

# Timing Generation & Measurements

## Objectives

- input capture 

to
to

 generate interrupts  
to measure period or pulse width
- output compare to create periodic interrupts  
generate square waves
- Both to make flexible & robust measurement systems

## Facts

- ① The timer systems on Motorola  $\mu$  controllers are very versatile
- |     |
|-----|
| i.e |
|-----|

 inexpensive yet powerful embedded systems have been made possible by the capabilities of timer systems.

PA3 OC5 / OC1  
 PA4 OC4 / OC1  
 PA5 OC3 / OC1  
 PA6 OC2 / OC1  
 PA7 PA1 / OC1

PA0 → IC3  
 PA1 → IC2  
 PA2 → IC1

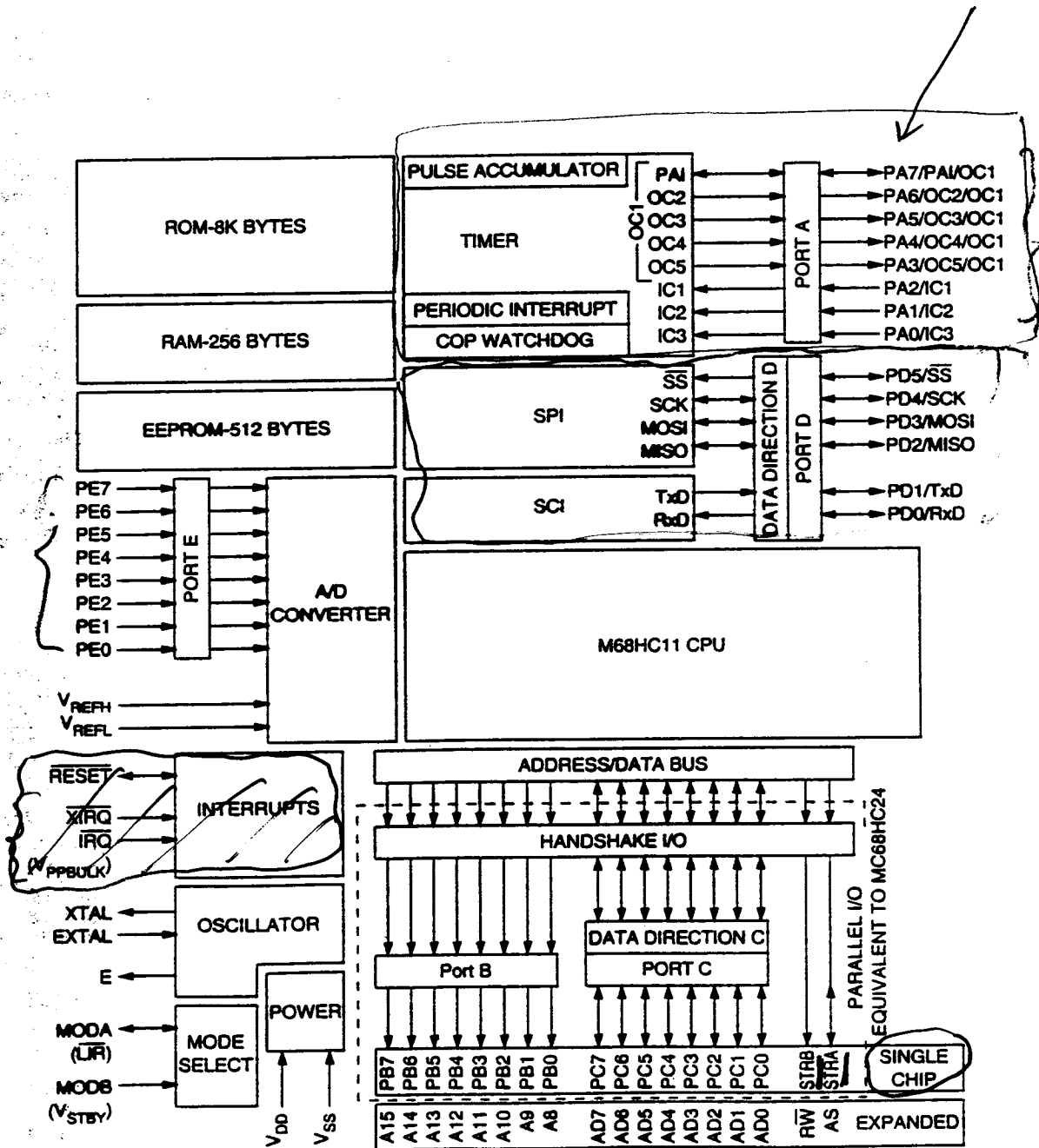


FIGURE 1.2 Internal block diagram of the MC68HC11A8 microcontroller (Courtesy of Motorola, Inc.)

# programmable timer operations:

## Registers

This subsystem has more registers than the others

### Control Registers

TCTL1	Timer control register 1
TCTL2	" " " 2
TMSK1	Main timer interrupt mask register 1
TMSK2	Miscellaneous timer interrupt mask reg 2
PACTL	pulse accumulator control register
OC1M	Action mask register
OC1D	Action Data register

### Data Registers

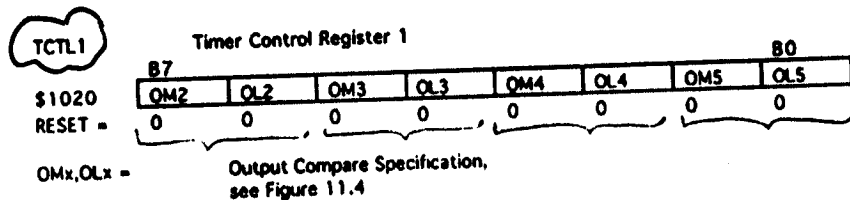
TCNT	Timer counter register
TIC1- TIC3	Timer input capture registers 1, 2, 3
TOC1- TOC5	Timer output compare registers 1, 2, 3, 4, 5
PACNT	pulse accumulator count register

### Status registers

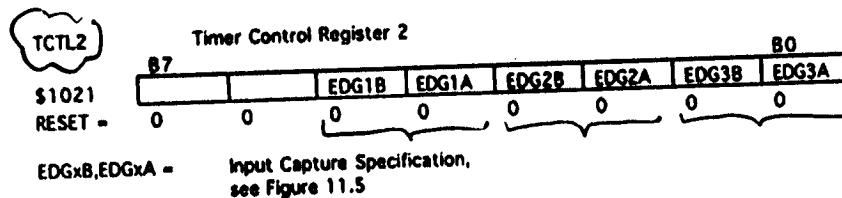
TFLAG1	Main timer interrupt flag reg 1
TFLAG2	Miscellaneous timer flag register 2

**FIGURE 11-1** Bit assignments for the Timer Control, Mask, and Flag Register

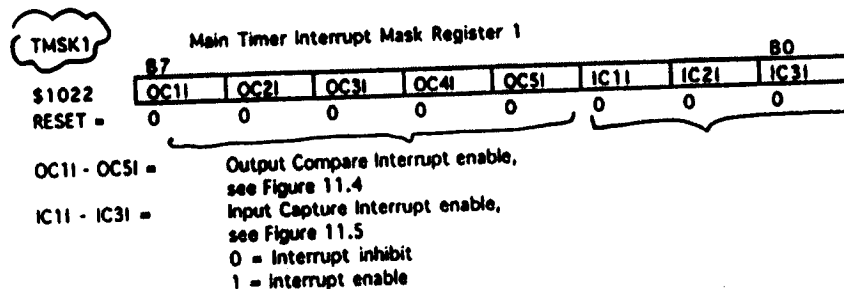
\* Allows user to program  
OCx, OCE



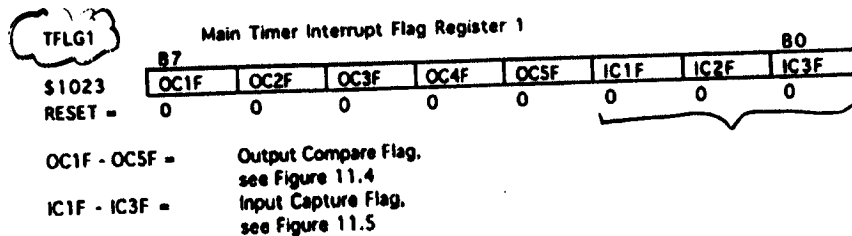
\* A specified edge at an IC pin causes TIC register to latch the free running count



\* CPU enables IC interrupt by writing a binary '1' to TMSK1 bits 0, 1, 2



\* Each input capture function has its own status flag. This flag sets when its counterpart input capture pin toggles with a specified edge



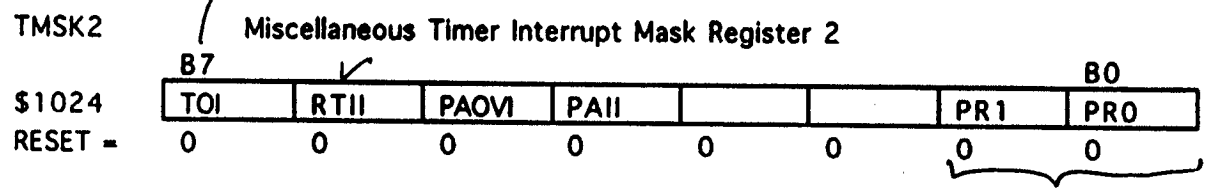
• Write with bit(s) set to clear corresponding flag(s).

OMx	OLx	OCx Action
0	0	no action
0	1	Toggle OCx
1	0	clear OCx
1	1	set OCx

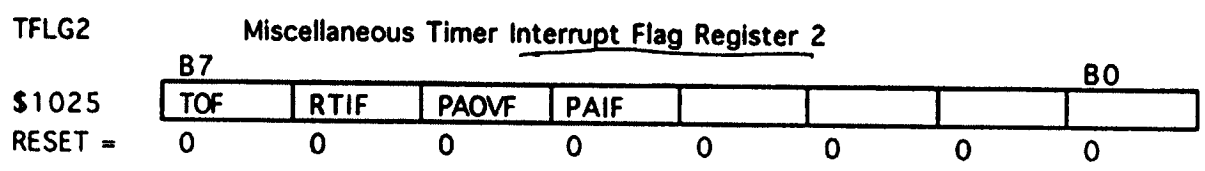
EDGxB	EDGxA	action
0	0	capture disabled
0	1	↑
1	0	↓
1	1	both

FIGURE 11-1 (continued)

*Enable interrupts each time  
free running count rolls over  
from \$FFFF to \$0000*



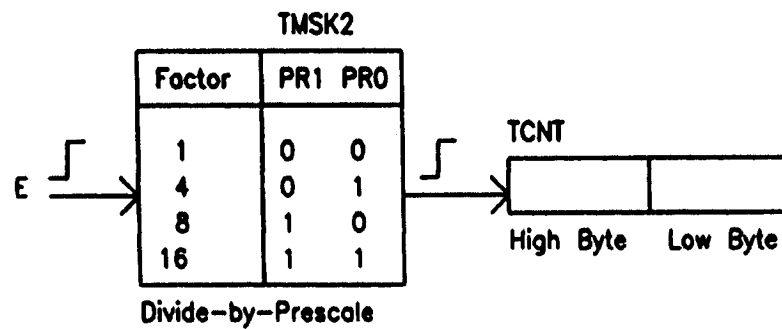
- TOI = Timer Overflow enable, →
- RTII = Real Time Interrupt (RTII) enable
- PAOVI = Pulse Accumulator Overflow Interrupt enable,  
see Figure 11.9
- PAII = Pulse Accumulator Interrupt enable,  
see Figure 11.9
- 0 = Interrupt inhibit
- 1 = Interrupt enable
- PR1, PRO = Timer Prescale select (Time protected), ✓  
see Figure 11.2



