Electrical Engineering
New Mexico Institute of Mining and Technology

EE 308/MENG 483   Spring 2017

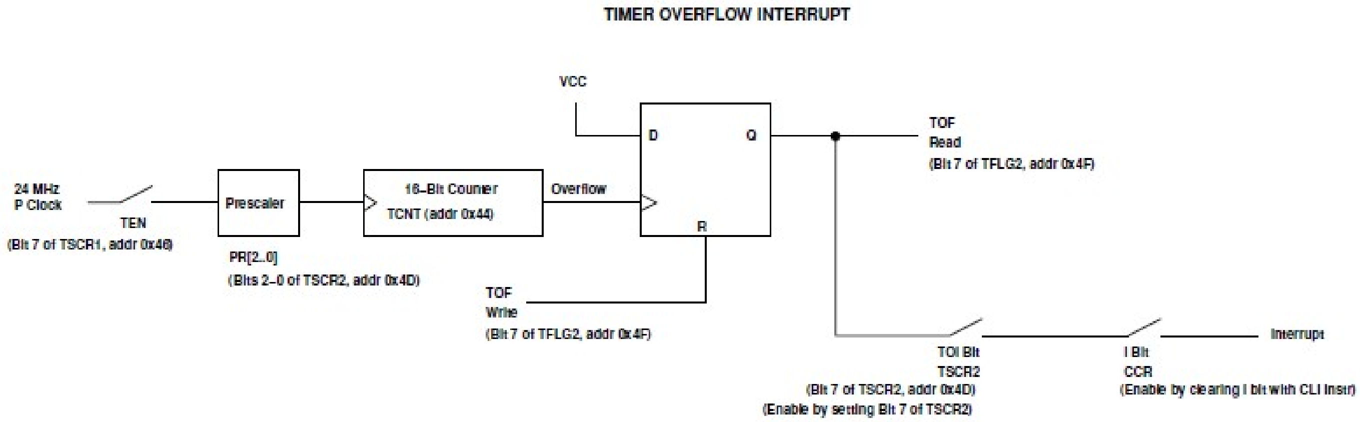
Exam II Review April 2017

Introduction to the MC9S12 Timer Subsystem

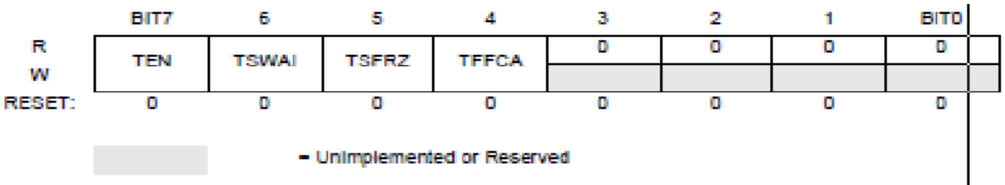
The MC9S12 has a 16-bit counter that runs with a 24 MHz.

The clock starts at 0x0000, counts up until it gets to 0xFFFF.

It takes 2.7307 ms (65,536 counts/24,000,000 counts/sec) for the counter to count from 0x0000 to 0xFFFF and roll over to 0x0000.


TIMER OVERFLOW INTERRUPT
VCC
D
Q
R
TOF
Read
(Bit 7 of TFLG2, addr 0x4F)
24 MHz
P Clock
TEN
(Bit 7 of TSCR1, addr 0x46)
Prescaler
PR[2..0]
(Bits 2–0 of TSCR2, addr 0x4D)
16–Bit Counter
TCNT (addr 0x44)
Overflow
TOF
Write
(Bit 7 of TFLG2, addr 0x4F)
TOI Bit
TSCR2
(Bit 7 of TSCR2, addr 0x4D)
(Enable by setting Bit 7 of TSCR2)
I Bit
CCR
(Enable by clearing I bit with CLI instr)
Interrupt

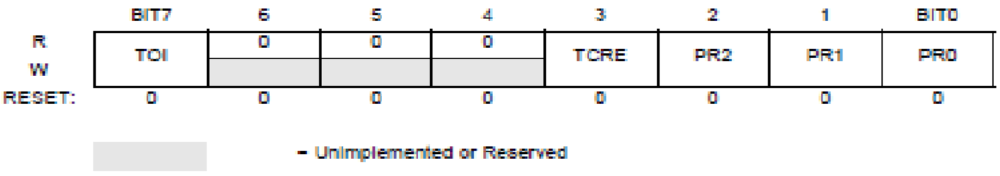
Registers used to enable Timer Overflow



	BIT7	6	5	4	3	2	1	BIT0
R	TEN	TSWAI	TSFRZ	TFFCA	0	0	0	0
W					▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0

▨ - Unimplemented or Reserved

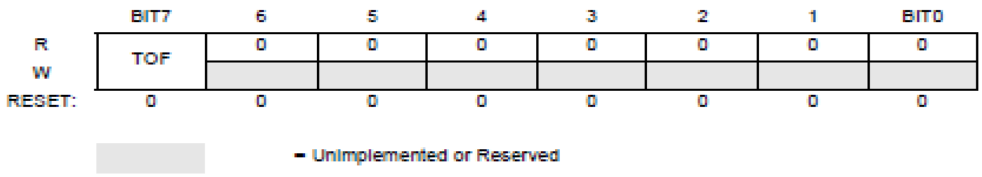
Figure 3-6 Timer System Control Register 1 (TSCR1)



	BIT7	6	5	4	3	2	1	BIT0
R	TOI	0	0	0	TCRE	PR2	PR1	PR0
W		▨	▨	▨				
RESET:	0	0	0	0	0	0	0	0

▨ - Unimplemented or Reserved

Figure 3-11 Timer System Control Register 2 (TSCR2)



	BIT7	6	5	4	3	2	1	BIT0
R	TOF	0	0	0	0	0	0	0
W		▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0

▨ - Unimplemented or Reserved

Figure 3-13 Main Timer Interrupt Flag 2 (TFLG2)

Code needed to use the timer overflow:

```
...
asm(sei);                  /* Disable interrupts */
TSCR1 = 0x80;              /* Turn on timer */
TSCR2 = 0x85;              /* Enable timer overflow interrupt, set */
                           /*  prescaler */
TFLG2 = 0x80;              /* Clear timer interrupt flag */
UserTimerOvf = (unsigned short)&toi_isr;

asm(cli);                  /* Enable interrupts (clear I bit) */
while (1)
{
     /* Put code here to do things */
}
...

void interrupt toi_isr(void)
{
     ...
     TFLG2 = 0x80;                 /* Clear timer interrupt flag */
}
```

By setting prescaler bits PR2,PR1,PR0 of TSCR2 you can slow down the clock:

PR	Divide	Freq	Overflow Rate
000	1	24 MHz	2.7307 ms
001	2	12 MHz	5.4613 ms
010	4	6 MHz	10.9227 ms
011	8	3 MHz	21.8453 ms
100	16	1.5 MHz	43.6907 ms
101	32	0.75 MHz	87.3813 ms
110	64	0.375 MHz	174.7627 ms
111	128	0.1875 MHz	349.5253 ms

Interrupt vectors for the HCS12

The interrupt vectors for the MC9S12DP256 are located in memory from 0xFF80 to 0xFFFF.

