CpE 390: Microprocessor Systems

Lecture 8

68HC12 Timer Functions

Why are Timer Functions Important?

- 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

A Summary of the 68HC12 Timer Functions

Main timer

- 16-bit timer
- Can be started and stopped any time

Input capture function

- Up to 8 channels -- 0 to 7
- Each channel has a 16-bit latch, edge-detection logic, flag bit, and interrupt logic
- Will load the current main timer value into the input capture register when the selected signal edge is detected
- Can be used to measure the signal frequency, period, and pulse width and as time reference

A Summary of the 68HC12 Timer Functions

3. Output compare functions

- Eight channels (OC1…OC5)
- Each channel has a 16-bit comparator, 16-bit register, action pin, interrupt request circuit, and forced-compare function
- Continuously compares the value of the 16-bit compare register with that of the main timer and may optionally
  - trigger an action on a pin
  - generate an interrupt
  - sets a flag in a register.
- Is often used to create a time delay and generate a waveform

4. Real-time interrupt

- Generates periodic interrupts when enabled
- Interrupt period is programmable

A Summary of the 68HC12 Timer Functions

5. Pulse accumulator

- Has an 16-bit counter
- Can be used to measure events, frequency of unknown signal, or the duration of a pulse width

6. Pulse Width Modulation (PWM)

- Can generate periodic waveforms without the intervention of CPU after the duty cycle and frequency have been set up.
- All except the 812A4 have 4 channels of PWM function.

Timer Counter Register (TCNT)

-Required for input capture and output compare functions.

-Must be read in one 16-bit operation in order to obtain the correct value.

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

Timer System Control Register (TSCR)

-The contents of TSCR are shown in Figure 8.1.

- Setting and clearing the bit 7 of TSCR will start and stop the counting of the TCNT.

- Setting the bit 4 will enable fast timer flag clear function, which means a read from input-capture register or a write to an output-compare register will clear the flag. If this bit is clear, then the user must write a one to a timer flag in order to clear it.

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

read: anytime
write: anytime

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

TSBCK -- timer stops while in background mode bit

0 = allows timer to continue running while in background mode
1 = disables timer when MCU is in background debug mode

TFFCA -- timer fast flag clear all bit

0 = allows timer flag clearing to function normally
1 = For TFLG1 ($8E), a read from an input capture or a write to the output compare channel ($90-$9F) causes the corresponding channel flag, CnF, to be cleared.
For TFLG2 ($8F), any access to the TCNT register clears the TOF flag. Any access to the PACNT register clears the PAOVF and PAIF flags in the PAFLG register.

Figure 8.1 Timer system control register (TSCR)

Timer Interrupt Mask Register 2 (TMSK2)

- Bit 7 is the TCNT overflow interrupt enable bit.
- Timer port pin pull-up resistors can be enabled by setting bit 5 of this register.
- Timer port drive can be reduced when necessary by setting bit 4 of TMSK2.
- TCNT can be reset to 0 when TCNT equals TC7 by setting bit 3 of TMSK2
- The clock input to TCNT can be prescaled by a factor by selecting bits 2 to 0 of TMSK2.
- The contents of TMSK2 are shown in Figure 8.2.

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

read: anytime
write: anytime

TOI -- timer overflow interrupt enable bit

0 = interrupt inhibited
1 = interrupt requested when TOF flag is set

PUPT -- timer pullup resistor (when configured for input) enable bit

0 = disable pullup resistor function
1 = enable pullup resistor function

RDPT -- timer drive reduction bit

0 = normal output drive capability
1 = enable output drive reduction function

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 never gets set even though TCNT counts from $0000 to $FFFF.

Figure 8.2 Timer interrupt mask 2 register (TMSK2)

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	reserved
1	1	1	reserved

Timer Interrupt Flag 2 Register (TFLG2)

- Only bit 7 (TOF) is implemented. Bit 7 will be set whenever TCNT overflows.

Input Capture Function

- *Physical time* is often represented by the contents of the main timer.
- The occurrence of an *event* is represented by a signal edge (rising or falling edge).
- The time when an event occurs can be recorded by latching the count of the main timer when a signal edge arrives, as illustrated in Figure 8.3.
- The 68HC12 has eight input capture channels. Each channel has a 16-bit capture register, an input pin, edge-detection logic, and interrupt generation logic.
- Input capture channels share most of the circuit with output compare functions. For this reason, they cannot be enabled simultaneously.
- The selection of input capture and output compare is done by programming the TIOS register.
-

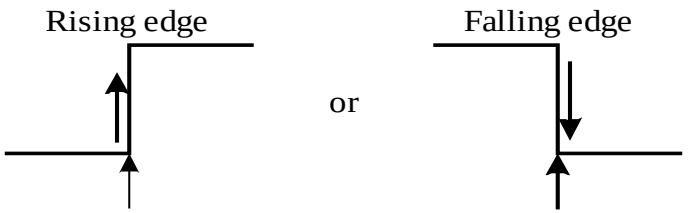


Figure 8.3 Events represented by signal edges

- The contents of the TIOS register are shown in Figure 8.4. Setting a bit selects the output compare function. Otherwise, the input capture function is selected.

	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

read: anytime
write: anytime

IOS[7:0] -- Input capture or output compare channel cinfiguration bits

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

Figure 8.4 Timer input capture/output compare select register

- The following instruction will enable the output-compare channels 7..4 and input-capture channels 3..0:

```
movb    #$F0,TIOS
```

Timer Port Pins

- Each port can be used as a general I/O pin when timer function is not selected.
- Pin 7 can be used as input capture 7, output compare 7 action, and pulse accumulator input.
- When a timer port pin is used as general I/O pin, its direction is configured by the DDRT register.

Timer Control Register 3 and 4

- The signal edge to be captured is selected by 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.

	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

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 Mask Register 1 (TMSK1)

- The arrival of a signal edge may optionally generate an interrupt to the CPU.
- The enabling of the interrupt is controlled by the timer mask register 1.

	7	6	5	4	3	2	1	0
	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I
value after 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 8.6 Timer interrupt mask 1 register (TMSK1)

Timer Interrupt Flag 1 Register (TFLG1)

- Whenever a signal edge arrives, the associated timer interrupt flag will be set to 1.

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

C7F-C0F: input capture/output compare interrupt flag bits
0 = interrupt condition has not occurred
1 = interrupt condition has occurred

Figure 8.7 Timer interrupt flag 1 register

How to clear a timer flag bit?

In normal mode, write a 1 to the flag bit to be cleared

Method 1. Use the BSET instruction with a 1 at the bit position (s) corresponding to the flag (s) to be cleared. For example,

```
        BSET   TFLG1, $01
```

will clear the C0F flag.

