# Interrupts

Internal:

|  |  |  |
|---|---|---|
| Caused by: | Internal I/O subsystems, | e.g. AD end of conversion |
|  | Exceptional error condition, | e.g. COP |

External:

Caused by: A signal on pins XIRQ or IRQ
(edge or level sensitive / software selectable)

MC68HC12 contains vectored interrupts with hardware priority resolution that can be customized by software.

Vectored interrupts means each interrupt source has its own vector (address of the ISR) in memory (a pre-specified location in the interrupt vector table)

Priority means which reset or interrupt is serviced first when simultaneous requests are made.

## Interrupts Enable/Disable

Interrupt sources may be enabled (unmasked) or disabled (masked) selectively.

The I and X bits in the CCR act globally. (setting the I bit disables external IRQ and other maskable internal interrupt sources)

To disable interrupts from IRQ, set the I bit in the CCR:

```
ORCC     #%00010000
```

To enable interrupts from IRQ , reset the I bit in the CCR :

```
ANDCC   #%11101111
```

At RESET X and I are set and all interrupts are disabled.

After reset, X & I can be reset to enable interrupts.

Once enabled, the X bit cannot be reset directly by software.

Once the interrupt is acknowledged the I flag is automatically set masking any further interrupts.

A maskable internal interrupt source may be enabled or disabled selectively by setting or resetting a bit in a control register.

This bit must be reset (or the flag in the interrupt source cleared) by the ISR if the ISR resets the I bit (not recommended because it may cause nested interrupts).

## Interrupt Sequence

The interrupt sequence: (done transparently)

1. The CPU completes the currently executing instruction ( longer instructions may be interrupted before completion)
2. The CPU determines the address of the ISR through vectoring system
3. The return address is pushed onto the stack
4. All CPU registers are pushed onto the stack :Y,X,B:A,CCR
5. The I bit in the CCR is set to mask further interrupts (If the interrupt is caused by XIRQ, both X & I are set)
6. The interrupt vector is placed in the PC

The interrupt return: (upon IRT instruction)  (done transparently)

1. The CPU registers are pulled from the stack CCR, B:A, X, Y ( this step re-enables interrupts as it restores the old CCR which should have the I bit reset)
2. The return address is pulled from the stack

The ISR must reset the flag that caused the interrupt.

# Interrupt Vectors

The interrupt vector is the memory address of the start of the ISR for that interrupt source.

Each interrupt source has its own vector in memory.

| Address | Interrupt Source | CCR Mask | Local Enable | | HPRIO |
|---|---|---|---|---|---|
| $FFFE-$FFFF | $\overline{\text{Reset}}$ | None | None | None | - |
| $FFFC-$FFFD | Clock monitor fail reset | None | COPCTL | CME, FCME | - |
| $FFFA-$FFFB | COP failure reset | None | None | COP rate selected | - |
| $FFF8-$FFF9 | Unimplemented instruction trap | None | None | None | - |
| $FFF6-$FFF7 | SWI | None | None | None | - |
| $FFF4-$FFF5 | XIRQ | X bit | None | None | - |
| $FFF2-$FFF3 | IRQ | I bit | INTCR | IRQEN | $F2 |
| $FFF0-$FFF1 | Real-time Interrupt | I bit | CRGINT | RTIE | $F0 |
| $FFEE-$FFEF | Timer channel 0 | I bit | TMSK1 | C0I | $EE |
| $FFEC-$FFED | Timer channel 1 | I bit | TMSK1 | C1I | $EC |
| $FFD2-$FFD3 | AD subsystem | I bit | ASCIE | ATDCTL | $D2 |

DEVICE-Specific

**Continue**

# Initializing the Interrupt Vectors

The interrupt vector locations must be initialized by program. (system must not have a debugging monitor)

Example:

```
; Initialize Timer Channel 0 interrupt vector
TC0V:       equu    $FFEE   ; address of vector
PROG:       equ     $E000   ; program location
;
            ORG     PROG
; Main program
            .
            .
TC0_ISR:                    ; ISR starts here
            .
            .
            rti             ; ISR ends here
;
; Locate the vector
            ORG     TC0V
            DW      TC0_ISR
```

## Interrupt Priorities

Simultaneous interrupts must be resolved in a vectored system.

