uOttawa

L'Université canadienne
Canada's university

CEG3136
Computer Architecture II
Timer Module

Notes for
Dr. Voicu Groza

Université d'Ottawa | University of Ottawa

www.uOttawa.ca

Topics of discussion

- Introduction to the HCS Standard Timer Module
- Output-compare functions
- Input-capture functions
- Pulse-width modulation
- Reading: FreeScale Documentation: ECT_16B8C Block User Guide V01.06, Chapter 14 (Sections 14.1 - 14.5 , 14.10)

Important Timing Functions

- Create and measure time delays
- Generating waveforms
- Generating periodic interrupts
- Measuring pulse widths
- Measuring frequencies
- Counting events
- Comparing arrival times

Timer module 16-bit counter

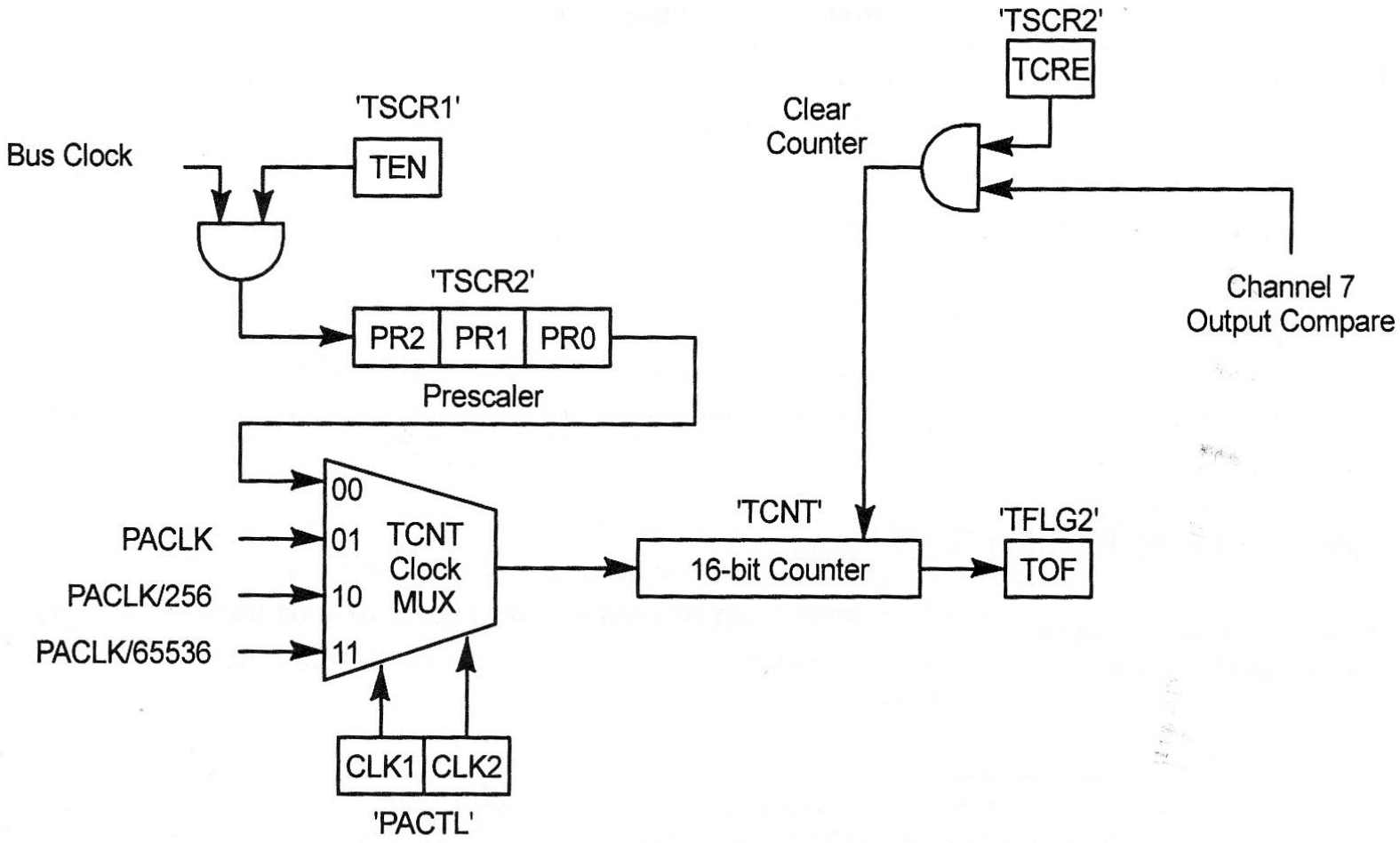
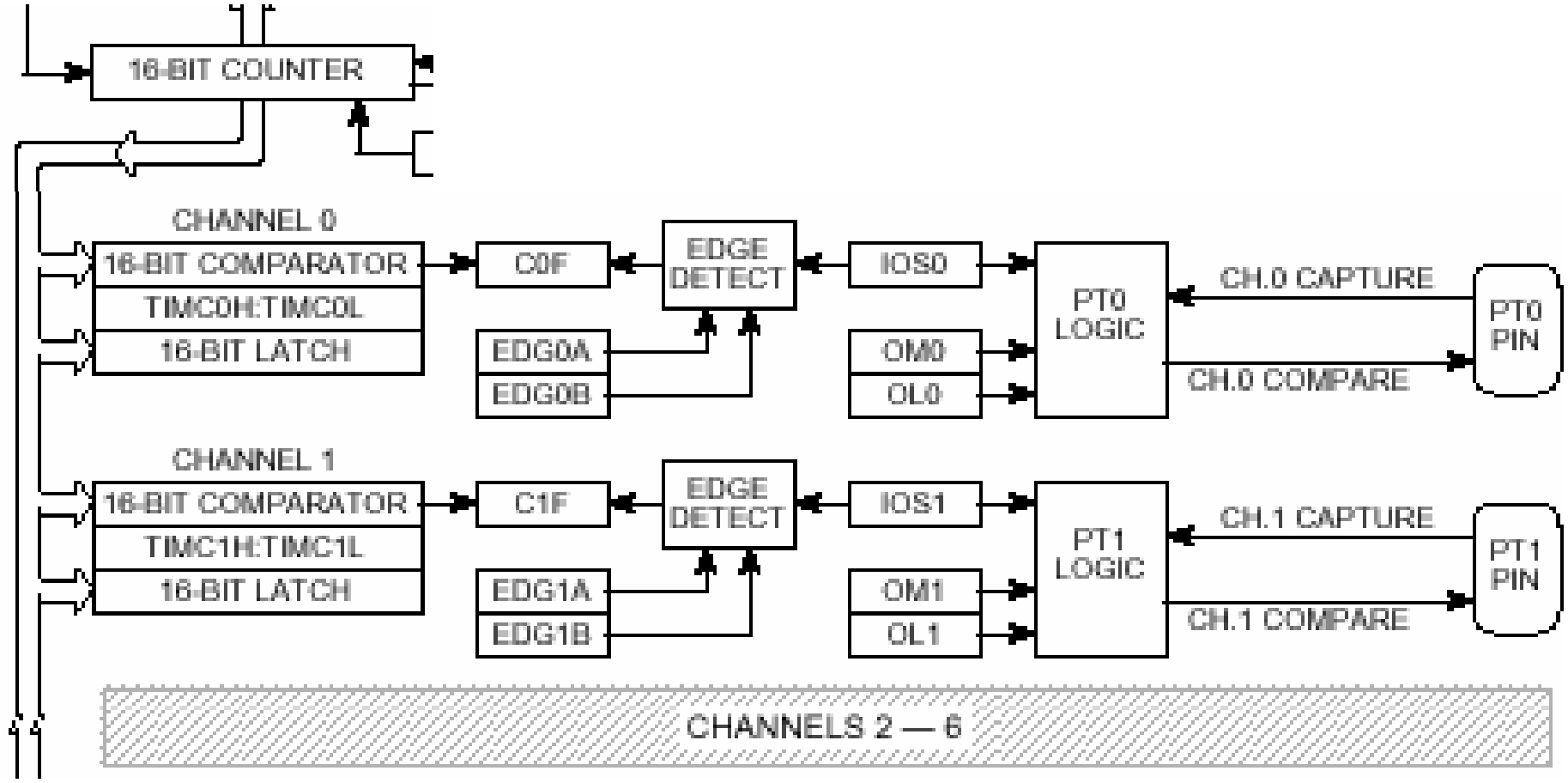


Figure 14-1 TCNT hardware.

Timer counter

- Counts from $0000 to $FFFF
  - And restarts.
- Bus-Clock
  - 125 ns per cycle (8 MHz system clock)
  - 41 2/3 ns per cycle (24 MHz system clock)
  - Can be divided using scale factor 1, 2, 4, 8, 16 32, 64, 128
  - Activated by flag (TEN)
- Counter overflow
  - When counter goes from $FFFF to $0000
  - Sets a flag (TOF)
  - Can also cause interruption (enabled by TOE)

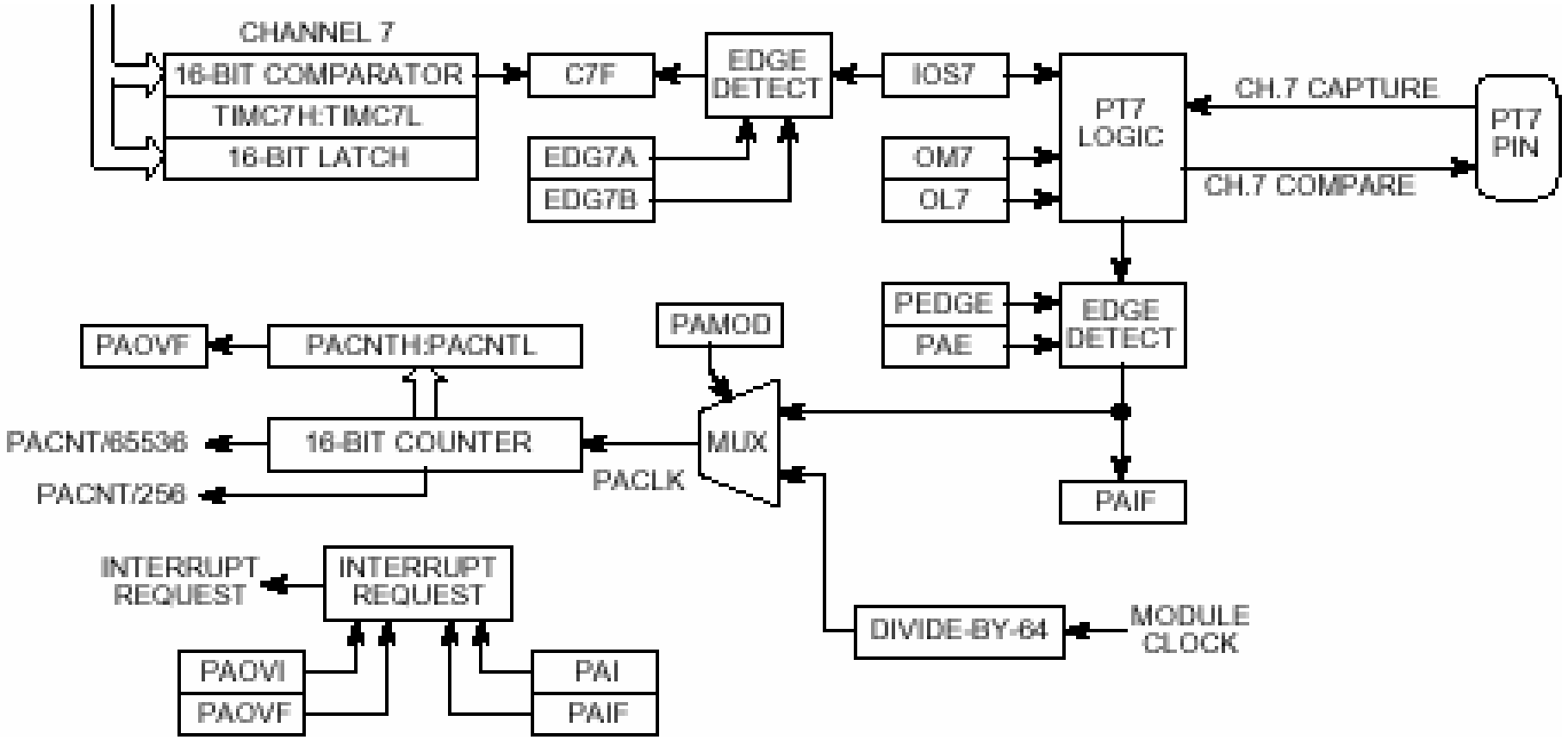
Timer Module Channels 0 to 6



Timer Channel

- Output compare – Channel n (0 to 6)
  - When a channel data register TCn equals TCNT, flag is set (CnF) (uses the comparator)
  - Can also cause an interrupt
  - Can also change level on pin PTn
    - Controls bits OMn and OLn determine the action on the pin
  - Can create a time delay or generate a waveform
- Input Capture – Channel n (0 to 6)
  - When an event is signalled on the pin PTn
    - The value of TCNT is captured in data registers TCn (with latch)
    - Flag CnF is set (an interrupt can also be generated)
  - Control bits EDGnA and EDGnB defines how event is signalled on the pin
  - Can be used to measure signal frequency, period, pulse width

Timer Channel 7



Timer Channel 7

- Provides same functions as found in channels 0 to 6
- In addition - can control outputs of all Port T pins
- In addition – can be used to initialise the 16-bit counter
- In addition – acts as input pin for the pulse accumulator

Timer Module Registers

$0040	TIOS	Timer Input Capture/Output Compare Select Register
$0041	CFORC	Timer Compare Force Register
$0042	OCM7M	Output Compare 7 Mask Register
$0043	OC7D	Output Compare 7 Data Register
$0044/ $0045	TCNT	Timer Count Register
$0046	TSCR1	Timer System Control Register 1
$0047	TTOV	Timer Toggle On Overflow Register 1

Timer Module Registers

$0048	TCTL1	Timer Control Register 1
$0049	TCTL2	Timer Control Register 2
$004A	TCTL3	Timer Control Register 3
$004B	TCTL4	Timer Control Register 4
$004C	TIE	Timer Interrupt Enable Register
$004D	TSCR2	Timer System Control Register 2
$004E	TFLG1	Main Timer Interrupt Flag 1 Register
$004F	TFLG2	Main Timer Interrupt Flag 2 Register

Timer Module Registers

$0050/$0051	TC0	Timer Input Capture/Output Compare Register 0
$0052/$0053	TC1	Timer Input Capture/Output Compare Register 1
$0054/$0055	TC2	Timer Input Capture/Output Compare Register 2
$0056/$0057	TC3	Timer Input Capture/Output Compare Register 3
$0058/$0059	TC4	Timer Input Capture/Output Compare Register 4
$005A/$005B	TC5	Timer Input Capture/Output Compare Register 5
$005C/$005D	TC6	Timer Input Capture/Output Compare Register 6
$005E/$005F	TC7	Timer Input Capture/Output Compare Register 7

Starting the Timer Module

- TCNT - Timer Counter Register
  - Main timer register
  - Must be read with a single instruction (16 bits)
  - Three registers related to TCNT: TSCR1, TSCR2, and TFLG2
- TSCR1 - Timer System Control Register 1
  - TSCR1 bit 7 (TEN) activates and deactivates the timer
  - TSCR1 bit 4 allows flags to be erased automatically

	BIT7	6	5	4	3	2	1	BIT0
R	TEN	TSWAI	TSFRZ	TFFCA	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

TSCR1

Figure 3-6 Timer System Control Register 1 (TSCR1)

TEN — Timer Enable

0 = Disables the main timer, including the counter. Can be used for reducing power consumption.

1 = Allows the timer to function normally.

TFFCA — Timer Fast Flag Clear All

0 = Allows the timer flag clearing to function normally.

1 = For TFLG1($0E), a read from an input capture or a write to the output compare channel ($10–$1F) causes the corresponding channel flag, CnF, to be cleared. For TFLG2 ($0F), any access to the TCNT register ($04, $05) clears the TOF flag. Any access to the PACN3 and PACN2 registers ($22, $23) clears the PAOVF and PAIF flags in the PAFLG register ($21). Any access to the PACN1 and PACN0 registers ($24, $25) clears the PBOVF flag in the PBFLG register ($31). This has the advantage of eliminating software overhead in a separate clear sequence. Extra care is required to avoid accidental flag clearing due to unintended accesses.