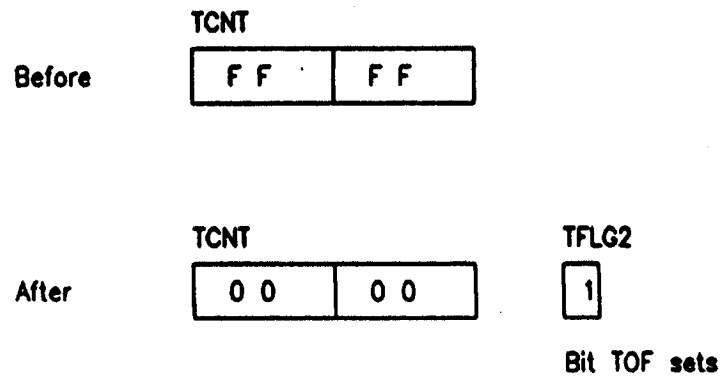
- TOF = Timer Overflow Flag,  
see Figure 11.2
- RTIF = Real Time (periodic) Interrupt Flag
- PAOVF = Pulse Accumulator Overflow Flag
- PAIF = Pulse Accumulator Input edge Flag

\* Write with bit(s) set to clear corresponding flag(s).

**FIGURE 11-2 Free-Running Counter (TCNT)**



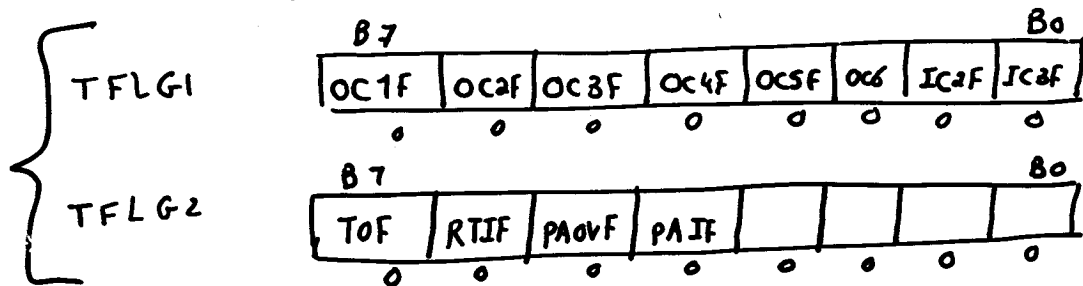
**(a) Counter System**



**(b) Overflow Action**

## Clearing Timer Flags

The flag bits in both registers TFLG1 & TFLG2



are cleared by writing a '1' to the bit to be cleared!

e.g.  
clears timer overflow flag TOF

{	L D A A	# \$80
	S T A A	T F L G 2, X

## Time protected Bits

Timer prescale bits PRI & PRO are time protected bits  
i.e. these bits can only be modified once

Any modification must be done within the first  
64-cycles after reset.

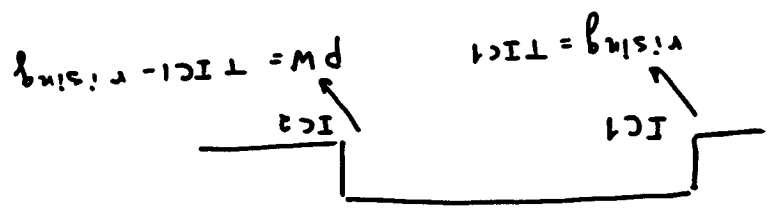
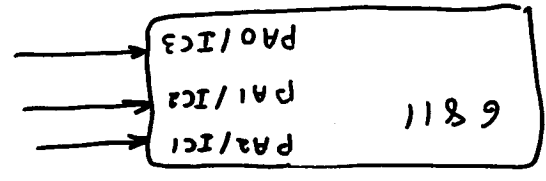
Recall if we set IRQE bit in register OPTION  
then IRQ can be made edge sensitive.

"  
Time protection is provided for those bits whose  
function is considered to be critical.

# Input capture

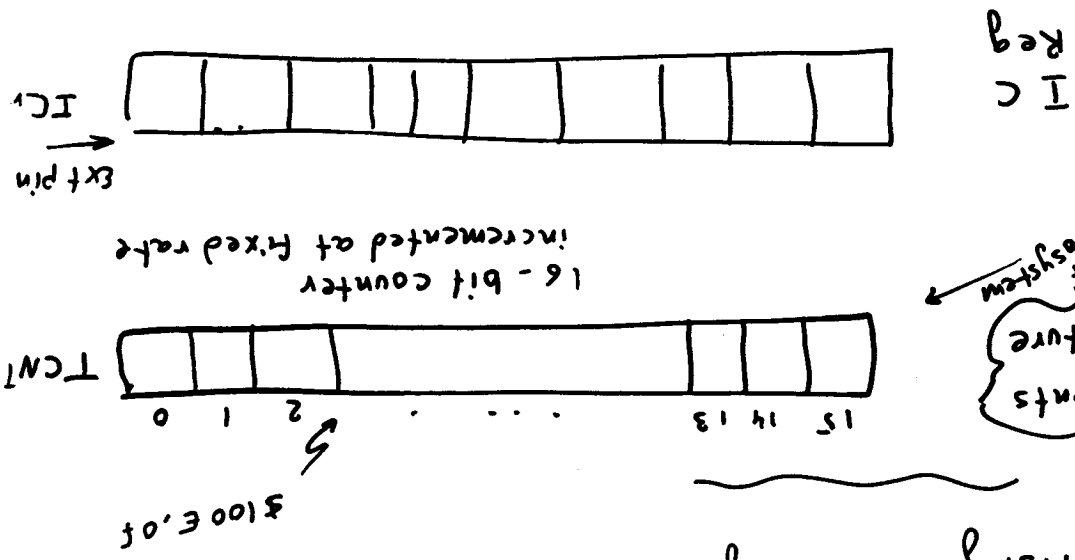
Basic principles

- We can use input capture to measure the period or pulse width of TTL level signals



The input capture system can also be used to trigger interrupts on rising or falling transitions of external signals

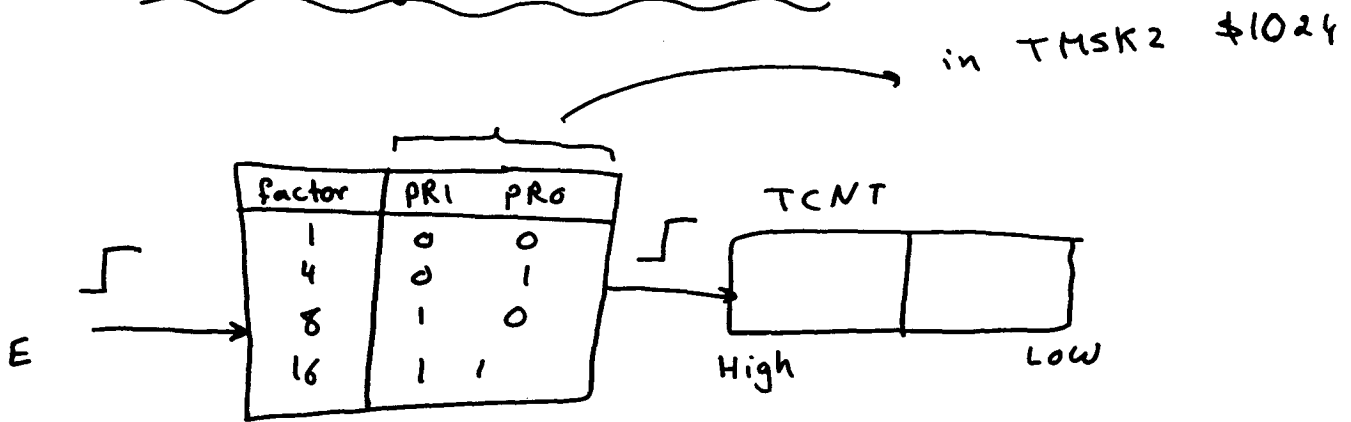
Basic components of input capture timer subsystem



will trigger capture specify whether  $\uparrow$  or both

interrupt mask

## Free running counter TCNT



- ① when MCU is reset, TCNT is also reset to zero
- ② a program cannot write to the register
- ③ only system clock can drive TCNT  
↳ rate determined by PRI PRO
- ④ TCNT is a 2-byte (16-bit) register, it can increment up to 65,535 before it overflows
- ⑤ An overflow occurs every time TCNT counts past \$FFFF to \$0000.

This causes overflow flag (TOF) in register

TFLG2 \$1025 to set

**Note** PRI, PRO are time protected bits (only modified once within the first 64 clock cycles after reset)

\* Our software can determine if an input capture event has occurred by reading TFLG1

0	1	2	3	4	5	6	7
IC3F	IC2F	IC1F	OC5F	OC4F	OC3F	OC2F	OC1F
PA0	PA1	PA2					

\$1023

\* We can arm or disarm input capture interrupt by initializing THSK1 register

0	1	2	3	4	5	6	7
IC3I	IC2I	IC1I	OC5I	OC4I	OC3I	OC2I	OC1I
PA0	PA1	PA2					

\$1022

\* we specify the active edge (i.e. edge that latches T CNT and sets flag) by initializing ICTLA register.

0	1	2	3	4	5	6	7
EDG3A	EDG3B	EDG2	EDG2	EDG1	EDG1	EDG1	0
PA0 mode	PA1 mode	PA2 mode					

\$1021

Active edge	EDGnA	EDGnB
capture on rising	0	0
capture on falling	1	1
capture on both	1	1

\* we specify the active edge (i.e. edge that latches T CNT and sets flag) by initializing ICTLA register.

