Chapter 8: Timer Functions

Many applications require a dedicated timer system to perform

- Time delay creation and measurement.
- Period and pulse width measurement.
- Frequency measurement.
- Event counting.
- Arrival time comparison.
- Time-of-day tracking.
- Periodic interrupt generation.
- Waveform generation.

HCS12 Timer Module

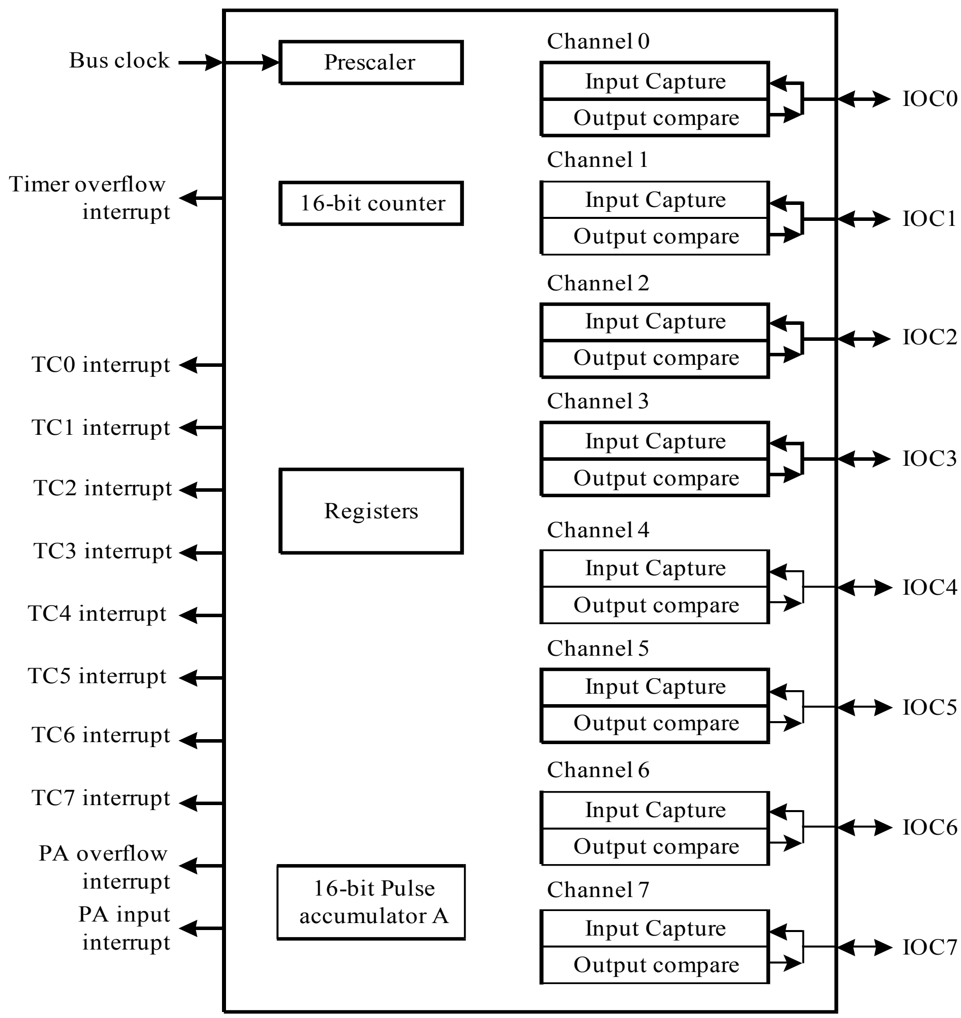


Figure 8.1 HCS12 Standard Timer (TIM) block diagram

HCS12 has a standard timer module, consisting of

- 8 channels of input-capture (IC) and output-compare (OC) modules. Each channel can be configured to perform IC or OC function, but not both at the same time.

- One 16-bit main timer counter (TCNT) serves as the base timer for these 8 channels. TCNT can be started and stopped at any time, and is driven by a clock signal derived from the E-clock divided with a prescaler.

- Port T pins PT7~PT0 correspond to the timer module inputs/outputs IOC7~IOC0.

- One 16-bit pulse accumulator, which can be used to count the number of events that has occurred. It shares the PT7 pins, which is referred as the PAI pin when the pin is used as the pulse-accumulator input. Usually used to count the events arriving in a certain interval or measure frequency or pulse width.

Timer Counter Register (TCNT) = counter

- TCNT is a 16-bit counter, required for IO and OC functions.

- Must be read in one 16-bit (high and low bytes) operation in one access in order to obtain the correct value. It is because the TCNT counter does not stop during the access operation. Thus the high byte and low byte won't be the same in two separate accesses.

- Three other registers are related to the operation of the TCNT counter: TSCR1, TMSK2, and TFLG2.

Timer System Control Register (TSCR1)

- Setting bit 7 =1 (or 0) in TSCR1 starts (or stops) the TCNT timer.

- Setting bit 4 = 1 enables timer fast flag clear function to clear the timer overflow flag (TOF) in the TFLG2 register. Afterward, a read from or a write to TCNT will clear TOF.

- If bit 4 = 0, then the user must manually write a 1 to TOF to clear the flag.

```
movb   #$80, TSCR          ; enable TCNT to count, manual TOF clear
                           ; but need "bset TFLG2,$80" to clear TOF

movb   #$90, TSCR          ; enable TCNT to count; fast TOF clear
```

	7	6	5	4	3	2	1	0
	TEN	TSWAI	TSFRZ	TFFCA	0	0	0	0
value after reset	0	0	0	0	0	0	0	0

TEN -- timer enable bit

0 = disable timer, this can be used to save power consumption
1 = allows timer to function normally

TSWAI -- timer stops while in wait mode bit

0 = allows timer to continue running during wait mode
1 = disables timer when MCU is in wait mode

TSFRZ -- timer and modulus counter stop while in freeze mode

0 = allows timer and modulus counter to continue running while in freeze mode
1 = disables timer and modulus counter when MCU is in freeze mode

TFFCA -- timer fast flag clear all bit

0 = allows timer flag clearing to function normally
1 = For TFLG 1, a read from an input capture or a write to the output compare channel causes the corresponding channel flag, CnF, to be cleared. For TFLG2, any access to the TCNT register clears the TOF flag. Any access to the PACN3 and PACN2 registers clears the PAOVF and PAIF flags in the PAFLG register. Any access to the PACN1 and PACN0 registers clears the PBOVF flag in the PBFLG register.

Figure 8.2 Timer system control register 1 (TSCR1)

	7	6	5	4	3	2	1	0
	TOI	0	0	0	TCRE	PR2	PR1	PR0
value after reset	0	0	0	0	0	0	0	0

TOI -- timer overflow interrupt enable bit
0 = interrupt inhibited
1 = interrupt requested when TOF flag is set

TCRE -- timer counter reset enable bit
0 = counter reset inhibited and counter free runs
1 = counter reset by a successful output compare 7
If TC7 = $0000 and TCRE = 1, TCNT stays at $0000 continuously. If TC7 = $FFFF and TCRE = 1, TOF will never be set when TCNT rolls over from $FFFF to $0000.

Figure 8.3 Timer system control register 2 (TSCR2)

Timer Interrupt Mask Register 2 (TMSK2) = Timer System Control Register 2 (TSCR2)

- If (TOI) bit 7 = 1 in TMSK2, an interrupt will be requested when the TCNT timer overflows (i.e., rolls over from $FFFF to $0000).

- When the content of the TCNT register is equal to the TC7 register, the TCNT register can be cleared to 0 by setting (TCRE) bit 3 = 1 of the TMSK2.

