Freescale Semiconductor
Application Note

Document Number: AN3257
Rev. 1, Draft A, 02/2007

Meeting IEC 60730 Class B Compliance with the MC9S08AW60

by: Dugald Campbell
Freescale Semiconductor
Microcontroller Division
East Kilbride, Scotland

1 Introduction

With the introduction of the IEC 60730, household appliance manufacturers must now consider introducing new design enhancements to their automatic electronic controls that ensure safe and reliable operation of their component.

IEC 60730 discusses mechanical, electrical, electronic, environmental, endurance, EMC, abnormal operation of AC appliances. Most specifically for MCUs, Annex H: Requirements for Electronic Controls details new test and diagnostic methods to ensure the safe operation of embedded control hardware and software for appliances.

Today, the majority of automatic electronic controls for appliance products utilize single-chip microcontrollers with embedded memory and input/output peripherals, known as MCUs. Manufacturers develop real-time embedded software that executes in the MCU and provides the hidden intelligence to control the home appliance.

Contents

Annex H. of IEC 60730 has three software classifications for automatic electronic controls:

1. Class A Control functions, which are not intended to be relied upon for the safety of the equipment. Examples are: room thermostats, humidity controls, lighting controls, timers, and switches.
2. Class B Control functions, which are intended to prevent unsafe operation of the controlled equipment. Examples are: thermal cut-offs and door locks for laundry equipment.
3. Class C Control functions, which are intended to prevent special hazards (e.g., explosion of the controlled equipment). Examples are: automatic burner controls and thermal cut-outs for closed, un-vented water heater systems.

Large appliance products, such as washing machines, dishwashers, dryers, refrigerators, freezers, and cookers/stoves will tend to fall under the classification of Class B. An exception is an appliance that could cause an explosion, such as a gas-fired controlled dryer, which would fall under class C.

This paper discusses the key components to be tested on a Class B electronic control with respect to a single-chip microcontroller implementation.   Examples of how to implement these new tests/checks for these components is described using an 8-bit MC9S08AW60 MCU from Freescale Semiconductor.

2 Class B Requirements

IEC 60730 specifies that the manufacturer of the automatic electronic control must design its software using one of the following structures:

- Single channel with functional test
- Single channel with periodic self test
- Dual channel without comparison

In a single channel with functional test structure, software is designed using a single CPU to execute functions as required. Prior to shipment, a functional test is performed to ensure that all critical features are functioning reliably.

In a single channel with periodic self-test structure, software is designed using a single CPU to execute functions as required, but periodic self tests occur while the electronic control is executing in its application. The CPU is expected to regularly check the various critical functions of the electronic control without conflicting with the end application's operation.

In a dual channel without comparison structure, software is designed using two CPUs to execute on critical functions. Before executing a critical function, both CPUs are required to share that they have completed their corresponding task. For example, when a laundry door lock is released, one CPU stops the motor spinning the drum, and the other CPU checks the drum speed to verify it had stopped. See Figure 1.

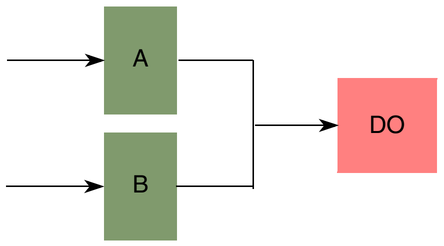


Figure 1. Dual Channel without Comparison Structure

Dual channel structure implementations will be more costly because two CPUs (or two MCUs) are required. Also, the need for two devices to regularly communicate with each other adds complexity. Single channel with functional test is the most popular structure implemented today, and most appliance manufacturers are moving to single channel with periodic self test implementations.

Annex H. Table H.11.12.7 details the components that must be tested, depending on the software classification. Generally, each component offers optional measures to verify/test the corresponding components, providing the manufacturer flexibility. Throughout this paper, the intention is to employ the most cost efficient measure by using, where possible, on-board features of a single-chip microcontroller.

To fulfill Class B IEC 60730 compliance, manufacturers of electronic controls are required to the test 14 components listed in Table 1. Only the eight components relevant to single-chip microcontroller implementations are discussed.

Table 1. Electronic Controls for Testing

	Class B IEC 60730 Components Required to be Tested on Electronic Control (see Table H.11.12.7 in Annex H)	Fault/Error
1	1.1 CPU registers	Stuck at
2	1.3 CPU program counter	Stuck at
3	2. Interrupt handling and execution	No interrupt or too frequent interrupt
4	3. Clock	Wrong frequency
5	4.1 Invariable memory	All single bit faults
6	4.2 Variable memory	DC fault
7[1]	4.3 Addressing (relevant to variable/invariable memory)	Stuck at
8[1]	Internal data path	Stuck at
9[2]	Addressing	Wrong address
10	External communications	Hamming distance 3
11	Timing	Wrong point in time/sequence
12[3]	I/O Periphery	Fault conditions specified in H.27
13[3]	7.2.1 Analog A/D and D/A converters	Fault conditions specified in H.27
14[3]	7.2.2 Analog multiplexor	Wrong addressing

[1] 7 and 8 are for expanded MCU and are covered by tests 5 and 6 on single-chip MCU systems

[2] 9 is for expanded memory MCU systems only

[3] 12, 13, and 14 are production plausibility checks

3 CPU Registers Stuck At Faults

The CPU is the main component of an automatic electronic control and IEC 60730 requires that the registers be checked for stuck at faults, insuring that bits in the registers are not stuck at a value.

