

not set.
----------

## Output Compare Software Checklist

Here are the steps to check when writing your software for timer output compares:

1. Initialize the interrupt vector(s) for the timer channel(s) if interrupts are to be used.
2. Set bit-7 (TEN) in TSCR1 to enable the timer.
3. Set bits in TIOS to enable the Output Compare channels.
4. Initialize the OMn and OLn bits in TCTL1 and TCTL2 if the timer output pins are to be activated.
5. Set the bits in OC7M and OC7D if any timer channel 7 override is to be done.
6. Load the current value for TCNT and add to it the required delay.
7. Store the (TCNT+delay) value into the TCn register to be used.
8. Reset the flag(s) in TFLG1.
9. Enable any required interrupts in TIE.
10. Unmask interrupts by clearing the I-bit in the condition code register.
11. Wait for the output compare to set the flag by either polling the flag or waiting for the interrupt.
12. After the output compare action occurs, re-initialize the TCn register with the new delay value by adding the delay to the current TCn value.
13. Reset the CnF flag bit to prepare for the next output compare.

## 13.5 Input Capture

<p><i>Input Capture</i> allows the TCNT register value to be latched when an external event occurs.</p>
---

The input capture hardware is shown in Figure 13-5. Again, the 16-bit free-running TCNT register is the heart of the system, and the same eight, 16-bit *Timer Input Capture/Output Compare* registers, *TC7 - TC0*, used for the output capture function, latch the value of the free-running counter in response to a program-selected, external signal coming from Port T. For example, the period of a pulse train can be found by capturing the TCNT at the start of the period, signified by a rising or falling edge, and storing it. The next rising or falling edge will capture the count at the end of the period. The difference in the two counts, taking into account timer overflows, will be the period in bus clock cycles. The length of the positive pulse can be measured by capturing the time at the rising edge and then again at the falling edge.

Two bits for each Input Capture channel, *EDGnB* and *EDGnA* in *Timer Control Registers 3* and *4*, control when the signal on Port T causes the capture event to occur. You may select rising, falling or both edges to be active.

The input capture interrupts operate just like output compare interrupts. The interrupt enable bits in TIE must be set. Then, when the flag in TFLG1 is set by the selected input capture edge, the interrupt request is forwarded to the CPU.

Example 13-21 shows a subroutine that measures the period of a wave form, so long as it is less than 8.192 ms. Input capture one is to be used; it is enabled in *line 25*, and a rising edge is selected in *lines 28* and *29*. The IC1 flag is reset in *lines 31* and *32*. The

program then waits until the first positive edge appears on Input Capture 1 in *line 35*. When this happens, the contents of the TCNT register are latched into the TC1 register and the program leaves the *line 35* spin loop. The first count is saved in a buffer in *line 38*, the IC1 flag reset, and the second spin loop (*line 44*) is entered. After the second rising edge, the duration of the pulse is calculated by subtracting the second TCNT value from the first. Note that modulo  $2^{16}$  arithmetic will return the correct number of clock pulses even if the second TCNT value is less than the first.

**TCTL3 – Base + \$004A – Timer Control Register 3**

**TCTL4 – Base + \$004B – Timer Control Register 4**

	Bit 7	6	5	4	3	2	1	0
TCTL3	EDG7B	EDG7A	EDG6B	EDG6A	EDG5B	EDG5A	EDG4B	EDG4A
Reset:	0	0	0	0	0	0	0	0
	Bit 7	6	5	4	3	2	1	0
TCTL4	EDG3B	EDG3A	EDG2B	EDG2A	EDG1B	EDG1A	EDG0B	EDG0A
Reset:	0	0	0	0	0	0	0	0

Read: Anytime. Write: Anytime.

**OMn:OLn**

**Input Capture Edge Control**

These pairs of bits configure the input capture edge selection.

**Table 13-5 Input Capture Edge Selection**

EDGnB	EDGnA	Edge to Capture TCNT
0	0	Capture disabled.
0	1	Capture on rising edges only.
1	0	Capture on falling edges only.
1	1	Capture on any edge (falling or rising).