- The clock input to the TCNT counter can be derived from the E-clock divided by a prescaler, which is selected by bits 2~0 of the TMSK2.

```
movb   #$07, TMSK2        ; prescaler=128; free-run mode (TCRE=0)
                          ; disable TCNT overflow interrupt (TOI=0)
```

Table 8.1 Timer counter prescale factor

PR2	PR1	PR0	Prescale Factor
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

Timer Interrupt Flag 2 Register (TFLG2)

- Only bit 7 (TOF) of this register is implemented.

- Bit 7 = 1 whenever TCNT rolls over from $FFFF to $0000 (or overflow).

- TOF can be cleared by writing a 1 to it.

```
bset    TFLG2,$80     ; writing a 1 to clear TOF
```

Input Capture (IC) Function

- Some applications need to know the arrival time of events (from a signal).
- Physical time is represented by the contents of the main timer (TCNT).
- The occurrence of an event is represented by a signal (rising or falling) edge.
- The time when an event occurs can be recorded by latching the count of TCNT when a signal edge arrives.
- HCS12 has 8 input-capture (IC) channels for this operation.
- Each channel has a 16-bit IC register, an input pin, edge-detection logic, and interrupt generation logic.
- Can be used to measure frequency, period, pulse width, duty cycle, and timing reference of the selected signal.

Input-Capture/Output-Compare Selection

- IC channels share most of the circuit (such as signal pins and registers) with OC functions. Thus, they cannot be enabled simultaneously.
- Selection of IC and OC is done by programming the Timer Input-Capture/Output-Compare Select (TIOS) register.

	7	6	5	4	3	2	1	0
	IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0
value after reset	0	0	0	0	0	0	0	0

IOS[7:0] -- Input capture or output compare channel configuration bits
0 = The corresponding channel acts as an input capture
1 = The corresponding channel acts as an output compare

Figure 8.5 Timer input capture/output compare select register (TIOS)

```
movb   #$F0,TIOS      ; IOC7~4 for OC operation; IOC3~0 for IC operation
```

Pins for Timer Port

- Port T has 8 signal pins (PT7~PT0) that can be used as IC/OC channels, or general I/O pins when timer function is not selected.

- Pin 7 can be program to act as input capture (IC7), output compare (OC7), or pulse accumulator input (PAI) functions.

- When a timer port pin is used as general I/O, its direction is configured by the DDRT register and its value is reflected in the associated bit in PortT.

Timer Control Register 3 and 4 (for Input Capture)

- The signal (rising or falling) edge to be captured is selected by the timer control registers, TCTL3 and TCTL4.

- The edge to be captured is selected by two bits. The user can choose to capture the rising edge, falling edge, or both edges.

- When an IC7~0 channel is selected but capture function is disabled (i.e., 00 at the corresponding two bits in TCTL3 and TCTL4), the associated pin can be used as a general-purpose I/O pin.

	7	6	5	4	3	2	1	0
	EDG7B	EDG7A	EDG6B	EDG6A	EDG5B	EDG5A	EDG4B	EDG4A
value after reset	0	0	0	0	0	0	0	0

(a) Timer control register 3 (TCTL3)

7	6	5	4	3	2	1	0
EDG3B	EDG3A	EDG2B	EDG2A	EDG1B	EDG1A	EDG0B	EDG0A
0	0	0	0	0	0	0	0

(b) Timer control register 4 (TCTL4)

EDGnB EDGnA -- Edge configuration

EDGnB	EDGnA	
0	0	: Capture disabled
0	1	: Capture on rising edges only
1	0	: Capture on falling edges only
1	1	: Capture on both edges

Figure 8.5 Timer control register 3 and 4

Timer Interrupt Mask Register 1 (TMSK1 or TIE) & Timer Interrupt Flag Register 1 (TFLG1)

- An IC7~0 channel can optionally generate an interrupt request to the CPU on the arrival of a selected edge by setting the associated interrupt enable bit (C7I~C0I) in TMSK1 (also aka TIE).

	7	6	5	4	3	2	1	0
	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I
reset:	0	0	0	0	0	0	0	0

C7I-C0I: input capture/output compare interrupt enable bits
0 = interrupt disabled
1 = interrupt enabled

Figure 8.7 Timer interrupt enable register (TIE)

	7	6	5	4	3	2	1	0
	C7F	C6F	C5F	C4F	C3F	C2F	C1F	C0F
reset:	0	0	0	0	0	0	0	0

CnF: input capture/output compare interrupt flag bits
0 = interrupt condition has not occurred
1 = interrupt condition has occurred

Figure 8.8 Timer interrupt flag register 1 (TFLG1)

- When a selected edge arrives at the IC7~0 pin, the corresponding flag (C7F~C0F) in TFLG1 will be set to 1.

- To clear a flag in the TFLG1 register, write a 1 to it.

```
movb   #$01,TFLG1   ; clear the channel 0 flag (C0F)
```

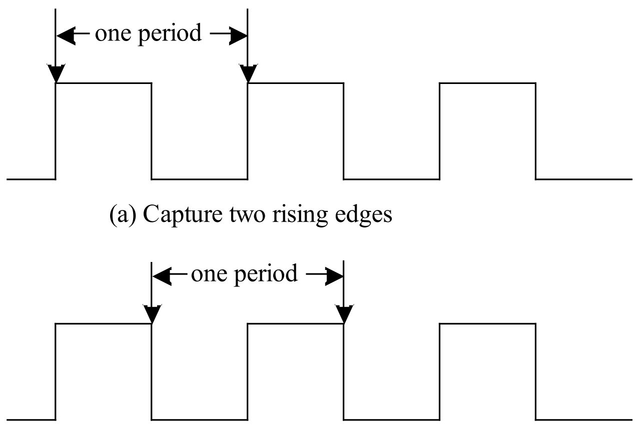
- A better way to clear the flag that incurs less overhead is by setting bit 4 =1 (timer fast clear all bits) of TSCR1. This clears a flag by reading the associated IC register or writing a new value into the OC register. This operation is normally needed for the normal operation of the IC/OC function.

- Each IC7~0 channel has a 16-bit register (TC7~TC0) to hold the count value when the selected signal edge arrives at the pin. This register is also used as the OC register when the output-compare function is selected.

Applications of Input Capture Function

1. ***Event arrival time recording***. Some applications, for example, a swimming competition, needs to compare the arrival times of several swimmers. The number of events that can be compared is limited by the number of IC channels.

2. ***Period measurement***. The IC function should be configured to capture the main timer values corresponding to two consecutive rising or falling edges.



(a) Capture two rising edges

(b) Capture two falling edges

Figure 8.9 Period measurement by capturing two consecutive edges

3. ***Pulse-width measurement.*** Capture two adjacent rising and falling edges.

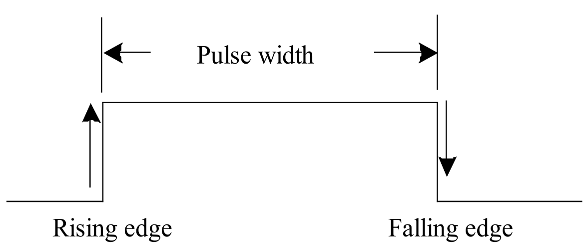


Figure 8.10 Pulse-width measurement using input capture

4. ***Interrupt generation.*** Each IC input can be used as an edge-sensitive interrupt source. Once enabled, interrupts will be generated on the selected edge(s).

5. ***Event counting***. An event can be represented by a signal edge. An IC channel can be used in conjunction with an OC function to count the number of events that occur during an interval. An event counter can be set up and incremented by the IC interrupt service routine by counting the number of signal edges arrived during a period.