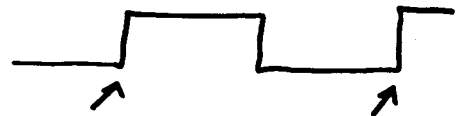
Two or three actions result from capture event

- ① current TCNT value will be copied into input capture register
- ② input capture flag is set
- ③ An interrupt is requested (if mask = 1)

### Three common applications

- ① Arm the flag bit so that an interrupt is requested on active edge of an external signal

- ② perform two rising edge input  
capture and subtract the two measurements to get period



- ③ perform a rising edge capture then a falling edge capture and subtract the two measurements to get pulse width



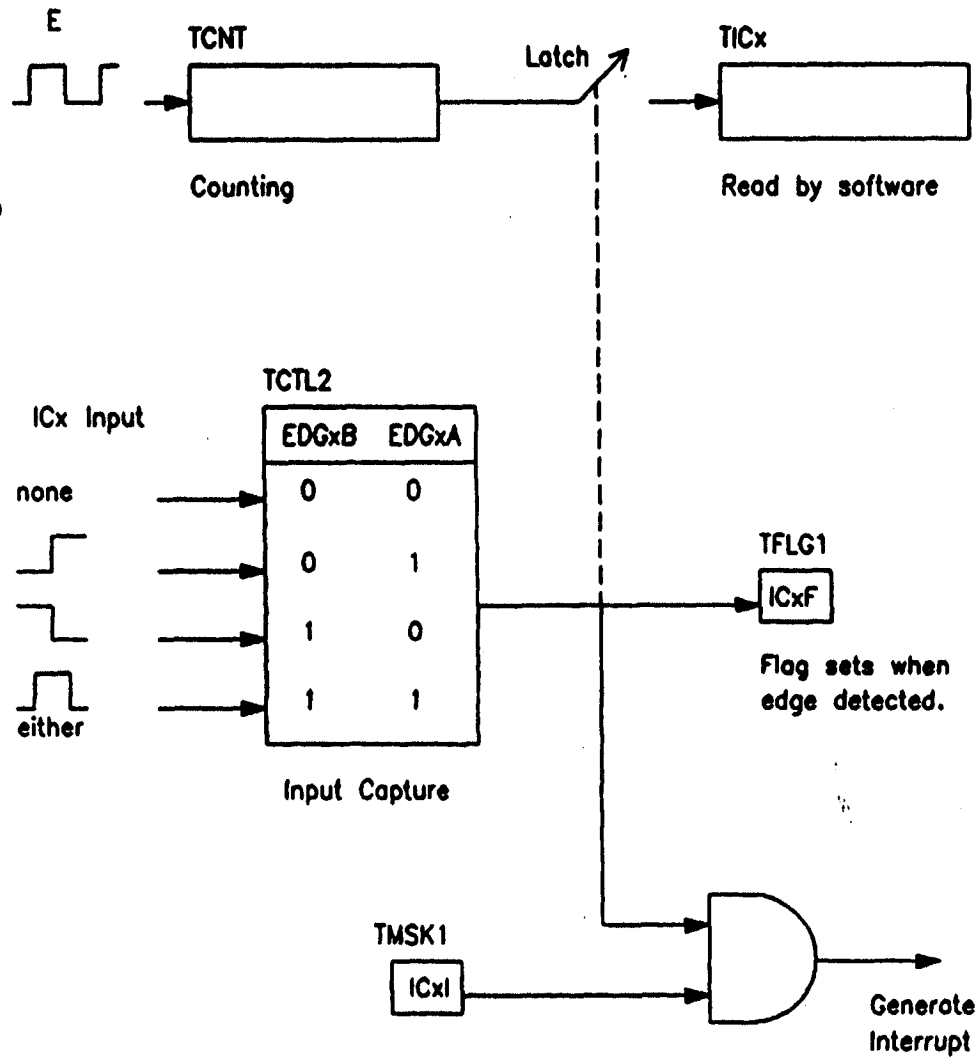
**FIGURE 11-7** Input Capture Sequence

① when edge is detected by IC pin

TCNT → latched into TICx

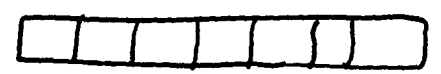
ICxIF flag is set

if ICxI interrupt enable bit is set the detected edge also generates an interrupt



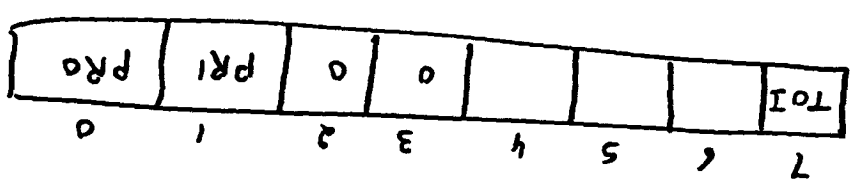
# M68HC11 input capture details

The TCNT (16-bit) unsigned counter is incremented

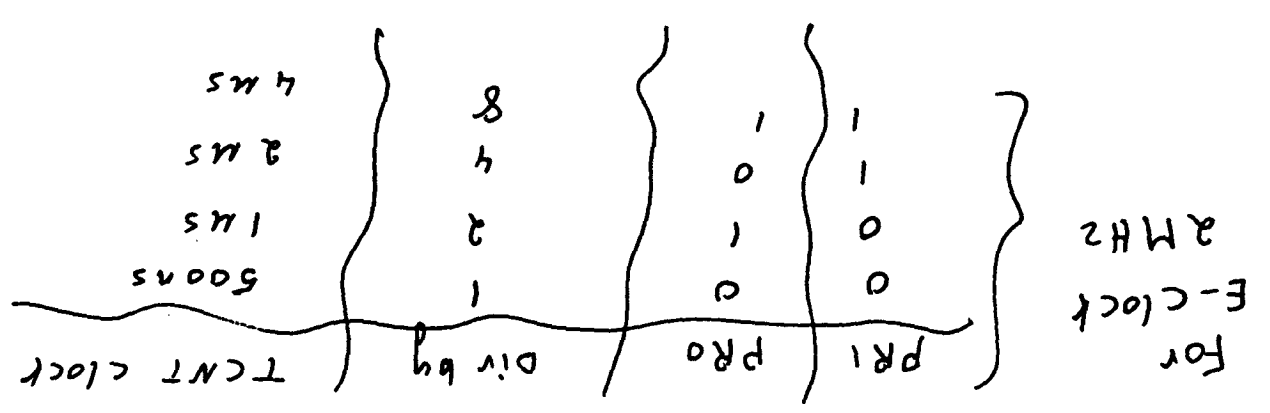


cannot be stopped or reset

at a rate determined by two bits (PRI, PRO) in THSK2 register



\$/024



# Fundamental approach

- connect an external TTL signal to PA0/PA1/PA2

- event occurs in  $\sqrt{\quad}$  or  $\downarrow$  or both

- 6811 will set input capture flag (IC1F, IC2F, IC3F)

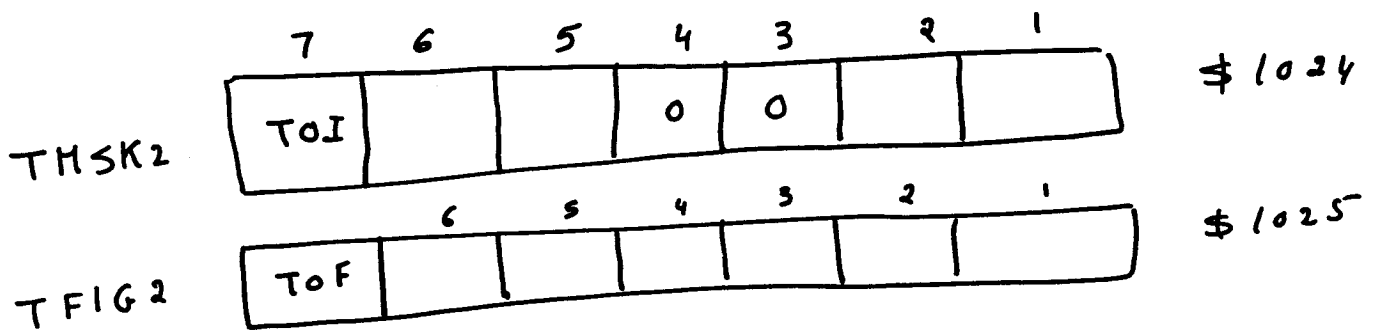
& latch the current 16-bit TCNT value into

the input capture latch (TIC1, TIC2, TIC3)

\* if input capture flag is armed  $\rightarrow$  interrupt

What happens when TCNT register overflows from \$FFFF to 0?

The TOF flag in the TFLG2 register is set  
also The TOF Flag condition will cause interrupt (iF)  
TOI = 1



How do we clear IC1F, IC2F, IC3F flags?

By writing a 1 into the specific flag bit

## pulse width example

• configure: capture on rising edge & clear capture flag

```
wait_for_rise
    If flag == 1,          then rise_time = capture time
                          else repeat wait_for_rise
```

Configure: capture on falling edge  
Clear capture flag

```
wait_for_fall
    If flag == 1,          then fall_time = capture time
                          else repeat wait_for_fall
pulse_width = fall_time - rise_time
```

IMP

→ There are limitations. If the pulse is too short, the second edge occurs before the software can capture it. One way to remedy this is to use two input capture functions. One is configured for a rising edge and the other is configured for a falling edge. If the pulse is longer than 65,536 TCNT cycles, overflow will give erroneous results. The program will have to count how many times the timer overflow bit (bit TOF in register TFLG2) has set.

We can write the pseudocode of Listing 11.7 in assembly language (Listing 11.8).

\*Listing 11.8

\*Measure time between a rising and a falling edge on IC1.

```
*-----
RISETIME    EQU    $10
PULSEWIDTH  EQU    $12

                ORG    $100

                LDAA   #$10        ;config. to capture rising edge
                STAA   TCTL2,X
                LDAA   #$04        ;clear flag IC1F if set
                STAA   TFLG1,X

*wait for rising edge
POLLRISE

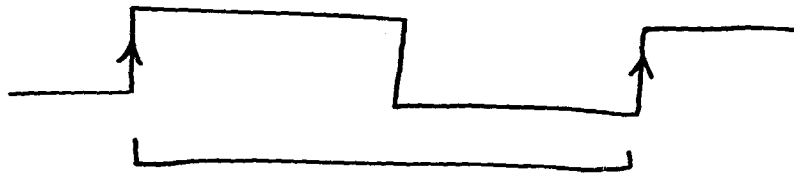
                BRCLR  TFLG1,X $04 POLLRISE
                LDD   TIC1,X        ;store the rise time
                STD   RISETIME
                LDAA   #$20        ;config. to capture falling edge
                STAA   TCTL2,X
                LDAA   #$04        ;clear flag IC1F
                STAA   TFLG1,X

*wait for falling edge
POLLFALL

                BRCLR  TFLG1,X $04 POLLFALL
                LDD   TIC1,X        ;read the fall time
                SUBD  RISETIME      ;width = fall - rise
                STD   PULSEWIDTH    ;store width
                BRA   *             ;stop here for now
```

Example

# period (Frequency) Measurement



measure or capture time of two successive

```
ORG 0

IC1DUN   RMB 1 ; flag: 0 → not done, 1 → ✓
IC1MOD   RMB 1 ; mode flag: FF - off, 0 → 1st, 1 - last
PER      RMB 2 ; period 16 bits
```

```
ORG $180
```

\* subroutine INITC1

\* configured input capture IC1 to measure high pulse

\* using interrupts

## INITC1

```
PSHA ; preserve register
LDA A # $10 ; EDGIB: EDGIA = 0:1 for rising
STAA TCTL2, X
LDA A # $FF ; mode flag off
STAA IC1MOD
CLR IC1DUN ; signal period not done
BCLR TFLG1, X, $FB ; clear ICIF if set
BSET TMSK1, X, $04 ; enable IC1 interrupt
CLI ; enable interrupts
PULA ; restore registers
RTS ; return.
```

