchX_RT_Test
Input	1. R7 - Start address of IRAM to be tested 2. R4(MSB) R5(LSB) - XRAM start address to store the IRAM data 3. R3 - Number of bytes to be tested, range 1 to X, where X = 256 - stack area - register banks
Output	PSW.CY 0 = Test Passed 1 = Test Failed
Stack size required	2

Table 7-5 IRAM MarchX Runtime Test Routine (cont'd)

Resource used/destroyed	PSW, ACC, R0-R7 of current register bank, XRAM memory area where contents of memory under test are stored MEX3 set to 0x1F
Execution time	400usec with R3 = 128bytes

7.2.3 XRAM Test At Runtime

The XRAM test is non-destructive and can therefore be executed when the application is running. It is tested in block, where the block length is to be specified by the user.

The XRAM data under test is stored into another XRAM location specified by the user. The data is restored before the test returns control to user code.

Precondition:

All interrupts are disabled.

Table 7-6 XRAM MarchX Runtime Test Routine

Routine	--: XRAM_MarchX_RT_Test
Input	1. R6(MSB),R7(LSB) - Start address of XRAM to be tested 2. R4(MSB),R5(LSB) - Start address of XRAM to store data 3. R3 - Number of bytes to be tested. If R3 = 0, 256 bytes of XRAM will be tested.
Output	PSW.CY 0 = Test Passed 1 = Test Failed
Stack size required	3
Resource used/destroyed	PSW, EO, DPTR0, DPTR1, ACC, R0-R7 of current register bank, XRAM memory area where contents of memory under test are stored
Execution time	720usec with R3=128bytes

8 System Framework

An example of a System Framework that monitors the status of the XC878-16FF microcontroller via a CAN communication interface, is described here.

The System Framework consists of 2 parts:

- CANscheduler to run in the XC878-16FF microcontroller target board.
- DriveMonitor to run on PC.

The framework provides the flexibility to monitor the internal variables using the DriveMonitor. It enables the user to set and poll the internal variables during runtime.

The CANscheduler will be provided to the user. The DriveMonitor software can be found in the IFX web site

http://www.infineon.com/cms/en/product/promopages/dave-drive-download/index.html

8.1 CANscheduler Operation Overview

Overview of the operations in the CANscheduler:

- DriveMonitor sends out a command via CAN to XC878-16FF target board
- Incoming CAN message will trigger a receive interrupt in the XC878-16FF microcontroller if the ID's match.
- In the CAN receive interrupt service routine, it will:
  - copy the message to the receive buffer;
  - set a flag to indicate that a new command is pending.
- In timer21 interrupt service routine, it will:
  - check if there is pending new command;
  - executes the new command accordingly;
  - Clear the flag to indicate the new command is executed.

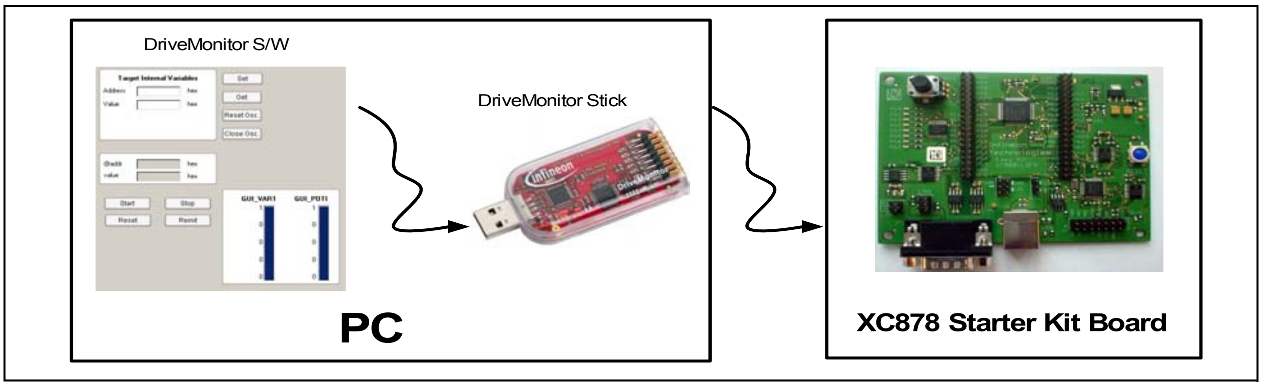

Figure 8-1 System Setup

8.2 XC878 Starter Kit Setting and Modification

This section describes the settings and modifications of the XC878 starter kit that are required before it can be used to run the system test.