In M68HC12, this problem is solved by hardware priorities, but can be dynamically changed by program.

Any single interrupt source can be elevated to a higher priority while the rest of orders remain unchanged.

**Exception Priority :**

A hardware priority hierarchy determines which reset or interrupt is serviced first when simultaneous requests are made.

Six sources are not maskable: (listed according to priority)

1. Power-on reset (POR) or RESET pin
2. Clock monitor reset
3. Computer operating properly (COP) watchdog reset
4. Unimplemented instruction trap
5. Software interrupt instruction (SWI)
6. XIRQ signal if X bit in CCR = 0

The remaining sources are maskable, and any one of them can be given priority over other maskable interrupts.

## Raising an Interrupt Priority

Raise the Timer Channel 2 to the highest priority.
Means write $EA to the HPRIO register.     ….HPRIO register address is $1F

Example:

```
HPRIO:      EQU     $1F
TC2V:       EQU     $FFEA   ;Channel 2 vector
;
;Disable interrupts while setting HPRIO
            ORCC    #$10     ; set I bit in CCR
;
; Raise the Timer Channel 2 to the highest priority
            LDD     #TC2V
            STAB    HPRIO
;Re-enable interrupts
            ANDCC   #$EF
```

## Latching of Interrupts

XIRQ is always level triggered and IRQ can be selected as a level-triggered interrupt.

Level-triggered interrupt pins should be released by ISR.

Generally, the interrupt service routine will handshake with the interrupting logic to release the pin.

If IRQ is selected as an edge-triggered interrupt, the hold time of the level after the active edge is independent of when the interrupt is serviced. As long as the minimum hold time is met, the interrupt will be latched inside the MCU. In this case, the IRQ edge interrupt latch is cleared automatically when the interrupt is serviced.

All of the remaining interrupts are latched by the MCU with a flag bit. These interrupt flags should be cleared by the ISR

# Interrupt Control Register (INTCR)
## (In MC9S12C)

| Address $001E | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | IRQE | IRQEN | 0 | 0 | 0 | 0 | 0 | 0 |
| Reset | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

**IRQE — IRQ Edge-Sensitive Only Bit**

1 = IRQ pin responds only to falling edges.

0 = IRQ pin responds to low levels.

IRQE can be written once in normal modes. In special modes, IRQE can be written anytime, but the first write is ignored.

**IRQEN — External IRQ Enable Bit**

1 = IRQ pin connected to interrupt logic

0 = IRQ pin disconnected from interrupt logic

IRQEN can be written anytime in all modes. The IRQ pin has an internal pullup.

# Highest Priority Control Register (HPRIO)
## (In MC9S12C)

| Address $001F | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | PSEL7 | PSEL6 | PSEL5 | PSEL4 | PSEL3 | PSEL2 | PSEL1 | 0 |
| Write | | | | | | | | ▨ |
| Reset | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |

To give a maskable interrupt source highest priority, write the low byte of the vector address to the HPRIO register.

For example, writing $F0 to HPRIO assigns highest maskable interrupt priority to the real-time interrupt ($FFF0).(from table)

If an unimplemented vector address or a non-I-masked vector address (a value higher than $F2) is written, then IRQ is the default highest priority interrupt.

## A External Interrupt Example

Assume that the IRQ pin of the HCS12DP256 is connected to a 1-Hz digital waveform and Port B is connected to eight LEDs.

It is required to increment the count on the LEDs every one second.

Write a program to:

- –configure Port B for output
- –Initialize the interrupt vector for IRQ
- –enable the IRQ interrupt
- –wait for interrupt

Also write the IRQ service routine that simply:

- –increments a counter
- –outputs it to Port B.

