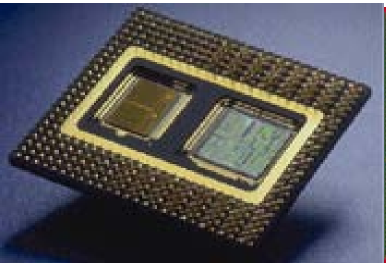
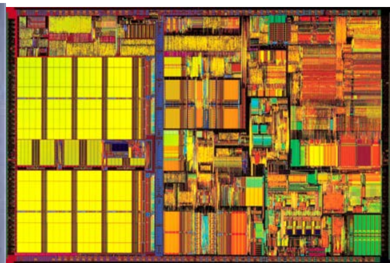
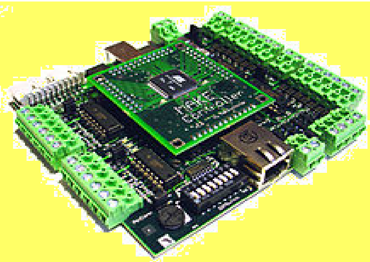
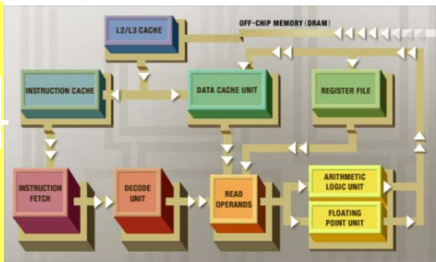
Lecture 12
Timer Functions

Bryan Ackland
Department of Electrical and Computer Engineering
Stevens Institute of Technology
Hoboken, NJ 07030

Adapted from HCS12/9S12 An Introduction to Software and Hardware Interfacing   Han-Way Huang, 2010

What are Timer Functions?

- A microprocessor operating in a real-time embedded application has to be able to:
  - generate (output) signals/waveforms with precise timing characteristics so as to accurately initiate and control events in external system
  - analyze temporal properties of (input) signals/events detected in external system so as to accurately determine state of external system and react accordingly

- Microprocessor may need to perform
  - time delay creation and measurement
  - period and pulse-width measurement
  - frequency measurement
  - event counting
  - arrival time comparison
  - time-of-day tracking
  - periodic interrupt generation
  - waveform generation

How Should Timer Functions be Implemented?

- Possible to implement most timer functions in software using interrupt driven real-time clock to measure and schedule events
  - very expensive in terms of available processing power
  - difficult to respond accurately to fast (short time period) events
  - difficult and tedious to program
- These operations can be handled more efficiently in hardware
  - most microcontrollers include some type of timer peripheral
- The HCS12 includes powerful timer module to support these time-based functions
  - we will study the detailed operation of HCS12 timer
  - general principals and functions applicable to broad array of microcontrollers

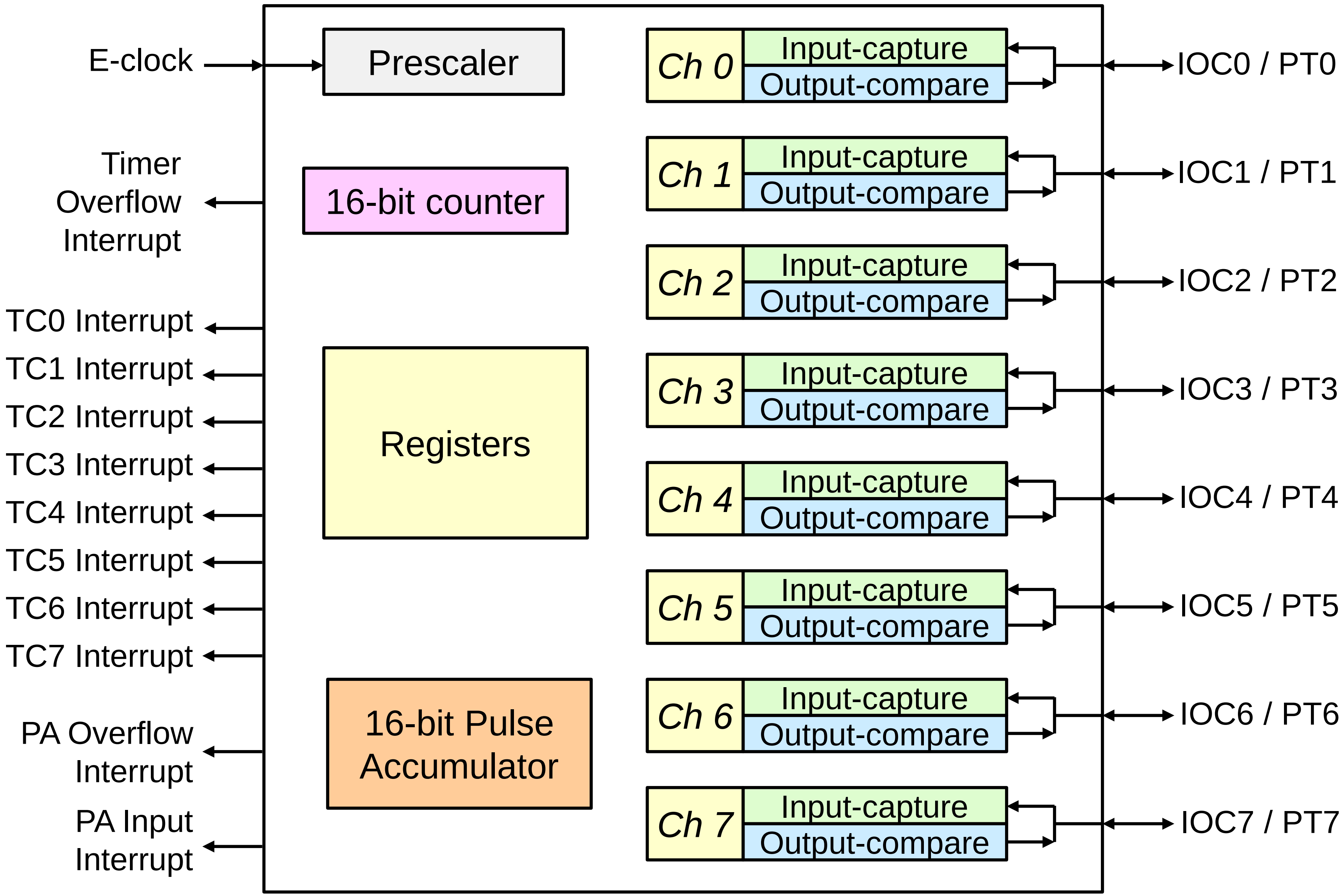
HCS12 Timer System

- The HCS12 has a standard timer module that is built around a 16-bit timer counter
  - counter is clocked by sub-multiple of bus clock (E-clock) and can be started and stopped at any time
- Provides 8 channels of input capture or output compare
- Input capture copies value of timer into a register when a specified input event (signal edge) occurs
  - can be used to measure pulse-width, period, duty cycle etc.
  - optionally generates an interrupt
- Output compare waits for the timer to be equal to a value in a register and optionally generates an output signal
  - can be used to generate time delay, trigger action at some future time, create a complex digital waveform etc.
  - optionally generates an interrupt