\* service routine RTIC1

(imp?) invoked when  
system detected  
edge ↓

RTIC1

```
L D X    # REG BAS    ; point to registers
I N G    IC1 MOD      ; $FF → 0 at first edge
*                               ; 0 → 1 at 2nd edge
B N E    EDGE2        ; if not 0 then this is 2nd
```

\* process first edge

EDGE1

```
L D D    TIC1, X      ; capture first edge time
S T D    PER          ; and save it
L D A A  # $04        ; clear flag IC1F
S T A A  TFLG1, X    ;
R T I                               ; return
```

\* process second edge

EDGE2

```
L D D    TIC1, X      ; capture 2nd edge time
S U B D  PER          ; second time - first time
S T D    PER          ; is low 2 bytes of result
B C L R  TCTL2, X, $30 ; stop recognizing edges
L D A A  # 1          ;
S T A A  IC1DUN       ; } ; one pulse measured
B C L R  TFLG1, X, $FB ; clean flag IC1F
R T I                               ; return from ISR
```

## Typical applications

- \* Rotating Mechanical Equipment have sensors that generate a pulse for every fraction of rotation. with this input, MCU can use the input capture function to measure speed, detect motion & record the distance traveled

## Automobiles

- Antiskid braking system (ABS)  
system senses wheel rotation to determine the braking pressure to apply.
- Traveling speed of the automobile is measured by sensors attached to the wheels.  
MCU can use this for cruise control
- Ignition timing signals in the engine

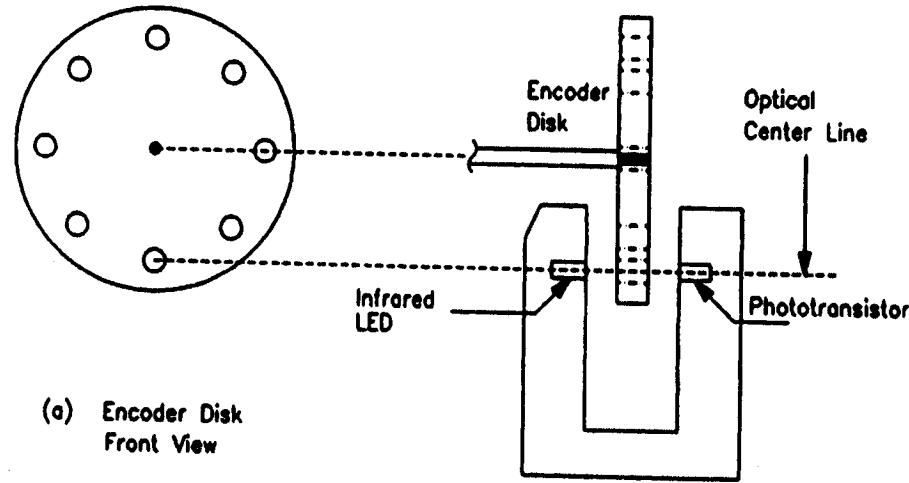
FIGURE 11-8 Optoelectronic Position/Motion Detection

\*  
 ① In many cases the MCU uses input capture to monitor and/or control speed.

② A common type of speed sensor is an encoder disk mounted on a shaft

③ Disk is located between a slotted optical switch

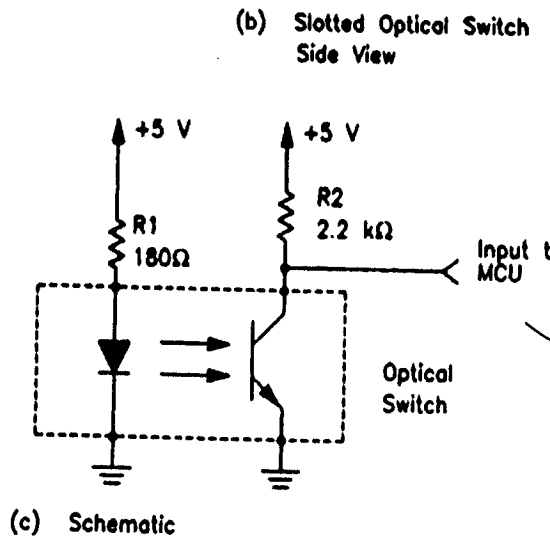
④ when encoder disk rotates the switch outputs pulses



eight pulses correspond to one disk revolution so pulse frequency is related directly to speed of revolution

\* so we can control speed accordingly.  
 PWM

LED emits infrared light



Encoder disk has holes that allow light to pass through phototransistor switches on only when disk rotates to a position such that a disk hole is aligned between led & trans.

⑤ To measure forward or reverse (bidirectional) speed use dual slotted optical switch!

# issues

① zero speed!

use timer overflow  
if time between pulses  $> 65,536$

② pulse frequency too high!

→ measurement of time  
between pulses becomes  
imprecise.

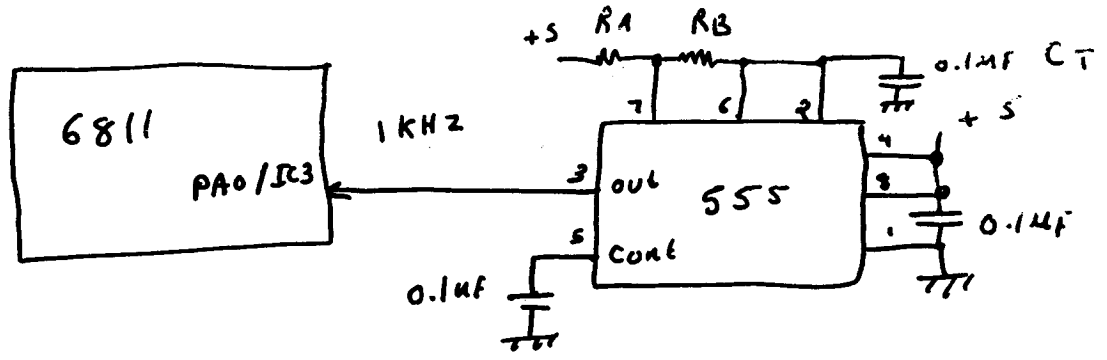
# motion detection

③ Antiskid braking system!!

To prevent skidding it is necessary  
to prevent any of the wheels from locking up.

Real Time interrupt using input capture

$R_A = 4.4k$   
 $R_B = 5k$



period of 555 is  $0.693 \times C_T \times (R_A + 2R_B)$



"An input capture interrupt occurs on each rise of the square wave generated by 555"

Other applications

→ see book section 11.3.2 p. 441  
 For pulse width Measurement

→ see book section 11.3.3 p. 443  
 For period measurement

# Pulse Accumulator

PA is an 8-bit counter that can count input edges or measure pulse width depending on operating mode.

## Typical applications

- ① monitoring tape position in VCR or automobile odometer
- ② monitoring total volume of fluid flowing through a flow meter

\* The pulse accumulator has one input pin (PAI) at port A pin PA7 (Bidirectional) direction can be determined by (PACTL) "pulse accumulator control register"

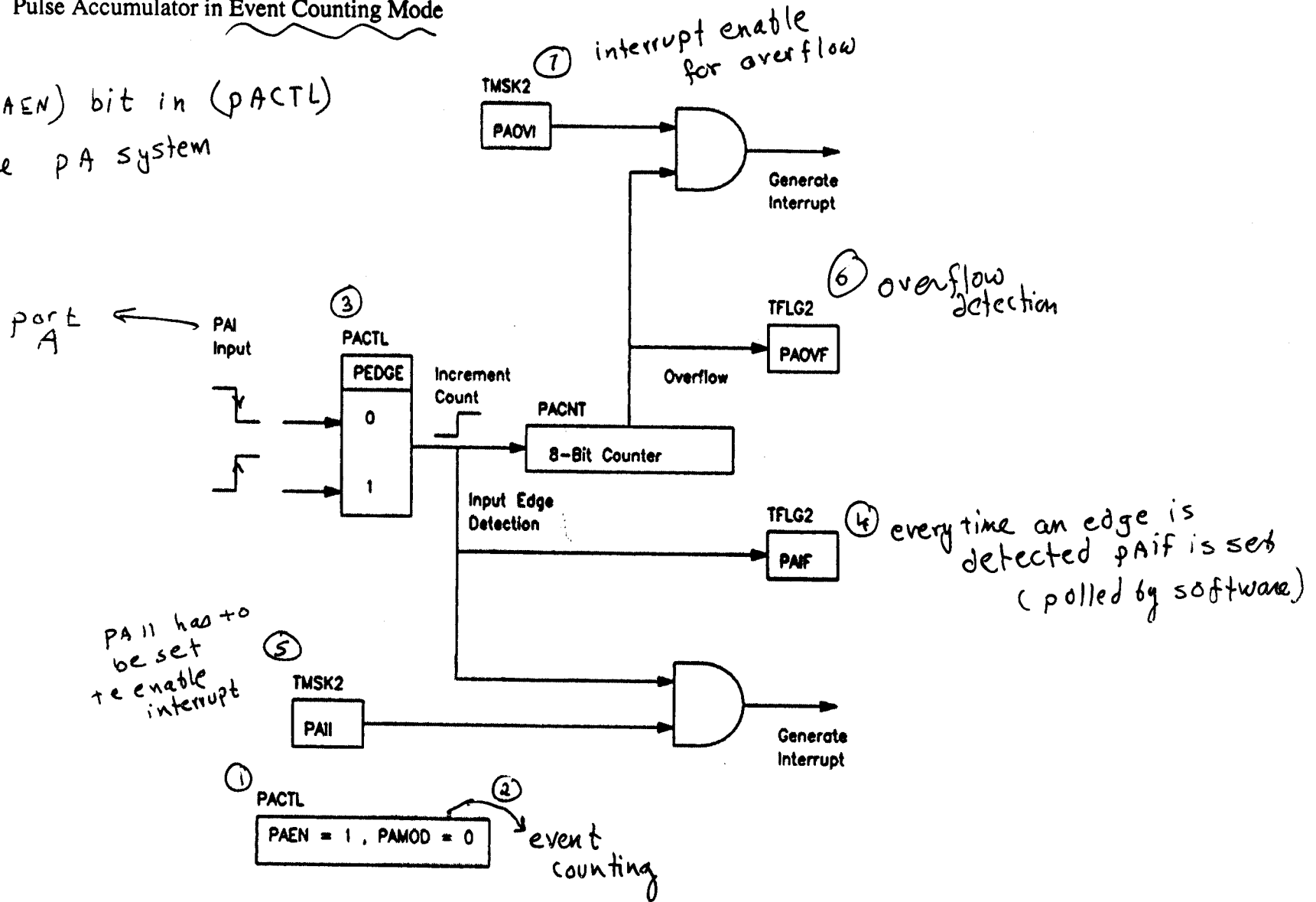
\* PACTL has a pulse Acc system enable bit (PAEN) which must be logic 1

\* PAMOD, PEDGE bits establish the functions operating mode.

\* PACNT (count register) can be read or written to.

**FIGURE 11-9** Pulse Accumulator in Event Counting Mode

① setting (PAEN) bit in (PACTL) enables the PA system



## I) Short counts

The pulse accumulator can be setup to produce an interrupt after a preset number of edges have been detected. (simple! if preset is  $< 256$ )

\*\*\* write two's complement of preset to PACNT.  
when preset number of events have been detected PACNT would have counted past \$FF to cause overflow condition.

