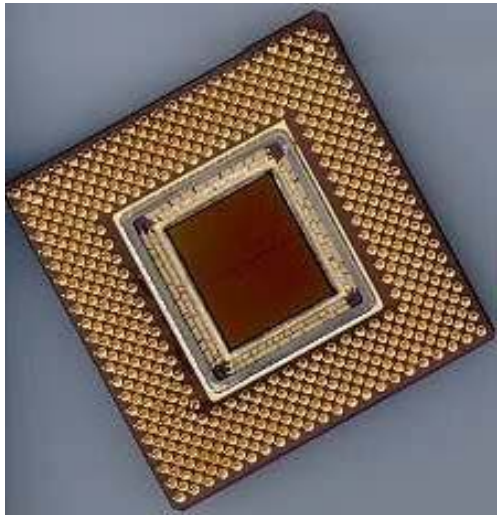
Tennessee Tech UNIVERSITY

Electrical & Computer Engineering



Chapter 5

HCS12 Timer Functions

ECE 3120

Dr. Mohamed Mahmoud

http://iweb.tntech.edu/mmahmoud/
mmahmoud@tntech.edu

Outline

5.1 Timer

5.2 Input Capture and Output Compare Functions

5.3 Applications on Input Capture Function

5.4 Applications on Output Compare Function

Why are Timer Functions Important?

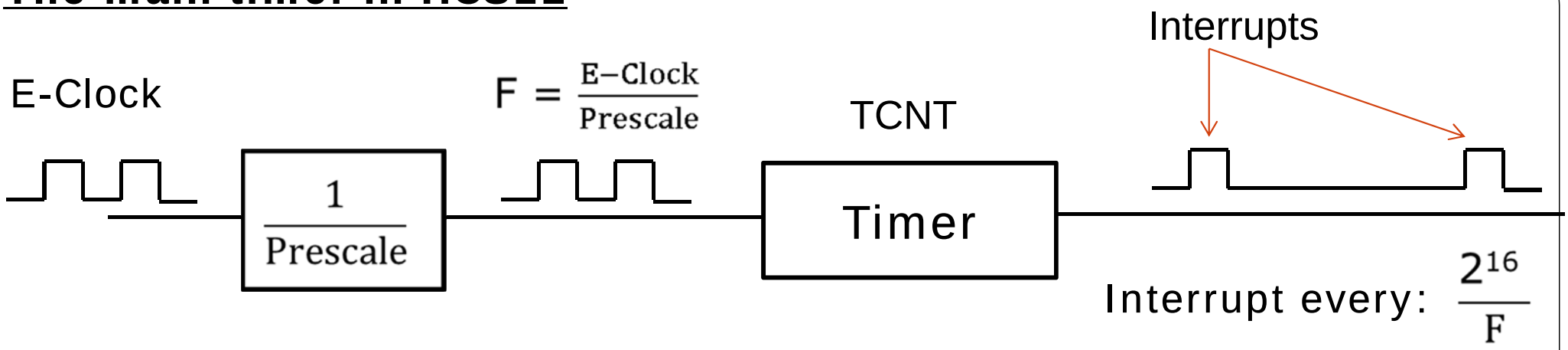
- Many useful applications are difficult to implement without a dedicated timer function.

- Examples:
  - Precise 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

<u>These applications can be done by the timer's circuit without the direct involvement of the CPU</u>

These applications will be discussed in this chapter.

The main timer in HCS12



- Timer Counter Register (TCNT): 16-bit counter timer
- It counts from 0000 to FFFF, then it rolls to 0000 and makes interrupt.
- One clock is needed to increment the counter by 1.
- It can make interrupt every 2^{16} counts (or clocks).

If E-Clock = 24MHz

Min. prescale value = 1 --→ Interrupt every 2.73 ms

Max. prescale value = 128 --→ Interrupt every 349.53 ms

To do an action every longer time, take action every n interrupts

Ex. when prescale = 128 and an action is taken every 4 interrupts --→ the action is taken every 1.4 second

How to program the timer

1- Interrupt vector: $FFDE

```
org $FFDE
dc.w Timer_ISR   ;load timer interrupt routine vector
```

Table 6.1 Interrupt vector map

Vector address	Interrupt source	CCR mask	Local Enable	HPRIO value to elevate to highest I bit
$FFFE	Reset	none	none	-
$FFFC	Clock monitor reset	none	COPCTL(CME,FCME)	-
$FFFA	COP failure reset	none	COP rate selected	-
$FFF8	Unimplemented instruction trap	none	none	-
$FFF6	SWI	none	none	-
$FFF4	XIRQ	X bit	none	-
$FFF2	IRQ	I bit	INTCR(IRQEN)	$F2
$FFF0	Real time interrupt	I bit	RTICTL(RTIE)	$F0
$FFEE	Timer channel 0	I bit	TMSK1(C0I)	$EE
$FFEC	Timer channel 1	I bit	TMSK1(C1I)	$EC
$FFEA	Timer channel 2	I bit	TMSK1(C2I)	$EA
$FFE8	Timer channel 3	I bit	TMSK1(C3I)	$E8
$FFE6	Timer channel 4	I bit	TMSK1(C4I)	$E6
$FFE4	Timer channel 5	I bit	TMSK1(C5I)	$E4
$FFE2	Timer channel 6	I bit	TMSK1(C6I)	$E2
$FFE0	Timer channel 7	I bit	TMSK1(C7I)	$E0
$FFDE	Timer overflow	I bit	TMSK2(TOI)	$DE

Priority

2- Local timer flow interrupt enable bit

- Timer System Control Register 2 (TSCR2)
- Bit 7 is the timer overflow interrupt enable bit.
- Bits 0, 1, and 2 are used to select the prescale.

	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

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

TOI enables/disables interrupts, but it does not start/stop counting

When interrupt is disabled, the timer counts but an interrupt is not generated when it rolls from FFFF to 0000

Start/stop counting

Timer System Control Register 1 (TSCR1)

	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

- Setting and clearing TEN (bit 7 of TSCR1) will start and stop the counting of the TCNT.

TEN starts/stops counting. It does not enable/disable interrupts but when the counter stops counting, the interrupts stop automatically because it will not roll from FFFF to 0000

3- Timer Interrupt Flag

- Timer Interrupt Flag 2 Register (TFLG2)

- Bit 7 (TOF) will be set whenever TCNT overflows from $FFFF to $0000 – when the timer makes interrupt.

Two ways to clear TOF

Timer System Control Register 1 (TSCR1)

	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

1- TFFCA (bit 4) = 0: The user must write one to TOF flag to clear it.

```
movb    #%10000000,TFLG2
```

2- TFFCA (bit 4) = 1: Enable fast timer flag clear function.

TOF is automatically cleared when you access (read/write) TCNT or timer functions' registers (input capture read -or- output compare write).

Helps reduce software overhead

- Just like the RTI system, we can make a timer out of a counter attached to the system clock

- But, unlike the RTI system, we attach the counter to some additional hardware to create some additional functionality

- These functions are: Input capture and Output compare

Outline

5.1 Timer

5.2 Input Capture and Output Compare Functions

5.3 Applications on Input Capture Function

5.4 Applications on Output Compare Function

The HCS12 Timer System

1- 16-bit timer counter (TCNT)

2- Eight channels programmed as input capture or output compare. The pins of these channels are port T.

3- 16-bit Pulse Accumulator A

<u>11 interrupt sources</u>

1 for Timer overflow

8 for channels

2 for Pulse Accumulator A

When a channel is not used, you can use its pin as general input or output port pin.

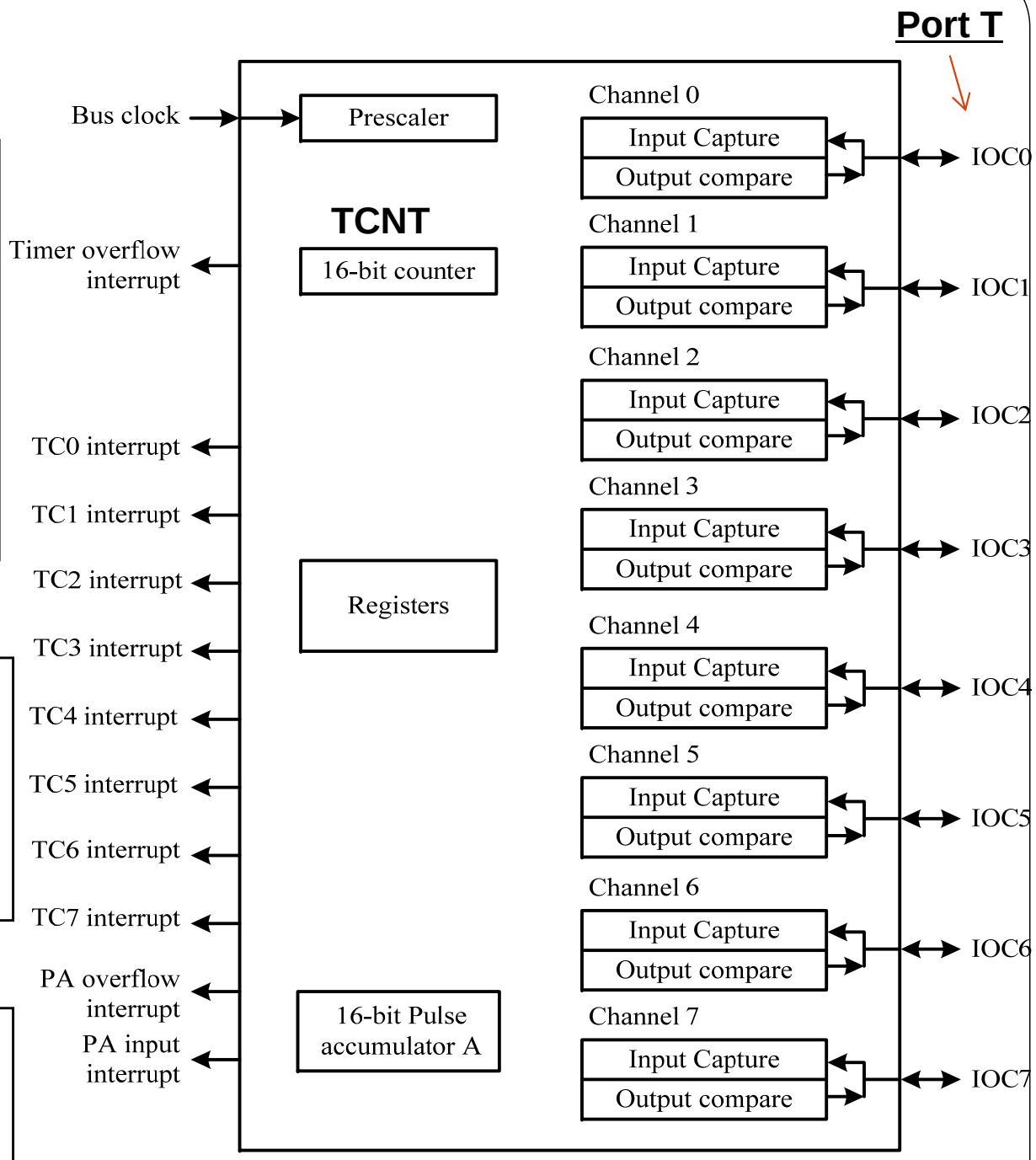
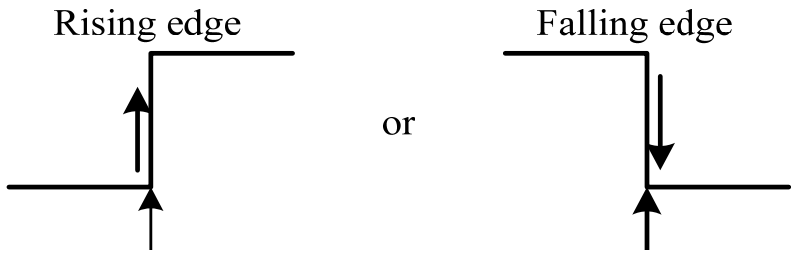


Figure 8.1 HCS12 Standard Timer (TIM) block diagram

Input Capture Functions

- When an event happens, the event time (the content of the timer (TCNT)) is recorded in a register and makes interrupt (if enabled).
- The occurrence of an event is represented by a signal edge (rising or falling edge).