HCS12 Timer System (2)

- The HCS12 also provides:
- Pulse Accumulator – includes a second 16-bit counter to count input events arriving in a certain interval
  - can be used to simply count occurrences of some external event or measure frequency
- Pulse Width Modulation – can be used to generate simple waveforms without intervention of CPU
  - user sets up period and duty cycle
- Timer module shares I/O pins (IOC0~IOC7) with Port T (PT0~PT7)
  - Port T pins are not available as general purpose parallel port pins when they are being used by Timer module.

Timer Block Diagram



Timer Counter Register

- **Timer Counter Register (TCNT)** is the primary 16-bit counter
  - can be directly read/written by user
  - always use 16-bit (word) access to guarantee correct read/write
  - three other registers related to operation of TCNT:

- **Timer Interrupt Flag Register 2 (TFLG2)**

	7	*6*	*5*	*4*	*3*	*2*	*1*	*0*
	TOF	0	0	0	0	0	0	0
at reset:	0	0	0	0	0	0	0	0

TOF: timer overflow flag

- this flag is set whenever TCNT rolls over from $FFFF to $0000
- flag can be cleared by writing a ‘1’ to it

Timer Counter Registers

- **Timer System Control Register 1 (TSCR1)**

	7	*6*	*5*	*4*	*3*	*2*	*1*	*0*
	TEN	TSWAI	TSFRZ	TFFCA	0	0	0	0
at reset:	0	0	0	0	0	0	0	0

TEN: timer enable bit
‘0’ disables timer
‘1’ allows timer to count

TSWAI**:** timer stop in wait mode bit (used in power-down situations)*

TSFRZ**:** timer stop in freeze mode bit (used in debugging)*

TFFCA: timer fast flag clear all bits
‘0’ allows timer flag clearing to function normally
‘1’ causes flag to be cleared when corresponding data register is read

** we will not be using these bits*

Timer Counter Registers

- **Timer System Control Register 2 (TSCR2)**

	7	*6*	*5*	*4*	*3*	*2*	*1*	*0*
	TOI	0	0	0	TCRE	PR2	PR1	PR0
at reset:	0	0	0	0	0	0	0	0

TOI: timer overflow interrupt enable bit
‘0’ interrupt disabled
‘1’ interrupt when TOF flag is set
(i.e. when TCNT overflows)

TCRE: timer counter reset enable bit*
‘0’ counter free runs
‘1’ counter reset by successful output-compare 7

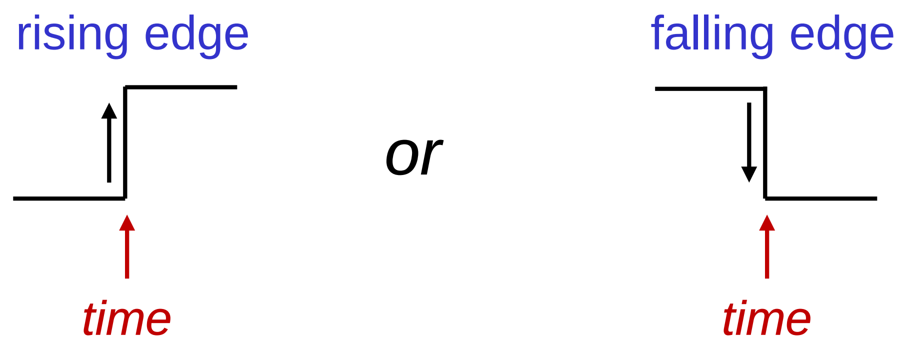
PR2~0: sets counter clock pre-scale
(E-clock is divided by this factor)

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

** we will not be using this bit*

Input Capture Function

- Input capture records the physical time of an external event
- Physical time is represented by contents of timer (TCNT)
- An event is represented by (rising or falling) edge on input pin



- When an event occurs:
  - value of timer is latched into a 16-bit register
  - flag is set (which may optionally generate an interrupt)
- HCS12 can employ up to eight input capture channels
  - each including a input pin, capture register and interrupt logic
- Input capture channels share circuitry with output compare function, so each channel can only be one or the other
  - TIOS register selects between these two functions

TIOS Register

- Port T has eights I/O (signal) pins that can be used:
  - to implement input-capture, OR
  - to implement output-compare OR
  - as a general purpose Port T parallel I/O pin*

- **Timer Input-Capture/Output-Compare Select (TIOS)**

	7	*6*	*5*	*4*	*3*	*2*	*1*	*0*
	IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0
at reset:	0	0	0	0	0	0	0	0

IOS[7:0]: input-capture/output-compare configuration bits
‘0’: the corresponding channel acts as an input-capture
‘1’: the corresponding channel acts as an output-compare

* To use a pin as a general purpose Port T pin, set the IOS bit to ‘0’ and see TCTL3 & 4

Registers Associated with Input Capture

- **Timer Control Registers 3 and 4 (TCTL3 and TCTL4)**

	7	6	5	4	3	2	1	0
TCTL3:	EDG7B	EDG7A	EDG6B	EDG6A	EDG5B	EDG5A	EDG4B	EDG4A
at reset:	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
TCTL4:	EDG3B	EDG3A	EDG2B	EDG2A	EDG1B	EDG1A	EDG0B	EDG0A
at reset:	0	0	0	0	0	0	0	0

EDG*n*B	EDG*n*A	Edge Configuration
0	0	capture disabled
0	1	capture on rising edge only
1	0	capture on falling edge only
1	1	capture on both edges

– When an input capture channel is selected, but capture is disabled, the associated pin can be used as general purpose I/O (Port T)

Registers Associated with Input Capture (2)

