Using RS08 Microcontrollers with Variable Interrupt Priorities

Program Association of RS08 Microcontrollers

by: Kenny Ji
Asia and Pacific Operation Microcontroller Division

1 Introduction

Freescale Semiconductor’s RS08 family of microcontrollers (MCUs) uses a reduced version of the HCS08 central processor unit (CPU). The RS08 MCUs are targeted for small embedded applications.

The interrupt mechanism in RS08 MCUs does not interrupt the normal flow of instructions; instead, it wakes up the RS08 MCUs from wait and stop modes. In run mode, interrupt events must be polled by the CPU.

This application note describes a shortcut for using variable priority interrupts in RS08 MCUs.
Typical RS08 MCU Schedules

2 Typical RS08 MCU Schedules

Because an RS08 does not implement an HCS08 style interrupt handler, interrupts must be managed in software by checking if interrupts are present. A polling mode must be used in an RS08 MCUs’ core. In run mode, the main loop checks the system interrupt pending (SIP1) register. When an interrupt is detected, the corresponding bit in SIP1 is set, by which the scheduler can dispatch its interrupt service routine (ISR). In wait and stop mode, the main loop is activated when an enabled interrupt is detected. Immediately after wake up, the main loop checks the SIP1 register for the interrupt detected and then dispatches its ISR.

As shown in Figure 1, point A shows that an interrupt is detected in run mode. Then, the corresponding ISR is launched. After executing the ISR at B, the program returns to the beginning of main loop at point C. When the scheduler has completed all tasks, it enters wait or stop mode for saving power at point D. When an interrupt is activated in wait or stop mode, the scheduler wakes up. After the interrupt source in SIP1 is checked, the ISR is dispatched at point E. After the ISR executes at point F, the program returns to the beginning of the main loop at point G.

After ISR execution, the program returns to the beginning of the main loop because the ISR being executed must be the highest priority one among all the detected interrupts. In this case, the high priority interrupt is detected prior to low priority interrupt. Therefore, the rule of RS08 scheduler is:

- Sort all available interrupts in reverse. The interrupt with the highest priority is detected first.
- Add a branch instruction at the end of interrupt service routine. The branch brings the program to the beginning of the scheduler.

Example 1 shows a typical schedule code in RS08 MCUs. There are five interrupts under software control. When an interrupt is detected in the main loop idle, the schedule executes the corresponding interrupt service routine (ISR). The polling order is in MTIM, ACMP, KBI, RTI, and LVD. Any ISR associated with low priority must be executed after high priority ISRs are completed. For example, RTI ISR is executed after MTIM, ACMP, and KBI ISRs are completed. If an ACMP interrupt occurs right after KBI ISR is
completed, the program continues executing ACMP ISR instead of RTI ISR. The RTI interrupt is pended until the ACMP ISR returns and no other higher priority interrupt occurs.

Example 1. RS08 Typical Schedule

```assembly
_Startup:
   MOV #HIGH_6_13(SIP1), PAGESEL

; Polled interrupt code
idle: wait
    BRSET SIP1_MTIM, MAP_ADDR_6(SIP1), mtim_ISR
    BRSET SIP1_ACMP, MAP_ADDR_6(SIP1), acmp_ISR
    BRSET SIP1_KBI, MAP_ADDR_6(SIP1), kbi_ISR
    BRSET SIP1_RTI, MAP_ADDR_6(SIP1), rti_ISR
    BRSET SIP1_LVD, MAP_ADDR_6(SIP1), lvd_ISR

; feed watchdog
    feed_watchdog

; check other interrupts
    BRA idle

mtim_ISR: <code>
    BRA idle

acmp_ISR: <code>
    BRA idle

kbi_ISR: <code>
    BRA idle

rti_ISR: <code>
    BRA idle

lvd_ISR: <code>
    BRA idle
```

3 Restrictions on Traditional RS08 Schedulers

After the pending is sorted, you cannot change the polling order. In some applications, you may want to change the order of polling; the traditional RS08 scheduler can restrict the user program. In Example 1, if polling to KBI interrupts occurs before that of RTI, RTI ISR cannot be processed before the KBI ISR is completed.

4 Making Interrupt Priorities Variable

In this section, an advanced scheduler is introduced to deal with the restriction described in Section 3, “Restrictions on Traditional RS08 Schedulers”. You can schedule the program easily with a simple scheduler and a set of macros.
4.1 Data Structure

The simple scheduler uses a small data array as a priority table in RAM. There are three bytes in each cell. The first byte is used for interrupt masks. The latter two are used to store ISR high and low addresses, respectively. The high priority ISRs are allocated in low address, while the low ones are allocated in high address.

![Priority Table](image)

Figure 2 shows that the mask is the bit that indicates the pending interrupt in SIP1 register. In 9RS08KA2, the MTIM pending interrupt indicator is the bit 2 in SIP1. Then, 0x04 must be put in the mask byte before entering normal schedule. The high ISR stands for the high 6-bit address of the corresponding ISR, while the low ISR stands for the low 8-bit address. After a certain interrupt is detected, the schedule executes the associated ISR, whose address is fetched from these two bytes.

The number of priorities used is named MaxISR, which is also the maximum index of priority table.

