Atmel®

AVR 8-bit Microcontrollers

AVR998: Guide to IEC60730 Class B Compliance with AVR Microcontrollers

APPLICATION NOTE

Introduction

The IEC60730 is a functional safety standard for household appliances. Microcontrollers (MCU) are widely used electronics control components in household appliances. The functional safety requirements on an MCU system are presented in the standard's Annex H "requirements for electronics control", with details on test items, diagnostics, and measures to provide proper and safe operations of embedded software and hardware in household appliances. This application note gives a brief introduction of the IEC60730 standard and provides guidance of functional checking for each MCU component. The application note also provides IEC60730 Class B firmware examples respectively with Atmel® megaAVR® and tinyAVR®.

Features

- Guidelines of IEC60730 Class B compliant functional test
- Guidelines of IEC60730 Class B compliant periodic test
- Class B Firmware example for ATtiny817 and ATmega328PB

Table of Contents

1. Overview

The International Electrotechnical Commission (IEC) has introduced the IEC60730 referring to household appliances development. The IEC 60730 is a safety standard for household appliances that addresses many aspects of both product design and operation. This standard is also referred to by other standards for safety-critical devices, for example IEC 60335. System-wide compliance with this standard is necessary for an appliance to be certified as safe to operate. This application note is a guide to compliance with Annexure H of the standard, which regards electronic controls. Annexure H of IEC60730 describes classifications for control software:

- Class A - control functions which are not intended to be relied upon for the safety of the equipment (H.2.21.1)
- Class B - software that includes code intended to prevent hazards if a fault, other than a software fault, occurs in the appliance (H.2.21.2)
- Class C - software that includes code intended to prevent hazards without the use of other protective devices (H.2.21.3)

This application note deals with Class B, which applies to a large range of products, including: dishwashers, washing machines, refrigerators, freezers, and cookers. According to IEC60730, household appliance manufacturers must now design their product following Class B rules.

Most of the products listed above use a single-chip microcontroller with embedded memory and peripherals, referred to as single channel application.

According to IEC60730, single channel application (only one microcontroller used) must have firmware designed based on one of the two techniques below.

- Single channel functional test
- Single channel periodic self-test

Single channel using functional test is the most popular mechanism used today and the easiest to implement. Most appliance manufacturers now implement single channel with periodic self test in their new designs. This document deals with these two techniques and gives additional guidelines on how to comply with the Class B requirement when using Atmel AVR® microcontrollers.

All the features of the AVR microcontrollers are tested during factory production. Some features are more sensitive to harsh environment than others. For example, CPU registers, Stack Pointer register, and Status register are sensitive to stuck bits. On the other hand, a large area on devices such as RAM, Flash, or EEPROM memories may need more complex verification algorithm. In this document we give a compromise between feature sensitivity and the test to be implemented. Customers may decide to simplify or add extra tests according to their requirements.

2. Guidance of Single Channel Functional Test for AVR

The order in which the test is performed is important and the tests must be run in the same order as it is presented below. For example, it is important to test the RAM memory before the Flash content because the Flash content test will use RAM memory to run. Some tests are also firmware intrusive (interrupt tests for example) and may be integrated in end application firmware. All errors in functional test may stop the application and generate an error message.

Errors in periodic tests may lead to watchdog timeout in order to restart the application.

Test implementation is a compromise between start-up time, used resources, and protection level of the application.

Failure information can also be saved by the application to help failure diagnostics.

2.1. CPU Registers Test (General Purpose Register R0 to R25, X, Y, and Z Register Test)

Purpose of the test: Control that no bits of these registers are stuck.

The CPU registers test will test all the general purpose working registers from R0 to R31 and the Status register. The goal of this test is to detect if one bit of these registers is stuck to ‘1’ or to ‘0’. This test is done by successively writing, reading, and checking 0x55 and 0xAA values into those registers. This test is done in assembler language just after Stack pointer initialization R29 to R31 are tested first, then these registers are used for testing R0 to R28.

2.2. Stack Pointer Register Test

Purpose of the test: Test that stack pointer bits are not stuck.

This test is done by successively write, read, and check 0x55 and 0xAA value into these registers. No RET or RETI instructions are executed before this test.