- **Timer Interrupt Flag Register 1 (TFLG1)**

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

C7F:C0F: input-capture-output-compare interrupt flag bits
‘0’ interrupt (selected edge) condition has not occurred
‘1’ interrupt (selected edge) condition has occurred

- **Timer Interrupt Enable Register (TIE)**

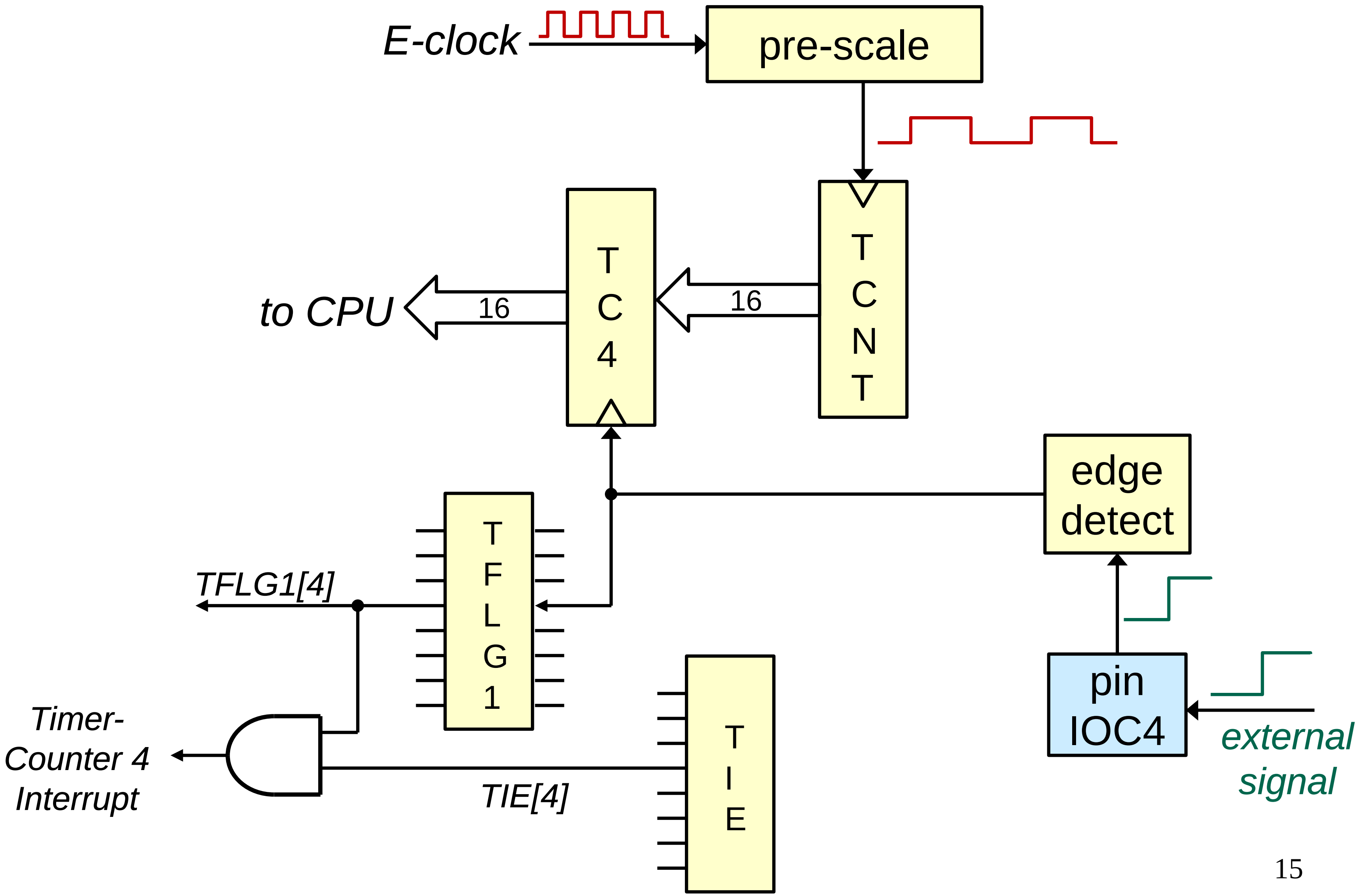
	7	*6*	*5*	*4*	*3*	*2*	*1*	*0*
	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I
at reset:	0	0	0	0	0	0	0	0

C7I:C0I: input-capture-output-compare interrupt enable bits
‘0’ interrupt disabled
‘1’ interrupt enabled
generates interrupt when corresponding bit of TFLG1 register is set

Registers Associated with Input Capture (3)

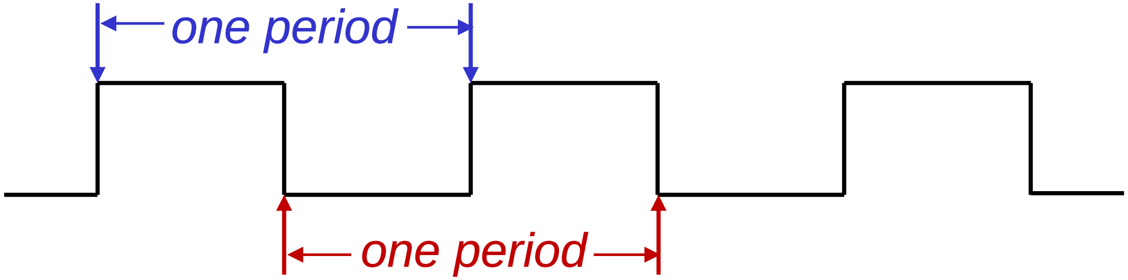
- **Timer Counter Data Registers (TC7~TC0)**
  - Each input-capture channel has a 16-bit register TC*n* which holds count value captured when the selected signal edge arrives at the pin
    - this register is also used by the output-compare function when this function has been selected
- **How to clear an input-capture interrupt flag:**
  - When selected edge event has been detected, interrupt flag in TFLG1 corresponding to that input channel is set
    - If corresponding interrupt enable bit in TIE register is set, this will generate an interrupt
  - When processing an event it is important to clear this interrupt flag to (a) get ready for the next event and (b) prevent further interrupts
    - Interrupt flag can be manually cleared by writing a ‘1’ to the interrupt flag bit in the TFLG1 register
    - Alternatively, if we set bit 4 (TFFCA) of the TSCR1 register, the interrupt flag will be automatically cleared whenever we read the value in the corresponding Timer Counter Data Register (TC*n*)
    - Note that flag cannot be manually cleared if TFFCA is set