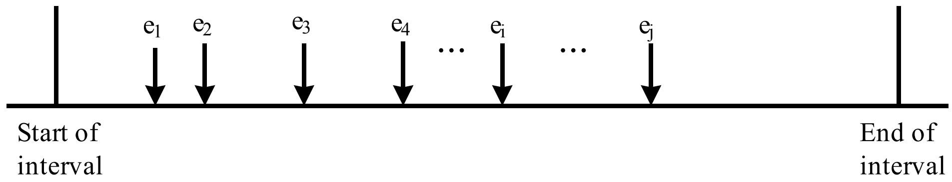


Figure 8.11 Using an input-capture function for event counting

6. ***Time referenced action***. An IC function is used in combination with an OC function. For example, to activate an output signal a certain number of clock cycles after detecting an input event. The IC function would be used to record the time at which the edge is detected. A number corresponding to the desired delay would be added to this captured value and stored to an OC register.

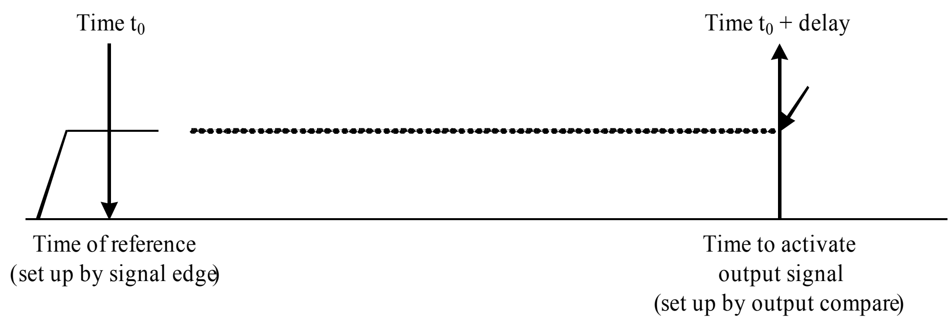
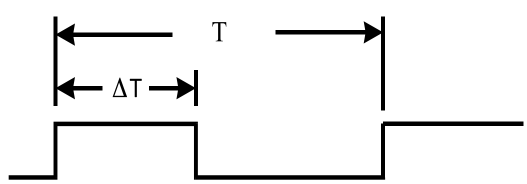


Figure 8.12 A time reference application

7. ***Duty Cycle Measurement***. The duty cycle is the percent of time that the signal is high within a period in a periodic digital signal.



$$\text{duty cycle} = \frac{\Delta T}{T} * 100\%$$

Figure 8.13 Definition of duty cycle

8. ***Phase Difference Measurement***. Phase difference is defined as the difference of arrival times (in % of a period) of two signals that have the same frequency but do not coincide in their rising and falling edges.

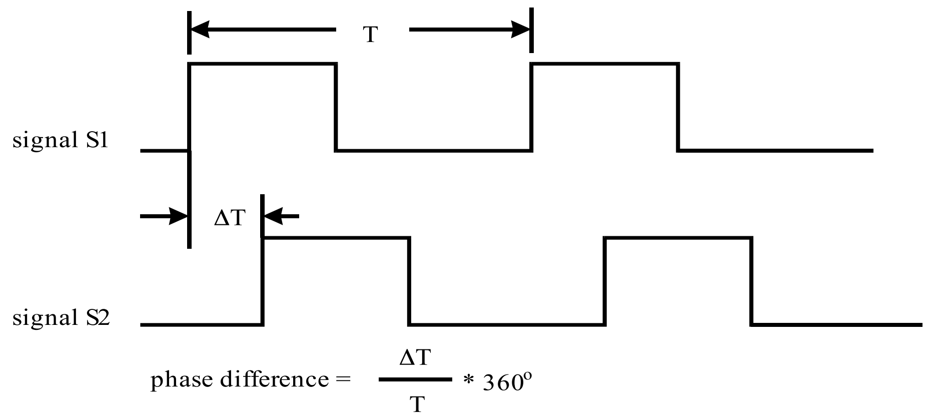


Figure 8.14 Phase difference definition for two signals

Example: Displaying the number of "overflows" in the TCNT timer on the LEDs.

The clock input to the 16-bit TCNT timer is equal to E-clock/prescaler. At E-clock = 24 MHz and prescaler=128, the TCNT timer overflows every 2^16/(24MHz/128) = 0.35 sec.

```
#include "reg9s12.h"

        org     $2000
        movb    #$FF,DDRB     ; set PTB as output for LEDs
        movb    #$FF,DDRJ     ; set PTJ as output  (required by Dragon12+)
        movb    #$00,PTJ      ; set PTJ1=0 to enable LEDs
        movb    #$FF,DDRP     ; turn off 7-segment displays and RGB LEDs
        movb    #$FF,PTP      ;

        movb    #$80,TSCR     ; (=TSCR1) enable timer; manual TOF clear
        movb    #$07,TMSK2    ; (=TSCR2) prescaler=128; free-run; no overflow interrupt

        ldaa    #0            ; A stores the number of times the timer overflowing
over    bset    TFLG2,$80     ; clear the TOF by writing a 1 & start to count
        brclr   TFLG2,$80,*   ; wait 0.35s for timer overflow; if TOF=1, exit loop
        inca                  ; add 1 to the number of timer overflows
        staa    PortB         ; display the content of A on LEDs
        bra     over

        end
```

Possible project: Display the measured period, pulse width, duty cycle, etc of an unknown signal on the 7-segment or LCD displays.

Example: Use IC channel 0 (IC0) to measure the period of an unknown signal connected to PT0.

As each IC register is 16 bits, the maximum measurable period and accuracy depend on the value of prescaler. For example,

1. Set the prescale factor to TCNT to 1 and keep track of the number of times that TCNT overflows. [2^{16}/24MHz = 2.73 ms per TCNT overflow; accuracy = 1/24MHz = 0.04167 μs]

2. Set the prescale factor to 64 and do not keep track of the number of times that TCNT overflows. [2^{16}/(24MHz/64) = 174.76 ms per TCNT overflow; accuracy = 1/(24MHz/64) = 2.667 μs]

Here, use prescaler = 64. The period measurement will be in number of clock cycles. The period of each clock cycle is 1/(24MHz/64) = 2.667 μs.