Two possible measurements are provided to meet IEC 60730 Class B:

- **Functional test H.2.16.5** — A single-channel structure which introduces test data to the functional unit prior to its operation.
- **Periodic self-test H.2.16.6** — A single-channel structure which periodically tests components of the control during operation using either:
  - **Static memory test H.2.19.6** — a fault/error control technique which detects only static errors.
  - **Word protection with single-bit parity H.2.19.8.2** — a fault/error control technique which adds a single bit to each word in the memory area under test and saves it, creating either even or odd parity. A parity check is conducted as each word is read.

Word protection with single-bit redundancy requires additional hardware to monitor the CPU and additional memory to store the parity of each memory location. This type of circuit adds significant cost to a MCU product so it is not an appliance manufacturer’s first choice. The alternatives, using functional tests prior to shipment or in-line periodic self-tests that cost less to implement, are viewed more favorably by appliance manufacturers.

The MC9S08AW60 MCU utilizes the 8-bit HCS08 CPU. The HCS08 CPU comprises an 8-bit accumulator, a 16-bit index register (H:X), a 16-bit stack pointer, a 16-bit program counter (PC), and a 6-bit condition code register (CCR).

With the HCS08 CPU’s von-Neuman architecture, it is very easy to implement small routines to test the CPU registers for stuck at faults. Example assembly routines follow. These small routines can be called on power-up and periodically, take only tens of microseconds to execute, and will verify that the CPU registers are not stuck.

Example -1. Accumulator Stuck At Test Code

```
/* 8bit Accumulator check for static error

LDA      #$55
STA      PortA    ; optional output of ACC to external world
CMP      #$55
BEQ      next1
BRA      Error
Next1:   COMA
STA      PortA    ; optional output of ACC to external world
CMP      #$AA
BEQ      next2
BRA      Error
Next2
```

Example -2. Index Register Stuck At Test Code

```
/* Index Register
         LDHX     #$5555
         STNX     temp1     ; stores H in temp1 and X in temp 1+1
         LDA      temp1+1
STA      PortA              ; optional output of ACC to external world
         CMP      #$55
BEQ      next3
BRA      Error
Next3:   STX      PortA
         CPX      #$55
BEQ      next4
BRA      Error
Next4:   LDHX     #$AAAA
         STHX     temp1     ; stores H in temp1 and X in temp1+1
         LDA      temp1+1
STA      PortA              ; optional output of ACC to external world
         CMP      #$AA
BEQ      next5
BRA      Error
Next5    STX      PortA
         CPX      #$AA
BEQ      next6
BRA      Error
```

Example -3. Stack Pointer Stuck At Test Code

```
/* Stack pointer check
Next6    RSP
         LDA      $FF
         STA      temp1     ; store reset stack pointer data to temp
         PULA
         CMP      temp1
         BEQ      next7
         BRA      error
Next7    LDA      $AAAA
         STA      temp1
         LDHX     #$AAAA
         TXS                ; transfer AAAA -1 into SP
         PULA               ; contents of $AAAA into ACC
         CMP      temp1
         BEQ      next8
         BRA      error
Next8    LDA      $5555
         STA      temp1
         LDHX     #$5555
         TXS                ; transfer 5555 -1 into SP
         PULA               ; contents of $5555 into ACC
         CMP      temp1
         BEQ      next9
         BRA      error
Next9
```

Example -4. Program Counter Stuck At Test Code

```
/* Program Counter Check

/* Pre load in Flash address $5555 CD AA AA 81 (jsr $AAAA, rts)
/* $AAAA CD     $abcd 81 (jsr $abcd, rts)
        ORG     $abcd
        LDA     $FF       ; examine PC pushes onto stack.
        CMP     #$55
        BEQ     Here 1
        JMP     error
Here1:  LDA     $FE
        CMP     #$55
        BEQ     Here2
        JMP     error
Here2:  LDA     $FD
        CMP     #$AA
        BEQ     Here3
        JMP     error
Here 3: LDA     $FC
        CMP     #$AA
        BEQ     Here4
        JMP     Error
Here 4: LDA     #$DC      ; send alternative pass value
        RTS

Next 9: RSP               ; reset SP
        LDA     #$FF      ; clear 1st 4 locations on stack
        STA     $FF
        STA     $FE
        STA     $FD
        STA     $FC
        JSR     $5555
        CMP     #$DC      ; if Pushes of PC were confirmed then PC has been checked.
        BEQ     NextA
        BRA     error
```

```
NextA: ................................
```

Example -5. Condition Code Register Stuck At Test Code

```
/* CCR condition code register.

NextA    LDA      $55
         TAP               ; transfer setting C, N, & H flags
         BCS      nextB    ; Carry is set
         BRA      error
nextB    BNE      nextC    ; Z is clear
         BRA      error
nextC    BMI      nextD    ; N is set
         BRA      error
nextD    BMC      nextE    ; I = =0 ?
         BRA      error
nextE    BHCS     nextF    ; H==1 ?
         BRA      error
nextF    BLT      nextG    ; V == 0,N==1
         BRA      error
nextG    COMA
         TAP               ; transfer $AA in CCR setting V, I Z flags
         BCC      nextH    ; Carry is clr
         BRA      error
nextH    BEQ      nextI    ; Z is set
         BRA      error
nextI    BPL      nextJ    ; N is zero
         BRA      error
nextJ    BMS      nextK    ; I ==1 ?
         BRA      error
nextK    BHCC     nextL    ; H==0 ?
         BRA      error
nextL    BLT      nextM    ; V == 1,N==0
         BRA      error
```

```
NextM: ................................
```

4 Program Counter Stuck At Faults

By supplying the address, the CPU's program counter allows access to all memory and peripherals. If the program counter has a stuck at fault, the MCU would not function correctly.

