Application Note

AN2617
Rev. 0, 2/2004

A Software Interrupt Priority Scheme for HCS12 Microcontrollers

MOTOROLA intelligence everywhere™ | digital dna™

By Grant M More and Daniel Malik
Applications Engineering,
Motorola, East Kilbride

Introduction

The HCS12 series of microcontrollers provides an abundance of interrupt sources and discrete vectors to allow handling of each of the interrupts. However, by default, nested interrupts are not permitted. Although it is possible for nested interrupts to be enabled, any new interrupt request will interrupt a presently active service routine. This is undesirable, as any interrupt service routines that are designated critical can be interrupted by any other interrupt service request. If this is unacceptable, it is necessary for interrupts to be prioritized. This is not supported in hardware on the HCS12 series (it is supported on the HCS12X), but can be easily implemented in software.

This application note describes a priority scheme that can be used to assign priorities to interrupt sources, and to schedule the interrupts based on the assigned priority. This includes the provision for an interrupt of a higher priority halting interrupts of a lower priority, assuming core processing time and returning control to the lower priority interrupt after the higher priority service routine has been completed.

Overview of HCS12 Hardware Interrupt Handling Capability

This section summarizes the hardware interrupt handling capabilities of the HCS12 microcontroller. A degree of background knowledge of the hardware scheme is necessary to understand the implementation of the software scheme. Full details of exception processing can be found in the appropriate core user guide documentation.

Interrupt Sources

Interrupts are handled by the microcontroller, according to the definition of the source. Exceptions are defined as resets, software interrupts, or interrupt requests. The XIRQ non maskable interrupt is a special case and is defined separately.

Reset Interrupts

The resets are defined as: (1) the reset vector; (2) the computer operating properly (COP) reset vector; and (3) the clock monitor (CM) reset vector. These are non maskable and will always be processed when requested.

Software Interrupts

The software interrupts are: (1) the SWI vector, requested when either a software interrupt instruction is executed or a BDM vector request is asserted (for example at a breakpoint); and (2) the unimplemented opcode trap vector, requested when a trap instruction is executed. Again, these requests are non maskable.

XIRQ Non Maskable Interrupt

The XIRQ is a single special case interrupt. This vector is maskable only with the X-bit in the condition code register (CCR). Full details can be found below.

Interrupt Requests

All other interrupts fall into standard interrupt request category. These interrupts are maskable with the I-bit in the CCR.

XIRQ – Non Maskable Interrupt

The XIRQ input is a special non maskable interrupt which is typically used to deal with major system failures, such as power failure. For this reason, it is assigned high priority and, once enabled by clearing the X-bit in the core condition code register, is capable of interrupting all other interrupt service routines on a hardware level and can only be disabled by a system reset. All interrupt sources are automatically disabled during the servicing of an XIRQ request. For this reason, if enabled, the XIRQ interrupt must be considered as an interrupt of the highest priority in any software based priority scheme — above the priority of the highest software assigned priority.

IRQ – Maskable Interrupts

Most interrupts on the HCS12 microcontroller fall into the maskable category. These interrupts are masked by the I-bit in the CCR and, by default, are not nested. The I-bit is automatically set during exception processing, when entering a service routine. This prevents any maskable service routine being interrupted by any other maskable routine. This is the default configuration. The I-bit can be manually cleared upon entry to the service routine, but then any maskable routine can interrupt, and no consideration is given to priority.

Having said that, maskable interrupts are configured on a priority launch scheme. If two or more interrupt requests are received concurrently, the request with the highest address (closest to $FFFF) is serviced first. For example, if the IIC and IRQ interrupts are received simultaneously, the IRQ interrupt will be serviced first. However, the IRQ interrupt will not be able to interrupt the IIC interrupt if the IIC interrupt service routine is already in progress when the IRQ interrupt is requested, unless the I-bit is cleared, and then any interrupt can interrupt any other interrupt.