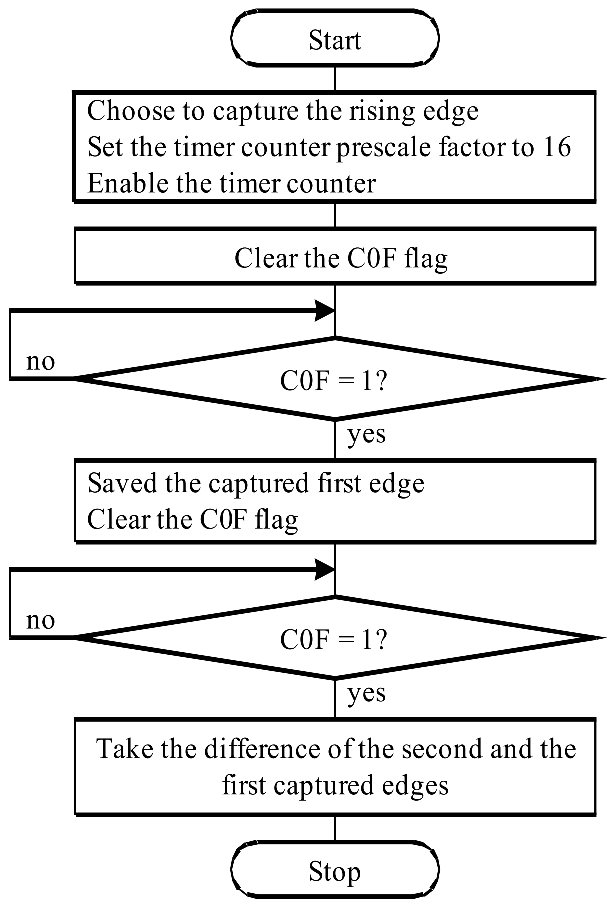


Figure 8.16 Logic flow of period measurement program

```
#include "reg9s12.h"

        org     $1000
edge    ds.b    2                ; hold the arrival time of the first edge
period  ds.b    2                ; store the period

        org     $2000
        movb    #$90,TSCR        ; (=TSCR1) enable timer; enable fast TOF clear
        movb    #$06,TMSK2       ; prescaler=64; free runs; no timer overflow interrupt
        bclr    TIOS,$01         ; enable IOC at channel 0
        movb    #$01,TCTL4       ; capture rising edge of the PT0 signal
        movb    #$01,TFLG1       ; clear the C0F interrupt flag to wait for an event to occur

        brclr   TFLG1,$01,*      ; wait for the arrival of the first rising edge
        ldd     TC0              ; if C0F=1, the edge triggers an interrupt. The TC0 timer
        std     edge             ; content gives 1st edge time; C0F is cleared automatically
                                 ; due to the use of #$90 in TSCR
        brclr   TFLG1,$01,*      ; clear C0F to wait for the arrival of the 2nd rising edge
        ldd     TC0              ; load the current TC0 timer content
        subd    edge             ; period = TCO - edge1
        std     period
        swi
        end
```

Measurement Steps"

1) Set up the function generator with Hi-Z, 4.7Vpp (i.e., 0-4.7V), square wave, 1 kHz, and 50% duty cycle. (Important: Use the oscilloscope to confirm the setting!)
2) Connect the positive and ground terminals of the function generator to the PT0 and GND socket holes, respectively, using two pieces of wire. The two holes are found around the middle of the left-side (vertical) socket on Dragon12+.
3) Load the program to Dragon12+.
4) Type "g 2000" to make one measurement. The content of D gives the number of clock cycles in hex format.
5) Convert the hex value to decimal value. Then, multiple it with 2.667 μs as the measured period of the "unknown" signal.
6) Set the function generation with a different frequency setting, then repeat Steps 4-6.

For 1 kHz signal, the program gives D = $0177, which is equal to 375 clock cycles. This gives the period of 375 x 2.667 μs = 1 ms.

For 2 kHz, D = $00BB = 187 clock cycles, giving the period of 187 x 2.667 μs = 0.5 ms.

For 500 Hz, D = $02EE = 750 clock cycles, giving the period of 750 x 2.667 μs = 2 ms.

Example: Write a program to measure the pulse width of a signal connected to PT0 using IC function at channel 0 (IC0).

Setting the prescale factor to 32, each count of the TCNT timer (or clock cycle) is 1/(24MHz/32) = 1.333 μs, which is used as the unit of measurement.

Since the pulse width may be longer than 2^{16} clock cycles, we need to keep track of the number of times that the TCNT timer overflows. Each TCNT overflow adds 2^{16} clock cycles to the pulse width.

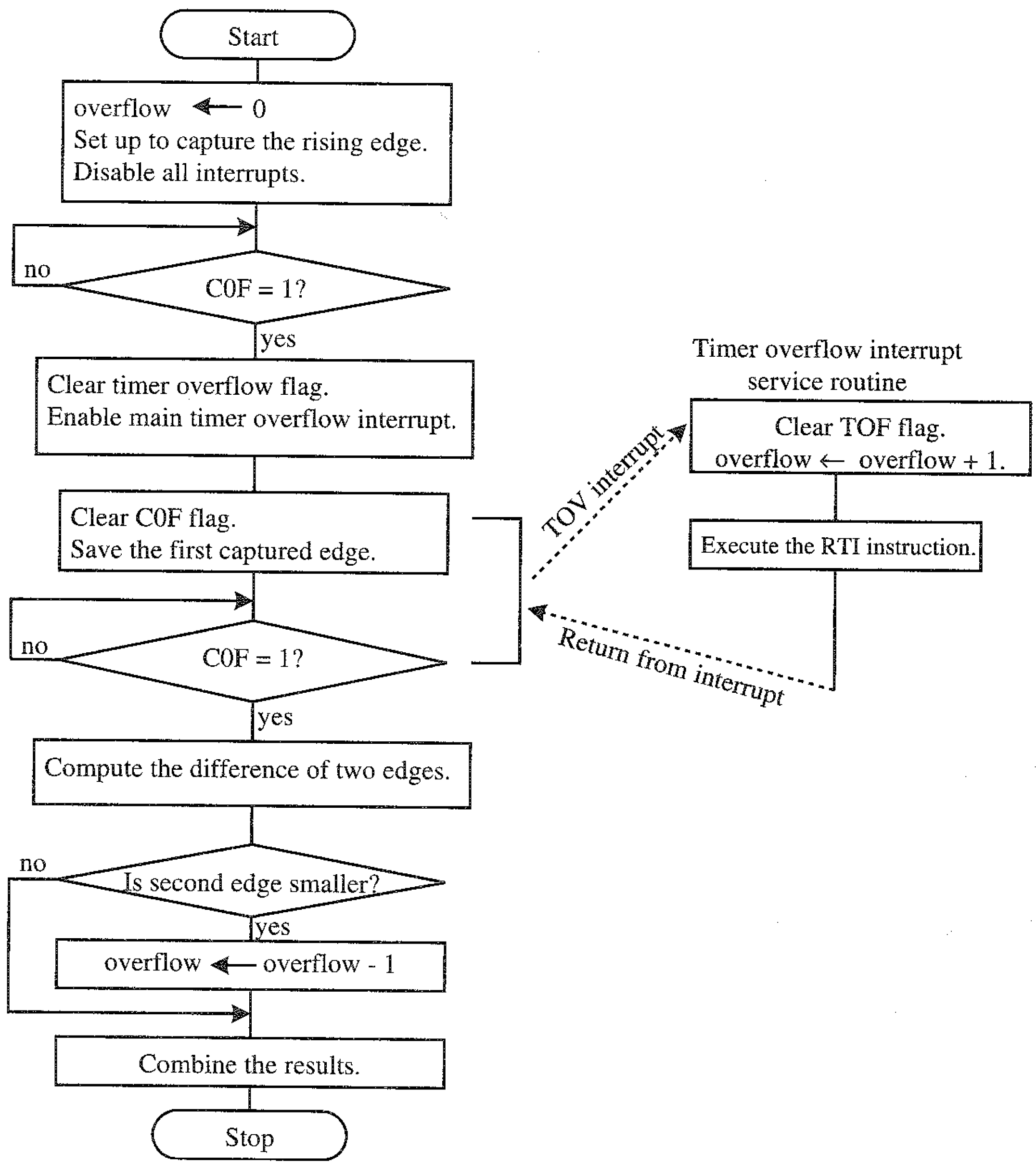


Figure 8.16 ■ Logic flow for measuring pulse width of slow signals

ovcnt = TCNT timer overflow count
edge1 = the captured time of the first edge
edge2 = the captured time of the second edge

The pulse width can be calculated by

case 1 $edge2 \geq edge1$

$$\text{pulse width} = (2^{16} - edge1) + (ovcnt - 1) \times 2^{16} + edge2$$
$$= ovcnt \times 2^{16} + | edge2 - edge1 |$$

case 2 $edge2 < edge\ 1$

$$\text{pulse width} = (ovcnt - 1) \times 2^{16} + | edge2 - edge1 |$$

In case 2, the timer overflows at least once even if the pulse width is shorter than $2^{16} - 1$ clock cycles. Thus, we need to subtract 1 from the timer overflow count in order to get the correct result.