Starting the Timer Module

- TSCR2 - Timer System Control Register 2
  - Another control register
  - Bit 7 (TOI) enables interrupt at TCNT overflow (from $FFFF to $0000)
  - Bit 3 (TCRE) Allows channel 7 to reinitialise the counter.
  - Bits 2 to 0 (PR2 to PR0) define the pre-scale value for the clock
- TFLG2 – (Main Timer Interrupt Flag 2 Register)
  - Only bit 7 is used (TOF) – to indicate a TCNT overflow (from $FFFF to $0000)

	BIT7	6	5	4	3	2	1	BIT0
R	TOI	0	0	0	TCRE	PR2	PR1	PR0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-11 Timer System Control Register 2 (TSCR2)

TOI — Timer Overflow Interrupt Enable

0 = Interrupt inhibited

1 = Hardware interrupt requested when TOF flag set

TSCR2

TCRE — Timer Counter Reset Enable

This bit allows the timer counter to be reset by a successful output compare 7 event. This mode of operation is similar to an up-counting modulus counter.

0 = Counter reset inhibited and counter free runs

1 = Counter reset by a successful output compare 7

If TC7 = $0000 and TCRE = 1, TCNT will stay at $0000 continuously. If TC7 = $FFFF and TCRE = 1, TOF will never be set when TCNT is reset from $FFFF to $0000.

Table 3-4 Prescaler Selection

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

TSCR2

	BIT7	6	5	4	3	2	1	BIT0
R	TOF	0	0	0	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

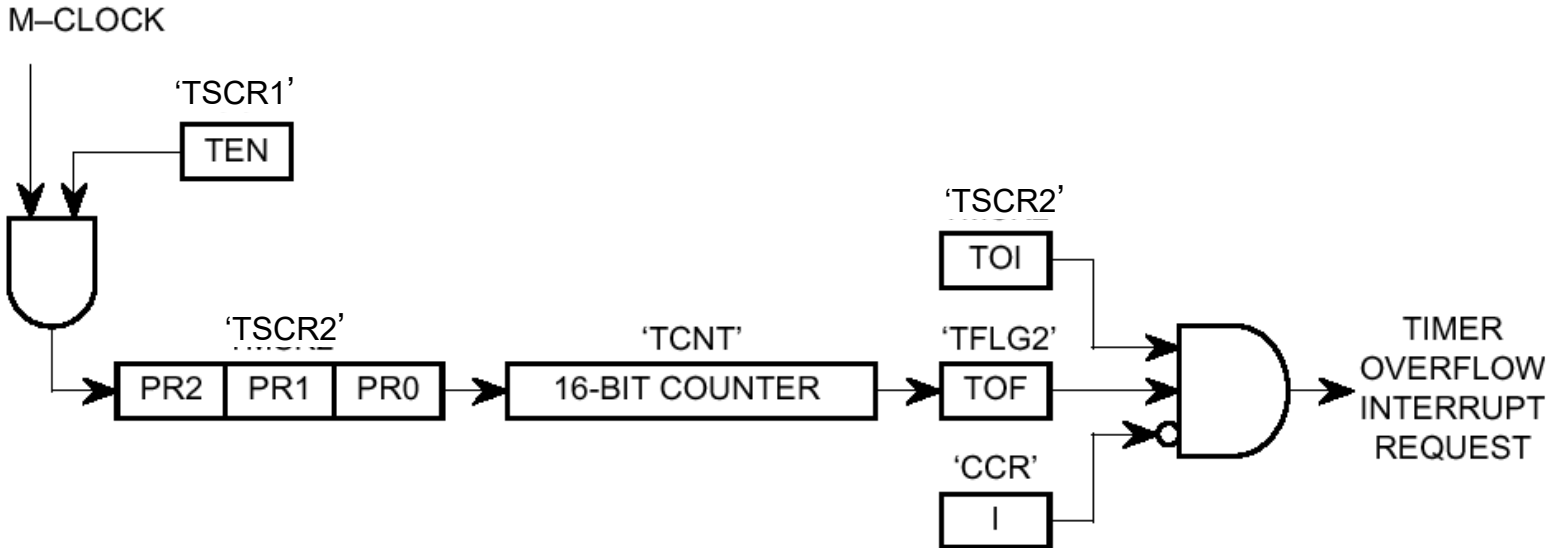
= Unimplemented or Reserved

TFLG2

Figure 3-13 Main Timer Interrupt Flag 2 (TFLG2)

Counter overflow

- Indicated by flag T0F.
- How is timer configured to generate an interruption when a counter overflow occurs?
- How is timer configured to create an 10.923 millisecond delay between interrupts?



Topics of discussion

- Introduction to the HCS Standard Timer Module
- Output-compare functions
- Input-capture functions
- Pulse-width modulation
- Reading: FreeScale Documentation: ECT_16B8C Block User Guide V01.06, Chapter 14 (Sections 14.1 - 14.5 , 14.10)

Output Compare

- The HCS12 can provide up to 8 output-compare channels with
  - A 16-bit compare comparator
  - A 16-bit comparison register (data register)
  - An action on an output pin (PTn)
  - An interrupt
  - A forced output-compare (CFOCn)

Output Compare

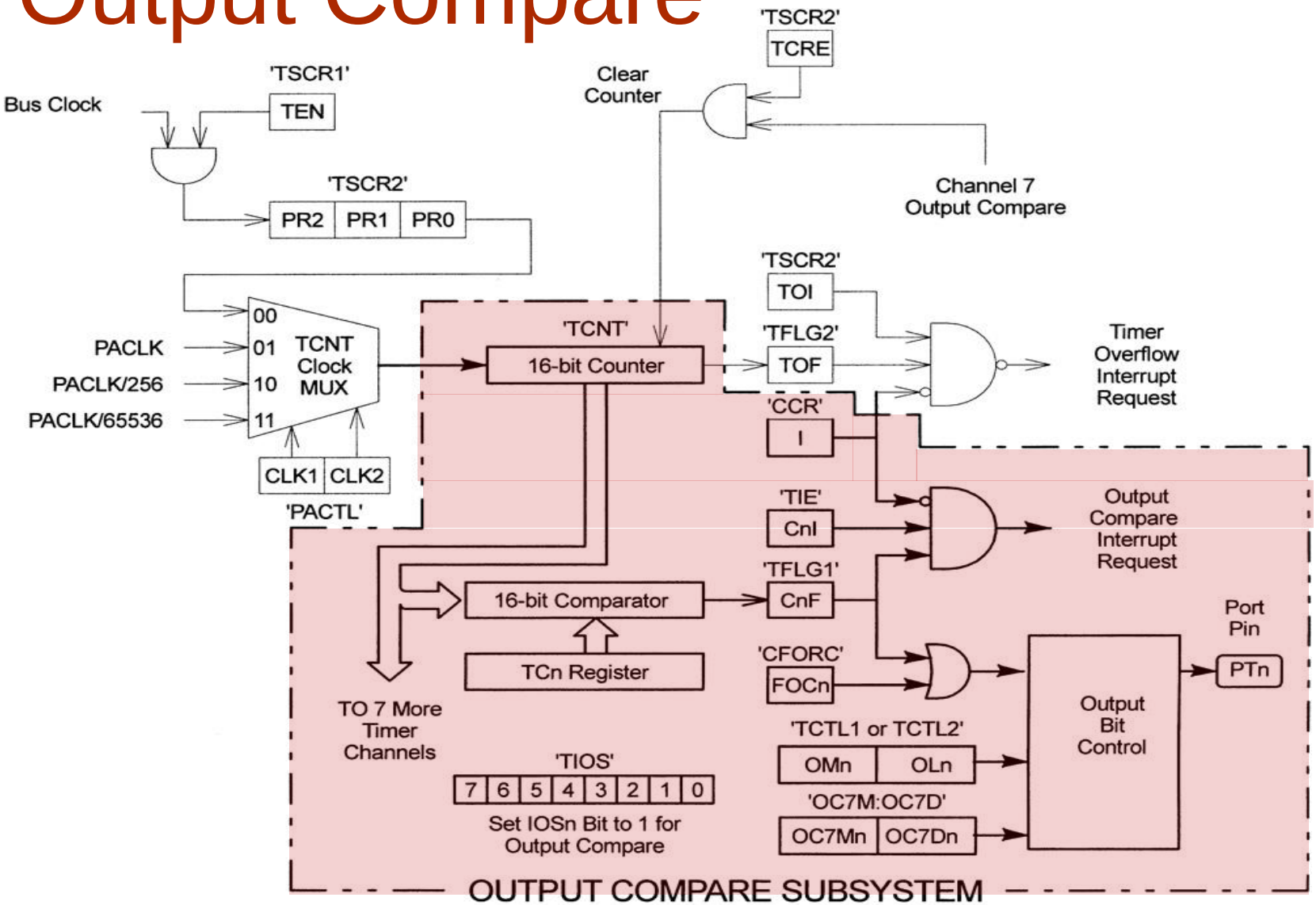


Figure 14-3 Timer output compare hardware.

TIOS Register

- Defines the function provided by each channel

	BIT7	6	5	4	3	2	1	BIT0
R W	IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0
RESET:	0	0	0	0	0	0	0	0

Figure 3-1 Timer Input Capture/Output Compare Register (TIOS)

Read or write anytime.

IOS[7:0] — Input Capture or Output Compare Channel Configuration
0 = The corresponding channel acts as an input capture
1 = The corresponding channel acts as an output compare.

Operation of the Output Compare Function

- Compares TCNT to the contents of all data registers in channels configured as output-compare
- When TCNT is equal to one of the data registers
  - Set channel flag CnF (in register TFLG1)
  - Can also raise an interrupt (enabled by bit in the TIE register)
- Actions done on channel pin can be:
  - Set pin PTn to 1 (+Vcc)
  - Clear pin PTn (GND)
  - Toggle voltage on pin (from 0v to 5v and vice versa)
  - Action is configured by TCTL1 and TCTL2 registers

TFLG1 and TIE Registers

- When the TCNT is equal to the content of a channel comparison register, the corresponding flag in TFLG1 is set
  - Writing a 1 to the bit clears the bit (and removes any interrupt signal)
- An interruption occurs if the channel enable bit is set to 1 in TIE.

	BIT7	6	5	4	3	2	1	BIT0
R W	C7F	C6F	C5F	C4F	C3F	C2F	C1F	C0F
RESET:	0	0	0	0	0	0	0	0

Figure 3-12 Main Timer Interrupt Flag 1 (TFLG1)

	BIT7	6	5	4	3	2	1	BIT0
R W	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I
RESET:	0	0	0	0	0	0	0	0

Figure 3-10 Timer Interrupt Enable Register (TIE)

	BIT7	6	5	4	3	2	1	BIT0
R W	OM7	OL7	OM6	OL6	OM5	OL5	OM4	OL4
RESET	0	0	0	0	0	0	0	0

	BIT7	6	5	4	3	2	1	BIT0
R W	OM3	OL3	OM2	OL2	OM1	OL1	OM0	OL0
RESET	0	0	0	0	0	0	0	0

Figure 3-8 Timer Control Register 1/Timer Control Register 2 (TCTL1/TCTL2)

Table 3-2 Compare Result Output Action

OMn	OLn	Action
0	0	Timer disconnected from output pin logic
0	1	Toggle OCn output line
1	0	Clear OCn output line to zero
1	1	Set OCn output line to one

Example 14-10

- Write a program to create a one second delay with the timer module (channel 2. Assume a 24MHz Clock)
  - Make LED blink with 1 sec interval.
- Solution:
  - Many possible solutions exists.
  - Shall use interrupts
  - Set the pre-scale factor to 16 for the TCNT – this provides a 2/3 μs increment time for TCNT.
  - Count 6000 ticks gives a delay of 4ms, i.e. generate an interrupt every 4 ms.
  - Use a counter to count 250 interrupts to get 1 second delay.

Example 14-10 - Assembler

```
LED1   EQU    0x01   ; Pin 1 is attached to LED
NTIMES  EQU 250     ; number of inter's for 1 sec delay
MS_4   EQU    6000   ; Counter ticks for 4 ms delay
       bset DDRB, LED1    ;  Set Port B Bit 1 to output
       bclr PORTB, LED1   ; Turn on LED
;---- The Timer
       movb %10010000,TSCR1 ; Turn on and fast clear
       bset TIOS, IOS2   ; Use channel 2 for Output/Comp.
       movb %00000100,TSCR2 ; 2/3 micro-s ticks
       movb #NTIMES, counter   ; setup  the counter
       ldd TCNT
       addd #MS_4   ; first interrupt occurs in 4 ms
       std TC2
       bset TIE, C2I   ; Enable TC2 interrupts
       cli             ; allow global interrupt
;---- Note setup of the interrupt vector is not shown
```

Example 14-10 – Assembler (cont'd)

```
spin: wai;          ; waits for first interrupt
      tst counter ; is counter at zero
      bne spin
      movb #NTIMES, counter ; re-initialise counter
      ldaa PORTB
      eora PORTB   ; toggles the PORT Bits
      staa PORTB
      bra spin

;---------------ISR
oc2_isr:    dec counter    ; decrements the counter
            ldd  TC2
            addd #MS_4
            std TC2       ; sets up next interrupt
            rti

counter     DS.B  1       ; 1 byte counter
```

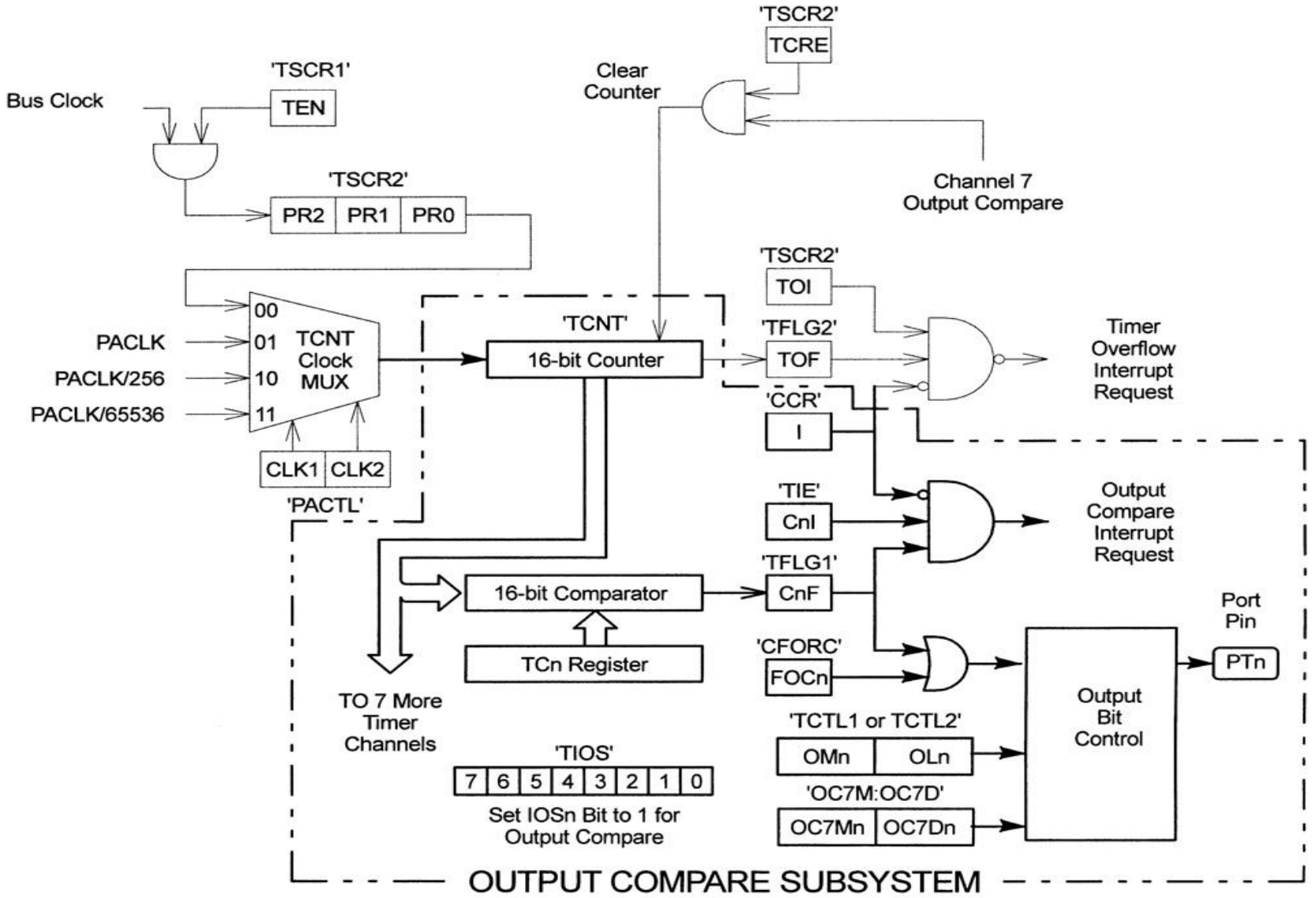


Figure 14-3 Timer output compare hardware.

