- **The Real Time Interrupt**
- Huang Section 6.6
- CRG Block User Guide
    - Exceptions on the 9S12
    - Using interrupts on the 9S12
    - The Real Time Interrupt on the 9S12

Using Interrupts on the 9S12

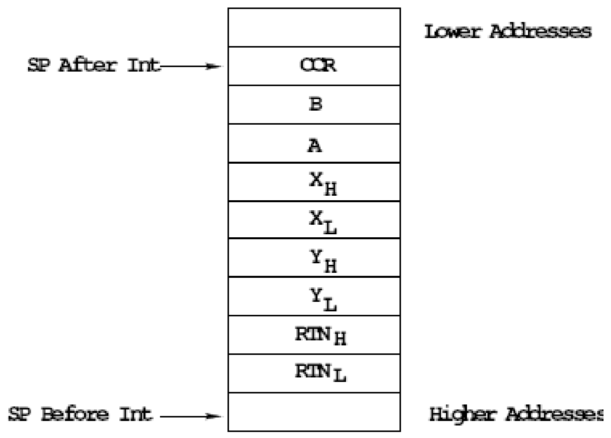
What happens when the 9S12 receives an unmasked interrupt?

1. Finish current instruction

2. Clear instruction queue

3. Calculate return address

4. Push Return Address, Y, X, A, B, CCR onto stack (SP is decremented by 9)



5. Set I bit of CCR

6. If XIRQ interrupt, set X bit of CCR

7. Load Program Counter from interrupt vector for highest priority interrupt which is pending

8. The following (from theMC9S12DG256 Device User Guide) shows the exception priorities. The Reset is the highest priority, the Clock Monitor Fail Reset the next highest, etc.

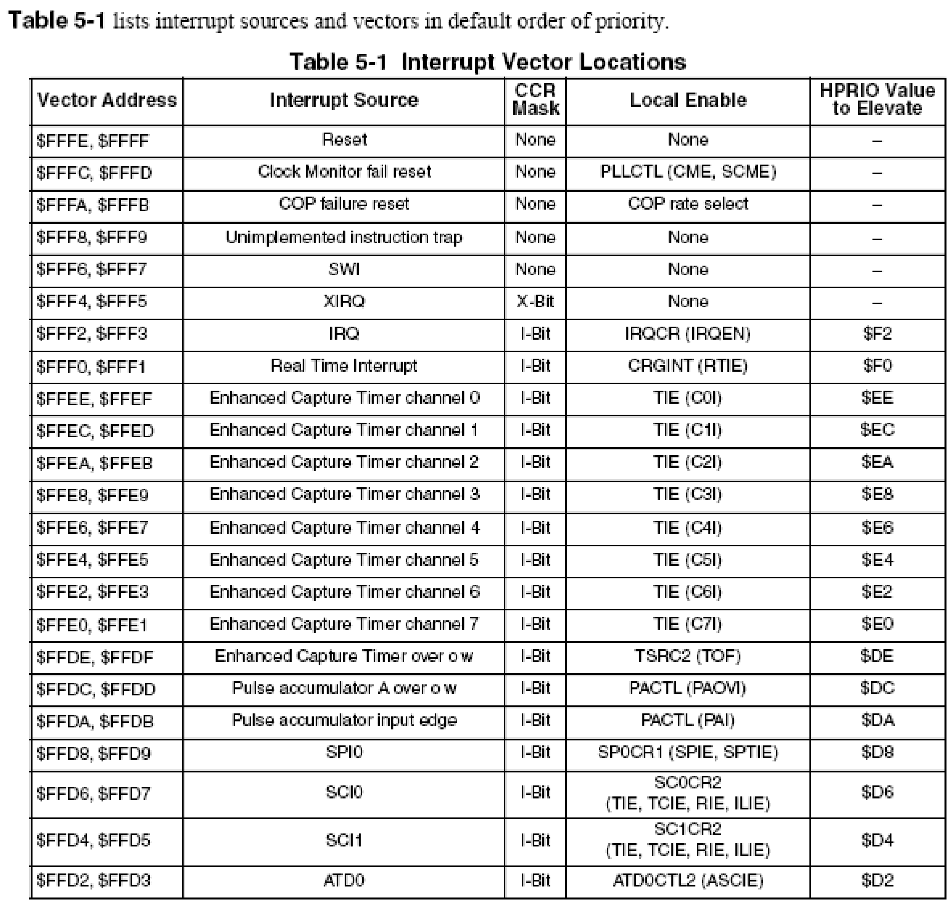


Table 5-1 lists interrupt sources and vectors in default order of priority.