These vectors are programmed into Flash EEPROM and are very difficult to change

DBug12 redirects the interrupts to a region of RAM where they are easy to change

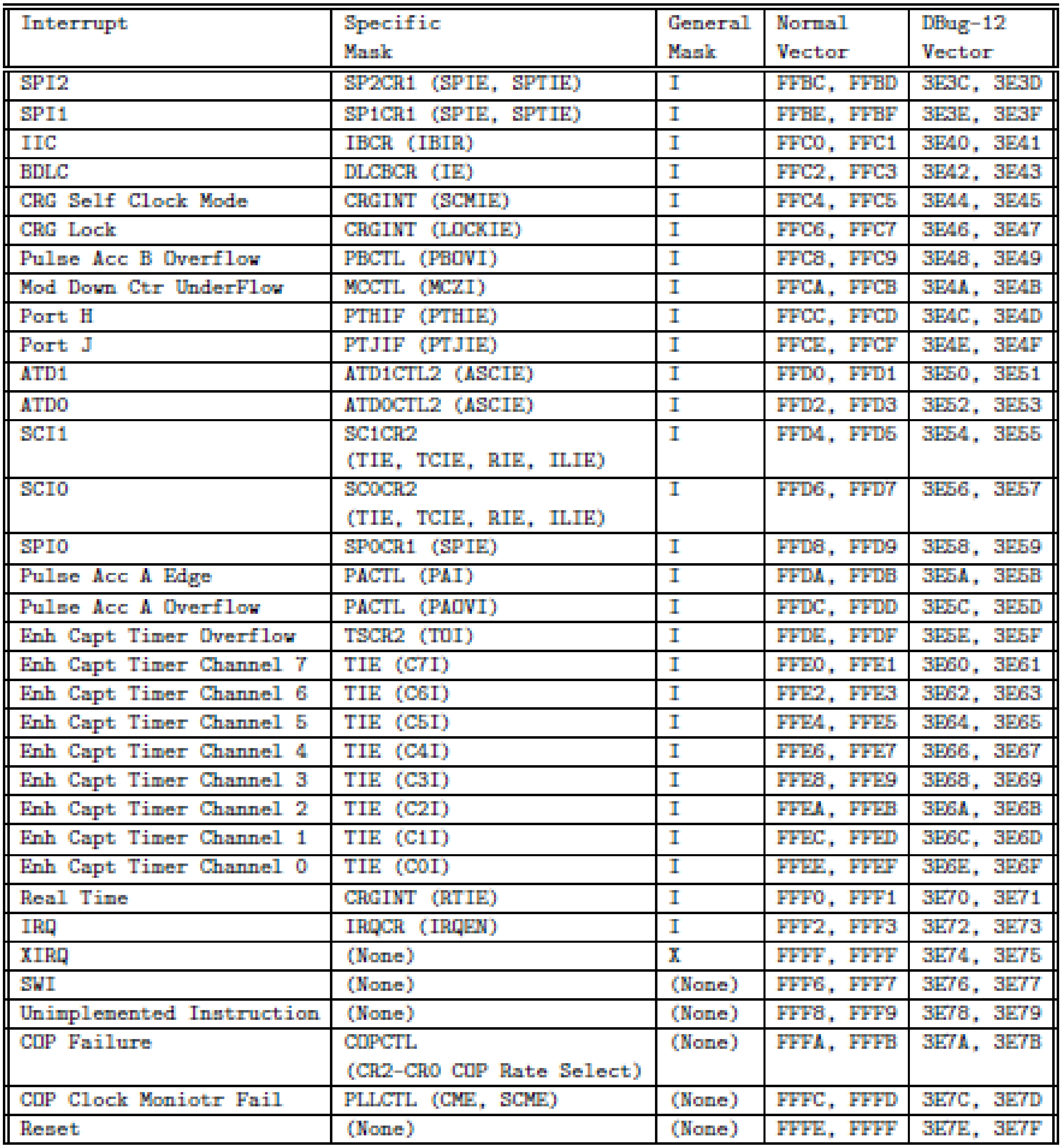
For example, when the MC9S12 gets a TOF interrupt:

– It loads the PC with the contents of **0xFFDE** and **0xFFDF**.

– The program at that address tells the MC9S12 to look at address **0x3E5E** and **0x3E5F**.

For C programs, the vectors are defined in the file vectors12.h.
Here is the define statement for the TOF:

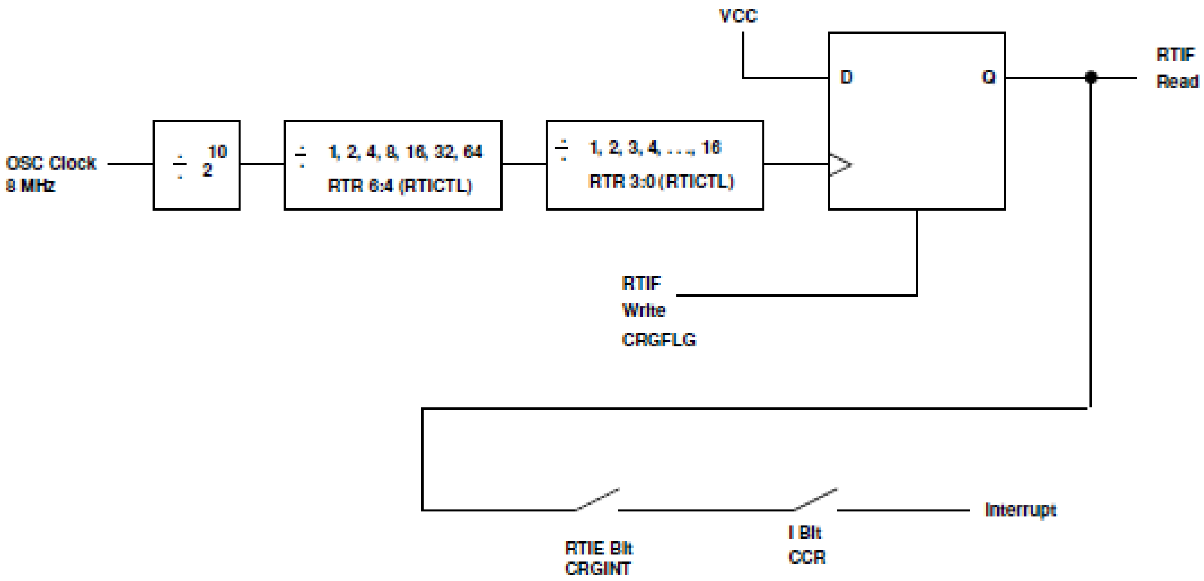
```
#define UserTimerOvf _VEC16(47)  /* Maps to 0x3E5E */
```