Example 14-10 – In C

```
#include <mc9s12c256.h>
#define LED1 0x1
#define NTIMES 250 /* number of int's for 1 sec*/
#define MS_4 6000 /* # of ticks for 4ms delay */
void interrupt VectorNumber_Vtimch2 oc2_isr(void); //Proto
int counter; // ISR counter
/*-----------*/
void main(void) {
        DDRB = DDRB | LED1; // Bit 1, Port B, output
   PORTB = PORTB | LED1; // Turn on LED
   TSCR1 = 0b10010000;  // Enable, Fast clear
   TSCR2 = 0b00000100;  // 2/3 micro-s ticks
   TIOS = TIOS | 0b00000100; // OC for channel 2
   counter = NTIMES;
   TC2 = TCNT + MS_4; // first interrupt in 4 ms
   asm bset TIE, 0b00000100; // Channel 2 Interrupt
   asm cli; // Enable interrupts
   for(;;) {
      while(counter != 0)  asm wai;
         PORTB = PORTB ^ LED1; // Toggle the bit
         counter = NTIMES;
   }
}
```

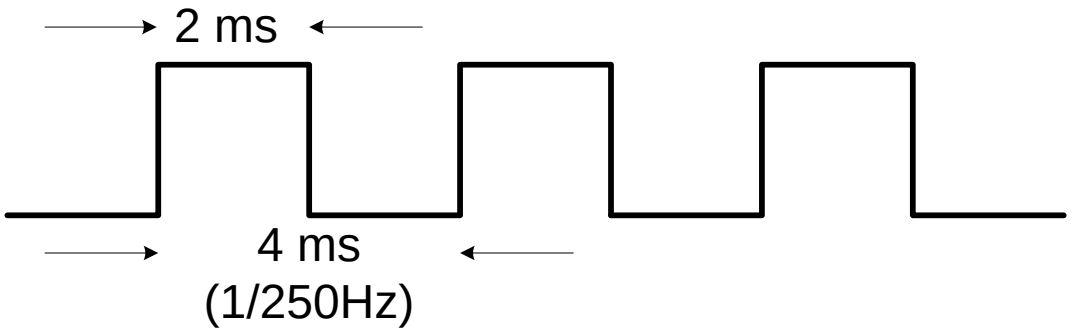
Vector Address	Interrupt Source	CCR Mask	Local Enable	HPRIO Value to Elevate
$FFEE, $FFEF	Enhanced Capture Timer channel 0	I-Bit	TIE (C0I)	$EE
$FFEC, $FFED	Enhanced Capture Timer channel 1	I-Bit	TIE (C1I)	$EC
$FFEA, $FFEB	Enhanced Capture Timer channel 2	I-Bit	TIE (C2I)	$EA
$FFE8, $FFE9	Enhanced Capture Timer channel 3	I-Bit	TIE (C3I)	$E8
$FFE6, $FFE7	Enhanced Capture Timer channel 4	I-Bit	TIE (C4I)	$E6
$FFE4, $FFE5	Enhanced Capture Timer channel 5	I-Bit	TIE (C5I)	$E4
$FFE2, $FFE3	Enhanced Capture Timer channel 6	I-Bit	TIE (C6I)	$E2
$FFE0, $FFE1	Enhanced Capture Timer channel 7	I-Bit	TIE (C7I)	$E0
$FFDE, $FFDF	Enhanced Capture Timer over o w	I-Bit	TSRC2 (TOF)	$DE

Example 14-10 – In C (continued)

```
/* Interrupt Service Routine */
void interrupt VectorNumber_Vtimch2 oc2_isr(void)
{
   counter--;
   TC2 = TC2 + MS_4;
}
```

Example 14.15

- Generate a 250 Hz frequency digital square wave (see below) on pin PT0. Use interrupts. The Bus-clock has an 24 MHz frequency.
- Solution:
  - Using a pre-scale factor of 16, the clock period to increment TCNT is set to 2/3 microsecond. Thus half a period (2 ms) takes 3000 TCNT ticks.



Example 14-15 – In C

```
#define HALF_P 3000   // 3000 ticks = 2 ms
void interrupt VectorNumber_Vtimch0 oc0_isr(void); //Proto.
void main(void) {
  // Initialise the Timer
  TSCR1 = 0b10010000;  // Enable, Fast clear
  TSCR2 = 0b00000100;  // 2/3 micro-s ticks
  // Setup Channel 0
  TIOS = TIOS | 0b00000001; // OC for channel 0
  TCTL2 = 0b00000001;  // Channel 0 set to toggle
  TC0 = TCNT + HALF_P; // first interrupt in 2 ms
  // Enable Interrupts
  asm bset TIE, 0b00000001; // Channel 0 Interrupt
  asm cli; // Global bit
  for(;;) { asm wai; } /* wait forever */
}
/* Interrupt Service Routine */
void interrupt VectorNumber_Vtimch0 oc0_isr(void)
{
   TC0 = TC0 + HALF_P;
}
```

	BIT7	6	5	4	3	2	1	BIT0
R W	OM7	OL7	OM6	OL6	OM5	OL5	OM4	OL4
RESET	0	0	0	0	0	0	0	0

	BIT7	6	5	4	3	2	1	BIT0
R W	OM3	OL3	OM2	OL2	OM1	OL1	OM0	OL0
RESET	0	0	0	0	0	0	0	0

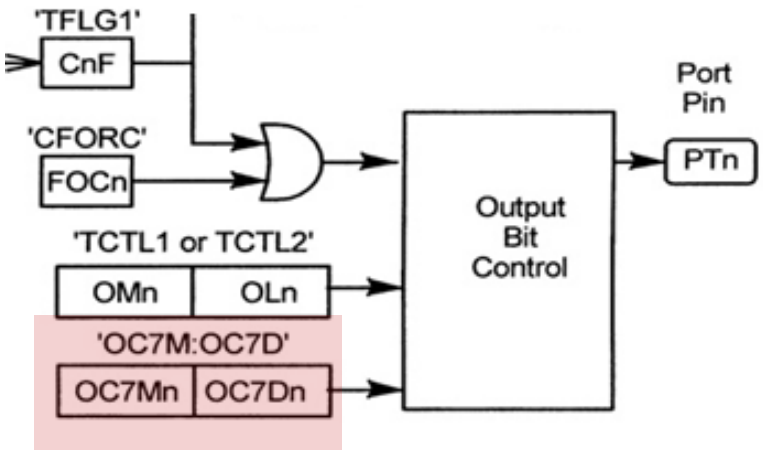
Figure 3-8 Timer Control Register 1/Timer Control Register 2 (TCTL1/TCTL2)

Table 3-2 Compare Result Output Action

OMn	OLn	Action
0	0	Timer disconnected from output pin logic
0	1	Toggle OCn output line
1	0	Clear OCn output line to zero
1	1	Set OCn output line to one

Channel 7 – controlling 8 output pins

- Channel 7 can control up to 8 pins
- The register OC7M specifies which pins can be controlled by channel 7
- Register OC7D specifies the level to place on the output pin when TCNT equals TC7 (output-compare action) (as well as output-compare events from other channels).
- This function allows control of many pins with a single output-compare event
  - Used for applications where signal changes on multiple pins require synchronization



Channel 7 – controlling 8 output pins

	BIT7	6	5	4	3	2	1	BIT0
R W	OC7M7	OC7M6	OC7M5	OC7M4	OC7M3	OC7M2	OC7M1	OC7M0
RESET:	0	0	0	0	0	0	0	0

Figure 3-3 Output Compare 7 Mask Register (OC7M)

Read or write anytime.

Setting the OC7Mn (n ranges from 0 to 6) bit of OC7M register configures the corresponding port to be an output port when the IOS7 bit and the corresponding IOSn (n ranges from 0 to 6) bit of TIOS register are set to be an output compare. Refer to the note on Section **4.2.4** for more insight.

NOTE: *A successful channel 7 output compare overrides any channel 6:0 compares. For each OC7M bit that is set, the output compare action reflects the corresponding OC7D bit.*

Channel 7 – controlling 8 output pins

	BIT7	6	5	4	3	2	1	BIT0
R W	OC7D7	OC7D6	OC7D5	OC7D4	OC7D3	OC7D2	OC7D1	OC7D0
RESET:	0	0	0	0	0	0	0	0

Figure 3-4 Output Compare 7 Data Register (OC7D)

Read or write anytime.

A channel 7 output compare can cause bits in the output compare 7 data register to transfer to the timer port data register depending on the output compare 7 mask register.

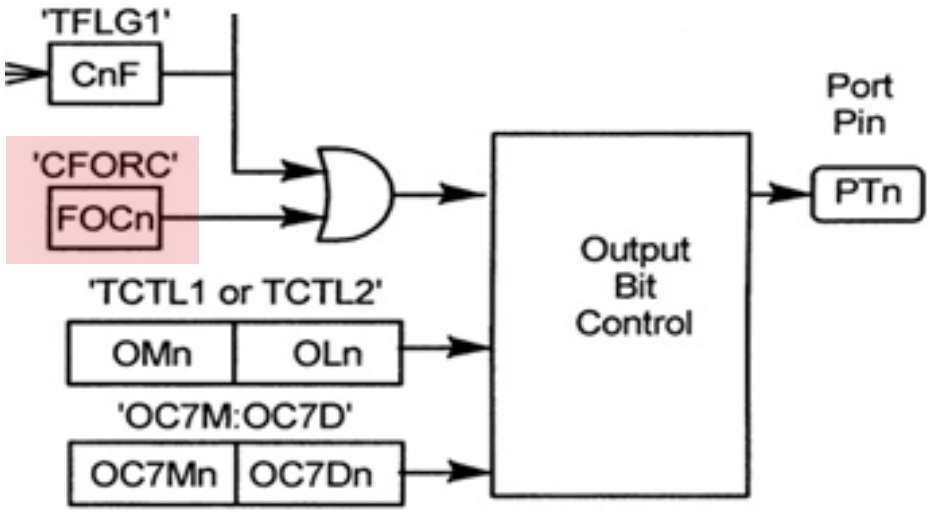
Forced Output-Compare

- To force an immediate action
- Write a 1 in the CFORC register to force the operation
- The operation only affects the pin value. The flag is not set and no interrupt occurs.

	BIT7	6	5	4	3	2	1	BIT0
R	0	0	0	0	0	0	0	0
W	FOC7	FOC6	FOC5	FOC4	FOC3	FOC2	FOC1	FOC0
RESET:	0	0	0	0	0	0	0	0

Figure 3-2 Timer Compare Force Register (CFORC)

Read anytime but will always return $00 (1 state is transient). Write anytime.


'TFLG1'
CnF
'CFORC'
FOCn
'TCTL1 or TCTL2'
OMn
OLn
'OC7M:OC7D'
OC7Mn
OC7Dn
Output Bit Control
Port Pin
PTn

Output Compare Software Checklist

1. Initialize the interrupt vector(s) for the timer channel(s) if interrupts are to be used.
2. Set bit 7 (TEN) in TSCR1 to enable the timer.
3. Set bits in TIOS to enable the Output Compare Channels.
4. Initialize the OMn and OLn bits in TCTL1 and TCTL2 if the timer output pins are to be activated.
5. Set the bits in OC7M and OC7D if any timer channel 7 override is to be done.
6. Load the current value for TCNT and add to it the required delay.
7. Store the (TCNT + delay) value into the TCn register to be used.
8. Reset the flag(s) in TFLG1.
9. Enable any required interrupts in TIE.
10. Unmask interrupts by clearing the I bit in the CC register.
11. Wait for the output compare to set the flag by either polling the flag or waiting for the interrupt.
12. After the output compare occurs, reinitialize the TCn register with the new delay value by adding the delay to the current TCn value.
13. Reset the CnF flag bit to prepare for the next output compare.

Topics of discussion

- Introduction to the HCS Standard Timer Module
- Output-compare functions
- Input-capture functions
- Pulse-width modulation
- Reading: FreeScale Documentation: ECT_16B8C Block User Guide V01.06, Chapter 14 (Sections 14.1 - 14.5 , 14.10)

Input-Capture

- Time is represented by the main counter register (TCNT)
- An event is represented by an ascending or descending signal transition
- At an event, the time of the event is registered by capturing the TCNT value in the channel data register
- The HCS12 can provides 8 channels
  - 16-bit register for capturing the time value
  - An input pin
  - Logic for detecting signal transitions and generating interrupts

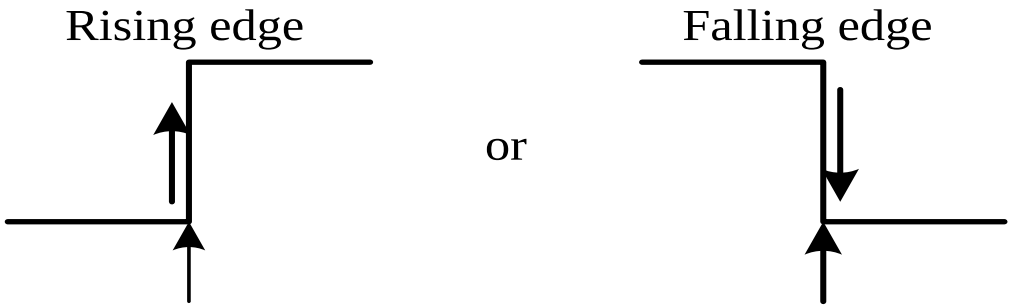


Figure 8.3 Events represented by signal edges

Input-Capture

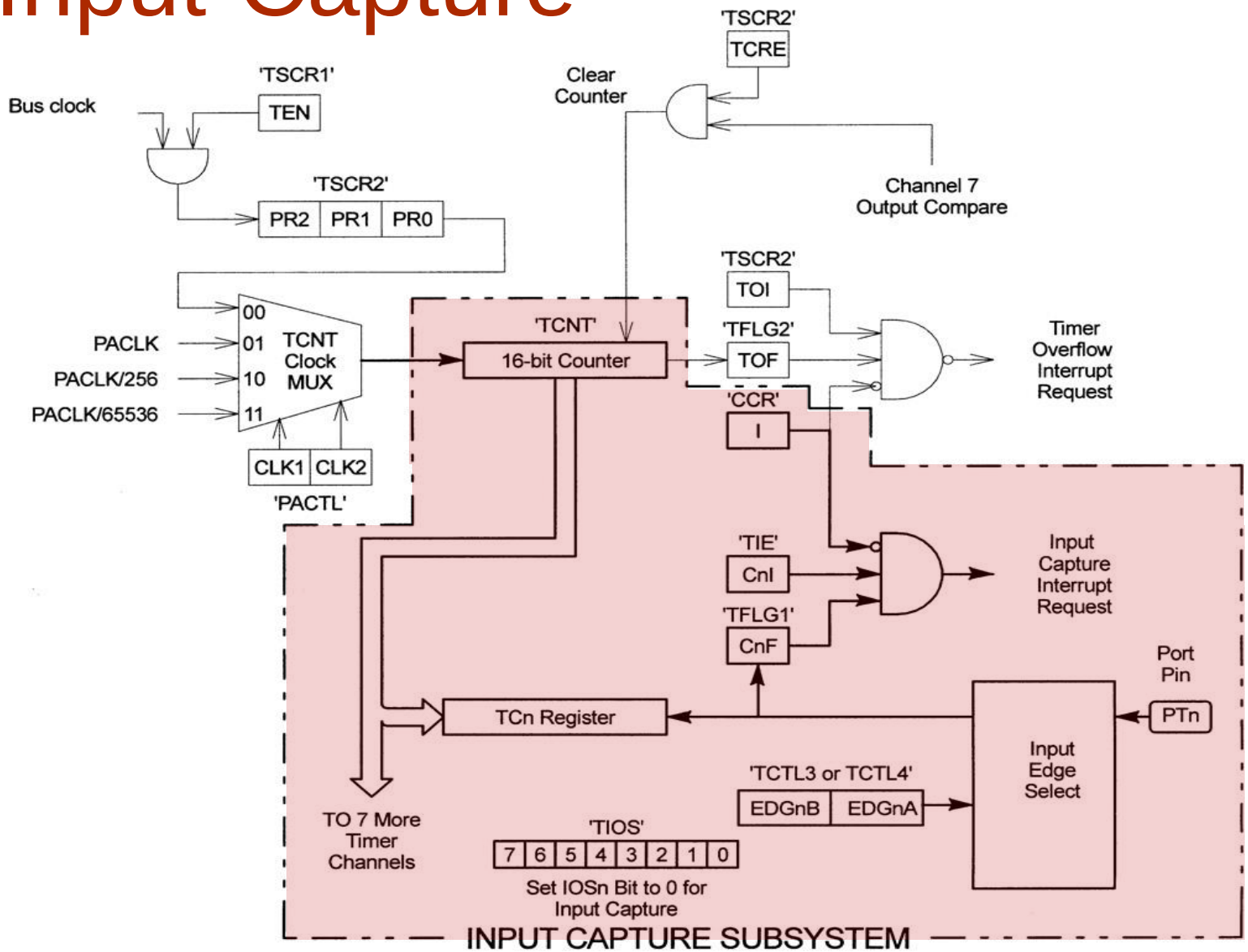


Figure 14-5 Input capture hardware.

