Freescale Semiconductor
Mask Set Errata

MSE812A4_0K51E
Rev. 1, 10/2004

Mask Set Errata for 68HC812A4, Mask 0K51E

Introduction

This mask set errata applies to this 68HC812A4 MCU mask set:

- 0K51E

MCU Device Mask Set Identification

The mask set is identified by a 5-character code consisting of a version number, a letter, two numerical digits, and a letter, for example 2J88Y. All standard devices are marked with a mask set number and a date code.

MCU Device Date Codes

Device markings indicate the week of manufacture and the mask set used. The date is coded as four numerical digits where the first two digits indicate the year and the last two digits indicate the work week. For instance, the date code “0401” indicates the first week of the year 2004.

MCU Device Part Number Prefixes

Some MCU samples and devices are marked with an SC, PC, or XC prefix. An SC prefix denotes special/custom device. A PC prefix indicates a prototype device which has undergone basic testing only. An XC prefix denotes that the device is tested but is not fully characterized or qualified over the full range of normal manufacturing process variations. After full characterization and qualification, devices will be marked with the MC or SC prefix.

Errata Summary

HC12_AR_289: SPI
SPI in Conjunction with General-Purpose I/O

Description

Not a functional problem, but operation of general-purpose I/O may be confusing when used in conjunction with SPI.

Workaround

Avoid getting a mode fault: do not set SPE and MSTR bits with active low on the SS pin when DDRS[7] is cleared. Initialize DDRS[7] to be an output before enabling SPI.

HC12_AR_311: ATD
(V_{RH}–V_{RL})/2 Internal Reference Conversion

Description

The (V_{RH}–V_{RL})/2 internal reference conversion result may be $7F, $80, or $81.

Workaround

If the (V_{RH}–V_{RL})/2 internal reference is used (perhaps for system diagnostics), expected pass result may be $7F, $80, or $81.

HC12_AR_527: INT
Disabling Interrupts with I mask bit clear can cause SWI

Description

If the source of an interrupt is taken away by disabling the interrupt without setting the I mask bit in the CCR, an SWI interrupt may be fetched instead of the vector for the interrupt source that was disabled.

Workaround

Before disabling an interrupt using a local interrupt control bit, set the I mask bit in the CCR.

HC12_AR_562: CGM
Interrupt out of STOP with DLY=1

Description

Stop mode cannot be exited using interrupts when DLY=1 depending on where the real-time-interrupt (RTI) counter is when the STOP instruction is executed. The RTI counter is free-running during normal operation and is only reset at the beginning of reset, during power-on-reset, and after entry into stop. The free-running counter will generate a 1-cycle pulse every 4096 cycles. If that pulse occurs at the exact same time that the stop signal from the CPU is asserted, then the OSC is stopped but the internal stop signal will remain low. In this state, the OSC is shut off until reset.

Workaround

- If you are not using the real-time interrupt function, you can wait for an RTI flag before entering into stop to guarantee the counter is in a safe state. When executing the following code, all interrupt sources except for those used to exit stop mode must be masked to prevent a loss of synchronization. A loss of synchronization can occur if an interrupt is processed between the setting of the RTIF and the execution of the STOP instruction. Also, you must enable the RTI counter in the initialization code, set to the fastest RTI time-out period, and the RTIE bit should NOT be set.

```
. BRCLR    RTIFLG,#RTIF,RTIFClr    ; RTIF flag is already clear
  LDAB     #RTIF                   ; if it's set, clear the flag.
  STAB     RTIFLG
RTIFClr:
  BRCLR    RTIFLG,#RTIF,*          ; wait until the RTIFLG is set.
  NOP
  NOP
  NOP
  STOP                             ; enter stop mode
```

- If you are driving a clock in (not using the OSC) then you could set DLY=0.

Some applications are required to exit stop, execute a small amount of code, and then return to stop. In this case, the requirement to wait for RTIF as described in the first bullet of this workaround can unduly extend the duration of code execution and hence increase average I_{DD}. In this situation an alternative workaround may be used:

- On exit from STOP, code may be executed for a defined number of clock cycles before reentering stop mode, without regard to the RTIF flag, when DLY=1. The number of cycles permitted is up to 4095 ECLK cycles, which must include the interrupt sequence used to exit stop mode and the execution of the STOP instruction to reenter stop mode. If this condition is met, the MCU will successfully exit stop by means of an external interrupt. If code execution should take any longer than this defined number of clock cycles, the workaround described in the first bullet should be used.