Interrupt	Specific Mask	General Mask	Normal Vector	DBug-12 Vector
SPI2	SP2CR1 (SPIE, SPTIE)	I	FFBC, FFBD	3E3C, 3E3D
SPI1	SP1CR1 (SPIE, SPTIE)	I	FFBE, FFBF	3E3E, 3E3F
IIC	IBCR (IBIR)	I	FFC0, FFC1	3E40, 3E41
BDLC	DLCBCR (IE)	I	FFC2, FFC3	3E42, 3E43
CRG Self Clock Mode	CRGINT (SCMIE)	I	FFC4, FFC5	3E44, 3E45
CRG Lock	CRGINT (LOCKIE)	I	FFC6, FFC7	3E46, 3E47
Pulse Acc B Overflow	PBCTL (PBOVI)	I	FFC8, FFC9	3E48, 3E49
Mod Down Ctr UnderFlow	MCCTL (MCZI)	I	FFCA, FFCB	3E4A, 3E4B
Port H	PTHIF (PTHIE)	I	FFCC, FFCD	3E4C, 3E4D
Port J	PTJIF (PTJIE)	I	FFCE, FFCF	3E4E, 3E4F
ATD1	ATD1CTL2 (ASCIE)	I	FFD0, FFD1	3E50, 3E51
ATD0	ATD0CTL2 (ASCIE)	I	FFD2, FFD3	3E52, 3E53
SCI1	SC1CR2 (TIE, TCIE, RIE, ILIE)	I	FFD4, FFD5	3E54, 3E55
SCI0	SC0CR2 (TIE, TCIE, RIE, ILIE)	I	FFD6, FFD7	3E56, 3E57
SPI0	SP0CR1 (SPIE)	I	FFD8, FFD9	3E58, 3E59
Pulse Acc A Edge	PACTL (PAI)	I	FFDA, FFDB	3E5A, 3E5B
Pulse Acc A Overflow	PACTL (PAOVI)	I	FFDC, FFDD	3E5C, 3E5D
Enh Capt Timer Overflow	TSCR2 (TOI)	I	FFDE, FFDF	3E5E, 3E5F
Enh Capt Timer Channel 7	TIE (C7I)	I	FFE0, FFE1	3E60, 3E61
Enh Capt Timer Channel 6	TIE (C6I)	I	FFE2, FFE3	3E62, 3E63
Enh Capt Timer Channel 5	TIE (C5I)	I	FFE4, FFE5	3E64, 3E65
Enh Capt Timer Channel 4	TIE (C4I)	I	FFE6, FFE7	3E66, 3E67
Enh Capt Timer Channel 3	TIE (C3I)	I	FFE8, FFE9	3E68, 3E69
Enh Capt Timer Channel 2	TIE (C2I)	I	FFEA, FFEB	3E6A, 3E6B
Enh Capt Timer Channel 1	TIE (C1I)	I	FFEC, FFED	3E6C, 3E6D
Enh Capt Timer Channel 0	TIE (C0I)	I	FFEE, FFEF	3E6E, 3E6F
Real Time	CRGINT (RTIE)	I	FFF0, FFF1	3E70, 3E71
IRQ	IRQCR (IRQEN)	I	FFF2, FFF3	3E72, 3E73
XIRQ	(None)	X	FFFF, FFFF	3E74, 3E75
SWI	(None)	(None)	FFF6, FFF7	3E76, 3E77
Unimplemented Instruction	(None)	(None)	FFF8, FFF9	3E78, 3E79
COP Failure	COPCTL (CR2-CR0 COP Rate Select)	(None)	FFFA, FFFB	3E7A, 3E7B
COP Clock Moniotr Fail	PLLCTL (CME, SCME)	(None)	FFFC, FFFD	3E7C, 3E7D
Reset	(None)	(None)	FFFE, FFFF	3E7E, 3E7F

The Real Time Interrupt

Like the Timer Overflow Interrupt, the Real Time Interrupt allows you to interrupt the processor at a regular interval.



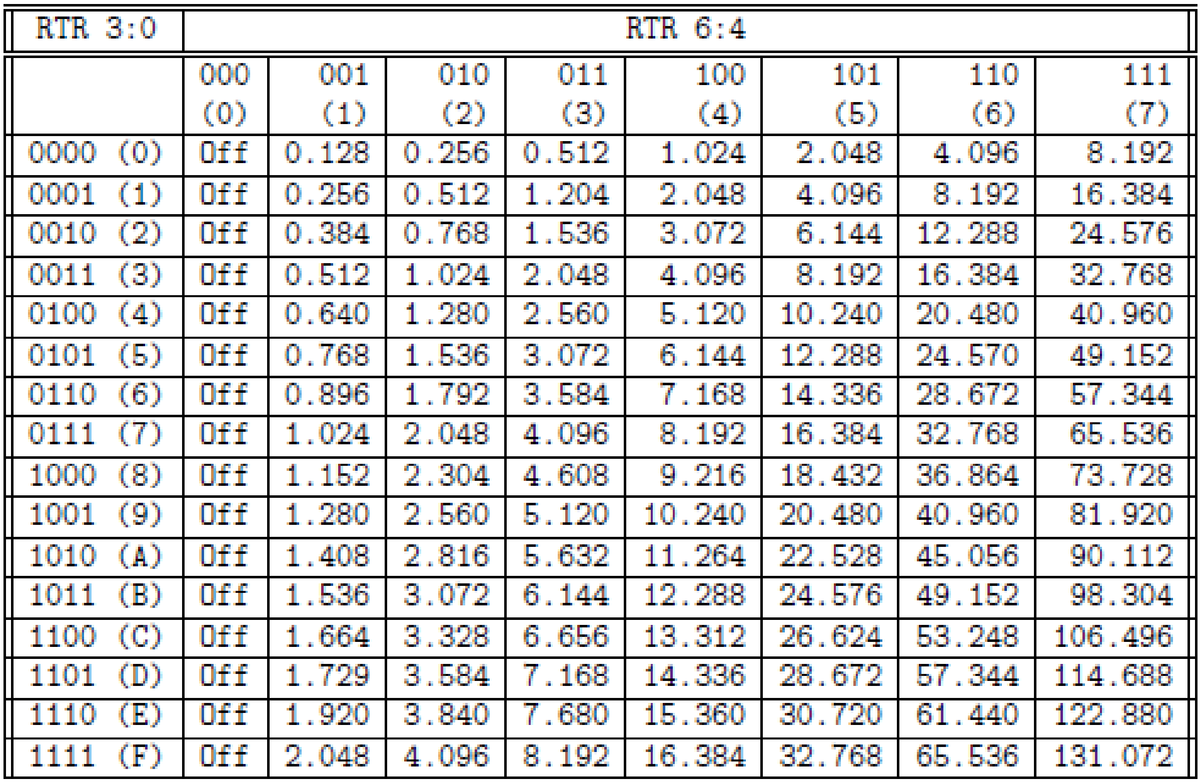
Registers you need to work with RTIs are:

RTIF	PORF	0	LOCKIF	LOCK	TRACK	SCMIF	SCM	0x0037	CRGFLG

RTIE	0	0	LOCKIE	0	0	SCMIE	0	0x0038	CRGINT

0	RTR6	RTR5	RTR4	RTR3	RTR2	RTR1	RTR0	0x003B	RTICTL

The following table shows all possible values, in ms, selectable by the RTICTL register (assuming a 8 MHz oscillator):



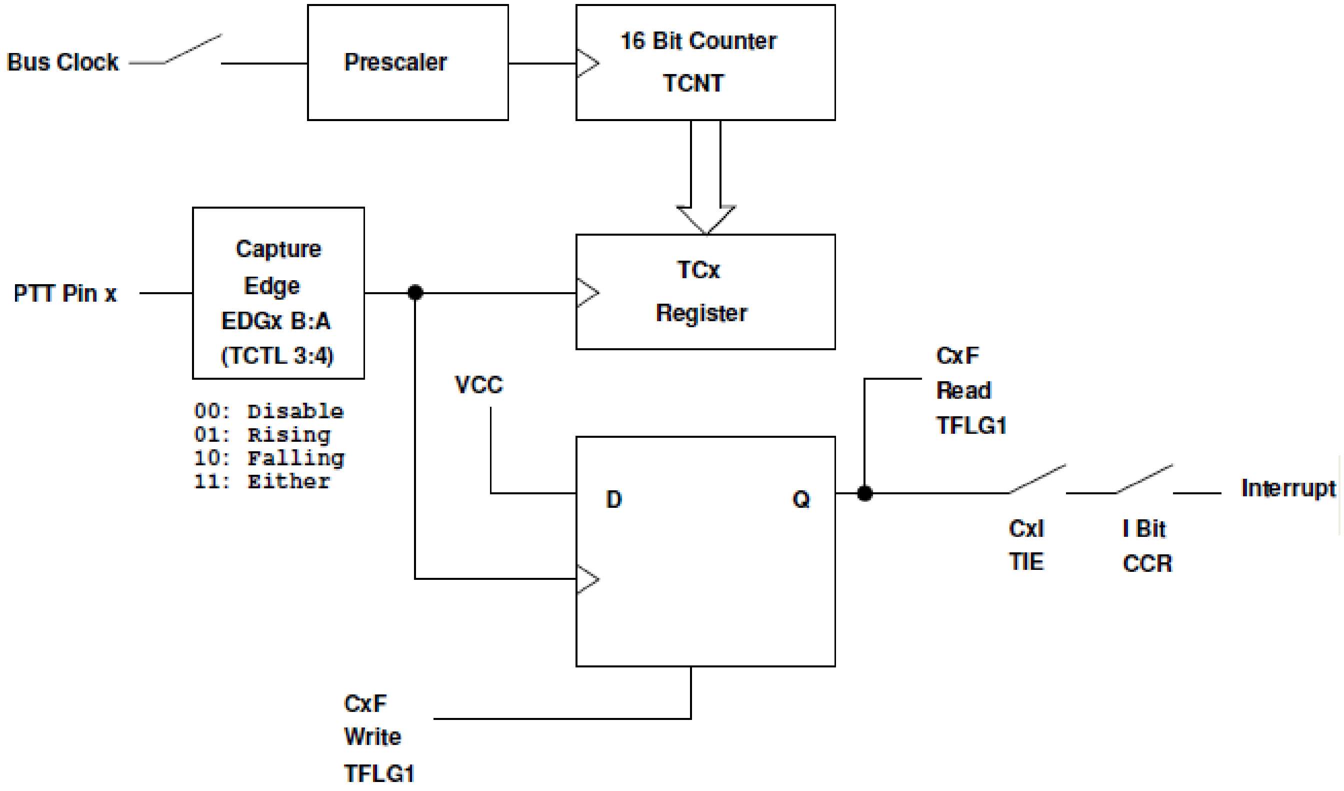
RTR 3:0	RTR 6:4							
	000 (0)	001 (1)	010 (2)	011 (3)	100 (4)	101 (5)	110 (6)	111 (7)
0000 (0)	Off	0.128	0.256	0.512	1.024	2.048	4.096	8.192
0001 (1)	Off	0.256	0.512	1.204	2.048	4.096	8.192	16.384
0010 (2)	Off	0.384	0.768	1.536	3.072	6.144	12.288	24.576
0011 (3)	Off	0.512	1.024	2.048	4.096	8.192	16.384	32.768
0100 (4)	Off	0.640	1.280	2.560	5.120	10.240	20.480	40.960
0101 (5)	Off	0.768	1.536	3.072	6.144	12.288	24.570	49.152
0110 (6)	Off	0.896	1.792	3.584	7.168	14.336	28.672	57.344
0111 (7)	Off	1.024	2.048	4.096	8.192	16.384	32.768	65.536
1000 (8)	Off	1.152	2.304	4.608	9.216	18.432	36.864	73.728
1001 (9)	Off	1.280	2.560	5.120	10.240	20.480	40.960	81.920
1010 (A)	Off	1.408	2.816	5.632	11.264	22.528	45.056	90.112
1011 (B)	Off	1.536	3.072	6.144	12.288	24.576	49.152	98.304
1100 (C)	Off	1.664	3.328	6.656	13.312	26.624	53.248	106.496
1101 (D)	Off	1.729	3.584	7.168	14.336	28.672	57.344	114.688
1110 (E)	Off	1.920	3.840	7.680	15.360	30.720	61.440	122.880
1111 (F)	Off	2.048	4.096	8.192	16.384	32.768	65.536	131.072

Code needed to use the real time interrupt

```
...
asm(sei);               /* Disable interrupts
RTICTL = 0x63;          /* Set rate to 16.384 ms */
CRGINT = 0x80;          /* Enable RTI interrupts */
CRGFLG = 0x80;          /* Clear RTI Flag */
UserRTI = (unsigned short) &rti_isr;
asm(cli);               /* Enable interrupts */

while (1)
{
        _ _asm(wai);        /* Do nothing -- wait for interrupt */
}
```

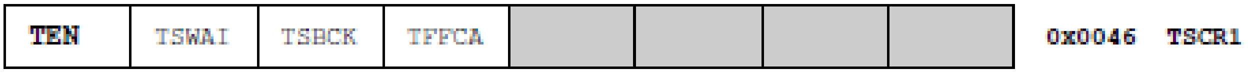
```
interrupt void rti_isr(void)
{
        ...
        CRGFLG = 0x80;
}
```

The MC9S12 Input Capture Function



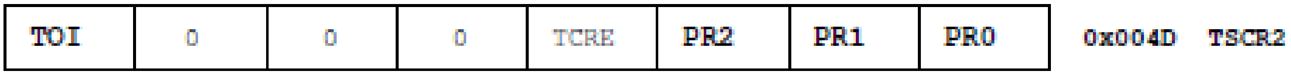
Registers used to enable Input Capture Function

