3.4 Real Time Interrupt (RTI)

The RTI is a module part of the clock system used to generate a hardware interrupt, every period of time previously defined in its configuration register. It is a convenient way of implementing a time tick that can be used for multiple purposes, that also has the advantage of only depending on the crystal oscillator speed (OSCCLK), which is 16[MHz] in our micro board, and not on the actual bus speed that can be changed using the PLL.

There are two simple steps to configure the RTI to 1[ms] interrupt event:

STEP 1

- Configure CRG RTI Control Register (RTICTL) according to the following table:

RTR[3:0]	**RTR[6:4]**							
	000 (1x10^3)	001 (2x10^3)	010 (5x10^3)	011 (10x10^3)	100 (20x10^3)	101 (50x10^3)	110 (100x10^3)	111 (200x10^3)
0000 (÷1)	1x10^3	2x10^3	5x10^3	10x10^3	20x10^3	50x10^3	100x10^3	200x10^3
0001 (÷2)	2x10^3	4x10^3	10x10^3	20x10^3	40x10^3	100x10^3	200x10^3	400x10^3
0010 (÷3)	3x10^3	6x10^3	15x10^3	30x10^3	60x10^3	150x10^3	300x10^3	600x10^3
0011 (÷4)	4x10^3	8x10^3	20x10^3	40x10^3	80x10^3	200x10^3	400x10^3	800x10^3
0100 (÷5)	5x10^3	10x10^3	25x10^3	50x10^3	100x10^3	250x10^3	500x10^3	1x10^6
0101 (÷6)	6x10^3	12x10^3	30x10^3	60x10^3	120x10^3	300x10^3	600x10^3	1.2x10^6
0110 (÷7)	7x10^3	14x10^3	35x10^3	70x10^3	140x10^3	350x10^3	700x10^3	1.4x10^6
0111 (÷8)	8x10^3	16x10^3	40x10^3	80x10^3	160x10^3	400x10^3	800x10^3	1.6x10^6
1000 (÷9)	9x10^3	18x10^3	45x10^3	90x10^3	180x10^3	450x10^3	900x10^3	1.8x10^6
1001 (÷10)	10x10^3	20x10^3	50x10^3	100x10^3	200x10^3	500x10^3	1x10^6	2x10^6
1010 (÷11)	11x10^3	22x10^3	55x10^3	110x10^3	220x10^3	550x10^3	1.1x10^6	2.2x10^6
1011 (÷12)	12x10^3	24x10^3	60x10^3	120x10^3	240x10^3	600x10^3	1.2x10^6	2.4x10^6
1100 (÷13)	13x10^3	26x10^3	65x10^3	130x10^3	260x10^3	650x10^3	1.3x10^6	2.6x10^6
1101 (÷14)	14x10^3	28x10^3	70x10^3	140x10^3	280x10^3	700x10^3	1.4x10^6	2.8x10^6
1110 (÷15)	15x10^3	30x10^3	75x10^3	150x10^3	300x10^3	750x10^3	1.5x10^6	3x10^6
1111 (÷16)	16x10^3	32x10^3	80x10^3	160x10^3	320x10^3	800x10^3	1.6x10^6	3.2x10^6