Table 5-1 Interrupt Vector Locations

Vector Address	Interrupt Source	CCR Mask	Local Enable	HPRIO Value to Elevate
$FFFE, $FFFF	Reset	None	None	–
$FFFC, $FFFD	Clock Monitor fail reset	None	PLLCTL (CME, SCME)	–
$FFFA, $FFFB	COP failure reset	None	COP rate select	–
$FFF8, $FFF9	Unimplemented instruction trap	None	None	–
$FFF6, $FFF7	SWI	None	None	–
$FFF4, $FFF5	XIRQ	X-Bit	None	–
$FFF2, $FFF3	IRQ	I-Bit	IRQCR (IRQEN)	$F2
$FFF0, $FFF1	Real Time Interrupt	I-Bit	CRGINT (RTIE)	$F0
$FFEE, $FFEF	Enhanced Capture Timer channel 0	I-Bit	TIE (C0I)	$EE
$FFEC, $FFED	Enhanced Capture Timer channel 1	I-Bit	TIE (C1I)	$EC
$FFEA, $FFEB	Enhanced Capture Timer channel 2	I-Bit	TIE (C2I)	$EA
$FFE8, $FFE9	Enhanced Capture Timer channel 3	I-Bit	TIE (C3I)	$E8
$FFE6, $FFE7	Enhanced Capture Timer channel 4	I-Bit	TIE (C4I)	$E6
$FFE4, $FFE5	Enhanced Capture Timer channel 5	I-Bit	TIE (C5I)	$E4
$FFE2, $FFE3	Enhanced Capture Timer channel 6	I-Bit	TIE (C6I)	$E2
$FFE0, $FFE1	Enhanced Capture Timer channel 7	I-Bit	TIE (C7I)	$E0
$FFDE, $FFDF	Enhanced Capture Timer over o w	I-Bit	TSRC2 (TOF)	$DE
$FFDC, $FFDD	Pulse accumulator A over o w	I-Bit	PACTL (PAOVI)	$DC
$FFDA, $FFDB	Pulse accumulator input edge	I-Bit	PACTL (PAI)	$DA
$FFD8, $FFD9	SPI0	I-Bit	SP0CR1 (SPIE, SPTIE)	$D8
$FFD6, $FFD7	SCI0	I-Bit	SC0CR2 (TIE, TCIE, RIE, ILIE)	$D6
$FFD4, $FFD5	SCI1	I-Bit	SC1CR2 (TIE, TCIE, RIE, ILIE)	$D4
$FFD2, $FFD3	ATD0	I-Bit	ATD0CTL2 (ASCIE)	$D2