Write a 1 to Bit 7 of TSCR1 to turn on timer

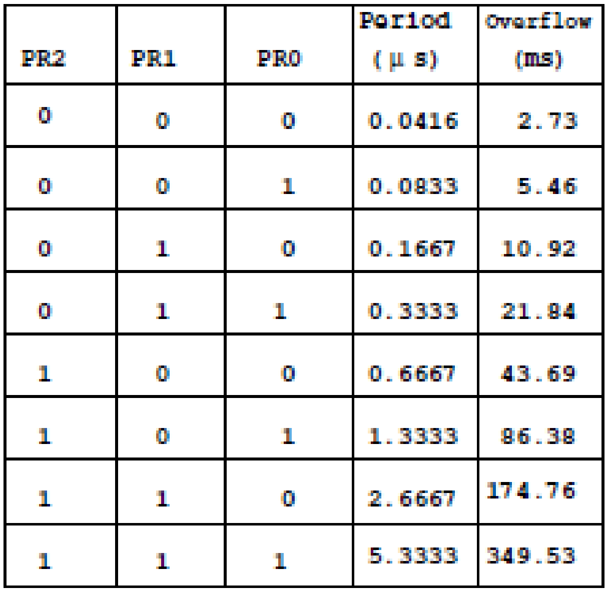


TEN	TSWAI	TSBCK	TFFCA					0x0046 TSCR1

Set the prescaler in TSCR2

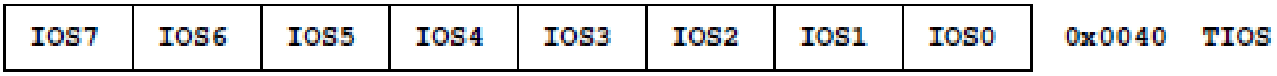


TOI	0	0	0	TCRE	PR2	PR1	PR0	0x004D TSCR2



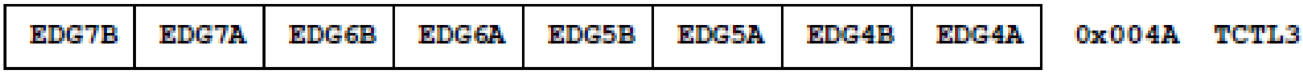
PR2	PR1	PR0	Period (μ s)	Overflow (ms)
0	0	0	0.0416	2.73
0	0	1	0.0833	5.46
0	1	0	0.1667	10.92
0	1	1	0.3333	21.84
1	0	0	0.6667	43.69
1	0	1	1.3333	86.38
1	1	0	2.6667	174.76
1	1	1	5.3333	349.53

Write a 0 to the bits of TIOS to make those pins input capture

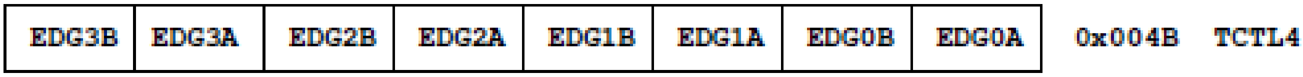


IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0	0x0040 TIOS

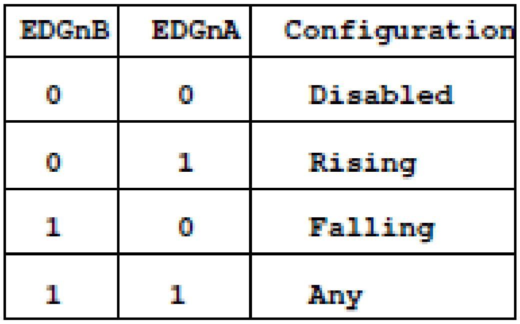
Write to TCTL3 and TCTL4 to choose edge(s) to capture



EDG7B	EDG7A	EDG6B	EDG6A	EDG5B	EDG5A	EDG4B	EDG4A	0x004A	TCTL3

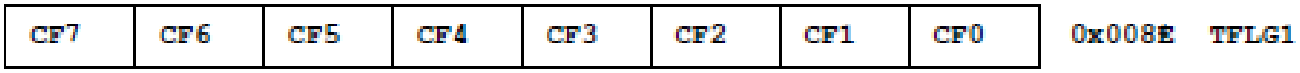


EDG3B	EDG3A	EDG2B	EDG2A	EDG1B	EDG1A	EDG0B	EDG0A	0x004B	TCTL4



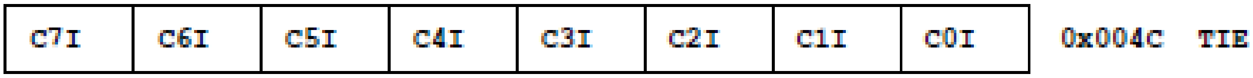
EDGnB	EDGnA	Configuration
0	0	Disabled
0	1	Rising
1	0	Falling
1	1	Any

To clear the flag, write a 1 to the bit you want to clear



CF7	CF6	CF5	CF4	CF3	CF2	CF1	CF0	0x008E	TFLG1

To enable interrupt when specified edge occurs, set corresponding bit in TIE register



