

Chapter 13

M68HCS12 Timer

Objectives

This chapter describes the M68HCS12 timer. The timer system includes a *free running counter* and eight timer channels. These channels may be configured in any combination of timer comparison channels, called *output compares*, or *input capture* channels, which capture the time when an external event occurs. There is a *real-time periodic interrupt* and a counter for external events called the *pulse accumulator*. The interrupting capabilities of the timer are covered and programming examples are given. We also describe the *pulse width modulated* waveform generator although it is a separate hardware module.

13.1 Introduction

The timer section in the M68HCS12 is based on a 16-bit counter operating from a system clock called the *bus clock*. We will discuss how this clock is generated in Chapter 20 and concentrate now on learning about the timer capabilities. The 16-bit counter provides basic, real-time functions with the following features:

- A *timer overflow* to extend the 16-bit capability of the timer section counter.
- Up to eight *output compare functions* that can generate a variety of output waveforms by comparing the 16-bit timer counter with the contents of a programmable register.
- Up to eight *input capture functions* that can latch the value of the 16-bit counter on selected edges of eight timer input pins.
- A programmable, periodic interrupt generator called the *real-time interrupt*.
- A 16-bit *pulse accumulator* to count external events or act as a gated timer counting internal clock pulses.

The timer system is by far the most complex subsystem in the M68HCS12. It uses many I/O control registers and many control bits. All timer functions have interrupt controls and separate interrupt vectors. Figures in each of the following sections show block diagrams of each timer sub-system.

All timer functions have similar programming and operational characteristics. They all have flags in a control register that are set when some programmable condition is satisfied and that must be reset by the program. They all have interrupts that are enabled or disabled by a bit in a control register. Thus, when the operation of one timer function has been learned, the procedures are easily transferred to the others.

13.2 Basic Timer

A 16-bit *TCNT* register forms the basis of all timer functions.

The key to the operation of the M68HCS12 timer is the 16-bit, free-running counter called *TCNT* shown in Figure 13-1. Its input clock is from the system bus clock, which may be prescaled by dividing it by 1, 2, 4, 8, 16, 32, 64 or 128. A typical bus clock frequency is 8 MHz. The counter starts at \$0000 when the microcontroller is reset and runs continuously after that unless you turn it off or cause it to stop in WAIT or background debug mode. The counter cannot be set to a particular value by the program when the CPU is operating in a normal mode, but it can be written in special modes. It can also be reset when a successful output compare occurs on timer channel seven. The *TCNT* register's current value can be read anytime. Every 65,536 pulses the counter reaches a maximum and overflows. When this occurs, the counter is reset to \$0000 and a *Timer Overflow Flag* is set. This flag can extend the counter's range.

As shown in Figure 13-1 the clock for the *TCNT* counter may be selected from a prescaled bus clock or from other sources generated in the pulse accumulator. Normally the prescaled bus clock is used but we will discuss the other clock sources in Section 13.7.