HPRIO – Highest Priority I Interrupt Register

This register allows one interrupt request to be given priority over any other interrupt request. In practical terms, this means that the interrupt identified by this register will be serviced in preference to any other requests that are active simultaneously. However, this will not allow the identified interrupt to interrupt any other executing interrupt service routine. In other words, the high priority interrupt only allows prioritization of the interrupt request, not of the interrupt service routine.

Interrupt Scheduling

Default Strategy

As discussed, the default configuration of interrupts on HCS12 devices does not support interrupt prioritization or nesting. Interrupts of a higher priority cannot interrupt interrupts of a lower priority. The execution of the low priority interrupt must complete before the processing of the high priority interrupt can begin. This is shown in **Figure 1**.

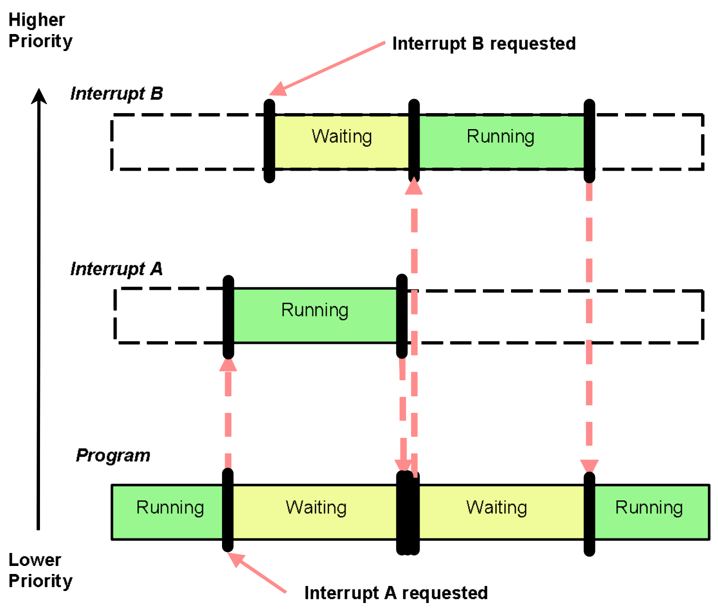

Figure 1. Default HCS12 Interrupt Processing Strategy

In this example, the requirement for a prioritization scheme is clearly shown. The high priority interrupt, interrupt B, is delayed by low priority interrupt A before being executed.

Allowing Maskable Interrupts

The problem with simply enabling all maskable interrupts by clearing the I-bit is demonstrated in **Figure 2**. As can be seen, the high priority interrupt is interrupted by a low priority interrupt and again made to wait. The priority here is governed by the timing of the interrupt requests rather than the priority. In most applications this is an undesirable strategy.

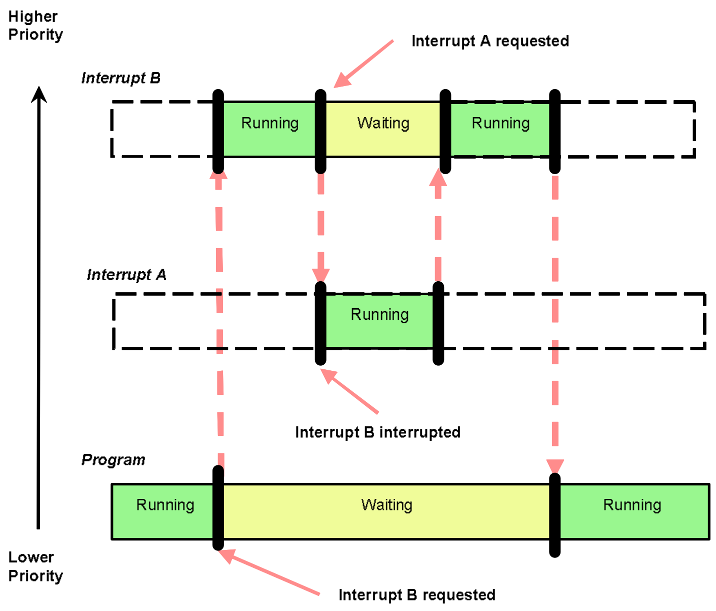

Figure 2. Effect of Enabling Maskable Interrupts in Maskable Interrupt Service Routines

In order to prevent interrupts of a lower priority interrupting those of a higher priority, it is necessary for the low priority interrupts to be selectively disabled, based on their assigned priority.