2.3. Status Register Test

Purpose of the test: Control that all bits of the register are not stuck to ‘0’ or ‘1’.

The goal of this test is to detect if one bit of the register is stuck to ‘1’ or to ‘0’. This test is done by successively writing, reading, and checking 0x55 and 0xAA value into these registers.

The Status register is used by previous tests, so we can consider three ways to reach this test:

1. The Status register is fully functional, therefore all previous test results are good.
2. The Status register is wrong, therefore previous tests may have failed as the test itself was good and we cannot reach at this test.
3. The Status register is wrong and we go through the previous tests, but we will stop here.

In all cases, any problems with one of the previous tests or the Status register test will lead to stop the program.

2.4. RAM Memory Test

Purpose of the test: Control that no bit of the RAM memory is stuck at ‘1’ or ‘0’.

The RAM test will test all the RAM memory locations. This test is done by successively writing, reading, and checking 0x55 and 0xAA value into the RAM memory. The firmware sets a bit in a “RAM Test Status Register”. The program verifies that the bit is set at the end of the test.

Another test allowing to control the address encoder is to write the complement of the address of a location into RAM location. There is also a read and verify mechanism to control the values stored. The initial values of the RAM memory are (when doing periodic functional tests) saved before the test and restored afterwards to enable stacked data to be retrieved after the test.

March B test is implemented to test all the RAM. The test is divided in two parts tested separately. Between each part test, the stack content is saved to the other part and the stack pointer points to the other part. The size and overlap of both parts are configurable and must be set according to physical memory organization.

2.5. Flash Memory Control

Purpose of the test: Control the flash content.

The Flash memory test must prevent any flash corruption. It can be done by a simple checksum of the flash content, or a more complex and time consuming Cyclic Redundancy Check. The reference result is stored in a particular place of the Flash memory at programming time. The calculated result is compared to this reference result at running time.

2.5.1. Note on the Use of the SPM Instruction

We recommend when it is possible to not use the SPM instruction in a harsh environment.

The SPM (Store Program Memory) is the instruction that allows to write into the Flash memory. The SPM instruction can access the entire Flash, including the boot load section. For example, if a function uses the SPM instruction and a power loss occurs, the Flash memory can be corrupted.

The protection level for the Boot Loader section against SPM instruction use can be selected by the Boot Loader Lock bits available on most of the AVR products.

2.6. EEPROM Memory Test/Control

Purpose of the test: Control the EEPROM contents.

The EEPROM memory test must prevent any EEPROM corruption. It can be done by a simple checksum of the EEPROM content, or a more complex and time consuming Cyclic Redundancy Check. As content of the EEPROM may vary during application life, the optimal solution should be to update the reference result at each write of the EEPROM to be able to have a dynamic comparison of the memory content during the product life.

2.7. Watchdog Test

Purpose of the test: Verify that the watchdog is functional.

This test will check the functionality of the watchdog reset. Upon reset, the test will check if a reset occurs from watchdog reset. If not, the watchdog will be started and the test will wait until it occurs.

Note: The watchdog must be fuse enabled.

2.8. Interrupt Functionality Test

Purpose of the test: Test if the interrupt controller works correctly.

All interrupts which are not used by the application can be activated by the interrupt test function in order to check the correct behavior of the interrupt controller. The interruption is activated by software and the corresponding interrupt vector generates a signal to the interrupt test function.

2.9. I/O Registers

Purpose of the test: Test that I/O bits are not stuck at ‘1’ or ‘0’.

The I/0 test will test all the I/0s. This test is done by successively writing, reading, and checking 0x55 and 0xAA value into the I/0 registers. **This test is application dependent and can be run only if the hardware allows it**.

2.10. Clock Frequency

Purpose of the test: Test of internal clock frequency.

To test the internal clock of the processor, it needs to have a reference clock available. For applications using external crystals, we can use the internal RC oscillator to check crystal presence and also to verify frequency oscillation of the external crystal.

Another way is to use a communication bus like SPI to measure the SPI clock duration. The result will be compared to the theoretical value.

The firmware sets a bit in the “Clock Test Status Register”. The program verifies that the bit is set at the end of the test.

2.11. Analog to Digital Converter

Purpose of the test: Test the ADC analog functions.