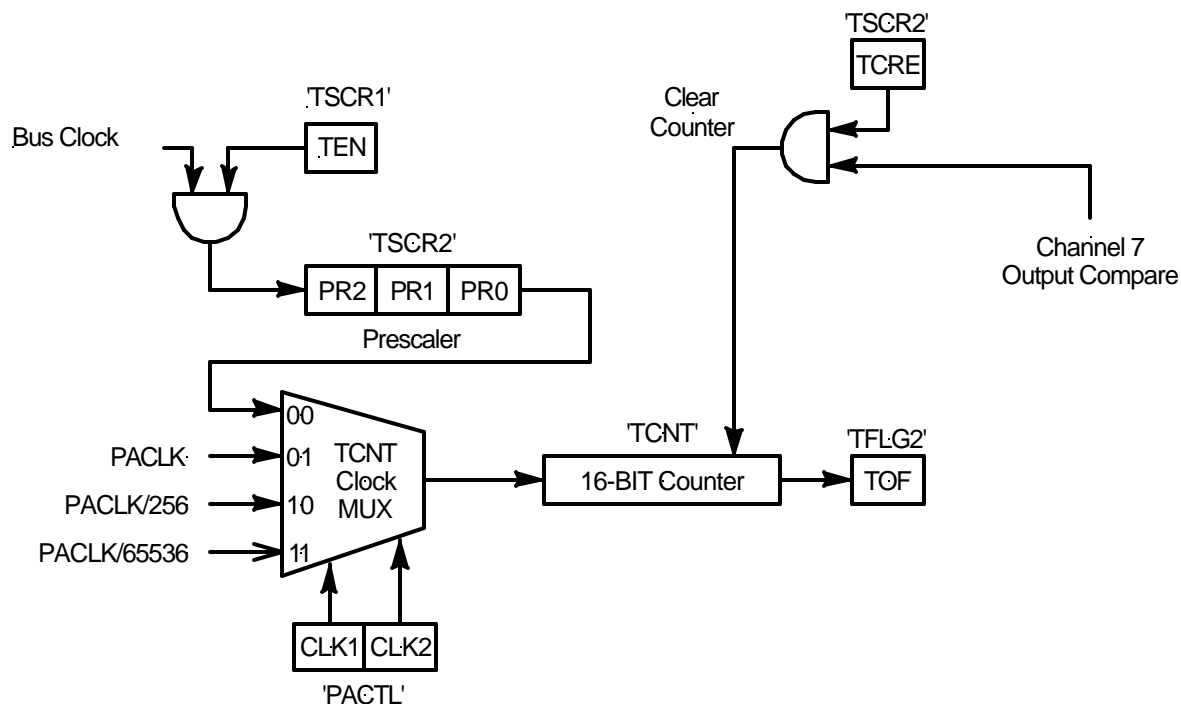


Figure 13-1 TCNT hardware.

Prescaler

The clock source for the *TCNT* counter is the system bus clock, which can be prescaled by a programmable divider, or three other sources from the pulse accumulator sub-

system. The prescaler shown in Figure 13-1 is controlled by the three PR2:PR1:PR0 bits in the *Timer System Control Register 2 (TSCR2)*.

TSCR2 – Base + \$004D –Timer System Control Register 2

	Bit 7	6	5	4	3	2	1	0
Read:	TOI	0	0	0	TCRE	PR2	PR1	PR0
Write:								
Reset:	0	0	0	0	0	0	0	0

Reset: = reserved, unimplemented or cannot be written to.

Read: Anytime. Write: Anytime.

TOI

Timer Overflow Interrupt Enable

0 = Interrupt inhibited (default).

1 = Hardware interrupt requested when the timer overflows setting the TOF flag.

For more information, see Section 13.3.

TCRE

Timer Counter Reset Enable

0 = Counter reset inhibited and TCNT free runs (default).

1 = TCNT is reset to \$0000 when a successful output compare occurs on timer channel 7.

PR2:PR1:PR0

Timer Prescaler Select

Table 13-1. Timer Clock Selection.

PR2	PR1	PR0	Bus Clock Divisor
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Sixteen-Bit Free-Running TCNT Register

The *TCNT* free-running counter is the heart of the timer system

The timer must be enabled after reset to start the *TCNT* register counting. This is done by setting the *Timer Enable – TEN* bit in the *Timer System Control Register 1* register.

TSCR1 – Base + \$0046 – Timer System Control Register 1

	Bit 7	6	5	4	3	2	1	0
Read:	TEN	TSWAI	TSFRZ	TFFCA	0	0	0	0
Write:								

Reset:

= reserved, unimplemented or cannot be written to.

Read: Anytime. Write: Anytime.

TEN**Timer Enable**

0 = Disables the main timer, including the counter. This can be used to reduce power consumption (default).

1 = Enables the timer to run normally.

TSWAI**Timer Module Stops While in Wait Mode**

0 = Allows the timer to continue running during wait mode (default).

1 = Disables the timer when the microcontroller is in wait mode. Timer interrupts cannot be used to exit wait mode.

TSFRZ**Timer Stops While in Freeze Mode**

0 = Allows the timer to continue running during freeze mode (default).

1 = Disables the timer whenever the microcontroller is in freeze mode. This is useful for emulation.