Four possible measurements are given to test the main CPU's program counter:

- **Functional test H2.16.5**
- **Periodic self-test H2.16.6**
- **Independent time-slot monitoring or H.2.18.10.4** — a fault/error control technique which monitors the programming function and sequence by periodically triggering timing devices with an independent time base. An example is a watchdog timer.
- **Logical monitoring of the program sequence. H.2.18.10.2** — a fault/error control technique which monitors the logical execution of the programming sequence. Examples are the use of counting routines or selected data in the program itself or by independent monitoring devices.

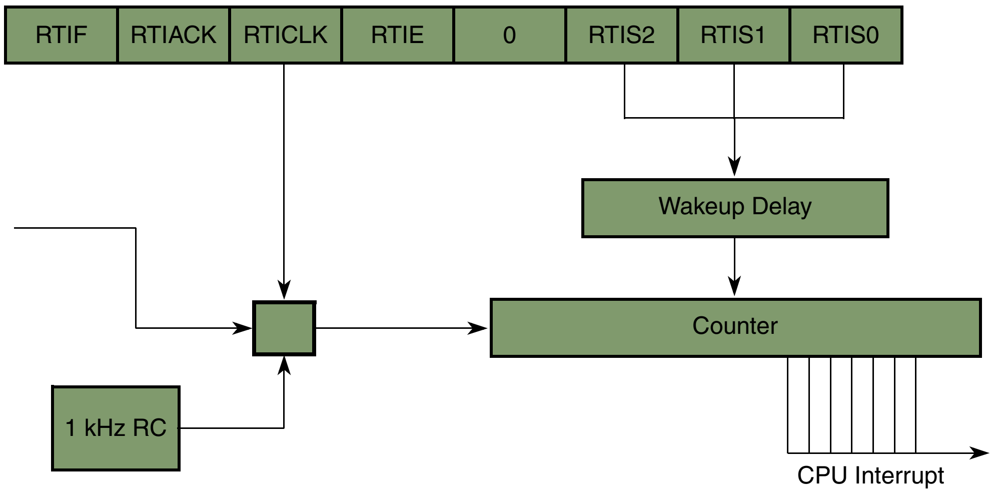
The assembly routine supplied to check the program counter has no stuck at fault by either a functional test or periodic self test.

Alternatively, the 9S08AW60 has a real-time interrupt (RTI) feature that can be configured to be clocked by an internal 1 kHz RC oscillator, providing interrupts to the CPU every 8 ms/32 ms/64 ms/128 ms/ 256 ms/512 ms/1.024 seconds. The internal RC oscillator is independent from the CPU clock source. The CPU can be clocked from another internal RC oscillator or external clock with options to multiply either source with an on-chip frequency lock loop.

To check program function and sequence, the interrupt service routine (ISR) can create code using the RTI function (clocked by a 1 kHz source) to check program function and sequence. For example, ISR for the RTI could examine the stack to determine where the PC counter was prior to an interrupt and compare with the address saved from the last ISR. If the address is the same for three or more RTIs, an indication that the PC is stuck would trigger a check to see if the PC is caught in a software loop.

Additionally, the 9S08AW60 has a watchdog feature that is clocked from the CPU bus clock to provide a divide of 218 or 213 bus cycles. If the PC was permanently stuck to one value, then a refresh of the watchdog would not occur, resulting in a timeout and an internal reset. The internal reset would force all inputs/outputs to tri-state or a default safe starting condition. If the PC remained stuck on one address, no further CPU execution would occur.

The independently clocked RTI feature of the 9S08AW60 is intended to be used as logical monitoring of the program sequence. Normal tasks would update tokens, such as counts, and within the RTI ISR, the tokens would be analyzed, and a check on the correct logical execution would be verified. If the program sequence did not follow expected logical execution flow, preventive measures could be taken by forcing the MCU into a safe condition.



RTIS2.1.0	1 kHz Clock	External Clock
0.0.0	Disabled	Disabled
0.0.1	8 ms	Disabled/256
0.1.0	32 ms	Disabled/1,024
0.1.1	64 ms	Disabled/2,048
1.0.0	128 ms	Disabled/4,096
1.0.1	256 ms	Disabled/8,192
1.1.0	512 ms	Disabled/16,384
1.1.1	1.04 s	Disabled/32,768

Figure 2. Block Diagram of MC9S08AW60 Real-Time Interrupt

5 Interrupt Handling and Execution – No Interrupt or too Frequent Interrupts

In a real-time embedded application, an MCU will almost always use interrupts to help react to real-time events, and to help prioritize the CPU on critical tasks or functions. IEC 60730 requires verification that the interrupt functionality for a critical function occurs as predicted. If no interrupt occurs, or too many occur, the electronic control functions safely. Two possible measures to meet IEC 60730 Class B are:

- **Functional test H.2.16.5** — a single-channel structure which introduces test data to the functional unit prior to its operation.
- **Independent time-slot monitoring or H.2.18.10.4** — a fault/error control technique which monitors the programming function and sequence by periodically triggering timing devices with an independent time base. An example is a watchdog timer.