- Each channel has a 16-bit capture register (TCx and x = {0, 1, .. ,7}) and an input pin on port T.
- If the time of event 1 is 0005 and the time of event 2 is 0009, then
  1- Event 1 happened before event 2
  2- The time difference between the 2 events = (0009 – 0005) * (1/F) where F is the timer clock and (0009 – 0005) is the number of clocks between the two events.
  3- Event 1 happened at time (0005 * 1/F) from starting the timer
  4- Event 2 happened at time (0009 * 1/F) from starting the timer

Output Compare Functions

1- Write a value in a compare register (TCx and x = {0, 1, .. ,7})

2- Once the value of the main timer (TCNT) = the value in TCx:

   1- Interrupt happens (if enabled) and action is taken.

   2- The action can be outputting 0, 1 or toggle the PTx pin

- The HCS12 has eight output compare channels. Each channel has a 16-bit compare register (TCx) and an output action pin (PTx) in port T.

- One of the applications of the output-compare function is to trigger an action at a specific time in the future.

- Example, if you wanna do an action after 166.67 ms, the action should be taken after 166.67ms/(1/24MHz) = 4000 clocks with assuming E-clock = 24MHz and prescale = 1.

An output-compare register (TCx, x = 0..7) = the current contents of the TCNT register + a value equal to the desired delay (4000)

Select either Input-Capture or Output-Compare Function

- The same pins can be used for Input-Capture and Output-Compare Functions
- Only one function can be selected at a time.
- Timer input capture/output compare (TIOS) register is programmed to select either input-capture or output-compare for each channel

	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 #$00,TIOS   ; all channels are input capture
movb #$FF,TIOS   ; all channels are output capture
movb #$F0,TIOS   ; the first 4 channels are input capture and
                    the last 4 channels are output capture
```

1- Interrupt vectors for Timer Channels:

Priority

$FFEE	Timer channel 0	I bit	TMSK1(C0I)	$EE
$FFEC	Timer channel 1	I bit	TMSK1(C1I)	$EC
$FFEA	Timer channel 2	I bit	TMSK1(C2I)	$EA
$FFE8	Timer channel 3	I bit	TMSK1(C3I)	$E8
$FFE6	Timer channel 4	I bit	TMSK1(C4I)	$E6
$FFE4	Timer channel 5	I bit	TMSK1(C5I)	$E4
$FFE2	Timer channel 6	I bit	TMSK1(C6I)	$E2
$FFE0	Timer channel 7	I bit	TMSK1(C7I)	$E0

Example:

Set the interrupt vector of the timer channel 7

```
org $FFE0
dc.w Timer_Ch7   ;load Channel 7 ISR vector
```

2- Local interrupt enable bits

- The enabling of the interrupt is controlled by the Timer Interrupt Enable Register (TIE)
- Channels can generate interrupts if it is enabled or its bit in TIE is 1

	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)

```
movb #$1,TIE ; enable interrupt of channel 1
movb #$FF,TIE ; enable the interrupts of all channels
```

3- Interrupt Flags

- Whenever an interrupt happens, the associated timer interrupt flag in Timer Interrupt Flag 1 (TFLG1) register will be set to 1.

	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)

Flag CxF is **cleared** by writing a **"1"** to bit x of this register

Example: movb #$01,TFLG1 will clear the C0F flag.

However, in case of "fast flag clear" mode (see slide 5-6), just reading (input capture) or writing (output compare) the channel register will automatically clear these flags

The actions that can be activated on an output compare

- The actions that can be activated on an output compare pin include
  1. pull up to high
  2. pull down to low
  3. 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 8.18 Timer control register 1 and 2 (TCTL1 & TCTL2)

Input capture respond to rising or falling edge?

- The signal edge to be captured is selected by Timer Control Register 3 and 4 (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 System Control Register 2 (TSCR2)

	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

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.

- TCRE (bit 3) = 0, counter free run. It counts 0000 to FFFF and then 0000 to FFFF and repeats.

- TCRE (bit 3) = 1, the timer can be reset to 0000 when TCNT equals TC7.

Summary

1- Programming the timer interrupt

```
org $FFDE
 dc.w timer_isr ; set up TCNT overflow interrupt vector
```

```
movb   #$80,TSCR1     ;enable timer counter
movb   #$86,TSCR2     ;enable TCNT overflow interrupt, set prescaler to 64
movb   #%10000000,TFLG2    ;clear Timer interrupt flag
```

```
timer_isr: movb #%10000000,TFLG2  ; clear TOF flag

          ; code is here
          rti
```

2- Programming the timer without using interrupt

```
movb   #$80,TSCR1     ;enable timer counter
movb   #$06,TSCR2  ;disable TCNT overflow interrupt, set prescaler to 64
```

3- Programming input capture interrupt

```
org $FFEE
 dc.w PT0_ISR   ;load Channel 0 ISR vector
```

```
bclr TIOS,#$01  ;bit 0 is input-capture (not Output Compare )
movb #$01,TCTL4     ;capture the rising edge of PT0 signal
bset TIE,#$1        ;enable interrupt of channel 0
movb #$01,TFLG1     ; clear the C0F flag
```

```
PT0_ISR:
 movb #$01,TFLG1     ; clear the C0F flag.
; code is here
 rti
```

4- Programming input capture with polling C0F bit (no interrupt)

```
bclr TIOS,#$01      ; bit 0 is input-capture (not Output Compare )
movb   #$01,TCTL4   ; capture the rising edge of PT0 signal
```

Outline

5.1 Timer

5.2 Input Capture and Output Compare Functions

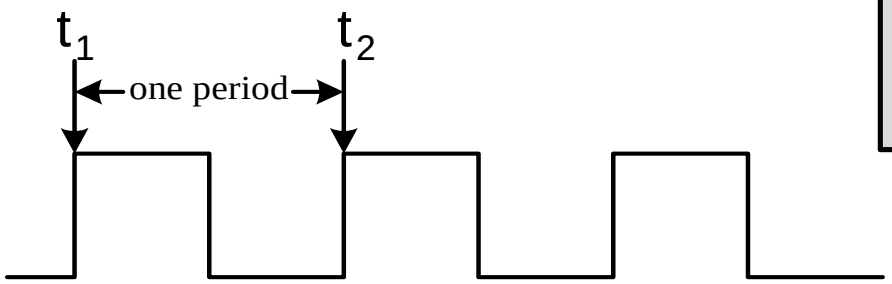
5.3 Applications on Input Capture Function

5.4 Applications on Output Compare Function

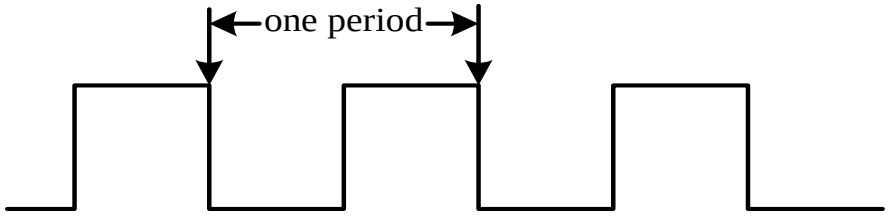
Period measurement

- Need to capture the main timer values (t_1 and t_2) corresponding to two consecutive rising or falling edges

This program can be used to measure the time between two events



(a) Capture two rising edges



(b) Capture two falling edges

Figure 8.9 Period measurement by capturing two consecutive edges

Example: t_1 = 6000 and t_2 = 9000, then the timer counted 3000 counts (or clocks) between two consecutive rising edges. The period = 3000 x 1/F, where 1/F is the duration of one clock.

If each rising edge is an event, the period is the time between two consecutive events

Use the IC0 to measure the period of an unknown signal. The period is known to be shorter than 128 ms. Assume that the E clock frequency is 24 MHz. Use the number of clock cycles as the unit of the period.

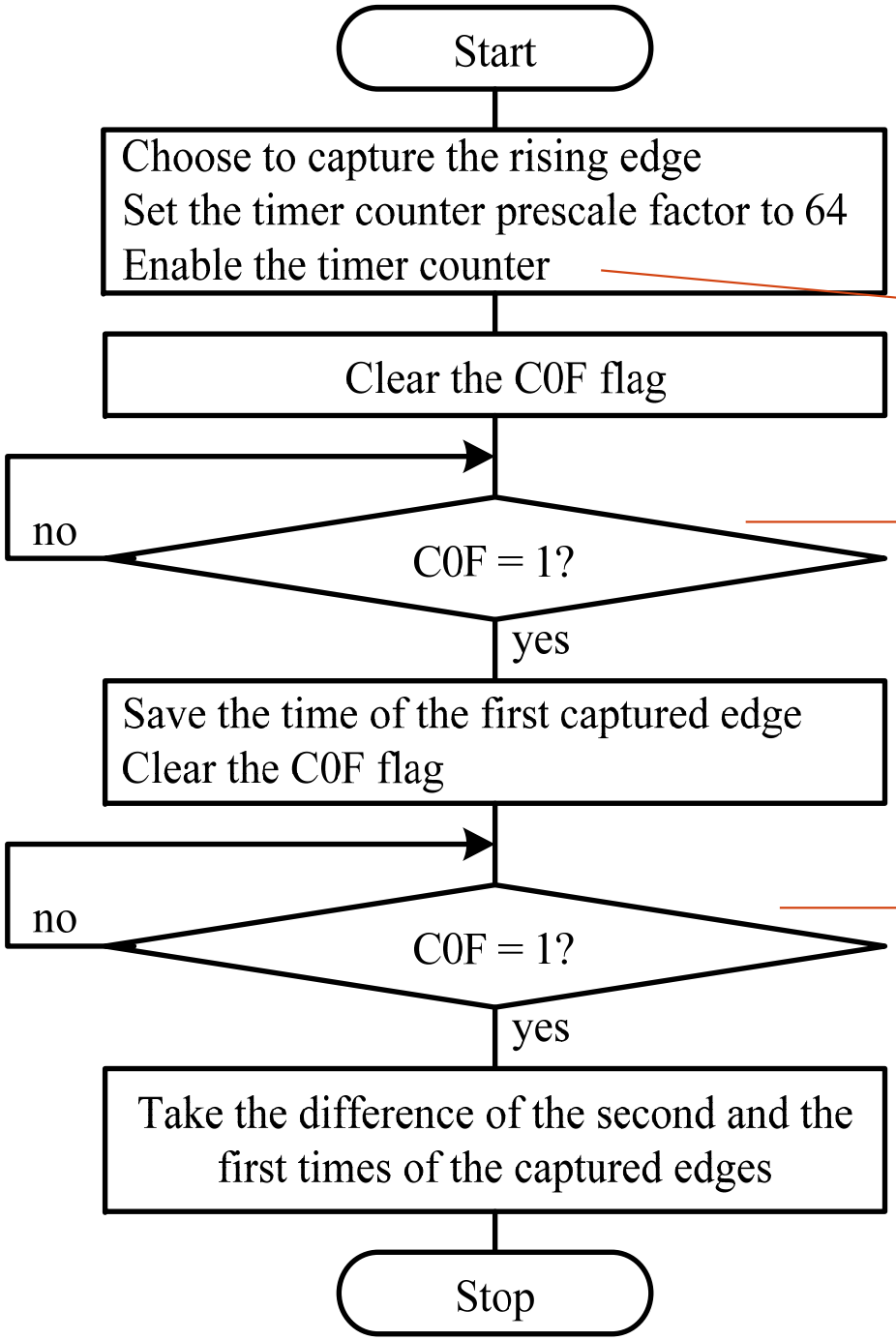
To measure a period that is equal to 128 ms, we have two options:

One: (**will be used in next example**)

- If prescale = 1, the longest period of the signal that can be measured is $2^{16} \div 24$ MHz = 2.73 ms. This means the timer may overflow several times until it captures the falling edge.
- Keep track of the number of times the timer overflows.
- The number of overflows should be taken into account when calculating the period.

Two: (**will be used in this example**)

- Set the prescale factor to 64. The timer overflow time = 174.72 ms > the signal period (128ms) → the two edges can be captured before the timer overflows
- No need to keep track of the number of times the timer overflows.



Interrupt is not used but polling C0F bit is used.

Timer does not count if it is not enabled.

C0F = 1, when the first rising edge comes

C0F = 1, when the second rising edge comes

Figure 8.16 Logic flow of period measurement program