$FFD0, $FFD1	ATD1	I-Bit	ATD1CTL2 (ASCIE)	$D0
$FFCE, $FFCF	Port J	I-Bit	PTJIF (PTJIE)	$CE
$FFCC, $FFCD	Port H	I-Bit	PTHIF(PTHIE)	$CC
$FFCA, $FFCB	Modulus Down Counter under o w	I-Bit	MCCTL(MCZI)	$CA
$FFC8, $FFC9	Pulse Accumulator B Over o w	I-Bit	PBCTL(PBOVI)	$C8
$FFC6, $FFC7	CRG PLL lock	I-Bit	CRGINT(LOCKIE)	$C6
$FFC4, $FFC5	CRG Self Clock Mode	I-Bit	CRGINT (SCMIE)	$C4
$FFC2, $FFC3	BDLC	I-Bit	DLCBCR1(IE)	$C2
$FFC0, $FFC1	IIC Bus	I-Bit	IBCR (IBIE)	$C0
$FFBE, $FFBF	SPI1	I-Bit	SP1CR1 (SPIE, SPTIE)	$BE
$FFBC, $FFBD	SPI2	I-Bit	SP2CR1 (SPIE, SPTIE)	$BC
$FFBA, $FFBB	EEPROM	I-Bit	ECNFG (CCIE, CBEIE)	$BA
$FFB8, $FFB9	FLASH	I-Bit	FCNFG (CCIE, CBEIE)	$B8
$FFB6, $FFB7	CAN0 wake-up	I-Bit	CAN0RIER (WUPIE)	$B6
$FFB4, $FFB5	CAN0 errors	I-Bit	CAN0RIER (CSCIE, OVRIE)	$B4
$FFB2, $FFB3	CAN0 receive	I-Bit	CAN0RIER (RXFIE)	$B2
$FFB0, $FFB1	CAN0 transmit	I-Bit	CAN0TIER (TXEIE2-TXEIE0)	$B0
$FFAE, $FFAF	CAN1 wake-up	I-Bit	CAN1RIER (WUPIE)	$AE
$FFAC, $FFAD	CAN1 errors	I-Bit	CAN1RIER (CSCIE, OVRIE)	$AC
$FFAA, $FFAB	CAN1 receive	I-Bit	CAN1RIER (RXFIE)	$AA
$FFA8, $FFA9	CAN1 transmit	I-Bit	CAN1TIER (TXEIE2-TXEIE0)	$A8
$FFA6, $FFA7	Reserved			
$FFA4, $FFA5				
$FFA2, $FFA3				
$FFA0, $FFA1				
$FF9E, $FF9F				
$FF9C, $FF9D				
$FF9A, $FF9B				
$FF98, $FF99				
$FF96, $FF97	CAN4 wake-up	I-Bit	CAN4RIER (WUPIE)	$96
$FF94, $FF95	CAN4 errors	I-Bit	CAN4RIER (CSCIE, OVRIE)	$94
$FF92, $FF93	CAN4 receive	I-Bit	CAN4RIER (RXFIE)	$92
$FF90, $FF91	CAN4 transmit	I-Bit	CAN4TIER (TXEIE2-TXEIE0)	$90
$FF8E, $FF8F	Port P Interrupt	I-Bit	PTPIF (PTPIE)	$8E
$FF8C, $FF8D	PWM Emergency Shutdown	I-Bit	PWMSDN (PWMIE)	$8C
$FF80 to $FF8B	Reserved			

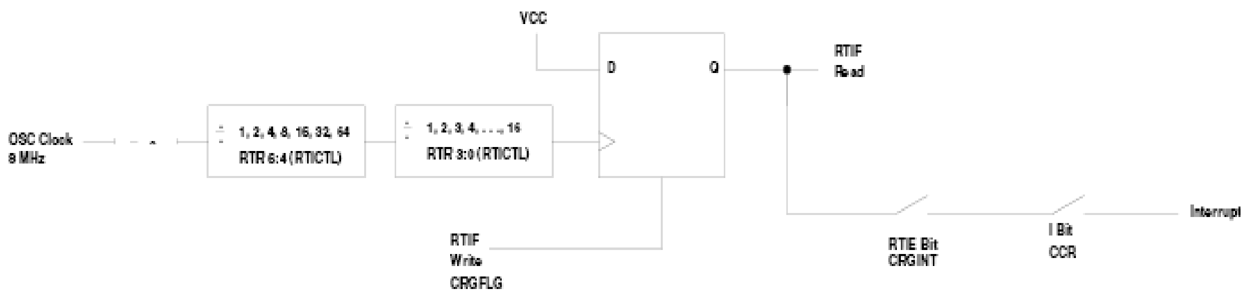
The Real Time Interrupt

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

• Information on the Real Time Interrupt is in the **CRG Block User Guide.**

• There are **two clock sources** for 9S12 hardware.
– Some hardware uses the **Oscillator Clock**. The **RTI** system uses this clock.
  * For our 9S12, the oscillator clock is **8 MHz**.
– Some hardware uses the **Bus Clock**. The Timer system (including the **Timer Overflow Interrupt**) use this clock.

- For our 9S12, the bus clock is **24 MHz**.



• The specific **interrupt mask for the Real Time Interrupt** is the **RTIE bit** of the **CRGINT** register.

• When the **Real Time Interrupt occurs**, the **RTIF bit** of the **CRGFLG** register **is set**.
  – To clear the Real Time Interrupt write a 1 to the RTIF bit of the CRGFLG register.