C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I	0x004C	TIE

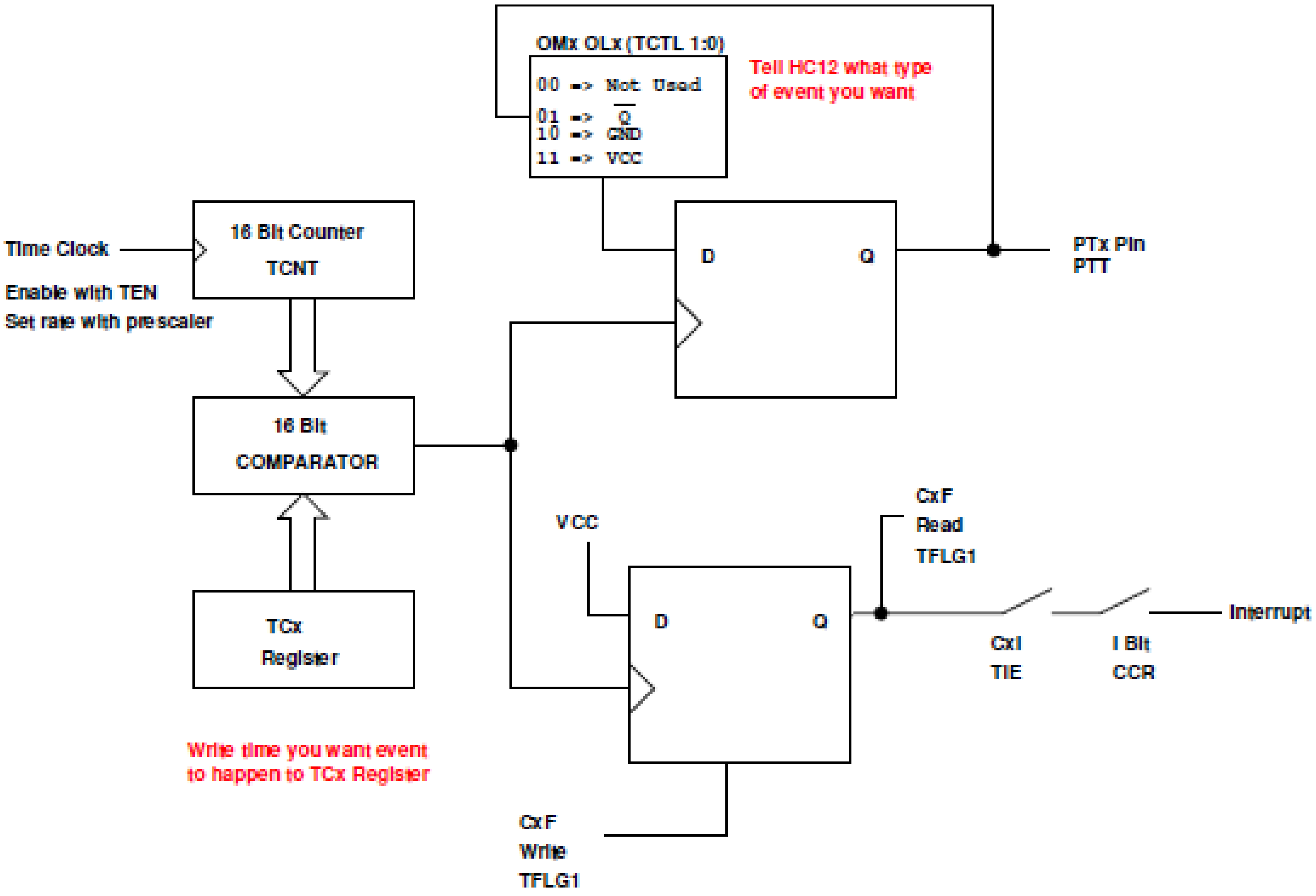
To determine time of specified edge, read 16−bit result registers TC0 thru TC7

Consider to wait until an event occurs on Pin 2 of PTT:

```
...
asm(sei);
done = FALSE;

/* Turn on timer subsystem */
TSCR1 = 0x80;

/* Set prescaler to 32 (87.38 ms), no TOF interrupt */
TSCR2 = 0x05;

/* Setup for IC1 */
TIOS = TIOS & ~0x02;                  /* Configure PT1 as IC */
TCTL4 = (TCTL4 | 0x04) & ~0x08;      /* Capture Rising Edge */
TFLG1 = 0x02;                         /* Clear IC1 Flag */
UserTimerCh1 = (short) &tic1_isr;
TIE = TIE | 0x02;                     /* Enable IC1 Interrupt */

/* Setup for IC2 */
TIOS = TIOS & ~0x04;                  /* Configure PT2 as IC */
TCTL4 = (TCTL4 | 0x10) & ~0x20;      /* Capture Rising Edge */
TFLG1 = 0x04;                         /* Clear IC2 Flag */
UserTimerCh2 = (short) &tic2_isr;
TIE = TIE | 0x04;                     /* Enable IC2 Interrupt */

/* Enable interrupts by clearing I bit of CCR */
asm(cli);
while (!done)
{
        asm(wai);                     /* Low power mode while waiting */
}
time = second - first;                /* Calculate total time */
...
```

```
interrupt void tic1_isr(void)
{
        first = TC1;
        TFLG1 = 0x02;
}

interrupt void tic2_isr(void)
{
        second = TC2;
        done = TRUE;
        TFLG1 = 0x04;
}
```

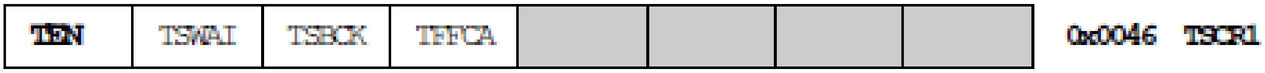
The MC9S12 Output Compare Function



Tell HC12 what type of event you want

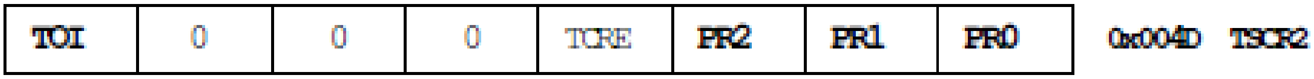
Write time you want event to happen to TCx Register

Write a 1 to Bit 7 of TSCR1 to turn on timer

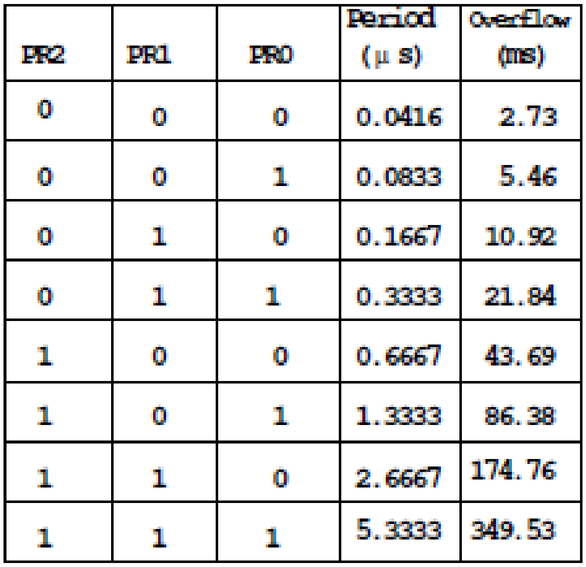


TEN	TSWAI	TSBCK	TFFCA					0x0046 TSCR1

Set the prescaler in TSCR2

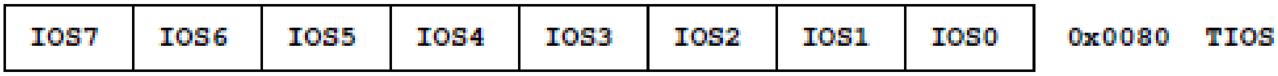


TOI	0	0	0	TCRE	PR2	PR1	PR0	0x004D TSCR2



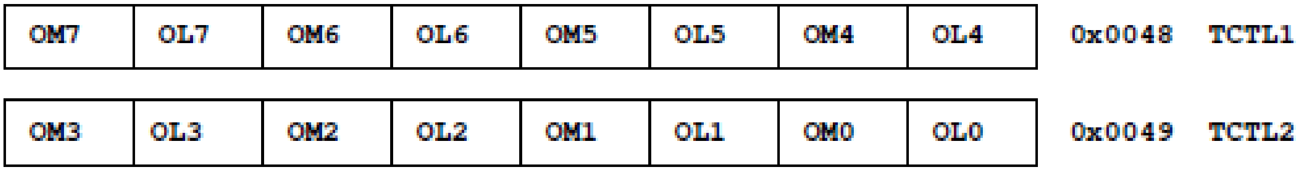
PR2	PR1	PR0	Period (μ s)	Overflow (ms)
0	0	0	0.0416	2.73
0	0	1	0.0833	5.46
0	1	0	0.1667	10.92
0	1	1	0.3333	21.84
1	0	0	0.6667	43.69
1	0	1	1.3333	86.38
1	1	0	2.6667	174.76
1	1	1	5.3333	349.53