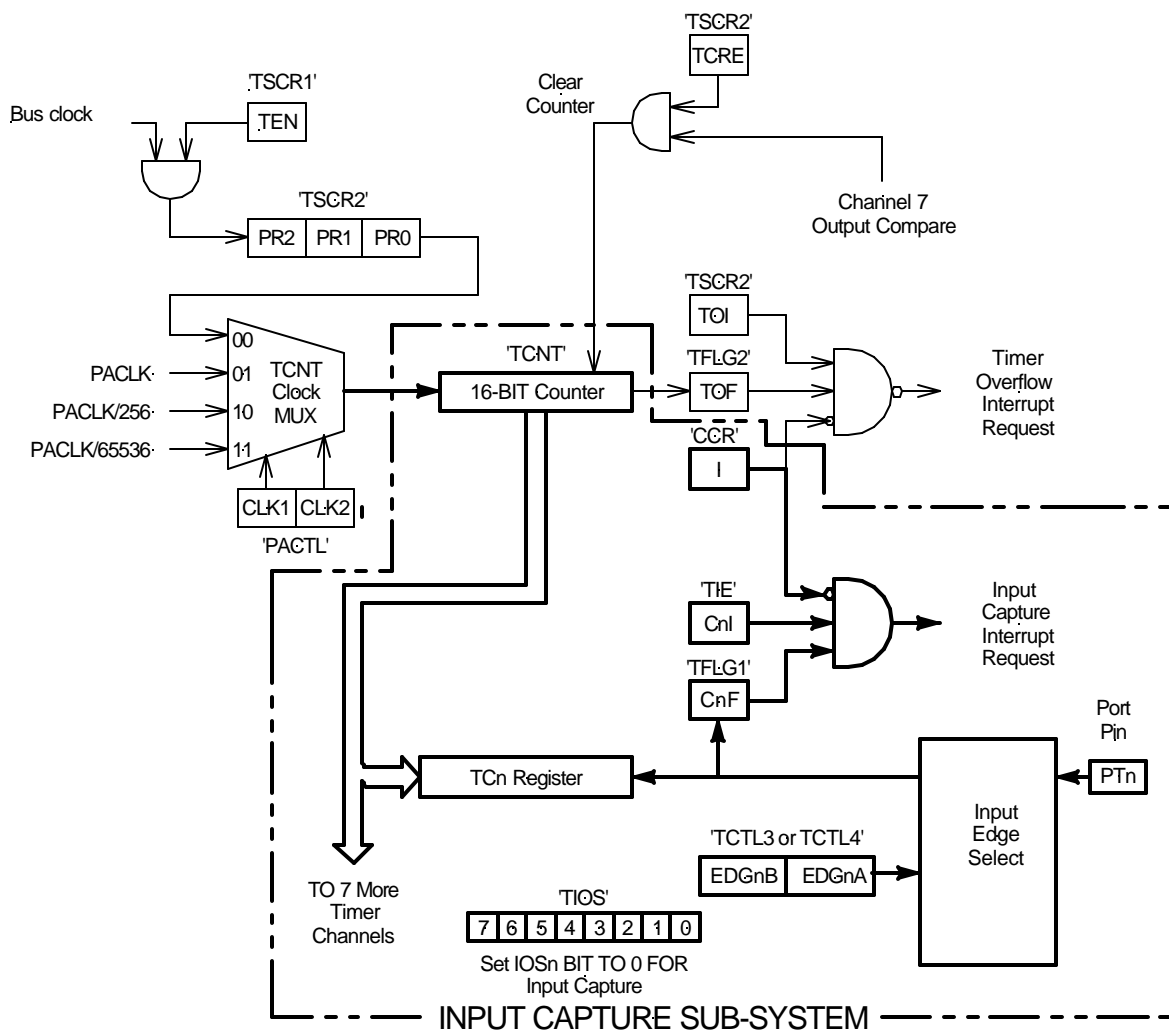


Figure 13-5 Input capture hardware.

**Example 13-21 A Subroutine to Measure the Period of a Waveform**

Write a C-callable function in assembly language that will measure the period of a waveform on input capture channel 2.

*Solution:*

Metrowerks HC12-Assembler  
 (c) COPYRIGHT METROWERKS 1987-2003

```

Rel.Loc   Obj. code Source line
-----
1          ;*****
2          ; Subroutine get_period
3          ;*****
4          ; C Callable Function
5          ; int get_period( void );
6          ;*****
7          ; Subroutine to measure the period of
8          ; a wave. It measures the time
9          ; in TCNT units between two successive
    
```

```

10             ; rising edges on Input Capture channel 1.
11             ; Limited to waveforms with frequency
12             ; higher than 122 Hz with an 8 MHz clock..
13             ; Input Parameters: None
14             ; Output Parameters: D register returns
15             ;     the period in clock cycles
16             ; Registers Modified: A, B, CCR
17             ; Stack Use: 2 bytes
18             ;*****
19             ; Define the entry point for the main program
20             XDEF     get_period
21             INCLUDE timer.inc ; Timer defs
22             ;*****
23 get_period:
24             ;*****
25             ; Enable the timer system
26000000 4C46 80             bset  TSCR1,TEN
27             ; Reset TIOS bit to enable input capture
28000003 4D40 02             bclr  TIOS,IOS1 ; Channel 1
29             ; Initialize IC1 for rising edge
30             ; EDG1B=0, EDG1A=1
31000006 4D4B 08             bclr  TCTL4,EDG1B
32000009 4C4B 04             bset  TCTL4,EDG1A
33             ; Reset C1F flag
3400000C 8602             ldaa  #C1F
3500000E 5A4E             staa  TFLG1
36             ; Wait for the next rising edge
37 spin1:
38000010 4F4E 02FC         brclr TFLG1,C1F,spin1
39             ; Now get the count that was latched
40000014 DC52             ldd  TC1
41000016 3B             pshd
42             ; Reset C1F flag
43000017 8602             ldaa  #C1F
44000019 5A4E             staa  TFLG1
45             ; Wait for the next rising edge
46 spin2:
4700001B 4F4E 02FC         brclr TFLG1,C1F,spin2
48             ; Get the ending count
4900001F DC52             ldd  TC1
50             ; Calculate the period
51000021 A380             subd  0,SP
52             ; Return with the value in D
53000023 1B82             leas  2,sp ; Adjust SP
54000025 3D             rts
55             ;*****

```

---

### Example 13-22

Show how the calculation in line 51 of Example 13-21 which calculates the number of TCNT clock cycles between two successive rising edges can return the correct answer if the second number captured in line 49 is less than the first number captured in line 40.

*Solution:*

Modulo  $2^{16}$  arithmetic is being used on the 16-bit number. To see how this works, take a simple example with, say, a 3-bit number. Let the first count be  $101_2$  and the second number  $100_2$ . For this to occur, 7 clock cycles must have elapsed. The 3-bit arithmetic,  $100_2 - 101_2 = 111_2$ .