The solution to this problem is to clear the maskable interrupt I-bit at the beginning of any interrupt service routine, allowing all interrupts, and to implement a selective interrupt enable scheme to only allow interrupts of a higher priority than the priority of the current interrupt. This is the idea behind the software interrupt priority scheme. An example of this strategy is shown in **Figure 3**.

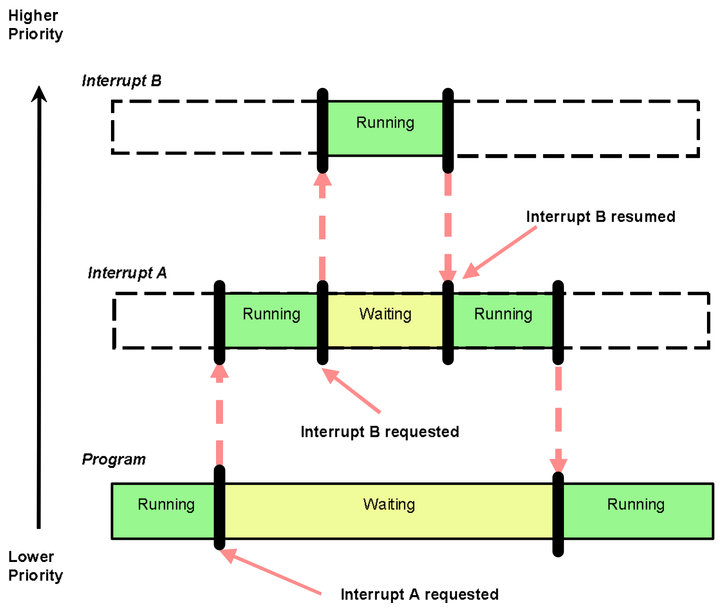

Figure 3. Effect of Interrupt Priority Scheme Implementation

Setting the Priority Level

Two-level Priority Scheme

A simple interrupt priority scheme can be implemented very easily. The suitability of this approach will depend on the demands of the application. In this scheme, one interrupt can be assigned low priority; one or more can be assigned high priority. The I-bit is cleared in the low priority interrupt, allowing the high priority interrupt to interrupt it. (Note that the low priority interrupt can also interrupt itself – handling provision should be made if this may occur.) However, the I-bit is not cleared in the high priority interrupt service routine, preventing the low priority interrupt from interrupting.

Multi-level Priority Scheme

The implementation of a multi-level priority scheme takes a little more care, as more interrupts are inevitably used across an increased number of priority levels. A key feature of the HCS12 is that all maskable interrupts have local enable bits as well as the global mask bit in the CCR. The local enable bits are fundamental to assigning the interrupts to priority levels as discussed below. An example of the listing of the interrupt vectors and associated enable bits, as can be found in the device user guide, is shown in Table 1.

Table 1. Example of Local Enable Bit Listing

Vector Address	Interrupt Source	CCR Mask	Local Enable	HPRIO Value to Elevate
$FFEE, $FFEF	Enhanced Capture Timer Channel 0	I-bit	TIE (C0I)	$EE
$FFEC, $FFED	Enhanced Capture Timer Channel 1	I-bit	TIE (C1I)	$EC
$FFEA, $FFEB	Enhanced Capture Timer Channel 2	I-bit	TIE (C2I)	$EA

The global priority level is set and varied, by setting and clearing the local mask bits, as appropriate. For example, consider a priority scheme that requires three interrupt priority levels (and one for the non-interrupted standard program flow), and that has three interrupts, one on each priority level, as shown in Figure 4.