```
          ; Assembly Program for Period Measurement
 ABSENTRY Entry
 INCLUDE 'mc9s12dp256.inc'

 org   $1000
edge1  ds.b 2          ; memory to hold the first edge
period ds.b 2          ; memory to store the period

org    $1500
Entry:  movb #$90,TSCR1  ;enable timer counter and enable fast timer flags clear
        movb #$06,TSCR2  ;disable TCNT overflow interrupt, set prescaler to 64

        bclr TIOS,#$01  ; bit 0 is input-capture (not Output Compare )
        movb #$01,TCTL4; capture the rising edge of PT0 signal
        movb #$01,TFLG1; clear the C0F flag

Here:   brclr TFLG1,#$01,Here   ; wait for the arrival of the first rising edge
        ldd  TC0 ;save the first edge, C0F is cleared automatically
        std  edge1

Here1:  brclr TFLG1,#$01,Here1  ; wait for the arrival of the second edge

        ldd  TC0 ; d = the second edge
        subd edge1      ; compute the period
        std  period
```

movb #$90,TSCR1 ;enable timer counter and enable fast timer flags clear

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

TFFCA -- timer fast flag clear all bit

0 = allows timer flag clearing to function normally
1 = For TFLG1, 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.

bclr TIOS,#$01 ; bit 0 is input-capture (not Output Compare)

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

movb #$06,TSCR2 ; disable TCNT overflow interrupt, set prescaler to 64

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

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

movb #$01,TCTL4 ; capture the rising edge of PT0 signal

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

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

```
movb   #$01,TFLG1      ; clear the C0F flag
```

	0	0	0	0	0	0	0	1
	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)

Repeat pervious program but by using interrupts

```
      ; Assembly Program for Period Measurement using interrupts
 ABSENTRY Entry
 INCLUDE 'mc9s12dp256.inc'

 org   $1000
edge1  ds.b 2          ; memory to hold the first edge
edge2  ds.b 2          ; memory to hold the first edge
Interruptsno ds.b 1    ; memory to number of interrupts
period ds.b 2          ; memory to store the period
 org $FFEE
 dc.w PT0_ISR   ;load Channel 0 ISR vector

org    $1500
Entry: movb #$90,TSCR1  ;enable timer counter and enable fast timer flags clear
       movb #$06,TSCR2  ; disable TCNT overflow interrupt, set prescaler to 64

       bclr TIOS,#$01  ; bit 0 is input-capture (not Output Compare )
       movb #$01,TCTL4; capture the rising edge of PT0 signal
       movb #$01,TFLG1; clear the C0F flag
       bset TIE,#$1 ; enable interrupt of channel 0
       clr Interruptsno
```

```
Again: ldaa Interruptsno
       cmpa #2
       bne again  ; loop until two edges come
 ldd edge2
 subd  edge1; compute the period
 std   period
here:  bre here


PT0_ISR: movb #$01,TFLG1    will clear the C0F flag.
       Inc Interruptsno
       ldaa Interruptsno
       cmpa #1
       bne two
       ldd  TC0 ;save the first edge's time at the first interrupt
       std  edge1
       rti

two:   ldd TC0 ;save the second edge's time at the second interrupt
       std edge2
       bclr TIE,#$1  ;disable interrupt of channel 0
        rti
```

Pulse width measurement: Write a program to measure the pulse width of a signal connected to the PT0 pin. E-clock = 24 MHz

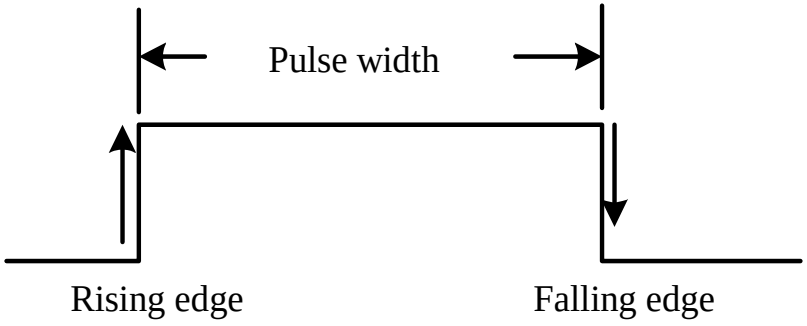


Figure 8.10 Pulse-width measurement using input capture

- Need to capture the rising and falling edges

STEPS:

1. Set the prescale to 32.
2. The pulse width may be longer than 2^{16} clock cycles, we need to keep track of the number of times the timer overflows.
3. Variables: *overflow* = 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

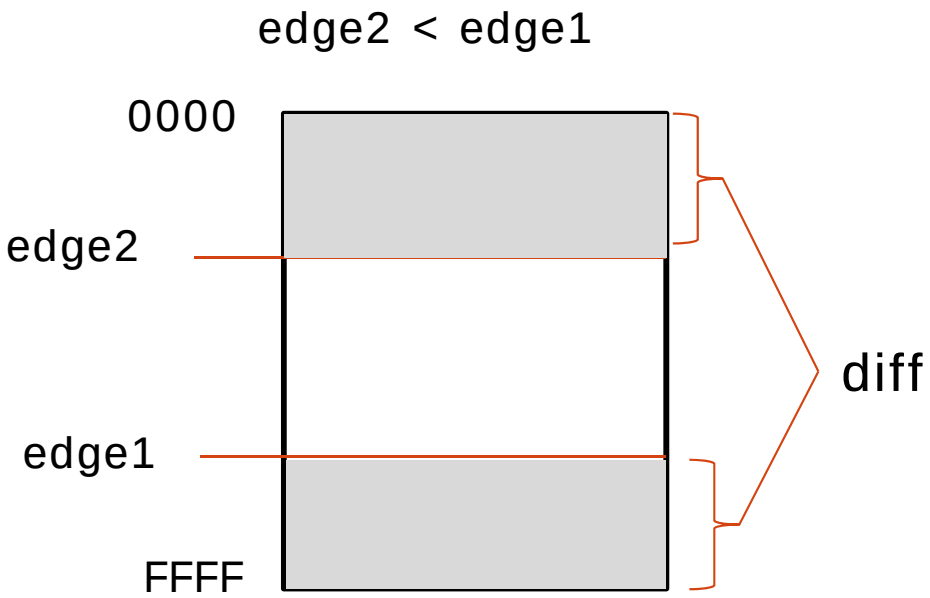
4. The pulse width can be calculated by the following equations:

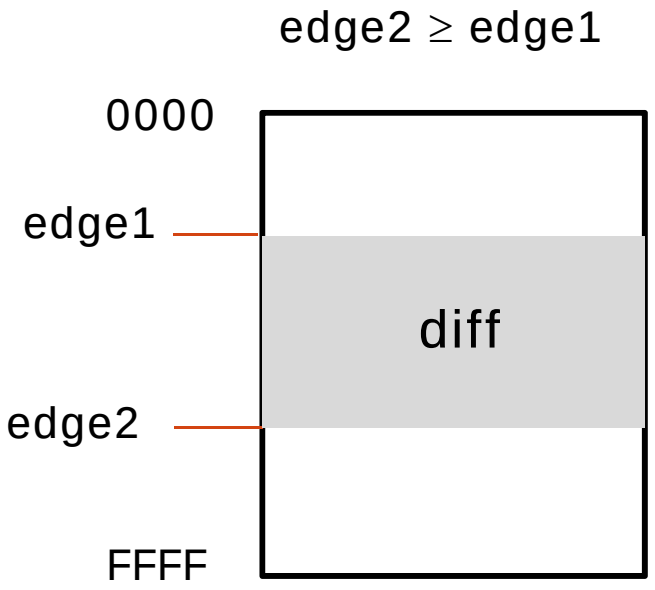
Case 1: edge2 ≥ edge1

pulse width = overflow × 2^{16} + diff (= edge2 - edge1)

Case 2: edge2 < edge 1

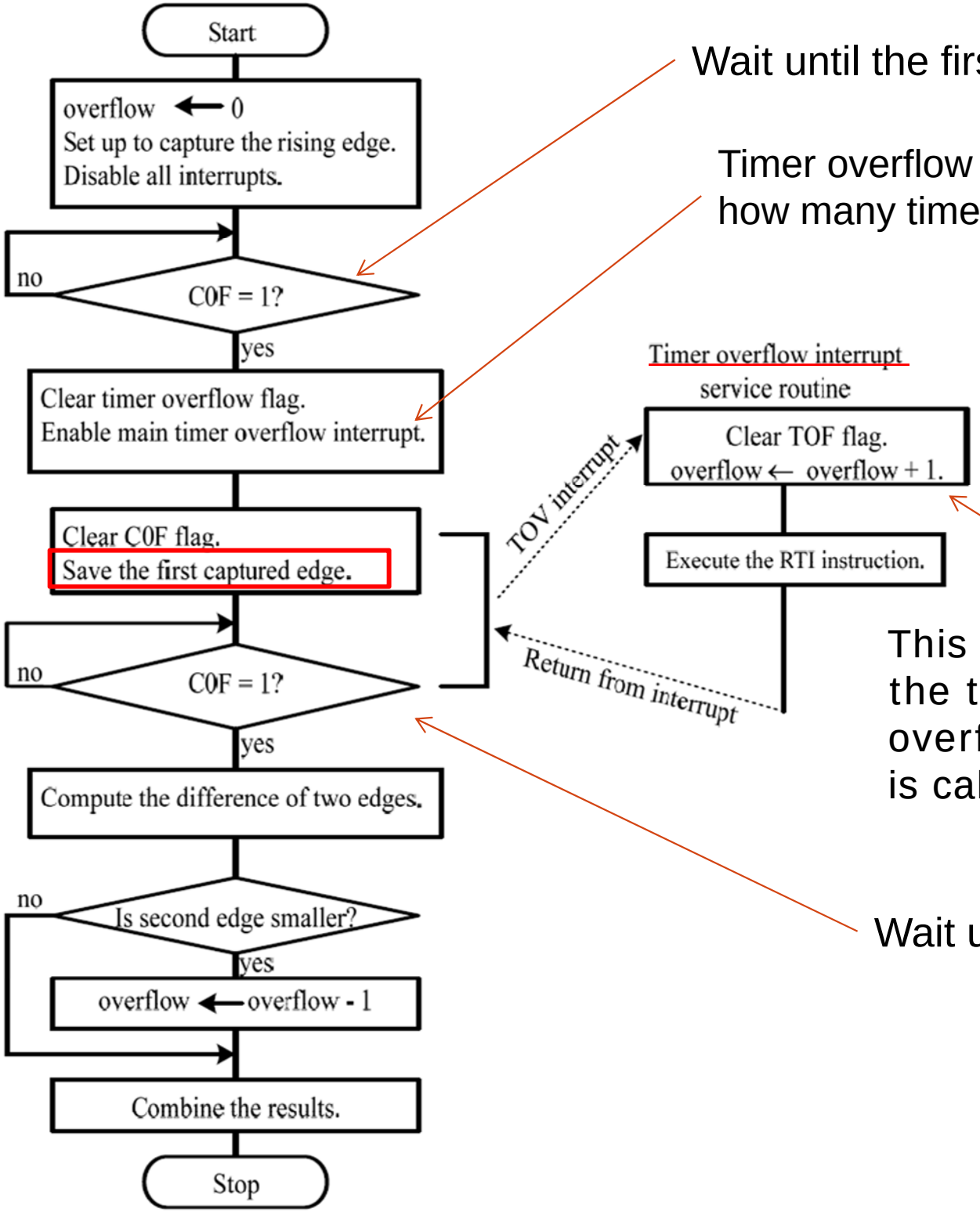
pulse width = (overflow – 1) × 2^{16} + diff





From edge1 to edge 2, the number of counts = diff and number of overflows is one, then add 2^{16} for every overflow after the first one.

From edge1 to edge 2, the number of counts = diff, then add 2^{16} for every overflow



Wait until the first edge comes

Timer overflow interrupt is enabled to count how many times the timer overflows

This routine is called each time the timer overflows. Increment overflow each time the routine is called.

Wait until the second edge comes

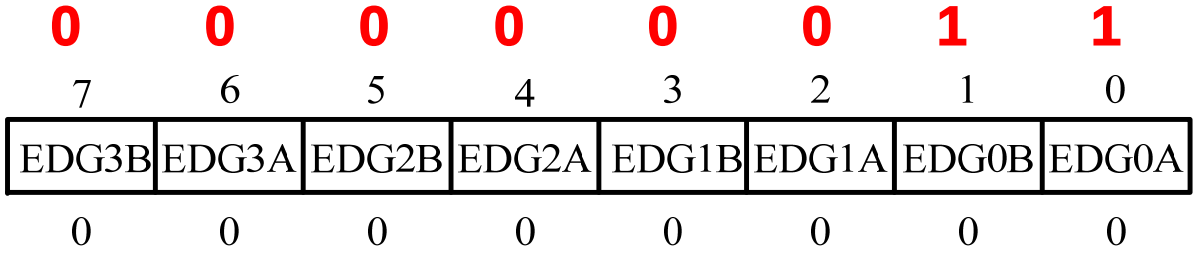
```
; Assembly Program to measure pulse width
ABSENTRY Entry
 INCLUDE 'mc9s12dp256.inc'