Finally, the pulse width is obtained by computing pulse width x 1.333 μs.

```
#include  "reg9s12.h"

          org     $1000
edge      ds.w    1
ovcnt     ds.w    1
PW        ds.w    1

          org     $2000
          lds     #$2000         ; set up stack pointer
          clr     ovcnt          ; clear the timer overflow counter
          movb    #$90,TSCR      ; enable timer; fast timer flag clear
          movb    #$05,TMSK2     ; disable TCNT interrupt, prescaler=32
          bclr    TIOS,$01       ; select IC0

          movb    #$01,TCTL4     ; set to capture rising edge on PT0 pin
          movb    #01,TFLG1      ; clear C0F flag by writing a 1 to wait for event
          brclr   TFLG1,01,*     ; wait for the first rising edge
          movw    TC0,edge       ; save the first edge; clear the C0F flag auto

          movb    #$80,TFLG2     ; clear TOF flag of the TCNT timer for next event
          bset    TMSK2,$80      ; enable TCNT overflow interrupt
          cli                    ; enable global interrupt

          movb    #$02,TCTL4     ; set to capture the falling edge on PT0 pin
```

```
            brclr   TFLG1,01,*      ; wait for the arrival of the falling edge
            ldd     TC0
            subd    edge1           ; calculate pulse width
            std     PW
            bcc     next            ; is the second edge smaller?
            ldx     ovcnt           ; second edge is smaller, so decrement
            dex                     ; overflow count by 1
            stx     ovcnt           ;     "
next        swi

tov_isr     movb    #$80,TFLG2      ; clear TOF flag for next event to arrive
            ldx     ovcnt
            inx
            stx     ovcnt
            rti

            org     $3E5E
            fdb     tov_isr         ; set up TCNT overflow interrupt vector
            end
```

Measurement Steps

1) Set up the function generator with Hi-Z, 4.7Vpp (i.e., 0-4.7V), square wave, 1 kHz, and 50% duty cycle. (Important: Use the oscilloscope to confirm the setting!)
2) Connect the positive and ground terminals of the function generator to the PT0 and GND socket holes, respectively, using two pieces of wire. The two holes are found around the middle of the left-side (vertical) socket on Dragon12+.
3) Load the program to Dragon12+.
4) Type “g 2000” to make one measurement. The content of D gives the number of clock cycles in hex format.
5) Convert the hex value to decimal value. Then, multiple it with 1.333 μs as the measured pulse width of the “unknown” signal.
6) Set the function generation with a different duty-cycle setting, then repeat Steps 4-6.

For 1 kHz signal with 50% duty cycle, the program gives D = $0177, which is equal to 375 clock cycles. This gives the pulse width of 375 x 1.333 μs = 0.5 ms.

For 1 kHz signal with 30% duty cycle, D = $00E1 = 225 clock cycles, giving the pulse width of 225 x 1.333 μs = 0.3 ms.

For 2 kHz signal with 50% duty cycle, D = $00BB = 187 clock cycles, giving the pulse width of 187 x 1.333 μs = 0.25 ms.

Output Compare Functions

HCS12 has up to 8 OC channels. Each channel consists of

- a 16-bit comparator.
- a 16-bit OC register TC7~0 (also used as IC register).
- an output action pin (PT7~0) that can be pulled high, pulled low, or toggled).
- control logic and an interrupt request circuit.

Operation of the Output-Compare Function

- One of the major applications of the OC function is to trigger an action at a specific time in the future (when the value of the 16-bit TCNT timer equals the value stored in the selected TC register).

- To use an OC function, the user
  1. makes a copy of the current contents of the TCNT register
  2. adds to this copy a value equals to the desired delay
  3. stores the sum into an OC register (TC7~0)

	7	6	5	4	3	2	1	0
	OM7	OL7	OM6	OL6	OM5	OL5	OM4	OL4
value after reset	0	0	0	0	0	0	0	0

(a) TCTL1 register

	7	6	5	4	3	2	1	0
	OM3	OL3	OM2	OL2	OM1	OL1	OM0	OL0
value after reset	0	0	0	0	0	0	0	0

(b) TCTL2 register
read: anytime
write: anytime

OMn	OLn	: output level
0	0	no action (timer disconnected from output pin)
0	1	toggle OCn pin
1	0	clear OCn pin to 0
1	1	set OCn pin to high

Figure 8.18 Timer control register 1 and 2 (TCTL1 & TCTL2)

- The actions that can be activated on an OC pin (PT7~0) by programming the Timer Control registers (TCTL1 and TCTL2) include
  1. pull up to high
  2. pull down to low
  3. toggle

- When either OMx or OLx is 1, the pin associated with OCx becomes an output tied to OCx regardless of the state of the associated DDRT bit.

- The comparator compares the value of the TCNT timer and that of the specified OC register (TC7~0) in every clock cycle. If they are equal, the specified action on the OC pin (PT7~0) is activated and the associated status bit in TFLG1 will be set to 1. An interrupt request will be generated if it is enabled.

- An interrupt may be optionally requested if the associated interrupt enable bit in TMSK1 (or TIE) is set. The same bit will enable either the IC or OC interrupt depending on which one is being selected.

Applications of Output Compare Function

1. An OC function can be programmed to perform a variety of functions, such as generation of a single pulse, a square wave, and a specific delay.

2. To generate square wave, the user may use the OC function to continuously toggle the selected Port T pin with appropriate delay added in each OC operation.

Example: Sound the buzzer on PT5 using channel 5 timer with OC operation.

Prescaler = 128 gives period of each clock cycle (count) = 1/(24MHz/128) = 5.333 μs.

Applying 5000 clock-cycle delay gives 2.667ms of 1 and 2.667ms of 0, this corresponds to the generation of a square wave with a period of 5.33ms and frequency of 18.75Hz.

```
#include "reg9s12.h"

        org     $2000
        lds     #$2000
        movb    #$80,TSCR       ; (=TSCR1) enable timer; clear TOF manually
        movb    #$07,TMSK2      ; (=TSCR2) prescaler=128; free runs
        bset    TIOS,$20        ; set OC at chan 5 (PT5)
        movb    #$04,TCTL1      ; OM5=0 & OL5=1 to enable PT5 toggle
```

```
again   ldd     TCNT              ; get current count of TCNT
        addd    #5000             ; change this number for different tone (pulse width)
        std     TC5               ; add 5000x5.33µs = 26.67ms delay
here    brclr   TFLG1,$20,here    ; wait for TCNT=TC5. If yes, execute next line
        bset    TFLG1,$20         ; write 1 to clear the C5F flag for next round
                                  ; no need of this instruction if set TSCR=#$90
        bra     again             ; forever loop to toggle PT5
        end
```

Example: Generate a delay of 5 seconds using OC6. A LED will be turned on at the beginning and then turned off after 5 seconds.

```
#include "reg9s12.h"

                org     $1000
flag5           rmb     1
counter5        rmb     2

                org     $2000
                lds     #$2000
                movb    #$FF,DDRB       ; set PTB as output for LEDs
                movb    #$FF,DDRP       ; set PTP as output
                movb    #$FF,DDRJ       ; set PTJ as output
                movb    #$00,PTJ        ; PJ1=1 to enable LEDs (in Dragon12+)
                movb    #$FF,PTP        ; turn off 7-segment displays & RGB LEDs

                movb    #$80,TSCR       ; enable TCNT timer; manual TOF clear
                movb    #$40,TIOS       ; set chan 6 for OC operation
                movb    #$40,TIE        ; enable TC6 interrupt
                cli                     ; enable all interrupts

                bset    PortB,$01       ; turn on LED0 at PB0
                clr     counter5        ;
                clr     counter5+1
                clr     flag5
delay           ldaa    flag5           ; wait for flag5 to be set to 1
                beq     delay           ; generate 5s delay
                bclr    PortB,$01       ; turn off LED PB0
                swi

d5s_isr         ldx     counter5        ;
                inx
```

```
            stx     counter5
            cpx     #5000           ; generate 5000x1ms=5s delay
            bne     d1ms            ;
            clr     counter5
            clr     counter5+1
            ldaa    #1
            staa    flag5
            rti

d1ms        ldd     #24000          ; 24,000 x 1/24MHz = 1ms delay
            addd    TC6             ; add to the current count in TC6 to for
            std     TC6             ; round of 1ms delay interrupt
            bset    TFLG1,$40       ; clear C6 flag by writing a 1 to restart
            rti                     ; why use RTI?

            org     $3E62
            fdb     d5s_isr

            end
```

Example: A square-wave generator using OC6. 500Hz square wave is seen on the PB0 LED and 2Hz square wave on the PB7 LED.