Method 2. Use the BCLR instruction with a 0 at the bit position (s) corresponding to the flag (s) to be cleared. For example,

```
   BCLR   TFLG1, $FE
```

will clear the C0F flag.

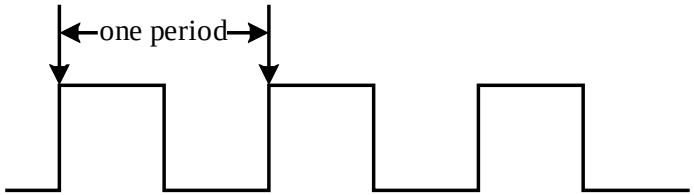
Method 3. Use the MOVB instruction with a 1 at the bit position (s) corresponding to the flag (s) to be cleared. For example,

```
        MOVB   #$01,TFLG1
```

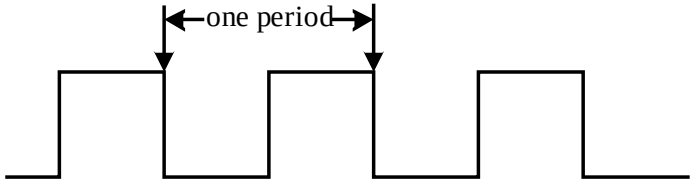
will clear the C0F flag.

Applications of Input Capture function

- Event arrival time recording
- Period measurement: the input capture function captures the main timer values corresponding to two consecutive rising or falling edges



(a) Capture two rising edges



(b) Capture two falling edges

Figure 8.8 Period measurement by capturing two consecutive edges

- Pulse width measurement: capture the rising and falling edges

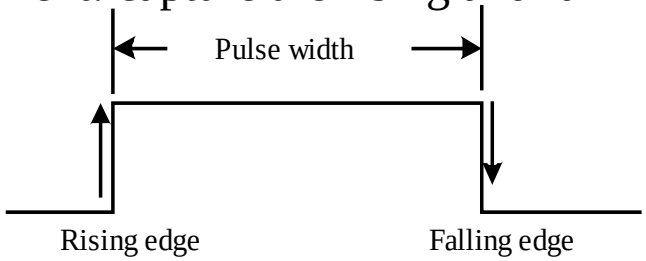


Figure 8.9 Pulse-width measurement using input capture

- Interrupt generation: Each input capture functions can be used as an edge-sensitive interrupt sources.
- Event counting: by counting the number of signal edges arrived during a period

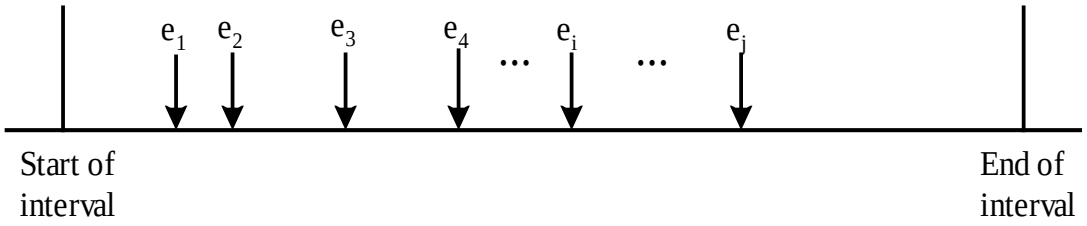


Figure 8.10 Using an input-capture function for event counting

- Time reference: often used in combination with an output compare function

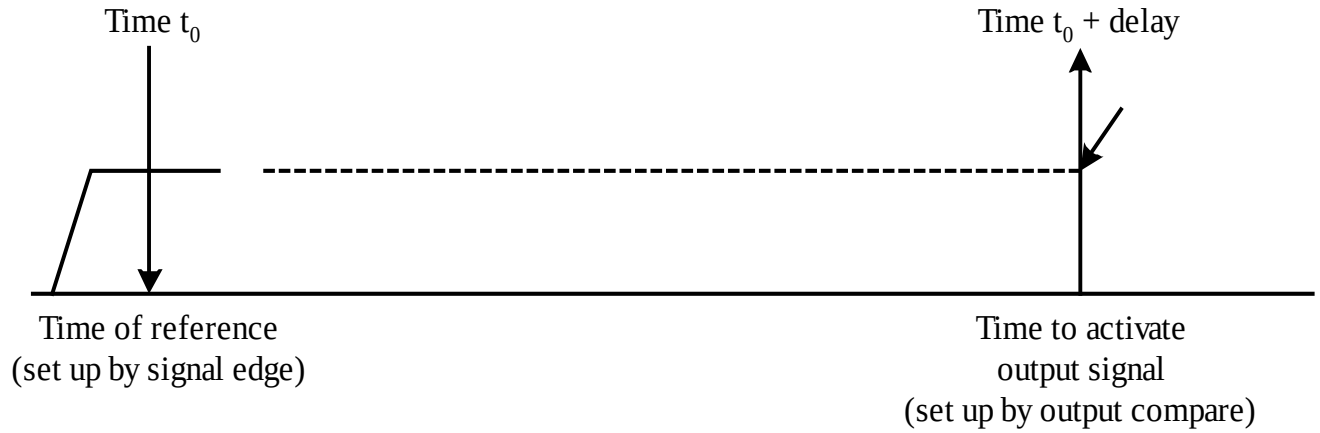
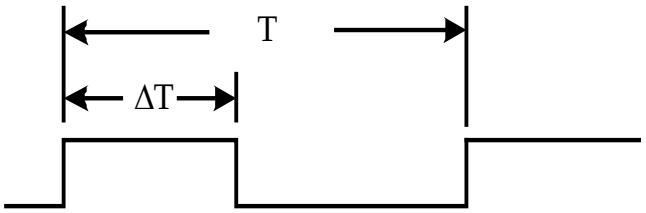