---

## Input Capture Software Checklist

Here is a list of the steps to check when writing your software for input captures:

1. Initialize the interrupt vector(s) for the timer channel(s) if interrupts are to be used.
2. Set bit-7 (TEN) in TSCR to enable the timer.
3. Reset bits in TIOS to disable the Output Compare channels.
4. Initialize the EDGnB and EDGnA bits in TCTL3 and TCTL4 to select the active edge for the input capture trigger signal.
5. Reset the flag(s) in TFLG1.
6. Enable any required interrupts in TMSK1.
7. Unmask interrupts by clearing the I-bit in the condition code register.
8. Wait for the input capture to set the flag by either polling the flag or waiting for the interrupt.
9. After the input capture event occurs, use the data in the TCn register.
10. Reset the CnF flag bit to prepare for the next input capture.

### 13.6 Pulse Accumulator

The *pulse accumulator* can be used to count external events.

The pulse accumulator is a 16-bit counter that can operate as an *event counter*, counting external clock pulses, or a *gated time accumulator*. In gated time operation, the bus clock is divided by 64 and gated by the pulse accumulator input into the accumulator.

These operating modes are shown in Figure 13-6. Timer channel 7 (Port T-7) is used as the input for the pulse accumulator. You may select the edge (positive or negative) for event counting or the level (high or low) for gated time accumulation. There are three registers to be programmed when using the pulse accumulator. The *PACTL* register enables the system, selects the mode of operation, selects for the TCNT clock source (but not used in the pulse accumulator), enables a prescaler for the clock, and enables pulse accumulator interrupts. The *PAFLG* register contains the interrupt flags and the 16-bit *PACNT* register records the accumulated input pulses.

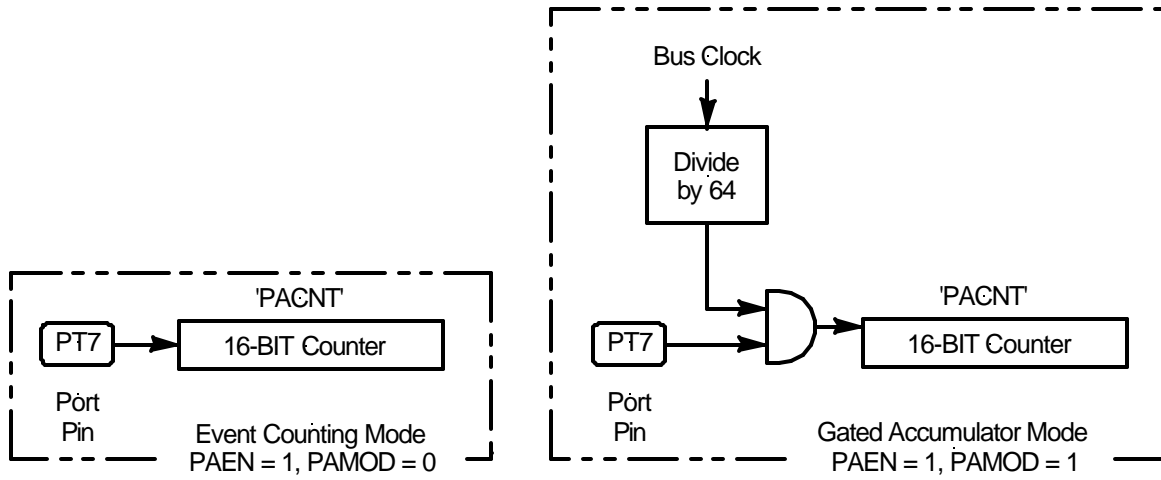


Figure 13-6 Pulse accumulator operating modes.

**PACTL – Base + \$0060 – 16-bit Pulse Accumulator Control Register**

	Bit 7	6	5	4	3	2	1	0
Read:	0	PAEN	PAMOD	PEDGE	CLK1	CLK0	PAOVI	PAI
Write:								
Reset:	0	0	0	0	0	0	0	0

0 = reserved, unimplemented or cannot be written to.

Read: Anytime. Write: Anytime.

**PAEN****Pulse Accumulator System Enable**

0 = Pulse accumulator system disabled (default).

1 = Enabled.

PAEN is independent of the timer system enable bit TEN; The pulse accumulator can function (if PAEN = 1) when the timer is disabled (TEN = 0).

**PAMOD****Pulse Accumulator Mode Select**

0 = Event counter mode (default).

1 = Gated time accumulation mode.

**PEDGE****Pulse Accumulator Edge Control**

For PAMOD = 0 (event counter mode).

0 = Falling edges on PT7 cause the count to be increased (default).

1 = Rising edges are counted.

For PAMOD = 1 (gated time accumulation mode).

0 = PT7 going high enables the bus clock/64 to pulse the accumulator. The subsequent falling edge of PT7 sets the *Pulse Accumulator Input Edge (PAIF)* flag.

1 = PT7 going low enables the clock to the accumulator. The subsequent rising edge of PT7 sets the PAIF flag.

(Note: If the timer is not enabled (TEN = 0), there is not divided by 64 clock since the this clock is generated by the timer prescaler.

**CLK1:CLK0****Clock Select Bits**

The clock select bits control a multiplexer which selects the clock used by the timer. (See **Error! Reference source not found.**). If the pulse accumulator is disabled (PAEN = 0), the prescaler clock is always used as an input clock to the timer counter. For more information, see Section 13.7.