Priority Level	Interrupt	Local Mask Bit
Zero (main flow)	None	N/A
One	Timer Channel 0	TIE C0I
Two	Timer Channel 1	TIE C1I
Three	Timer Channel 2	TIE C2I

Figure 4. Simple Priority Scheme Example

In order to move to an interrupt priority level, it is necessary to clear the mask bits relating to all the interrupts on the current priority level and below. This ensures that only interrupts of a higher priority than the current interrupt will be recognized. All interrupts that occur on any lower level will be ignored until the global priority level is lowered to the level below the assigned interrupt priority. For example, consider the above scheme when a timer channel two interrupt occurs. The global priority is set to level two by masking priority levels one and two using the C0I and C1I bits. Practically, this will only allow interrupts on level three to interrupt the timer channel 2 service routine.

Defining and Using Interrupt Priority Levels

Global Priority Level

It is necessary to define a global priority level to determine which interrupts are enabled and disabled. The global priority level should be a global variable that stores the value of the current interrupt priority. Only interrupts of a higher priority can interrupt the current interrupt (or program flow). The priority level is a global concern, and should be set at the beginning of an interrupt service routine, before the I-bit interrupt mask is cleared. The initial priority level must be stored (before any priority level manipulation takes place) at the very beginning of any service routine and re-established at the end, to ensure that the previous priority level is resumed upon completion of the interrupt service routine.

Choosing Priority Levels

The number of priority levels in a system should be chosen based on a number of factors, most of which are outwith the scope of this application note (for example, prioritization of service routines based on timing sensitivity of interrupt tasks). However, as a guide, the number of interrupt levels will never exceed the number of maskable interrupts plus one, as it is necessary to assign a priority level to any code that is executed outwith the interrupt service routines. (Do not forget that a higher priority level is also implemented if the XIRQ is enabled.) Secondly, for minimum latency and service time overhead, interrupts should be organized into the smallest number of priority levels possible. The reasons for this are described below.

Assigning Priority Levels to Interrupts

Every enabled interrupt should be assigned to a priority level based on the various demands of the application. This will allow the relevant mask bits to be set and cleared when moving between priority levels.

Limitations and Timing Overhead

The implementation of this scheme involves an additional degree of latency in the processing of interrupts that are prioritized.

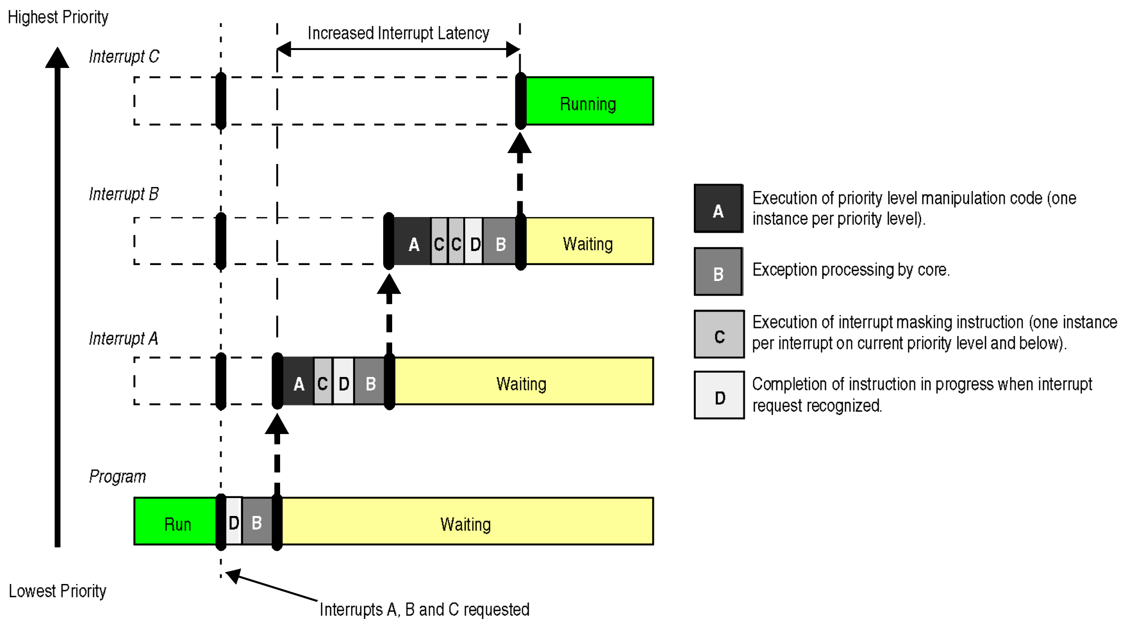
Highest Priority
Lowest Priority
Increased Interrupt Latency
Interrupt C
Running
Interrupt B
A C C D B
Waiting
Interrupt A
A C D B
Waiting
Program
Run
D B
Waiting
Interrupts A, B and C requested
A Execution of priority level manipulation code (one instance per priority level).
B Exception processing by core.
C Execution of interrupt masking instruction (one instance per interrupt on current priority level and below).
D Completion of instruction in progress when interrupt request recognized.

