ECET 310-001
Chapter 6

Interrupts and Resets
w/intro to Ports (Chapter 7)

W. Barnes, 9/2006

Ref. Huang, Han-Way, *The HCS12/9S12: An Introduction to Software and Hardware Interfacing,* Thomson/Delmar.

Interrupts and Resets

Often referred to as Exceptions:

- Unscheduled, asynchronous, usually high priority, events that interrupt the normal flow of a program

Basics of Interrupts

- What is an interrupt?
  - A special event (initiated by SW or HDW) that causes the CPU to stop normal program execution and perform a service related to the event.
  - Examples of interrupts: I/O completion, timer time-out, illegal opcodes, arithmetic overflow, divide-by-0, swi, etc.

- Functions of Interrupts
  - Coordinating I/O activities and avoiding the use of polling
  - Providing a graceful way to exit from errors
  - Reminding the CPU to perform routine tasks

- Interrupt maskability
  - Maskable
    - Interrupts that can be ignored by the CPU
    - Enabled by setting an enable flag (global to disable all and local for specific interrupts)
  - Nonmaskable interrupts can't be disabled by software instructions.

Basics of Interrupts Continued

Interrupt priority

– Resolves the order of service for multiple pending interrupts

A complete interrupt service cycle includes, in order:

1. Complete the current instruction
2. Disable the global interrupt flag (set I to prevent other maskable interrupts)
3. Save the address of next instruction in the stack
4. Save the CPU status [registers X(1st),Y,B,A,CCR(last)] in the stack
5. PC← ISR address (found in the corresponding interrupt vector)
6. Execute the interrupt service routine up to RTI
7. RTI retrieves the CPU status, returns address from the stack, clears I mask enabling other maskable interrupts
8. Resume the interrupted code

NOTE: Overhead of execution of the routine plus the storing and retrieving from the stack is about 20 E cycles or 833 ns for a 24 MHz clock.

Basics of Interrupts Continued

- Interrupt vector
  - Starting address of the interrupt service routine

- Interrupt vector table
  - Stored in upper 128 bytes of 64k
  - Six highest addresses used for resets and non-maskable vectors

- Methods of determining interrupt vectors
  - Predefined locations (Microchip PIC18, 8051 variants)
  - Fetching the vector from a predefined memory location (HCS12)
  - Executing an interrupt acknowledge cycle to fetch a vector number in order to locate the interrupt vector (68000 and x86 families)

- Basic parts of interrupt programming
  1. Interrupt Service Routine (ISR)- what to do when interrupt occurs
  2. Interrupt vector table (tells computer location of ISR)
  3. Enabling the interrupt with flag(s)

Partial Interrupt Vector Table

Vector Address (2 byte)	Interrupt Source	CCR Mask	Local Enable	HPRIO value to elevate to highest I bit
$FFFE	Reset	Non-maskable	None	----
$FFFC	Clock monitor reset	Non-maskable	COPCTL(CME,FCME)	----
$FFFA	COP failure reset	Non-maskable	COP rate selected	----
$FFF8	Unimplemented Opcode trap	Non-maskable	None	----
$FFF6	SWI	Non-maskable	None	----
$FFF4	/XIRQ	X bit	None	----
$FFF2	/IRQ	I bit (global enable)	INTCR (IRQEN)	$F2
$FFF0	Real time interrupt	I bit (global enable)	RTICTL(RTIE)	$F0

Reset (nonmaskable)

- Includes the power-on reset, reset pin manual reset, the COP reset (Computer Operate Properly), and clock monitor reset

- Some CPU registers, flip-flops, and the control registers in I/O interface chips must be initialized in order for the computer to function properly. Reset will do this.

More Interrupt Details

- Maskable interrupts, including IRQ pin
  - Different HCS12 members (versions) implement different number and types of peripheral functions, and hence may have different number of maskable interrupts.
  - One of the maskable interrupts can be raised to the highest priority among the maskable interrupt group and receive quicker service by programming the HPRIO register shown below.