# Code for External Interrupt Example

```
IRQCR    equ  $1E
         org      $1000
count   ds.b     1                          ; reserve 1 byte in RAM for count
         org     $1500                      ; start code in FLASH
         lds      #$1500                    ; set up the stack pointer to RAM end
         movw  #ISR,$FFF2                   ; set up interrupt vector for IRQ
         clr       count
         movb   #$FF,DDRB                   ; configure Port B for output
         movb   count,PORTB                 ; clear LEDs
         movb   #$C0,IRQCR                  ; enable IRQ and select edge triggering
         cli                                ; clear I bit in CCR to enable interrupts
wLoop bra      wLoop                        ; wait for IRQ pin interrupt
;
; This is the IRQ service routine.
ISR      inc       count                    ; increment count
         movb    count,PORTB                ; and display count on LEDs
         rti
         end
```

# Real Time Interrupt (RTI)

The main function of the RTI circuit is to generate hardware interrupts periodically.

RTI is enabled by setting RTIE=1 in CRGINT register

The RTI period is selected by writing to the RTICTL register.

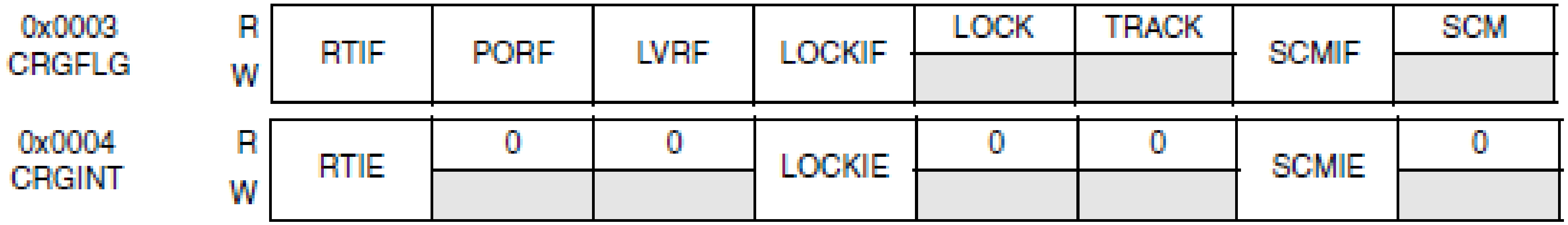
The interrupt period is made of multiples of OSCCLK cycles.

The RTI runs with a gated OSCCLK

At the end of the RTI time-out period the RTIF flag is set to 1 and a new RTI time-out period starts immediately.