Configuring a Channel for Input-Capture

	BIT7	6	5	4	3	2	1	BIT0
R W	IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0
RESET:	0	0	0	0	0	0	0	0

Figure 3-1 Timer Input Capture/Output Compare Register (TIOS)

Read or write anytime.

IOS[7:0] — Input Capture or Output Compare Channel Configuration
0 = The corresponding channel acts as an input capture
1 = The corresponding channel acts as an output compare.

- What does the following instruction configure?

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

Interpreting Signals

	BIT7	6	5	4	3	2	1	BIT0
R W	EDG7B	EDG7A	EDG6B	EDG6A	EDG5B	EDG5A	EDG4B	EDG4A
RESET:	0	0	0	0	0	0	0	0

	BIT7	6	5	4	3	2	1	BIT0
R W	EDG3B	EDG3A	EDG2B	EDG2A	EDG1B	EDG1A	EDG0B	EDG0A
RESET:	0	0	0	0	0	0	0	0

Figure 3-9 Timer Control Register 3/Timer Control Register 4 (TCTL3/TCTL4)

Table 3-3 Edge Detector Circuit Configuration

EDGnB	EDGnA	Configuration
0	0	Capture disabled
0	1	Capture on rising edges only
1	0	Capture on falling edges only
1	1	Capture on any edge (rising or falling)

Input-Capture Flags

	BIT7	6	5	4	3	2	1	BIT0
R W	C7F	C6F	C5F	C4F	C3F	C2F	C1F	C0F
RESET:	0	0	0	0	0	0	0	0

Figure 3-12 Main Timer Interrupt Flag 1 (TFLG1)

TFLG1 indicates when interrupt conditions have occurred. To clear a bit in the flag register, write a one to the bit.

Read anytime. Write used in the clearing mechanism (set bits cause corresponding bits to be cleared). Writing a zero will not affect current status of the bit.

When TFFCA bit in TSCR register is set, a read from an input capture or a write into an output compare channel ($10–$1F) will cause the corresponding channel flag CnF to be cleared.

Input-Capture Interrupts

	BIT7	6	5	4	3	2	1	BIT0
R W	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I
RESET:	0	0	0	0	0	0	0	0

Figure 3-10 Timer Interrupt Enable Register (TIE)

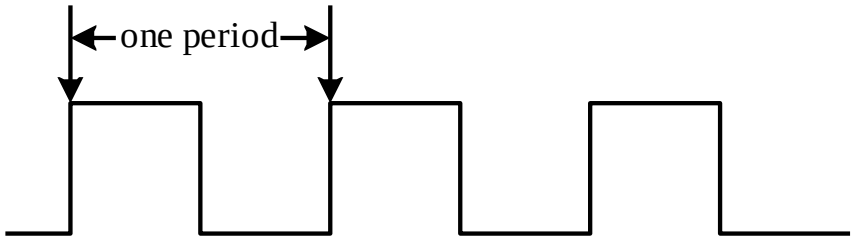
Read or write anytime.

The bits in TIE correspond bit-for-bit with the bits in the TFLG1 status register. If cleared, the corresponding flag is disabled from causing a hardware interrupt. If set, the corresponding flag is enabled to cause a interrupt.

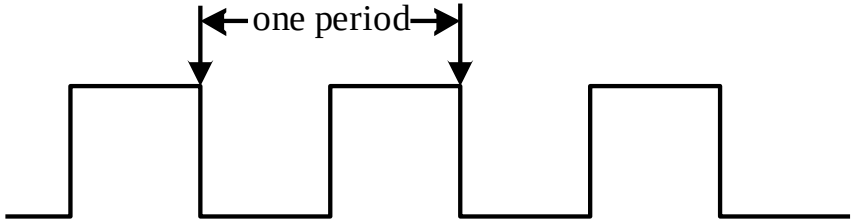
C7I–C0I — Input Capture/Output Compare "n" Interrupt Enable

Application: Measuring Periods

- Capture the time between two ascending transitions to measure the width of a period.



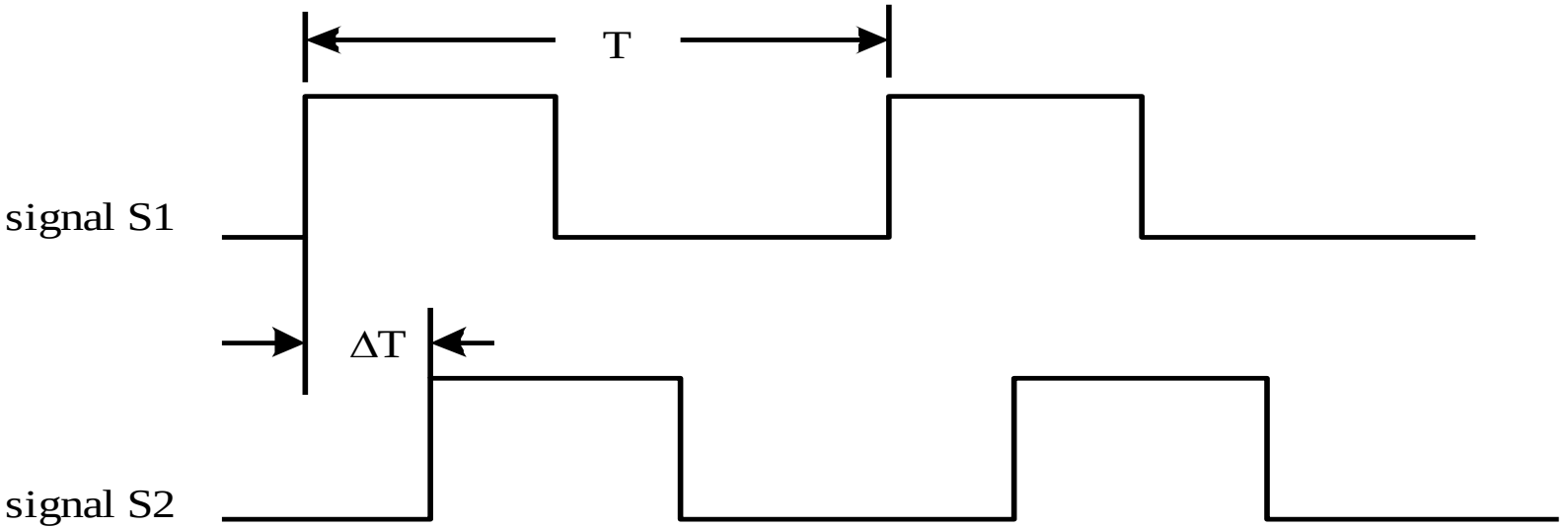
(a) Capture two rising edges



(b) Capture two falling edges

Figure 8.8 Period measurement by capturing two consecutive edges

Application: Measuring Phase Difference



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

Figure 8.13 Phase difference definition for two signals

Application: Pulse Width

- Find the time difference between the ascending transition and descending transition.

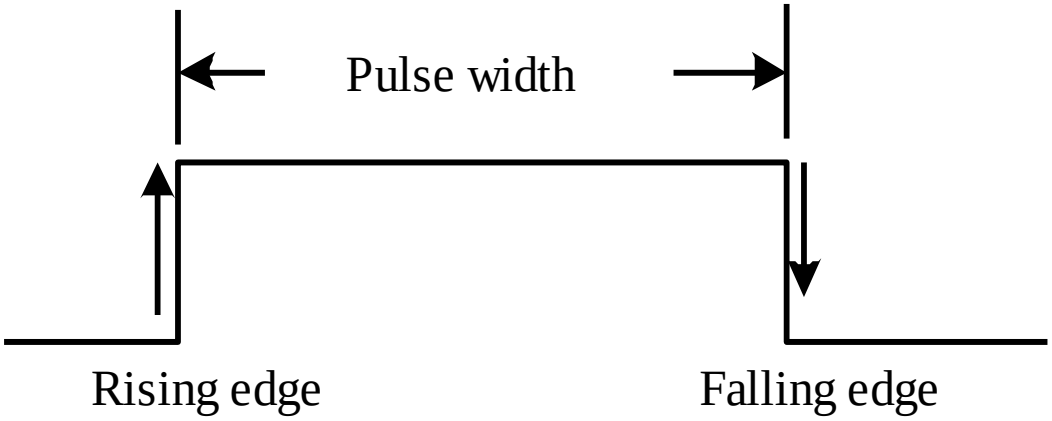


Figure 8.9 Pulse-width measurement using input capture

Application: Time Reference

- Combine with output compare functions.

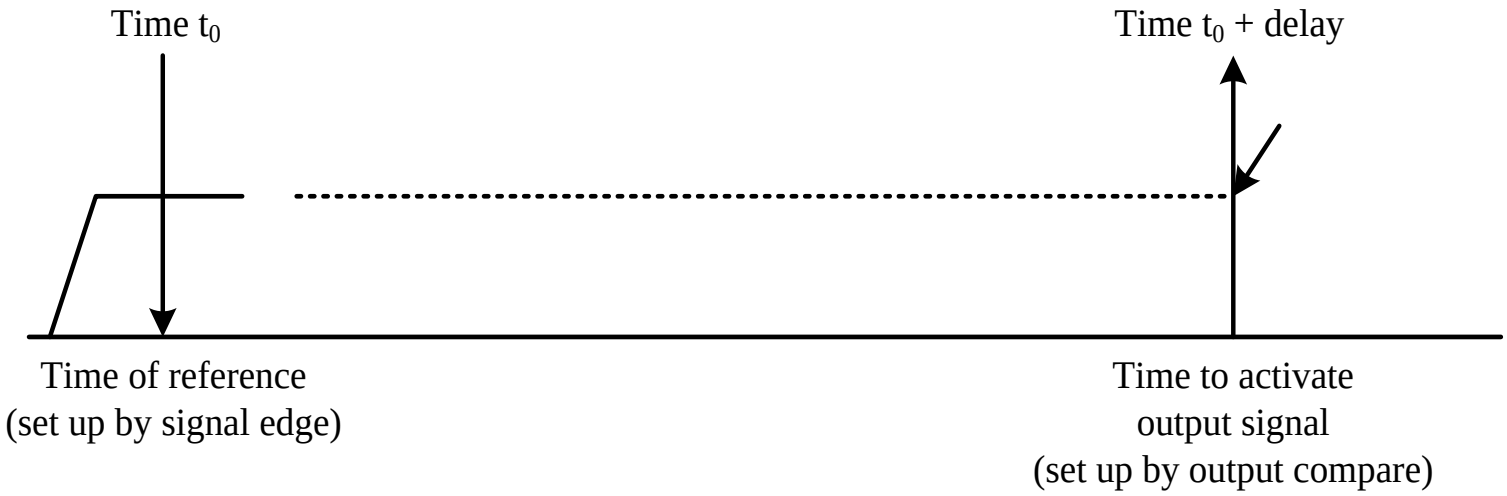


Figure 8.11 A time reference application

Application: Counting Events

- Count the events corresponding to signal transitions (use interrupts).

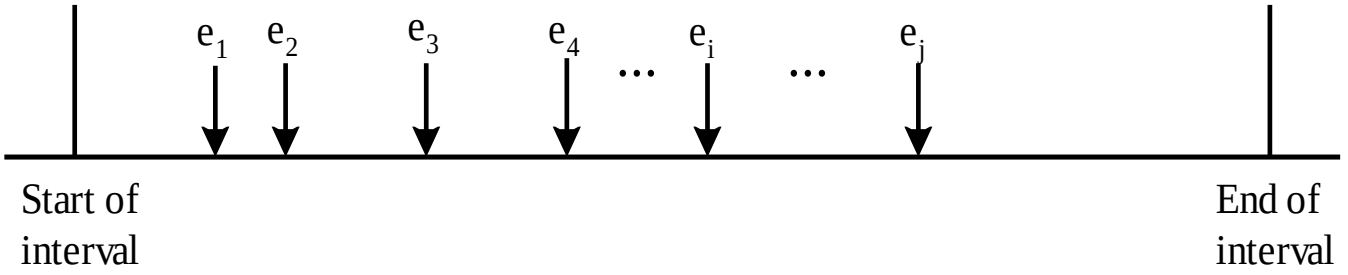


Figure 8.10 Using an input-capture function for event counting

Example 14.21

- Use the input-capture feature of channel 1 to measure the width of an unknown pulse. The width is less than 300ms. The Bus-clock is set to 24 MHz.
- Solution
  - With a pre-scale factor of 1, the maximum width that can be measured is 2^{16} x 41 2/3 ns = 2.731 ms
  - With a pre-scale factor of 128, the maximum width that can be measured is 2^{16} x 41 2/3 ns x 128 = 349.5 ms. (use this value)

Example 14.21

```
getperiod:
  bset TSCR1,%10010000  ; Enable timer, fast clear
  bclr TIOS,%00000010   ; Input capture, TC1
  bclr TCTL4,mTCTL4_EDG1B
  bset TCTL4,mTCTL4_EDG1A     ; TC1 set for rising edge 01
spin1: brclr TFLG1,mTFLG1_C1F,spin1  ; waits til flag is set
  ldd TC1    ; get current count, also clears C1F
  pshd       ; saves count on the stack
  bset TCTL4,mTCTL4_EDG1B
  bclr TCTL4,mTCTL4_EDG1A     ; TC1 set for falling edge 10
spin2: brclr TFLG1,mTFLG1_C1F,spin2  ; waits til flag is set
  ldd TC1    ; get current count, also clears C1F
  subd 0,SP  ; subtract initial count
  leas 2,SP  ; restore the stack
  rts        ; count returned in D
```

	BIT7	6	5	4	3	2	1	BIT0
R W	EDG7B	EDG7A	EDG6B	EDG6A	EDG5B	EDG5A	EDG4B	EDG4A
RESET:	0	0	0	0	0	0	0	0

	BIT7	6	5	4	3	2	1	BIT0
R W	EDG3B	EDG3A	EDG2B	EDG2A	EDG1B	EDG1A	EDG0B	EDG0A
RESET:	0	0	0	0	0	0	0	0

Figure 3-9 Timer Control Register 3/Timer Control Register 4 (TCTL3/TCTL4)

Table 3-3 Edge Detector Circuit Configuration

EDGnB	EDGnA	Configuration
0	0	Capture disabled
0	1	Capture on rising edges only
1	0	Capture on falling edges only
1	1	Capture on any edge (rising or falling)

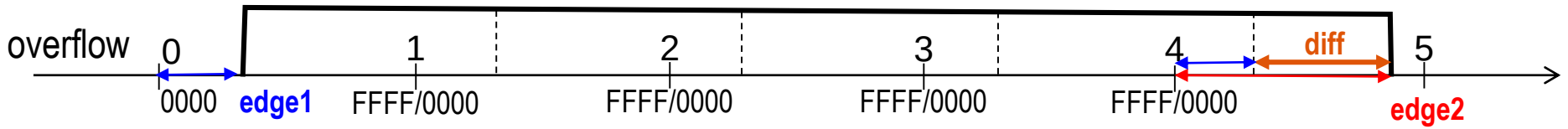
Example 14-21, modified+

- C program to measure the pulse width on pin PT1 larger than 350ms. Bus-clock is set to 24 MHz
- Solution:
  - Pre-scale factor set to 128 (5 1/3 millisecond tick)
  - Pulse can be longer than 2^{16} ticks
  - Use the counter (TCNT) overflow