Figure 8.11 A time reference application

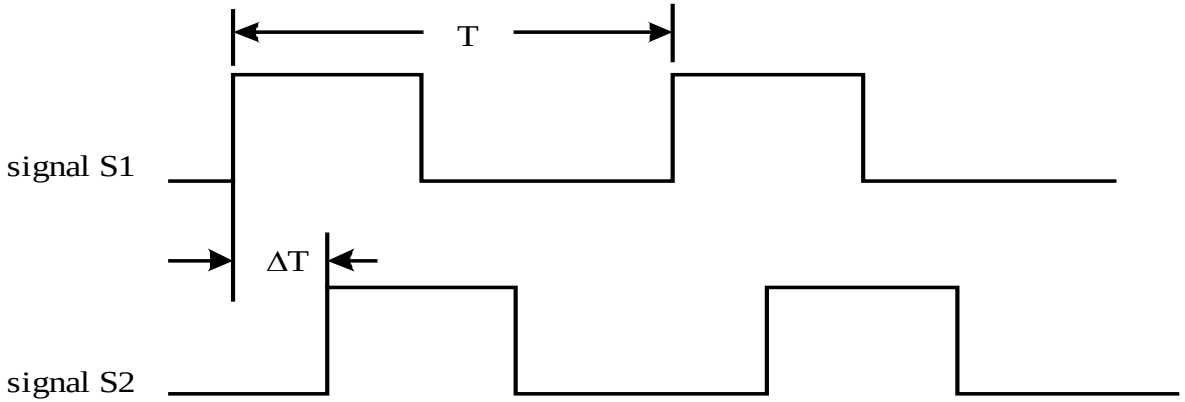
Duty Cycle Measurement



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

Figure 8.12 Definition of duty cycle

Phase Difference Measurement



$$\text{phase difference} = \frac{\Delta T}{T} * 360^{o}$$

Figure 8.13 Phase difference definition for two signals

Example 8.2 Use the input capture channel 0 to measure the period of an unknown signal. The period is known to be shorter than 128 ms. Write a program to set up IC0 to measure its period. The E clock frequency is 8 MHz.

Solution:

1. When the prescale factor is 1, the longest period that can be measured is
   $2^{16} \div 8$ MHz = 8.192 ms
2. When the prescale factor is 16, then longest period measurable is
   $2^{16} \div (8\text{MHz} \div 16)$ = 128 ms

We will set the prescale factor to TCNT to 16.

Registers to be used:

- TSCR to enable timer
- TIOS to set up input capture
- TMASK2 to set up prescale as 16
- TCTL4 to setup capture on rising edge only
- TFLG1 to setup interrupt flag bit
- TOC: 16-bit register to hold the counter value when the selected signal edge arrives at the pin

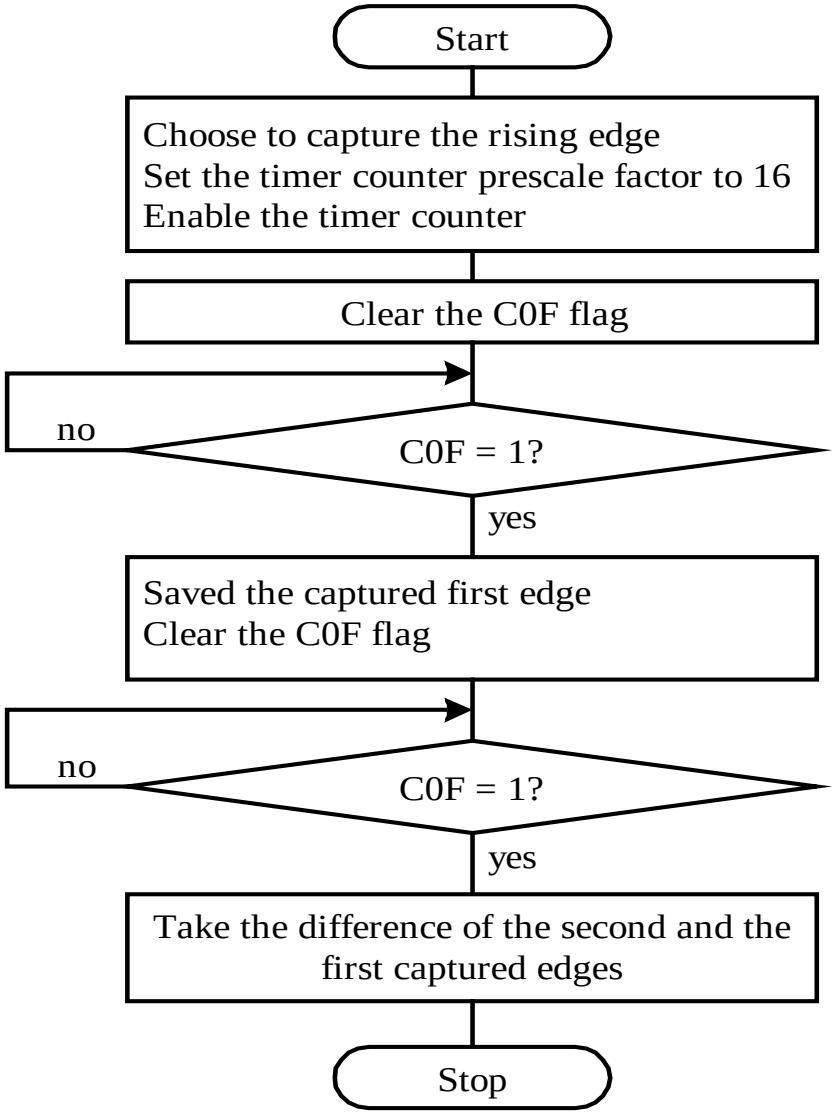


Figure 8.15 Logic flow of period measurement program

Assembly Program for Period Measurement

```
#include  "d:\miniide\hc12.inc"
              org       $800
edge_1st      rmb       2                ; memory to hold the first edge
period        rmb       2                ; memory to store the period
              org       $1000
              movb      #$FE,DDRT        ; configure pin PTO for input
              movb      #$90,TSCR        ; enable timer counter and fast timer flags clear
              bclr      TIOS,$01         ; select input capture 0
              movb      #$24,TMSK2       ; disable TCNT overflow interrupt,  set prescale
                                         ; factor to 16 (check Table 8.1), enable pullup
              movb      #$01,TCTL4       ; choose to capture the rising edge of PT0 pin
              bset      TFLG1,$01        ; clear C0F flag
              brclr     TFLG1,$01,*      ; wait for the arrival of the first rising edge
              ldd       TC0              ; save the  first edge and clear C0F flag
              std       edge_1st
              brclr     TFLG1,$01,*      ; wait for the arrival of the second edge
              ldd       TC0
              subd      edge_1st
              std       period
              swi
              end
```

Example 8.3 Write a program to measure the pulse width of a signal connected to the PT0 pin. The E clock frequency is 8 MHz.