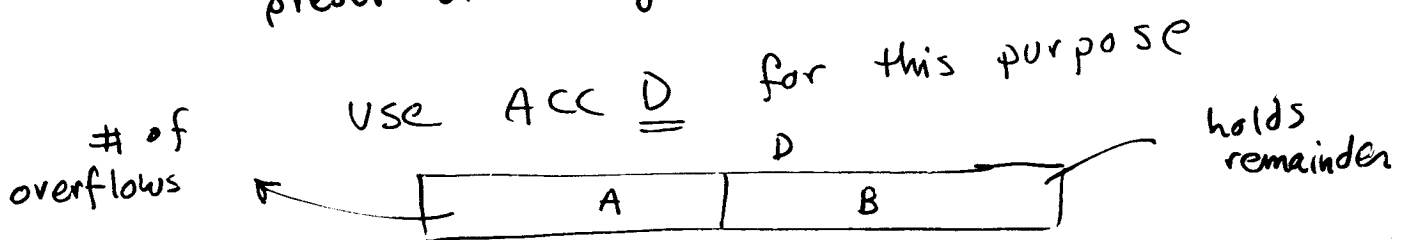
Example 24  $\approx$  (\$18)  $\rightarrow$  2's comp \$E8

## II) Longer counts

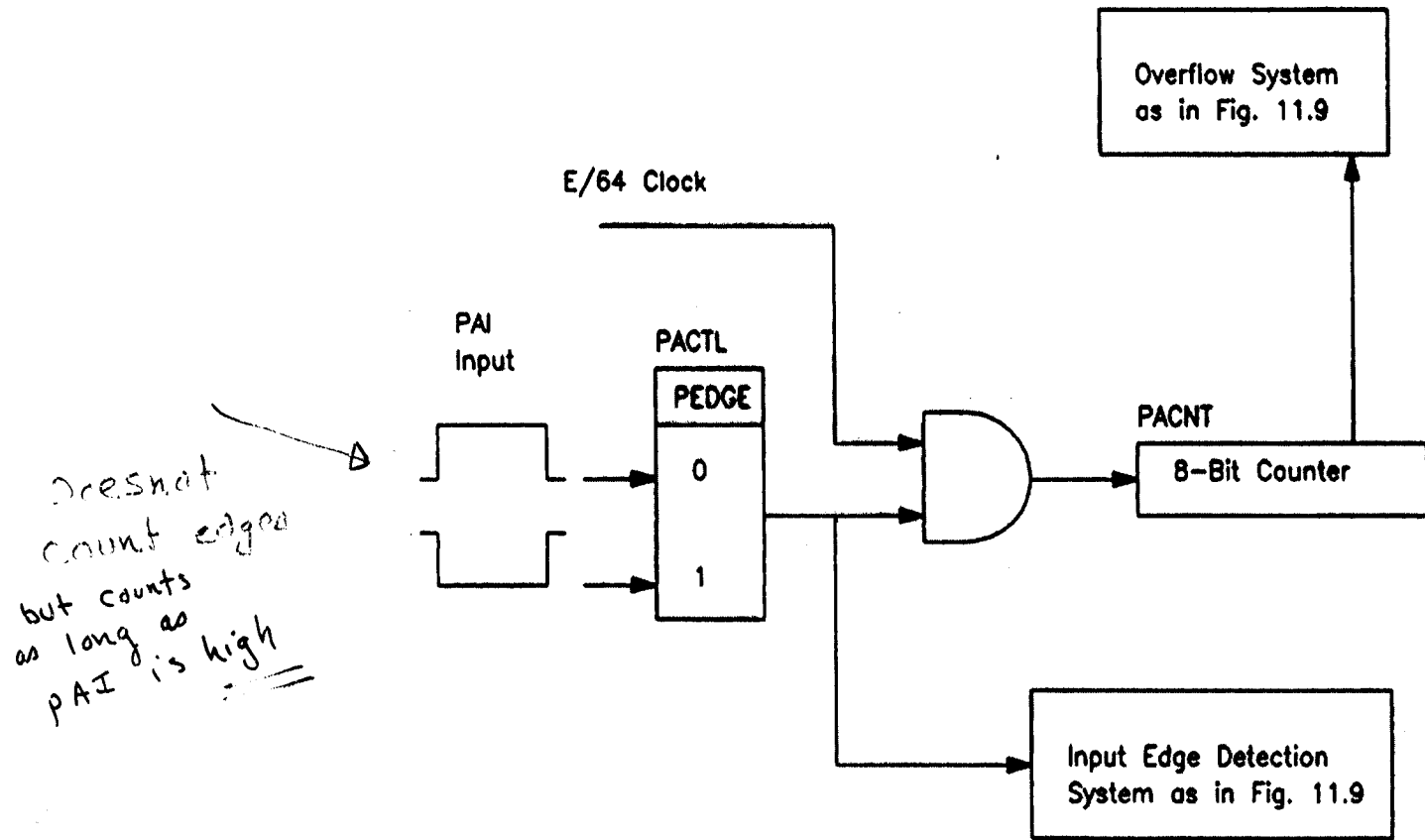
More than 256 events can be counted by using software to keep track of the number of pulse accumulator overflows.

\* If counter starts at zero, 256 events will cause an overflow.

So the number of overflows required is the preset divided by 256.



**FIGURE 11-10** Pulse Accumulator in Gated-Time Accumulation Mode



Doesn't count edges but counts as long as PAI is high

e.g. if PEDGE is 0 the counter will not increment as long as input PAI is low!!

PACTL  
PAEN = 1 , PAMOD = 1

PACNT counts once every 64 clock cycles.

Application of Gated time Accumulation



How?

write a value to counter PCNT that is half way

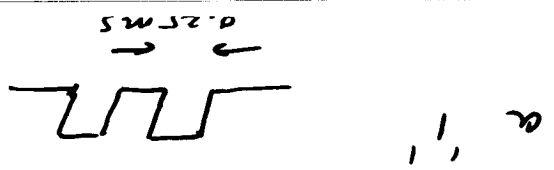
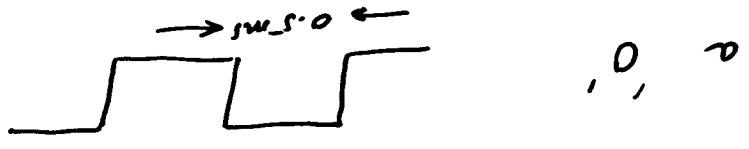
between the values of the narrow & wide pulse widths

● when a narrow pulse occurs (arrives), the counter accumulates and trailing edge is

● detected before overflow occurs when a wide pulse arrives the counter overflows

Audio Cassette Interface

use pulse width to encode data

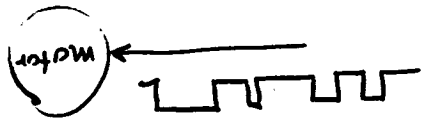


# Output compare

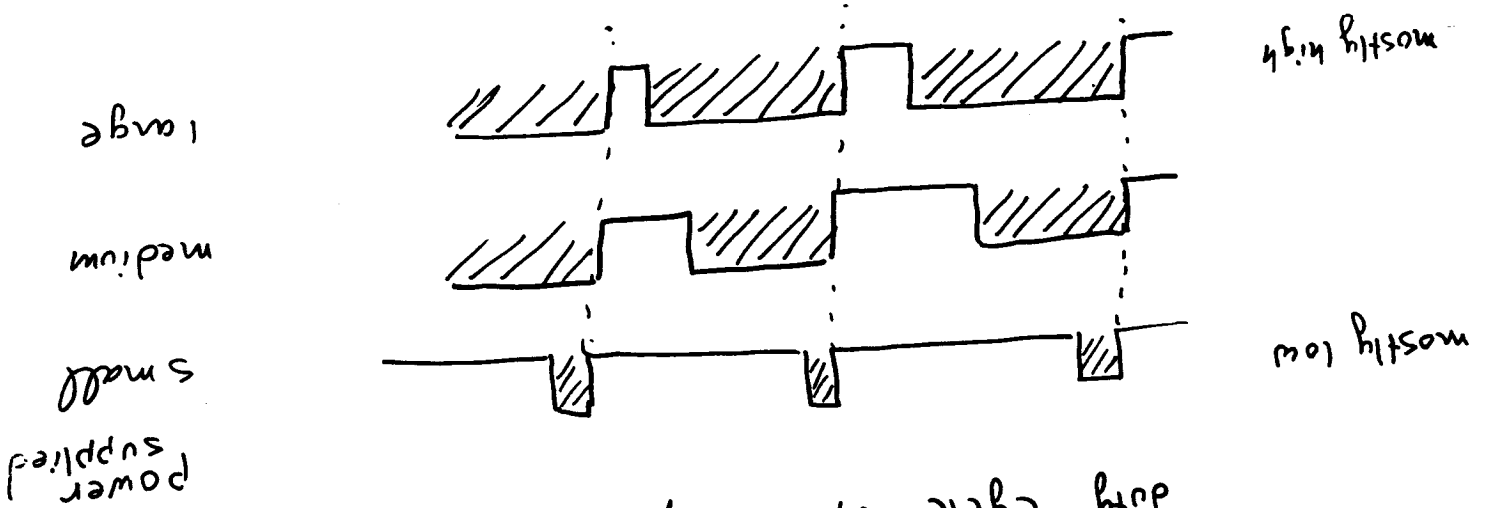
can be used to

- create square waves
- generate pulses
- implement time delays
- execute periodic interrupts

Also PWM pulse width modulation "variable duty cycle"