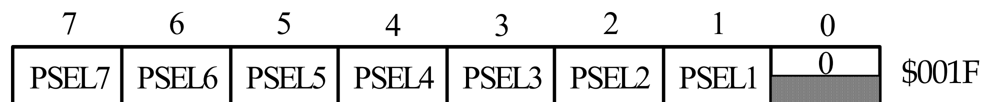

Figure 6.1 Highest priority I interrupt register

IRQ Pin Interrupt

- The only external maskable interrupt for the HCS12. (Recall XIRQ and Reset are non-maskable)
- Programmable as edge-triggered or level-triggered.
  - /IRQE= 0, responds to low level
  - /IRQE= 1, responds to falling edge
- Has a local enable mask in the IRQ Control Register (IRQCR)
  - /IRQEN = 1, enabled
  - /IRQEN = 0, disabled (masked)
- Contents of the IRQCR register are (On Reset 0100 0000 is stored):

IRQE	**IRQEN**	**0**	**0**	**0**	**0**	**0**	**0**

Edge vs. Level Sensitive

Making IRQ Level-sensitive

- Pros
  - Multiple interrupt sources can be tied to this pin.
- Cons
  - Need to make sure that the IRQ signal has become inactive before the IRQ service routine is complete if there is only one interrupt request pending.

Making IRQ Edge-sensitive

- Pros:
  - No need to control the duration of the IRQ pulse.
- Cons:
  - Not suitable for noisy environment because every falling edge caused by noise will be recognized as an interrupt.

IRQ Example (based on Ex. 6-1 in text)

- Problem Statement: Develop an ISR which will respond to a falling edge on the /IRQ pin by incrementing and displaying a count on the LEDs.
- Steps for the solution:
  - Set up output port
  - Enable (locally and globally) interrupt and select triggering
  - Output message to terminal
  - Run endless loop while awaiting interrupt
  - ISR (entered on negative edge at IRQ)
    - Increment the count
    - Send Count to LEDs
    - Return to main
- Steps for testing:
  - Connect square wave generator to /IRQ (shares pin with PE1)
  - Run program and observe LEDs, try varying the frequency

Three Approaches to the IRQ Example

1. **Short Version, webex_irq0.asm**
   - Uses absolute addresses of Ports, DDRs and Registers
   - Code brief but not flexible and reusable

2. **Long Version, webex_irq1.asm**
   - Assigns clear names to Ports, DDRs and Registers
   - Longer but names can now be used throughout program
   - Also works with individual bits in Ports J and P

3. **Shorter by using header file, webex_irq2.asm**
   - Imports header file, hcs12.inc, to pre-define ports, etc.
   - Convenient but must be careful to use correct names
   - Also works with individual bits in Ports J and P