1. Ensure that the jumper, COM, is set to activate CAN node 0, as shown in **Figure 8-2**.

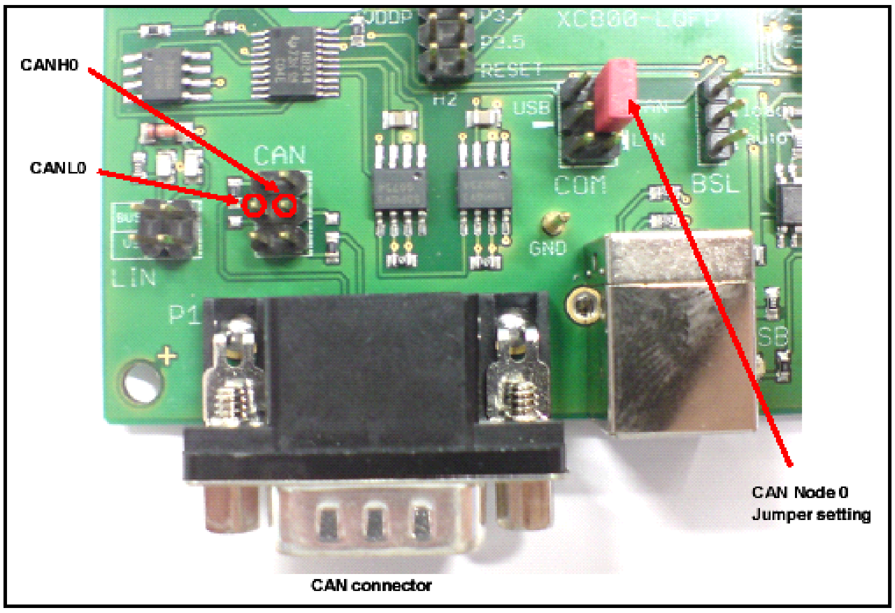

Figure 8-2 Select CAN node 0

2. Connect CANH0 and CANL0 to the JTAG connector, OCDS, as shown in **Figure 8-3**.

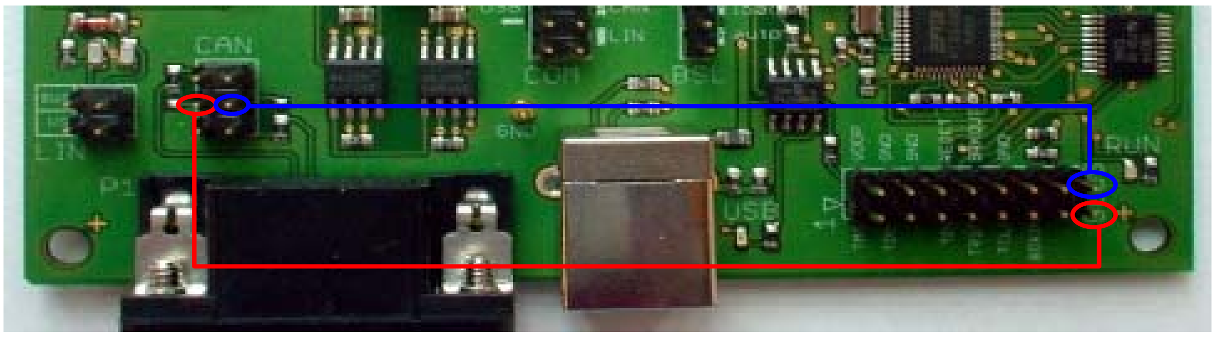

Figure 8-3 Connection of CANH0 And CANL0 to JTAG Connector

3. Remove resistors R201 and R202 so that the starter kit can be powered using USB from PC.

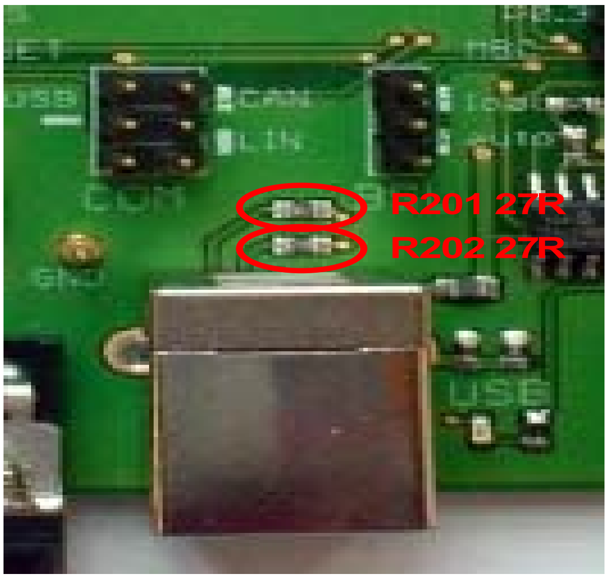

Figure 8-4 R201 and R202 Locations.

4. Connect the DriveMonitor stick to the starter kit board using the supplied ribbon cable.

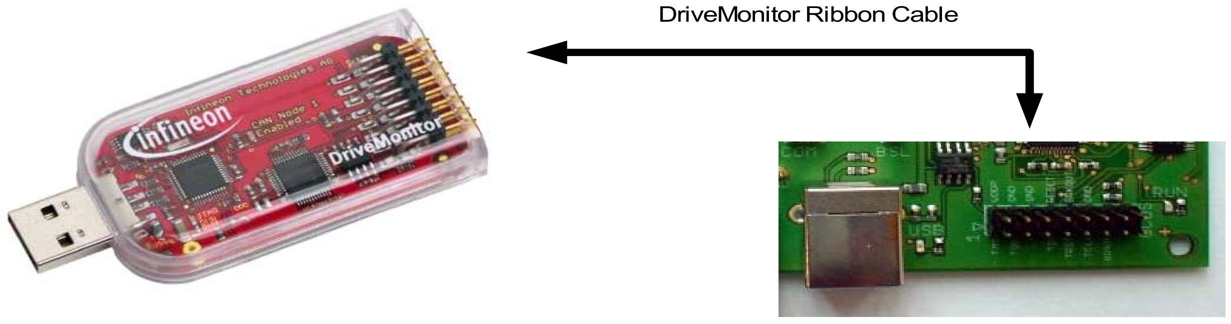

Figure 8-5 DriveMonitor to XC878 Starter Kit Board connection

8.3 Resources Requirements

In this framework, the following resources are required in order to run the Class B software library:

- Watchdog timer (WDT): The WDT requires periodic servicing to ensure that the PC is not stuck, otherwise a Watchdog reset will occur.
- Timer 0 (T0): This timer is used to keep track of the Watchdog window. Timer overflow will cause an interrupt. WDT servicing is performed at the application level.
- Timer 1 (T1): Class B runtime test routines will be executed whenever the timer overflows.

Other resources used in the framework are:

- Timer21 which is used for task scheduling, and
- CAN module for communication with PC
- Port 3 is used as runtime status indication. 'OR' logic is used to set the status, to ensure that an error will be captured throughout the test.
  - P3.0 = 1, CPU_Registers_Test() fail
  - P3.2 = 1, IRAM_MarchC_RT_Test() fail
  - P3.3 = 1, IRAM_MarchX_RT_Test() fail
  - P3.4 = 1, XRAM_MarchC_RT_Test() fail
  - P3.5 = 1, XRAM_MarchX_RT_Test() fail
  - P3.6 = 1, PLL loss of lock, PLL NMI interrupt service routine is executed
  - P3.7 = 1, Double bit ECC detected in Flash, ECC NMI interrupt service is executed

8.4 Flowcharts of the CANscheduler

Figure 8-6 to **Figure 8-11** show the flowcharts of the CANscheduler.