Solution:

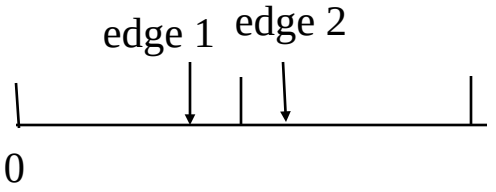
1. Set the prescale factor to TCNT to 8. Each period of TCNT is then set to 1 μs.
2. The pulse width may be longer than 2^{16} clock cycles, we need to keep track of the number of times that the TCNT timer overflows. Let

ovcnt = TCNT counter overflow count
diff = the difference of two consecutive edges
edge1 = the captured time of the first edge
edge2 = the captured time of the second edge

The pulse width can be calculated by the following equations:

Case 1
edge2 ≥ edge1
pulse width = ovcnt 2^{16} + diff

Case 2
edge2 < edge 1
pulse width = (ovcnt – 1) 2^{16} + diff



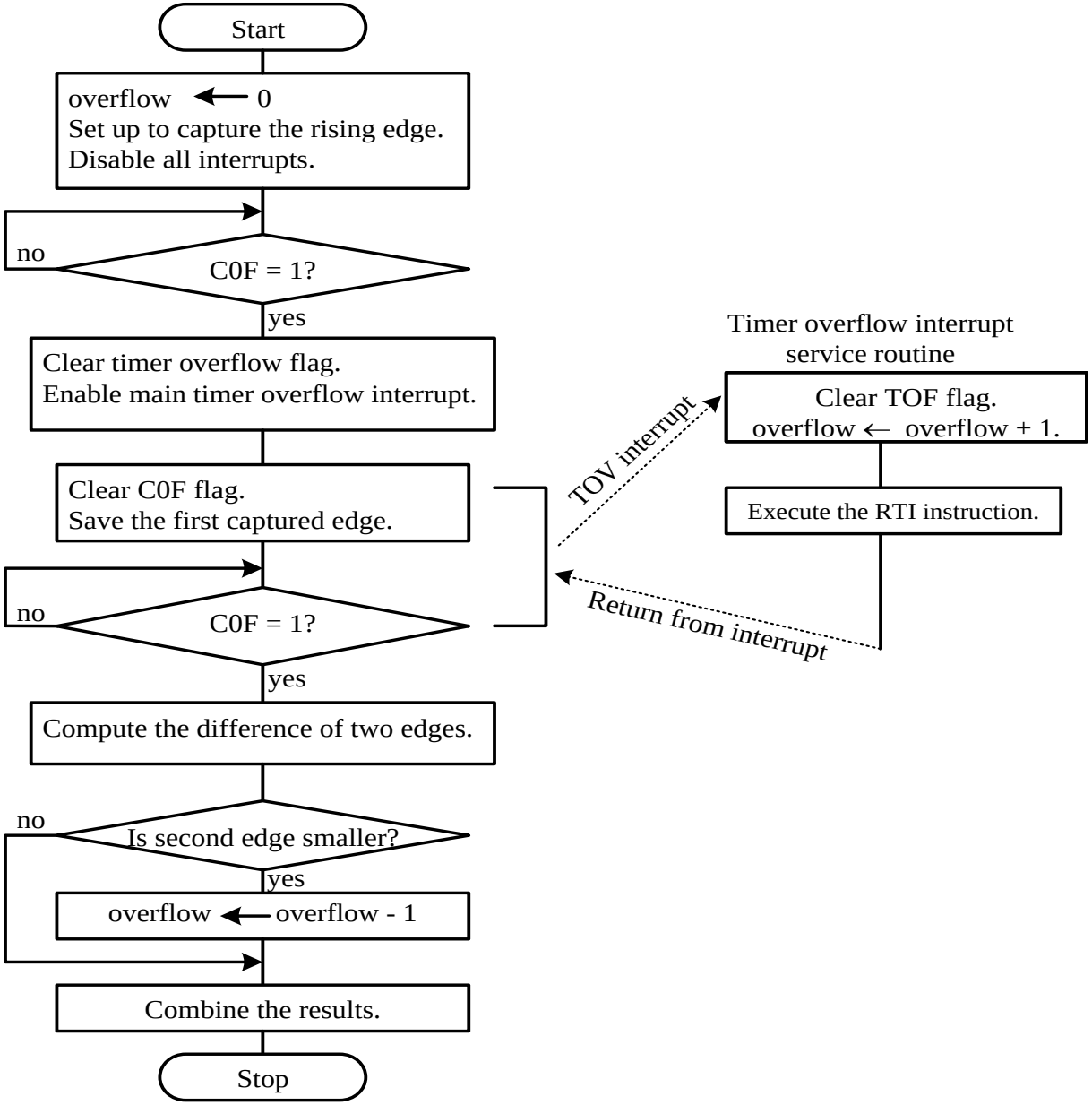


Figure 8.16 Logic flow for measuring pulse width of slow signals

```
#include "d:\miniide\hc12.inc"
setuservector equ     $F69A
tov_vec_no    equ     23
              org     $800
edge1         rmb     2
overflow      rmb     2
pulse_width   rmb     2
              org     $1000
; the following 7 instructions set up TOV interrupt jump vector
              ldd     #tov_isr
              pshd
              ldab    #tov_vec_no
              clra
              ldx     setuservector
              jsr     0,x
              leas    2,sp
              ldd     #0              ; 0->overflow
              std     overflow
              movb    #$90,TSCR    ; enable timer
; disable TCNT overflow interrupt,  enable pull-up, set prescale factor to 16
              movb    #$24,TMSK2
              bclr    TIOS,$01      ; enable input capture 0
              bclr    DDRT,$01
```

```
        movb    #$01,TCTL4      ; capture the rising edge on PT0 pin
        bset    TFLG1,$01       ; clear C0F flag
        brclr   TFLG1,$01,*     ; wait for the arrival of the first rising edge
        ldd     TC0             ; save the captured first edge & clear C0F flag
        std     edge1
        bset    TFLG2,$80       ; clear TOF flag
        bset    TMSK2,$80       ; enable TCNT overflow interrupt
        cli
        movb    #$02,TCTL4      ; capture the falling edge on PT0 pin
        brclr   TFLG1,$01,*     ; wait for the arrival of the falling edge
        ldd     TC0
        subd    edge1
        std     pulse_width
        bcc     next            ; is the second edge smaller?
        ldx     overflow        ; second edge is smaller, so decrement
        dex                     ; overflow count by 1
        stx     overflow
next    swi
tov_isr bset    TFLG2,$80       ; clear TOF flag
        ldx     overflow
        inx
        stx     overflow        ; overflow = overflow + 1
        rti
        end
```

Output Compare Functions

- Up to 8 output compare functions.
- a 16-bit comparator
- a 16-bit compare register TCx (also used as input capture register)
- an output action pin (PTx, can be pulled high, pulled low, or toggled)
- an interrupt request circuit
- a forced-compare function (CFOCx)
- control logic

Operation of the Output-Compare Function

- To trigger an action at a specific time in the future (when the value of the TCNT equals the selected TC register), you have to

1. makes a copy of the current contents of the TCNT register,
2. adds to this copy a value equals to the desired delay, and
3. stores the sum into an output-compare register (TCx, x = 0..7).