• The interrupt rate is set by the **RTR 6:4** and **RTR 2:0** bits of the **RTICTL** register. The RTR 6:4 bits are the Prescale Rate Select bits for the RTI, and the RTR 2:0 bits are the Modulus Counter Select bits to provide additional graunularity.

	7	6	5	4	3	2	1	0
R	RTIE	0	0	LOCKIE	0	0	SCMIE	0
W		▨	▨		▨	▨		▨
RESET:	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 3-5 CRG Interrupt Enable Register (CRGINT)

Read: anytime

Write: anytime

RTIE — Real Time Interrupt Enable Bit.
1 = Interrupt will be requested whenever RTIF is set.
0 = Interrupt requests from RTI are disabled.

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

▨ = Unimplemented or Reserved

NOTES:
1. PORF is set to 1 when a power on reset occurs. Unaffected by non-POR resets.

Figure 3-4 CRG Flags Register (CRGFLG)

Read: anytime

Write: refer to each bit for individual write conditions

RTIF — Real Time Interrupt Flag

RTIF is set to 1 at the end of the RTI period. This flag can only be cleared by writing a 1. Writing a 0 has no effect. If enabled (RTIE=1), RTIF causes an interrupt request.
1 = RTI time-out has occurred.
0 = RTI time-out has not yet occurred.

	7	6	5	4	3	2	1	0
R	0	RTR6	RTR5	RTR4	RTR3	RTR2	RTR1	RTR0
W	▨							
RESET:	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 3-8 CRG RTI Control Register (RTICTL)

Read: anytime

Write: anytime

NOTE: *A write to this register initializes the RTI counter.*

RTR[6:4] — Real Time Interrupt Prescale Rate Select Bits

These bits select the prescale rate for the RTI. See **Table 3-2**.

RTR[3:0] — Real Time Interrupt Modulus Counter Select Bits

These bits select the modulus counter target value to provide additional granularity.**Table 3-2** shows all possible divide values selectable by the RTICTL register. The source clock for the RTI is OSCCLK.

Table 3-2 RTI Frequency Divide Rates

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*	$2x2^{10}$	$2x2^{11}$	$2x2^{12}$	$2x2^{13}$	$2x2^{14}$	$2x2^{15}$	$2x2^{16}$
0010 (÷3)	OFF*	$3x2^{10}$	$3x2^{11}$	$3x2^{12}$	$3x2^{13}$	$3x2^{14}$	$3x2^{15}$	$3x2^{16}$
0011 (÷4)	OFF*	$4x2^{10}$	$4x2^{11}$	$4x2^{12}$	$4x2^{13}$	$4x2^{14}$	$4x2^{15}$	$4x2^{16}$
0100 (÷5)	OFF*	$5x2^{10}$	$5x2^{11}$	$5x2^{12}$	$5x2^{13}$	$5x2^{14}$	$5x2^{15}$	$5x2^{16}$
0101 (÷6)	OFF*	$6x2^{10}$	$6x2^{11}$	$6x2^{12}$	$6x2^{13}$	$6x2^{14}$	$6x2^{15}$	$6x2^{16}$
0110 (÷7)	OFF*	$7x2^{10}$	$7x2^{11}$	$7x2^{12}$	$7x2^{13}$	$7x2^{14}$	$7x2^{15}$	$7x2^{16}$
0111 (÷8)	OFF*	$8x2^{10}$	$8x2^{11}$	$8x2^{12}$	$8x2^{13}$	$8x2^{14}$	$8x2^{15}$	$8x2^{16}$
1000 (÷9)	OFF*	$9x2^{10}$	$9x2^{11}$	$9x2^{12}$	$9x2^{13}$	$9x2^{14}$	$9x2^{15}$	$9x2^{16}$
1001 (÷10)	OFF*	$10x2^{10}$	$10x2^{11}$	$10x2^{12}$	$10x2^{13}$	$10x2^{14}$	$10x2^{15}$	$10x2^{16}$

• To use the Real Time Interrupt, set the rate by writing to the **RTR 6:4** and the **RTR 3:0** bits of the **RTICTL**, and enable the interrupt by setting the **RTIE bit** of the **CRGINT** register.