Free analog inputs can be wired to known external voltages to control the correct behavior of the ADC.

This test can also be done by using the internal bandgap reference as the ADC input. For example, the ATmega16 internal bandgap reference delivers a 1.22V voltage, which can be regularly converted to test the ADC.

3. Guidance of Single Channel Periodic Self-test on AVR

Periodic tests are embedded with the firmware and will regularly check that everything is functioning normally. Prior to the execution of the application firmware, functional tests are executed. All errors in periodic tests may lead to watchdog timeout in order to restart the application.

3.1. Interrupt Periodic Event

Purpose of the test: Check that interrupt occurs regularly in a defined lapse of time. In the same way, by the use of a counter, the test can detect if an interrupt occurs too frequently.

At each interrupt vector address, the firmware sets a bit in a user-defined “IT Test Status Register”. The program verifies periodically that bits are set, then clears the “IT Test Status Register”. On errors a watchdog event is called.

Interruptions which are not used by the application can be used by the test to regularly check the interrupt controller.

3.2. Function Periodic Event

Purpose of the test: Check that some functions are called regularly in a defined lapse of time.

Using the same mechanism as the interrupt test, each function sets a bit in “Function Event Status Register”. The program verifies periodically that bits are set and then clears the “ Function Event Status Register”. On errors, a watchdog event is called.

3.3. Error Event Detection

Purpose of the test: Check that every function gives a correct value on a regular basis.

This test is slightly different from the previous ones. This test checks that functions give a valid value. It will allow to restart if values are out of range or if any features experience trouble and do not return the correct values (ADC conversion result for example). The periodic function will generate a watchdog event on wrong values or on several wrong values from specific functions.

4. Additional Guidelines

4.1. External Communication

All communication must be secured by transfer redundancy when possible, in order to enable the receiver to check for data corruption during transfers. Timeout detection may be implemented to prevent failure and endless loops.

4.2. Watchdog

Intensive watchdog use prevents code fault and error values from the firmware.

5. IEC60730 Class B Firmware Example

This application note provides IEC60730 Class B firmware examples for ATmega328PB and ATtiny817 to show how to add a class B test to the main application. Users can further enhance safety test based on their specific applications. The examples can be downloaded from Atmel | START and they can easily be ported to other megaAVR or tinyAVR microcontrollers.