4. **Additional Notes for running the program**
   - Set up waveform generator using offset-check signal with scope
   - Load program before connecting generator
   - Disconnect generator between changing programs

```
;webex_irq0.asm, 12/2006, simplest and shortest version
;revision of text ex. 6.1 for use w/Dragon12
;to operate, this program requires connection to signal generator and ground
        org     $1500
count   rmb     1
        org     $2000
        lds     #$2000
        clr     count           ;start count at 0
        movb    #$FF,$0003      ;make Port B output by sending 1's to its DDR
        movb    #$FF,$026A      ;make Port J output by sending 1's to its DDR
        movb    #$FF,$025A      ;make Port P output by sending 1's to its DDR
        movb    #$00,$0268      ;enable LEDs using PJ1 (see schematics)
        movb    #$FF,$0258      ;disable 7-segment LEDs (send 1's to cathodes)
        movw    #IRQISR,$3E72   ;put ISR starting address in vector location
        movb    #$C0,$001E      ;select falling edge and enable(b7 and b6) using IRQCR
        cli                     ;global enable of interrupts
forever nop
        bra  forever            ;just waiting for interrupt to occur

;IRQ service routine
irqisr  inc     count
        movb    count,$0001
        rti
        end
```

```
;webex_irq1, 12/2006, using equates for ports, DDRs, useriq and intcr
;revision of text ex. 6.1 for use w/Dragon12
;to operate, this program requires connection to signal generator and ground
1.                  org     $1500
2.   count          rmb     1
3.   portb          equ     $0001       ;LEDs and 7-segment
4.   portj          equ     $0268                       ;Used to enable/disable LEDs
5.   portp          equ     $0258       ;Used to disable 7-segments via the cathodes
6.   ddrb           equ     $0003
7.   ddrj           equ     $026A
8.   ddrp           equ     $025A
9.   UserIRQ        equ      $3E72      ;vector for IRQ
10.  INTCR          equ     $001E       ;location of interrupt control register
11.  output         equ     $FF
12.                 org     $2000
13.                 lds      #$2000
14.                 clr     count                       ;start count at 0
15.                 movb    #output,ddrb
16.                 movb    #output,ddrj
17.                 movb    #output,ddrp
18.                 bclr    portj,$02                   ;enable LEDs using PJ1 (see schematics)
19.                 bset    portp,$07                   ;disable 7-segment LEDs (send 1's to cathodes)
20.                 movw    #IRQISR,UserIRQ             ;put ISR starting address in vector location
21.                 movb    #$C0,INTCR                  ;select falling edge and enable (b7 and b6)
22.                 cli                                 ;global enable of interrupts
23.  forever        nop
24.                 bra   forever                       ;just waiting for interrupt to occur
25.
26.  ;IRQ service routine
27.  irqisr         inc   count
28.                 movb   count,portb                  ;send count to LEDs
29.                 rti
30.                 end
```

```
;webex_irq2.asm, 12/2006, improve irq1 by including hcs12.inc (provides definitions)
;revision of text ex. 6.1 for use w/Dragon12
;to operate, this program requires connection to signal generator and ground
1.  #include         hcs12.inc                    ; include register equates
2.                   org    $1500
3.  count            rmb    1
4.  output           equ    $FF
5.        org        $2000
6.        lds        #$2000
7.        clr        count                        ;start count at 0
8.        movb       #output,ddrb
9.        movb       #output,ddrj
10.       movb       #output,ddrp
11.       bclr       ptj,$02                      ;enable LEDs using PJ1 (see schematics)
12.       bset       ptp,$07                      ;disable 7-segment LEDs (send 1's to cathodes)
13.                  movw    #IRQISR,UserIRQ      ;put ISR starting address in vector location
14.                  movb    #$C0,INTCR           ;select falling edge and enable (b7 and b6)
15.                  cli                          ;global enable of interrupts
16. forever          nop
17.                  bra    forever               ;just waiting for interrupt to occur
18.
19. ;IRQ service routine
20. irqisr           inc    count
21.                  movb   count,portb           ;send count to LEDs
22.                  rti
23.                  end
```

Nonmaskable Interrupts

- XIRQ pin (shared with PE0 pin) interrupt
  - Disabled during a system reset and upon entering the service routine of another XIRQ interrupt.
  - Software can clear the X bit of the CCR register to enable the (using the **andcc #$BF** instruction) XIRQ interrupt. Software cannot reset the X bit once it has been set.
  - When a nonmaskable interrupt is recognized, both the X and I bits are set after CPU registers are saved.
  - RTI instruction, at the end of the XIRQ service routine, restores the X and I bits to the pre-interrupt request state.

- Trap for Unimplemented Opcode
  - There are 202 unimplemented opcodes (16-bit opcode).
  - These unimplemented opcodes share the same vector $FFF8:$FFF9.

- Software interrupt instruction (SWI)
  - Execution causes an interrupt without a hardware signal.
  - Commonly used in the debug monitor to implement breakpoints, display register contents and transfer control from a user program to the debug monitor.