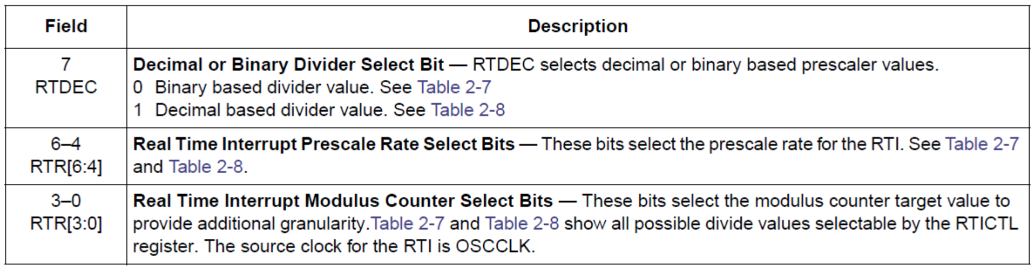
Table 1 - Clock Division Table

RTR[3:0]	**RTR[6:4]**							
	000 (1x10^3)	001 (2x10^3)	010 (5x10^3)	011 (10x10^3)	100 (20x10^3)	101 (50x10^3)	110 (100x10^3)	111 (200x10^3)
0000 (÷1)	62.5[us]	125[us]	312.5[us]	625[us]	1.25[ms]	3.125[ms]	6.25[ms]	12.5[ms]
0001 (÷2)	125[us]	250[us]	625[us]	1.25[ms]	2.5[ms]	6.25[ms]	12.5[ms]	25[ms]
0010 (÷3)	187.5[us]	375[us]	937.5[us]	1.875[ms]	3.75[ms]	9.375[ms]	18.75[ms]	37.5[ms]
0011 (÷4)	250[us]	500[us]	1.25[ms]	2.5[ms]	5[ms]	12.5[ms]	25[ms]	50[ms]
0100 (÷5)	312.5[us]	625[us]	1.5625[ms]	3.125[ms]	6.25[ms]	15.625[ms]	31.25[ms]	62.5[ms]
0101 (÷6)	375[us]	750[us]	1.875[ms]	3.75[ms]	7.5[ms]	18.75[ms]	37.5[ms]	62.5[ms]
0110 (÷7)	437.5[us]	875[us]	2.1875[ms]	4.375[ms]	8.75[ms]	21.875[ms]	43.75[ms]	75[ms]
0111 (÷8)	500[us]	1[ms]	2.5[ms]	5[ms]	10[ms]	25[ms]	50[ms]	100[ms]
1000 (÷9)	562.5[us]	1.125[ms]	2.8125[ms]	5.625[ms]	11.25[ms]	28.125[ms]	56.25[ms]	112.5[ms]
1001 (÷10)	625[us]	1.25[ms]	3.125[ms]	6.25[ms]	12.5[ms]	31.25[ms]	62.5[ms]	125[ms
1010 (÷11)	687.5[us]	1.375[ms]	3.4375[ms]	6.875[ms]	13.75[ms]	34.375[ms]	68.75[ms]	137.5[ms
1011 (÷12)	750[us]	1.5x[ms]	3.75[ms]	7.5[ms]	15[ms]	37.5[ms]	75[ms]	150[ms
1100 (÷13)	812.5[us]	1.625[ms]	4.0625[ms]	8.125[ms]	16.25[ms]	40.625[ms]	81.25[ms]	162.5[ms]
1101 (÷14)	875[us]	1.75[ms]	4.375[ms]	8.75[ms]	17.5[ms]	43.75[ms]	87.5[ms]	175[ms]
1110 (÷15)	937.5[us]	1.875[ms]	4.6875[ms]	9.375[ms]	18.75[ms]	46.875[ms]	93.75[ms]	187.5[ms]
1111 (÷16)	1[ms]	2[ms]	5[ms]	10[ms]	20[ms]	50[ms]	100[ms]	200[ms]

Table 2 - Time Event Table for 16MHz OSCLK

CRG RTI Control Register (RTICTL) *[datasheet 2.3.2.8]*

RTDEC	RTR6	RTR5	RTR4	RTR3	RTR2	RTR1	RTR0
BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
1	0	0	1	0	1	1	1



Field	Description
7 RTDEC	**Decimal or Binary Divider Select Bit** — RTDEC selects decimal or binary based prescaler values. 0 Binary based divider value. See Table 2-7 1 Decimal based divider value. See Table 2-8
6–4 RTR[6:4]	**Real Time Interrupt Prescale Rate Select Bits** — These bits select the prescale rate for the RTI. See Table 2-7 and Table 2-8.
3–0 RTR[3:0]	**Real Time Interrupt Modulus Counter Select Bits** — These bits select the modulus counter target value to provide additional granularity.Table 2-7 and Table 2-8 show all possible divide values selectable by the RTICTL register. The source clock for the RTI is OSCCLK.