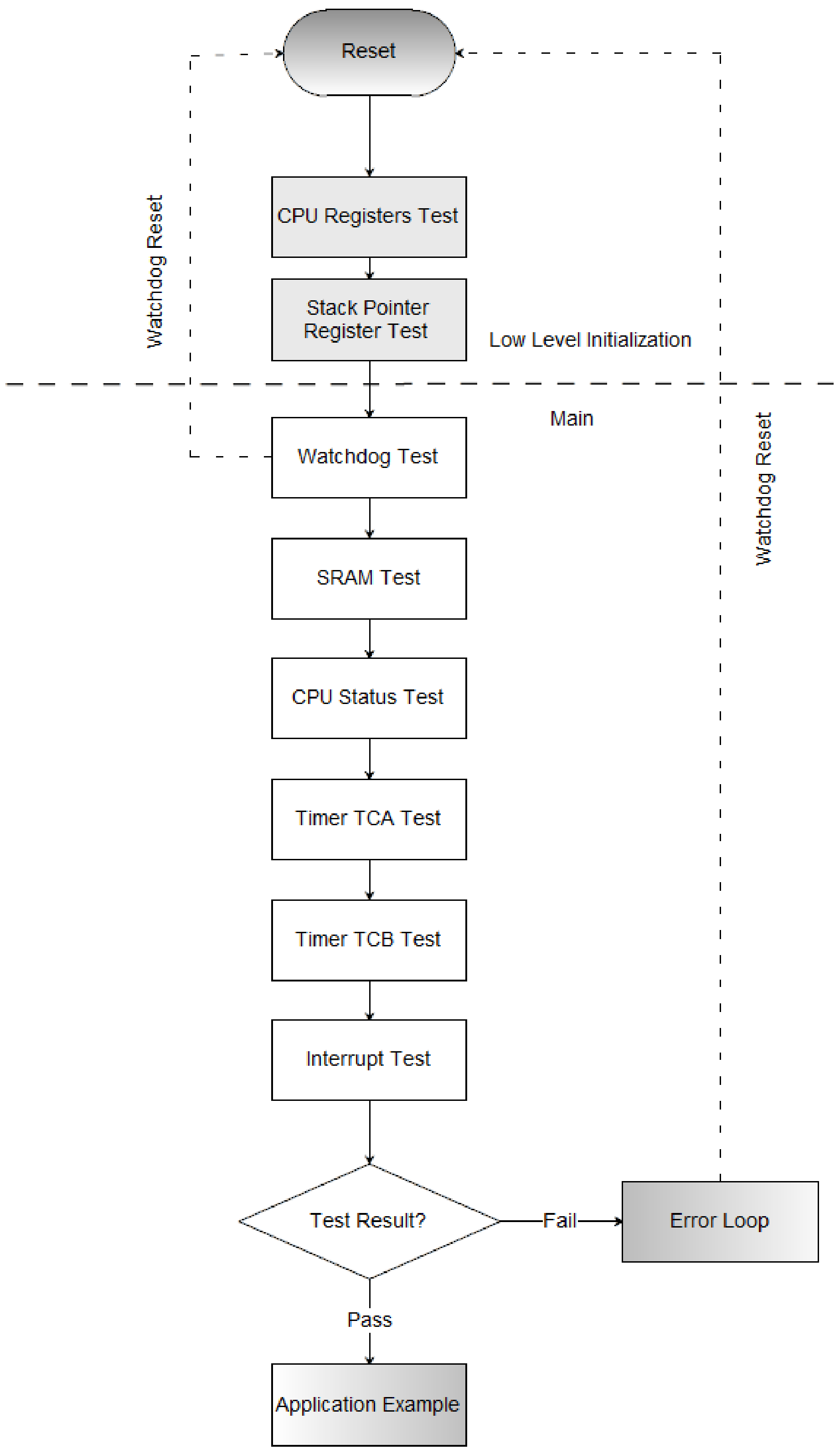
The main files in the examples are:

- `main.c` file is the main application
- `low_level_init.c` file embeds class B test functions, which must be executed before the initialization of the C context
- `classB.c` file contains the class B tests

Note: Examples requires Atmel Studio version 7.0.1006 (or later) or IAR version 6.80 (or later).

The overall test procedures are shown in the figure below. CPU registers and Stack Pointer register tests are performed during low level initialization before entering Main function. Other tests are performed in Main function.

Figure 5-1. Overall Test Procedures



5.1. Test Routines

A series of test routines are implemented to exemplify typical tests. The test items and the corresponding APIs in tinyAVR (ATtiny817) example are shown in the table below.

Table 5-1. Test Routines in tinyAVR Example

Test items	API	Source file
CPU Registers Test	__low_level_init()	low_level_init.c
Stack Pointer Register Test	__low_level_init()	low_level_init.c
Watchdog Test	watchdog_test()	classB.c
SRAM Test	sram_test()	classB.c
CPU Status Test	cpu_status_test()	classB.c
Timer TCA Test	timer_tca_test()	classB.c
Timer TCB Test	timer_tcb_test()	classB.c
Interrupt Test	interrupt_test()	classB.c

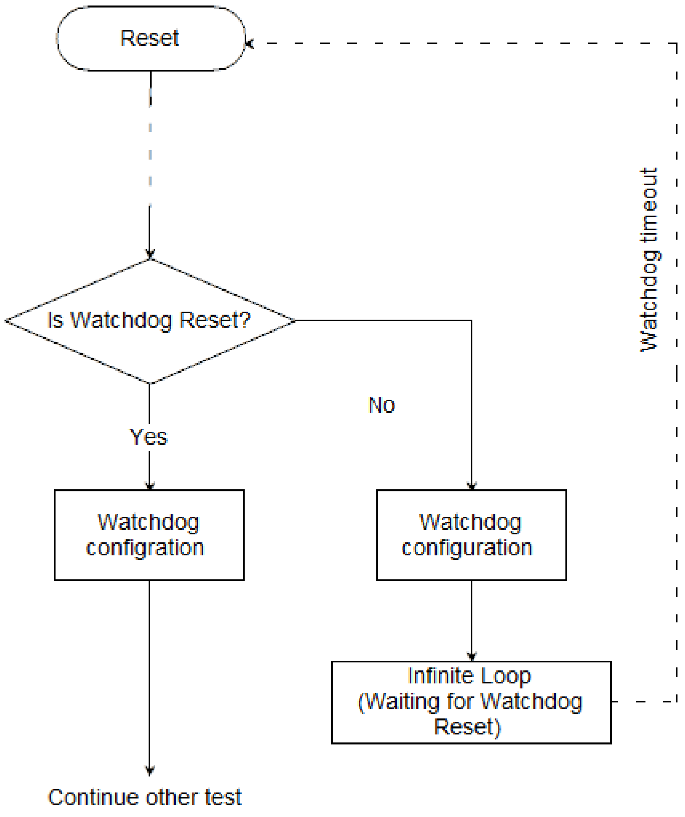
5.1.1. __low_level_init()