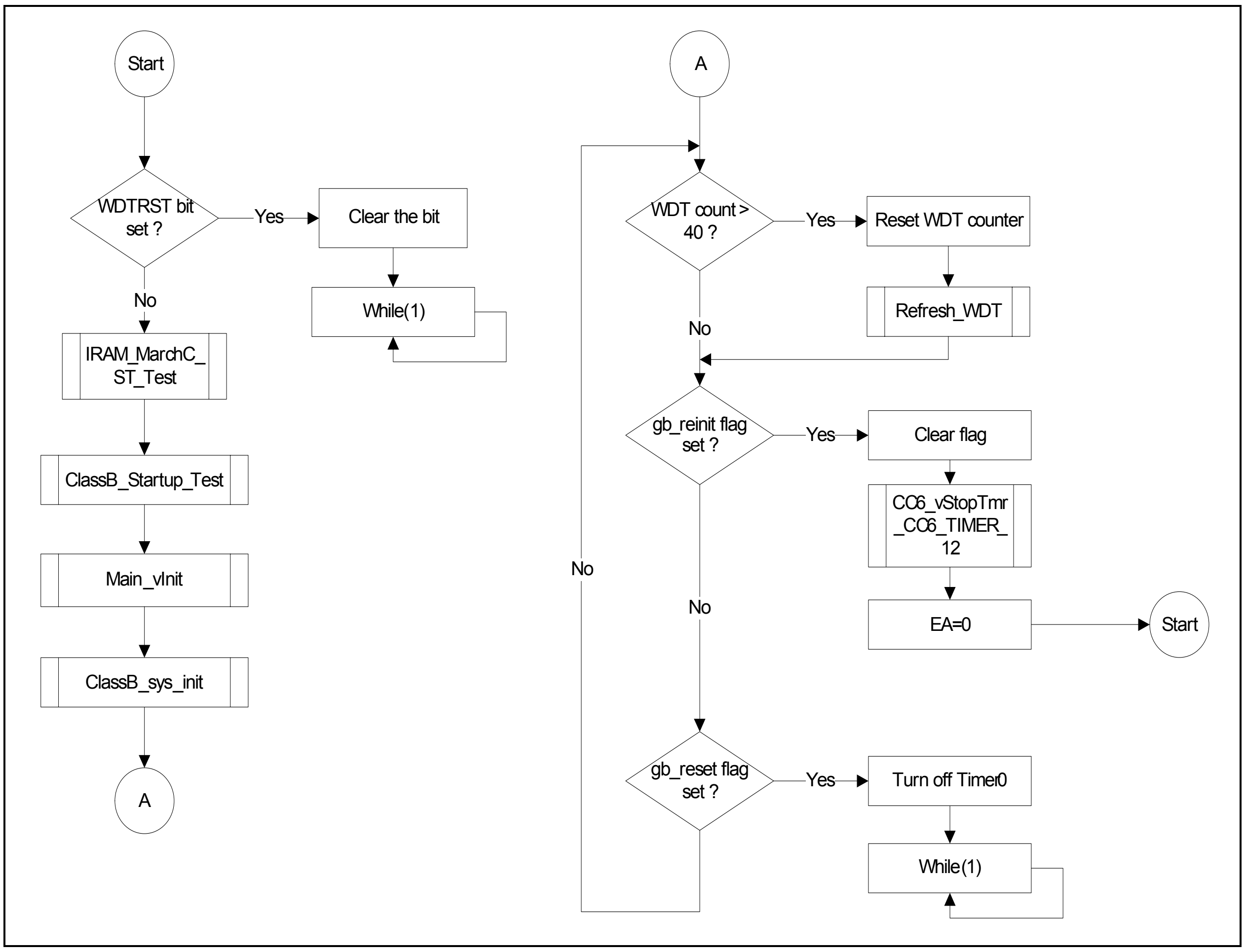

Figure 8-6 Main Routine Flow

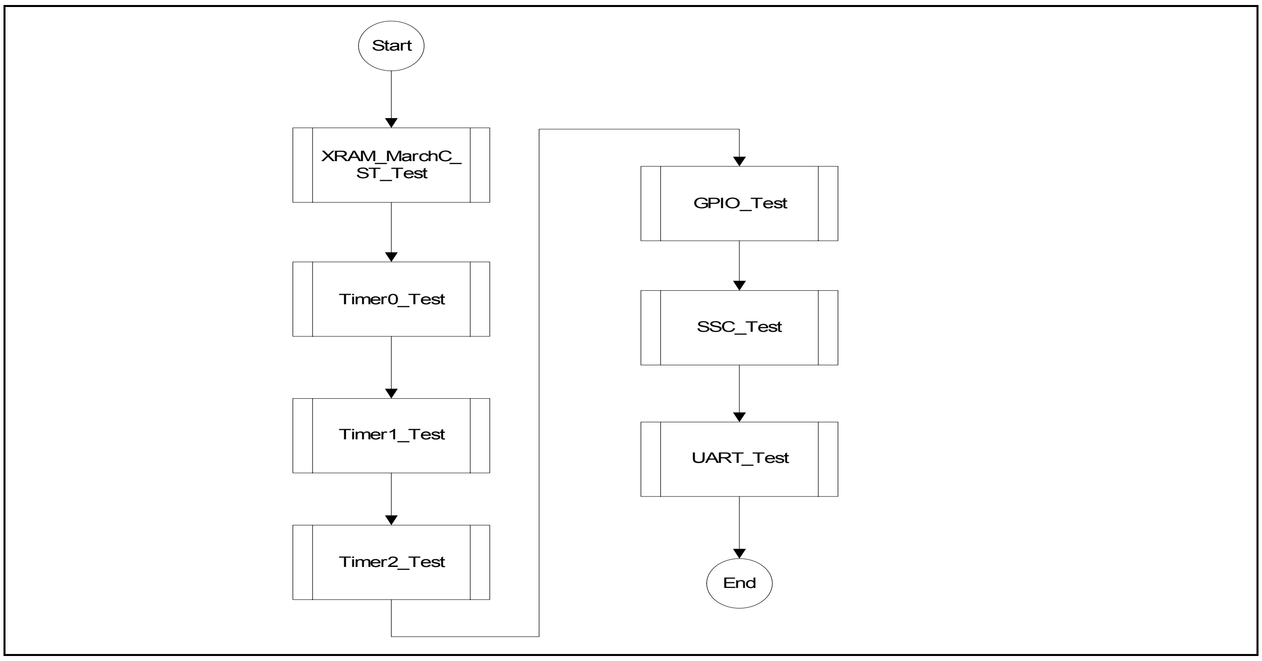

Figure 8-7 Class B Startup Tests