Write a 1 to the bits of TIOS to make those pins output compare

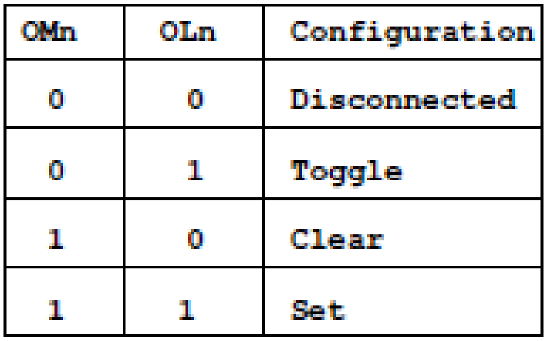


IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0	0x0080 TIOS

Write to TCTL1 and TCTL2 to choose action to take



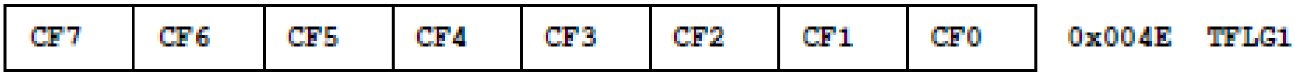
OM7	OL7	OM6	OL6	OM5	OL5	OM4	OL4	0x0048 TCTL1
OM3	OL3	OM2	OL2	OM1	OL1	OM0	OL0	0x0049 TCTL2



OMn	OLn	Configuration
0	0	Disconnected
0	1	Toggle
1	0	Clear
1	1	Set

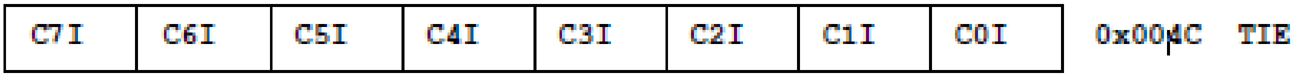
Write time you want event to occur to TCn register. To have next event occur T cycles after last event, add T to TCn.

To clear the flag, write a 1 to the bit you want to clear (0 to all others)



CF7	CF6	CF5	CF4	CF3	CF2	CF1	CF0	0x004E	TFLG1

To enable interrupt when compare occurs, set corresponding bit in TIE register



C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I	0x004C	TIE

Program to implement a 100 Hz square wave on output pin PT2: Make output toggle ever 5 ms, for a 10 ms period: 5 ms * 24 × 10^6 cycles/s = 120, 000 cycles. TCNT can only count up to 65,536 cycles. Need to use prescaler to get correct frequency. A prescaler of 4 divides the clock by 16, so: 5 ms * (24 × 10^6 cycles/s / 16) = 7, 500 cycles.

```
...
#define PERIOD 15000
#define HALF_PERIOD (PERIOD/2)
...

asm(sei);                          /* disable interrupts */
TSCR1 = 0x80;                      /* Turn on timer subsystem */
TSCR2 = 0x04;                      /* Set prescaler to 0.666 us */
TIOS = TIOS | 0x04;                /* PT2 as Output  Compare */
TCTL2 = (TCTL2 | 0x10) & ~0x20;    /* to toggle on compare */
TFLG1 = 0x04;                      /* Clear Channel 2 flag */
```

```
UserTimerCh2 = (unsigned short) &toc2_isr;
TIE = TIE | 0x04;                    /* Enable interrupt on Channel 2*/
asm(cli);                            /* global enable */

while (1)
{
        _ _asm(wai);
}
...


interrupt void toc2_isr(void)
{
        TC2 = TC2 + HALF_PERIOD;
        TFLG1 = 0x04;
}
```

Pulse Width Modulation on the MC9S12

Because PWM is used so often the MC9S12 has a built-in PWM system.

The MC9S12 PWM does not use interrupts.

The PWM system on the MC9S12 is very flexible.

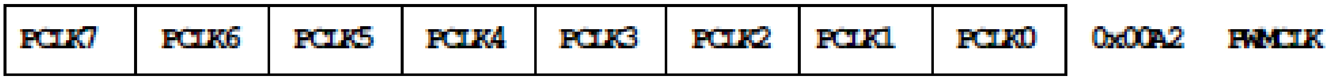
To enable the PWM output on one of the pins of Port P, write a 1 to the appropriate bit of PWME

PWME7	PWME6	PWME5	PWME4	PWME3	PWME2	PWME1	PWME0	0x00A0	PWME

PWMPOLn − Choose polarity 1 => high polarity 0 => low polarity
We will use high polarity only. PWMPOL = 0xFF;

PPOL7	PPOL6	PPOL5	PPOL4	PPOL3	PPOL2	PPOL1	PPOL0	0x00A1	PWMPOL

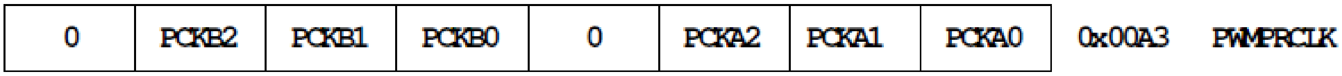
PWMCLKn − Choose clock source for Channel n



PCLK7	PCLK6	PCLK5	PCLK4	PCLK3	PCLK2	PCLK1	PCLK0	0x00A2	PWMCLK

CH5, CH4, CH1, CH0 can use either A (0) or SA (1)
CH7, CH6, CH3, CH2 can use either B (0) or SB (1)

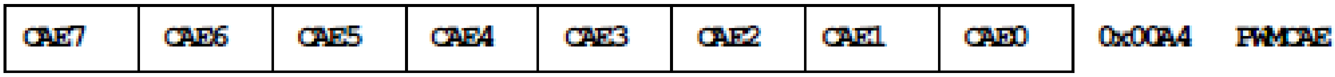
$$SB = \frac{B}{2 \times PWMSCLB} \qquad SA = \frac{A}{2 \times PWMSCLA}$$



0	PCKB2	PCKB1	PCKB0	0	PCKA2	PCKA1	PCKA0	0x00A3	PWMPRCLK

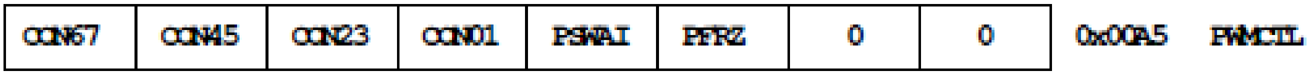
This register selects the prescale clock source for clocks A and B independently

Select center aligned outputs (1) or left aligned outputs (0)