Case 1

edge2 ≥ edge1

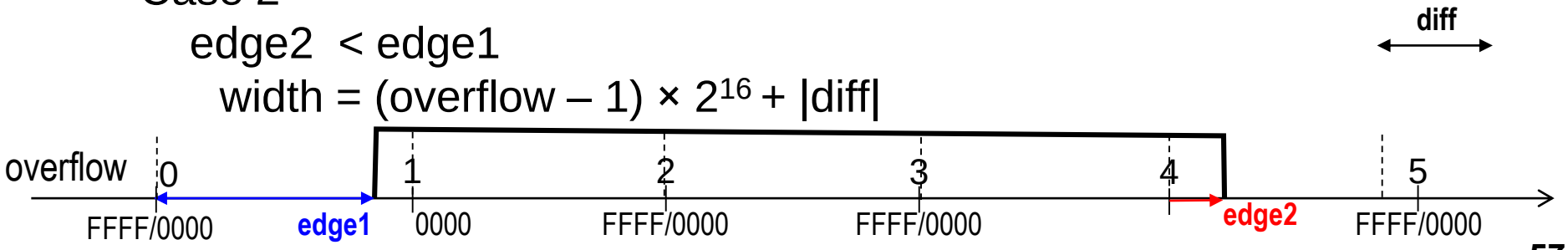
width = overflow × 2^{16} + |diff|



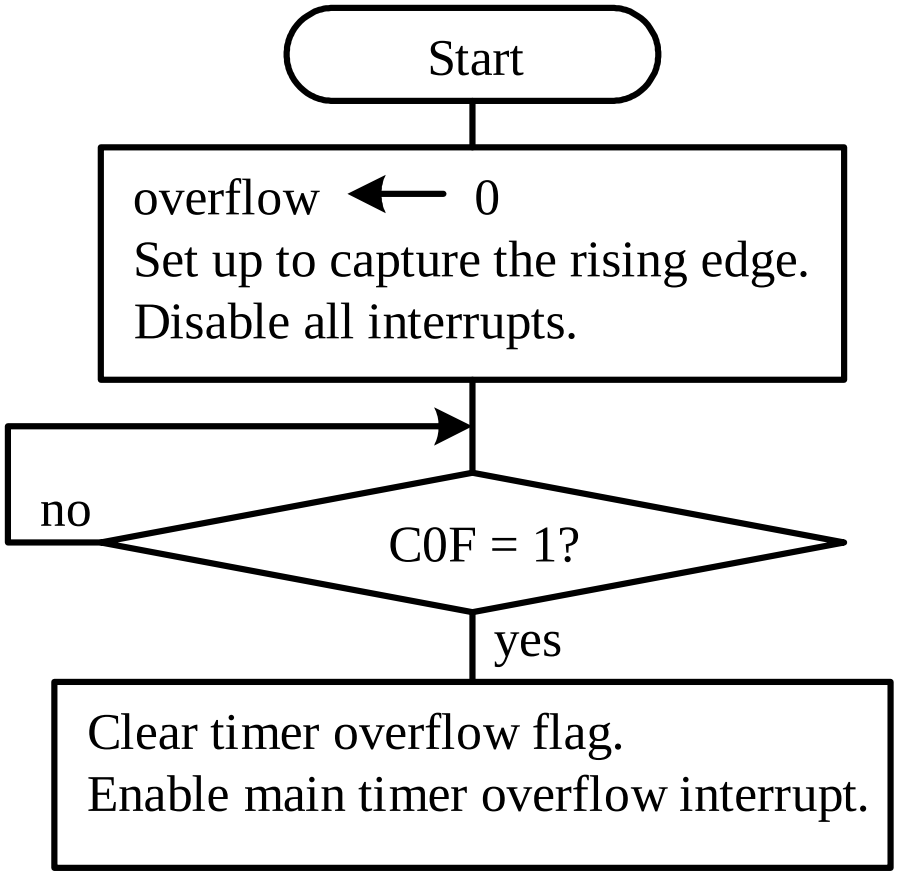
Case 2

edge2 < edge1

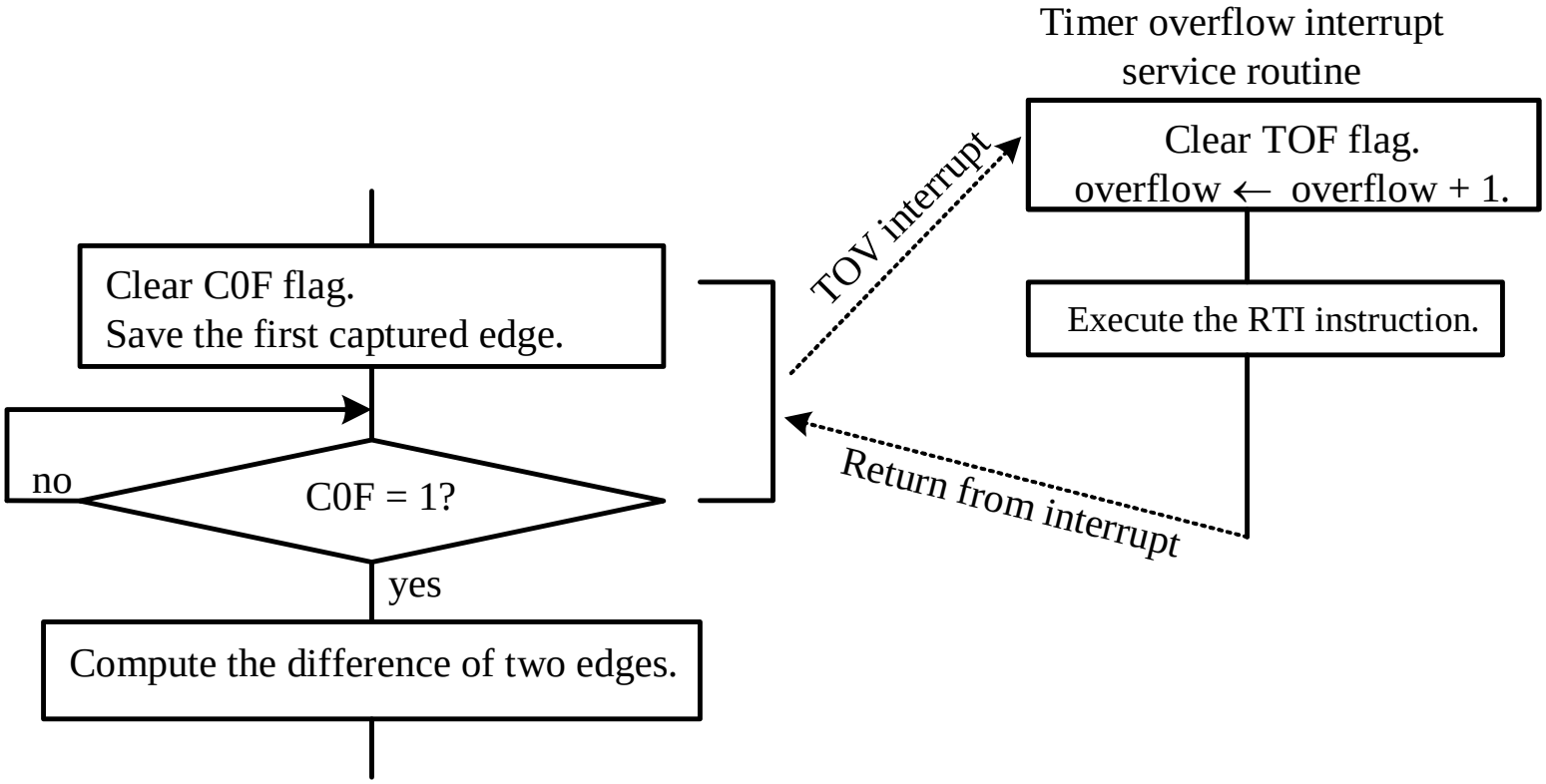
width = (overflow – 1) × 2^{16} + |diff|



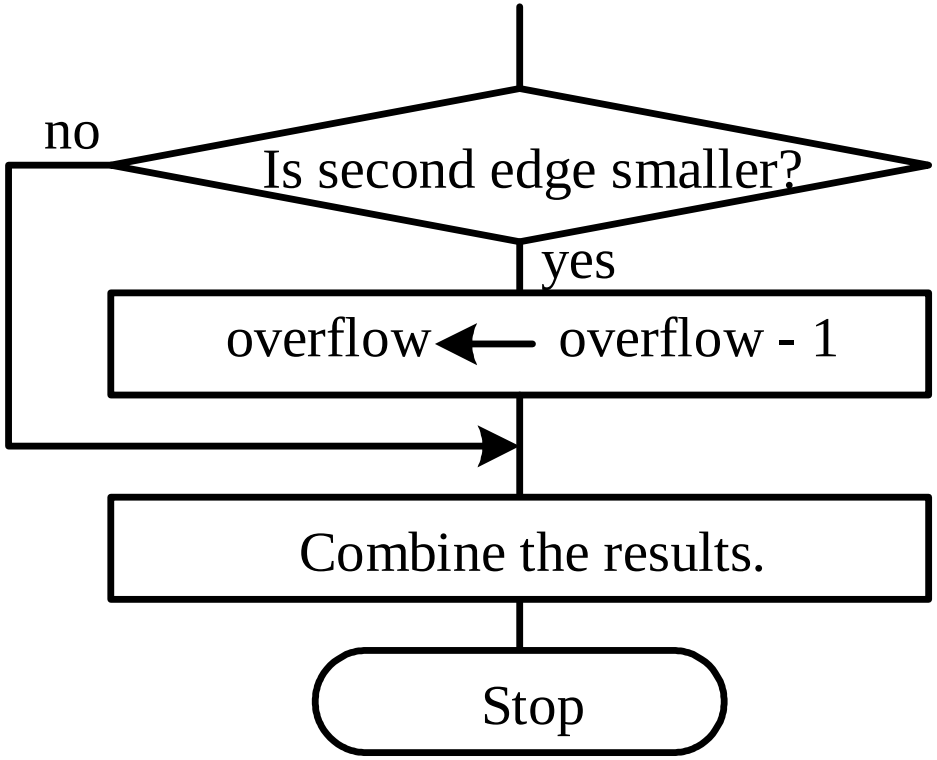
Initialisation



Measure events



Time Calculation



```
/*-------------------------------------------
Function: getperiod
Description: measures the width of a pulse
             on PT1, using overflow interrupts
-------------------------------------------*/
int overflowCount;  // count overflows
unsigned long getperiod()
{
   unsigned int edge1;
   unsigned int edge2;
   unsigned long totalTicks;
   /* setup the timer */
   TSCR1 = 0b1000000;  // Enable
   TSCR2 = 0b00000111;  // 5 1/3 micro-s ticks
   // Setup Channel 1
   TIOS = TIOS & 0b11111101; // set to input/capture
   asm bclr TCTL4,0b00001000;
   asm bset TCTL4,0b00000100;  // detect rising edge

   // continued on next slide . . .
```

```
/* getperiod() – continued */

  // Detect the pulse
   overflowCount = 0; // Setup overflow count
   while(!(TFLG1 & 0b00000010)) /* Wait for rising edge */ ;
   edge1 = TC1;  // clears flag
   asm bset TSCR2,0b1000000;  // set TOI bit - enable inter.
   asm bset TCTL4,0b00001000;
   asm bclr TCTL4,0b00000100;  // detect falling edge
   while(!(TFLG1 & 0b00000010)) /* Wait for falling edge */ ;
   edge2 = TC1;
   asm bclr TSCR2,0b1000000;  //clear TOI bit - disable inter.
   // Compute total ticks
   if(edge2 > edge1)
      totalTicks = overflowCount * 65536 + (edge2-edge1);
   else
      totalTicks = (overflowCount-1) * 65536 + (edge1-edge2);
   return(totalTicks);
}
```

```
/*-------------------------------------------
ISR: overflow_isr
Description: Counting Overflows
-------------------------------------------*/
void interrupt VectorNumber_Vtimovf overflow_isr(void)
{
  overflowCount++;
  TFLG2 = 0b1000000; // Clear interrupt
}
```

Input Capture Software Checklist

1. Initialize the interrupt vector(s) for the timer channel(s) if interrupts are to be used.
2. Set bit 7 (TEN) in TSCR1 to enable the timer.
3. Reset bits in TIOS to disable the Output Compare Channels.
4. Initialize the EDGnB and EDGnA bits in TCTL3 and TCTL4 to select the active edge for the input capture trigger signal.
5. Reset the flag(s) in TFLG1.
6. Enable any required interrupts in TIE.
7. Unmask interrupts by clearing the I bit in the CC register.
8. Wait for the input capture to set the flag by either polling the flag or waiting for the interrupt.
9. After the input capture occurs, use the data in the TCn register.
10. Reset the CnF flag bit to prepare for the next input capture.

Other HCS12 Timing Functions

- Real-Time Interrupts (RTI)
  - Generates regular interrupts when enabled
  - Frequency of interrupts is programmable
- Pulse Accumulator (part of Timer Module)
  - Uses a 16-bit counter register
  - Two modes of operation
  - Can be used to count events, frequencies or pulse duration.
- Another module – Pulse Width Modulator

Topics of discussion

- Introduction to the HCS Standard Timer Module
- Output-compare functions
- Input-capture functions
- Pulse-width modulation
- Reading: FreeScale Documentation: ECT_16B8C Block User Guide V01.06, Chapter 14 (Sections 14.1 - 14.5 , 14.10)

Pulse Width Modulation (PWM)

- Many applications require the generation of digital waveforms
- Output compare function can be used to generate digital waveforms but incur too much overhead
- Pulse width modulation requires only the initial setup of period and duty cycle for generating the digital waveforms
- The MC9S12DP256 has an 8-channel PWM module
- Each PWM channel has a period register, a duty cycle register, a control register, and a dedicated counter
- The clock input to PWM is programmable through a two-stage circuitry
- There are four possible clock sources for the PWM module: clock A, clock SA, clock B, and clock SB
- Clock SA is derived by dividing the clock A by an even number ranging from 2 to 512
- Clock SB is derived by dividing the clock B by an even number ranging from 2 to 512
- Clock A and clock B are derived by dividing the E clock by a power of 2. The power can range from 0 to 7

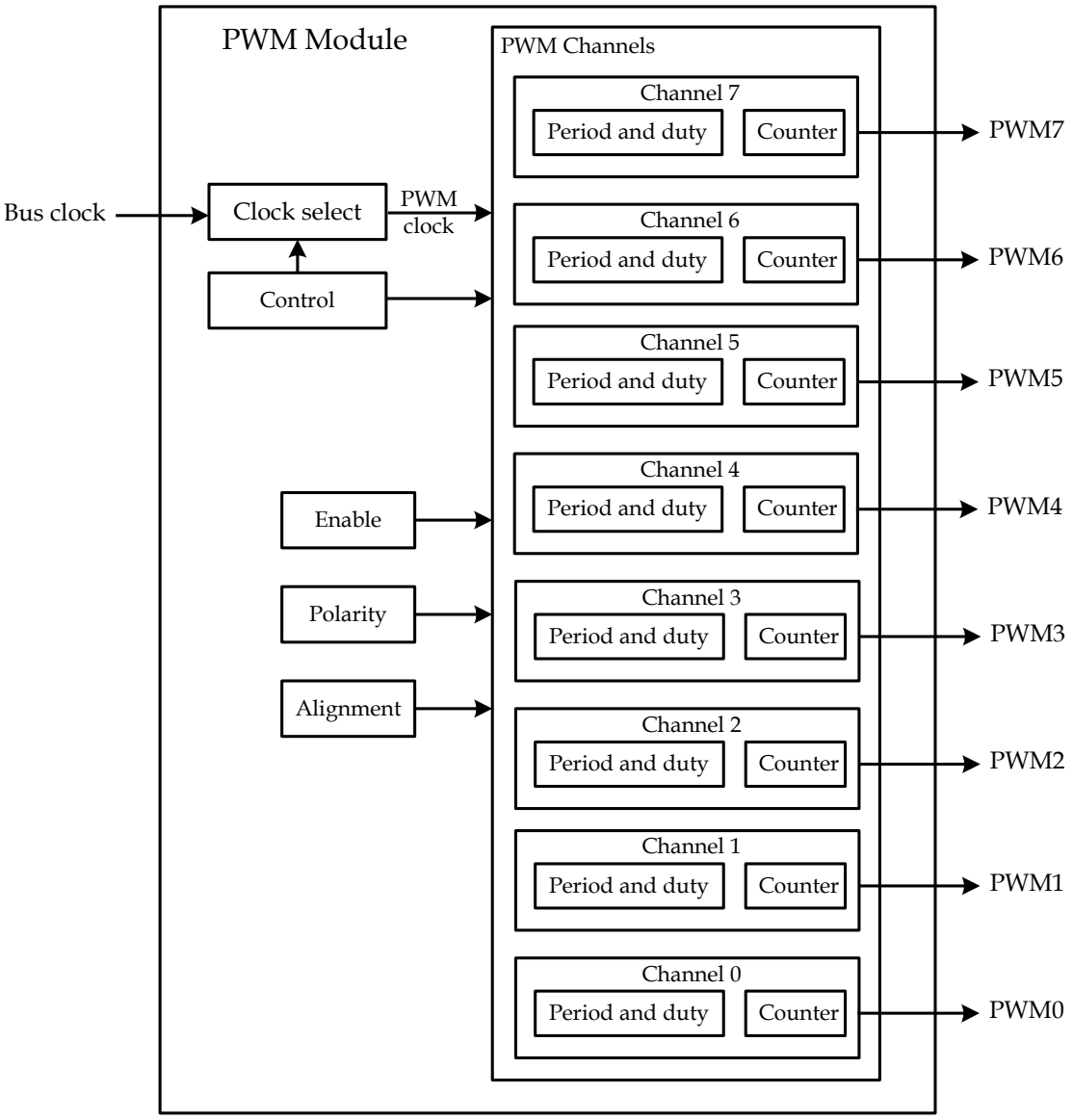
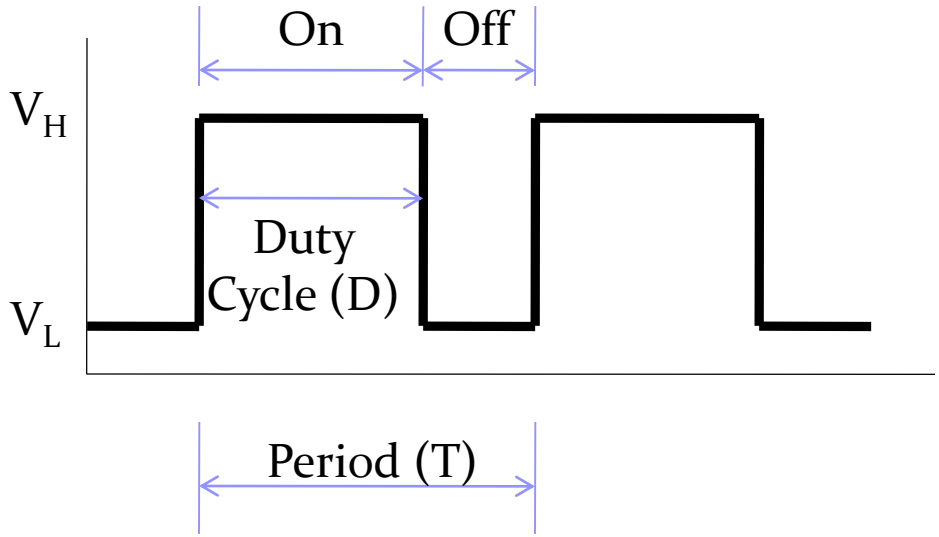