**Table 13-6 Timer Clock Selection**

CLK1	CLK0	Timer Clock	Timer Clock Frequency
0	0	Use timer prescaler clock as timer counter clock	Bus clock/prescaler
0	1	Use PCLK as input to the timer counter clock.	Bus clock/64
1	0	Use PCLK/256 as timer counter clock.	Bus clock/16,384
1	1	Use PCLK/65536 as timer counter clock.	Bus clock/16,777,216

**PAOVI****Pulse Accumulator Overflow Interrupt Enable**

0 = Interrupt disabled (default).

1 = Interrupt requested if PAOVF is set.

**PAIF****Pulse Accumulator Input Interrupt Enable**

0 = Interrupt disabled (default).  
 1 = Interrupt requested if PAIF is set

**PACNT – Base + \$0062, \$0063 – 16-bit Pulse Accumulator Counter Register**

	Bit 15	14	13	12	11	10	9	8
	PACNT15	PACNT14	PACNT13	PACNT12	PACNT11	PACNT10	PACNT9	PACNT8
Reset:	0	0	0	0	0	0	0	0
	Bit 7	6	5	4	3	2	1	0
	PACNT7	PACNT6	PACNT5	PACNT4	PACNT3	PACNT2	PACNT1	PACNT0
Reset:	0	0	0	0	0	0	0	0

Read: Anytime. Write: Anytime

This 16-bit register holds the number of accumulated input pulses (active edges) since the last reset. You may write it at any time and you should use a 16-bit read to simultaneously read the high and low bytes. When PACNT overflows from \$FFFF to \$0000, the PAOVF overflow flag in the PAFLG register is set.

**Pulse Accumulator Interrupts**

The pulse accumulator can interrupt when the counter register overflows and/or when an input edge is detected.

The pulse accumulator interrupts operate like the other functions in the timer section. A flag is set by the hardware when the appropriate condition is true, and if an interrupt enable is set, the interrupt request is generated. There are two flags and two interrupts that can be generated; these are controlled by the *PAOVI* and *PAI* bits in the *PACTL* register. When the pulse accumulator overflows, the *PAOVF* flag bit in the *PAFLG* register is set. *PAOVI* enables this flag to generate an interrupt request. *PAI* is the *Pulse Accumulator Input Edge* interrupt enable. When the selected input edge occurs (chosen by *PEDGE* in *PACTL*), the *Pulse Accumulator Input Edge Flag (PAIF)* in the *PAFLG* is set. If *PAI* is set, an interrupt is generated. These interrupts are enabled by following the procedures outlined in the previous sections.

The pulse accumulator can interrupt the processor after a number of external events have occurred. Let us say that a sensor on a conveyor belt is detecting a product passing by and a crate is to be filled after twenty-four counts. Example 13-23 shows pseudocode to initialize the interrupt and to do the interrupt service routine.

**PAFLG – Base + \$0061 – Pulse Accumulator Flag Register**

	Bit 7	6	5	4	3	2	1	0
Read:	0	0	0	0	0	0	PAOVF	PAIF
Write:								
Reset:	0	0	0	0	0	0	0	0

= reserved, unimplemented or cannot be written to.  
 Read: Anytime. Write: Anytime.

**PAOVF**

---

**Pulse Accumulator Overflow Flag**  
 This bit is set when the 16-bit pulse accumulator overflows from \$FFFF to \$0000. The bit is cleared by writing a one to the bit (see Section 13.8).

**PAIF**

---

**Pulse Accumulator Input Edge Flag**  
 This bit is set when the selected edge (*PEDGE*) is detected at the *IOC7* input pin. In event mode

(PAMOD = 0) the event edge sets PAIF. In the gated time accumulation mode (PAMOD = 1) the trailing edge of the gate signal sets PAIF.  
This bit is cleared by writing a one to the bit (see Section 13.8).

## Pulse Accumulator Interrupt Vectors

**Table 13-7 Pulse Accumulator Interrupt Vectors**

Priority	Vector Address	Interrupt Source	Local Enable Bit	See Register	HPRIO Value to Promote
18	\$FFDA:FFDB	Pulse Accumulator Input Edge	PAI	PACTL	\$DA
17	\$FFDC:FFDD	Pulse Accumulator Overflow	PAOV	PACTL	\$DC

### Example 13-23 Pulse Accumulator Overflow Interrupt Pseudocode Design

An interrupt service routine will be used. The pulse accumulator is initialized with -24 and is incremented with each external event. After 24 counts, the pulse accumulator overflows and generates an interrupt. The pseudocode design is:

```

ENABLE the pulse accumulator in event counter mode and select the
correct input edge in PACTL.
CLEAR the pulse accumulator overflow flag PAOVF in PAFLG.
SET the PAOVI bit in PACTL to enable pulse accumulator overflow
interrupts.
INITIALIZE the pulse accumulator register (PACNT) to -24.
CLEAR the interrupt mask in the condition code register.
DO Foreground Job

```

The interrupt service routine pseudocode is:

```

DO Whatever is needed at the 24th count.
INITIALIZE the pulse accumulator register (PACNT) to -24.
CLEAR the pulse accumulator overflow flag PAOVF in PAFLG.
RETURN from interrupt.