Summary of Input Capture (Channel 4)

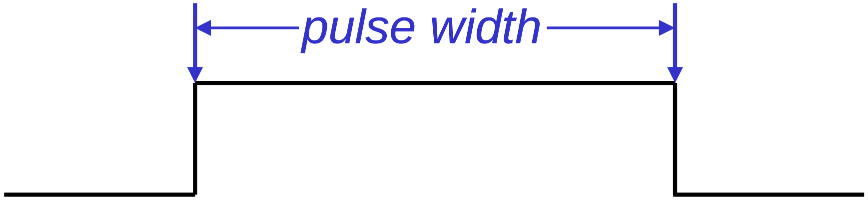


Applications of Input Capture

- Event arrival-time recording
  – e.g. logging personnel entry and exit in electronic key-card system, or recording arrival times of different swimmers in swimming competition
- Period Measurement
  – capture times of two successive rising or falling edges

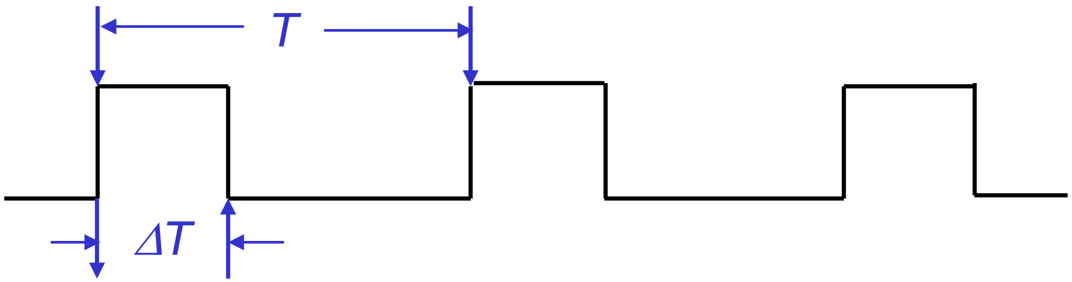


- Pulse-width Measurement
  – capture time of rising edge and next falling edge



Applications of Input Capture (2)

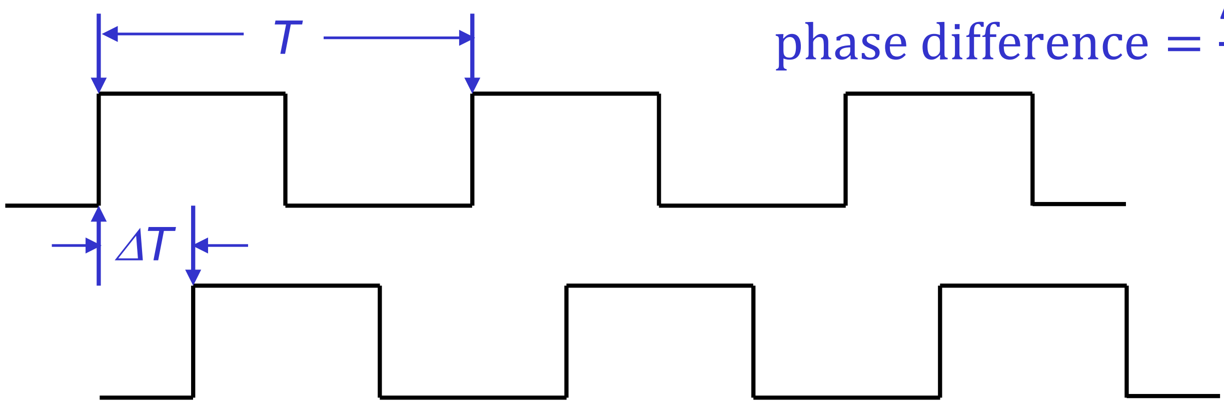
- Duty Cycle Measurement
  - percentage of time that a periodic signal is high within single period



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

- Phase Difference Measurement
  - difference in arrival times (as percentage of period) of two signals of the same frequency

$$\text{phase difference} = \frac{\Delta T}{T} \times 360°$$

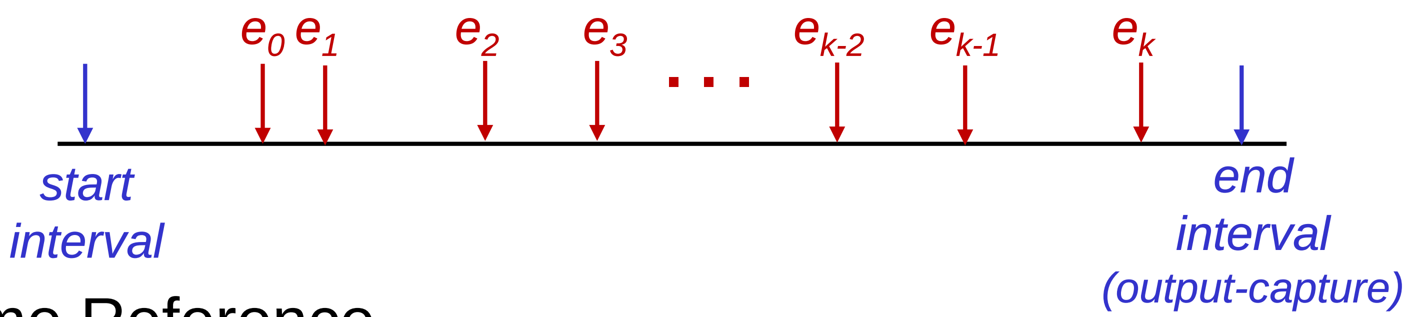


signal S1:

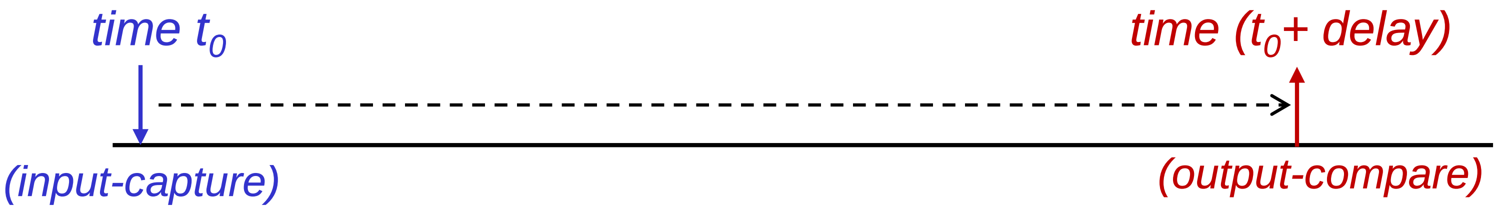
signal S2:

Applications of Input Capture (3)

- Interrupt Generation
  - Each input-capture function can be used as a distinct edge-sensitive interrupt source
- Event Counting
  - can be used in conjunction with output-compare function to count number of occurrences of certain event during specified time interval
  - counter incremented each time we get an input-capture interrupt

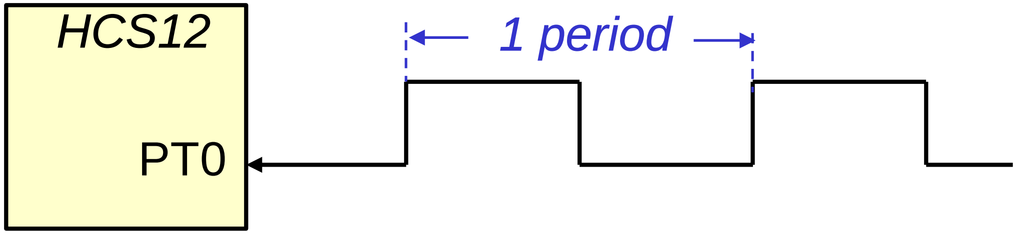


- Time Reference
  - activate an output specified period after detecting input event



Example: Period Measurement

- Use input-capture channel 0 to measure period of an unknown repetitive signal. Period is known to be shorter than 128ms. Assume the E-clock frequency is 24 MHz



Since input capture register is 16-bit, longest period we can measure with pre-scaler set to 1 is $2^{16}/24\ MHz = 2.73\ ms$

To measure a period up to 128 ms, we have two options:

(a) set pre-scale = 1 and count no. of times timer counter overflows

(b) set pre-scale = 64 and know that timer counter will not overflow

Option (a) gives greater accuracy, but is more difficult to program

We will use option (b)

Steps in Period Measurement

1. Enable timer-counter
   Set timer-counter pre-scale to 64
   Enable rising edge events on channel 0
   Clear C0F flag

2. Wait for C0F = 1

3. Save time of captured first edge
   Clear C0F flag

4. Wait for C0F = 1

5. Read time of captured second edge
   Take difference between second and first captured edges

Result will be number of clock cycles × clock period ($= 2.67\mu s$*)*

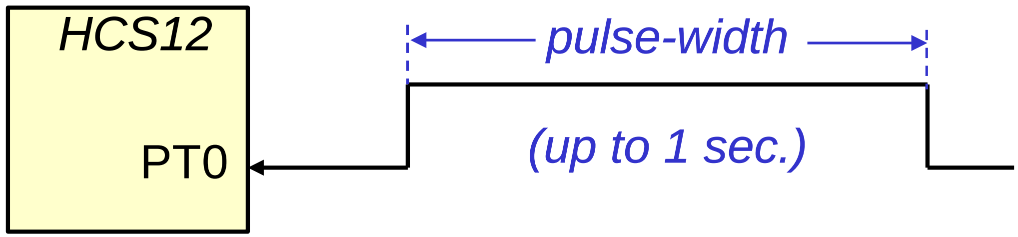
Code for Period Measurement

```
                include   hcs12.inc
                ORG       $800
edge1:          DS.W      1                 ;location to save first edge
period:         DS.W      1                 ;location to save period (in pre-scaled cycles)

                ORG       $4000
                movb      #$90, TSCR1       ;enable timer counter and fast flag clear option
                bclr      TIOS, $01         ;enable input-capture 0
                movb      #$06, TSCR2       ;disable TCNT overflow and set pre-scale=64
                movb      #$01, TCTL4       ;set to capture rising edge of PT0 signal
                ldd       TC0               ;clear the C0F flag
                brclr     TFLG1, $01, *     ;wait for arrival of first edge
                ldd       TC0               ;save first edge and clear C0F flag
                std       edge1
                brclr     TFLG1, $01, *     ;wait for second edge
                ldd       TC0
                subd      edge1             ;compute the period (in pre-scaled cycles)
                std       period            ;save result
                swi
```

Example: Pulse-Width Measurement

- Use input-capture channel 0 to measure (with $\pm 1\mu s$ resolution) pulse-width of a signal. Assume the E-clock frequency is 24 MHz



For 1μs resolution, set pre-scaler = 16 (resolution = $16/24\ MHz = 670\ ns$)

For long pulses (> 43 ms), the timer-counter may overflow many times

Record # times timer-counter overflows using interrupts and store overflow count in 16-bit memory location

Maximum pulse width will now be $(2^{32} \times 16)/24\ MHz = 2{,}863\ s$ (~ 48 mins)

To calculate pulse width (PW) given capture counts of first and second edges (edge1, edge2) and counter overflow count (ovcnt):

$$if\ (edge2 \geq edge1)\ then\ PW = (ovcnt \times 2^{16}) + (edge2 - edge1)$$

$$if\ (edge2 < edge1)\ then\ PW = ((ovcnt - 1) \times 2^{16}) + (edge2 - edge1)$$

Steps in Pulse-Width Measurement

1. Set up timer-counter overflow interrupt vector
   Clear overflow count
   Enable timer-counter
   Set timer-counter pre-scale to 16
   Enable rising edge events on channel 0
   Clear C0F flag

2. Wait for C0F = 1

3. Save time of captured first edge
   Clear C0F flag
   Enable counter-timer overflow interrupt
   Enable falling edge events

4. Wait for C0F = 1

5. Disable interrupts
   Read time of captured second edge
   Take difference between second and first captured edges
   If second edge count is smaller than first, decrement overflow count

TOF ISR

Clear TOF flag
increment overflow count
return from interrupt

Code for Pulse-Width Measurement

```
        include  hcs12.inc
        ORG      $800
edge1:  DS.W     1                  ;location to save first edge
ovflow: DS.W     1                  ;'ovflow' with 'PW' gives a 4-byte pulse-width
PW:     DS.W     1                  ;measurement (in E-clock cycles)

        ORG      $4000
        movw     #tof_isr, $3E5E    ;set up TCNT overflow interrupt vector
        lds      #$5000             ;set up stack pointer
        movw     #0, ovflow         ;clear overflow count
        movb     #$90, TSCR1        ;enable timer counter and fast flag clear option
        bclr     TIOS, $01          ;enable input-capture 0
        movb     #4, TSCR2          ;disable TCNT interrupt and set pre-scale=16
        movb     #$01, TCTL4        ;capture rising edge of PT0 signal
        ldd      TC0                ;clear the C0F flag
        brclr    TFLG1, $01, *      ;wait for arrival of first edge
        movw     TC0, edge1         ;save first edge and clear C0F flag
        movb     #$80, TFLG2        ;clear TOF flag
        bset     TSCR2, $80         ;enable TOF interrupt
        cli                         ;enable (global) maskable interrupts
```