TFFCA**Timer Fast Flag Clear All**

0 = Allows the timer flag clearing to function normally (writing a one to the flag bit) (default).

1 = When this bit is set, flags are cleared by reading from the register that set the flag.

For more information see Section 13.8.

The *TCNT* register starts at \$0000 when the processor is reset and counts continuously until it reaches the maximum count of \$FFFF. On the next pulse, the counter rolls over to \$0000, sets the *Timer Overflow Flag, TOF*, and continues to count.

The timer counter is designed to be read with a 16-bit read instruction such as `ldd $44` or `ldx $44`. If you were to do two, 8-bit read instructions, for example,

```
ldaa    $44    ; Get the high 8-bits
ldab    $45    ; Get the low 8-bits
```

the low 8-bits of the counter will be incremented and will be different by the time the second load instruction is executed.

TCNT – Base + \$0044, \$0045 -- Timer Counter Register

	Bit 15	14	13	12	11	10	9	8
	TCNT15	TCNT14	TCNT13	TCNT12	TCNT11	TCNT10	TCNT9	TCNT8
Reset:	0	0	0	0	0	0	0	0
	Bit 7	6	5	4	3	2	1	0
	TCNT7	TCNT6	TCNT5	TCNT4	TCNT3	TCNT2	TCNT1	TCNT0
Reset:	0	0	0	0	0	0	0	0

Read: Anytime. Write: Only in special modes.

Timer Overflow Flag

The *TOF* bit is set when the TCNT register overflows and is reset by writing a one to bit-7 in the *TFLG2* register.

The timer overflow flag, *TOF*, is set when the timer rolls over from \$FFFF to \$0000. The programmer can extend the range of the count by detecting the overflow and incrementing another counter in the program. You may also achieve longer time intervals between overflows by setting a different prescaler value but this, of course, alters all timing done by the timer. The timer overflow flag is in the *TFLG2* register. Figure 13-2 shows the timer overflow hardware in the M68HCS12.

TFLG2 – Base + \$004F – Timer Interrupt Flag Register 2

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

= reserved, unimplemented or cannot be written to.

Read: Anytime. Write: Used in the clearing process (writing a one clears the bit).

TOF**Timer Overflow Flag**

This flag is set when the TCNT register overflows from \$FFFF to \$0000. The bit is cleared by writing a ONE to bit-7. It may be cleared by any access to TCNT if the fast flag clear all bit (TFFCA) in TSCR1 is set.

For more information, see Section 13.8.

Basic Timer Overflow Programming

The TOF can be used in two ways - polling or interrupting. In polling, the program is responsible for watching the TOF flag (by reading bit-7 in the *TFLG2* register). When the flag is set, the program can increment its local counter. If you are using the timer overflow flag, it *must* be reset by the program each time it is set by the counter. This is done by writing a one to bit-7 in the *TFLG2* register. Example 13-1 shows how to use the timer overflow bit to generate a delay in increments of 8.192 ms when using an 8-MHz bus clock.