Connect PB0 and PB7 to Ch1 and Ch2 of an oscilloscope, the associated square waves can be displayed. Verify the frequencies using the Time Measure function in the scope.

```
#include "reg9s12.h"

            org     $1000
counter     rmb     1

            org     $2000
            lds     #$2000
            movb    #$FF,DDRB       ; set PTB as output for LEDs
            movb    #$FF,DDRP       ; set PTP as output
            movb    #$FF,DDRJ       ; set PTJ as output
            movb    #$00,PTJ        ; PJ1=1 to enable LEDs (in Dragon12+)
            movb    #$FF,PTP        ; turn off 7-segment displays & RGB LEDs

            movb    #$80,TSCR       ; enable TCNT timer; manual TOF clear
            movb    #$40,TIOS       ; set chan 6 for OC operation
            movb    #$40,TIE        ; enable TC6 interrupt
```

```
        cli                     ; enable all interrupts
        clr     counter         ;

loop    ldaa    counter         ; this section toggles PB7 every 250ms;
        cmpa    #250            ; 250ms high and 250 ms low;
        bne     loop            ; gives a square wave with period=500ms
        clr     counter         ; and frequency = 2Hz
        ldaa    PortB           ; PT7 = LED7
        eora    #$80            ; toggle LED7 every 250ms => 2Hz square
        staa    PortB           ;
        jmp     loop

d1msISR inc     counter         ;
        ldd     #24000          ; 24,000 x 1/24MHz = 1ms
        addd    TC6             ; add to the current count in TC6 to for
        std     TC6             ; round of 1ms delay interrupt
        bset    TFLG1,$40       ; clear C6 flag by writing a 1 for next event
        ldaa    PortB           ;
        eora    #$01            ; toggle LED0 every 1ms => 500Hz square
        staa    PortB           ;
        rti

        org     $3E62
        fdb     d1msISR

        end
```

Example: Generate a 1 kHz digital waveform with 30% duty cycle from the PT0 pin.

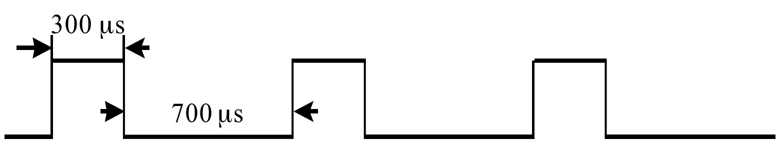


Figure 8.19 1 KHz 30 percent duty cycle waveform

Setting the prescaler to the TCNT to 8, then the period of the clock signal to the TCNT will be 1/(24MHz/8) = 1/3 μs. The numbers of clock cycles that the signal is high and low are 900 and 2100, respectively.

Steps:

1) Pull PT0 to low. A new total count in the TC0 timer is calculated by the current TC0 count + the number of clock cycles for low voltage.
2) When TC0 reaches that new total count, pull PT0 to high.

3) A new total count in the TC0 timer is calculated by the current TC0 count + the number of clock cycles for high voltage.
4) When TC0 reaches that new total count, repeat Step 1.

```
#include "reg9s12.h"

hi_time     equ   900
lo_time     equ   2100

            org    $2000
            movb   #$90,TSCR          ; (=TSCR1) enable timer; auto TOF clear
            movb   #$03,TMSK2         ; (=TSCR2) prescaler=8; no TCNT interrupt
            bset   TIOS,$01           ; enable OC at chan 0 (PT0)

            movb   #$03, TCTL2        ; OM0=1 & OL0=1 to select pull high
                                      ; as pin action; this sets PT0=0V initially
            ldd    TCNT               ; load current count in TCNT timer
loop        addd   #lo_time           ; start an OC0 operation with 700us as delay
            std    TC0                ;
low         brclr  TFLG1,$01,low      ; wait until OC0 pin goes high

            movb   #$02,TCTL2         ; OM0=1 & OL0=0 to select pull low
                                      ; as pin action; this sets PT0 = 5V
            ldd    TC0                ; load current count in TCNT timer
            addd   #hi_time           ; start an OC operation with 300us as delay
            std    TC0                ;
high        brclr  TFLG1,$01,high     ; wait until OC0 pin goes low

            movb   #$03,TCTL2         ; repeat
            ldd    TC0
            bra    loop

            end
```

Possible project: Make a waveform generator that allows a user to select the period, duty cycle, pulse width, etc.

Making Sound Using the Output-Compare Function

A sound can be generated by creating a digital waveform with appropriate frequency and using it to drive a speaker or a buzzer. The simplest song is a two-tone siren.

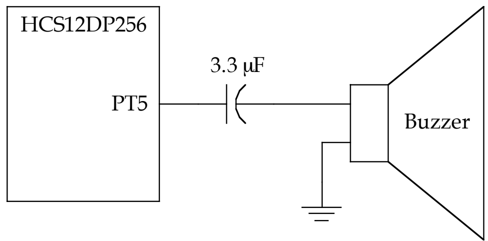


Figure 8.21 Circuit connection for a buzzer

Algorithm for Generating a Siren

Step 1: Enable an OC channel (OC5 in Drgaon12+) to drive the buzzer (or speaker).
Step 2: Start an OC operation with a delay count equal to half the period of the siren.
Step3: Enable the OC interrupt.
Step 4: Set up a 0.5-second loop. During this time, interrupts will be requested many times by the OC function to sound the siren. The ISR simply restarts the OC operation for another round of interrupt.
Step 5: At the end of the loop, choose a different delay count for the OC operation so that the siren sound for a different frequency.
Step 6: Go to Step 4.

Example A two-tone siren that oscillates between 300 Hz and 1200 Hz for 0.5 seconds each tone.

- Set the prescaler to TCNT to 8.
- Delay count for the 300Hz tone = (24000000 ÷ 8) ÷ 300 ÷ 2 = 5000.
- Delay count for the 1200Hz tone = (24000000 ÷ 8) ÷ 1200 ÷ 2 = 1250.

```
#include "reg9s12.h"

            org     $1000
delay       ds.w    1               ; store the delay for OC operation

            org     $2000
            lds     #$2000
            movb    #$90,TSCR           ; enable TCNT, fast timer flag clear
            movb    #$03,TMSK2          ; set main timer prescaler to 8
```

```
                bset    TIOS,$20            ; enable OC5
                movb    #$04,TCTL1          ; select toggle for OC5 pin action

                ldd     #1250               ; 1200Hz
                std     delay               ; use high-frequency delay count first
                ldd     TCNT                ; start timer
                addd    delay               ;       "
                std     TC5                 ;       "
                bset    TIE,$20             ; enable OC5 interrupt
                cli                         ;       "
forever         ldy     #25                 ; tone last for 0.5 second
                jsr     d20ms               ;       "

                movw    #5000,delay         ; switch to low-frequency delay count
                ldy     #25                 ; tone last for 0.5 second
                jsr     d20ms
                movw    #1250,delay         ; switch to high-frequency delay count
                bra     forever

OC5ISR          ldd     TC5                 ; interrupt service routine to toggle OC5 pin
                addd    delay               ; according to the tone frequency in use
                std     TC5                 ;
                rti

d20ms           bset    TIOS,#$01           ; enable OC0
                ldd     TCNT                ;
again           addd    #60000              ; start an output compare operation
                std     TC0                 ; for 60,000/(24MHz/8) = 20 ms delay
wait            brclr   TFLG1,$01,wait
                ldd     TC0
                dbne    y,again
                rts

                org     $3E64               ; from chapter 6
                fdb     OC5ISR

                end
```

Playing Songs Using the Output Compare Function

- Place the frequencies and durations of all notes in a table.
- For every note, uses the OC function to generate the digital waveform with the specified frequency and duration.