Using the time-slot monitoring method, the MC9S08AW60 has the following possible interrupts:

- Software interrupt instruction (SWI)
- IRQ pin
- Low voltage detection
- Timer/PWM module 1
- Timer/PWM module 2
- SPI
- SCI 1
- SCI 2
- IIC
- Keyboard
- ADC
- Real-time interrupt (RTI)

Each interrupt function can deploy a count byte that is incremented within each of the corresponding ISR routines. Count bytes can be created using the 9S08AW60s RTI function code they can be created to verify the number of interrupts that have occurred and decide if the control system is executing in the correct manner. For example, if the RTI is set to interrupt every 8 ms, there might be two to three SCI interrupts within this time period, one timer overflow interrupt every 16 RTIs, and a timer capture interrupt every 300 RTIs. Code can be written to track and compare the number of RTIs with the number of occurrences of other interrupts.

Alternatively, functional tests can independently validate each interrupt function prior to the control unit's operation. Using the S908AW60's background debug mode, the user can download routines to RAM and test each available interrupt in turn. Some interrupts may require external stimulus to occur (e.g., receive an SCI or IIC byte, or timer capture input) and can be created by using other features of the MCU prior to testing. For example, the IIC interrupts can be tested by writing a software routine to emulate a IIC transmission and feeding the appropriate pins to the IIC pins.

Or, the control unit may be set up in a pseudo end application environment featuring application of forced stimuli applied and monitoring of the application's functions. The background debug mode (BDM) feature allows set up and monitoring of interrupt watch points to ensure interrupts have occurred without influencing operation of the user application.

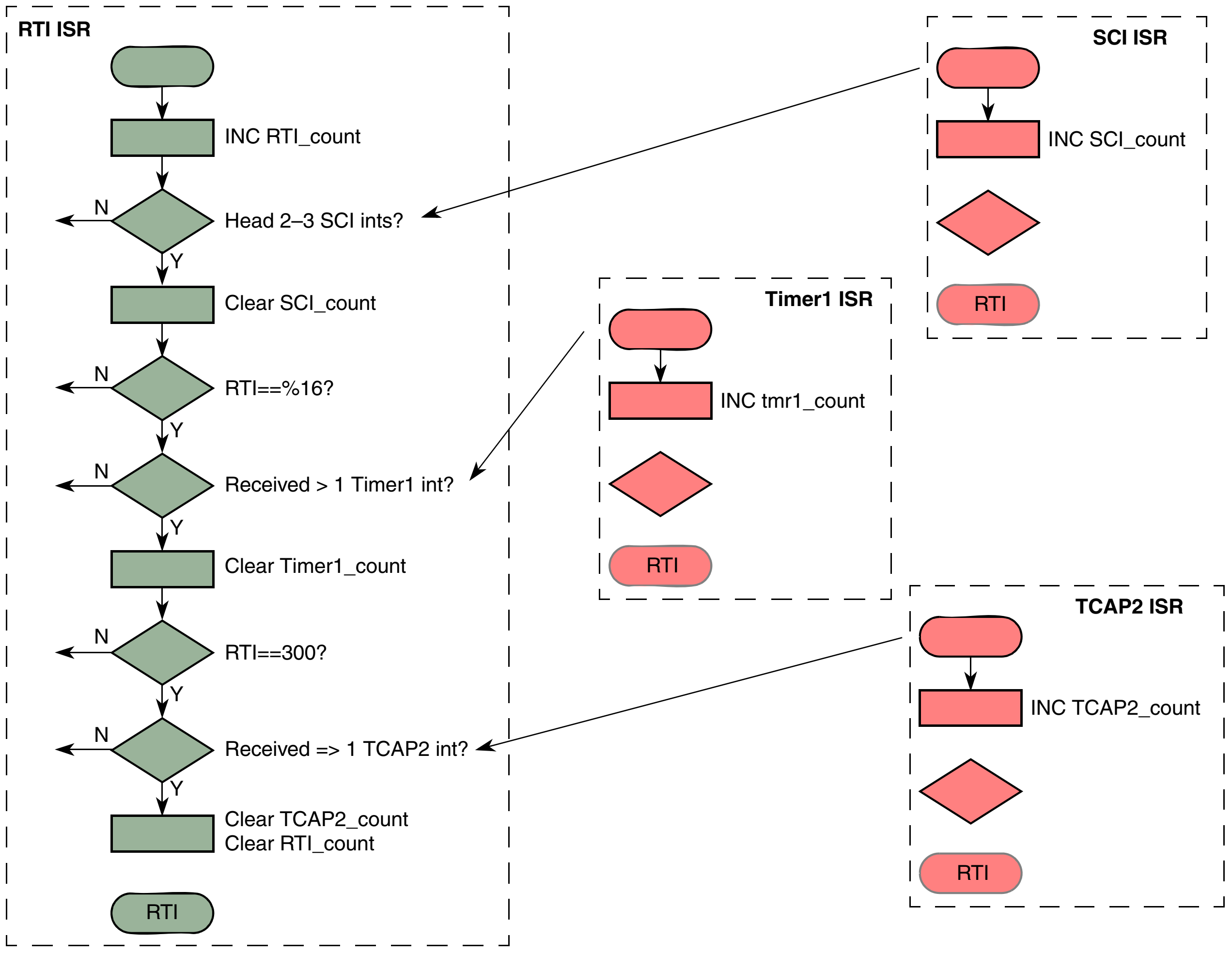


Figure 3. Using a Count Token in ISR Routines

6 Clock — Wrong Frequency

The following options can ascertain that the main system clock (the CPU bus clock is functioning reliably at the correct frequency (neither too slow nor too fast):

- **Frequency monitoring H.2.18.10.1** — a fault/error control technique which compares the clock frequency with an independent fixed frequency. An example is comparison with the line frequency.
- **Independent time-slot monitoring or H.2.18.10.4** — a fault/error control technique which monitors the programming function and sequence by periodically triggering timing devices~~,~~ with an independent time base. An example is a watchdog timer.