Example 13-1 Polling the Timer Overflow Flag

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

Rel.Loc Obj. code Source line

```

1          ; This is an example of using the timer
2          ; overflow flag to generate a delay of
3          ; approximately 1 sec. (The delay will
4          ; be 122 timer overflow flags which is
5          ; 0.999 sec with an 8 MHz bus clock.
6          ; The program will alternately blink
7          ; LED1 and LED2 on the Student Learning
8          ; Kit with a CSM-12C32 board
9          ;*****
10         ; Define the entry point for the main program
11         XDEF   Entry, main
12         XREF   __SEG_END_SSTACK
13         INCLUDE timer.inc   ; Timer defns1
14         INCLUDE base.inc    ; Reg base defn
15         INCLUDE bits.inc    ; Bit defns
16         INCLUDE porta.inc   ; Port adr defns
17         INCLUDE portb.inc
18         ;*****
19         ; Constants
20         0000 0001 LED1: EQU   %00000001 ; LED1 on Port A-0
21         0000 0010 LED2: EQU   %00010000 ; LED2 on Port B-4
22         ; Code Section
23         MyCode: SECTION
24         Entry:
25         main:
26000000 CFxx xx          lds   #__SEG_END_SSTACK
27         ; Initialize the I/O
28000003 4C02 01          bset  DDRA,LED1 ; Make PA-0 output
29000006 4C02 10          bset  DDRB,LED2 ; Make PB-4 output
30         ; Clear the TOF flag
31000009 8680            ldaa  #TOF
3200000B 5A4F            staa  TFLG2
3300000D 8601            ldaa  #LED1      ; Initial LED display
3400000F C600            ldab  #!LED2
35         ; Enable the timer
36000011 4C46 80          bset  TSCR1,TEN
37         main_loop:
38         ; DO
39         ; Display on the LEDs
40000014 5A00            staa  PTA
41000016 5B00            stab  PTB
42         ; Delay approximately one sec
43000018 16xx xx          jsr   tof_1_sec_delay
4400001B 41              coma           ; Next display value
4500001C 51              comb
46         ; FOREVER
4700001D 20F5            bra   main_loop
48         ;*****
49         ; Timer Overflow 1 second delay
50         ; Registers modified: CCR
51         ;*****
52         ;
53         ; Constant Equates
54         0000 007A SEC_1: EQU   122 ; Number of TOFs for 1 sec
55
56         SubCode: SECTION
57         ; One second delay using the TOF
58         tof_1_sec_delay:
59000000 36              psha

```

¹ A listing of all include files used in these examples may be found in Appendix D.

```

60000001 34                pshx
61                ; Initialize the TOF counter
62000002 CE00 7A          ldx   #SEC_1
63                ; DO
64                ;   Wait until the TOF occurs
65                spin:
66000005 F700 4F          tst   TFLG2
67000008 2AFB            bpl   spin
68                ; Reset the TOF
6900000A 8680            ldaa  #TOF
7000000C 5A4F            staa  TFLG2
71                ; Decrement the TOF counter
7200000E 09              dex
73                ; ENDO
74                ; WHILE (number overflows < SEC_1)
7500000F 26F4            bne   spin
76                ; Now done, return
77000011 30              pulx
78000012 32              pula
79000013 3D              rts
80                ;*****
81                ; No constant or variable data section needed

```

Explanation of Example 13-1

Example 13-1 shows how to use the timer overflow occurring at intervals of 8.192 ms to delay for longer periods. The initialization section (*lines 28 – 36*) set the data direction registers for the bits in Port A and Port B that are connected to the LEDs. The TOF bit is reset in *line 32*, the timer is enabled in *line 36*, and an overflow counter is initialized in *line 62*. *Lines 66 and 67* are a spin loop waiting for the TOF bit to be set. When it is, the flag is reset in *lines 69 and 70* and the counter is decremented (*line 72*). If the counter is not zero, the spin loop is re-entered, otherwise the subroutine returns to the calling program. Example 13-2 shows a C program version of this example.

The delay in Example 13-1 has a resolution one-half of the period of the timer overflow, ± 4.096 ms. If you would like to generate an exact delay, to the resolution of the bus clock (125 ns), you could count the extra clock cycles needed to make up the delay. A better and easier way is to use the *output compare* function discussed in Section 13.4.

Example 13-2 C Program Version of the Timer Overflow Flag Program

```

/*****
* This is an example of using the timer overflow flag
* to generate a delay of approximately 1 sec. (The delay
* will be 122 timer overflow flags which is 0.999 sec
* with an 8 MHz bus clock.
* The program will alternately blink LED1 and LED2 on
* the Student Learning Kit with a CSM-12C32 board
*****/
#include <mc9s12c32.h>2          /* derivative information */