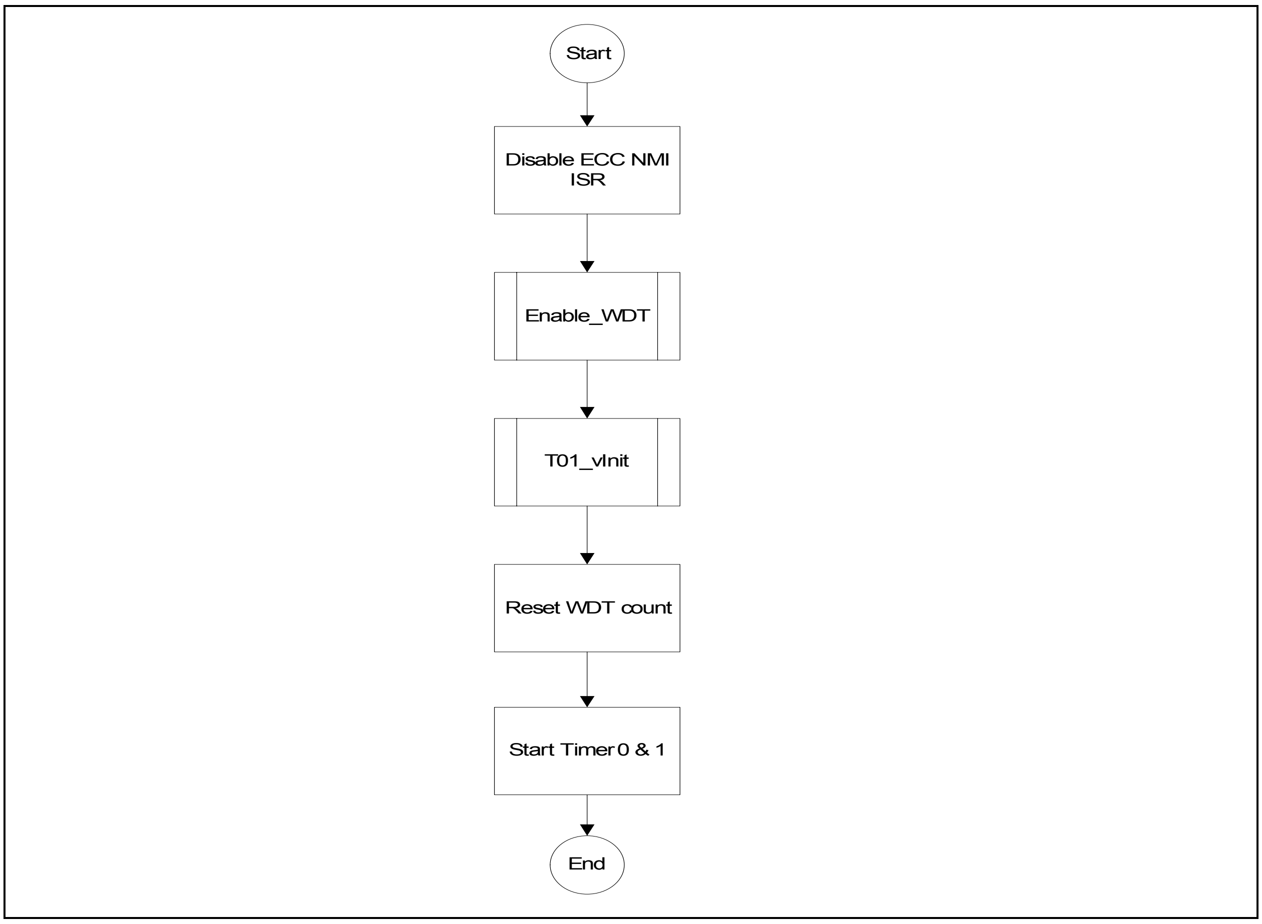

Figure 8-8 Class B System Init

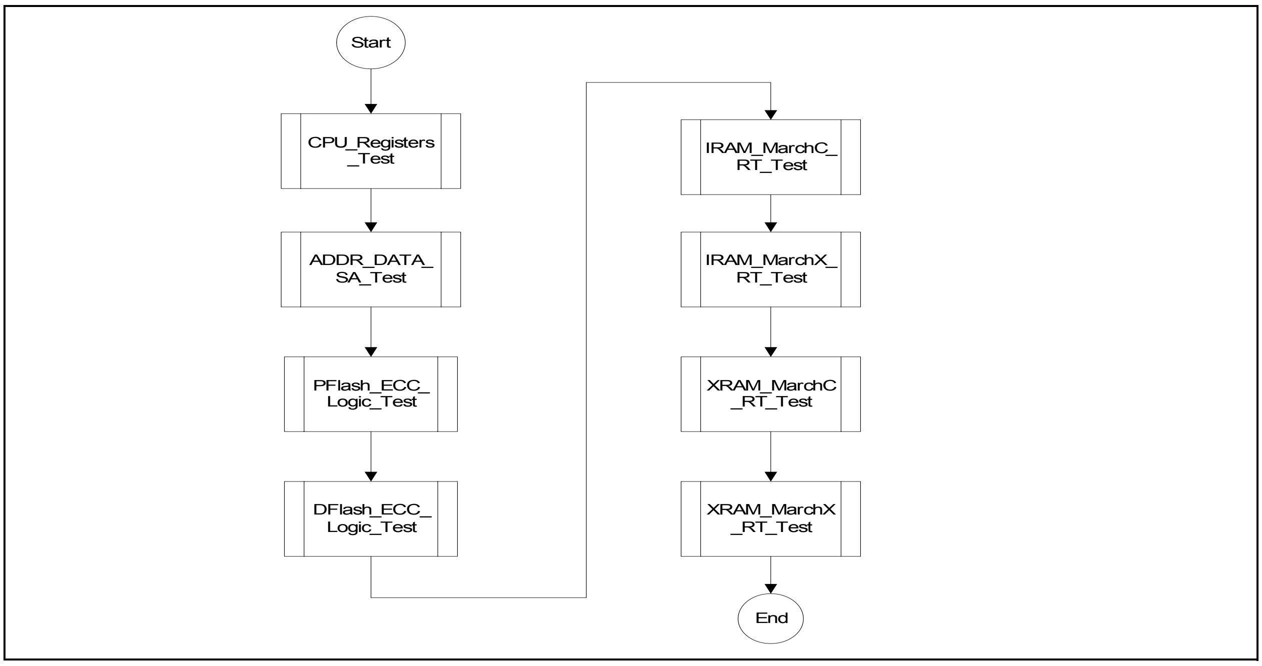

Figure 8-9 Class B Runtime Tests

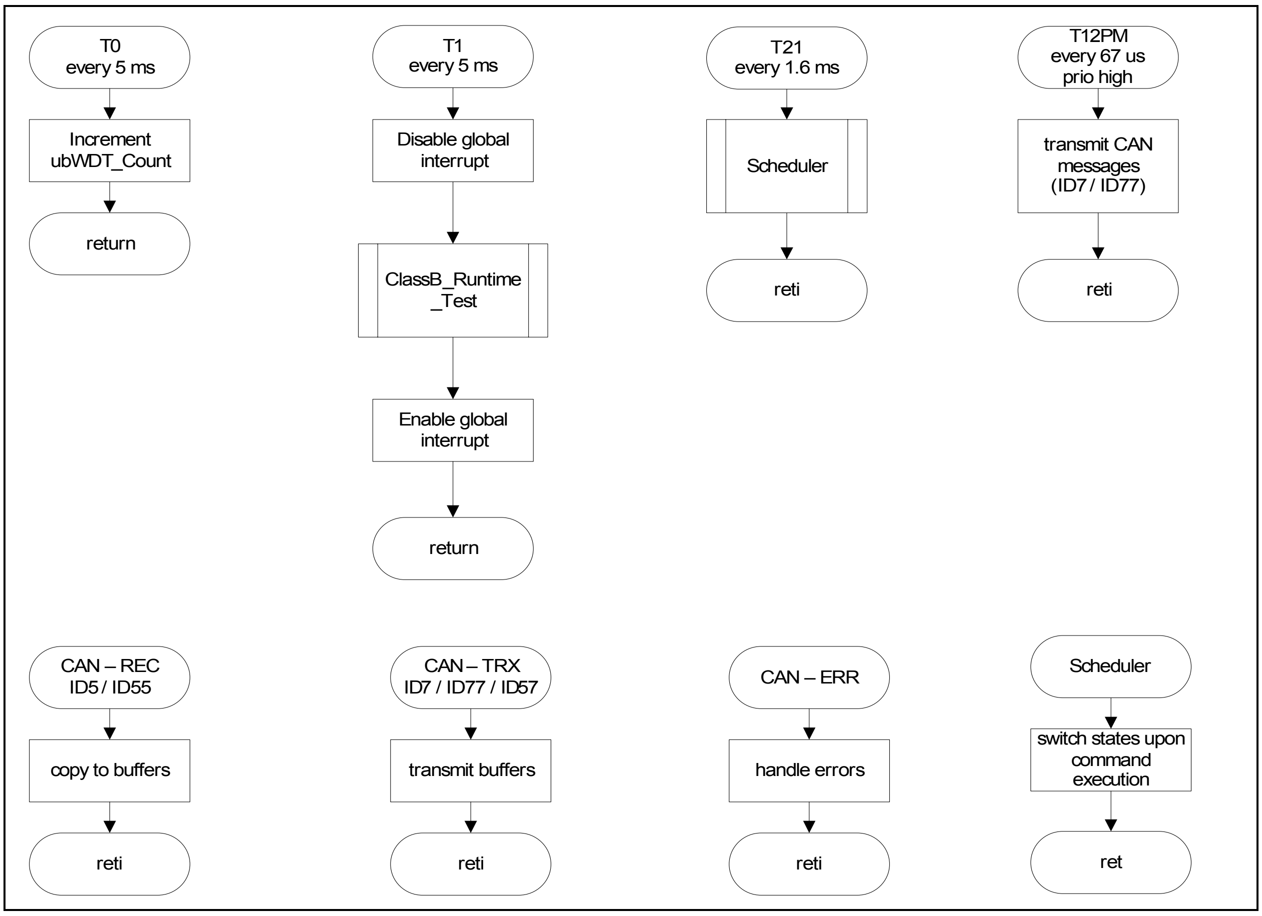