HC12_AR_600: INT
WAIT can not be exited if XIRQ/IRQ Level deasserts within window of time

Description

The device can get trapped in WAIT mode if, on exiting the WAIT instruction, the deassertion timing of the XIRQ or level-sensitive IRQ occurs within a particular time frame. Only reset will allow recovery. Noise/bounce on the pins could also cause this problem.

Workaround

- Use edge-triggered IRQ (IRQE=1) instead of XIRQ or level-triggered IRQ.
- Use RTI, timer interrupts, KWU or other interrupts (except level-sensitive IRQ or XIRQ) to exit wait. If using RTI, it must be enabled in wait (RSWAI=0) and the COP must be disabled (CME=0).
- Assert XIRQ or level-sensitive IRQ until the interrupt subroutine is entered.
- Add debouncing logic to prevent inadvertent highs when exiting WAIT.

HC12_AR_630: ATD
Word read of $x06E($x1EE) will return $0000 regardless

Description

In the I/O register space, a word read of reserved location $x06E ($x1EE) will return a value of $00 for location $x06F ($x1EF), the port AD data input register, regardless of the logic levels present on the port AD input pins. Note: address locations in parentheses apply to the second A/D converter module if present.

Workaround

To ensure reading the proper data from the Port AD Data Input Register, only access location $x06F ($x1EF) using byte read operations. Do not use a word read.

HC12_AR_645: ECT
PA Overflow flag not set when event is concurrent with write of $FFFF

Description

When the value $FFFF is written to PACA or PACB and, at the same time, an external clocking pulse is applied to the PAC, the pulse accumulator may overflow from $FFFF to $0000, but the pulse accumulator overflow flag [PAFLG,PBFLG] is not set. Same situation may happen with 8-bit pulse accumulators PAC1 and PAC3.

Workaround

The input capture function for the subject channel be enabled prior to writing a value to the PACA or PACB. Write to the pulse accumulator register. Then do one NOP (to allow the input capture to update the interrupt flag) followed by a read of the input capture interrupt flag to see if it set. If yes, a check must be made for a missing pulse accumulator event. Steps for software workaround to see if event happens while writing to PAC:

1. Enable Input Capture on same pin as the pulse accumulator (and same type of event).
2. Clear the appropriate CxF in the timer interrupt flag register.
3. Read PAC and store as “Old PAC”.
4. Calculate desired PAC value and write it to the PAC.
5. Execute 1 NOP.
6. Read CxF in the timer interrupt flag register.

   If flag is not set, done (no events happened while writing to the PAC).