```

² All registers and bits in registers are defined in the mc9s12c32.h header file supplied by the CodeWarrior™ environment. The header defines the memory addresses for the control registers, such as DDRA, and assignment to these are unsigned char types. The header file defines bits within registers as, for example, DDRA_BIT0. Assignments to these are bit assignments using bset or bclr instructions. Your C program can access registers as bytes or individual bits.

```

void tof_1_sec_delay( void );
void main(void) {
    /* Initialize the I/O */
    DDRA_BIT0 = 1; /* Make Port A bit-0 output */
    DDRB_BIT4 = 1; /* Port B bit-4 output */
    /* Clear the timer overflow flag */
    TFLG2 = TFLG2_TOF_MASK;
    /* Enable the timer */
    TSCR1_TEN = 1;
    /* Set LED1 on and LED2 off */
    PORTA_BIT0 = 0; /* Active low LEDs */
    PORTB_BIT4 = 1;
    /* DO */
    for(;;) {
        /* Delay 1 second */
        tof_1_sec_delay();
        /* Toggle the display bits */
        if (PORTA_BIT0 == 1) PORTA_BIT0 = 0;
        else PORTA_BIT0 = 1;
        if (PORTB_BIT4 == 1) PORTB_BIT4 = 0;
        else PORTB_BIT4 = 1;
    } /* wait forever */
    /* FOREVER */
}
/*****
 * Timer overflow 1 sec delay
 *****/
#define SEC_1 122 /* Number of TOFs per sec */
void tof_1_sec_delay( void ) {
    int counter;
/*****
    /* Initialize the counter */
    counter = SEC_1;
    do {
        /* Wait until the TOF */
        while (TFLG2_TOF == 0){/* spin */}
        /* Reset the TOF flag */
        TFLG2 = TFLG2_TOF_MASK;
    }
    while (--counter > 0);
}

```

13.3 Timer Overflow Interrupts

Timer interrupts allow your program to do other things while waiting for a timing event to occur.

A shortcoming of the program in Example 13-1 is that the TOF bit must be polled until 122 overflows have occurred. During this time the program could be doing other things but an overflow might be missed. An interrupt can allow the program to go about some other business while waiting for an event, the timer overflow for

example, to occur. To use the timer overflow interrupt, the *TOI* bit in *TSCR2* must be enabled, the interrupt vector must be initialized as described in Chapter 11, and the I-bit in the condition code register must be unmasked.

The *timer overflow enable* bit must be set to allow the interrupt request to be generated.

The timer overflow flag (TOF) is ANDed with the timer overflow interrupt enable bit (TOI) to generate the interrupt request as shown in Figure 13-2. This request is further qualified

by the interrupt mask bit (I-bit) in the condition code register as we discussed in Chapter 11. The timer overflow interrupt vector must be initialized properly to be able to transfer to the interrupt service routine. An interrupt service routine using the timer overflow flag is given in Example 13-3