Figure 8-10 Interrupt Services Routines

There are 2 NMI being enabled:

- NMI PLL to detect the PLL loss of lock
- NMI ECC to detect the double bits error in the Program flash and Data flash.

Once an error is detected, the respective Port 3 pin is set and the system enters an endless loop, although the user could replace the endless loop with their own error handling code.

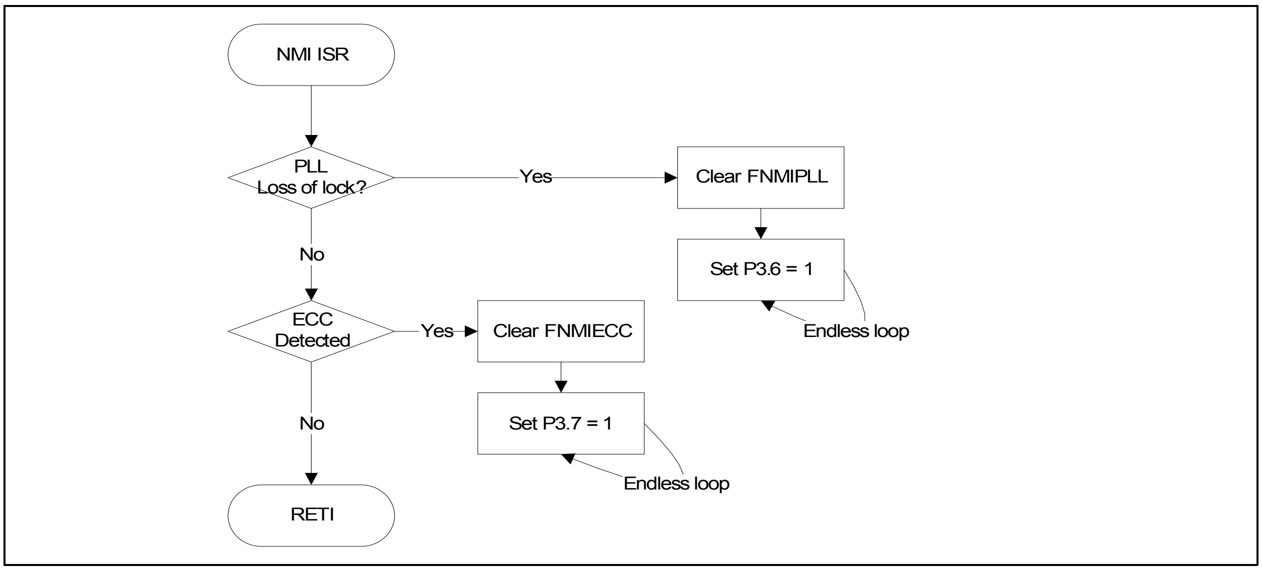

Figure 8-11 NMI PLL and NMI ECC Service Routine