- The actions that can be activated on an output compare pin include
  pull up to high; pull down to low; toggle

-- The action is determined by the timer control register 1 & 2 (TCTL1 & TCTL2):

	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 8.17 Timer control register 1 and 2 (TCTL1 & TCTL2)

- A successful compare will set the corresponding flag bit in the TFLG1 register.
- An interrupt may be optionally requested if the associated interrupt enable bit in the TMSK1 register is set.

Example 8.4 Generate an active high 1 KHz digital waveform with 30 percent duty cycle from the PT0 pin. Use the polling method to check the success of the output compare operation. The frequency of the E clock is 8 MHz.

Solution:

- An active high 1 KHz waveform with 30 percent duty cycle is shown in Figure 8.18.

- Setting the prescaler to the TCNT to 8, then the period of the clock signal to the TCNT will be 1 μs. The numbers of clock cycles that the signal is high and low are 300 and 700, respectively.

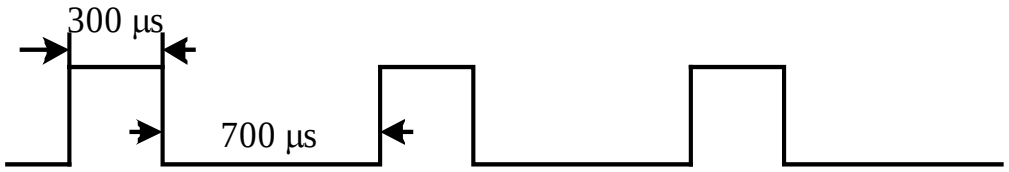


Figure 8.18 1 KHz 30 percent duty cycle waveform

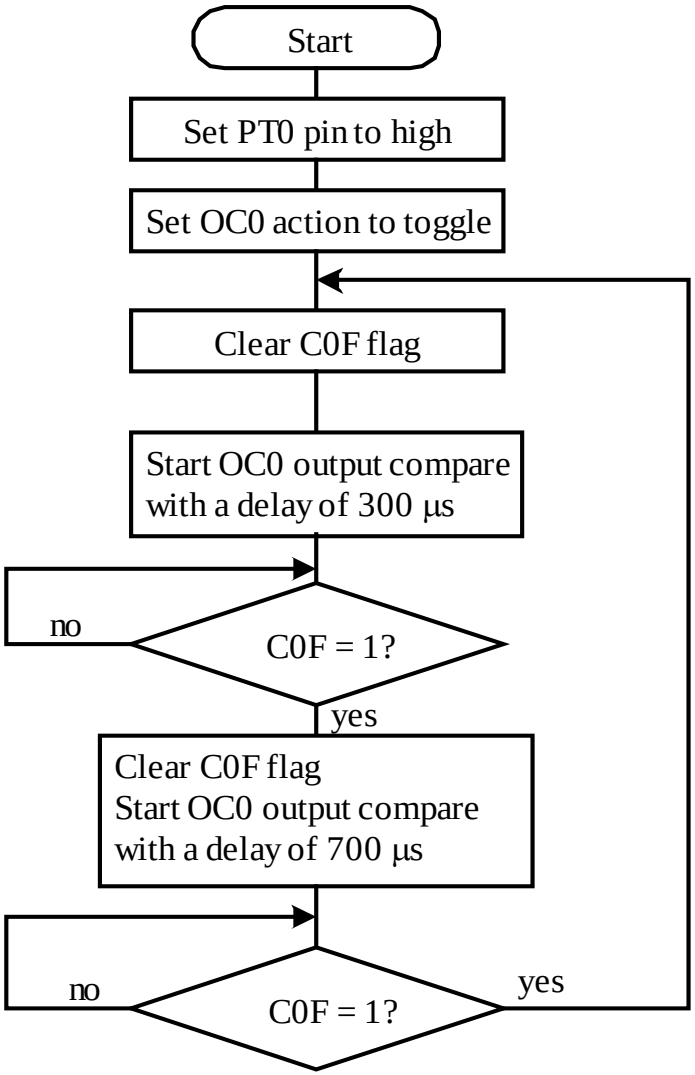


Figure 8.19 The program logic flow for digital waveform generation

```
#include "d:\miniide\hc12.inc"
high_time  equ    300
low_time   equ    700
           org    $1000
           bset   TIOS,$01          ; select OC0 function
           movb   #$03,TMSK2        ; set prescale factor to 8
           movb   #$01,TCTL2        ; select toggle as the output compare action
           movb   #$90,TSCR         ; enable TCNT and fast timer flag clear
           bset   PORTT,$01         ; set PT0 pin to high
           ldd    TCNT
           addd   #high_time
           std    TC0               ; start an OC0 operation with 300 cycles as the delay
high       brclr  TFLG1,$01,high    ; wait until C0F flag is set
           ldd    TC0
           addd   #low_time
           std    TC0               ;
low        brclr  TFLG1,$01,low     ; wait until C0F flag is set
           ldd    TC0
           addd   #high_time
           std    TC0
           bra    high
           end
```

Example 8.5 Write a function to generate a time delay of 1 second. Also write an instruction sequence to test this function assuming that the E clock frequency is 8 MHz.

Solution: There are many ways to create a 1-second time delay using the output-compare function. One method is

1. Set the prescale factor to the TCNT to 8 so that each clock period to the TCNT is 1 μs.
2. Perform 20 output-compare operations with each operation creating 50 ms delay.

The assembly language version of the function:

```
delay_1s  pshx
          movb   #$90,TSCR      ; enable TCNT & fast flags clear
          movb   #$03,TMSK2     ; configure prescale factor to 8
          movb   #$01,TIOS      ; enable OC0
          ldx    #20            ; prepare to perform 20 OC0 actions
          ldd    TCNT
again     addd   #50000         ; start an output compare operation
          std    TC0            ; with 50 ms time delay
          brclr  TFLG1,$01,*
          ldd    TC0
          dbne   x,again
          pulx
          rts
```

Example 8.6 *Combining the use of input-capture and output-compare functions.* Suppose a signal with unknown frequency is connected to the PT0 pin. Write a program to measure the frequency of this signal.

Solution: The method is

1. Use one of the output-compare function to create a one-second time base.
2. Keep track of the number of rising (or falling) edges that arrived at the PT0 pin within one second.