```

## 13.7 Plain and Fancy Timing

The Pulse Accumulator Control Register (PACTL) has two clock select bits (CLK1:CLK0) that can select the clock source for the TCNT register. Normally, when the Pulse Accumulator is disabled (PAEN = 0), the bus clock is prescaled by the PR2:PR1:PR0 bits (in TSCR2) giving us a counter clock frequency ranging from 8 MHz to 250 kHz. However, as shown in Figure 13-7 and Table 13-8, when the pulse accumulator is enabled (PAEN = 1), the TCNT clock source can be derived from either the event signal on PT-7 in event counting mode (PAMOD = 0) or the bus clock further divided. PAEN:CLK1:CLK0 are the select inputs for the TCNT clock multiplexer and PAEN and PAMOD select different sources for the clock. Table 13-8 gives all the various combinations. Special counting and timing requirements, such as counting long pulse streams or long periods can use these special features of the M68HCS12 timer.

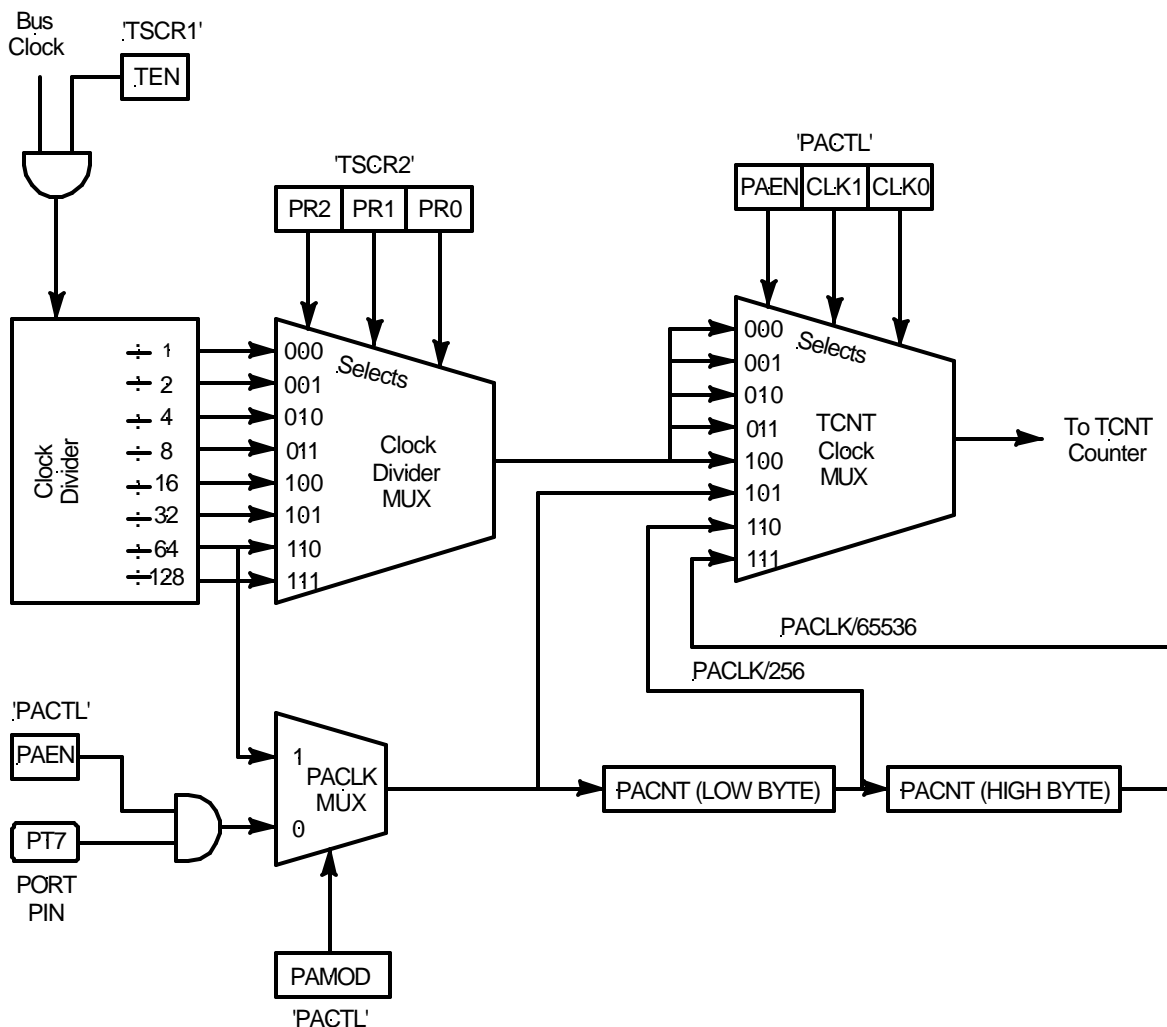


Figure 13-7 TCNT Clock Generator

Table 13-8 TCNT Counter Clock Selection

PAMOD	PAEN	CLK1	CLK0	Pulse Accumulator Mode	Clock Used for TCNT
X	0	x	x	Disabled	The normal prescaled Bus
x	1	0	0	Enabled, either mode	Normal prescaled bus clock
0	1	0	1	Event counter	Pulse accumulator input (PT7)
0	1	1	0	Event counter	PT7 divided by 256
0	1	1	1	Event counter	PT7 divided by 65,536
1	1	0	1	Gated accumulator	Bus clock divided by 64
1	1	1	0	Gated accumulator	Bus clock divided by (64 X 256)
1	1	1	1	Gated accumulator	Bus clock divided by (64 X 65,536)

Table 13-9 Pulse Accumulator Interrupt Vectors

Priority	Vector Address	Interrupt Source	Local Enable Bit	See Register	HPRIO Value to Promote
18	\$\$FDA:FFDB	Pulse Accumulator Input Edge	PAI	PACTL	\$DA
17	\$\$FDC:FFDD	Pulse Accumulator Overflow	PAOV	PACTL	\$DC

## 13.8 Clearing Timer Flags

All timer flags are cleared by writing a one to the bit

A common theme for all elements of the timer is the setting and resetting of various flags. The hardware, such as the timer overflow, sets the flag and the software you write must reset it.