- We SET(1) RTDEC to select Decimal presale

- According to the prescale rate (RTR), we configure BIT6 - BIT4 to 001b = 2x10^3, so we can divide the 16MHz Clock (Crystal) by 2,000. Therefore, 16MHz / 2,000 = 8KHz

- According to the Modulus counter setting, we configure BIT3 - BIT0 to 0111b (Mod8 Counter) so we can divide even further the now 8KHz clock by 8 so we can get a perfect 1KHz clock. Therefore, 8KHz / 8 = 1KHz (1ms event).

Example

```
  //Decimal, divider 2000, mod8 counter -> 1ms
  RTICTL = 0b10010111;
```

STEP 2

- Enable the RTI module. We just have to **SET(1) RTIE** to enable the RTI.

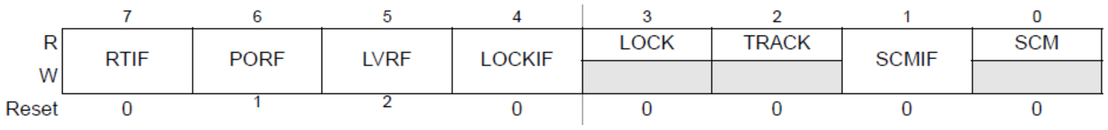
Example

```
CRGINT |= CRGINT_RTIE_MASK;    //0b10000000,  Enable RTI
```

3.4.1 RTI Interrupt Handling

To handle the interrupt, we perform the following actions in the Interrupt Service Routine (ISR)

- Clear Flag in the CRGFLG register by putting a (1) in the RTIF bit

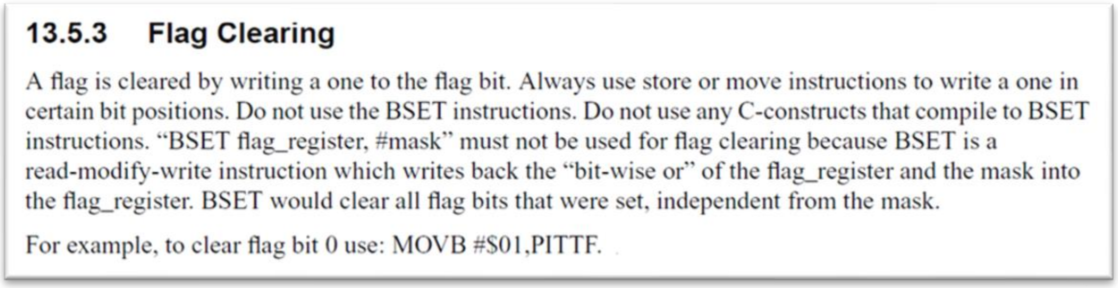


	7	6	5	4	3	2	1	0
R	RTIF	PORF	LVRF	LOCKIF	LOCK	TRACK	SCMIF	SCM
W					▨	▨		▨
Reset	0	1	2	0	0	0	0	0

- Add Any action desired to the ISR.

```
interrupt VectorNumber_Vrti void Vrti_ISR(void)
{
   CRGFLG = CRGFLG_RTIF_MASK;    //clear flag;
  //Perform some action here
}
```

The process for clearing the flag is quite unlike for most microcontrollers. An active flag will have a “1” in the **CRGFLG** register bit masked, which could make us think that writing a zero in the same bit would clear it. The process is quite the opposite; according to **13.5.3** of the datasheet, the flag is cleared by writing a “1” into the flag bit that needs to be cleared and zeroes in every other bit. However, the operation is done with “=” and not “ |= ”, meaning the entire register must be written since an “or-equals” involves a read-modify-write instruction that will end up clearing all the active flags.



13.5.3 Flag Clearing

A flag is cleared by writing a one to the flag bit. Always use store or move instructions to write a one in certain bit positions. Do not use the BSET instructions. Do not use any C-constructs that compile to BSET instructions. “BSET flag_register, #mask” must not be used for flag clearing because BSET is a read-modify-write instruction which writes back the “bit-wise or” of the flag_register and the mask into the flag_register. BSET would clear all flag bits that were set, independent from the mask.

For example, to clear flag bit 0 use: MOVB #$01,PITTF.

Figure 1 -Extracted from the 9S12XDP512 Datasheet.