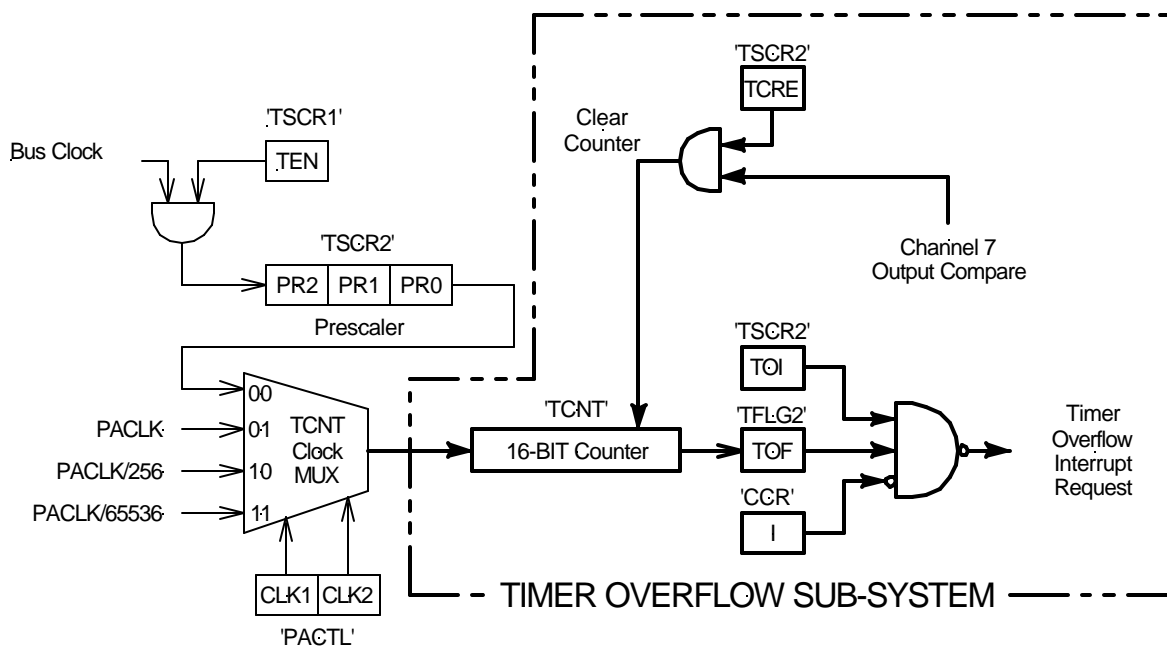


Figure 13-2 Timer overflow interrupt hardware.

Table 13-2 Timer Overflow Interrupt Vector

Priority	Vector Address	Interrupt Source	Local Enable Bit	See Register	HPRIO Value to Promote
16	\$FFDE:FFDF	Timer Overflow	TOI	TSCR2	\$DE

Example 13-3 Timer Overflow Interrupts

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

```

Rel.Loc   Obj. code Source line
-----
1          ; This is a test program showing the use
2          ; of the Timer Overflow Interrupts.
3          ; The program uses the WAI instruction
4          ; while waiting for an interrupt.
5          ; After 1/2 second has elapsed, LED1 on
6          ; the Student Learning Kit CSM-12C32
7          ; board is toggled.
8          ;*****
9          ; Define the entry point for the main program
10         XDEF      Entry, main
11         XDEF      tof_pa0_isr
12         XREF      __SEG_END_SSTACK

```

```

13             INCLUDE timer.inc ; Timer defns
14             INCLUDE bits.inc  ; Bit defns
15             INCLUDE base.inc  ; Reg base defn
16             INCLUDE porta.inc ; Port A defn
17             ;*****
18             ; Constants
19             0000 0001 LED1: EQU BIT0 ; LED1 on Port A - 0
20             0000 003D HALF_SEC: EQU 61 ; TOFs for 1/2 sec
21             ; Code Section
22             MyCode: SECTION
23             Entry:
24             main:
25000000 CFxx xx         lds  #__SEG_END_SSTACK
26             ; DO Initialize the I/O
27000003 4C02 01         bset  DDRA,LED1; Make PA-0 output
28000006 8601           ldaa  #LED1 ; Initial LED display
29             ; Display on the LEDs
30000008 5A00           staa  PTA
31             ; Initialize the 1/2 sec counter
3200000A 180B 3Dxx      movb  #HALF_SEC,half_sec_count
33             00000E xx
34             ; Enable the Timer
3400000F 4C46 80         bset  TSCR1,TEN
35             ; Reset timer overflow flag
36000012 8680           ldaa  #TOF
37000014 5A4F           staa  TFLG2
38             ; Enable the TOF interrupts
39000016 4C4D 80         bset  TSCR2,TOI
40             ; Unmask interrupts
41000019 10EF           cli
42             ; ENDO Initialize the I/O
43             ; DO
44             main_loop:
4500001B 3E             wai          ; Wait for interrupt
46             ; IF the toggle flag is set
4700001C 1Fxx xx80      brclr toggle_flag,BIT7,end_if
48             000020 0A
49             ; THEN toggle the bit and clear the flag
49000021 9600           ldaa  PTA
50000023 8801           eora  #LED1 ; Toggle the bit
51000025 5A00           staa  PTA
52000027 1Dxx xx80      bclr  toggle_flag,BIT7 ; Clear flag
53             end_if:
5400002B 20EE           bra   main_loop
55             ; FOREVER
56             ;*****
57             ; Timer overflow ISR
58             ;*****
59             tof_pa0_isr:
60             ; Blinks the LED1 on Port A, Bit 0 at 1 Hz
61             ; Decrement the counter
6200002D 73xx xx         dec   half_sec_count