The assembly program is as follows:

```
#include  "d:\miniide\hc12.inc"
CR          equ     $0D
LF          equ     $0A
printf      equ     $F686
            org     $800
oc_cnt      rmb     1
frequency   rmb     2
            org     $1000
; install the interrupt vector to the RAM of CME-12BC demo board
            ldd     #tc0_isr
            pshd
            ldab    #23             ; timer overflow vector number
            clra
            ldx     $F69A
            jsr     0,x
```

```
            leas    2,sp
            movb    #$90,TSCR       ; enable TCNT and fast timer flags clear
            movb    #$00,TMSK2      ; set prescale factor to 1
            movb    #$02,TIOS       ; select OC1 and IC0
            ldaa    #200
            staa    oc_cnt          ; prepare to perform 200 OC1 operation, each
                                    ; creates 5 ms delay and total 1 second
            ldd     #0
            std     frequency       ; initialize frequency count to 0
            movb    #$01,TCTL4      ; prepare to capture PT0 rising edges
            bset    TFLG1,$01       ; clear the C0F flag
            bset    TMSK1,$01       ; enable IC0 interrupt
            cli                     ;          "
            ldd     TCNT
continue    addd    #40000
            std     TC1             ; start the OC1 operation with 5 ms delay
            brclr   TFLG1,$02,*     ; wait for 5 ms
            ldd     TC1
            dec     oc_cnt
            bne     continue
            ldd     frequency
            pshd
```

```
            ldd     #msg
            jsr     [printf,PCR]
            leas    2,sp
            swi
msg         db      CR,LF,"The frequency is %d",CR,LF,CR,LF,0
tc0_isr     ldd     TC0             ; clear C0F flag
            ldx     frequency       ; increment frequency count by 1
            inx                     ;           "
            stx     frequency       ;           "
            rti
            end
```

Using OC7 to Control Multiple Output-Compare Functions

- The OC7 function can control up to 8 output-compare functions at the same time.
- The OC7M register specifies which output-compare functions are controlled by OC7.
- The OC7D register specifies the signal level that each PTx pin is to assume when the TCNT timer equals the TC7 register.

	7	6	5	4	3	2	1	0
	OC7M7	OC7M6	OC7M5	OC7M4	OC7M3	OC7M2	OC7M1	OC7M0
value after reset	0	0	0	0	0	0	0	0

OC7Mn n = 0..7

0 = PTn pin is not affected by OC7 function
1 = A successful OC7 action will override a successful OC6-OC0 compare action during the same cycle and the OCn action taken will depend on the corresponding OC7D bit.

Figure 8.20 Output Compare 7 Mask Register (OC7M)

Example 8.7 What value should be written into OC7M and OC7D if we want pins PT2, PT3, and PT4 to assume the values of 1, 0, and 1, respectively when OC7 compare succeeds?

Solution: Bits 4, 3, and 2 of OC7M must be set to 1, and bits 4, 3, 2, of OC7D should be set to 1, 0, and 1, respectively.
To set up this requirement in assembly language:

```
movb    #$1C,OC7M
movb    #$14,OC7D
```

Using two output-compare functions to control the same pin

- OC7 can be used in conjunction with one or more other output-compare functions to achieve more time flexibility.
- We can generate a digital waveform with a given duty cycle by using the OC7 and any other output compare function.

Example 8.9 Use OC7 and OC0 together to generate a 2 KHz digital waveform with 40 percent duty cycle on the PT0 pin. Assume E clock frequency is 8 MHz.

Solution:

- Set the prescaler to the TCNT to 4 and set the clock frequency to 2 MHz.
- The 40% duty cycle in a 2 KHz waveform corresponds to 400 cycles in one period.
- Use OC7 to pull the PT0 pin to high every 1000 clock cycles.
- Use OC0 to pull the PT0 pin to low 400 clock cycles later.
- Use interrupt-driven method.

The assembly language program is as follows:

```
#include "d:\miniide\hc12.inc"

high_cnt        equ     400
setuservector equ       $F69A
                org     $1000
                lds     #$8000
; set up OC7 interrupt pseudo vector under D-Bug12 monitor
                ldd     #oc7_isr
```

```
            pshd
            ldab    #16             ; OC7 timer overflow vector number
            clra
            ldx     $F69A
            jsr     0,x
            leas    2,sp
; set up OC0 interrupt pseudo vector under D-Bug12 monitor
            ldd     #oc0_isr
            pshd
            ldab    #23             ; OC0 timer overflow vector number
            clra
            ldx     setuservector
            jsr     0,x
            leas    2,sp
            movb    #$90,TSCR       ; enable TCNT and fast flags clear
            movb    #$81,TIOS       ; select OC7 & OC0
            movb    #$01,OC7M       ; allow OC7 to control OC0 pin
            movb    #$01,OC7D       ; OC7 action on PT0 pin is to pull high
            movb    #$22,TMSK2      ; enable pullup resistor, set prescale factor to 4
            movb    #$02,TCTL2      ; select pull low as the OC0 action
            movb    #$81,TMSK1      ; enable OC7 and OC0 to interrupt
            ldd     TCNT
            addd    #1000
```

```
        std     TC7             ; start an OC7 operation
        addd    #high_cnt
        std     TC0             ; start an OC0 operation
        cli                     ; enable interrupt
loop    bra     loop            ; infinite loop to wait for interrupt
        swi

oc7_isr ldd     TC7             ; start the next OC7 action with 1000
        addd    #1000           ; clock cycles delay, also clear the C7F
        std     TC7             ; flag
        rti
oc0_isr ldd     TC0             ; start the next OC0 action with 1000
        addd    #1000           ; clock cycles delay, also clear the C0F
        std     TC0             ; flag
        rti
        end
```

Forced Output-Compare

- Useful when the user requires the output compare to succeed immediately after being started.
- Write a 1 to the corresponding bit of the CFORC register to force an output compare operation.
- The forced output compare operation only causes pin action. Neither the flag is set to 1 nor the interrupt is generated.

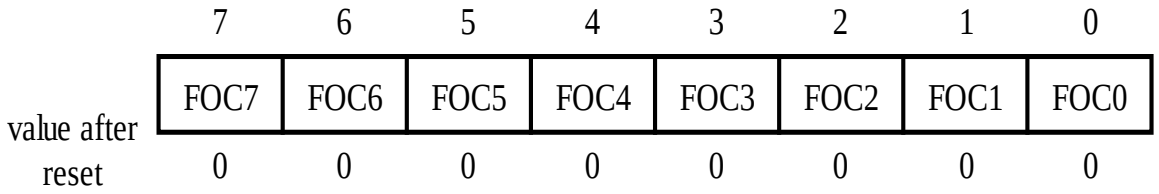


Figure 8.23 Contents of the CFORC register

Example 8.10 Suppose the contents of the TCTL1 and TCTL2 registers are $D6 and $6E, respectively. The contents of the TFLG1 are $00. What would occur on pins PT7 to PT0 on the next clock cycle if the value $7F is written into the CFORC register?

Solution:

- TCTL1 & TCTL2 configure the output-compare channels 6 to 0 actions as shown in Table 8.2.
- Since [TFLG1] = 0, none of the started output-compare operations have succeeded.
- The actions specified in Table 8.2 will be forced immediately.