 org $1000
edge1 ds.b 2
overflow ds.b 2  ; counter to the number of overflows.
diff ds.b 2
PW dc.b     0,0,0,0
 org $$FFDE
 dc.w #timer_isr ; set up TCNT overflow interrupt vector

org    $1500
Entry:
       lds  #$1500    ; set up stack pointer
       movw #0,overflow
       movb #$90,TSCR1 ; enable TCNT and fast timer flag clear
       movb #$05,TSCR2      ; disable TCNT interrupt, set prescaler to 32
       bclr TIOS,#$01  ; select IC0
       movb #$01,TCTL4 ; capture rising edge
       movb #$01,TFLG1 ; clear C0F flag
Here:  brclr TFLG1,#$01,here ; wait for the first rising edge
       movw TC0,edge1 ; save the first edge & clear the C0F
```

```
flag: movb #%10000000,TFLG2      ; clear Timer interrupt flag
      bset TSCR2,#$80      ; enable TCNT overflow interrupt
 cli
 movb #$02,TCTL4 ; capture the falling edge on PT0 pin
here1: brclr  TFLG1,C0F,here1 ;Wait for the arrival of the falling

edge: ldd TC0          ; d = edge2
      subd    edge1  ;d = diff = edge2-edge1
      std     diff  ; std does not change the carry
      bcc     next        ; is the second edge smaller?
      ldy overflow       ; second edge is smaller, so decrement
      dey                ; overflow count by 1
      sty overflow          ;       "

 ldd #$FFFF ; d = 2^16
 Emul  ; D * Y = overflow (or overflow-1)* 2^16 → Y:D
 Addd diff
 Std PW ; store the first two bytes of the result
 xgdy ; exchange D and Y
```

```
 Adcb PW+2 ; compute and store the third byte of the result
 stab PW+2  ; ''
 Adca PW+3  ; compute and store the fourth byte of the result
 staa PW+3   ; ''

next:   bra next


timer_isr:   movb #%10000000,TFLG2       ; clear TOF flag
             ldx overflow
             inx
             stx overflow
             rti
```

Interrupt can be used to capture the falling and rising edges.

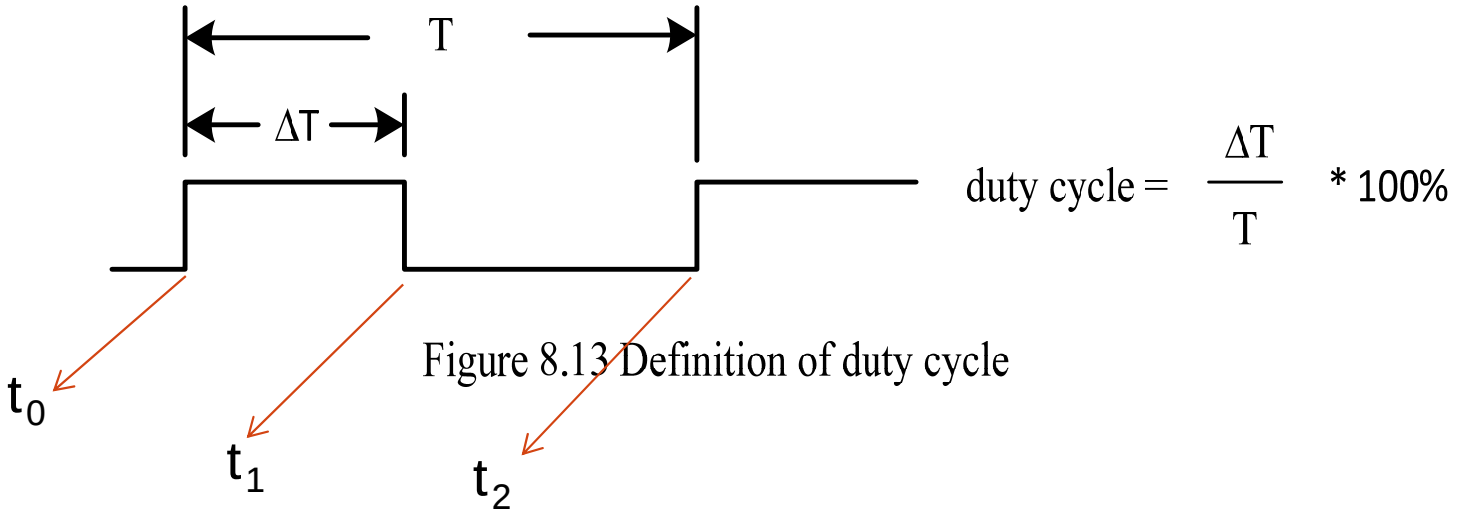
```
movb #$03,TCTL4       ; capture the rising and falling edges of channel 0
```



Other applications for input capture function

(1) Duty Cycle Measurement

- Duty cycle is the percent of time that the signal is high within a period in a periodic digital signal
- Capture the rising and falling edges.
- Record t_0, t_1 and t_2.
- Use them to calculate ΔT and T and the duty cycle.



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

Figure 8.13 Definition of duty cycle

(2) Phase Difference Measurement

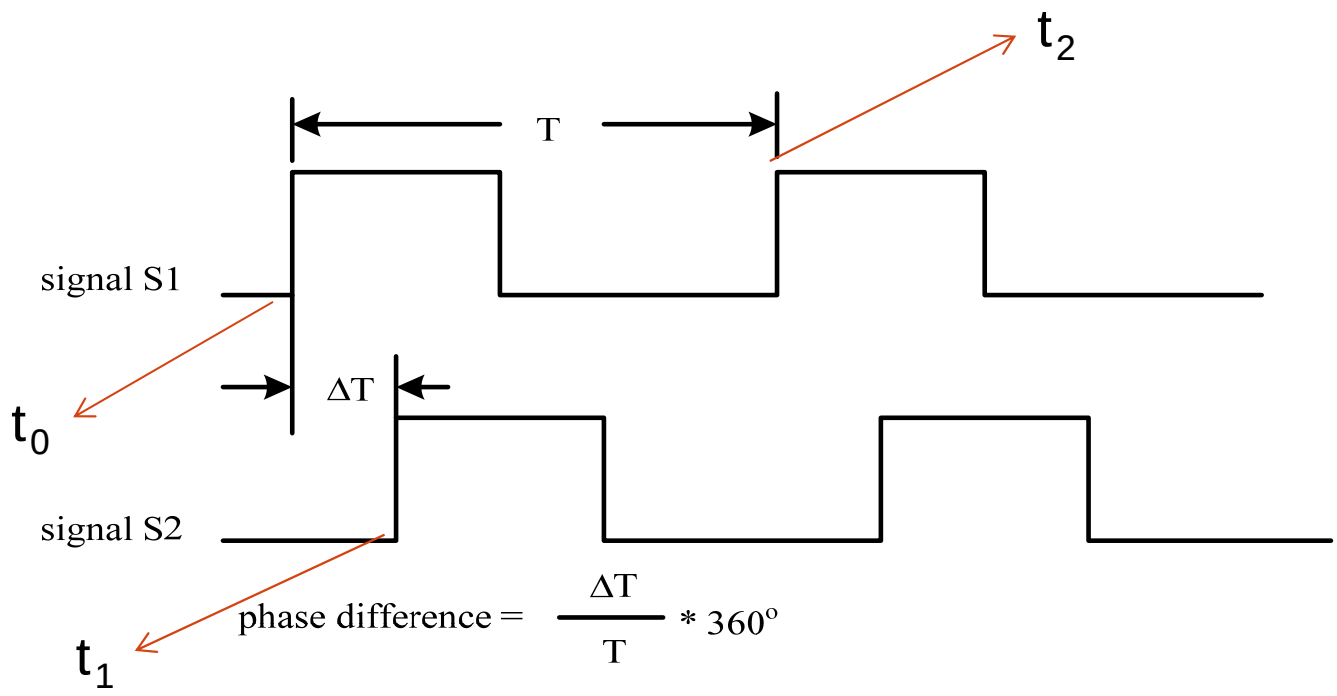


Figure 8.14 Phase difference definition for two signals

- **Phase difference** is the difference of arrival times (in percentage of a period) of two signals that have the same frequency but do not coincide in their rising and falling edges.

- Two channels are used to capture the rising time of each signal (t_0 and t_1)

- Record $\Delta T = t_1 - t_0$ and $T = t_2 - t_0$

(3) Interrupt generation

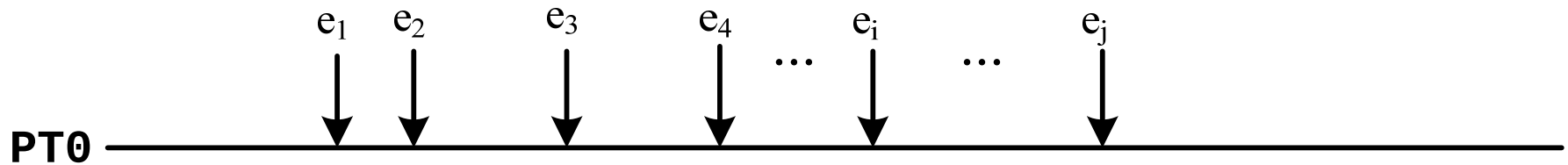
- An interrupt can be generated when an event happens (falling/rising edge on a pin).
- Similar to IRQ, we can use input capture to generate an interrupt when an event happens instead of using it as an event time recorder.
- For example, input capture pin can be connected to a switch that makes falling (or rising) edge when pressed.
- Write in the input capture routine what you wanna do when the switch is pressed, for example, turn on/off a functionality.
- Input capture pin can be connected to a circuit or a sensor that needs to interrupt the microcontroller to ask it to do something.
- Input capture pin can be connected to events' signals to count them. Simply, an event generates an interrupt when it comes. The interrupt routine counts how many times it is called.

A switch is connected to pin 0 of port T to make rising edge when pressed. Write a program to increment the binary number displayed on the LEDs (on port B) each time the switch is pressed.