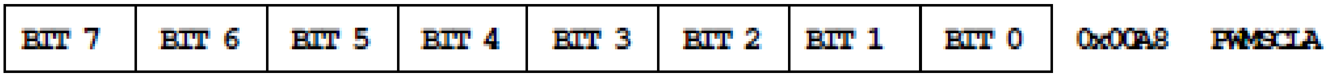
CAE7	CAE6	CAE5	CAE4	CAE3	CAE2	CAE1	CAE0	0x00A4	PWMCAE

Choose PWMCTL = 0x00 to choose 8−bit mode



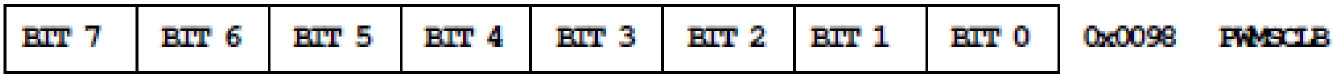
CON67	CON45	CON23	CON01	PSWAI	PFRZ	0	0	0x00A5	PWMCTL

PWMSCLA adjusts frequency of Clock SA



BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	0x00A8	PWMSCLA

PWMSCLB adjusts frequency of Clock SB



BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	0x0098	PWMSCLB

How to set the Period for PWM Channel 0

To get a 0.5 ms PWM period, you need 12,000 cycles of the 24 MHz clock.

$$12,000 = \begin{cases} \text{PWMPER0} \times 2^{\text{PCKA}} & \textbf{if PCLK0 == 0} \\ \text{PWMPER0} \times 2^{\text{PCKA+1}} \times \text{PWMSCLA} & \textbf{if PCLK0 == 1} \end{cases}$$

Code to set up PWM Channel 0 for 0.5 ms period:
2 kHz frequency, PWM with 60% duty cycle

```
PWMCTL = 0x00;                          /* 8-bit Mode */
PWMPOL = 0xFF;                          /* High polarity mode */
PWMCAE = 0x00;                          /* Left-Aligned */

PWMPRCLK = PWMPRCLK & ~0x07;        /* PCKA = 0 */
PWMCLK = PWMCLK | 0x01;             /* PCLK0 = 1 */
PWMSCLA = 30;
PWMPER0 = 200;
PWME = PWME | 0x01;     /* Enable PWM Channel 0 */
PWMDTY0 = 120;          /* 60% duty cycle on Channel  0 */
```

MC9S12 Analog/Digital Converter

A 10-bit A/D converter is used to convert an input voltage. The reference voltages are V_{RL} = 0V and V_{RH} = 5V.

– What is the quantization level of the A/D converter?

$\Delta V = (V_{RH} - V_{RL})/(2^b - 1) = 4.88$ mV

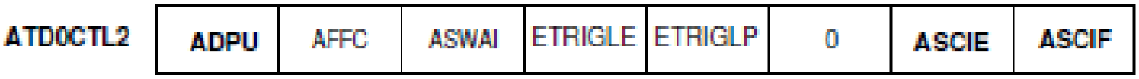
If the value read from the A/D converter is 0x15A, what is the input voltage?

Vin = $V_{RL} + [(V_{RH} - V_{RL})/(2^b-1)]$*ADvalue = 0 V + 4.88 mV × 346 = 1.6894 V

To program the HCS12 A/D converter you need to set up the A/D control registers **ATD0CTL2, ATD0CTL3, ATD0CTL4 and ATD0CTL5.**

The registers needed to use the A/D are:

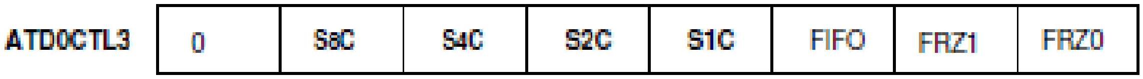
To Power up A/D Converter (ADPU = 1 in ATD0CTL2)



ATD0CTL2	ADPU	AFFC	ASWAI	ETRIGLE	ETRIGLP	0	ASCIE	ASCIF

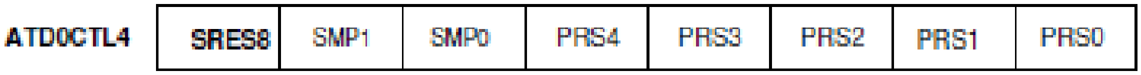
To Select number of conversions per sequence (S8C S4C S2C S1C in ATD0CTL3)

S8C S4C S2C S1C = 0001 to 0111 for 1 to 7 conversions
S8C S4C S2C S1C = 0000 or 1xxx for 8 conversions



ATD0CTL3	0	S8C	S4C	S2C	S1C	FIFO	FRZ1	FRZ0

To select 8-bit mode write 0x85 to ATD0CTL4
To select 10-bit mode write 0x05 to ATD0CTL4
*Other values of ATD0CTL4 either will not work or will result in slower A/D conversion rates



ATD0CTL4	SRES8	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0

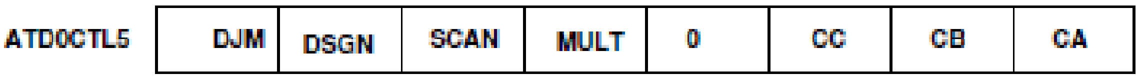
To select:
DJM = 0 => Left justified data in the result registers
DJM = 1 => Right justified data in the result registers

DSGN = 0 => Unsigned data representation in the result register
DSGN = 1 => Signed data representation in the result register

SCAN = 0: Convert one sequence, then stop
SCAN = 1: Convert continuously

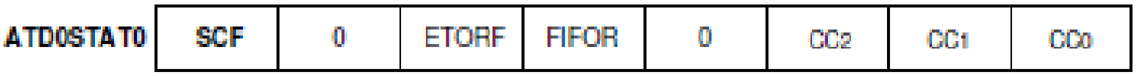
MULT = 0: Convert one channel the specified number of times
- Choose channel to convert with CC, CB, CA.

MULT = 1: Convert across several channels.
- First channel to be converted is CC, CB, CA



ATD0CTL5	DJM	DSGN	SCAN	MULT	0	CC	CB	CA

After writing to ATD0CTL5, the A/D converter starts, and the SCF bit is cleared. After a sequence of conversions is completed, the SCF flag is set.



ATD0STAT0	SCF	0	ETORF	FIFOR	0	CC2	CC1	CC0

You can read the results in ATD0DRx.

To setup the A/D to do 8 conversions, in 8-bit mode, on Channel 4:

```
    ...
    ATD0CTL2 = 0x80; /* Power up A/D, no interrupts */
    ATD0CTL3 = 0x00; /* Do eight conversions */
    ATD0CTL4 = 0x85; /* 8-bit mode */
    ATD0CTL5 = 0xA4; /* 1 0 1 0 0 1 0 0
                        | | | |   \___/
                        | | | |     |
                        | | | |     \__ Bit 4 of Port AD
                        | | |  \____ MULT = 0 => one channel only
                        | |  \___ Scan = 1 => continuous conversion
                        | \_____DSGN = 0 => unsigned
                        \______ DJM = 1 => right justified
                    */
```