Example: US National Anthem.

- Set the prescaler to TCNT to 8.
- Timer count for the 300Hz tone = (24000000 ÷ 8) ÷ 300 ÷ 2 = 5000.

```
#include "reg9s12.h"

G3          equ     7653    ; delay count to generate G3 note (with prescaler=8)
B3          equ     6074    ; delay count to generate B3 note
C4          equ     5733    ; delay count to generate C4 note
C4S         equ     5412    ; delay count to generate C4S (sharp) note
D4          equ     5108    ; delay count to generate D4 note
E4          equ     4551    ; delay count to generate E4 note
F4          equ     4295    ; delay count to generate F4 note
F4S         equ     4054    ; delay count to generate F4S note
G4          equ     3827    ; delay count to generate G4 note
A4          equ     3409    ; delay count to generate A4 note
B4F         equ     3218    ; delay count to generate B4F note
B4          equ     3037    ; delay count to generate B4 note
C5          equ     2867    ; delay count to generate C5 note
D5          equ     2554    ; delay count to generate D5 note
E5          equ     2275    ; delay count to generate E5 note
F5          equ     2148    ; delay count to generate F5 note
ZZ          equ     20      ; delay count to generate an inaudible sound

notes       equ     101     ; number of notes in the song

            org     $1000
delay       ds.w    1               ; store the delay for OC operation
rep_cntds.b 1               ; repeat the song this many times
ip          ds.b    1               ; remaining notes to be played

            org     $2000
            lds     #$2000
            movw    #OC5ISR,$3E64       ; set the interrupt vector
            movb    #$90,TSCR           ; enable TCNT, fast timer flag clear
            movb    #$03,TMSK2          ; set main timer prescaler to 8
            bset    TIOS,$20            ; enable OC5
            movb    #$04,TCTL1          ; select toggle for OC5 pin action

            ldx     #song               ; use as a pointer to score table
            ldy     #duration           ; points to duration table
            movb    #1,rep_cnt          ; play the song twice
            movb    #notes,ip           ; set up the note counter
            movw    2,x+,delay          ; start with zeroth note
```

```
            ldd     TCNT            ; play the first note
            addd    delay           ;       "
            std     TC5             ;       "
            bset    TIE,$20         ; enable OC5 interrupt
            cli                     ;       "

forever     pshy                    ; save duration table pointer in stack
            ldy     0,y             ; get the duration of the current note
            jsr     d10ms           ; play the note for "duration x 10ms"
            puly                    ; get the duration pointer from stack
            iny                     ; move the duration pointer
            iny                     ;       "
            ldd     2,x+            ; get the next note, move pointer
            std     delay           ;       "
            dec     ip              ; if not the last note, play again
            bne     forever         ;
            dec     rep_cnt         ; check how many times left to play song
            beq     done            ; if not finish playing, re-start from 1st note
            ldx     #song           ; pointers and loop count
            ldy     #duration       ;       "
            movb    #notes,ip       ;       "
            movw    0,x,delay       ; get the first note delay count
            ldd     TCNT            ; play the first note
            addd    #delay          ;       "
            std     TC5
            bra     forever
done        swi

OC5ISR      ldd     TC5             ; restart the OC function
            addd    delay
            std     TC5
            rti

; Create a time delay of 10ms Y times (prescaler = 8)
d10ms       bset    TIOS,#$01       ; enable OC0
            ldd     TCNT
again1      addd    #30000          ; start an output compare operation
            std     TC0             ; for 10ms delay
            brclr   TFLG1,C0F,*
            ldd     TC0
            dbne    y,again1
;           bclr    TIOS,OC0        ; disable OC0
            rts

; store the notes of the whole song
song        dc.w    D4,B3,G3,B3,D4,G4,B4,A4,G4,B3,C4S
```

```
            dc.w    D4,ZZ,D4,ZZ,D4,B4,A4,G4,F4S,E4,F4S,G4,ZZ,G4,D4,B3,G3
            dc.w    D4,B3,G3,B3,D4,G4,B4,A4,G4,B3,C4S,D4,ZZ,D4,ZZ,D4
            dc.w    B4,A4,G4,F4S,E4,F4S,G4,ZZ,G4,D4,B3,G3,B4,ZZ,B4
            dc.w    B4,C5,D5,ZZ,D5,C5,B4,A4,B4,C5,ZZ,C5,ZZ,C5,B4,A4,G4
            dc.w    F4S,E4,F4S,G4,B3,C4S,D4,ZZ,D4,G4,ZZ,G4,ZZ,G4,F4S
            dc.w    E4,ZZ,E4,ZZ,E4,A4,C5,B4,A4,G4,ZZ,G4,F4S,D4,ZZ,D4
            dc.w    G4,A4,B4,C5,D5,G4,A4,B4,C5,A4,G4

; each number is multiplied by 10 ms to give the duration of the corresponding note
duration    dc.w    30,10,40,40,40,80,30,10,40,40,40
            dc.w    80, 3,20,3,20,60,20,40,80,20,20,40,3,40,40,40,40
            dc.w    30,10,40,40,40,80,30,10,40,40,40,80,3,20,3,20
            dc.w    60,20,40,80,20,20,40,3,40,40,40,40,20,3,20
            dc.w    40,40,40,3,80,20,20,40,40,40,3,80,3,40,60,20,40
            dc.w     80,20,20,40,40,40,80,3,40,40,3,40,3,20,20
            dc.w    40, 3,40,3,40,40,20,20,20,20,3,40,40,20,3,20
            dc.w    60,20,20,20,80,20,20,60,20,40,80

            end
```