4.2 Scheduling Operation

Initialize the priority table before entering normal operation. Immediately after initialization, the scheduler enters a loop in which it executes every ISR whose interrupt is detected. This behavior is similar to that of a traditional RS08 scheduler. The traditional scheduler uses fixed instructions to detect the interrupt and jumps to a fixed address in opcode. However, the scheduler detects the interrupt with mask and executes its ISR with a routine address stored in RAM. This significant difference makes the scheduler flexible enough to meet a variable environment. You can change the priority by changing masks and addresses stored in priority table in RAM. Figure 3 shows a typical work flow of the scheduler.
4.3 Changing Interrupts Priorities in Program

As mentioned in the previous section, the scheduler uses RAM instead of fixed instructions. You can change interrupt priorities in the program. A macro, INT_PLUG, simplifies the code, so you can change the interrupt priorities conveniently. Table 1 shows the definition of INT_PLUG:

| Int_Plug    MACRO                                                                 |
|-------------|-----------------------------------------------------------------------------|
| MOV #\1,PriorityTable + 3 * \3 + 0                                           |
| MOV #HIGH(\2), PriorityTable + 3 * \3 + 1                                   |
| MOV #LOW(\2), PriorityTable + 3 * \3 + 2                                   |
| ENDM                                                   |

There are three parameters in this macro. The first indicates the interrupt mask used in SIP1. The second stands for the 14-bit address of the ISR. The third shows the new priority of this interrupt. The new priority must be between 0 to MaxISR - 1.

For example, LVD and RTI’s ISR are the first and second priority interrupts as shown below.

**Example 2. Initialize Interrupt Priority**

```assembly
Int_Plug mSIP1_LVD, lvd_ISR, 0 ; plug the 1st interrupt
Int_Plug mSIP1_RTI, rti_ISR, 1 ; plug the 2nd interrupt
```

You can write the following code to swap their priorities:

**Example 3. Change Interrupt Priority**

```assembly
Int_Plug mSIP1_LVD, lvd_ISR, 1 ; plug the 1st interrupt
Int_Plug mSIP1_RTI, rti_ISR, 0 ; plug the 2nd interrupt
```
Then, the RTI interrupt occurs prior to the LVD from the next schedule.

**NOTE**

Whatever priorities are set, make sure the write operation does not exceed the priority table. Otherwise, there might be a fatal error leading to reset.

## Appendix A
### The Scheduler Code

The code is implemented in 9RS08KA2 with five interrupts available.

### Table 2. The List of Scheduler Code

```assembly
;*******************************************************************
;* This stationary serves as the framework for a user application. *
;* For a more comprehensive program that demonstrates the more advanced functionality of this processor, please see the demonstration applications, located in the examples directory.
;*******************************************************************

; Include derivative-specific definitions
INCLUDE 'derivative.inc'

; export symbols

XDEF _Startup
ABSENTRY _Startup

;*******************************************************************
;* The macros defined below are used by the scheduler
;*******************************************************************

; MaxISR indicating the maximum number of interrupts or ISRs
MaxISR     equ 5

; Int_Plug plugs a special interrupt to a certain interrupt priority
Int_Plug     MACRO
    MOV #\1,PriorityTable + 3 * \3 + 0
    MOV #HIGH(\2), PriorityTable + 3 * \3 + 1
    MOV #LOW(\2), PriorityTable + 3 * \3 + 2
ENDM

; variable/data section

ORG    RAMStart         ; Insert your data definition here
ExampleVar: DS.B   1

;*******************************************************************
; Priority Table includes interrupt priority and subroutine address
; every 3 bytes includes one set of information, the format shows below
;*******************************************************************
```

Using RS08 Microcontrollers with Variable Interrupt Priorities, Rev. 0

Freescale Semiconductor
; byte #0 : priority
; byte #1 : high 6-bit address of ISR
; byte #2 : low 8-bit address of ISR
;
;**************************

PriorityTable:
  DS  3 * MaxISR

; code section
;
  ORG  ROMStart

; Variable Priority Interrupt System

_Startup:
  MOV #HIGH_6_13(SIP1), PAGESEL

; Interrupt Priority Initialization

PriorityTable_Init:
  LDX #PriorityTable
  Int_Plug mSIP1_LVD, lvd_ISR, 0 ; plug the 1st interrupt
  Int_Plug mSIP1_RTI, rti_ISR, 1 ; plug the 2nd interrupt
  Int_Plug mSIP1_KBI, kbi_ISR, 2 ; plug the 3rd interrupt
  Int_Plug mSIP1_ACMP,acmp_ISR, 3 ; plug the 4th interrupt
  Int_Plug mSIP1_MTIM,mtim_ISR, 4 ; plug the 5th interrupt

mainLoop:
  LDA   MAP_ADDR_6(SIP1)  ; SIP1 -> A
  BRNE  Dispatch          ; goto dispatch
  LDA   #3                ; 3 -> A , 2 cycle
  TAX                     ; X + 3 -> X , 2 cycle
  BRA   Next              ; next loop
  BRA   mainLoop          ; it can not be reached normally
Dispatch:

INCX ; X++
LDA ,X ; High address -> A
SHA ; High address -> SPCs
INCX ; X++
LDA ,X ; Low address -> A
SLA ; Low address -> SPC
RTS ; goto subroutine
BRA mainLoop ; it can not be reached normally

mtim_ISR:
BRA mainLoop

acmp_ISR:
BRA mainLoop

kbi_ISR:
BRA mainLoop

rti_ISR:
BRA mainLoop

lvd_ISR:
BRA mainLoop

;*************************************************************************
;*                          Startup Vector                            *
;*************************************************************************
ORG 0x3FFD
JMP _Startup ; Reset
Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized use, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-compliant and/or non-Pb-free counterparts. For further information, see http://www.freescale.com or contact your Freescale sales representative.

For information on Freescale’s Environmental Products program, go to http://www.freescale.com/epp.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. © Freescale Semiconductor, Inc. 2007. All rights reserved.