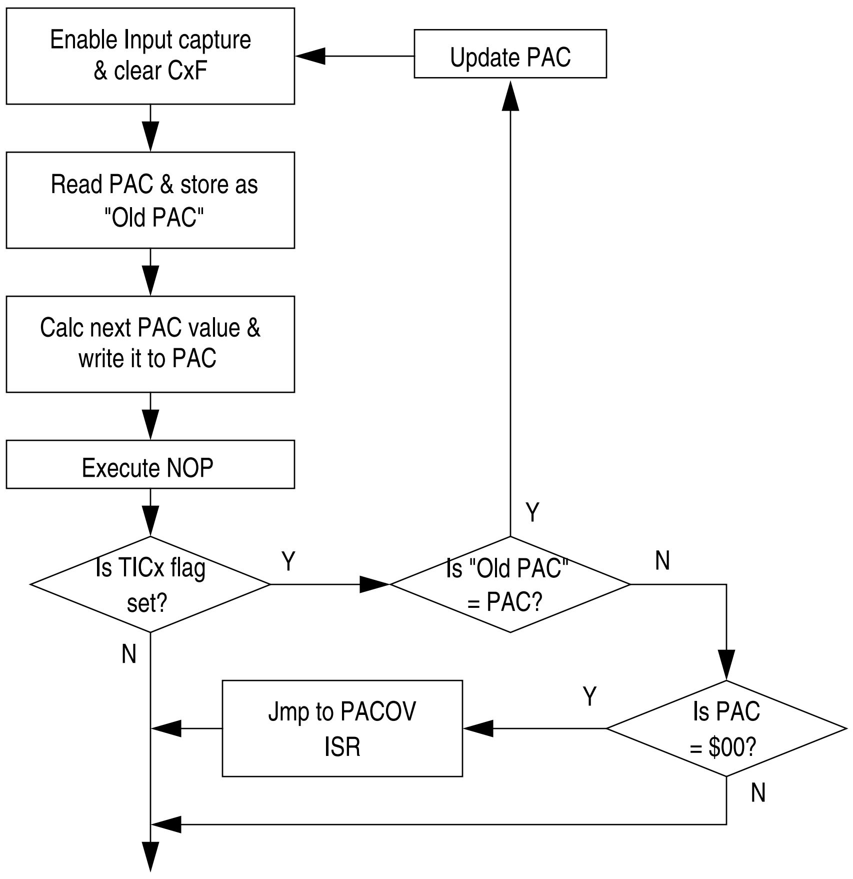
   If flag is set read PAC

      If “Old PAC” = PAC, then update PAC (event happened while writing to PAC and the PAC did not capture it). Note, if the updated PAC value is $00 jump to PACOV ISR.

If “Old PAC” does not equal PAC, does PAC = $00 ?

If yes, jump to PACOV ISR.

If no, done (event happened while writing to the PAC and PAC captured it). Read CxF in the timer interrupt.



Enable Input capture & clear CxF
Update PAC
Read PAC & store as "Old PAC"
Calc next PAC value & write it to PAC
Execute NOP
Is TICx flag set?
Y
N
Is "Old PAC" = PAC?
Y
N
Jmp to PACOV ISR
Y
Is PAC = $00?
N

HC12_AR_650: CGM
XIRQ during last cycle of STOP instruction causes run away

Description

If an XIRQ interrupt occurs during the execution of the STOP instruction with the control bit DLY=0 (located in the INTCR register), the CPU may not run the software code as designed.

Workaround

1. Set the delay control bit DLY=1 so that a delay will be imposed prior to coming out of STOP. The user must also implement the workaround for AR_562 (Exiting STOP with DLY=1) as well.
2. If using XIRQ with a stable external clock and DLY=0, contact Freescale Applications Department for a detailed workaround.

HC12_AR_662: CPU
XIRQ interrupt can prevent the CPU from generating the vector request signal for the IRQ

Description

If all of the following conditions are met, the XIRQ asynchronous path can prevent the CPU from generating the vector request signal for the IRQ:

- Using an MCU in the HC12 Family (not the HCS12 Family)
- Using XIRQs (X-bit is cleared in the CCR by software)
- Asserting an XIRQ interrupt (through the XIRQ pin)
- XIRQ interrupt occurs at the start of an IRQ interrupt exception processing

Because XIRQs interrupt IRQs, the XIRQ stack will follow the IRQ stack. The lack of the IRQ vector request signal will cause the XIRQ stack to have an invalid return address. As soon as the XIRQ finishes executing the XIRQ interrupt service routine, the XIRQ RTI (return from interrupt) causes the CPU to use that invalid return address, leading to code runaway.

The potential failure window is only a few nanoseconds and varies with process, temperature, design, etc.

Workaround

There are two identified workarounds: one hardware and one software.

Hardware

Because the failure window is small and occurs near the T1 cycle, the external XIRQ signal could be gated to the rising edge of ECLK.

Software

Because the XIRQ interrupt service routine (ISR) still executes correctly, code can be added to the XIRQ ISR to determine whether the error may have occurred and use software to work around the situation. Because the problem only occurs if the XIRQ interrupts an IRQ before any ISR instructions are executed, the CCR in the XIRQ stack could be checked to determine whether the I-bit was set and two stack frames were created (first one for the IRQ and second one for the XIRQ). Further checks can then be done to determine whether the two stacks are identical except for the return address. If they are, use the IRQ stack as the return address for the XIRQ.