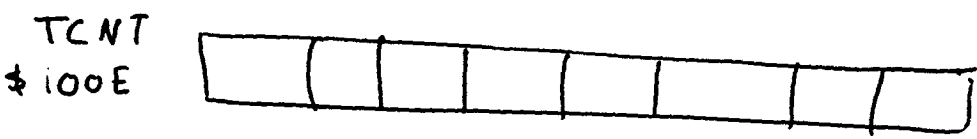


The output power is controlled by varying the duty cycle of a square wave

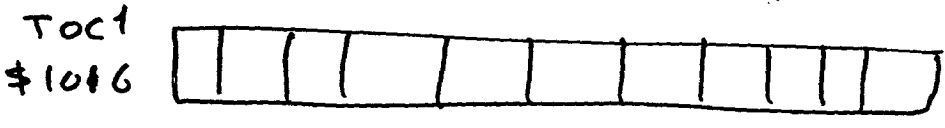


# output compare details

6811

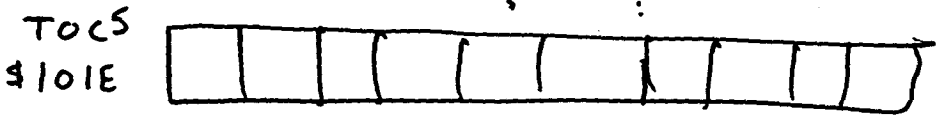


incremented every  
E-clock ; rate  
determined by  
PRI, PRO

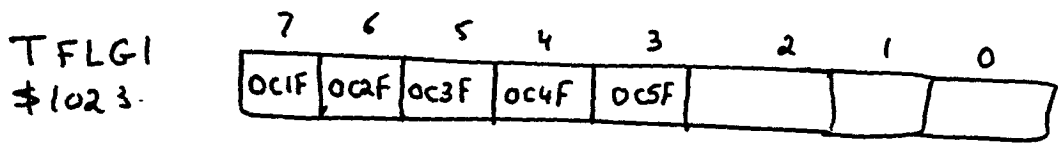


OC1F set when  
TOC1 = TCNT

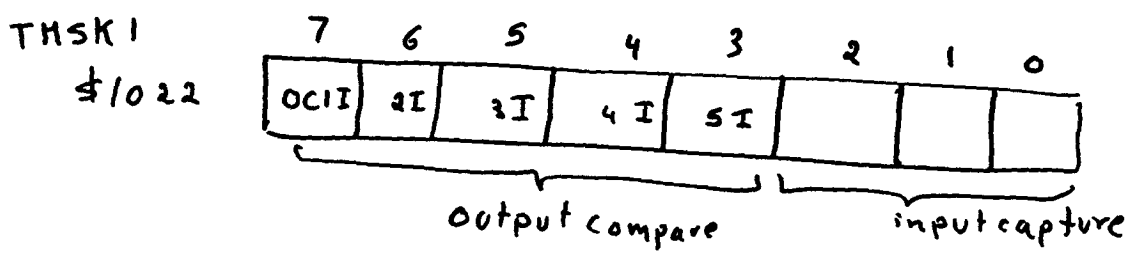
⋮



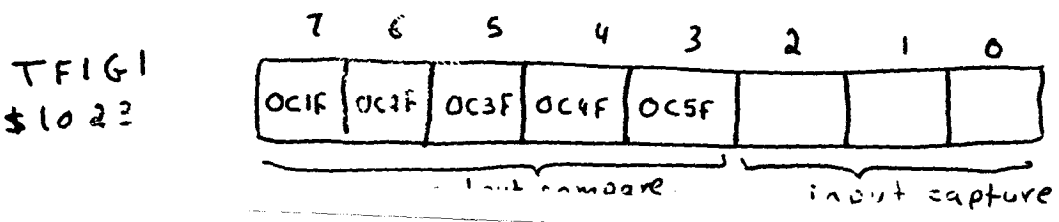
OC5F set when  
TOC5 = TCNT



\* We can arm or disarm the individual output compare interrupts  
by initializing the TMSK1 register

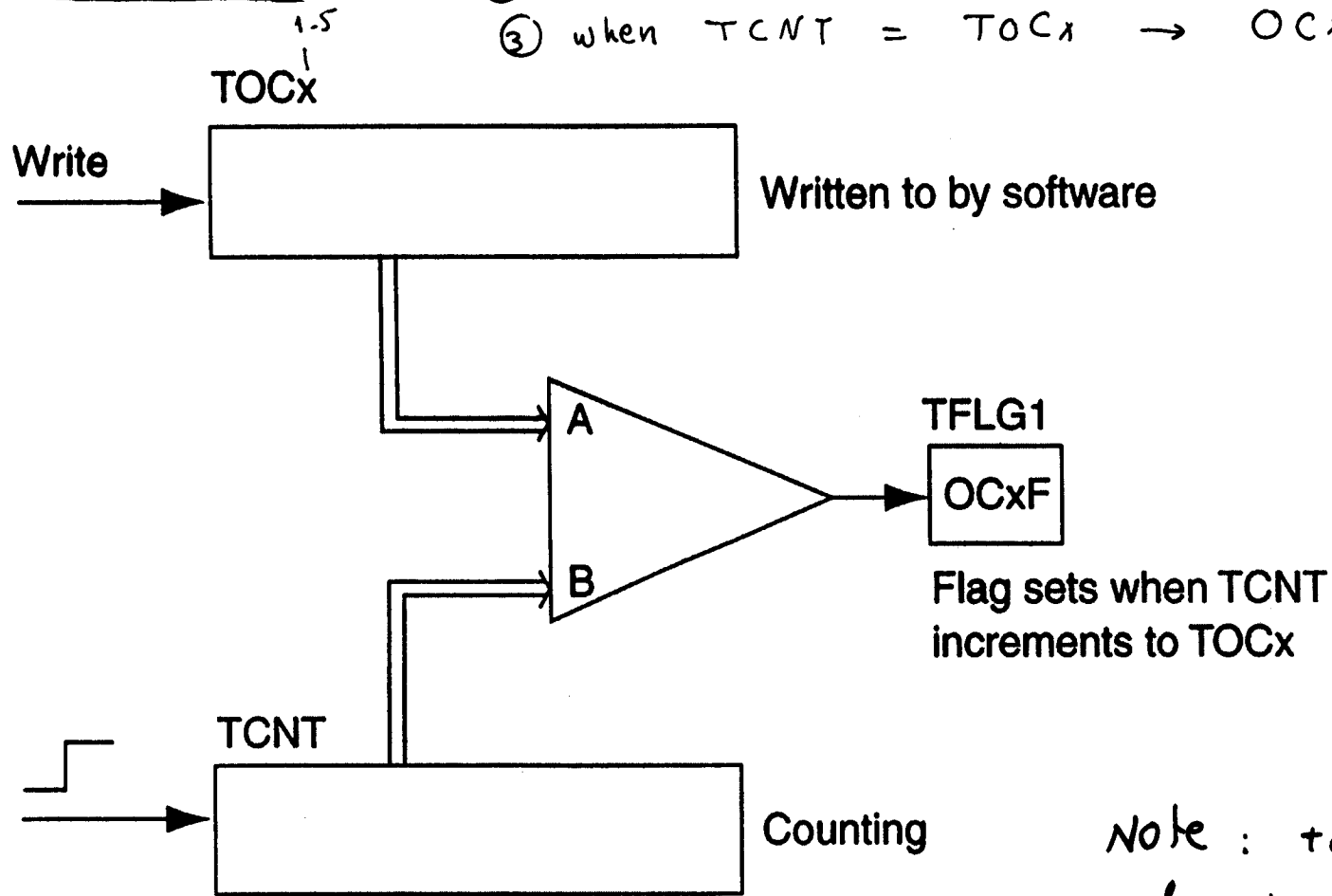


\* software can determine if output compare event has occurred by reading TFIG1 register



imp  
Flags clear  
by writing '1'

FIGURE 11-3 Output Compare Action



- ① recall there are five output compare functions
- ② software writes a value to register (TOCx)
- ③ when  $TCNT = TOCx \rightarrow OCxF$  flag is set

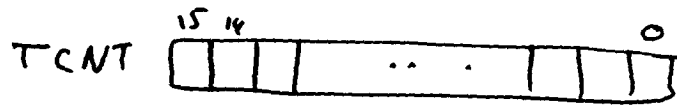
x = 1, 2, 3, 4, 5,

Note : to write a value to TOCx use a double byte inst  
`ST D TOCS,x`

if two separate instructions are used to write 2-bytes there is a possibility of an error

Note

The same 16-bit TCNT incremented at a fixed rate is used for input capture

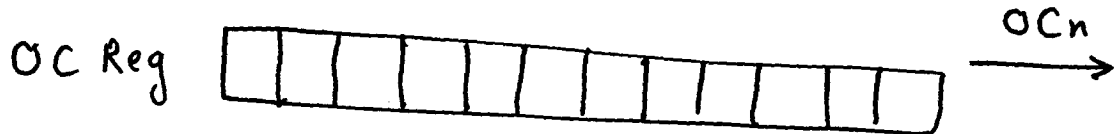


input capture  
output compare

Each output compare module has

6811 has Five output compare modules

1



- 16-bit output compare register
- External output pin  $OC_n$

2



a flag bit — in TFLG1

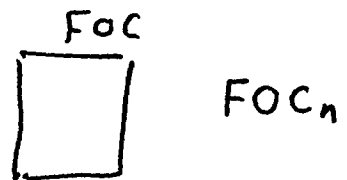
3



an interrupt mask bit → in TMSK1

4

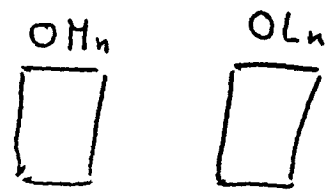
A Force compare control bit

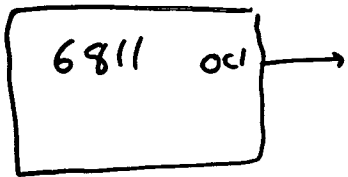


5

Two control bits  $OM_n, OL_n$

in TCTL1





output compare pin is an output of the computer, hence can be used to control an external device.

**Q** When does an output compare event happen?

**Ans** when either

1. The 16-bit TCNT matches the 16-bit OC register
2. software writes a 1 to the FOC bit  
(i.e Force compare control bit)  
& Flag is set

**Q** what is the role of  $OM_n$ ,  $OL_n$  bits?   $OM_n$    $OL_n$

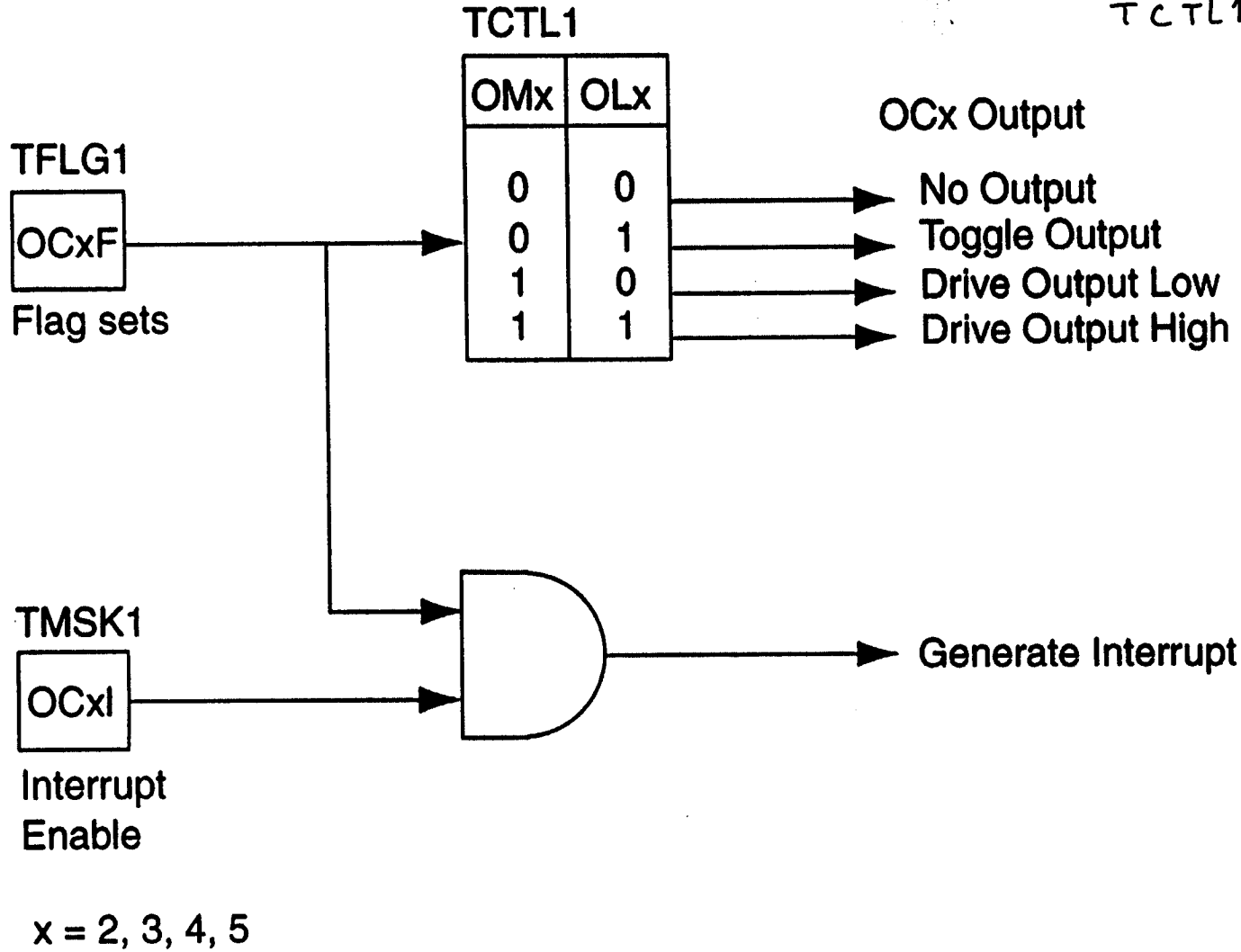
**Ans** they basically specify what affect the output compare event will have on output pin