The test on general registers and Stack Pointer register are implemented in __low_level_init(), which is executed during initialization even before main() function. The test starts from R31/R30, then down to R1/R0, and ends with Stack Pointer register.

5.1.2. watchdog_test()

Watchdog test is demonstrated in watchdog_test() with the process shown in the figure below.

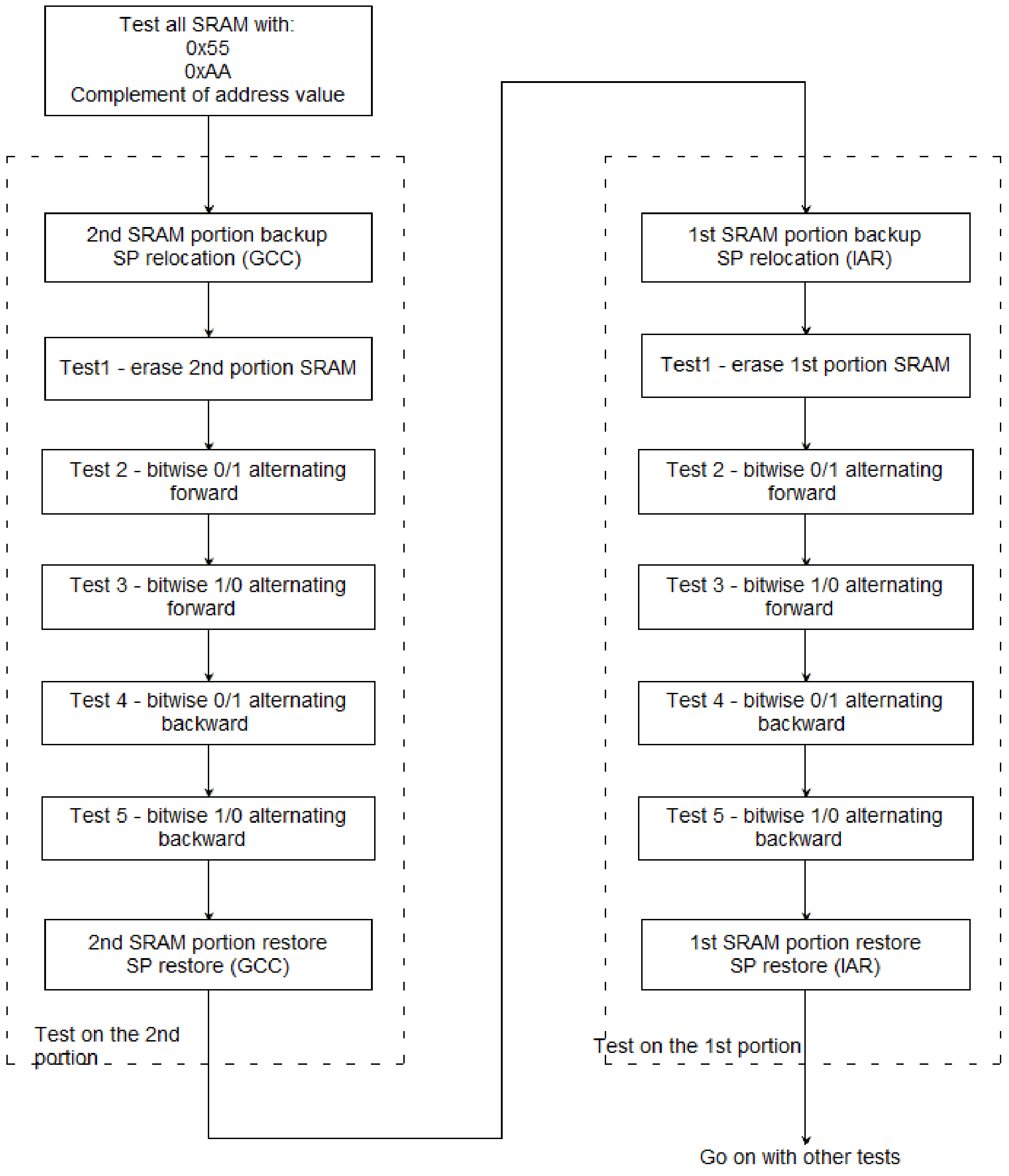
Figure 5-2. Process Diagram of Watchdog Test