Code for Pulse-Width Measurement (2)

```
         movb    #$02, TCTL4      ;capture falling edge of PT0 signal
         brclr   TFLG1, $01, *    ;wait for second edge
         sei                      ;turn off interrupts
         ldd     TC0
         subd    edge1            ;compute the period (in pre-scaled cycles)
         std     PW               ;save result
         bcc     done             ;is second edge smaller? (could use bhs)
         ldx     ovflow           ;yes – then decrement
         dex
         stx     ovflow
done:    swi

tov_isr: movb    #$80, TFLG2      ;clear TOF flag
         ldx     ovflow
         inx                      ;increment overflow count
         stx     ovflow
         rti
```

Output-Compare Function

- Output-compare used to trigger some action at a specific time in the future
- HCS12 supports up to eight output-compare channels, including:
  - 16-bit compare register TCx (same register as used in input-capture)
  - 16-bit comparator
  - output action pin PTx (can be pulled high, low, or toggled)
  - interrupt request option
- To set up an output-compare operation, the user:
  - activates output-compare channel & selects output pin function
  - makes a copy of current contents of TCNT register
  - adds to this a value equal to desired delay
  - stores the sum into output-compare register
- A successful compare will
  - set corresponding flag in TFLG1 register
  - optionally perform output pin operation
  - optionally generate interrupt

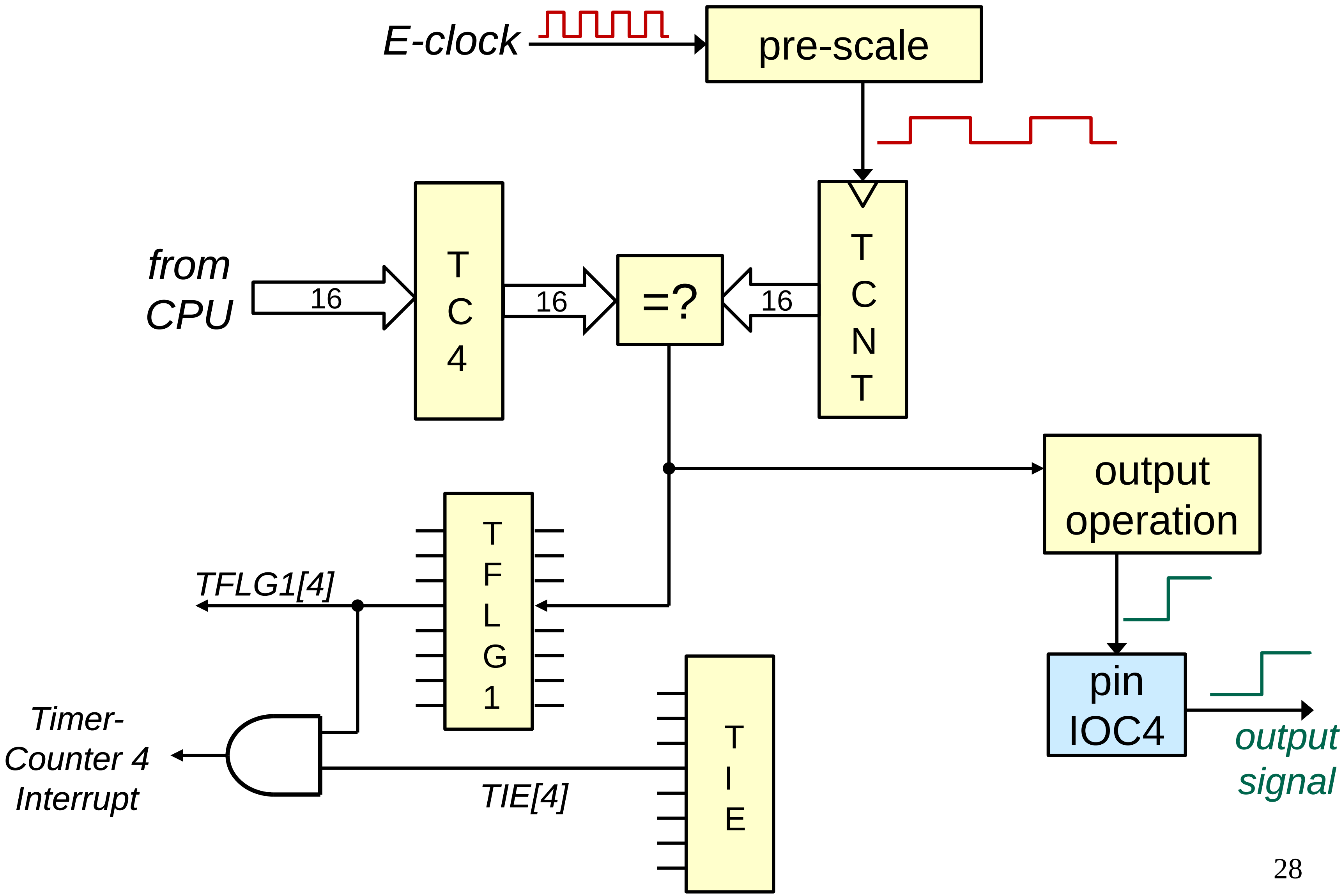
Output-Compare Registers

- In addition to registers already described under input-capture:
  - TCNT, TSCR1, TSCR2, TFLG1, TFLG2, TIOS and TIE
  - these registers perform essentially same function for output-compare
- **Timer Control Registers 1 and 2 (TCTL1 and TCTL2)**

	7	*6*	*5*	*4*	*3*	*2*	*1*	*0*
TCTL1:	OM7	OL7	OM6	OL6	OM5	OL5	OM4	OL4
at reset:	0	0	0	0	0	0	0	0
TCTL2:	OM3	OL3	OM2	OL2	OM1	OL1A	OM0	OL0
at reset:	0	0	0	0	0	0	0	0