When interrupts are enabled, the setting of the flag also generates the interrupt (if the I-bit is clear). The flag must always be reset, either in the interrupt service routine, or in the polling software. In all cases, the flag is reset by *writing a 1* to the flag. For example, resetting the timer channel 7 interrupt flag can be done with the following code sequences:

```
ldaa  #%10000000
staa  TFLG1
```

An alternative is:

```
bclr  TFLG1, #%01111111
```

The bit clear (BCLR) instruction has a mask byte with one's in the bit positions at which zeros are to be written (bits cleared). The way this instruction works is as follows:

The data byte is read from TFLG1, say %11000000 (timer channels 7 and 6 flags both set).

The mask byte is complemented %01111111 -> %10000000.

The complemented mask byte is ANDed with the register value and this result is written back to the register; i.e., TFLG1 is written with %10000000.

Perversely, the bit set instruction (BSET) will not work. The instruction

```
bset  TFLG1, #%10000000
```

operates like this:

The data byte is read from TFLG1, say %11000000 (timer channels 7 and 6 flags are both set).

The mask byte is ORed with the data byte and written back to TFLG1, i.e.,

TFLG1 is written with %11000000! This resets both the channel 7 and 6 flags.

### Clearing Timer Flags in C

The principles shown above must be followed when clearing flags in a C program, and we must avoid the bset and bclr instructions when resetting the interrupt flags. Example 13-24 shows a code segment to use the store instruction.

The CodeWarrior system gives us a header file that defines all registers and bits within registers. We should use unsigned char assignments to the registers to reset the flags as shown in Example 13-25.

---

**Example 13-24 Resetting Flags in C**

```

1:  /*****
2:  * Resetting flags in C
3:  *****/
4:  volatile unsigned char TFLG1 @(0x0043); /* TFLG1 location */
5:  #define C0F 1 /* Timer flag channel 0 */
6:  #define C1F 2 /* Timer flag channel 1 */
7:  #define C2F 4 /* Timer flag channel 2 */
8:  /* etc */
9:  void main(void) {
10: /* . . . */
11: /* Reset bit-0 in TFLG1 */
12: TFLG1 = C0F;
0000 c601 LDAB #1
0002 5b00 STAB TFLG1
13: /* Reset bit-0 and bit-2 in TFLG1 */
14: TFLG1 = C0F|C2F;
0004 8605 LDAA #5
0006 5a00 STAA TFLG1
15: /* . . . */
16: } 0008 3d RTS

```

---



---

**Example 13-25 Resetting Flags with CodeWarrior C**

```

4:  /*****
5:  * Resetting flags in C
6:  *****/
7:  #include <mc9s12c32.h> /* derivative information */
8:  void main(void) {
9:  /*****
10: /* . . . */
11: /* Reset bit-0 in PIFP */
12: PIFP = PIFP_PIFP0_MASK;
0000 c601 LDAB #1
0002 7b0000 STAB _PIFP
13: /* Reset bit-7 and bit-6 in PIFP */
14: PIFP = PIFP_PIFP7_MASK | PIFP_PIFP6_MASK;
0005 86c0 LDAA #192
0007 7a0000 STAA _PIFP
15: /* . . . */
16: }
000a 3d RTS

```

---

### Fast Timer Flag Clearing

Each of the methods for clearing flags shown above incur some software overhead. The *Timer System Control Register 1 (TSCRI)* has the *Timer Fast Flag Clear All (TFFCA)* bit that operates as follows when TFFCA = 1:

TFLG1 contains the flags for all eight timer channels; a read from an input capture or output compare channel (TC0 - TC7) causes the corresponding channel flag, CnF in TFLG1, to be reset automatically.

TFLG2 has the timer overflow flag; any access to the TCNT register (reading it for example) clears the TOF flag in TFLG2.

PAFLG has the pulse accumulator overflow and input edge flags. any access to the pulse accumulator count register (PACNT) clears both the PAOVF and PAIF flags in the PAFLG register.

This method eliminates software overhead associated with clearing the flag but the programmer must be wary of accidental flag clearing if an unintended access to the registers occurs.

### 13.9 Real-Time Interrupt

The real-time interrupt (RTI) operates like the timer overflow interrupt except that the rate at which interrupts are generated can be selected. See Figure 13-8. The Real-Time Interrupt is a component of the Clocks and Reset Generator (CRG) module. Chapter 20 describes the various clock sources produce by this module and used in the HCS12 in detail.

The real-time interrupt rate is generated by a 10-bit counter that divides the system oscillator clock (twice the frequency of the bus clock) by 1024. The clock can be divided further by a programmable prescaler like the prescaler used for the TCNT register. The control bits for this are in the *Real-Time Interrupt Control Register (RTICTL)* and Table 13-10 shows the interrupt intervals assuming a 16 MHz oscillator clock.

Real-time interrupts are enabled by the RTIE enable bit in the *CRG Interrupt Enable Register (CRGINT)* register. The real-time interrupt flag, *RTIF*, is in the *CRG Flags Register (CRGFLG)*, and a bit which controls the RTI operation in wait mode is found in the *CRG Clock Select Register (CLKSEL)*. Table 13-11 gives the interrupt vector for the real-time interrupt. See Chapter 20 for more information about the RTI.