Example: Fur Elise.

```
#include "reg9s12.h"

            org     $1000
spk_tone    rmb     2
sound_dur   rmb     1
xsound_save rmb     2
sound_repeat rmb    1
xsound_beg  rmb     2
sound_start rmb     1
rest_note   rmb     1
count_5ms   rmb     1

; 2400000 / 2 / 261.63 Hz = note count for c3 (with prescaler =1)

c3          equ     45866           ; 261.63 Hz at 24 MHz
c3s         equ     43293           ; 277.18 Hz at 24 MHz
d3          equ     40864           ; 293.66 Hz at 24 MHz
d3s         equ     38569           ; 311.13 Hz at 24 MHz
e3          equ     36404           ; 329.63 Hz at 24 MHz
f3          equ     34361           ; 349.23 Hz at 24 MHz
f3s         equ     32433           ; 369.99 Hz at 24 MHz
```

```
g3      equ     30613       ; 391.99 Hz at 24 MHz
g3s     equ     28894       ; 415.31 Hz at 24 MHz
a3      equ     27273       ; 440.00 Hz at 24 MHz
a3s     equ     25742       ; 466.16 Hz at 24 MHz
b3      equ     24297       ; 493.88 Hz at 24 MHz

c4      equ     22934       ; 523.25 Hz at 24 MHz
c4s     equ     21646       ; 554.37 Hz at 24 MHz
d4      equ     20431       ; 587.33 Hz at 24 MHz
d4s     equ     19285       ; 622.25 Hz at 24 MHz
e4      equ     18202       ; 659.26 Hz at 24 MHz
f4      equ     17181       ; 698.46 Hz at 24 MHz
f4s     equ     16216       ; 739.99 Hz at 24 MHz
g4      equ     15306       ; 783.99 Hz at 24 MHz
g4s     equ     14447       ; 830.61 Hz at 24 MHz
a4      equ     13636       ; 880.00 Hz at 24 MHz
a4s     equ     12871       ; 932.32 Hz at 24 MHz
b4      equ     12149       ; 987.77 Hz at 24 MHz

c5      equ     11467       ; 1046.50 Hz at 24 MHz
c5s     equ     10823       ; 1108.73 Hz at 24 MHz
d5      equ     10216       ; 1174.66 Hz at 24 MHz
d5s     equ     9642        ; 1244.51 Hz at 24 MHz
e5      equ     9101        ; 1318.51 Hz at 24 MHz
f5      equ     8590        ; 1396.91 Hz at 24 MHz
f5s     equ     8108        ; 1479.98 Hz at 24 MHz
g5      equ     7653        ; 1567.98 Hz at 24 MHz
g5s     equ     7225        ; 1661.22 Hz at 24 MHz
a5      equ     6818        ; 1760.00 Hz at 24 MHz
a5s     equ     6435        ; 1864.66 Hz at 24 MHz
b5      equ     6074        ; 1975.53 Hz at 24 MHz

c6      equ     5733        ; 2093.00 Hz at 24 MHz
c6s     equ     5412        ; 2217.46 Hz at 24 MHz
d6      equ     5109        ; 2349.32 Hz at 24 MHz
d6s     equ     4821        ; 2489.02 Hz at 24 MHz
e6      equ     4551        ; 2637.02 Hz at 24 MHz
f6      equ     4295        ; 2793.83 Hz at 24 MHz
f6s     equ     4054        ; 2959.96 Hz at 24 MHz
g6      equ     3827        ; 3135.97 Hz at 24 MHz
g6s     equ     3612        ; 3322.44 Hz at 24 MHz
a6      equ     3409        ; 3520.00 Hz at 24 MHz
a6s     equ     3218        ; 3729.31 Hz at 24 MHz
b6      equ     3037        ; 3951.07 Hz at 24 MHz

c7      equ     2867        ; 4186.01 Hz at 24 MHz
```

```
c7s         equ   2706          ; 4434.92 Hz at 24 MHz
d7          equ   2554          ; 4698.64 Hz at 24 MHz
d7s         equ   2411          ; 4978.03 Hz at 24 MHz
e7          equ   2275          ; 5274.04 Hz at 24 MHz
f7          equ   2148          ; 5587.66 Hz at 24 MHz
f7s         equ   2027          ; 5919.92 Hz at 24 MHz
g7          equ   1913          ; 6271.93 Hz at 24 MHz
g7s         equ   1806          ; 6644.88 Hz at 24 MHz
a7          equ   1705          ; 7040.00 Hz at 24 MHz
a7s         equ   1609          ; 7458.63 Hz at 24 MHz
b7          equ   1519          ; 7902.13 Hz at 24 MHz
c8          equ   1             ; for rest note

note_c      equ   0
note_cs     equ   1
note_d      equ   2
note_ds     equ   3
note_e      equ   4
note_f      equ   5
note_fs     equ   6
note_g      equ   7
note_gs     equ   8
note_a      equ   9
note_as     equ   10
note_b      equ   11

; dur18= 1/8 note,  dur14= 1/4 note,  $FE= rest_note, $FF = end of song
dur18       equ   50
dur14       equ   100

            org   $2000
            lds   #$2000
            movw  #timer6,$3E62
            movw  #timer5,$3E64
            movw  #3429,spk_tone      ; 3500 Hz
            movb  #$80,TSCR           ; enable timer; manual TOF clear
            movb  #$60,TIOS           ; chan 5 & 6 (PT5&6) for OC operation
            movb  #$60,TIE            ; enable OC5&6 interrupts
            clr   count_5ms
            cli

            movb  #$04,TCTL1          ; set toggle OC5 action for speaker
            jsr   start_sound
again       jmp   again

timer6      inc   count_5ms
```

```
            ldaa    count_5ms
            cmpa    #5
            bne     timer3
            clr     count_5ms
            ldaa    sound_start         ; processing every 5ms
            beq     timer3
            ldaa    sound_dur           ; duration
            deca
            staa    sound_dur
            bne     timer3
            ldx     xsound_save

repeat      ldab    0,X
            cmpb    #255
            beq     sound_end
            ldaa    1,X
            cmpa    #255
            beq     sound_end
            staa    sound_dur
            inx
            inx
            stx     xsound_save
            cmpb    #$FE
            bne     not_rest
            ldaa    #1
            staa    rest_note
            movw    #$08,TCTL1          ; turn off OC5 = turn off speaker
            jmp     timer3

not_rest    clr     rest_note
            movb    #$04,TCTL1          ; toggle speaker
            ldx     #note_table
            aslb
            abx
            ldx     0,X
            stx     spk_tone
            jmp     timer3

sound_end   ldaa    sound_repeat
            beq     no_rep
            ldx     xsound_beg
            jmp     repeat

no_rep:     movw    #3429,spk_tone      ; 3500 Hz
            movb    #$08,TCTL1          ; turn off OC5 = turn off speaker
            clr     sound_start
```

```
timer3      ldd     #24000              ; 1 ms time base
            addd    TC6
            std     TC6
            movb    #$40,TFLG1          ; clear C6F flag
            rti

timer5      ldd     spk_tone
            addd    TC5
            std     TC5
            movb    #$20,TFLG1          ; clear C5F flag
            rti

start_sound ldx     #song
            stx     xsound_beg
            ldaa    #1
            staa    sound_repeat
            ldab    0,x
            ldaa    1,x
            staa    sound_dur
            inx
            inx
            stx     xsound_save
            ldx     #note_table
            aslb
            abx
            ldx     0,X
            stx     spk_tone
            ldaa    #1
            staa    sound_start
            rts

note_table:
            fdb     c3,c3s,d3,d3s,e3,f3,f3s,g3,g3s,a3,a3s,b3
            fdb     0,0,0,0                                   ; dummy byte
            fdb     c4,c4s,d4,d4s,e4,f4,f4s,g4,g4s,a4,a4s,b4
            fdb     0,0,0,0                                   ; dummy byte
            fdb     c5,c5s,d5,d5s,e5,f5,f5s,g5,g5s,a5,a5s,b5
            fdb     0,0,0,0                                   ; dummy byte
            fdb     c6,c6s,d6,d6s,e6,f6,f6s,g6,g6s,a6,a6s,b6
            fdb     0,0,0,0                                   ; dummy byte
            fdb     c7,c7s,d7,d7s,e7,f7,f7s,g7,g7s,a7,a7s,b7
            fdb     0,0,0,0                                   ; dummy byte
            fdb     c8

song        fcb     $20+note_e,dur18
```

```
fcb     $20+note_ds,dur18
fcb     $20+note_e,dur18
fcb     $20+note_ds,dur18
fcb     $20+note_e,dur18
fcb     $10+note_b,dur18
fcb     $20+note_d,dur18
fcb     $20+note_c,dur18
fcb     $10+note_a,dur14
fcb     $00+note_e,dur18
fcb     $00+note_a,dur18
fcb     $10+note_c,dur18
fcb     $10+note_e,dur18
fcb     $10+note_a,dur18
fcb     $10+note_b,dur14
fcb     $00+note_gs,dur18
fcb     $10+note_d,dur18
fcb     $10+note_e,dur18
fcb     $10+note_gs,dur18
fcb     $10+note_b,dur18
fcb     $20+note_c,dur14
fcb     $00+note_e,dur18
fcb     $00+note_a,dur18
fcb     $10+note_e,dur18
fcb     $FE,dur14
fcb     255,255

end
```

Possible project: Display the playing notes and durations on the 7-segment or LCD displays. Make a digital piano.

End of Chapter 8