$OM_n$	$OL_n$	Effect
0	0	Doesnot affect $OC_n$
0	1	Toggle $OC_n$
1	0	clear $OC_n = 0$
1	1	set $OC_n = 1$

FIGURE 11-4

Results of a Successful Compare

① when OCxF sets, it causes an event to occur depending on the configuration setting in Reg TCTL1 & TMSK1



Q] What are the actions resulting from an output compare event?

Ans

- ① The Ocn output bit changes
- ② output compare flag is set
- ③ An interrupt is requested (when mask=1)

Simple Application:

Create a fixed time delay.

steps

- ① Read the current value of 16-bit TCNT
- ② calculate the value  $\boxed{\text{fixed} + \text{TCNT}}$
- ③ set the 16-bit output compare register to  $\boxed{\text{fixed} + \text{TCNT}}$
- ④ clear the output compare flag.
- ⑤ Wait for the output compare flag to be

set → gdfly  
→ interrupt

Important Remarks

① if we perform the addition of TCNT + fixed \$2000 \$5000

so we wish to have a delay = \$2000 → 8192 cycles

the addition will result in a sum = \$1000

so if we put the \$1000 into OC register

obviously it will take correct number

② one of the output compare modules PA7/OC1 can be configured such that

an output compare event on it will cause

changes on some or all other output compare pins

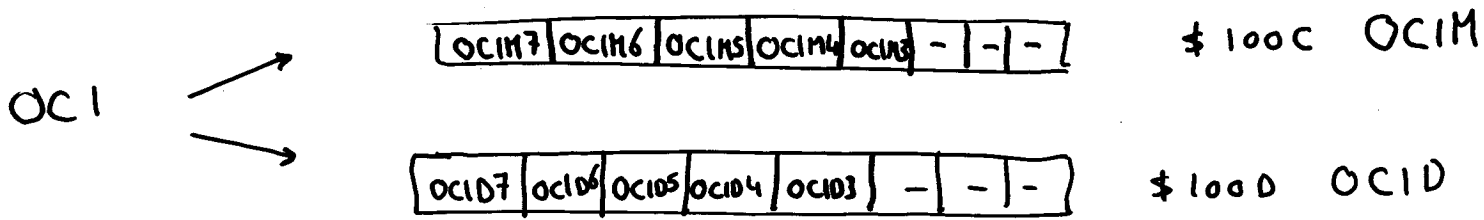
\* This coupled behavior can be used to create synchronized signals.

ie) pulses that start together and together

see p 425 for example.

recall

• Each output compare (OCx) except for OC1 has one pin associated with it



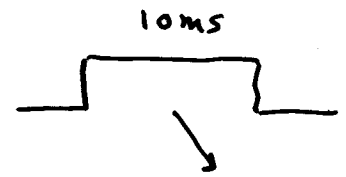
→ OC1M → determines which port A pins OC1 controls  
"1" connects to PA3, PA4, PA5 ...  
"0" disconnects

→ when successful compare occurs data in OC1D is sent to port A pin.

example

```
L DX # REGBAS ; point to registers
BSET PACTL, X, $80 ; make PA7 output
LDAA # $98 ; set OC1M3, 4, 7
STAA OC1M, X ; enable PA3, 4, 7
LDAA # $18 ; set OC1D3, 4 clear OC1D7
STAA OC1D, X ; drive PA3, 4 high ; PA7 low
BSET TMSK1, X, $80 ; enable interrupt
CLI
```

# One shot pulse Example



2-MHZ clock → 0.5 μs every E cycle

```

    ORG    $100

PWIDTH  EQ    20000
    →     LDD    TCNT,X      ; prevent premature
          STD    TOC2,X      ; OC2 compare

PULSE
    →     BSET   PORTA,X,$40 ; drive PA6/OC2 high
          [ LDAA  # $80      ; configure OC2 to clear
            STAA  TCTL1,X    ; & disconnect other OCx's
          [ LDAA  # $40      ; clear OC2F if set
            STAA  TFLG1,X    ; arm TOC2 for ms trigger
          [ LDD    TCNT,X
            ADDD  #PWIDTH-17
            STD   TOC2,X

* wait for trigger

PULSE1
    *     BRCLR  TFLG1,X,$40 PULSE1 ; by polling OC2F high
          →     BCLR  PORTA,X,$40 ; output OC2 low
                                     clear PA6
          [ LDAA  # $40      ; clear OC2F
            STAA  TFLG1,X    ; before
          [ BCLR  TCTL1,X,$80 ; disconnecting OC2
          BRA    *
    
```

# Square wave Generation example



- \*Listing 11.3
- \*Shows routines for generating square waves
- \*using output OC3.
- \*A main application program varies frequency
- \*by modifying 16-bit data in addresses HPERIOD.
- \*Frequency =  $1/(2 * HPERIOD)$
- \*User responsibility to set up vector addresses
- \*and other initialization as required by application.

ORG \$180

```
*Subroutine INITOC3
*Initializes timer output OC3 for
*square-wave output, interrupt driven.
*Calling registers
* IX = register block address
*Return registers
* None, except CCR affected
INITOC3
    PSHA                ;preserve registers
    PSHB
    LDAA #$10           ;OM3:OL3=0:1
    STAA TCTL1,X       ;to toggle OC3
    LDAA #$20           ;clear OC3F if set
    STAA TFLG1,X
    STAA TMSK1,X       ;set OC3I to enable
    CLI                 ;interrupt
    PULB                ;restore registers
    PULA
    RTS                 ;return
```

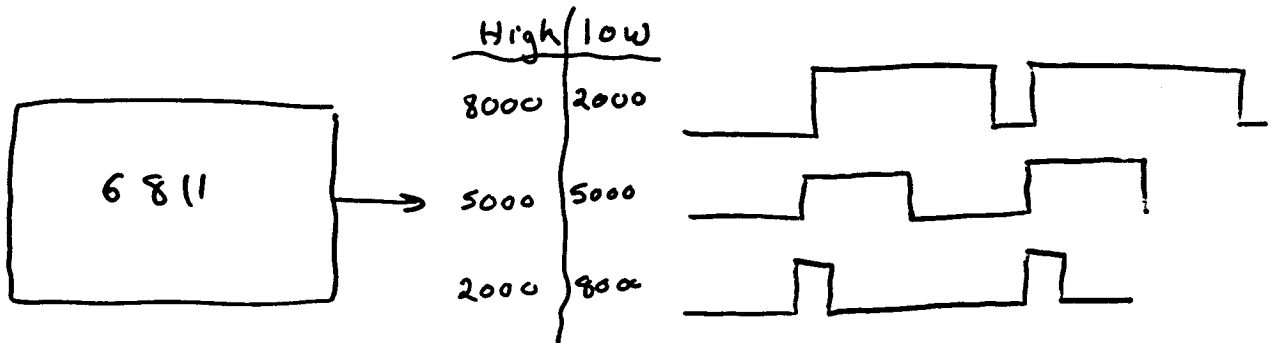
- \*Service routine RTOC3
- \*Drives OC3 output for square wave by scheduling
- \*time delay for next edge to be toggled.
- \*The minimum permissible half-period is the number of clock
- \*cycles in this routine plus those required to enter
- \*and exit the routine.
- \*Static variable (2 bytes)
- \*Address HPERIOD = OC3 half-period time duration
- \*This routine executed after TOC3 == TCNT occurs

```
RTOC3
    LDX #REGBAS        ;point to registers

    LDD HPERIOD        ;update TOC3
    ADDD TOC3,X        ;by adding half period
    STD TOC3,X         ;to latest TOC3 value
*                       ;to schedule the next interrupt
    BCLR TFLG1,X,$DF
*                       ;clear flag OC3F
    RTI                 ;return from service
```

# pulse width Modulation

This example generates a variable duty cycle square wave using output compare



$$\text{Duty cycle} = \frac{\text{high}}{\text{high} + \text{low}}$$

percentage of time the signal is high compared to period

see program in tex book page ~~329~~ 434

High + low will always equal 10,000 cycles  
or 5 ms

- RITUAL** initializes
- number cycles high
  - " " Low
  - TMSK1 reg → arm bits
  - TCTL1 reg → determines effect on OC pin
  - TFIG1 → clears flags

**Handler** see details in text book

**\*Listing 11.4**

- \*Shows routines for handling PWM using output OC2.
- \*A main application program varies the duty cycle by modifying 16-bit data in addresses OC2HI and OC2LO.
- \*Duty cycle = 100% \* OC2HI / (OC2HI + OC2LO)
- \*User responsibility to set up vector addresses
- \*and other initialization as required by application.