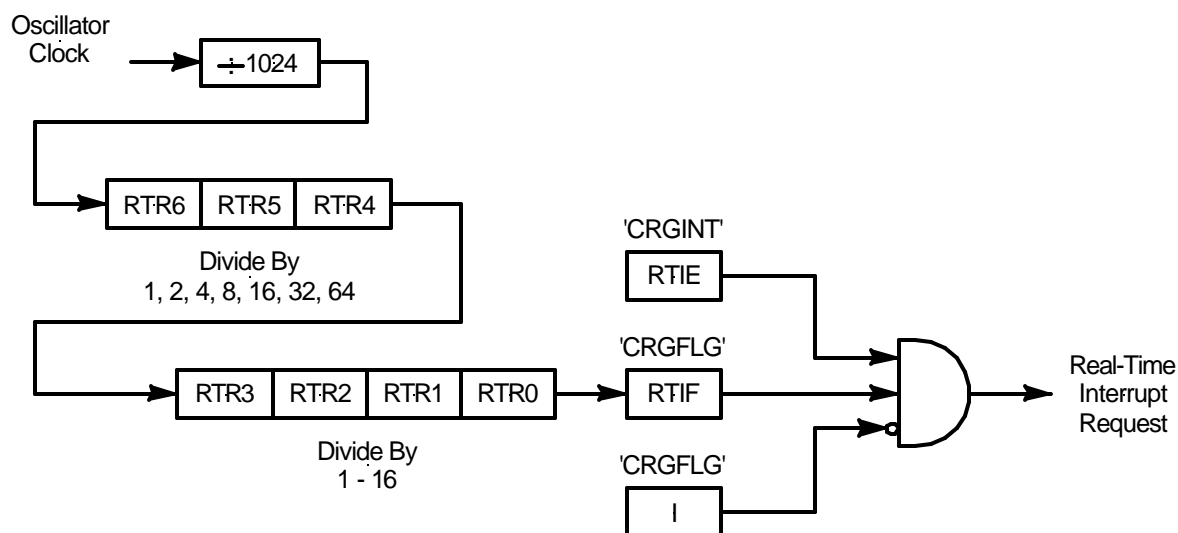


Figure 13-8 Real-time interrupt hardware

**RTICTL – Base + \$003B – RTI Control Register**

	Bit 7	6	5	4	3	2	1	0
Read:	0	RTR6	RTR5	RTR4	RTR3	RTR2	RTR1	RTR0
Write:								
Reset:	0	0	0	0	0	0	0	0

0 = reserved, unimplemented or cannot be written to.

Read: Anytime. Write: Anytime.

**RTR6:RTR0****RTR6:RTR4 – Real-Time Interrupt Prescale Select Bits**

These bits select the prescale rate for the RTI.

**RTR4:RTR0 – Real-Time Interrupt Modulus Counter Select Bits**

These bits select the modulus counter value to provide additional granularity.

See Table 13-10 for all possible values of RTR[6:4] and RTR[4:0].

**Table 13-10 RTI Interrupt Rate**

		<b>RTR[6:4] Divisor</b>							
		Off	2 <sup>10</sup>	2 <sup>11</sup>	2 <sup>12</sup>	2 <sup>13</sup>	2 <sup>14</sup>	2 <sup>15</sup>	2 <sup>16</sup>
		<b>RTR[6:4]</b>							
<b>RTR[3:0]</b>	<b>Divisor</b>	000	001	010	011	100	101	110	111
		<b>RTI Interrupt Interval – seconds (16 MHz Oscillator)</b>							
1	0000	Off	6.40E-05	1.28E-04	2.56E-04	5.12E-04	1.02E-03	2.05E-03	4.10E-03
2	0001	Off	1.28E-04	2.56E-04	5.12E-04	1.02E-03	2.05E-03	4.10E-03	8.19E-03
3	0010	Off	1.92E-04	3.84E-04	7.68E-04	1.54E-03	3.07E-03	6.14E-03	1.23E-02
4	0011	Off	2.56E-04	5.12E-04	1.02E-03	2.05E-03	4.10E-03	8.19E-03	1.64E-02
5	0100	Off	3.20E-04	6.40E-04	1.28E-03	2.56E-03	5.12E-03	1.02E-02	2.05E-02
6	0101	Off	3.84E-04	7.68E-04	1.54E-03	3.07E-03	6.14E-03	1.23E-02	2.46E-02
7	0110	Off	4.48E-04	8.96E-04	1.79E-03	3.58E-03	7.17E-03	1.43E-02	2.87E-02
8	0111	Off	5.12E-04	1.02E-03	2.05E-03	4.10E-03	8.19E-03	1.64E-02	3.28E-02
9	1000	Off	5.76E-04	1.15E-03	2.30E-03	4.61E-03	9.22E-03	1.84E-02	3.69E-02
10	1001	Off	6.40E-04	1.28E-03	2.56E-03	5.12E-03	1.02E-02	2.05E-02	4.10E-02
11	1010	Off	7.04E-04	1.41E-03	2.82E-03	5.63E-03	1.13E-02	2.25E-02	4.51E-02
12	1011	Off	7.68E-04	1.54E-03	3.07E-03	6.14E-03	1.23E-02	2.46E-02	4.92E-02
13	1100	Off	8.32E-04	1.66E-03	3.33E-03	6.66E-03	1.33E-02	2.66E-02	5.32E-02
14	1101	Off	8.96E-04	1.79E-03	3.58E-03	7.17E-03	1.43E-02	2.87E-02	5.73E-02
15	1110	Off	9.60E-04	1.92E-03	3.84E-03	7.68E-03	1.54E-02	3.07E-02	6.14E-02
16	1111	Off	1.02E-03	2.05E-03	4.10E-03	8.19E-03	1.64E-02	3.28E-02	6.55E-02

**CRGINT – Base + \$0038 – CRG Interrupt Enable Register**

	Bit 7	6	5	4	3	2	1	0
Read:	RTIE	0	0	LOCKIE	0	0	SCMIE	0
Write:								
Reset:	0	0	0	0	0	0	0	0

= reserved, unimplemented or cannot be written to.