5.1.3. sram_test()

SRAM test is exemplified in SRAM_test() with the process shown in the figure below.

Figure 5-3. Process Diagram of SRAM Test


Test all SRAM with:
0x55
0xAA
Complement of address value
2nd SRAM portion backup
SP relocation (GCC)
Test1 - erase 2nd portion SRAM
Test 2 - bitwise 0/1 alternating forward
Test 3 - bitwise 1/0 alternating forward
Test 4 - bitwise 0/1 alternating backward
Test 5 - bitwise 1/0 alternating backward
2nd SRAM portion restore
SP restore (GCC)
Test on the 2nd portion
1st SRAM portion backup
SP relocation (IAR)
Test1 - erase 1st portion SRAM
Test 2 - bitwise 0/1 alternating forward
Test 3 - bitwise 1/0 alternating forward
Test 4 - bitwise 0/1 alternating backward
Test 5 - bitwise 1/0 alternating backward
1st SRAM portion restore
SP restore (IAR)
Test on the 1st portion
Go on with other tests

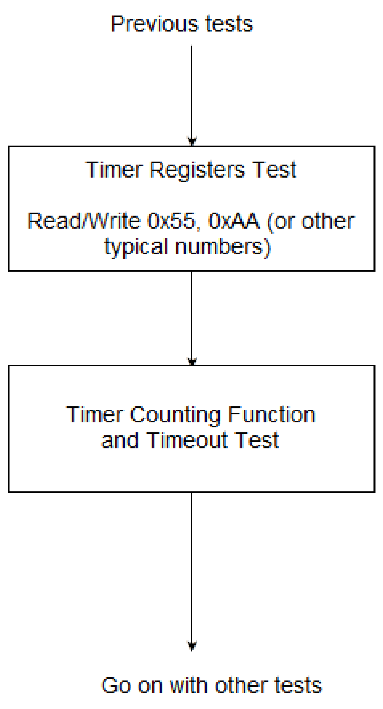
5.1.4. cpu_status_test()

CPU status register SREG is tested in cpu_status_test(). The SREG should be backed up before writing test data into it and restored after the test.

5.1.5. timer_tca_test()

Timer TCA is exemplified in timer_tca_test() with the process shown in the figure below.

Figure 5-4. Process Diagram of Timer TCA/TCB


Previous tests
Timer Registers Test
Read/Write 0x55, 0xAA (or other typical numbers)
Timer Counting Function and Timeout Test
Go on with other tests