Here is an example of that code:

```
ALL_ISRs:
              pshy                 ;First instruction of the ALL ISRs need to push something
              inx                  ; (Y for example) onto stack to separate the stack frames
                                   ; to help to determine if any instructions from the ISR were
                                   ; run.  That will determine if the workaround need to be
                                   ; done when in the XIRQ ISR.
                     ;Note: this must be done in all ISRs, also adding the inx makes it less likely
                     ; you would falsely think you fell into the erratum. Increment what you tend
                     ; to not use in ISRs first, you could also increment D and Y for even further
                     ; security that the software does not falsely think it fell into the erratum.

                     ;Normal user ISR code here [except no RTI (yet)].

              leas 2,SP     ; return SP to adjust for the pshy

              rti    ; normal user rti

XIRQ_ISR:
                                   ;normal user ISR code here [except no RTI (yet)].

              brset 0,SP,#$10,Check;If CCR had I-bit set in the stack, this is the first part
                                   ;of the workaround to determine if the XIRQ interrupted
                                   ;an IRQ or a section of code that had the I bit set
                                   ;If not just return since no problem.
Okrti:
              rti                  ;Normal user code (unstack registers, etc.)


Check:                      ;The I bit was set in the XIRQ stack so we need
                                   ; to further check and see if we should do the
                                   ; workaround.  Need to check to see if there are two
                                   ; nearly identical stack frames with the exception of the
                                   ; I-bit and the return address. If so adjust the Stack Pointer
                                   ; to point to the ISR stack frame before the XIRQ RTI.
                                   ;Note, non interruptible code that gets an XIRQ
                                   ; will also go here.  If X or Y was not used in the XIRQ
                                   ; ISR then this code could be reduced in size, (by removing
                                   ; the appropriate loads from below)

              ldd 1,SP             ;Load ACCD from XIRQ stack frame ACCB:ACCA value
                                   ;Note the values of A and B are interchanged from a
                                   ;normal pull for easier checking.
              cpd 10,SP     ;Compare ACCD from suspect IRQ stack frame with XIRQ
                                   ; stack. Note the values of A and B are still interchanged.
              bne Okrti     ;Not the same, so not in erratum, just return (RTI).
              ldx 3,SP             ;Load X with XIRQ stack frame X value.
              cpx 12,SP     ;Compare X from suspect IRQ stack frame with XIRQ stack.
              bne Okrti     ;Not the same, so not in erratum, just return (RTI).
              ldy 5,SP             ;Load Y with XIRQ stack frame Y value.
              cpy 14,SP     ;Compare Y from suspect IRQ stack frame with XIRQ stack.
```

```
            bne Okrti     ;Not the same, so not in erratum, just return (RTI).

                                 ;Next we check the CCR to see if they are the same except
                                 ; for the I bit which should be different.
            ldaa 0,SP     ;Load the CCR from the XIRQ stack into ACCA
            eora 9,SP     ;Exclusive OR XIRQ CCR with suspect IRQ CCR
            anda #$EF     ;AND with the I bit mask to not check the I bit.
            bne Okrti     ;Not the same, so not in erratum, just return (RTI).
                                 ;if all checks the same could be in the erratum
                                 ;could check return address from IRQ ISR as a further check
                                 ;to make sure it is a normal ISR program space
                                 ;could also check to make sure room for SP to back up
            leas 9,SP     ;add 9 to SP (to point to IRQ stack frame)

      rti                        ;Return using IRQ stack (unstack registers, etc)
```

As with most code workarounds, there are a few situations where there still may be an issue. For example, if you pushed information on to the stack that exactly matched the stack frame before you did the XIRQ and that XIRQ occurred while the I bit was set, the software could falsely think it was the ISR stack. This is a very rare situation.

This page is intentionally blank.

This page is intentionally blank.

How to Reach Us:

USA/Europe/Locations not listed:
Freescale Semiconductor Literature Distribution
P.O. Box 5405, Denver, Colorado 80217
1-800-521-6274 or 480-768-2130

Japan:
Freescale Semiconductor Japan Ltd.
SPS, Technical Information Center
3-20-1, Minami-Azabu
Minato-ku
Tokyo 106-8573, Japan
81-3-3440-3569

Asia/Pacific:
Freescale Semiconductor H.K. Ltd.
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
852-26668334

Learn More:
For more information about Freescale Semiconductor products, please visit
http://www.freescale.com

MSE812A4_0K51E
Rev. 1, 10/2004