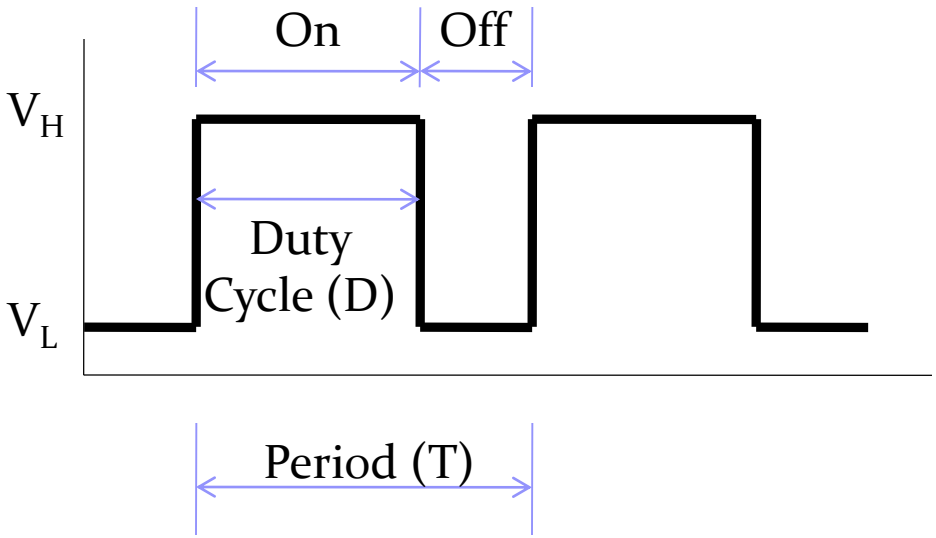
Figure 8.38 HCS12 PWM block diagram

Duty Cycle - Introduction

- The duty cycle (the width of the signal) is modulated
- It is a percentage measurement of how long the signal stays on



Duty Cycle - Definition



Duty Cycle is determined by:

$$Duty\ Cycle = \frac{On\ Time}{Period} \times 100\,\%$$

Average signal can be found as:

$$V_{avg} = D \cdot V_H + (1 - D) \cdot V_L$$

Usually, V_L is taken as zero volts for simplicity

PWM Clock Generation

- The prescale factors for clock A and clock B are determined by the PCKA2…PCKA0 and PCKB2…PCKB0 bits of the PWMPRCLK register
- Clock SA is derived by dividing clock A by the value of the PWMSCLA register and then dividing by 2
- Clock SB is derived by dividing clock B by the value of the PWMSCLB register and then dividing by 2
- The clock source selection is controlled by the PWMCLK register

	7	6	5	4	3	2	1	0
	0	PCKB2	PCKB1	PCKB0	0	PCKA2	PCKA1	PCKA0
reset:	0	0	0	0	0	0	0	0

Table 8.3 Clock B prescaler selects

PCKB2	PCKB1	PCKB0	value of clock B
0	0	0	E clock
0	0	1	E clock/2
0	1	0	E clock/4
0	1	1	E clock/8
1	0	0	E clock/16
1	0	1	E clock/32
1	1	0	E clock/64
1	1	1	E clock/128

Table 8.4 Clock A prescaler selects

PCKA2	PCKA1	PCKA0	value of clock A
0	0	0	E clock
0	0	1	E clock/2
0	1	0	E clock/4
0	1	1	E clock/8
1	0	0	E clock/16
1	0	1	E clock/32
1	1	0	E clock/64
1	1	1	E clock/128

Figure 8.41 PWM prescale clock select register (PWMPRCLK)

PWM Channel Timers

- The main part of each PWM channel x consists of an 8-bit counter (PWMCNTx), an 8-bit period register (PWMPERx), and an 8-bit duty cycle register (PWMDTYx)
- The waveform output period is controlled by the match between the PWMPERx register and PWMCNTx register
- The waveform output duty cycle is controlled by the match of the PWMDTYx register and the PWMCNTx register
- The starting polarity of the output is selectable on a per channel basis by programming the PWMPOL register
- A PWM channel must be enabled by setting the proper bit of the PWME register

	7	6	5	4	3	2	1	0
	PCLK7	PCLK6	PCLK5	PCLK4	PCLK3	PCLK2	PCLK1	PCLK0
reset:	0	0	0	0	0	0	0	0

PCLKx: PWM channel x clock select (x = 7, 6, 3, 2)
0 = clock B as the clock source
1 = clock SB as the clock source
PCLKy: PWM channel y clock select (y = 5, 4, 1, 0)
0 = clock A as the clock source
1 = clock SA as the clock source

Figure 8.42 PWM clock select register (PWMCLK)

PWM Channel Timers

	7	6	5	4	3	2	1	0
	PPOL7	PPOL6	PPOL5	PPOL4	PPOL3	PPOL2	PPOL1	PPOL0
reset:	0	0	0	0	0	0	0	0

PPOLx: PWM channel x polarity
0 = PWM channel x output is low at the start of a period, then goes high when the duty count is reached.
1 = PWM channel x output is high at the start of a period, then goes low when the duty count is reached.

Figure 8.43 PWM polarity register (PWMPOL)

	7	6	5	4	3	2	1	0
	PWME7	PWME6	PWME5	PWME4	PWME3	PWME2	PWME1	PWME0
reset:	0	0	0	0	0	0	0	0

PWMEx: PWM channel x enable
0 = PWM channel x disabled.
1 = PWM channel x enabled.

Figure 8.45 PWM enable register (PWME)

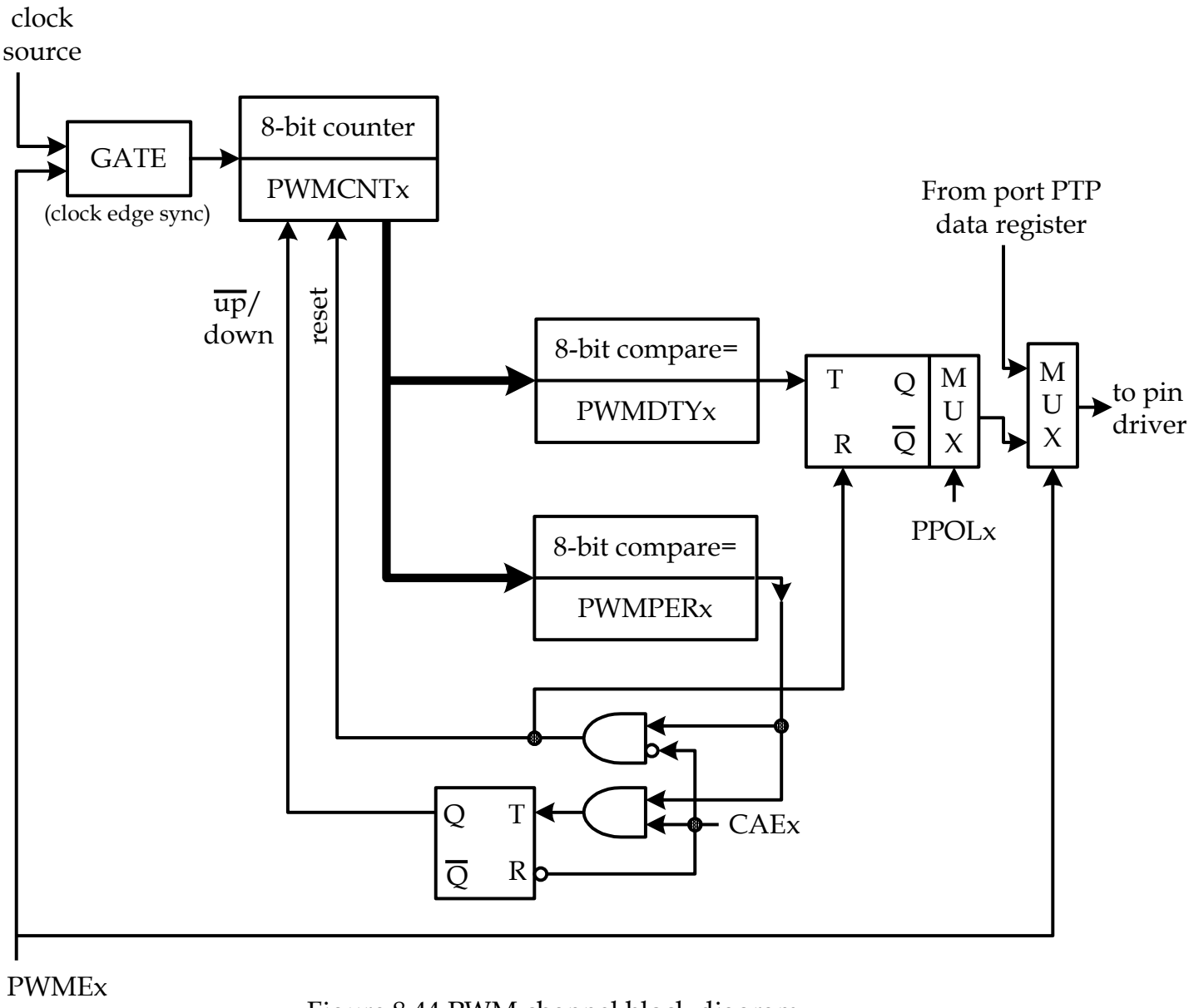


Figure 8.44 PWM channel block diagram

PWM Waveform Alignment

- PWM output waveform can be left-aligned or center-aligned
- The choice of alignment is controlled by the PWMCAE register

	7	6	5	4	3	2	1	0
	CAE7	CAE6	CAE5	CAE4	CAE3	CAE2	CAE1	CAE0
reset:	0	0	0	0	0	0	0	0

CAEx: Center aligned enable bit for channel x
0 = PWM channel x output is left aligned
1 = PWM channel x output is center alighed

Figure 8.46 PWM center align enable register (PWMCAE)

Left-Aligned Output

- The PWMCNTx counter is configured as a count-up counter
  - PWMx frequency = Clock(A, B, SA, SB frequency) ÷ PWMPERx
  - Polarity = 0
    - PWMx duty cycle = [(PWMPERx – PWMDTYx) ÷ PWMPERx] × 100%
  - Polarity = 1
    - PWMx duty cycle = [PWMDTYx ÷ PWMPERx] × 100%

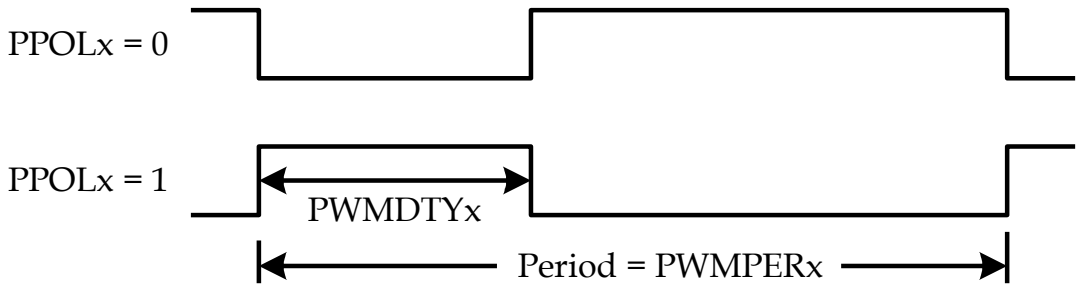


Figure 8.47 PWM left-aligned output waveform

Center-Aligned Mode

- PWM counter operates as an up/down counter and is set to count up whenever the counter is equal to $00
- When the counter matches the duty register the output flip-flop changes state causing the PWM output to also change state
- A match between the PWM counter and the period register changes the counter direction from an up-count to a down-count
- When the PWM counter decrements and matches the duty register again, the output flip-flop changes state causing the PWM output to also change state
- When the PWM counter decrements to 0, the counter direction changes from a down-count back to an up-count and the period and duty registers are reloaded from their buffers

Center-Aligned Mode

PWMx frequency = Clock (A, B, SA, or SB) frequency ÷ (2 × PWMPERx)

When polarity = 0,
PWMx duty cycle = [(PWMPERx – PWMDTYx) ÷ PWMPERx] × 100%

When polarity = 1,
PWMx duty cycle = [PWMDTYx ÷ PWMPERx] × 100%

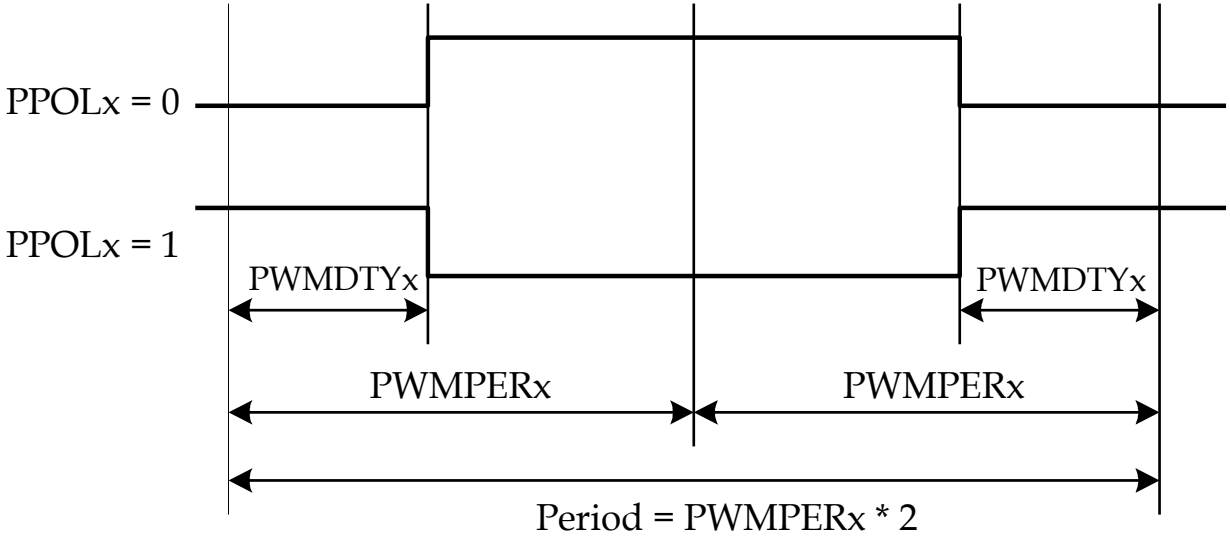


Figure 8.48 PWM center aligned output waveform

PWM 16-bit Mode (1 of 2)

- Two adjacent PWM channels can be concatenated into a 16-bit PWM channel
- The concatenation of PWM channels are controlled by the PWMCTL register
- When channel k and k+1 are concatenated, channel k is the high-order channel, whereas channel k+1 is the lower channel (k is an even number)
- A 16-bit channel outputs from the lower-order channel pin and is also enabled by the lower-order channel
- Both left-aligned and center-aligned mode apply to the 16-bit mode

PWM 16-bit Mode (2 of 2)

	7	6	5	4	3	2	1	0
	CON67	CON45	CON23	CON01	PSWAI	PFRZ	0	0
reset:	0	0	0	0	0	0	0	0

CONjk: concatenate channels j and k (j = 0, 2, 4, or 6; k = j+1)
- 0 = channel j and k are separate 8-bit PWMs
- 1 = Channels j and k are concatenated to create one 16-bit PWM channel. Channel j becomes the high order byte and channel k becomes the low order byte. Channel k output pin is used as the output for this 16-bit PWM. Channel k clock select bit determines the clock source, channel k polarity bit determines the polarity, channel k enable bit enables the output and channel k center aligned enable bit determines the output mode.