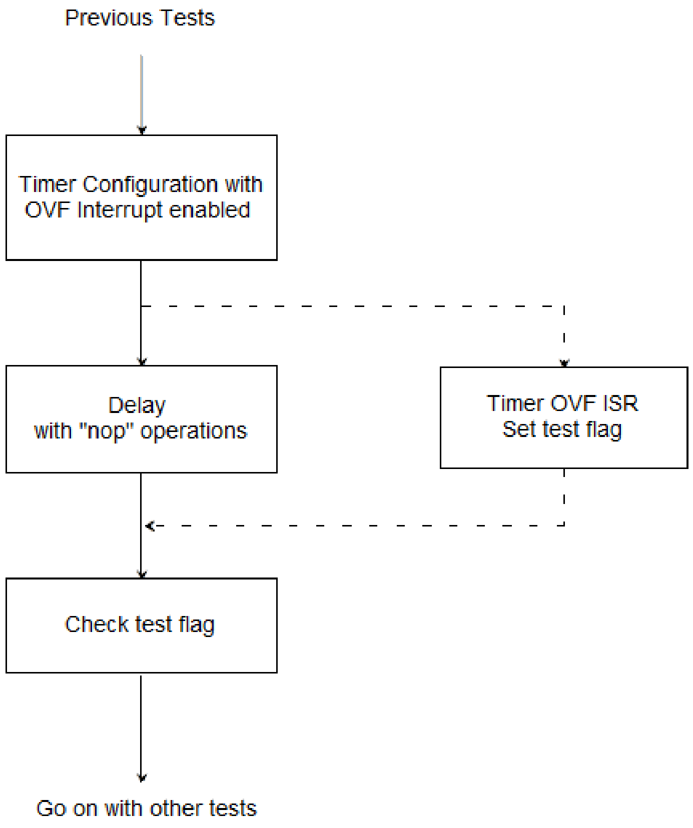
5.1.6. timer_tcb_test()

Timer TCB is exemplified in timer_tcb_test() with the process shown in Figure 5-4.

5.1.7. interrupt_test()

Interrupt test exemplified in interrupt_test() with the process shown in the figure below. Timer OVF interrupt is used in this routine. The customer should determine what interrupt should be used according to the specific application.

Figure 5-5. Process Diagram of Interrupt Test


Previous Tests
Timer Configuration with OVF Interrupt enabled
Delay with "nop" operations
Timer OVF ISR Set test flag
Check test flag
Go on with other tests

5.2. Memory Usage and Running Cycles

The test firmware memory usage and running cycle number depend on which compiler is used as well as its configuration. This section provides an overview on how much Flash and SRAM is needed and how many cycles are consumed to run every test. The data is acquired with Atmel Studio with Optimization "-Os" level and all data are for user reference only.

Table 5-2. Memory Usage and Cycle Numbers

Test routines	Flash size [Bytes]	SRAM size [Bytes]	Cycles
watchdog_test()	36	0	27 + Watchdog timeout
sram_test()	934	0	1404727
cpu_status_test()	32	0	36
timer_tca_test()	916	2	684
timer_tcb_test()	382	2	386

Test routines	Flash size [Bytes]	SRAM size [Bytes]	Cycles
interrupt_test()	152	1	93
__low_interrupt_init() (CPU general &SP registers)	812	0	344

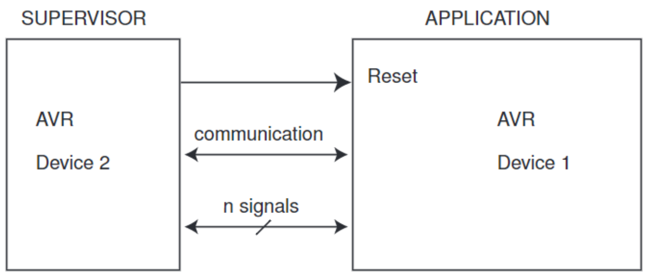
6. Application Note References

1. AVR040 : EMC Design Considerations - http://www.atmel.com/images/doc1619.pdf
2. AVR042 : AVR Hardware Design Considerations - http://www.atmel.com/images/atmel-2521-avr-hardware-design-considerations_applicationnote_avr042.pdf
3. AVR132 : Using the Enhanced Watchdog Timer - http://www.atmel.com/images/doc2551.pdf
4. AVR180 : External Brown-Out Protection - http://www.atmel.com/Images/doc1051.pdf
5. AVR236 : CRC check of Program Memory - http://www.atmel.com/images/doc1143.pdf
6. IEC60730 : Automatic electrical controls for household and similar use - http://ulstandards.ul.com/standard/?id=60730-1_4
7. March B : Various literatures about testing Static Random Access Memories are available on Internet. The annex of this document describes the Class B test.
8. AVR3004: QTouch® with Safety Features - http://www.atmel.com/images/doc42041.pdf
9. AVR1610: Guide to IEC 60730 Class B Compliance with XMEGA® - http://www.atmel.com/images/doc42008.pdf

7. Appendix

7.1. How to Secure a Design

The best way to secure a design is to use double channel configuration with two devices checking each other. The following picture shows such implementation. Of course, double channel implementation is more expensive than a single one, but it allows to have the best security coverage as each part can check what the other part is doing, and also be used as an external watchdog to prevent any oscillator issue (CPU or watchdog oscillator).