```

```

63                ; IF the counter is zero
64000030 2609          bne  endif
65                ; THEN set the flag to toggle the LED
66000032 1Cxx xx80     bset  toggle_flag,BIT7
67                ; Initialize the counter
68000036 180B 3Dxx     movb  #HALF_SEC,half_sec_count
    00003A xx
69                ; ENDIF counter is zero
70                endif:
71                ; Clear the TOF flag
7200003B 8680          ldaa  #TOF
7300003D 5A4F          staa  TFLG2
7400003F 0B           rti
75                ;*****
76                ;
77                Data: SECTION
78000000             half_sec_count: DS.B 1
79000001             toggle_flag: DS.B 1

```

Explanation of Example 13-3

Interrupts are used here like we described in Chapter 11. The main program consists on an initialization section, *lines 26 – 42* where the Port A direction is set and the LED turned on (*line 27, 28 and 32*), a counter used in the interrupt service routine is initialized (*line 32*) and the timer is enabled (*line 34*). Then the timer overflow flag is reset (*line 37*), the TOF interrupts enabled (*line 39*) and interrupts unmasked (*line 41*). The main loop is simply a *WAI* wait for interrupt instruction at *line 45*.

The interrupt service routine is entered each time the timer overflows, at intervals of 8.192 ms. At half-second intervals, or 61 timer overflows, a *toggle_flag* is set to be used in the main program to toggle LED. Before returning from the interrupt service routine, the timer overflow flag is reset in *lines 72, 73*. See Example 13-4 and Example 13-5

Example 13-4

Why is the TOF bit cleared in *line 40* in Example 13-3?

Solution:

If the TOF bit is set when the timer interrupts are enabled and interrupts are unmasked, an interrupt will occur immediately. This may upset the required timing for the first iteration of the program.

Example 13-5

Why is the TOF bit cleared in *line 68* in Example 13-3?

Solution:

If the TOF bit is not cleared in the interrupt service routine, as soon as the RTI instruction is executed the TOF bit will immediately generate another interrupt request. The program will appear to never get out of the ISR.

Example 13-6 Timer Overflow Interrupts in C

```

/*****
 * This is a test program showing the use of the Timer
 * Overflow interrupt. The program uses the WAI instruction
 * while waiting for an interrupt. After a 1/2 second has elapsed,
 * LED1 on the Student Learning Kit CSM-12C32 board
 * is toggled.
 *****/
#include <mc9s12c32.h>      /* derivative information */
#define HALF_SEC  61      /* TOFs per 1/2 sec */
static int half_sec_counter;
void main(void) {
/*****
 /* Initialize the I/O */
  DDRA_BIT0 = 1; /* Make Port A bit-0 output */
 /* Set LED1 on */
  PORTA_BIT0 = 0; /* Active low LEDs */
 /* Initialize 1/2 second counter */
  half_sec_counter = HALF_SEC;
 /* Clear the timer overflow flag */
  TFLG2 = TFLG2_TOF_MASK;
 /* Enable the timer */
  TSCR1_TEN = 1;
 /* Enable the TOF interrupts */
  TSCR2_TOI = 1;
 /* Unmask interrupts */
  EnableInterrupts;

 /* DO */
  for(;;) {
    /* Wait for the interrupt */
    asm (wai );
    /* If the half_sec_counter is zero */
    if (half_sec_counter == 0) {
      /* Then toggle the bit and reset the counter */
      if (PORTA_BIT0 == 1) PORTA_BIT0 = 0;
      else PORTA_BIT0 = 1;
      half_sec_counter = HALF_SEC;
    }
    /* FOREVER */
  }
}

/*****
 * Timer overflow interrupt service routine
 *****/
#define SEC_1  122 /* Number of TOFs per sec */
void interrupt 16 tof_pa0_isr( void ) {
  /* Decrement the counter */
  --half_sec_counter;
  /* Reset the TOF flag */
  TFLG2 = TFLG2_TOF_MASK;
}

```