PSWAI: PWM stops in wait mode
- 0 = allow the clock to the prescaler to continue while in wait mode
- 1 = stop the input clock to the prescaler whenever the MCU is in wait mode

PFRZ: PWM counters stop in freeze mode
- 0 = allow PWM to continue while in freeze mode
- 1 = disable PWM input clock to the prescaler whenever the part is in freeze mode.

Figure 8.40 PWM control register (PWMCTL)

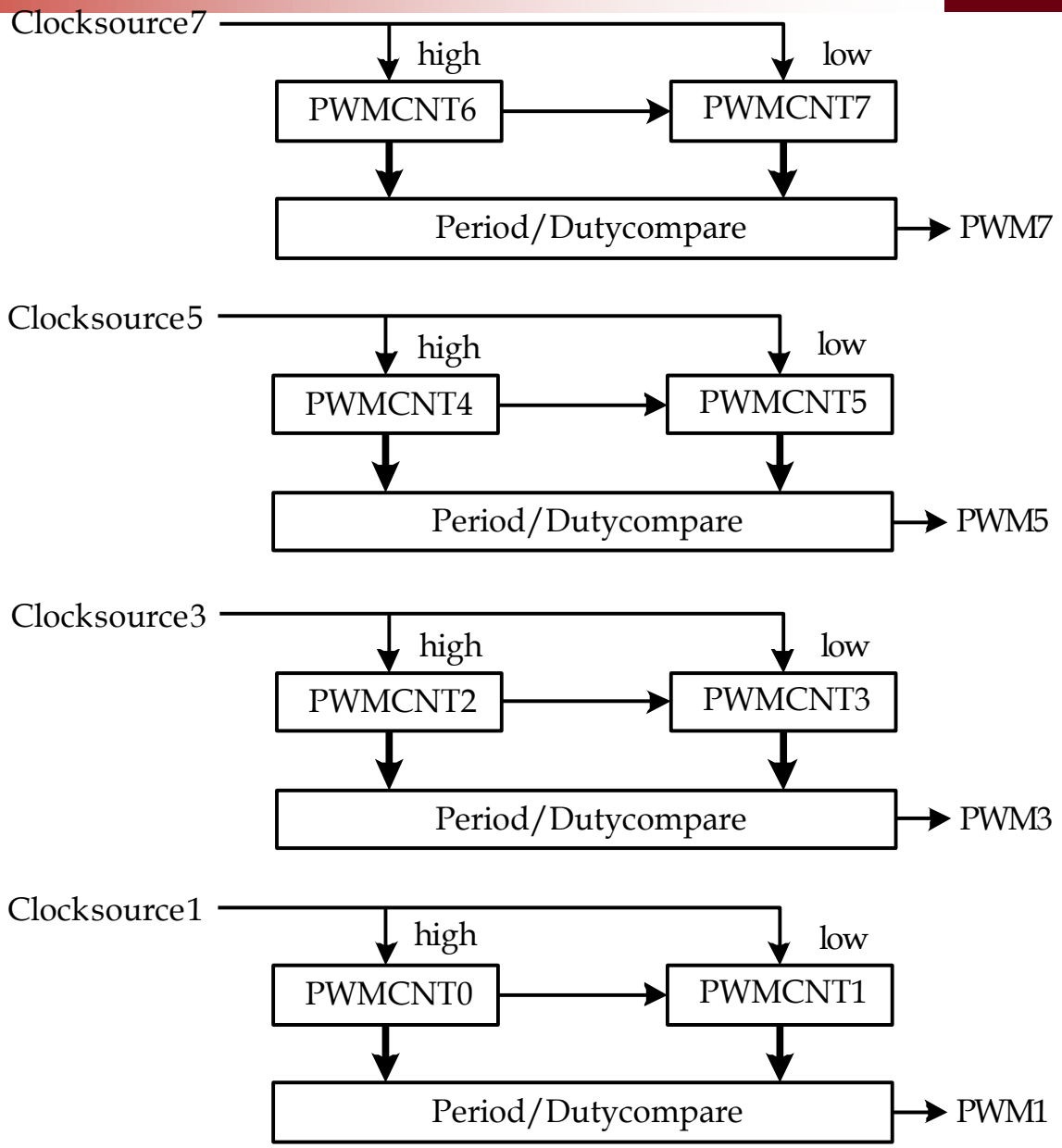


Figure8.49PWM16-bitmode

PWM Example #1

- Example 8.21 Write an instruction sequence to generate a 100KHz waveform with 50% duty cycle from the PWM0 pin (PP0). Assume that the E clock frequency is 24 MHz.
- Solution: Use the following setting:
  - Select clock A as the clock source to PWM0 and set its prescaler to 2.
  - Select left-aligned mode.
  - Load the value 120 into the PWMPER0 register (= 24000000 ÷100000 ÷2)
  - Load the value 60 into the PWMDTY0 register (= 120 × 50%)

```
#include “c:\miniide\hcs12.inc”
        …
        movb    #0,PWMCLK       ; select clock A as the clock source for PWM0
        movb    #1,PWMPRCLK     ; set clock A prescaler to 2
        movb    #1,PWMPOL       ; channel 0 output high at the start of the period
        movb    #0,PWMCAE       ; select left-aligned mode
        movb    #$0C,PWMCTL     ; 8-bit mode, stop PWM in wait and freeze mode
        movb    #120,PWMPER0    ; set period value
        movb    #60,PWMDTY0     ; set duty value
        movb    #0,PWMCNT0      ; reset the PWM0 counter
        bset    PWMEN,PWME0     ; enable PWM channel 0
```

PWM Example #1 Calculation

Example 8.21 Write an instruction sequence to generate a 100KHz waveform with 50% duty cycle from the PWM0 pin (PP0). Assume that the E clock frequency is 24 MHz.

E clock is @ 24 MHz, and the prescaler = 2 $\Rightarrow$ Clock A is @ 12 MHz

Waveform frequency = 100 KHz, then half of the period = $\frac{1}{10^5} \times \frac{1}{2} = 5 \times 10^{-6} = 5\,\mu s$

# of count for Clock A for half of the period, 5 μs = $\frac{5 \times 10^{-6}}{\frac{1}{12 \times 10^6}} = 5 \times 10^{-6} \times 12 \times 10^6 = 60$

[PWMPER0] = 120 and [PWMDTY0] = 60

PWM Example #2

- Example 8.22 Write an instruction sequence to generate a square wave with 20 μs period and 60% duty cycle from PWM0 and use center-aligned mode.
- Solution:
  - Select clock A as the clock source and set its prescaler to 2.
  - Load the value 120 into PWMPER0 register.
  - PWMPER0 = (20 × 24,000,000 ÷ 1,000,000) ÷ 2 ÷ 2 = 120
  - PWMDTY0 = PWMPER0 × 60% = 72.

```
movb   #0,PWMCLK      ; select clock A as the clock source
movb   #1,PWMPOL      ; set PWM0 output to start with high level
movb   #1,PWMPRCLK    ; set the PWM0 prescaler to clock A to 2
movb   #1,PWMCAE      ; select PWM0 center-aligned mode
movb   #$0C,PWMCTL    ; select 8-bit mode, stop PWM in wait mode
movb   #120,PWMPER0   ; set period value
movb   #72,PWMDTY0    ; set duty value
bset   PWME,PWME0     ; enable PWM channel 0
```

PWM Example #2 Calculation

Example 8.22a Write an instruction sequence to generate a square wave with 20 μs period and 60% duty cycle from PWM0 and use **center-aligned** mode.

E clock is @ 24 MHz, and the prescaler = 2 ⇒ Clock A is @ 12 MHz

The square wave has a 20 μs period.

# of count for Clock A for the period, 20 μs = $\dfrac{20\times10^{-6}}{\dfrac{1}{12\times10^{6}}} = 20\times10^{-6}\times12\times10^{6} = 240$

Note: For center-aligned mode: 2 × PWMPER0 = 240

[PWMPER0] = 120 and [PWMDTY0] = 120×60% = 72

PWM Example #3

- Example 8.23 Write an instruction sequence to generate a 50 Hz digital waveform with 80% duty cycle using the 16-bit mode from the PWM1 output pin.
- Solution: Using the following setting:
  - Select clock A as the clock source and set its prescaler to 16.
  - Select left aligned mode and select polarity 1.
  - Load the value 30000 into the PWMPER0:PWMPER1 register.
  - Load the value 24000 into the PWMDTY0:PWMDTY1 register.

```
movb   #0,PWMCLK       ; select clock A as the clock source
movb   #2,PWMPOL       ; set PWM0:PWM1 output to start with high level
movb   #4,PWMPRCLK     ; set prescaler to 16
movb   #$1C,PWMCTL     ; concatenate PWM0:PWM1, stop PWM in wait mode
movb   #0,PWMCAE       ; select left align mode
movw   #30000,PWMPER0  ; set period to 30000
movw   #24000,PWMDTY0  ; set duty to 24000
bset   PWME,PWME1      ; enable PWM0:PWM1
```

PWM Example #3 Calculation

Example 8.23 Write an instruction sequence to generate a 50 Hz digital waveform with 80% duty cycle using the 16-bit mode from the PWM1 output pin.

E clock is @ 24 MHz, and the prescaler = 16 $\Rightarrow$ Clock A is @ 1.5 MHz

Waveform frequency = 50 Hz, then the period = $\frac{1}{50} = 0.02 = 20\ ms$

# of count for the period, 20 ms = $\frac{0.02}{\frac{1}{1.5\times10^6}} = 0.02\times1.5\times10^6 = 0.03\times10^6 = 30,000$

[PWMPER0] = 30,000 and [PWMDTY0] = 30,000×80% = 24,000

PWM Example #4

- Example 8.24 Use PWM to dim the light bulb. Assume that we use the PWM0 output to control the brightness of a light bulb. Write a C program to dim the light to 10% brightness gradually in five seconds.
- The E clock frequency is 24 MHz.
- Solution:
  - Set duty cycle to 100% at the beginning.
  - Dim the brightness by 10% in the first second and then 20% per second in the following four seconds.
  - Load 100 into the PWMPER0 register at the beginning.
  - Decrement PWMPER0 by 1 every 100 ms during the first second and decrement PWMPER0 by 2 every 100 ms in the following four seconds.

PWM Example #4

```
void main ()
{
        int    dim_cnt;
        PWMCLK      = 0;          /* select clock A as the clock source */
        PWMPOL      = 1;          /* make waveform to start with high level */
        PWMCTL      = 0x0C;       /* select 8-bit mode */
        PWMPRCLK    = 2;          /* set clock A prescaler to 4 */
        PWMCAE      = 0;          /* select left-aligned mode */
        PWMPER0     = 100;        /* set period of PWM0 to 0.1 ms */
        PWMDTY0     = 100;        /* set duty cycle to 100% */
        PWME        |= 0x01;      /* enable PWM0 channel */
/* reduce duty cycle 1 % per 100 ms in the first second */
        for (dim_cnt = 0; dim_cnt < 10 ; dim_cnt ++) {
                delayby100ms(1);
                PWMDTY0--;
        }
/* reduce duty cycle 2% per 100 ms in the next 4 seconds */
        for (dim_cnt = 0; dim_cnt < 40; dim_cnt ++) {
                delayby100ms(1);
                PWMDTY0 -= 2;
        }
        asm ("swi");
}
```

PWM Software Checklist

1. Choose which clock source is to be used, Clock A or Clock B, and set the divisor in the PWMPRCLK register.
2. If a scaled clock, Clock SA or Clock SB, is to be used, set the scaler value in PWMSCLA or PWMSCLB.
3. Assign the clock source to the PWM channel being used in PWMCLK.
4. Set the polarity bit in PWMPOL.
5. Enable the left-aligned or center-aligned pulses in PWMCAE.
6. Set the period register, PWMPERx.
7. Set the duty register, PWMDTYx.
8. Enable the channel in PWME.

More Examples

Example T1

- Generate a 1kHz digital wave with the positive pulse taking 30% of the period on pin PT0. Do not use interrupts. The E-clock has an 8MHz frequency.
- Solution:
  - See the waveform below. The flowchart for the program is on the next slide.
  - Using a pre-scale factor of 8, the clock period to increment TCNT is set to 1 microsecond. Thus the positive pulse lasts for 300 cycles, and the delay between pulses is 700 cycles.

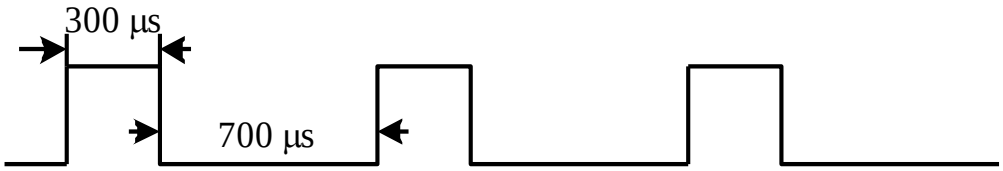
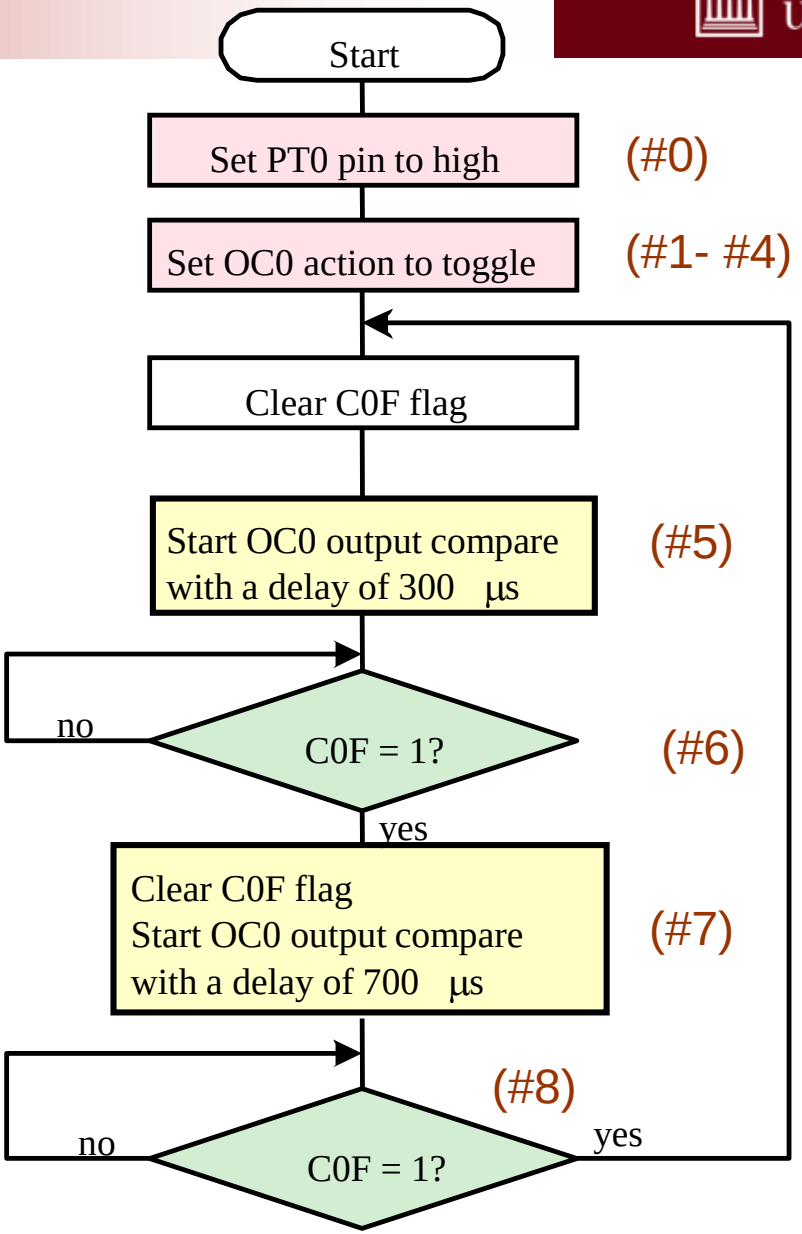