```
      ABSENTRY Entry
 INCLUDE 'mc9s12dp256.inc'

 org   $1000
count  ds.b 1          ; the number to be displayed on LEDs.

 org $FFEE
 dc.w PT0_ISR   ;load Channel 0 ISR vector

org    $1500
Entry: movb #$FF,DDRB ; configure port B for output
       bset DDRJ,$02 ;configure PJ1 pin for output
       bclr PTJ,$02 ;enable LEDs to light
       movb #$FF,DDRP ; disable 7 segments that are connecetd
       movb #$0F,PTP ; ‘’

       movb #$90,TSCR1 ;enable timer counter and enable fast timer flags clear
       bclr TIOS,#$01 ; bit 0 is input-capture (not Output Compare )
```

```
 movb  #$01,TCTL4    ; capture the rising edge of PT0 signal
 movb  #$01,TFLG1    ; clear the C0F flag
 movb #$1,TIE ; enable interrupt of channel 1
 clr count
 cli
here:  bra here   ; wait interrupts

PT0_ISR:
 movb #$01,TFLG1      will clear the C0F flag.
 inc count
 movb count,PORTB
 rti
```

Event counting: If a sequence of events (instead of a switch) is connected to PT0, count = the number of events, e.g, number of customers who entered a store.



(4) Number of events during a time period

- From previous example, input capture pin can count the number of events generated on input capture pin.
- Input capture function can be used to count the number of events during a period by enabling the interrupt at the beginning and disabling it at the end of the period.
- The program will be similar to the previous example but we add this functionality.

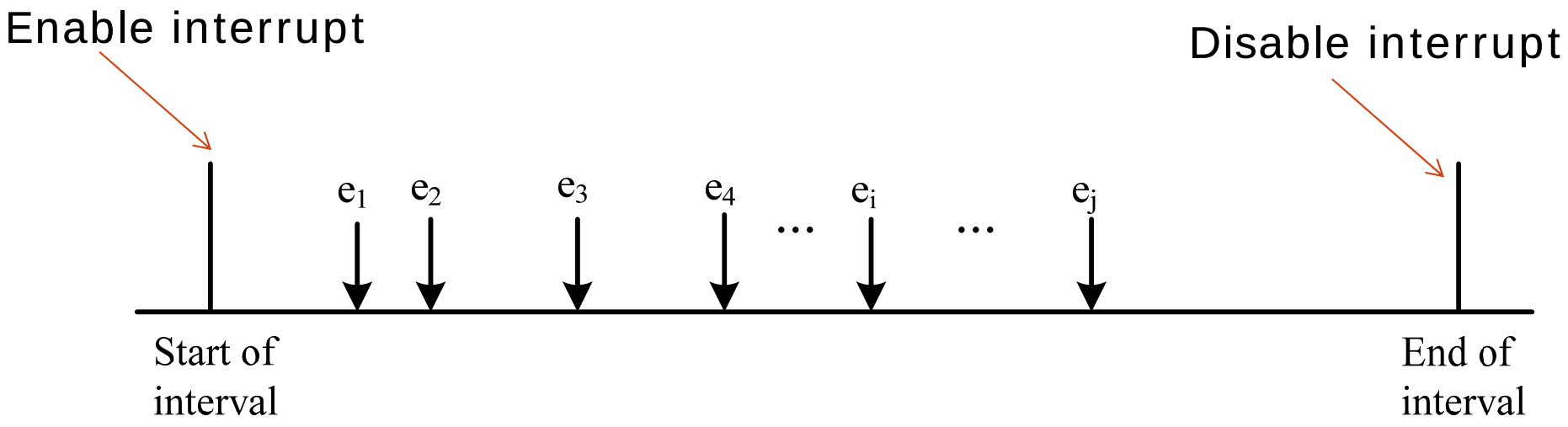


Figure 8.11 Using an input-capture function for event counting

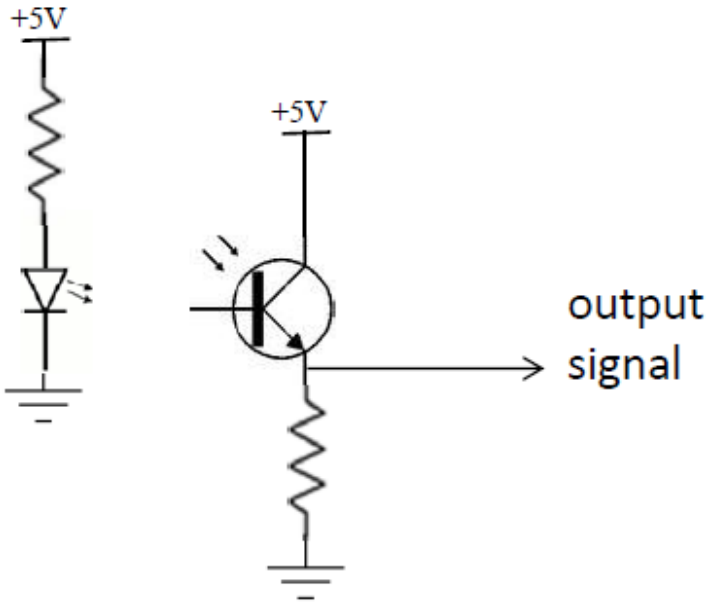
Write a program to count the number of events on pin 0 of port T in a certain period of time.

```
      ABSENTRY Entry
 INCLUDE 'mc9s12dp256.inc'

 org   $1000
count  ds.b 1    ; counter to the number of events.
Ovcnt ds.b  1    ; the number of timer overflows
org $FFEE
 dc.w PT0_ISR   ;load Channel 0 ISR vector
org $FFDE
 dc.w timer_isr ; set up TCNT overflow interrupt vector


org    $1500
Entry: movb #$80,TSCR1  ;enable timer counter
       bclr TIOS,#$01  ; bit 0 is input-capture (not Output Compare )
       movb #$86,TSCR2  ; enable TCNT overflow interrupt, set prescaler to 64
       movb #%10000000,TFLG2      ; clear Timer interrupt flag

       movb #$01,TCTL4; capture the rising edge of PT0 signal
       movb #$01,TFLG1; clear the C0F flag
       bset TIE,#$1 ; enable interrupt of channel 0
```

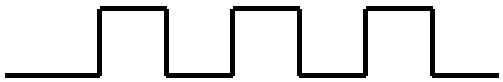
```
 clr Ovcnt
 clr count
 cli
here:  bra here   ; wait interrupts


PT0_ISR:
 movb #$01,TFLG1     will clear the C0F flag.
 inc count
 rti

timer_isr:   movb #%10000000,TFLG2       ; clear TOF flag
             inc Ovcnt
             ldaa Ovcnt
             cmpa #200    ; we want to count the event
              bne done    ;during the time period of 200 overflows
   ;; stop counting the event by disabling channel 0 interrupts
          bclr TIE,#$1 ; disable interrupt of channel 0
          done: rti
```

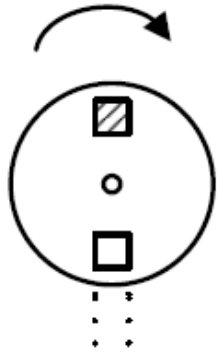
Modify this program to add this functionality. A switch is connected to IC1 (pin 1 on port T). Start counting when the switch is pressed.

An application for previous program: Motor speed



One rotation = 2 pulses





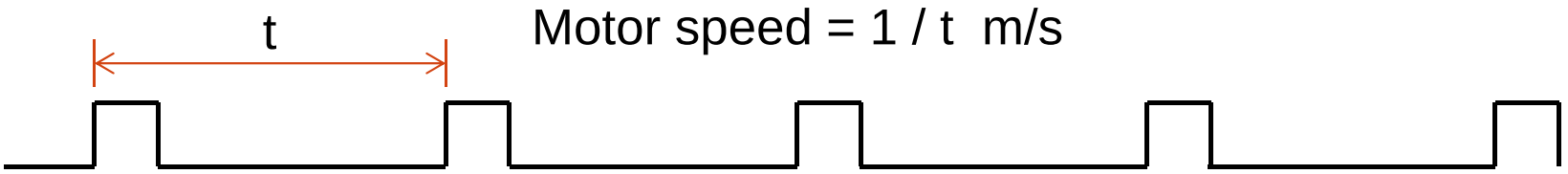
Optical encoder

integrated package

- An optical encoder sensor uses an LED and a phototransistor
- A disc having two holes is attached to the motor shaft.
- The disc rotates between the LED and the phototransistor.
- Two pulses will be generated when the disc makes a complete rotation.
- When the hole is between the LED and the phototransistor, the phototransistor conducts and the output is pulled high.
- Use previous program to count the number of pulses in a second. The half of this count gives the motor speed in rotations per second.

Control the speed of a conveyor

- To measure the speed of a conveyor, an optical encoder is used.
- Metal pieces are attached to the belt to make a pulse when each piece moves through the encoder.
- The distance between each two metal pieces is 1 meter. The time between each two pulses is the time for the conveyer to move 1 meter.



- The period measurement program in slides 5-27 and 5-28 can be used to measure t.
- Modify the program to periodically measure the time between each two pulses and to add a turn/off switch.
- If the measured time is less than a threshold (T1), increment the number on port A to increase the motor speed. If the time is more than a threshold (T2) decrement the number on port A to decrease motor speed.

Outline

5.1 Timer

5.2 Input Capture and Output Compare Functions

5.3 Applications on Input Capture Function

5.4 Applications on Output Compare Function

Programming output compare interrupt

```
org $FFEE
 dc.w PT0_ISR   ;load Channel 0 ISR vector
```

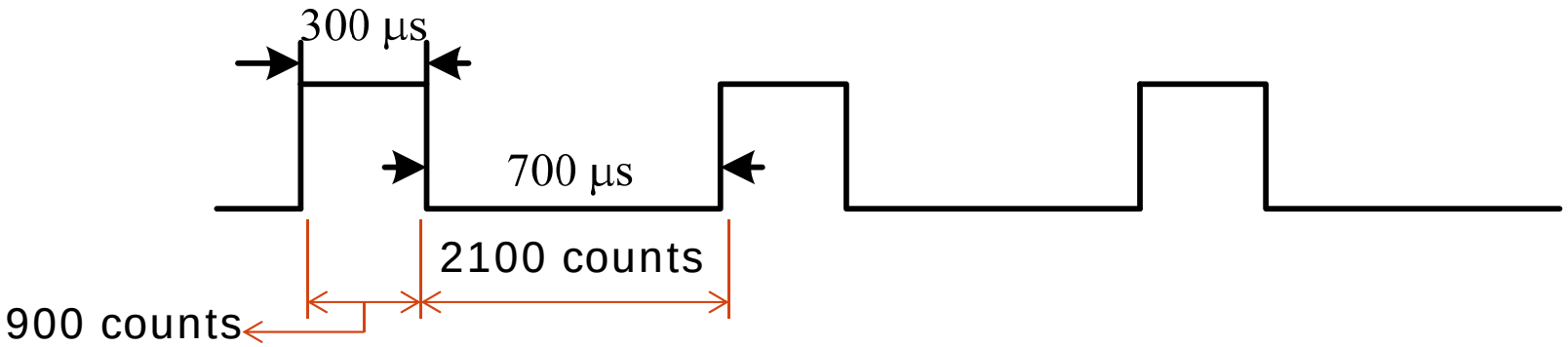
```
bset TIOS,#$01  ;bit 0 is Output Compare (not input-capture)
bset TIE,#$1        ;enable interrupt of channel 0
movb #$03,TCTL2 ;action to be taken 1 = toggle, 2 = clear, 3 = set
movb #$01,TFLG1     ; clear the C0F flag
```