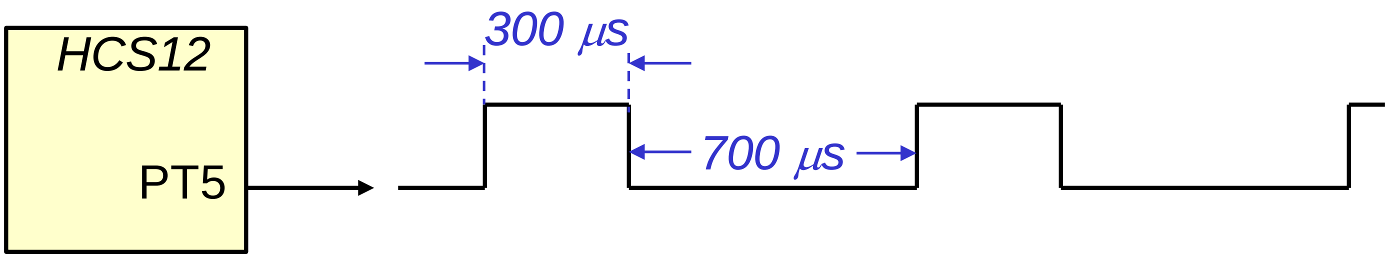
OM*n*	OL*n*	Output Level
0	0	no action
0	1	toggle OC*n* pin
1	0	clear OC*n* pin to 0
1	1	set OC*n* pin to 1

Summary of Output Capture (Channel 4)



Example: Waveform Generation

- Use output-compare channel 5 to generate an active-high 1.0 kHz waveform with a 30% duty cycle. Assume frequency of E-clock is 24 MHz.



Set pre-scaler = 8. Then TCNT period will be 1/3 μs

Number of clock cycles for high and low output will be 900 and 2100

Steps in Waveform Generation

Variables:
HiCnt is duration of high level (900 cycles)
LoCnt is duration of low level (2100 cycles)

1. Enable timer-counter
   Set timer-counter pre-scale to 8
   Set TIOS to enable OC5
2. Set OC5 pin action to “pull high”
   Start OC5 with count=LoCnt
3. Wait for C5F = 1
4. Change pin action to “pull low”
   Start OC5 with count=HiCnt
5. Wait for C5F=1
6. Go to step 2

Code for Waveform Generation

```
            include   hcs12.inc
HiCnt:      EQU       900
LoCnt:      EQU       2100

            ORG       $4000
            movb      #$90, TSCR1     ; enable TCNT with fast flag clear option
            movb      #$03, TSCR2     ; set prescaler to 8
            bset      TIOS, $20       ; enable OC5
low:        movb      #$0C, TCTL1     ; configure OC5 action to "pull high"
            ldd       TCNT            ; start OC5 with delay =LoCnt
            addd      #LoCnt
            std       TC5             ; this also clears C5F
            brclr     TFLG1, $20, *   ; wait until C5F=1 (which means PT5=1)
            movb      #$04, TCTL1     ; configure pin action to "pull low"
            ldd       TCNT            ; start OC5 with delay=HiCnt
            addd      #HiCnt
            std       TC5
            brclr     TFLG1, $20, *   ; wait for C5F=1 (which means PT5=0)
            bra       low             ; repeat
```

Example: Waveform Generation with Interrupts

- *Modify waveform generator to use interrupts, so that processor is free to do other useful work*

HiCnt is duration of high level (900 cycles)
LoCnt is duration of low level (2100 cycles)
HiLo indicates current output (0 or 1)

1. Set up OC5 interrupt vector
   Enable timer-counter
   Set timer-counter pre-scale to 8

2. Set OC5 pin action to “pull high”
   Set HiLo to 0
   Start OC5 with count=LoCnt
   Enable OC5 interrupt

3. Go do “other stuff”

OC5 ISR

1. if HiLo=1 go to step 2
   set pin action to “pull low”
   restart OC5 with count=HiCnt
   set HiLo=1
   return from interrupt

2. set pin action to “pull high”
   restart OC5 with count=LoCnt
   set HiLo=0
   return form interrupt

Code for Waveform Generation with Interrupts

```
        include hcs12.inc
HiCnt:  EQU     900
LoCnt:  EQU     2100
        ORG     $800
HiLo:   ds.b    1               ; flag to indicate current output level (0 or 1)

        ORG     $4000
        movw    #OC5isr, $3E64  ;set up OC5 interrupt vector
        lds     #$5000          ;set up stack pointer
        movb    #$90, TSCR1     ;enable TCNT with fast flag clear option
        movb    #$03, TSCR2     ;set prescaler to 8
        bset    TIOS, $20       ;enable OC5
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        movb    #$0C, TCTL1     ;configure OC5 action to “pull high”
        ldd     TCNT
        addd    #Lo_Cnt         ;clear C5F flag and start with delay=LoCnt
        std     TC5
        clr     HiLo            ;set current output flag = 0
```

Code for Waveform Generation with interrupts (2)

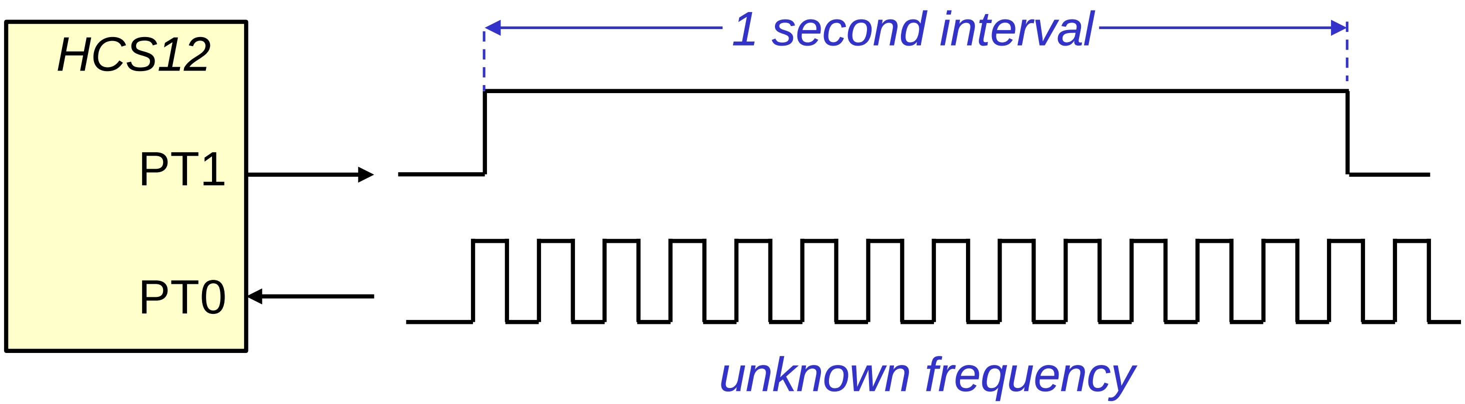
```
movb    #$20, TIE        ;enable OC5 interrupt
cli                      ;enable (global) interrupts
bra     other_stuff      ;while OC5 generates output waveform
```