The 9S08AW60’s RTI function is clocked from an independent 1 kHz internal RC oscillator. On each RTI interrupt, the ISR can ensure the CPU clock frequency is running as intended by reading one of the timer/PWM modules, taking a time stamp, and comparing with the last RTI reading. The timer/PWM modules are clocked from a multiple of the CPU bus clock.

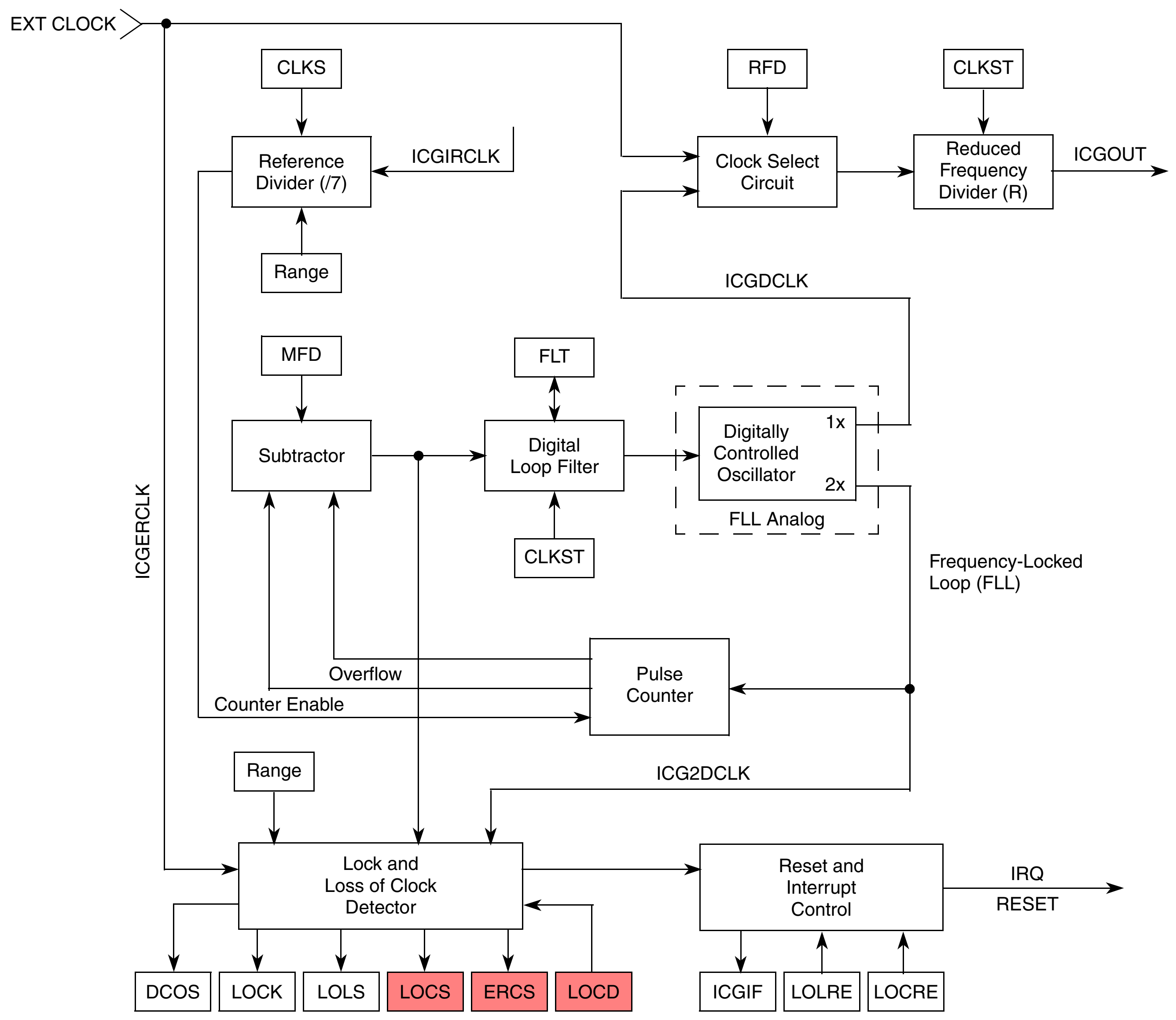


Figure 4. Detailed Frequency-Locked Loop Block Diagram

A monitor of the external clock's (crystal or external clock input) frequency, known as loss of clock detect (LOCD), is located in the MC9S08AW60's internal clock generator (ICG).

Loss of clock detection is enabled by clearing LOCD = 0 on the ICG control 1 register (ICGC1). A reset will be forced if the clock frequency falls below a minimum value (less than 500 kHz or 25 kHz, depending on range setting) by setting LOCRE = 1 (loss of clock reset enable) on the ICG control 2 register (ICGC2).

If the MC9S08AW60 is configured to utilize the FLL to multiply an external input clock (FEE or FEI mode) and the input clock is too fast, the FLL will loose lock, which is detected within the ICG via the loss of lock status (LOCS) bit the ICGC1 register. A reset will occur if the loss of lock reset enable (LOLRE) is set.

7 Invariable Memory — All Single-Bit Faults

Invariable memory is generally seen as the program memory using either NVM technology, such as EEPROM, EPROM, or FLASH and masked ROM.