Table 8.2 Pin actions on PT7-PT0 pins

Register	Bit positions	Value	Action to be triggered
TCTL1			
	7 6	1 1	set the PT7 pin to high
	5 4	0 1	toggle the PT6 pin
	3 2	0 1	toggle the PT5 pin
	1 0	1 0	pull the PT4 pin to low
TCTL2			
	7 6	0 1	toggle the PT3 pin
	5 4	1 0	pull the PT2 pin to low
	3 2	1 1	set the PT1 pin to high
	1 0	1 0	pull the PT0 pin to low

Real-Time Interrupt (RTI)

- Will generate periodic interrupts if enabled.
- Used to remind the 68HC12 to perform routine tasks in many applications.
- Configured by the RTICTL register. Contents shown in Figure 8.24.

	7	6	5	4	3	2	1	0
	RTIE	RSWAI	RSBCK	0	RTBYP	RTR2	RTR1	RTR0
value after reset	0	0	0	0	0	0	0	0

RTIE: real-time interrupt enable bit
- 0 = disable RTI interrupt
- 1 = enable RTI interrupt

RSWAI: RTI and COP stop while in wait bit
- 0 = allows the RTI and COP to continue running in wait.
- 1 = disable both the RTI and COP when the part goes into wait.

RSBCK: RTI and COP stop while in background debug mode bit
- 0 = allows the RTI and COP to continue running while in background mode.
- 1 = disables RTI and COP when the device is in background debug mode (useful for emulation).

RTBYP: real-time interrupt divider chain bypass bit
- 0 = divider chain functions normally
- 1 = divider chain is bypassed, allows faster testing. The divider chain is normally P divided by 8192, when bypass becomes P divided by 4.

RTR2..RTR0: real-time interrupt rate select bits. The rate selection is shown in Table 8.3.

Figure 8.24 Real-time interrupt control register (RTICTL)

Table 8.3 Real-time interrupt rates

RTR2	RTR1	RTR0	divide E by	timeout period E = 4.0 MHz	timeout period E = 8.0 MHz
0	0	0	off	off	off
0	0	1	2^{13}	2.048 ms	1.024 ms
0	1	0	2^{14}	4.096 ms	2.048 ms
0	1	1	2^{15}	8.192 ms	4.096 ms
1	0	0	2^{16}	16.384 ms	8.192 ms
1	0	1	2^{17}	32.768 ms	16.384 ms
1	1	0	2^{18}	65.536 ms	32.768 ms
1	1	1	2^{19}	131.72 ms	65.536 ms

Note: $2^{13} / 8{,}000{,}000 = 8192 / 8{,}000{,}000 = 1.024 ms$

The Pulse Accumulator

-16-bit pulse accumulator (PACNT)

-Two operation modes: *event counting* (count the events) and *gated accumulation* (measure the duration of a single pulse) modes

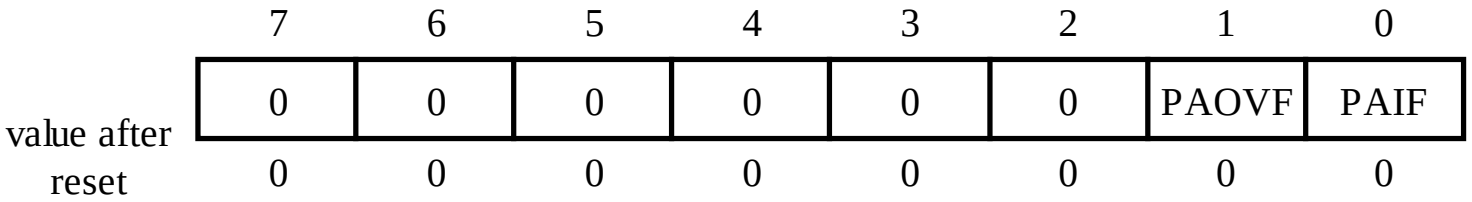
-PACNT is clocked by the PAI input in event counting mode.

-PACNT is clocked by the E-divided-by-64 clock in gated accumulation mode

-The PAI pin (PA7 pin) must be configured for input to enable pulse accumulator

-There are two interrupt sources: *PAI pin edge* and the *PACNT overflow*

- Three registers are related to the PA operation: PAFLG, PACTL, PACNT



PAOVF -- pulse accumulator overflow flag

This flag is set when PACNT overflows from $FFFF to $0000 and can be cleared by writing a 1 to it.

PAIF -- PAI pin edge flag

When in event counter mode, this bit is set when the selected edge on the PAI pin is detected.

When in gated accumulator mode, the selected trailing edge sets this flag.

Figure 8.26 Pulse accumulator flag register (PAFLG)

	7	6	5	4	3	2	1	0
	0	PAEN	PAMOD	PEDGE	CLK1	CLK0	PAOVI	PAI
value after reset	0	0	0	0	0	0	0	0

PAEN -- Pulse accumulator system enable bit
- 0 = disable
- 1 = enable

PAMOD -- pulse accumulator mode bit
- 0 = event counter mode
- 1 = gated time accumulation mode

PEDGE -- pulse accumulator edge control bit

For PAMOD = 0 (event counter mode)
- 0 = falling edges on the PAI pin cause the count to increment
- 1 = rising edges on the PAI pin cause the count to decrement

For PAMOD = 1 (gated time accumulation mode)
- 0 = PAI pin high enables E÷64 clock to pulse accumulator and the trailing falling edge on the PAI pin sets the PAIF flag
- 1 = PAI pin low enables E÷64 clock to pulse accumulator and the trailing rising edge on the PAI pin sets the PAIF flag

CLK1 and CLK0 -- clock select bits
- 00 = use timer prescaler clock as timer counter clock
- 01 = use PACLK as input to timer counter (TCNT) clock
- 10 = use PACLK/256 as timer counter clock frequency
- 11 = use PACLK/65536 as timer counter clock frequency

PAOVI -- pulse accumulator overflow interrupt enable bit
- 0 = disable
- 1 = enable

PAI -- PAI pin interrupt enable bit
- 0 = disabled
- 1 = enabled

Figure 8.25 Pulse accumulator control register (PACTL)

Example 8.11 ***Interrupt after N events***. Events are converted into signal edges and are connected to the PAI pin. N is smaller than 65535. Write a program to interrupt the 68HC12 after N event.