```
PT0_ISR:
 movb #$01,TFLG1     ; clear the C0F flag.
; code is here
 rti
```

Programming output compare with polling C0F bit (no interrupt)

```
bset TIOS,#$01  ;bit 0 is Output Compare (not input-capture)
movb #$03,TCTL2 ;action to be taken 1 = toggle, 2 = clear, 3 = set
```

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.



- If the prescale is set to 8, the timer clock is 24/8 = 3 MHz. The period of the clock signal to the timer will be 1/3 µs.
- The numbers of clock cycles that the signal is high = 300 µs/(1/3) = 900
- The numbers of clock cycles that the signal is low = 700/(1/3) = 2100
- We need to use two values for TC0:-

  1- Count for TC0 = TCNT + 2100, pull OC0 high after counting

  2- Count for TC0 = TCNT + 900, pull OC0 low after counting

  3- Go to step 1

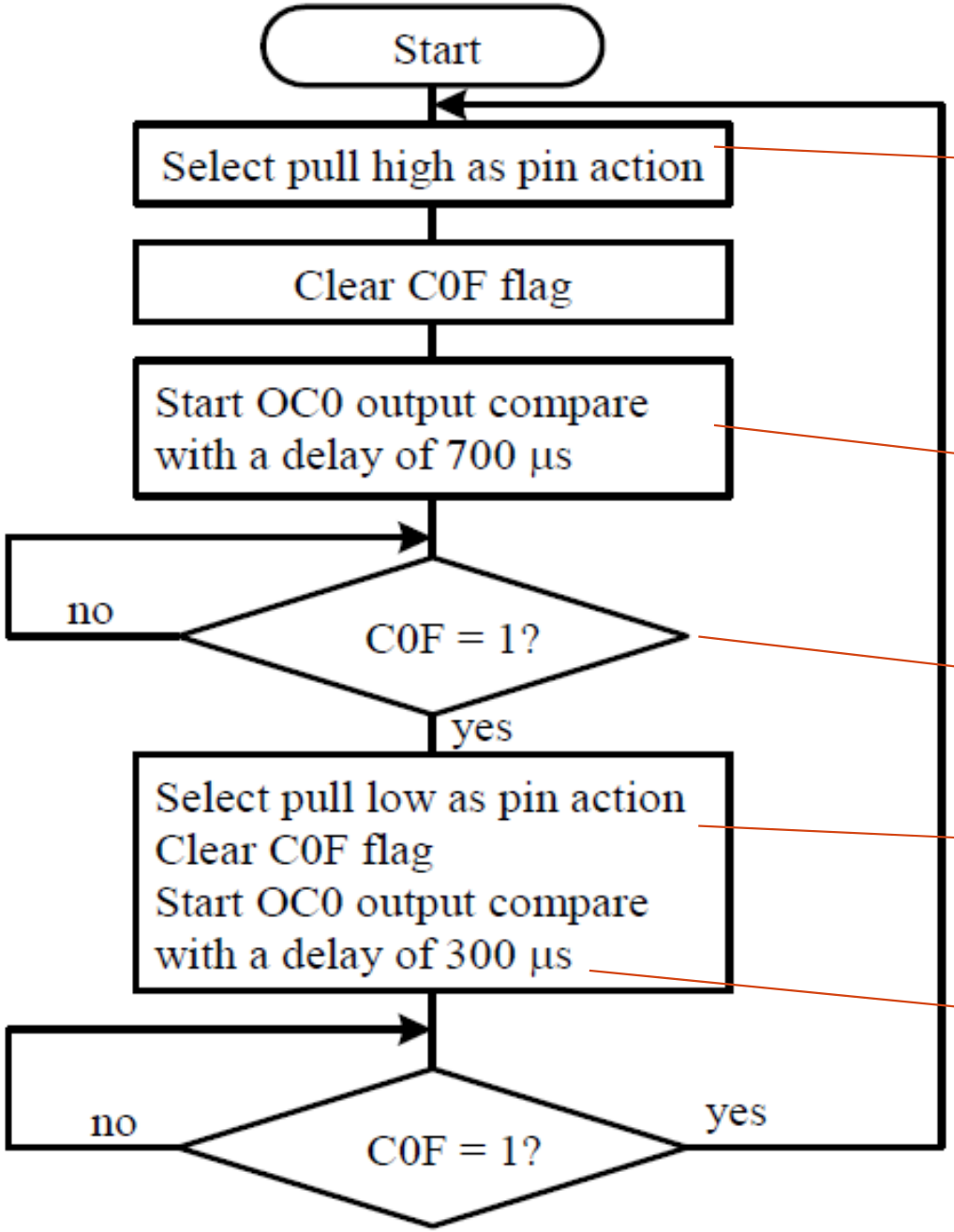


Figure 8.20 The program logic flow for digital waveform generation

```
hi_time equ  900
lo_time equ  2100

org   $1500
 movb #$90,TSCR1     ;enable TCNT with fast timer flag clear
 movb #$03,TSCR2     ;disable TCNT interrupt, set prescaler to 8

 bset TIOS,#$01      ;enable OC0
 movb #$03,TCTL2     ;select pull high as pin action

 ldd TCNT           ;start an OC0 operation with 900 us as delay
repeat: addd #lo_time    ; "
 std TC0            ;TC0 = TCNT + 2100

low: brclr TFLG1,C0F,low  ;wait until OC0 pin go high after counting 2100

 movb #$02,TCTL2       ;select pull low as pin action
 ldd TC0               ;start an OC operation with 300 us as delay
 addd #hi_time         ; "
 std TC0               ; TC0 = TCNT + 900

 high: brclr TFLG1,C0F,high ; wait until OC0 pin go low after counting 900

 movb #$03,TCTL2 ;select pull high as pin action
 ldd TC0

 bra repeat
```

We can use interrupt-driven method to generate the waveform so that the CPU can still perform other operations.

1- OC pin is toggled from 0 to 1
2- Generate interrupt. In interrupt routine: OC value = TCNT +900

1- OC pin is toggled from 1 to 0
2- Generate interrupt: In interrupt routine: OC value = TCNT + 2100

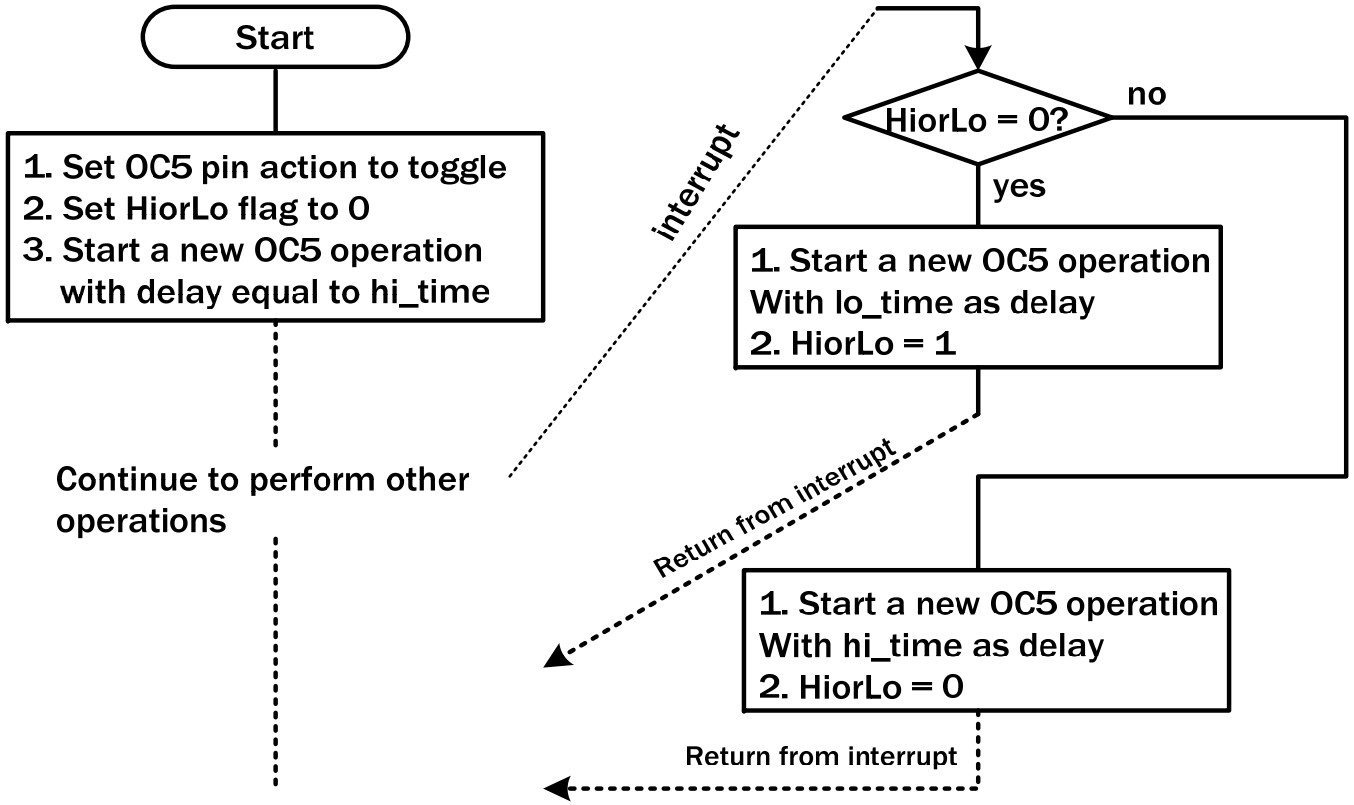


Figure 8.20 The program logic flow for digital waveform generation

```
hi_time equ  900 ; delay count for high interval of the waveform
lo_time equ  2100 ; delay count for low interval of the waveform

      org    $1000
HiorLo ds.b  1   ; flag to select hi_time (1) or lo_time (0)

 org $FFE4
 dc.w PT5_ISR   ;load Channel 0 ISR vector


      org    $1500
 ; configure timer
 movb  #$90,TSCR1    ; enable TCNT and fast timer flag clear
 movb  #$03,TSCR2    ; set TCNT clock prescaler to 8