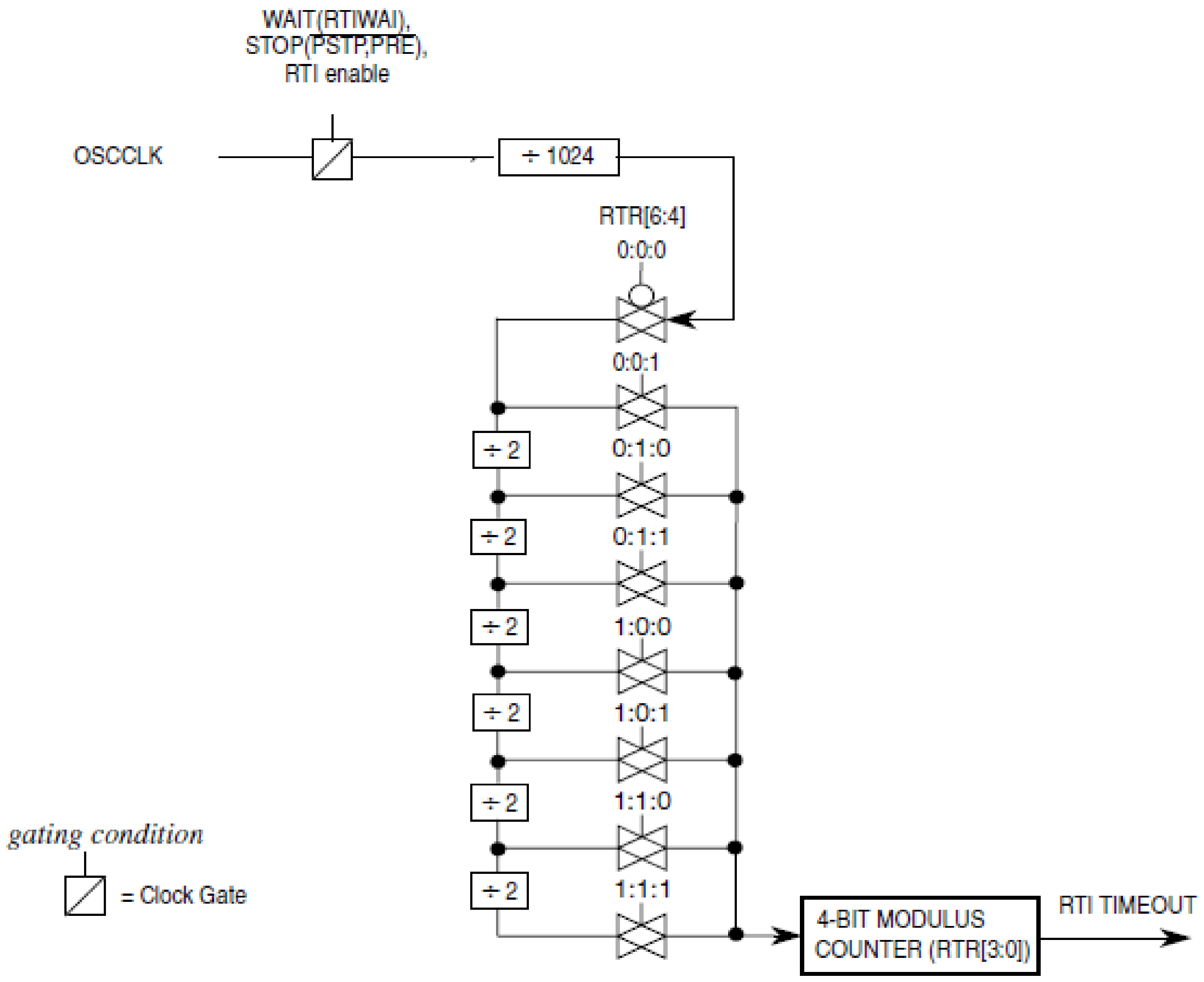
A write to the RTICTL register restarts the RTI time-out period.

The real-time interrupt flag (RTIF) is set to 1 when a timeout occurs, and is cleared to 0 by writing a 1 to the RTIF bit.



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0x0003 CRGFLG | R | RTIF | PORF | LVRF | LOCKIF | LOCK | TRACK | SCMIF | SCM |
| | W | | | | | | | | |
| 0x0004 CRGINT | R | RTIE | 0 | 0 | LOCKIE | 0 | 0 | SCMIE | 0 |
| | W | | | | | | | | |

## Clock Chain for RTI Generation



# Selecting RTI period in (RTICTL)

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | 0 | RTR6 | RTR5 | RTR4 | RTR3 | RTR2 | RTR1 | RTR0 |
| Reset: | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