Figure 5. Timing Overhead Example

Consider the example shown in **Figure 5**. This instance demonstrates what happens when three prioritized interrupts occur simultaneously. Interrupt A has a higher interrupt hardware priority than interrupt B, and interrupt B has a higher hardware priority that interrupt C. For this reason, interrupt A will be serviced by the hardware first, followed by interrupt B and then finally interrupt C. This will provide a worst case latency example, as the ISR of functions A and B must be entered, and the software priority level raised, before service routine C can be launched.

Fixed Overhead = Time For Exception Processing + Time To Process Priority Manipulation Code

Worst Case Overall Increased Interrupt Latency (in this example) = Time Spent In Interrupt A + Time Spent In Interrupt B

Time Spent In Interrupt A = Time to Finish Current Instruction + Fixed Overhead + (Number of Interrupts at Level 1 * Time Taken to Disable Each Interrupt)

Time Spent In Interrupt B = Time to Finish Current Instruction + Fixed Overhead + (Number of Interrupts at Level 1 and Level 2 * Time Taken to Disable Each Interrupt)

In the example code, the fixed overhead is 20 cycles. The time to disable each interrupt is 4 cycles. The worst case time to finish an instruction will be 13 cycles (for an EMACS instruction). Assuming there is only one interrupt on each of levels A and B,

Time Spent in Interrupt A (worst case) = 13 + 20 + (1 * 4) = 37 cycles

Time Spent in Interrupt B (worst case) = 13 + 20 + (2 * 4) = 41 cycles

Worst Cause Overall Increased Interrupt Latency = 37 + 41 = 78 cycles

This is obviously a worst case value, assuming EMACS (extended multiply and accumulate) instructions are being executed in both interrupted service routines when interruption occurs. If a typical example is considered where four cycle BSET (bit-set) instructions are executed, the increased latency is reduced to 60 cycles. This is still worst case in terms of interrupt hardware priority.

Library Macro Files and Software Example

A set of library files have been prepared to allow this interrupt priority scheme to be easily incorporated into any application. The function of these library files is demonstrated in the software written to accompany this application note. The library files utilize macros that set the relevant bit masks based on the priority level. The software is configured as a macro to provide the most efficient implementation of the priority scheme. Please refer to the ‘read me’ file included with the software for further information about the function and configuration of the library files. The files can be simply copied out of the example software and used in any application requiring interrupt priority functionality.

The example software can be built and loaded directly into a 9S12Dx256 device (or any other HCS12 device, with some minor modifications). The software implements a three-level timer interrupt prioritization example, as described in **Figure 4**. Each ISR contains identical code to set the associated port B line high, wait an identical time, and then set the port B line low. The example contains two demonstrations that can be switched between, by commenting and uncommenting code within the software project. Full details of the functionality of the software example can be found in the comments within the code. Brief descriptions of the functionality and expected results are given below.

Simultaneous Occurrence of Interrupts

The timer interrupts are configured to occur simultaneously by default in the example software. The interrupt processing priority is then arbitrated by the priority scheme software, and the highest priority interrupt is processed first.