– In the Real Time Interrupt ISR, you need to clear the RTIF flag by writing a 1 to the RTIF bit of the CRGFLG register.

• The following table shows all possible values, in ms, selectable by the RTICTL register (assuming the system uses 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

• Here is a C program which uses the Real Time Interrupt:

```
#include "hcs12.h"
#include "vectors12.h"
#include "DBug12.h"
#define enable() asm(" cli")
void INTERRUPT rti_isr(void);
main()
{
      DDRA = 0xff;
      PORTA = 0;
      RTICTL = 0x63; /* Set rate to 16.384 ms */
      CRGINT = 0x80; /* Enable RTI interrupts */
      CRGFLG = 0x80; /* Clear RTI Flag */
      UserRTI = (unsigned short) &rti_isr;
      enable();
      while (1)
      {
            asm(" wai"); /* Do nothing -- wait for interrupt */
      }
}

void INTERRUPT rti_isr(void)
{
     PORTA = PORTA + 1;
     CRGFLG = 0x80;
}
```

• Note that in the above program, the do-nothing loop has the instruction

asm("wai"); /* Do nothing -- wait for interrupt */

The assembly-language instruction WAI (Wait for Interrrupt) stacks the registers and puts the 9S12 into a low-power mode until an interrupt occurs.

• This allows the 9S12 to get into the ISR more quickly (because the time needed for pushing the registers on the stack has already been done), and lowers the power consumption of the 9S12 (because it doesn't have to execute a continuous loop while waiting for the interrupt).

What happens when a 9S12 gets in unmasked interrupt:

1. Completes current instruction

2. Clears instruction queue

3. Calculates return address

4. Stacks return address and contents of CPU registers

5. Sets I bit of CCR

6. Sets X bit of CCR if an XIRQ interrupt is pending

7. Fetches interrupt vector for the highest-priority interrupt which is pending

8. Executes ISR at the location of the interrupt vector

What happens when a 9S12 exits an ISR with the RTI instruction:

1. If no other interrupt pending,
- (a) 9S12 recovers stacked registers
- (b) Execution resumes at the return address

2. If another interrupt pending
- (a) 9S12 stacks registers
- (b) Subtracts 9 from SP
- (c) Sets I bit of CCR
- (d) Sets X bit of CCR if an XIRQ interrupt is pending
- (e) Fetches interrupt vector for the highest-priority interrupt which is pending
- (f) Executes ISR at the location of the interrupt vector

Capturing the Time of an External Event

• One way to determine the time of an external event is to wait for the event to occur, the read the TCNT register:

• For example, to determine the time a signal on Bit 0 of PORTB changes from a high to a low:

```
while ((PORTB & 0x01) != 0) ;       /* Wait while Bit 0 high */
time = TCNT;                        /* Read time after goes low */
```

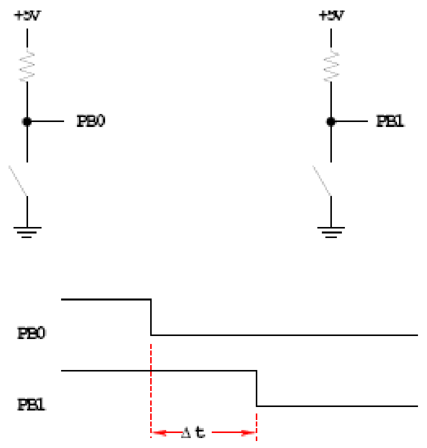
• Two problems with this:

1. Cannot do anything else while waiting
2. Do not get exact time because of delays in software

• To solve problems use hardware which latches TCNT when event occurs, and generates an interrupt.