With double channel configuration, security can be increased. Examples below show what kind of functional interconnection can be implemented to secure the application.

- Device 2 can check that device 1 functions by communication means or any periodic external signal time out, then resets Device 1 on any issue
- Device 1 can check that device 2 is running by any external signal timeout too
- When device 1 wants to perform any external action, it can send a message (by any means available such as SPI, TWI, UART….) to device 2 to confirm that the action is done. Then device 2 can check the action result and inform device 1, that the action is correctly done.
- Device 2 can also save the test result of device 1 for analysis in case of system failure

Device 2 has a role of supervisor only and it can be very low cost. For example an ATtiny13 can be used to supervise an ATmega128.

7.2. March B Test Description

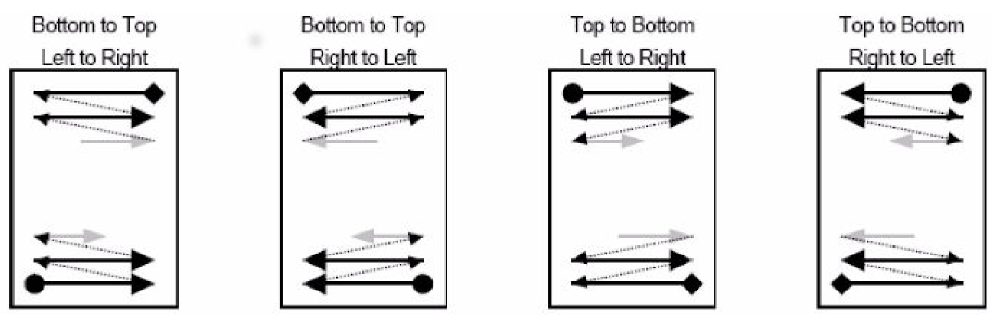
Static Random Access Memories' test is performed using March B algorithm. Using March conventions, this algorithm could be described as follow:

{**Down**(w0); **Up**(r0,w1,r1,w0,r0,w1); **Up**(r1,w0,w1); **Down**(r1,w0,w1,w0); **Down**(r0,w1,w0)}

Meaning:

1. Initialize the whole memory to '0' from last to first bit.
2. For each bit, from first to last one, read '0', write '1', read '1', write '0', read '0', write '1', then go to next bit.
3. For each bit, from first to last one, read '1', write '0', write '1', then go to next bit.
4. For each bit, from last to first one, read '1', write '0', write '1', write '0', then go to next bit.
5. For each bit, from last to first one, read '0', write '1', write '0', then go to next bit.

The walking order specified in the algorithm (Down, Up, Up, Down, Down) refers to physical addresses, not logical ones. It is relative to the first 'Down' which physically can correspond to four implementations:



Fault Coverage:

This algorithm can catch the following miss-processes in SRAM areas:

Stuck At Faults (SAF): The bit is stuck at a state and cannot be written.

Transition Faults (TF): Stuck at a state, once in that state. For instance; the bit's value is '1' it can be written to '0', but once at '0' it cannot be set to '1' any more.

Inversions Coupling Faults (CFin): A transition on a bit inverts the state of a second bit.

Idempotent Coupling Faults (CFid): A transition on a bit forces a second bit to a certain state.

Coupling Faults Bridging (BF): Two bits are shorted. The resulting state of those bits is a logical AND or a logical OR between the previous states of those bits.

State Coupling Faults (SCF): A coupled cell is forced to a certain value only if a coupling cell is in a given state.

Note: In some very specific configurations of linked faults, some of those faults could be undetected. Those configurations are considered to impact less than 1 part per million.

Getting a fail during this test means that one of the previous faults was found without distinction of any kind.

8. Revision History

Doc. Rev.	Date	Comments
7715C	09/2016	Firmware support for ATtiny817 is added
7715B	04/2008	
7715A		Initial document release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**