Figure 6 shows the priority interpretation of this situation, using the priority decoding scheme.

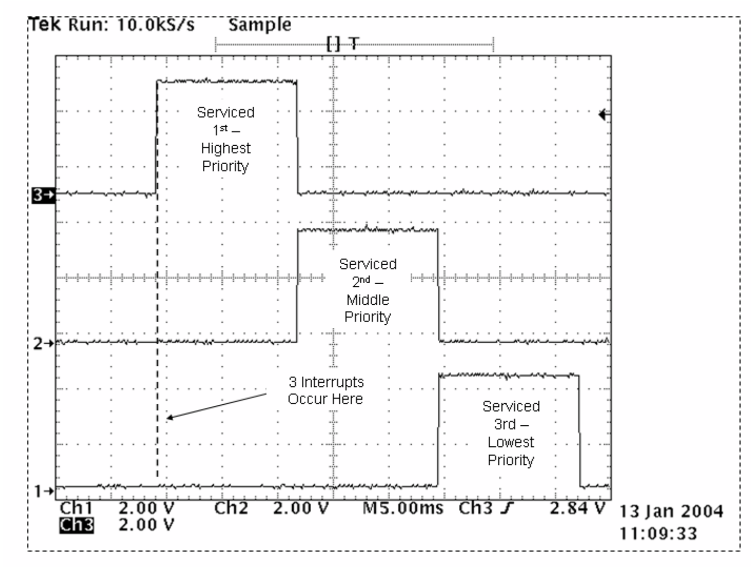

Figure 6. Interrupt Priority Execution — Simultaneous Occurrence of Interrupts

NOTE: *The in-built hardware priority of the interrupts used in **Figure 6** is the opposite of the priority specified in this example. The software priority control routines must arbitrate the priority levels in order to execute the ISRs in the correct order, as specified by the software interrupt priority scheme.*

Sequential Occurrence of Interrupts

The timer interrupts can be configured to occur in sequence by re-enabling the commented code in the setup routine. The interrupts are then triggered during the execution of the lower priority ISR (i.e. lowest priority first, middle priority next, highest priority last). Figure 7 shows the priority interpretation of this situation, using the priority decoding scheme.

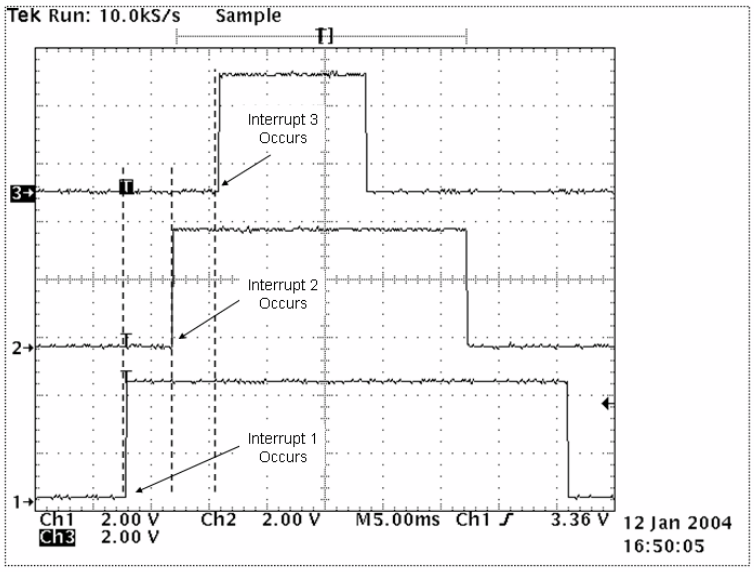

Figure 7. Interrupt Priority Execution — Consecutive Occurrence of Interrupts

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:
Motorola Literature Distribution
P.O. Box 5405, Denver, Colorado 80217
1-800-521-6274 or 480-768-2130

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

ASIA/PACIFIC:
Motorola Semiconductors H.K. Ltd.
Silicon Harbour Centre
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
852-26668334

HOME PAGE:
http://motorola.com/semiconductors

MOTOROLA