• Such hardware is built into the 9S12 — called the Input Capture System

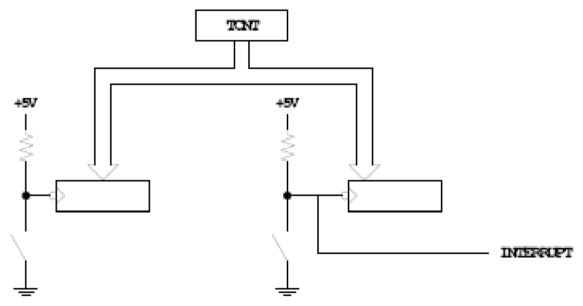
Measure the time between two events

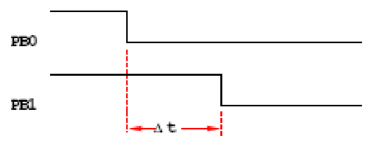


How to measure Δt?
Wait until signal goes low, then measure TCNT

```
while ((PORTB & 0x01) == 0x01) ;
start = TCNT;
while ((PORTB & 0x02) == 0x02) ;
end = TCNT;
dt = end - start;
```

Problems: 1) May not get very accurate time
2) Can't do anything while waiting for signal level to change.





Solution: Latch TCNT on falling edge of signal
Read latched values when interrupt occurs

The 9S12 Input Capture Function

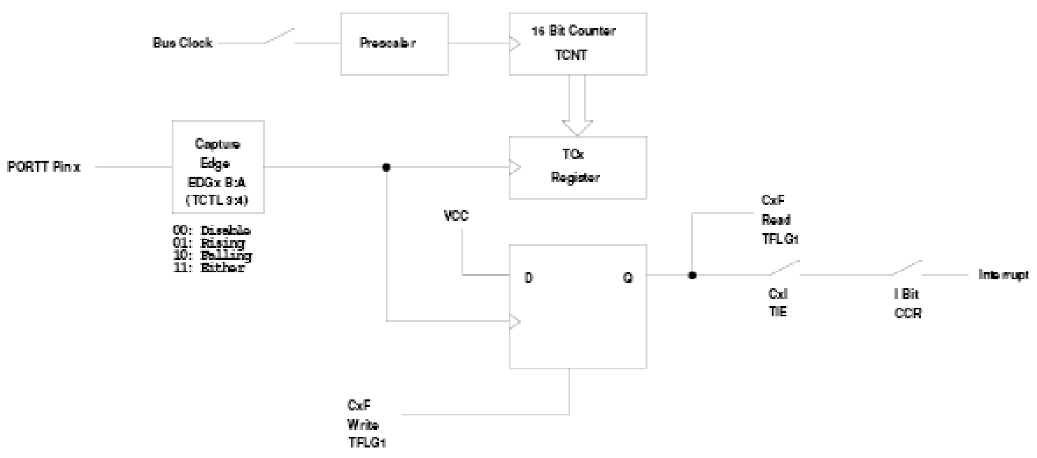
• The 9S12 allows you to **capture the time an external event** occurs on any of the eight **PORTT pins.**

• An external event is either a **rising** edge or a **falling** edge

• To use the Input Capture Function:
– Enable the timer subsystem (set **TEN** bit of **TSCR1**)
– Set the prescaler
– Tell the 9S12 that you want to use a particular pin of PORTT for input capture
– Tell the 9S12 which edge (**rising**, **falling**, or **either**) you want to capture
– Tell the 9S12 if you want an interrupt to be generated when the capture occurs

A Simplified Block Diagram of the 9S12 Input Capture Subsystem

Port T Pin x set up as Input Capture (IOSx = 0 in TIOS)



Registers used to enable Input Capture Function (see **ECT Block User Guide**)

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

	BIT7	6	5	4	3	2	1	BIT0
R	TEN	TSWAI	TSFRZ	TFFCA	PRNT	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)

Read or write anytime except PRNT bit is write once.All bits reset to zero.

TEN — Timer Enable

0 = Disables the main timer, including the counter. Can be used for reducing power consumption.
1 = Allows the timer to function normally.

To turn on the timer subsystem: TSCR1 = 0x80;

	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)

PR2, PR1, PR0 - Timer Prescaler Select

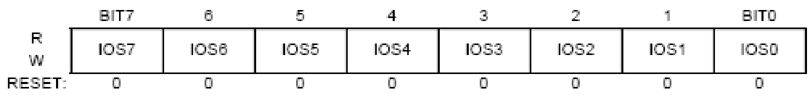
These three bits specify the division rate of the main Timer prescaler when the PRNT bit of register TSCR1 is set to "0". The newly selected prescale factor will not take effect until the next synchronized edge where all prescale counter stages equal zero.