| RTR[3:0] | RTR[6:4] | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 000 (off) | 001 ($2^{10}$) | 010 ($2^{11}$) | 011 ($2^{12}$) | 100 ($2^{13}$) | 101 ($2^{14}$) | 110 ($2^{15}$) | 111 ($2^{16}$) |
| 0000 (÷1) | off* | $2^{10}$ | $2^{11}$ | $2^{12}$ | $2^{13}$ | $2^{14}$ | $2^{15}$ | $2^{16}$ |
| 0001(÷2) | off* | $2 \times 2^{10}$ | $2 \times 2^{11}$ | $2 \times 2^{12}$ | $2 \times 2^{13}$ | $2 \times 2^{14}$ | $2 \times 2^{15}$ | $2 \times 2^{16}$ |
| 0010 (÷3) | off* | $3 \times 2^{10}$ | $3 \times 2^{11}$ | $3 \times 2^{12}$ | $3 \times 2^{13}$ | $3 \times 2^{14}$ | $3 \times 2^{15}$ | $3 \times 2^{16}$ |
| 0011 (÷4) | off* | $4 \times 2^{10}$ | $4 \times 2^{11}$ | $4 \times 2^{12}$ | $4 \times 2^{13}$ | $4 \times 2^{14}$ | $4 \times 2^{15}$ | $4 \times 2^{16}$ |
| 0100 (÷5) | off* | $5 \times 2^{10}$ | $5 \times 2^{11}$ | $5 \times 2^{12}$ | $5 \times 2^{13}$ | $5 \times 2^{14}$ | $5 \times 2^{15}$ | $5 \times 2^{16}$ |
| 0101 (÷6) | off* | $6 \times 2^{10}$ | $6 \times 2^{11}$ | $6 \times 2^{12}$ | $6 \times 2^{13}$ | $6 \times 2^{14}$ | $6 \times 2^{15}$ | $6 \times 2^{16}$ |
| 0110 (÷7) | off* | $7 \times 2^{10}$ | $7 \times 2^{11}$ | $7 \times 2^{12}$ | $7 \times 2^{13}$ | $7 \times 2^{14}$ | $7 \times 2^{15}$ | $7 \times 2^{16}$ |
| 0111 (÷8) | off* | $8 \times 2^{10}$ | $8 \times 2^{11}$ | $8 \times 2^{12}$ | $8 \times 2^{13}$ | $8 \times 2^{14}$ | $8 \times 2^{15}$ | $8 \times 2^{16}$ |
| 1000 (÷9) | off* | $9 \times 2^{10}$ | $9 \times 2^{11}$ | $9 \times 2^{12}$ | $9 \times 2^{13}$ | $9 \times 2^{14}$ | $9 \times 2^{15}$ | $9 \times 2^{16}$ |
| 1001 (÷10) | off* | $10 \times 2^{10}$ | $10 \times 2^{11}$ | $10 \times 2^{12}$ | $10 \times 2^{13}$ | $10 \times 2^{14}$ | $10 \times 2^{15}$ | $10 \times 2^{16}$ |
| 1010 (÷11) | off* | $11 \times 2^{10}$ | $11 \times 2^{11}$ | $11 \times 2^{12}$ | $11 \times 2^{13}$ | $11 \times 2^{14}$ | $11 \times 2^{15}$ | $11 \times 2^{16}$ |
| 1011 (÷12) | off* | $12 \times 2^{10}$ | $12 \times 2^{11}$ | $12 \times 2^{12}$ | $12 \times 2^{13}$ | $12 \times 2^{14}$ | $12 \times 2^{15}$ | $12 \times 2^{16}$ |
| 1100 (÷13) | off* | $13 \times 2^{10}$ | $13 \times 2^{11}$ | $13 \times 2^{12}$ | $13 \times 2^{13}$ | $13 \times 2^{14}$ | $13 \times 2^{15}$ | $13 \times 2^{16}$ |
| 1101 (÷14) | off* | $14 \times 2^{10}$ | $14 \times 2^{11}$ | $14 \times 2^{12}$ | $14 \times 2^{13}$ | $14 \times 2^{14}$ | $14 \times 2^{15}$ | $14 \times 2^{16}$ |
| 1110 (÷15) | off* | $15 \times 2^{10}$ | $15 \times 2^{11}$ | $15 \times 2^{12}$ | $15 \times 2^{13}$ | $15 \times 2^{14}$ | $15 \times 2^{15}$ | $15 \times 2^{16}$ |
| 1111 (÷16) | off* | $16 \times 2^{10}$ | $16 \times 2^{11}$ | $16 \times 2^{12}$ | $16 \times 2^{13}$ | $16 \times 2^{14}$ | $16 \times 2^{15}$ | $16 \times 2^{16}$ |

RTI period (in units of OSCCLK cycle)