Three optional measures can verify that the invariable memory has no single bit faults:

1. **Periodic modified checksum H.2.19.3.1** — a fault/error control technique which generates and saves a single word representing the contents of all words in memory. During self test, a checksum is formed from the same algorithm and compared with the saved checksum. This technique recognizes all odd errors and some of the even errors.
2. **Multiple checksum H.2.19.3.2** — a fault/error control technique which generates and tests separate words representing the contents of the memory areas to be tested. During self test, a checksum is formed from the same algorithm and compared with the saved checksum for that area. This technique recognizes all odd errors and some of the even errors.
3. **Word protection with single-bit redundancy H.2.19.8.2**

Periodic checksum can be created very easily with software. Some examples of byte checksum routines written in assembler and C are provided in Table 2 and Table 3, respectively. A checksum for the full Flash array of the 9S08AW60 will take ~ 108 ms at 20 Mhz bus speed (when the routine is in assembler).

Checksum routines are basically the result of exclusive-ORing all of the Flash locations together.

Table 2. One-Byte Checksum Execution Time (Routine in Assembler)

Number of Bytes	Cycles	8 MHz Bus	20 MHz Bus
1024	33,873	0.004234125	0.00169365
16,384	541,968	0.067746	0.0270984
32,768	1,083,936	0.135492	0.0541968
65,536	2,167,872	0.270984	0.1083936

Table 3. One-Byte Checksum Execution Time (Routine in C)

Number of Bytes	Cycles	8 MHz Bus	20 MHz Bus
1024	56,376	0.007047	0.0028188
16,384	902,016	0.112752	0.0451008
32,768	1,804,032	0.225504	0.0902016
65,536	3,608,064	0.451008	0.1804032

Example -6. Memory Checksum Routine in Assembler

```
/*********************************************************/
//
//
//
//    Single byte Checksum Routine
//
//    Date: 16 May 2005       Copyright: Freescale Ltd
//
//    Author: Dugald Campbell
//
//    Title:  checkasmsum_sb()
//
//    Entry: 0       Exit: single byte checksum
//
/*********************************************************/

char checkasmsum_sb(void)
{
    // Local Variables

    char chksum;         // returned one byte checksum

    // Memory Array to be examined co-ordinates.

    // set initial value of chksum
    chksum = 0;

    asm {
    LDA #0xC8        // EOR (ext) instruction
    STA RAM_subr
    LDA #(MEMORY_START/0x100)
    STA RAM_Maddr
    LDA #MEMORY_START
    STA RAM_Laddr
    LDA #0x81
    STA RAM_RTS
NEXT_EOR:
    LDA chksum
    JSR RAM_subr
    STA chksum

    INC RAM_Laddr
    BNE NEXT_EOR     // continue until we get to $xx00

    INC RAM_Maddr
    LDA RAM_Maddr
    CMP #(MEMORY_END/0x100)
    BNE NEXT_EOR
LAST_CHKS:
    LDA RAM_Laddr
    CMP #MEMORY_END
    BEQ  NO_MORE
FURTHER_EOR:
    LDA chksum
    JSR RAM_subr
    STA chksum
    INC RAM_Laddr
    BRA LAST_CHKS
NO_MORE:
    LDA chksum

        }

     return chksum;

 }  /* end of checksum_sb function */
```

Example -7. Memory Checksum Routine in C

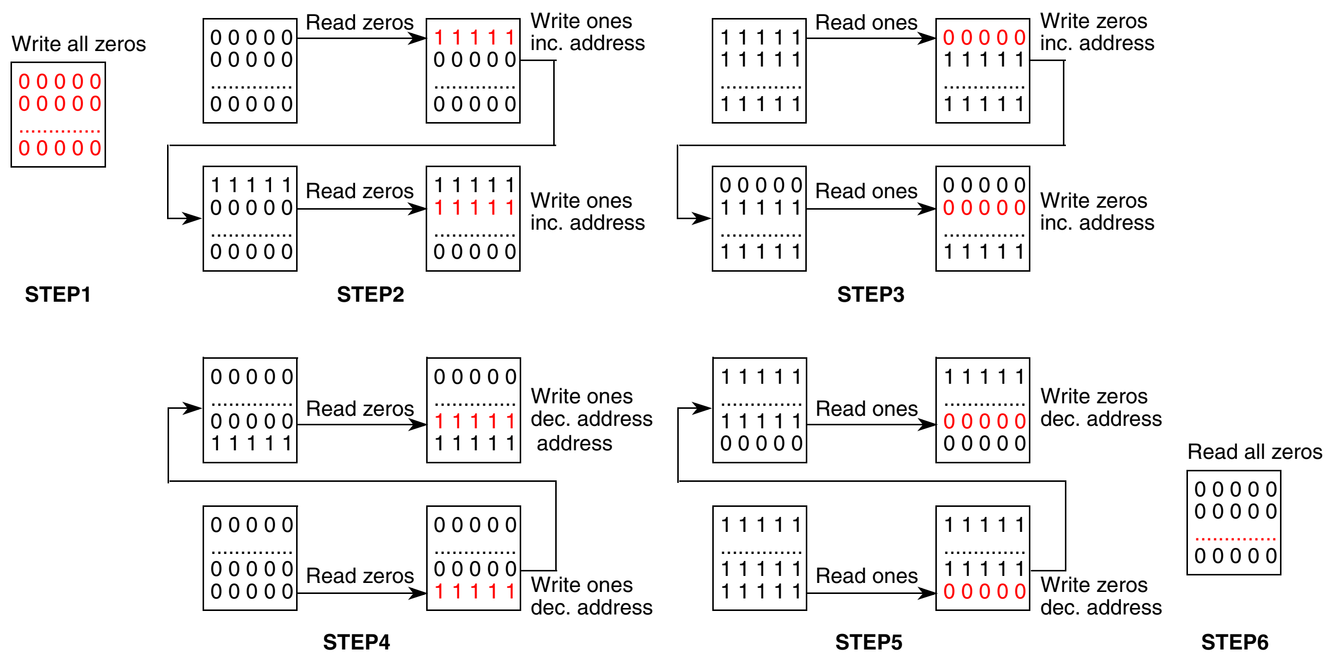
```
/**********************************************************/
//
//
//
//    Single byte Checksum Routine
//
//    Date: 16 May 2005       Copyright: Freescale Ltd
//
//    Author: Dugald Campbell
//
//    Title:  checksum_sb()
//
//    Entry: 0       Exit: single byte checksum
//
/**********************************************************/

char checksum_sb(void)
{
    // Local Variables
    char *memory_start;  //16bit pointer to array start
    char *memory_end;    //16bit pointer to array end
    short memory_range;  //resultant memory range
    char chksum;         //returned one byte checksum



    // Memory Array to be examined co-ordinates.
    memory_start = (char *)MEMORY_START;
    memory_end   = (char *)MEMORY_END;
    memory_range = (memory_end - memory_start);

    // set initial value of chksum
    chksum = 0;

    // for loop to step through memory array

    for ( ; memory_range != 0; memory_range-- )

         {
                  // EOR chksum with data at addr mem_start+mem_range
                   chksum = chksum ^ *(memory_start+memory_range);

         }

    // memory start not EOR'd within "for" loop so complete by
    // EOR with 1st byte of memory
    chksum = chksum ^ *(memory_start);


    return chksum;

 }  /* end of checksum_sb function */
```