Code for Waveform Generation with interrupts (3)

```
OC5isr:    tst    HiLo          ; what is current output level?
           bne    low_next      ; if one, then low level next
           movb   #$08, TCTL1   ; set output to "pull low"
           ldd    TCNT
           addd   #HiCnt        ;clear C5F flag and restart with delay=HiCnt
           std    TC5
           movb   #1, HiLo      ;set current output level=1
           rti
------------------------------------------------------------------------
low_next:  movb   #$0C, TCTL1   ; set output to "pull high"
           ldd    TCNT
           addd   #LoCnt        ;clear C5F flag and restart with delay=LoCnt
           std    TC5
           clr    HiLo          ;set current output level=0
           rti
```

Example: Measure Frequency

- **Combine the use of input-capture and output-compare functions to measure frequency.**
- Set up OC1 to define a one second measuring period. Use IC0 to count number of rising edges on TC0 during a one second interval. Assume E-clock is 8 MHz



Note that PT1 signal is not needed externally. OC1 is simply used to generate an "internal" time period for counting edges on PT0

Set pre-scaler = 8. Then TCNT period will be 1 μs

One second period can be measured as 100 times 10ms

Use interrupts to count PT0 edges

Code Frequency Measurement

```
         include  hcs12.inc
         ORG      $800
oc_cnt:  ds.b     1                ;variable to count 10ms periods
freq:    ds.w     1                ;variable to count edges on PT0

         ORG      $4000
         movw     #tc0isr, $3E6E   ;set up IC0 interrupt vector
         lds      #$5000           ;set up stack pointer
         movb     #$90, TSCR1      ;enable TCNT with fast flag clear option
         movb     #$03, TSCR2      ;set pre-scaler to 8
         movb     #$02, TIOS       ;enable OC1 and IC0
         movb     #100, oc_cnt     ;initialize 10ms period counter to 100 periods
         movw     #0, freq         ;initialize edge count to 0
         movb     #$01, TCTL4      ;configure IC0 to capture rising edges
         ldd      TC0              ;clear C0F flag
         bset     TIE, $01         ;enable interrupts on IC0
         cli                       ;enable (global) interrupts
```

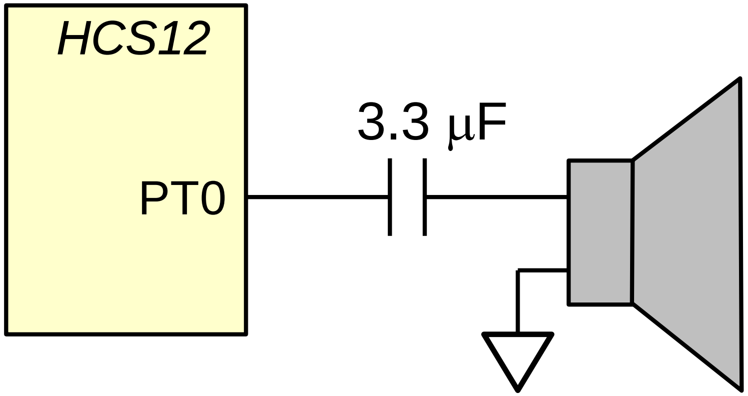
Code Frequency Measurement

```
continue:   ldd     TCNT
            addd    #10000          ;set OC delay to 10,000 cycles
            std     TC1
            brclr   TFLG1, $02, *   ; wait for 10 ms
            dec     oc_cnt          ;are we done yet?
            bne     continue;
            bclr    TIE, $01        ;disable IC0 interrupt to stop counting
            swi

tc0isr:     ldd     TC0             ;clear C0F flag
            ldx     freq
            inx                     ;increment edge-count
            stx     freq
            rti
```

Example: Siren Oscillator

- A small speaker is attached to PT0. Write a program to generate a siren that oscillates between 300 Hz and 1200 Hz at 0.5 second intervals. Assume E-clock = 24 MHz



Set pre-scaler to 8. Each count is then 1/3 μs

Use OC0 in interrupt mode to generate continuous square wave at specified frequency (300 or 1200 Hz)

Use OC4 in polling mode to switch frequencies every 500 ms

Code Siren Oscillator

```
         include  hcs12.inc
hi_freq: EQU      1250              ;1200 Hz half-period in units of 1/3 us
lo_freq: EQU      5000              ;300 Hz half-period in units of 1/3 us
         ORG      $800
delay:   ds.w     1                 ;delay variable to determine frequency

         ORG      $4000
         movw     #tc0isr, $3E6E    ;set up OC0 interrupt vector
         lds      #$5000            ;set up stack pointer
         movb     #$90, TSCR1       ;enable TCNT with fast flag clear option
         movb     #$03, TSCR2       ;set pre-scaler to 8
         bset     TIOS, $03         ;enable OC1 and OC0
         movb     #$01, TCTL2       ;configure OC0 to toggle output
         movw     #hi_freq, delay   ;start with high tone
         ldd      TCNT              ;set up half-period delay
         addd     delay
         std      TC0
         bset     TIE, $01          ;enable interrupts on OC0
         cli                        ;enable (global) interrupts
```

Code Siren Oscillator (2)

```
forever:  ldy    #50               ; count 50 x 10ms periods
hiloop:   ldd    TCNT
          addd   #30000            ; set up OC1 for 10ms delay
          std    TC1
          brclr  TFLG1, $02, *     ; wait until C1F is set
          dbeq   y, hiloop         ; repeat 50 times
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
          movw   #lo_freq, delay   ; change to low tone
          ldy    #50               ; count 50 x 10ms periods
loloop:   ldd    TCNT
          addd   #30000            ; set up OC1 for 10ms delay
          std    TC1
          brclr  TFLG1, $02, *     ; wait until C1F is set
          dbeq   y, loloop         ; repeat 50 times
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
          movw   #hi_freq, delay   ; change to high tone
          bra    forever

tc0isr:   ldd    TC0               ; re-arm OC0
          addd   delay
          std    TC0
          rti
```