Set the prescaler in TSCR2

Make sure the overflow time is greater than the time difference you want to measure

PR2	**PR1**	**PR0**	**Period (µs)**	**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

To have overflow rate of 21.84 ms:
TSCR2 = 0x03;



	BIT7	6	5	4	3	2	1	BIT0
R W	IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0
RESET:	0	0	0	0	0	0	0	0

Figure 3-1 Timer Input Capture/Output Compare Register (TIOS)

Read or write anytime.

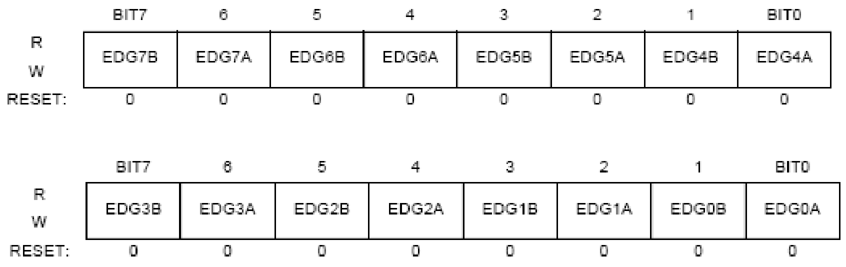
All bits reset to zero.

IOS[7:0] — Input Capture or Output Compare Channel Configuration

0 = The corresponding channel acts as an input capture.
1 = The corresponding channel acts as an output compare.

To make Pin 3 an input capture pin: TIOS = TIOS & ~0X08;

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



	BIT7	6	5	4	3	2	1	BIT0
R W	EDG7B	EDG7A	EDG6B	EDG6A	EDG5B	EDG5A	EDG4B	EDG4A
RESET:	0	0	0	0	0	0	0	0

	BIT7	6	5	4	3	2	1	BIT0
R W	EDG3B	EDG3A	EDG2B	EDG2A	EDG1B	EDG1A	EDG0B	EDG0A
RESET:	0	0	0	0	0	0	0	0

Figure 3-9 Timer Control Register 3/Timer Control Register 4 (TCTL3/TCTL4)

Read or write anytime.

All bits reset to zero.

EDGxB, EDGxA — Input Capture Edge Control

Table 3-3 Edge Detector Circuit Configuration

EDGxB	EDGxA	Configuration
0	0	Capture disabled
0	1	Capture on rising edges only
1	0	Capture on falling edges only
1	1	Capture on any edge (rising or falling)

To have Pin 3 capture a rising edge:
TCTL4 = (TCTL4 | 0x40) & ~0x80;

When specified edge occurs, the corresponding bit in TFLG1 will be set.
To clear the flag, write a 1 to the bit you want to clear (0 to all others)

	BIT7	6	5	4	3	2	1	BIT0
R W	C7F	C6F	C5F	C4F	C3F	C2F	C1F	C0F
RESET:	0	0	0	0	0	0	0	0

Figure 3-12 Main Timer Interrupt Flag 1 (TFLG1)

C7F–C0F — Input Capture/Output Compare Channel "x" Flag

A CxF flag is set when a corresponding input capture or output compare is detected. C0F can also be set by 16-bit Pulse Accumulator B (PACB). C3F-C0F can also be set by 8-bit pulse accumulators PAC3-PAC0.

If the delay counter is enabled, the CxF flag will not be set until after the delay.

To wait until rising edge on Pin 3:	while ((TFLG1 & 0x08) == 0);
To clear flag bit for Pin 3:	TFLG1 = 0x08;

	BIT7	6	5	4	3	2	1	BIT0
R W	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I
RESET:	0	0	0	0	0	0	0	0

Figure 3-10 Timer Interrupt Enable Register (TIE)

Read or write anytime.

All bits reset to zero.

The bits C7I-C0I correspond bit-for-bit with the flags in the TFLG1 status register.

C7I–C0I — Input Capture/Output Compare "x" Interrupt Enable
0 = The corresponding flag is disabled from causing a hardware interrupt.
1 = The corresponding flag is enabled to cause an interrupt.

To enable interrupt on Pin 3: TIE = TIE | 0x08;

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

To read time of edge on Pin 3:

```
unsigned int time;
time = TC3;
```