8 Variable Memory — DC Fault

Variable memory is generally seen as the data memory normally implemented in volatile RAM. Two optional measures can verify that the invariable memory has no single-bit faults:

1. **Periodic static memory test H.2.19.6** — a fault/error control technique which is intended to detect only static errors
2. **Word protection with single-bit redundancy H.2.19.8.2** — a periodic static memory test can be developed in software (assembler will help reduce code size and execution time) to clear all RAM to $00, verify, set all RAM to $FF, verify and a checkerboard of $AA, verify, $55 & verify. This routine will test for any static errors within the RAM.

There are many periodic static memory algorithms for checking for DC faults. The March C algorithm (van de Goor,1991) is commonly used and shown in Figure 5.



STEP1 — Write all zeros to array
STEP2 — Starting at lowest address, read zeros, write ones, increment up array.
STEP3 — Starting at lowest address, read ones, write zeros, increment up array.
STEP4 — Starting at highest address, read zeros, write ones, decrement down array.
STEP5 — Starting at highest address, read ones, write zeros, decrement down array.
STEP6 — Read all zeros from array.

Figure 5. A Commonly Used Periodic Static Memory Algorithm

It is necessary to develop specific custom tests on each register for input/output registers which are also variable memory. When setting a value, ensure that no unwanted condition occurs from the function. Writing a value to an SPI control register may enable the SPI and initialize the send of a byte; the application may not support this action and may require another approach to test this register.

9 Addressing Stuck At Fault in Variable and Memory

This component is the addressing mechanism for accessing the memory and other functions. For Class B products, only one measure is available to test this component:

- **Word protection with single-bit parity H.2.19.8.2** — a fault/error control technique which adds a single bit to each word in the memory area under test and saves it, creating either even or odd parity. A parity check is conducted as each word is read.

The addressing component 4.3 was specified for electronic control systems that utilize microcontrollers with external memory devices or for multiple microcontrollers sharing external memory devices.

With a single-chip microcontroller, such as the MC9S08AW60, all addressing to memory is internal. Measures such as the periodic static memory test on variable memory (RAM) and the periodic checksum on invariable memory (Flash) would highlight a stuck at fault on the internal address bus. For example, if the address bus had a stuck at fault, both of these tests would produce an error.

10 Internal Data Path Stuck At Fault

Similar to the previous component, Class B specifies a component internal data path for electronic control systems that utilize microcontrollers with external memory devices or for multiple microcontrollers sharing external memory devices.   On a single-chip microcontroller, such as the MC9S08AW60, implementing the periodic static memory test on variable memory (RAM) and the periodic checksum on invariable memory (Flash) would highlight a stuck at fault on the internal data bus.

11 Addressing — Wrong Address

Class B specifies Component 5.2 Addressing – wrong address to test the addressing mechanisms for microcontrollers using external memory; it is not required for single-chip microcontrollers.

12 External Communications – Hamming Distance 3

This component checks the reliability of communications to other external modules, such as external sensors and actuators that are not placed on the electronic control PCB.

Four possible measures are available to test this component:

- **Word protection with multi-bit redundancy including address H.2.19.8.1**
- **CRC-single word, H.2.19.4.1** — a fault/error control technique which generates a single word to represent the contents of memory. During self test, the same algorithm generates another signature word, which is compared with the saved word. The technique recognizes all one-bit errors and a high percentage of multi-bit errors.
- **Transfer redundancy H.2.18.2.2** — a form of code safety which transfers data at least twice in succession and then compares the data. This technique will recognize intermittent errors.
- **Protocol test H.2.18.14** — a fault/error control technique which transfers data to and from computer components to detect errors in the internal communications protocol.

Transfer redundancy is probably the most cost-efficient and easiest measure to deploy to ensure reliable and safe external communications. It can be implemented easily in the software used in the communications protocol to send and receive data twice before executing on commands or results received.