ORG \$180

- \*Subroutine INITOC2
- \*Initializes timer output OC2 for PWM output, interrupt driven.
- \*Calling registers
  - \* IX = register block address
- \*Return registers
  - \* None, except CCR affected

```
INITOC2
    PSHA                ;preserve registers
    PSHB
    LDD    TCNT,X      ;delay PWM generation
    STD    TOC2,X
    LDAA   #$C0        ;OM2:OL2=1:1 to set
    STAA  TCTL1,X     ;OC2 high first time
    LDAA   #$40        ;clear OC2F if set
    STAA  TFLG1,X
    STAA  TMSK1,X     ;set OC2I to enable
    CLI   ;interrupt
    PULB  ;restore registers
    PULA
    RTS                ;return
```

- \*Service routine RTOC2
- \*Drives OC2 output for PWM by scheduling time delay for next edge. Also reconfigures next edge opposite to that of current edge.
- \*Note that routine will not work properly with duty cycles close to 0% or 100%.
- \*Static variables (2 bytes each)
- \*Address OC2HI = OC2 time duration for high pulse
- \*Address OC2LO = OC2 time duration for low pulse

\*This routine executed after TOC2 == TCNT occurs

OC2HI EQU \$0006

```
OC2LO EQU $0008

RTOC2
    LDX   #REGBAS    ;point to registers
    *
    *
    BRCLR TCTL1,X,$40 GETOC2LO
    LDD   OC2HI      ; else load OC2HI
    BRA   NEWTOC2

GETOC2LO
    LDD   OC2LO

*update TOC2
NEWTOC2
    ADDD  TOC2,X
    LDAA  TCTL1,X    ;invert OL2 to toggle
    EORA  #%01000000 ;next OC2 edge
    STAA  TCTL1,X    ;by updating control reg.
    BCLR  TFLG1,X,$BF ;clear flag OC2F
    RTI   ;return from service
```

# PWM Functions

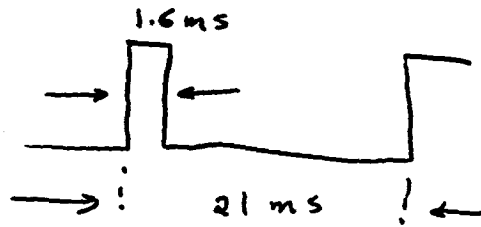
- Output compare functions are flexible but for PWM there is a software overhead required to generate square waves and PWM outputs

MC68HC711K4 has a built in PWM module (port H)

- 4 PWM channels
- 4 control registers to select features such as
  - ① clock source
  - ② clock scaling ..
- each channel has a separate counter, period duty cycle register
- Once these registers are setup, the PWM drives the outputs without further software intervention.
  - ↳ if an application program needs to change the period or duty cycle, it can write to their respective registers.

IMP

what is the fastest square wave that can be generated?



- Output compare interrupts will be requested at a rate twice as fast as the resulting square wave frequency.

\* one interrupt is required for the rising edge and another for falling edge.

To determine the fastest square wave we count the time it takes to process an interrupt

Component	6811
process interrupt	14 cycles
Execute entire Handler	53-56 cycles
total us	70 us

## Stepper Motor outputs

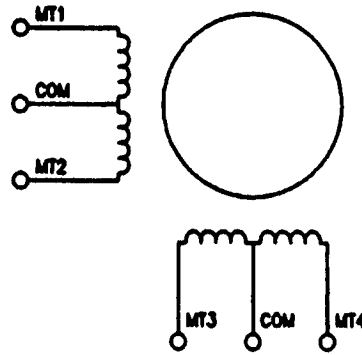
- One application for using output compare functions is the control of stepper motors.
- stepper motors are electric motors that rotate from one position to the next in steps (i.e. discrete intervals)
- A step refers to the fraction of a full  $360^\circ$  of motor shaft  
↳ Typical steps angles are  $1.8^\circ$ ,  $3.6^\circ$ , and  $7.5^\circ$

\* The primary stepper motor application is position control

- ① Disk drives use stepper motors to position R/W head
- ② some robots use them for positioning joints
- ③ Dot matrix printers use step motors to rotate the paper feed carriage and to position the print head horizontally relative to the paper

FIGURE 11-5 Four-Phase Stepper Motor

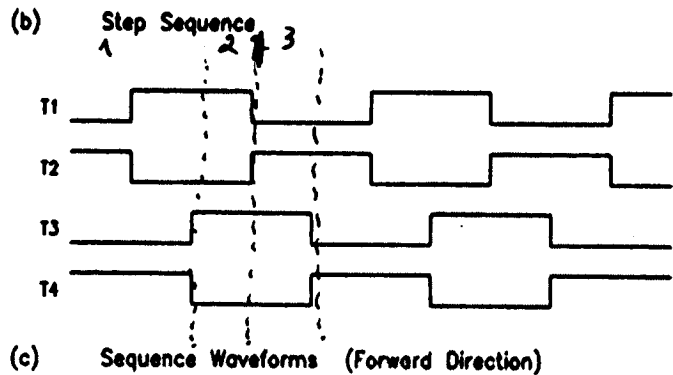
① Two sets of windings



(a) Schematic

Step No.	Step Logic Levels			
	T1	T2	T3	T4
1	1	0	0	1
2	1	0	1	0
3	0	1	1	0
4	0	1	0	1

0 = OFF  
1 = ON



(c) Sequence Waveforms (Forward Direction)

- (imp) - A high level of current is required to cause shaft rotation (rush current)
- During the period when no rotation occurs the motor winding must be supplied with a lower level of current (hold current)

- ② The direction of current through the windings determine which position the motor steps to next (rush currents)
- ③ Each change of polarity at the terminal of a winding is called a step of phase shift.

so a sequence of logic pulses steps the motor from one position to another

\* The phase sequence may begin anywhere (but) it must continue in order.

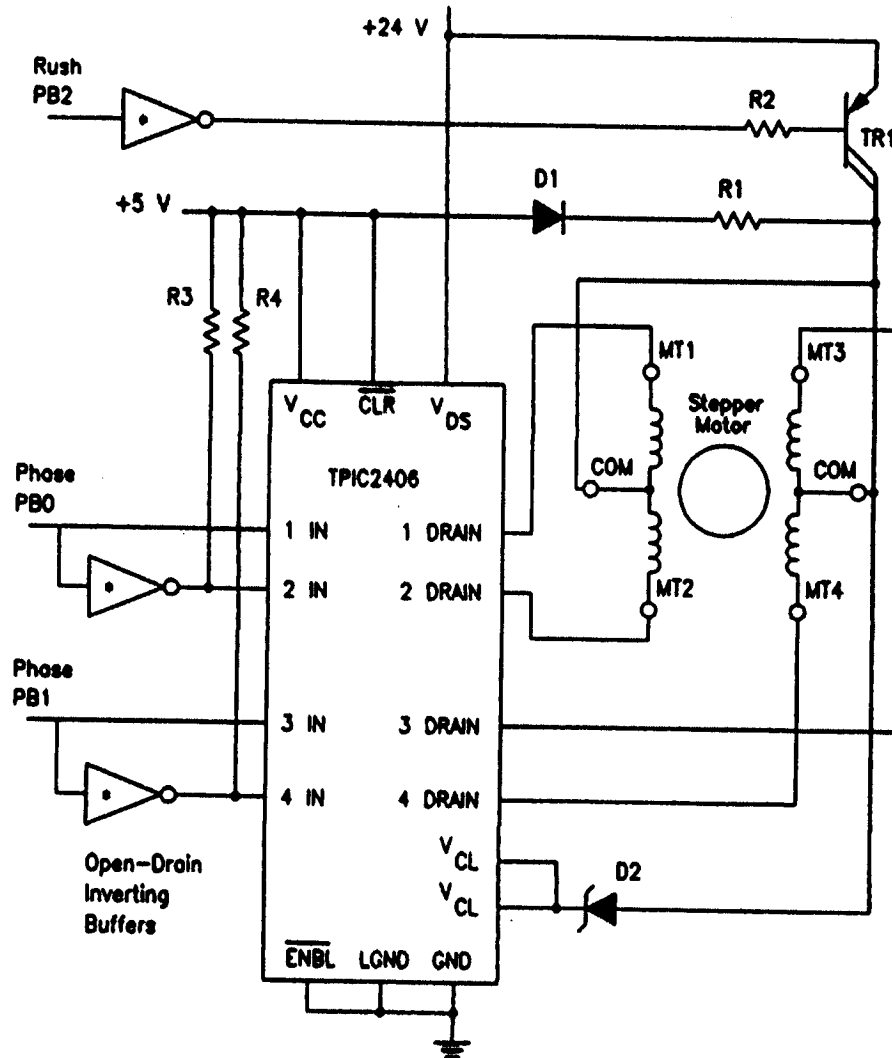
if step size is  $1.8^\circ$  then to drive it to  $10.8^\circ$  (we need six steps) originally in step 2  
3, 4, 1, 2, 3, 4

**FIGURE 11-6 Stepper Motor Drive Circuit**

\* Since high currents are required to drive stepper motor, a MOSFET latch with open drain outputs is used to supply pulses to the windings

\* TPIC2406 intelligent power quad MOSFET by Texas inst

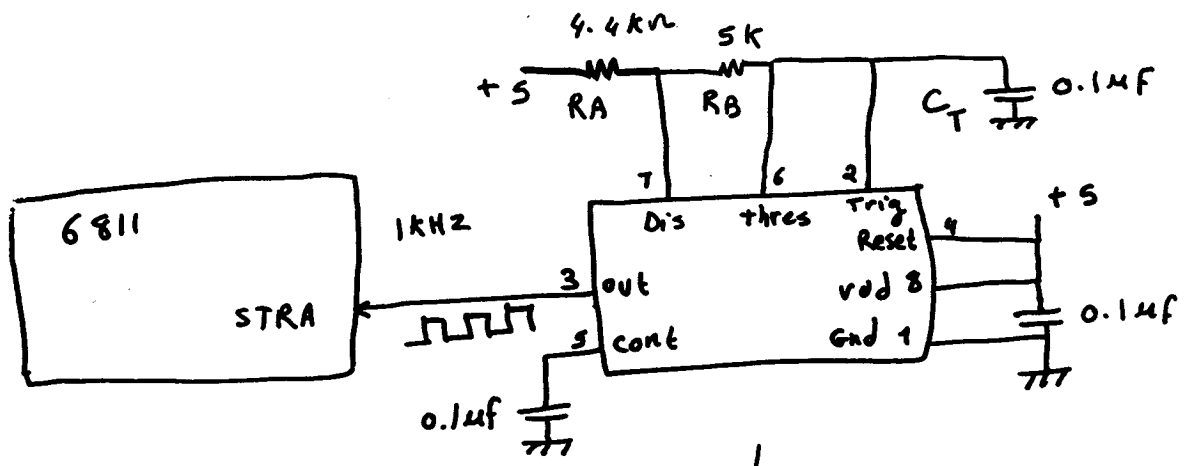
IT can drive high loads (3A) with built in protection



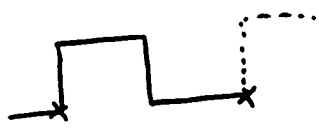
Check page 437 listing 11.5 & 11.6 for program to control stepper motor using OC

# Real Time interrupt using 6811 STRA

Although 6811 has an internal clock to create periodic interrupts  
HERE we use 555



⚡  
Astable multivibrator



period of 555 is  $0.693 \cdot C_T (R_A + 2R_B)$

## program

```

Time
Init
    rmb 2
    sei
    ldaa #$42
    staa $1002
    ldd #0
    std Time
    ldaa $1005
    cli
    * poll for zeros and ones
    IRQhan
    ldaa $1002
    cmpa #$C2
    beq CLKhan
    swi
    ldaa $1005
    ldx Time
    inx
    stx Time
    enable
    11000010 if interrupting
    Acknowledge
    
```

disable interrupts during Init  
PIOC STA1=1, HND5=0, EGA=1  
Arm interrupt on rise of STRA

initialize variable  
initially clear STRA

11000010 if interrupting

Acknowledge