 ; configure OC5
 bset  TIOS,#$%00100000    ; enable OC5 interrupts
 movb  #$04,TCTL1    ; change pin action to toggle
 bset TIE,#$%00100000    ;enable interrupt of channel 5

 clr HiorLo ; lo_time will be the delay count next time
 ldd   TCNT   ; start another OC5 operation with
 addd  #hi_time      ; delay count set to hi_time
 std   TC5    ;      "

here: bra    here   ; wait interrupts
```

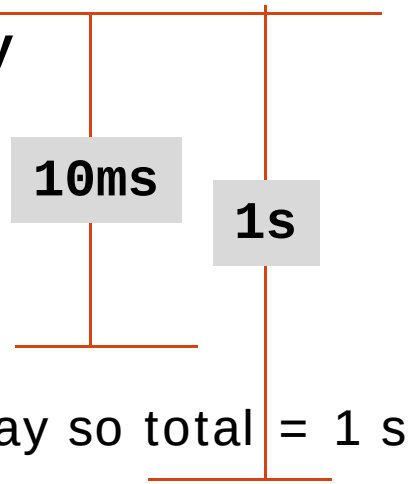
```
PT5_ISR:
      tst HIorLO    ; which delay count should be added?
      beq   addLow  ; if 0 then select lo_time

      ldd   TCNT    ; select hi_time as the delay count for
      addd  #hi_time; the new OC5 operation
      std   TC5     ;       "
      clr   HiorLo  ; toggle HiorLo flag
      rti

addLow: ldd TCNT    ; select lo_time as the delay count for
      addd  #lo_time; the new OC5 operation
      std   TC5     ;       "
      movb  #1,HiorLo       ; toggle HiorLo flag
      rti
```

Estimate frequency: Use an input-capture and an output-compare functions to measure the frequency of the signal connected to PT0 pin.

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

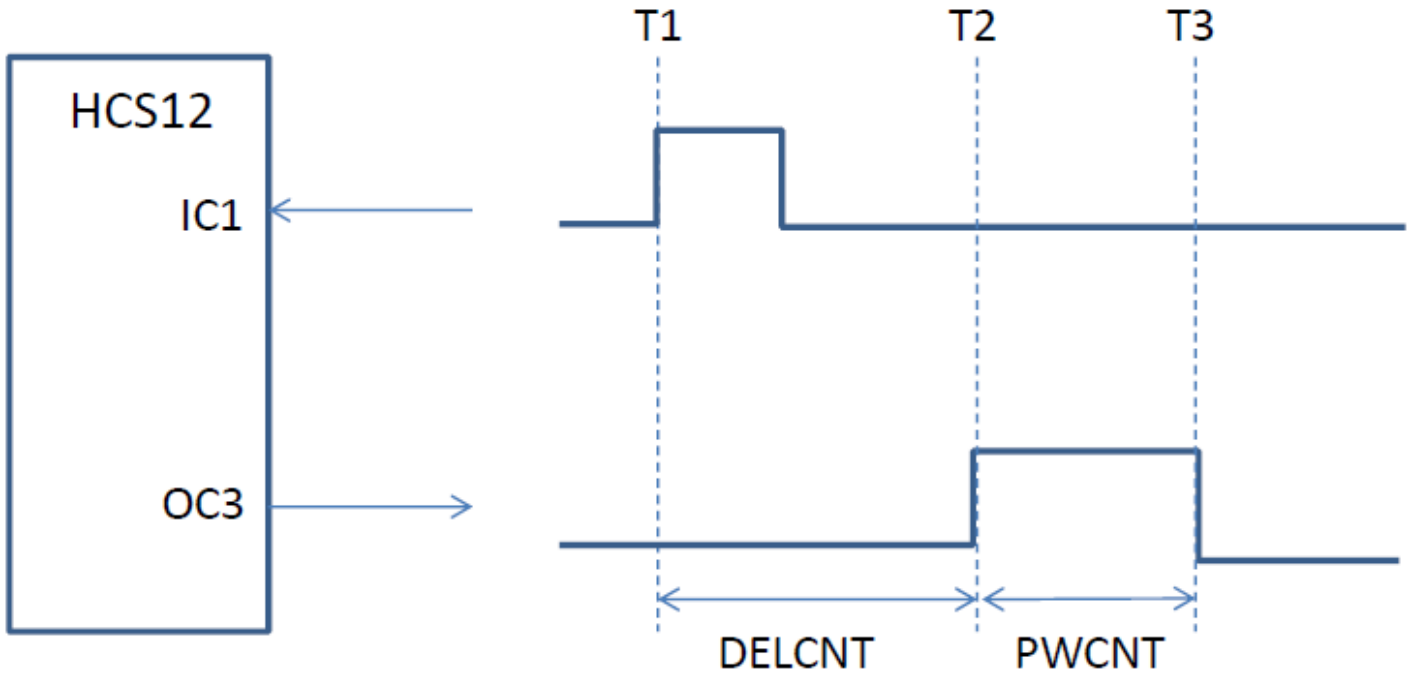
```
org $1000
frequency ds.b 2
org $FFEE     ; set up interrupt vector number for TC0
 dc.w TC0_isr   ;load Channel 0 ISR vector

 org $2000
movb #$90,TSCR1 ; enable TCNT and fast timer flags clear
movb #$02,TSCR2 ; set prescale factor to 4

movb #$02,TIOS ; enable OC1 and IC0
movb #$01,TCTL4 ; prepare to capture the rising edges of PT0
movb #C0F,TFLG1 ; clear the C0F flag
bset TIE,#01 ; enable IC0 interrupt

movw #0,frequency ; initialize frequency count to 0
cli ; "
```

```
;----- use OC1 to make a delay for 1 second
 ldy #100
continue:
 ldd TCNT ; start an OC1 operation with 10 ms delay
 addd #60000 ; "
 std TC1 ; "
w_lp: brclr TFLG1,#02,w_lp; wait for 10 ms

 dbne y,continue   ; 100 iteraions – each creates 10 ms delay so total = 1 s
;-----------------------------

here: bra here


TC0_isr:
       ldd TC0 ; clear C0F flag
       ldx frequency ; increment frequency count by 1
       inx ; "
       stx frequency ;
 rti
```



Generating a delayed pulse

- Input capture function can be used in conjunction with an output compare function to generate a delayed pulse
- In some applications, we need to generate an output pulse for a certain time on output compare pin after receiving an input pulse
- We'll set up input capture to look for a rising edge
- When found, the input capture ISR will set up output compare to make the output pin go high at time T2 = T1 + DELCNT
- When the output compare occurs, the output compare ISR will set up the output pin to go low at time T3 = T2 + PWCNT



- A sensor generates a pulse when the containers are in the proper place
- The microcontroller should wait (DELCNT) and then it generates a signal to turn on a valve for a certain time (PWCNT) to fill up containers

Main program

– Initialize IC1 to look for low-to-high edge
– Enable IC1 interrupts
– Disble OC3 interrupts
– Turn on interrupt system (cli)
– Wait forever

IC1_ISR

– Clear interrupt flag
– T2 = T1 + DELCNT
– Store T2 into TC3
– Set OC3 to go high on next match
– Clear OC3 flag
– Enable OC3 interrupt

OC3_ISR

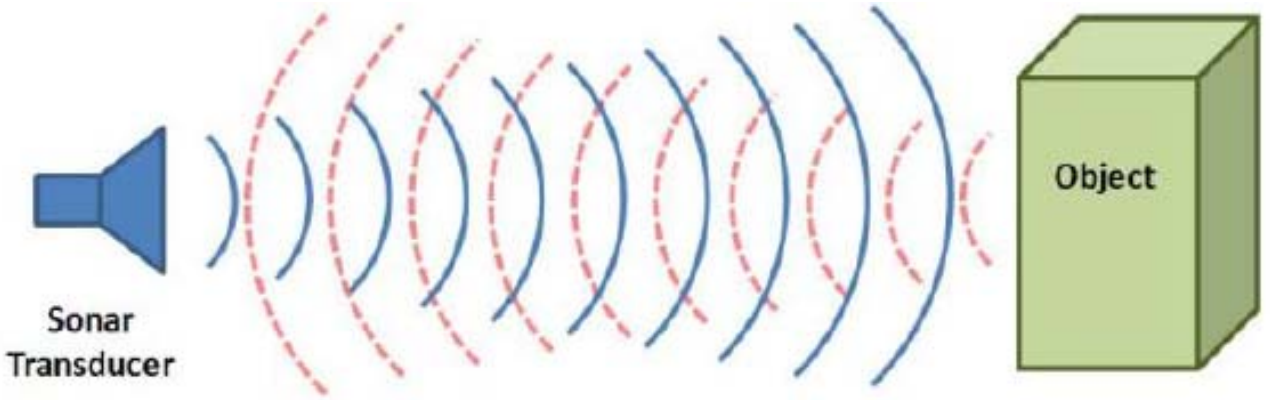
– Disable OC3 interrupts
– T3 = T2 + PWCNT
– Store T3 into TC3
– Set OC3 to go low on next match

Distance measurement

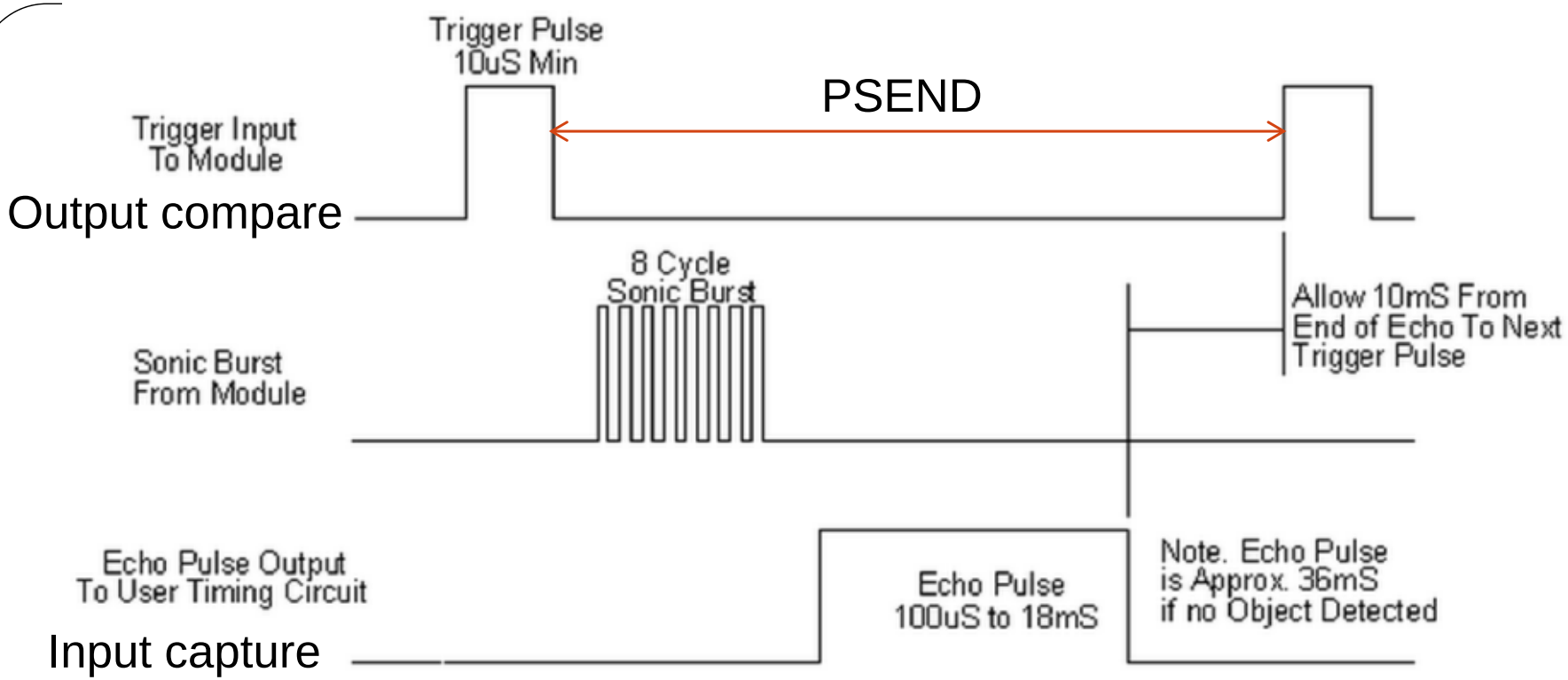
- An ultrasonic sensor emits a high frequency sound pulse, then waits for the reflected pulse
- The distance can be determined by the time of flight to the object (t). The distance can be calculated from the speed of sound = distance/t

-<u>To use the sensor</u>:-

1. Send a pulse to trigger the transmitter
2. The transmitter sends ultrasonic wave and pulls the receiver pin high
3. The receiver pin is low when the retuned signal is received.
4. The time of flight to the object (t) is the time interval the receiver is high



Basic sonar illustration – a transducer generates a sound pulse and then listens for the echo.



- Output compare pin can be used to trigger the sensor by generating a pulse at intervals of PSEND
- Input capture pin is used to get the pulse width of the return pulse (t)

Input capture ISR

Capture rising edge (t1) and falling edge (t2)

Compute t = t2-t1

Write a program to generate a number of pulses using an OC function. The specified high interval duration (12 ms) and low interval duration (8 ms). Use the interrupt-driven approach so that the CPU can perform other operations.

Let:

NN:	number of pulses to be generated
DelayHi:	high interval duration
DelayLo:	low interval duration
HiorLo:	flag to select DelayHi or DelayLo
pcnt:	number of OC0 operation to be performed

Steps:

1. Pull the PT0 pin high quickly using the OC0 operation.
2. Change the OC0 pin action to toggle. Start the next OC0 operation with delay equal to DelayHi.
3. pcnt ← 2 * NN - 1. HiorLo ← 0.
4. Enable OC0 interrupt.
5. The main program continues to perform other operations.