Read: Anytime. Write: Anytime.

**RTIE****Real-Time Interrupt Enable Bit**

0 = Interrupt requests from RTI are disabled (default).

1 = Enable the RTI system. Interrupts will be requested whenever RTIF is set.

**LOCKIE, SCMIE****LOCKIE – Lock Interrupt Enable Bit.****SCMIE – Self Clock Mode Interrupt Enable Bit.**

See Chapter 20 for the details of these bits.

**CRGFLG – Base + \$0037 – CRG Interrupt Enable Register**

	Bit 7	6	5	4	3	2	1	0
Read:	RTIF	PORF	LVRF	LOCKIE	LOCK	TRACK	SCMIF	SCM
Write:								
Reset:	0	-	-	0	0	0	0	0

= reserved, unimplemented or cannot be written to.

**RTIF****Real-Time Interrupt Flag**

0 = RTI time-out has not occurred (default).

1 = RTI time-out has occurred.

This bit is set at the end of an RTI period. If RTI interrupts are enabled (RTIE = 1), RTIF causes an interrupt request. The flag can be cleared only by writing a one to the bit (See Section 13.8).

**PORF, LVRF, LOCKIE, LOCK, TRACK, SCMIF, SCM****PORF – Power on Reset Flag.****LVRF – Low Voltage Reset Flag.****LOCKIF – PLL Lock Interrupt Flag.****LOCK – Lock Status Bit.****TRACK – Track Status Bit.****SCMIF – Self Clock Mode Interrupt Flag.****SCM – Self Clock Mode Status Bit.**

See Chapter 20 for the details of these bits.

**Table 13-11 Real-Time Interrupt Vectors**

Priority	Vector Address	Interrupt Source	Local Enable Bit	See Register	HPRIO Value to Promote
7	\$FFF0:FFF1	Real-Time Interrupt	RTIE	CRGINT	\$F0

**CLKSEL – Base + \$0039 – CRG Clock Select Register**

	Bit 7	6	5	4	3	2	1	0
Read:	PLLSEL	PSTP	SYSWAI	ROAWAI	PLLWAI	CWAI	RTIWAI	COPWAI
Write:								
Reset:	0	0	0	0	0	0	0	0

**RTIWAI**

---

**RTI Stops in Wait Mode**  
 0 = RTI keeps running in wait mode (default).  
 1 = RTI stops and initializes the RTI dividers whenever the microcontroller goes into wait mode.  
 See Section 13.8

**PLLSEL, PSTP, SYSWAI, ROAWAI, PLLWAI, CWAI, COPWAI**

---

**PLLSEL – PLL Select Bit.**  
**PSTP – Pseudo Stop Bit.**  
**SYSWAI – System Clocks Stop in Wait Mode.**  
**ROAWAI – Reduced Oscillator Amplitude in Wait Mode.**  
**PLLWAI – PLL Stops in Wait Mode.**  
**COPWAI – COP Stops in Wait Mode.**  
 See Chapter 20 for the details of these bits.

## 13.10 Pulse-Width Modulator

The HCS12 can output *pulse-width modulated waveforms* continuously without causing program overhead.

The HCS12 has a *pulse-width modulation (PWM)* module giving up to six independent pulse-width modulated waveforms. After the PWM module has been initialized and enabled, PWM waveforms will be output automatically with no further action required by the program. This is very useful in applications such as controlling

stepper motors.

Figure 13-9 shows a pulse-width modulated waveform. There are two times that must be specified and controlled. These are the period ( $t_{PERIOD}$ ) and the time the output is high ( $t_{DUTY}$ ). A term used to describe a pulse-width modulated waveform is *duty cycle*. Duty cycle is defined as the ratio of  $t_{DUTY}$  to  $t_{PERIOD}$  and is usually given as a percent.

$$Duty\ Cycle = \frac{t_{DUTY}}{t_{PERIOD}} \times 100\%$$

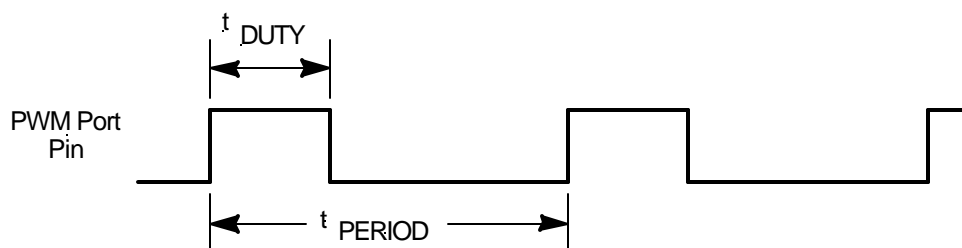


Figure 13-9 Pulse-width modulation waveform