Interrupts in D-Bug12 EVB Mode

- On-chip flash memory locations are not available for user to store interrupt vectors.
- D-Bug12 monitor provides SRAM-for interrupt vector table.
- The SRAM-based table starts at $3E00 and has 64 entries.
- The SCI0 interrupt has been used by the monitor and is not available to the user.
- Mnemonic names are defined for users to store their interrupt vectors in the table. Both the **hcs12.inc** and the **vectors12.h** (for C language) have the definitions for these entries.

Setting Up the Interrupt Vector

- The label (or name) of the IRQ interrupt service routine is **IRQISR**.

- In assembly language

```
movw  #IRQISR,UserIRQ  ; stores the vector at the designated address
```

I/O Ports H, J, and P (actually in Chapter 7 but needed here)

- These ports have edge-triggered (rising or falling) interrupt capability in the wired-OR fashion and are programmed through the Port Device Enable Register and the Port Polarity Select Register

- These three I/O ports each have a set of eight registers:
  1. Port I/O register (PTH, PTJ, PTP)
  2. Port Input Register (PTIH, PTIJ, PTIP)
  3. Port Data Direction Register (DDRH, DDRJ, DDRP)
  4. Port Reduced Drive Register (RDRH, RDRJ, RDRP)
     - This type of register is discussed under Ports E and T in text
  5. Port Pull Device Enable Register (PERH, PERJ, PERP)
     - This type of register is also discussed under Ports E and T in text
  6. Port Polarity (edge) Select Register (PPSH, PPSJ, PPSP)
  7. Port Interrupt Enable Register (PIEH, PIEJ, PIEP)*
  8. Port Interrupt Flag Register (PIFH, PIFJ, PIFP)*

*Discussed on next slide

Two Useful Registers for Ports H, J, and P

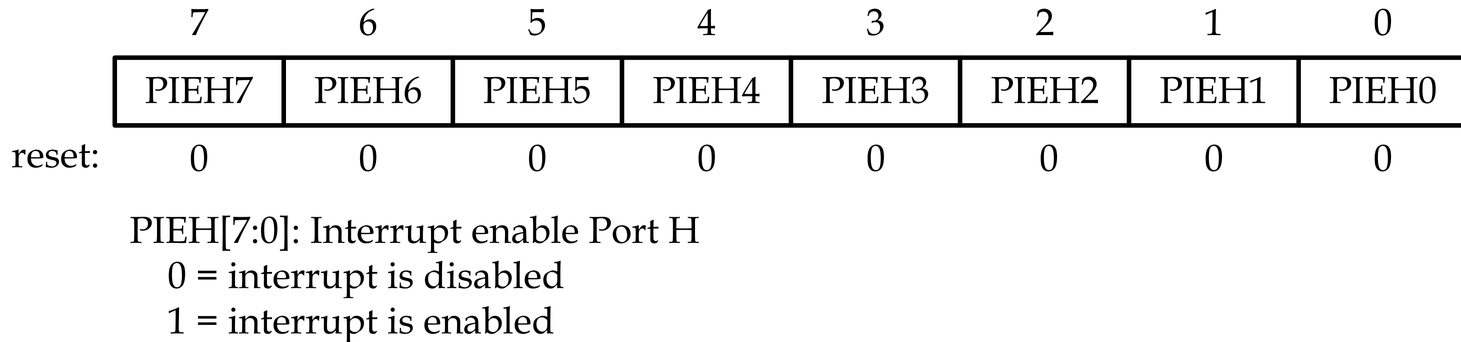

Figure 7.19 Port H Interrupt Enable Register (PIEH)