```
DelayHi equ 18000 ;pulse high interval duration time = 18000/1.5= 12 ms
DelayLo equ 12000  ; pulse low interval duration
NN equ 10          ;number of pulses to be created

 org  $1000
pcount ds.b 1  ; number of OC0 operations remaining to be performed
HiorLo ds.b 1 ; flag to choose DelayHi(1) or DelayLo (0)

 org $FFEE
 dc.w oc0ISR   ;load Channel 0 ISR vector

org   $1500
```

Configuration

```
      lds   #$1500
      movb  #$90,TSCR1
      movb  #$04,TSCR2 ; prescale = 16, f =24/16=1.5Mhz
      bset  TIOS,OC0  ; enable OC0
      movb  #01,TFLG1; clear C0F flag
      movb  #$03,TCTL2      ; set OC0 pin action to pull high
```

pull TC0 pin high

```
      ldd   TCNT      ; pull TC0 pin high
      addd  #12       ; quickly
      std   TC0       ;     "
here: brclr TFLG1,C0F,here ; wait until C0F flag is set

      movb  #$01,TCTL2      ; set OC0 pin action to toggle
      ldd   TCNT      ; start next OC0 operation with
      addd  #DelayHi  ; delay set to DelayHi
```

```
 std  TC0  ;       "
 movb 2*NN-1,pcount  ;prepare to perform pcount OC0 operations
      clr  HiorLo  ; next OC0 operation use DelayLo as delay
here1: bra here1

oc0ISR: ldaa HiorLo  ;check the flag to choose delay count
      beq  pulseLo    ;if flag is 0, then go and use DelayLo
      ldd  TCNT       ;start an OC0 operation and use
      addd #DelayHi  ; DelayHi as delay count
      std  TC0     ; "
      movb #0,HiorLo; toggle the flag
      bra  decCnt

pulseLo: ldd TC0   ; start an OC0 operation and use
      addd #DelayLo  ; DelayLo as delay count
      std  TC0     ; "
      movb #1,HiorLo; toggle the flag

decCnt: dec pcount
      bne  quit
      movb #0,TIE  ; disable OC0 interrupt
      bclr TIOS,$01  ; disable OC0

quit: rti
```

Disable interrupts after generated the required number of pulses

A highly accurate digital clock using the timer

– Use prescaler of 16 to divide 24 MHz bus clock down to 1.5 MHz (will cause TCNT register to increment every *0.667* ms)

– Use Timer Channel 7 so that the TCNT register can be *automatically reset* when the Output Compare occurs

– Set TC7 register to 15,000, so that interrupt will occur every 10 ms

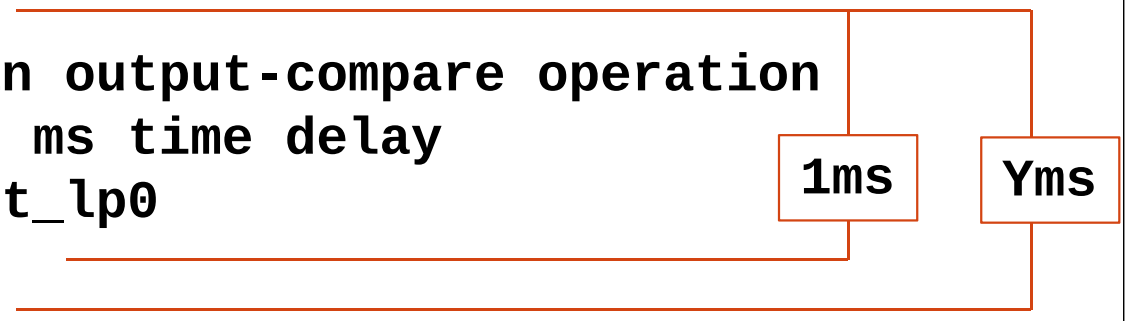
– Update the clock when 100 of these interrupts accumulate (= 1 second)

```
; Initialization code

movb #$80,TSCR1 ; Enable timer
movb #$0C, TSCR2 ; Set prescale factor to 16, enable the timer
                   and counter reset after OC7
movb #$80,TIOS ; Set Ch 7 for Output Compare
movb #$80,TIE ; Enable Ch 7 interrupt
movw #15000,TC7  ; Set up Ch 7 to generate 10 ms interrupt rate
```

Write a subroutine called **`Delayby1ms`** to generate a time delay that is a multiple of 1 ms. Assume that the E clock frequency is 24 MHz. The number of milliseconds is passed in Y.

- Set the prescale to 8
- Perform the number of output-compare operations (given in Y) with each operation creating a 1-ms time delay.
- The number to be added to TCNT is 3000. (3000 × (8/24MHz) = 1 ms)
- The number to be added to TCNT is 375 when prescale = 64

```
Delayby1ms:
          pshd
          movb #$90,TSCR1 ; enable TCNT & fast flags clear:
          movb #$03,TSCR2   ; configure prescaler to 8

          bset TIOS,#$01 ; bit 0 is Output Compare

again0:   ldd  TCNT
          addd #3000  ;start an output-compare operation
          std  TC0     ;with 1 ms time delay
wait_lp0:brclr TFLG1,#$01,wait_lp0

          dbne y,again0

          puld
          rts
```



Generating a Siren 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.

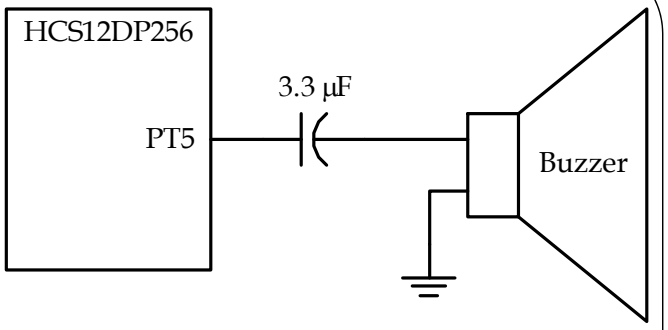


Figure 8.21 Circuit connection for a buzzer

Write a program to generate a two-tone siren that oscillates between 300 Hz and 1200 Hz.

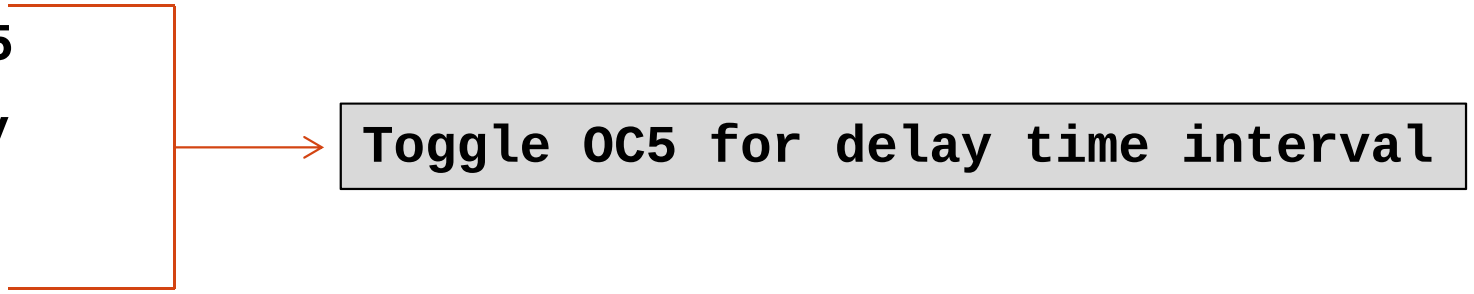
- Prescale = 8.
- The delay count for the low frequency tone is (24MHz ÷ 8) ÷ 300 ÷ 2 = 5000.
- The delay count for the high frequency tone is (24MHz ÷ 8) ÷ 1200 ÷ 2 = 1250.
- These delays are used for the low and high

Steps

1. Configure channel 5 as output compare and enable its interrupts.
2. Output compare is toggle
3. Set delay = 1250 for half a second and change it to 500 for half a second
4- Output compare interrupt routine use the value of delay to set the output signal high and low values.

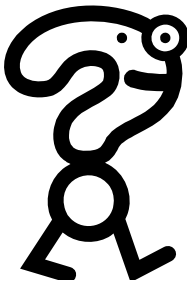
```
hi_freq equ 1250  ; delay count for 1200 Hz
lo_freq equ 5000  ; delay count for 300 Hz
 org $1000
delay ds.w 1 ;store the delay for output-compare operation
org $FFE4
 dc.w oc5_ISR  ;Initialize the interrupt vector entry

 org $2000
 lds #$2000
 movb #$90,TSCR1    ;enable TCNT, fast timer flag clear
 movb #$03,TSCR2    ;set main timer prescaler to 8
 bset TIOS,#%00100000 ;enable OC5
 movb #$04,TCTL1  ;select toggle for OC5 pin action
 ldd  #hi_freq
 std  delay       ; use high-frequency delay count first
 ldd TCNT         ; start timer
```

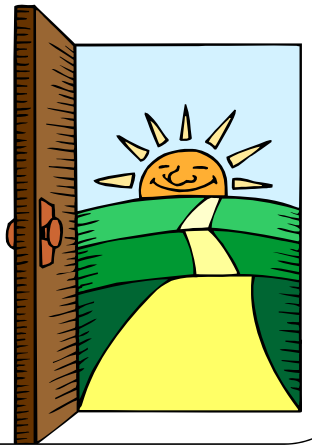
```
 addd delay      ; "
  std TC5   ;       "
  bset TIE,#%00100000; enable OC5 interrupt
  cli                       ;       "
Forever: ldy #500 ; wait for half a second
       jsr Delayby1ms       ;       "
       movw #lo_freq,delay  ;switch to low-frequency delay count
       ldy #500
       jsr Delayby1ms
       movw #hi_freq,delay ;switch to high-frequency delay count
       bra forever



oc5_isr: ldd TC5
       addd delay
       std TC5
       rti
```



Thank You!

Questions





Mohamed Mahmoud