CRC single word check is also easily implemented by appending a 16-bit signature of the data to the message being communicated. For communication protocols that exchange small data/message packets and have no time-constraints, CRC can be deployed in software as the following example shows.

Using C code, a CRC-CCITT 16-bit routine requires 120 bytes of program memory and takes approximately 650 instructions cycles to rotate one byte through the CRC signature generator (for a 20 MHz HCS08 CPU, approximately 31 μs).

Example -8. 16-Bit CRC Routine in C

```
/*******************************************************/
//
//    Initialise CRC CCITT Routine
//
//    Entry: 0       Exit: crc_16=$FFFF
//
/*******************************************************/

void init_crcCCITT(void)
         {
                  crc_16 =0xFFFF;
         }

/*******************************************************/
//
//
//
//    Update CRC CCITT Routine
//
//    Date: 18 May 2005       Copyright: Freescale Ltd
//
//    Author: Dugald Campbell
//
//    Title:  update_crcCCITT()
//
//    Entry: byte to rotate into CRC engine  Exit: 16bit CRC
//
/*******************************************************/


void update_crcCCITT(char f)
{
    // Local Variables
    //char f;          // entry for function
    char i;             // for loop purposes
    char c;                    // carry from shift left

    // Rotate Left
    for (i=8 ; i>0 ; i--)
    {


        if (crc_16&0x8000) c = 1; // MSB is 0 as crc_16 is signed
        else c=0;

           // set c with MS bit
         crc_16=crc_16<<1; // rotate left all by one bit

         // Now carry out EOR with root bits
         switch (c) // based on Carry
         {
         case 0x0: if (f&0x01) crc_16=crc_16|0x0001; // if LS bit of f
Set B0=1
               break;                     // else crc16 no change on EOR
         case 0x1: if (f&0x01) crc_16=crc_16^0x1020; // need EOR if
Carry 1
                          else crc_16=crc_16^0x1021;
                          break;
         }

         f=f>>1; // shift the supplied byte by 1

    }
```

13 External Communications Time — Wrong Point/Wrong Sequence in Time

Four possible measures are given to test this component.

- **Time-slot monitoring H.2.18.10.4**
- **Scheduled transmission H.2.18.18** — a communication procedure which allows information from a particular transmitter to be sent only at a predefined point in time and in sequence; otherwise, the receiver will treat it as a communication error.
- **Logical monitoring H.2.18.10.2** — a fault/error control technique which monitors the logical execution of the program sequence.
- Comparison of redundant communication channels by either:
  - **Reciprocal comparison H.2.18.15** — reciprocal comparison — a fault/error control technique used in dual channel (homogeneous) structures which compares data reciprocally exchanged between the two processing units. Reciprocal refers to an exchange of similar data.
  - **Independent hardware comparator H.2.18.3** — a device used for fault/error control in dual channel structures. The device compares data from the two channels and initiates a declared response if a difference is detected.

The MC9S08AW60 has several dedicated communications interfaces: SCIs, SPI, and IIC functions. Using these modules for communication for Class B requires time slot monitoring to ensure that communication occurs at the correct point in time. For example, if the SCI is configured as master node, the software will deploy code that ensures reception of the slave transmission within a specified time, or a fault condition will be initiated.

The RTI function can be used with its independent time base to operate as a scheduler. Working in conjunction with the timer/PWM module, the RTI function can check how external communications occur and take a time stamp using one of the timer/PWM modules. Using either the output compare function or a semaphore system (as used in real-time operating systems/kernels), the RTI function can monitor the timing of external communications to ensure they occur correctly and recognizably.

14 Plausibility Check

Plausibility check H.2.18.13 — a fault/error control technique which checks program execution, inputs or outputs for inadmissible program sequence, timing or data. Examples include the introduction of an additional interrupt after the completion of a certain number of cycles or checks for division by zero.

For Class B controls, a plausibility check is the only measure required to test the following components:

7.	I/O Periphery — fault conditions specified in H.27
7.2.1	A/D & D/A converters — fault conditions specified in H.27
7.2.2	Analog Multiplexer — wrong addressing

Plausibility checks on I/O periphery may consist of checking for known and unknown conditions on port pins, e.g. if 2 out of 4 inputs are always zero, then reading of 3 zeroes is faulty condition. For output pins, additional inputs are deployed to monitor that their condition is correct.

Plausibility check on A/D converters requires the user to force known calibrated dc voltage levels (usually a voltage reference is available) and comparing these with other known signals. Also, boundary conditions placed around the expected analogue values temperature are monitored, i.e. a temperature sensor will provide a value of 2.2V to 4.3V for operating temperatures of -40 to 180’C, and a value outside of this would indicate a faulty condition.

On D/A systems deploying an A/D channel to check on the various values, the D/A is set to provide a satisfactory check.

For analog multiplexor, a mechanism to force a known condition on all selectable channels is required, e.g. on a 3-channel A/D the application can place the applied signals to known and different values to allow testing of the multiplexor (a circuit could switch in various potential divider circuits on each A/D channel).

15 Summary and Conclusions

This application note demonstrates how using the MC9S08AW60 allows manufacturers of automatic electronic controls for household AC appliances to easily meet Class B IEC 60703 requirements by:

- Deploying diagnostic software routines which test reliability of CPU operation and both variable and invariable memory
- Designing deterministic software which uses the real-time interrupt module’s independent RC oscillator to test interrupts and CPU clock frequency
- Monitoring the ICS module’s loss of clock and loss of lock features, as well as monitoring additional CPU clock frequency

THIS PAGE IS INTENTIONALLY BLANK

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3257
Rev. 1, Draft A
02/2007