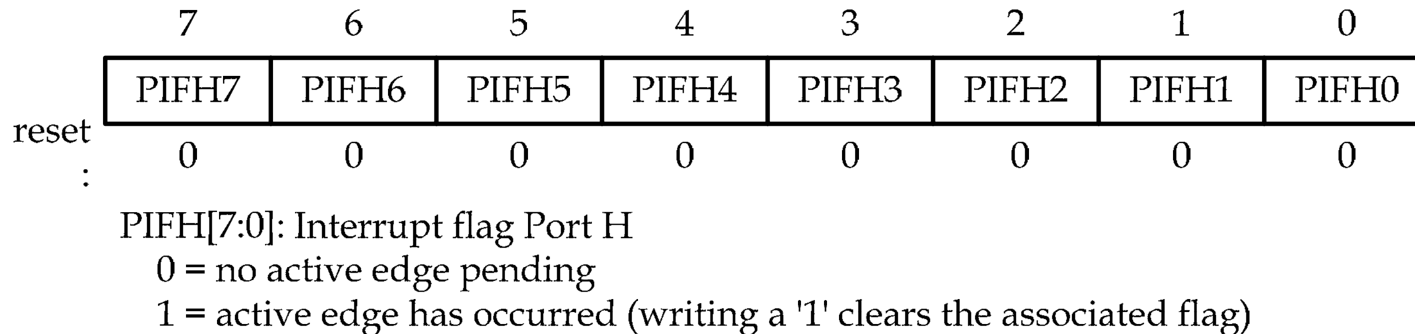

Figure 7.20 Port P Interrupt Flag Register (PIFH)

Some Useful Dragon12 Data

Mnemonic Name in hcs12.inc	Interrupt Source	RAM Vector Address
UserPortH	Port H Interrupt	$3E4C
UserPortJ	Port J Interrupt	$3E4E
UserIRQ	IRQ	$3E72
UserXIRQ	XIRQ	$3E74

Useful Information for Equates
(available in Appendix D of text, MC9S12P256 Registers from Freescale)

Register/Port	Address
PORTB	$0001
DDRB	$0003
PORTJ	$0268
DDRJ	$026A
PORTP	$0258
DDRP	$025A
PORTH	$0020
DDRH	$0262
PERH (pull device enable for H)	$0264
PPSH (+ or - edge selection for H)	$0265
PIEH (local mask for interrupt for H)	$0266
PIFH (interrupt flags for H)	$0267

Partial configuration of port H for interrupt operation using pushbutton 3 of the Dragon12

- From the Dragon12 schematic, rev. E, diagram 4 of 4: Pushbutton3 is connected to P50 and PH2 (i.e., bit 2 of port H)
- To make the bit to respond to a negative edge we need a ‘1’ in the appropriate bit of the PPSH register
  - From the Register table in appendix D, section Port Integration Module, location $0265, we will need:

```
movb #%00000100,ppsh ;ppsh previously defined as $0265
```

- To enable the bit (unmask) we need a ‘1’ in the appropriate bit of the PIEH register
  - From the Register table in appendix D, section Port Integration Module, location $0266, we will need:

```
movb #%00000100,pieh ;pieh previously defined as $0266
```

- Other registers need setting up also: PERH, DDRH, etc.
- While in the ISR, the local interrupt flag needs to be cleared before the rti
  - `bset  pifh,%00000100    ;pifh previously defined as $0267,note: a 1 clears flag`

7 Critical Issues for I/O and Interrupt Software

1. Which ports are needed for the input?
   - Associate port names with addresses using equ
   - Associate DDR names with addresses using equ
2. Which ports are needed for output?
   - Associate port names with addresses using equ
   - Associate DDR names with addresses using equ
3. Hardware at the ports
   - Any shared inputs? e.g., dip switches sharing with push buttons
   - Any shared outputs? e.g., LEDs sharing with 7-segment
4. Bitwise operations
   - All ports using 8 bits
   - Dividing up a port also does it change within program?

7 Critical Issues for I/O and Interrupt Software Continued

5. Other registers associated with ports
   - Triggering for interrupt operation
   - Pull device enabling
   - Enable/disable of local masks
   - Flags
   - Other
6. Masking (disabling) interrupts
   - Global
   - Local
7. Vector tables
   - Default for HCS12
   - Relocation; e.g., for evb operation with Dragon12