Solution: Write the two's complement of N into the PACNT register and enable the interrupt, then it will interrupt the CPU after N events.

```
#include   "d:\miniide\hc12.inc"
N               equ     1350
setuservector equ       $F69A
                org     $1000
; set up PACNT overflow interrupt pseudo vector for D-Bug12 monitor
                ldd     #paov_isr
                pshd
                ldab    #14
                clra
                ldx     setuservector
                jsr     0,x
                leas    2,sp
```

```
; the next five instructions place the 2's complement in PACNT
            ldd     #N
            coma
            comb
            addd    #1
            std     PACNT
; configure PA function: enable PA, select event counting mode, rising edge
; of PAI signal increments the PACNT counter
            movb    #$52,PACTL
            cli                     ; enable PAOV interrupt
            …
            swi
paov_isr    movb    #$02,PAFLG ; clear the PAOVF flag
            end
```

- The drawback of using the input-capture function to measure the frequency of a signal is that the interrupt overhead of every input signal edge sets the upper limit of the frequency that can be measured.

- Use the pulse accumulator system to measure the frequency can dramatically reduces the incurred interrupt overhead.

Procedure for Using PA to Measure Frequency

Step 1: Set up the pulse accumulator system to operate in event counter mode.

Step 2: Connect the unknown signal to the PAI (PT7) pin.

Step 3: Use one of the output-compare functions to create a one-second time interval.

Step 4: Use a memory location to keep track of the number of pulse accumulator counter overflow interrupts.

Step 5: Enable the PAOV interrupt.

Step 6: Disable the PAOV interrupt at the end of one second.

Example 8.12 Write a program to measure the frequency of a signal connected to the PAI pin.

Solution:

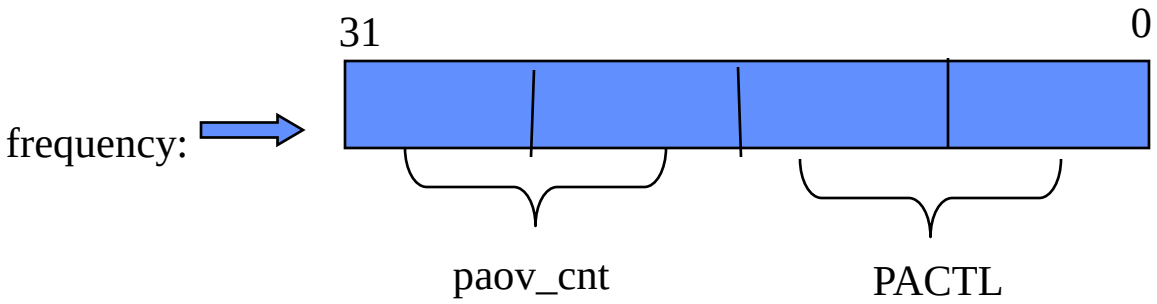
- Use OC0 to create a one-second time base.
- Keep track of PACNT overflow count.
- At the end of one second, the frequency is equal to the following expression:

frequency = paov_cnt × 2_{16} + PACNT

```
#include   "d:\miniide\hc12.inc"
setuservector equ   $F69A
                    org       $800
oc_cnt      rmb     1
paov_cnt    rmb     2         ; use to keep track PACNT overflow count
frequency   rmb     4         ; hold the signal frequency
```

```
            org     $1000
; set up PACNT overflow interrupt pseudo vector for D-Bug12 monitor
            ldd     #paov_isr
            pshd
            ldab    #14
            clra
            ldx     setuservector
            jsr     0,x
            leas    2,sp
            ldaa    #200
            staa    oc_cnt          ; prepare to perform OC0 actions
            ldd     #0
            std     PACNT           ; let PACNT count up from 0
            std     paov_cnt        ; initialize PACNT overflow count to 0
            std     frequency       ; initialize frequency to 0
            std     frequency+2     ;           "
            movb    #$90,TSCR       ; enable TCNT and fast flag clear
            bset    TIOS,$01        ; select OC0 function
            movb    #$20,TMSK2    ; enable pullup resistor & set prescale factor to 1
            movb    #$00,DDRT      ; configure all timer pins for input
; configure PA function: enable PA, select event counting mode, rising edge
; of PAI signal increments the PACNT counter, enable PAOV interrupt
            movb    #$52,PACTL
            cli                     ; enable PAOV interrupt
```

```
            ldd     TCNT
sec_loop    addd    #50000
            std     TC0
            brclr   TFLG1,$01,*
            ldd     TC0
            dec     oc_cnt
            bne     sec_loop
            movb    #0,PACTL        ; disable PA function
            sei                     ; disable interrupt
            ldd     PACNT
            std     frequency+2
            ldd     paov_cnt
            std     frequency
            swi
paov_isr    movb    #$02,PAFLG      ; clear the PAOVF flag
            ldx     paov_cnt        ; increment PACNT overflow
            inx                     ; count by 1
            stx     paov_cnt        ;           "
            end
```



Using the PA Function to Measure Pulse Width

Step 1: Select gated time accumulation mode, and initialize PACNT to 0.
Step 2: Select the falling edge as the active edge, which will enable TACNT to count when the PAI pin is high.
Step 3: Enable the PAI active edge interrupt and wait for the arrival of the active edge of PAI.
Step 4: Stop the pulse accumulator counter when the interrupt arrives.

To measure long pulse, we need to keep track of PA overflow:

$$\text{pulse\_width} = [(2^{16} \quad \text{paov\_cnt}) + \text{PACNT}] \quad 64T_E$$

Example 8.13 Write a program to measure the duration of an unknown signal connected to the PAI pin.

Solution: The assembly program is as follows:

```
#include   "d:\miniide\hc12.inc"
setuservector equ      $F69A
              org      $800
paov_cnt      rmb      1              ; use to keep track of the PACNT overflow count
pulse_width   rmb      3              ; hold the signal frequency
              org      $1000
; set up PACNT overflow interrupt pseudo vector for D-Bug12 monitor
              ldd      #paov_isr
              pshd
              ldab     #14
              clra
              ldx      setuservector
              jsr      0,x
              leas     2,sp
              ldd      #0
              std      PACNT          ; let PACNT count up from 0
              clr      paov_cnt       ; initialize PACNT overflow count to 0
              movb     #$20,TMSK2     ; enable timer port pullup resistor
; configure PA function: enable PA, select gated time accumulator mode, high level
; of PAI signal enables PACNT counter, enable PAOV interrupt
              movb     #$72,PACTL
              bclr     DDRT,$80       ; configure PAI pin for input
              cli                     ; enable PAOV interrupt
              brclr    PAFLG,$01,*    ; wait for the arrival of the falling edge of PAI
```

```
        movb    #0,PACTL    ; disable PA function
        sei                 ; disable interrupt
        ldd     PACNT
        std     pulse_width+1
        ldaa    paov_cnt
        staa    pulse_width
        swi
paov_isr movb   #$02,PAFLG  ; clear the PAOVF flag
        inc     paov_cnt    ; increment PACNT overflow count by 1
        end
```

Homework #7

- See course website: http://390.revan.us
  - click homework tab
- Please submit a hard copy