Figure 8.18 1 KHz 30 percent duty cycle waveform

Example T1 - flow-chart



The program logic flow for digital waveform generation

Example T1- code

```
#include "d:\miniide\HCS12.inc"
hi_time equ   300
lo_time  equ   700
         org   $1000
         bset  PORTT,$01          ; set PT0 pin to high (#0)
         bset  TIOS,$01           ; select OC0 function (#1)
         movb #$03,TSCR2          ; set prescale factor PR2:0 = 8 (#2)
         movb #$01,TCTL2 ; select toggle as the output compare action (OM0,OL0 = 01) (#3)
         movb #$90,TSCR ; enable TCNT (TEN) and fast timer flag clear (TFFCA =1) (#4)
         ldd   TCNT
         addd  #hi_time
         std   TC0  ; start an OC0 operation with 300 cycles as the delay + auto RESET T0F (#5)
high     brclr  TFLG1,$01,high   ; wait until C0F flag is set (#6)
         ldd   TC0 ; use TC0 rather than TCNT: the later might change its content by this time
         addd  #lo_time
         std   TC0 ; start an OC0 operation with 700 cycles as the delay + auto RESET T0F (#7)
low      brclr  TFLG1,$01,low    ; wait until C0F flag is set (#8)
         ldd   TC0 ; use TC0 rather than TCNT: the later might change its content by this time
         addd  #hi_time
         std   TC0  ; start an OC0 operation with 300 cycles as the delay + auto RESET T0F
         bra   high ; do the next period
         end
```

TC1 = Signal Generator

```
#include "mc9s12dg256.h"
 //Definitions
#define Hi_ms 600  // 600 * 4/3 micro-sec =  0.800 ms
#define Lo_ms 150   // 150 * 4/3 micro-sec = 0.200 ms
// Prototypes of local functions
void interrupt VectorNumber_Vtimch1 TimerISR (void);
void initTimer()
{
   TIOS |= 0b00000010;  // Set TC1 to output-compare
}
#define Hi 1
#define Lo 0
int levelTC1;  // level on TC1
void turnOnTimer()
{
   TCTL2 |= 0b00001100; // Sets “1” on pin 5 at out-compare
   CFORC = 0b00000010; // Force event on TC1 (“1” @ pin 1)
   levelTC1 = Hi;
   TCTL2 &= 0b11110111; // Set to toggle
   TC1 = TCNT + Hi_ms;
   TIE |= 0b00000010;  // Enable Interrupt.
}
void turnOffTimer()
{
   TIE &= 0b11111101;  // Disable Interrupt.
   TCTL2 |= 0b00001000;
   TCTL2 &= 0b11111011; // Sets “0”@ PT1 at output-compare
   CFORC = 0b00000010; // Force “0”@ pin PT1)
}
void interrupt VectorNumber_Vtimch1 TimerISR ()
{
   if(levelTC1 == Hi)
   {
      TC1 += Lo_ms;
      levelTC1 = Lo;
   }
   else
   {
      TC1 += Hi_ms;
      levelTC1 = Hi;
   }
}
void main()
{
                int duration=10000;
                initTimer();
                turnOnTimer();
                asm cli; // unmask all interrupts ~ Global bit
I=0
                while (duration) {duration--;}
                turnOffTimer();
                asm sei; // mask all interrupts ~ Global bit I=1
}
```

**** Filename:MC9S12DG256.h**

```
#define REG_BASE 0x0000 /* Base address of I/O register block */
/****** interrupt vector table *******/
#define Vtimovf  0x0000FFDE
#define Vtimch7  0x0000FFE0
...
#define Vtimch0  0x0000FFEE
/*** INTCR - Interrupt Control Register; 0x0000001E ***/
typedef union {
  byte Byte;
  struct {
    byte       :1;
    byte       :1;
    byte       :1;
    byte       :1;
    byte       :1;
    byte       :1;
    byte IRQEN :1; /* External IRQ Enable */
    byte IRQE  :1;/* IRQ select edge sensitive only */
  } Bits;
} INTCRSTR;
extern volatile INTCRSTR _INTCR @(REG_BASE + 0x0000001E);
#define INTCR       _INTCR.Byte

#define INTCR_IRQEN _INTCR.Bits.IRQEN
#define INTCR_IRQE  _INTCR.Bits.IRQE

#define INTCR_IRQEN_MASK 64
#define INTCR_IRQE_MASK 128
```

;Filename: MC9S12DG256.inc

```
;*** Memory Map and Interrupt Vectors

Vtimovf:    equ    $0000FFDE
Vtimch7:    equ    $0000FFE0
...
Vtimch0:    equ    $0000FFEE
;*** INTCR - Interrupt Control Register; 0x0000001E ***

INTCR: equ $0000001E ;**Intr Ctr Reg; 0x0000001E**
; bit numbers for usage in BCLR, BSET, BRCLR and BRSET
INTCR_IRQEN: equ 6 ;External IRQ Enable
INTCR_IRQE:  equ 7 ; IRQ Select Edge Sensitive Only
; bit position masks
mINTCR_IRQEN: equ    %01000000
mINTCR_IRQE:  equ    %10000000
```

Example T2

- Write a program to create a one second delay with the timer module.
- Solution:
  - Many possible solutions exist.
  - Set the pre-scale factor to 8 for the TCNT – this provides a 1 microsecond increment time for TCNT.
  - Use 20 output-compare operations, which means that each output-compare operation lasts 50ms.

Example T2

```
delay_1s pshx
         movb   #$90,TSCR       ; enable TCNT & fast flags clear
         movb   #$03,TSCR2      ; configure prescale factor to 8
         movb   #$01,TIOS       ; enable OC0
         ldx    #20             ; prepare to perform 20 OC0 actions
         ldd    TCNT
again    addd   #50000          ; start an output compare operation
         std    TC0             ; with 50 ms time delay
wait     brclr  TFLG1,$01,wait
         ldd    TC0
         dbne   x,again
         pulx
         rts
```

Example T3

- Use channels 7 and 0 together to generate a 2 KHz wave with the pulse taking 40% cycle. The E-clock is set to 8 MHz.
- Solution:
  - Pre-scale factor sets counter increment frequency to 2 MHz.
  - Pulse width equals 400 counter increments.
  - Use channel 7 to put 5V on PT0 every 1000 increments.
  - At 400 increments later, use channel 0 to set the pin back to 0V.
  - Lets use interrupts.

Example T3

```
#include "d:\miniide\HCS12.inc"

high_cnt        equ 400
setuservector equ $F69A
            org     $1000
            lds     #$4000
; set up OC7 interrupt pseudo vector under D-Bug12
    monitor
            ldd     #oc7_isr
            pshd
            ldd     #16
            ldx     setuservector
            jsr     0,x
            leas    2,sp
```

Example T3

```
; set up OC0 interrupt pseudo vector under D-Bug12 monitor
        ldd     #oc0_isr
        pshd
        ldd     #23
        ldx     setuservector
        jsr     0,x
        leas    2,sp
        movb    #$90,TSCR       ; enable TCNT and fast flags clear
        movb    #$81,TIOS       ; select OC7 & OC0
        movb    #$01,OC7M       ; allow OC7 to control OC0 pin
        movb    #$01,OC7D       ; OC7 action on PT0 pin is to pull high
        movb    #$22,TSCR2      ; enable pullup, set prescale factor to 4
        movb    #$02,TCTL2      ; select pull low as the OC0 action
        movb    #$81,TIE        ; enable OC7 and OC0 to interrupt
```

```
        ldd     TCNT
        addd    #1000
        std     TC7             ; start an OC7 operation
        addd    #high_cnt
        std     TC0             ; start an OC0 operation
        cli                     ; enable interrupt
loop    bra     loop            ; infinite loop to wait for interrupt
        swi

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

Example T4

- **Assembly language** program to measure the **pulse width** on pin PT0. E-clock is set to **8MHz**
- Solution:
  - Pre-scale factor set to 8 (1 microsecond tick)
  - Pulse can be longer than 2^{16} ticks
  - Use the counter (TCNT) overflow

Case 1

edge2 ≥ edge1

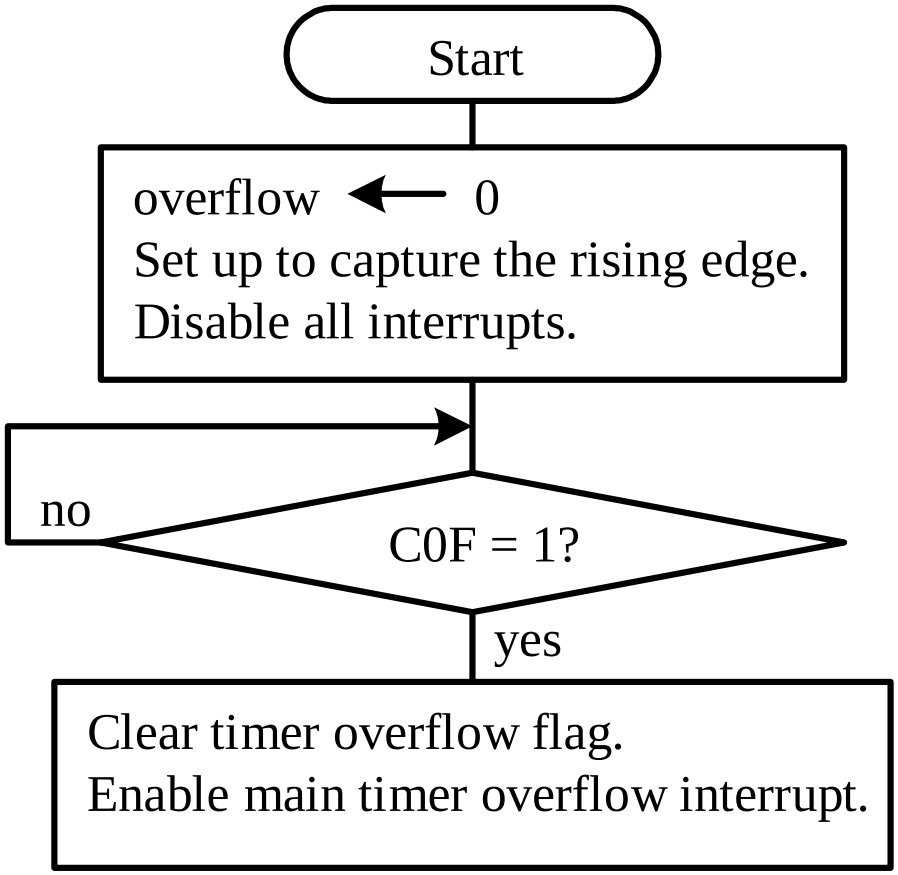
width = overflow × 2^{16} + diff

Case 2

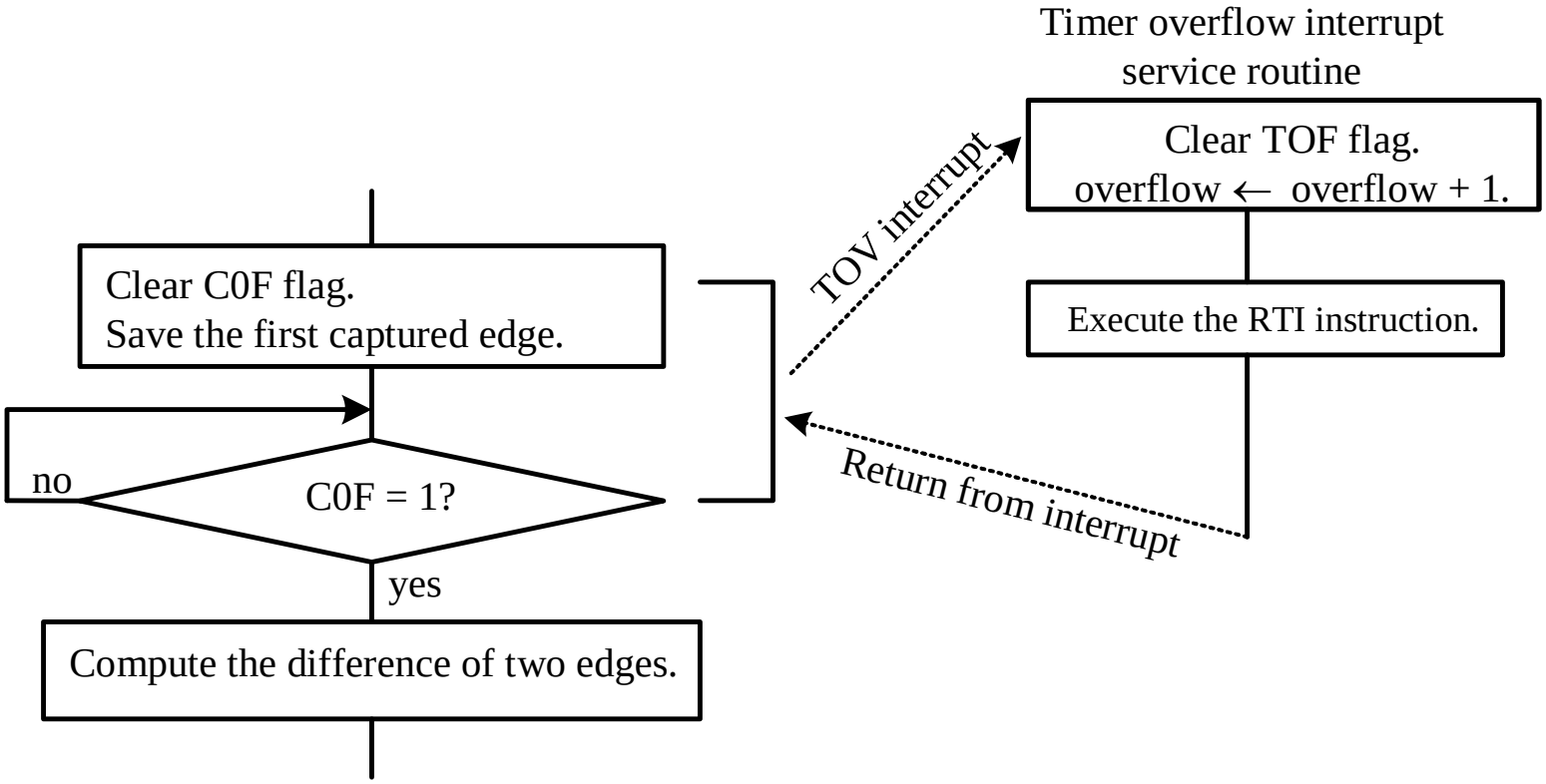
edge2 < edge1

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

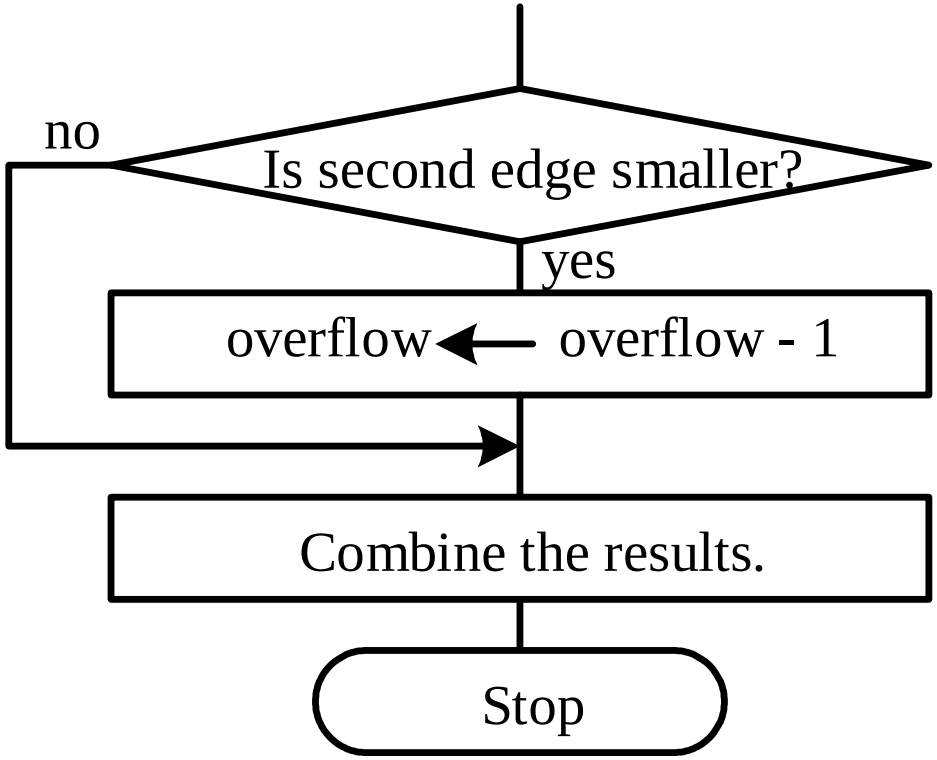
Initialisation



Mesure events



Time Calculation



```
#include "d:\miniide\HCS12.inc"
setuservector equ     $F69A
tov_vec_no    equ     23
              org     $400
edge1         rmb     2
overflow      rmb     2
pulse_width   rmb     2
              org     $1000
; the following 7 instructions set up TOV interrupt jump vector
              ldd     #tov_isr
              pshd
              ldab    #tov_vec_no
              clra
              ldx     setuservector
              jsr     0,x
              leas    2,sp
              ldd     #0
              std     overflow
              movb    #$90,TSCR
; disable TCNT overflow interrupt,  enable pull-up, set prescale factor to 16
              movb    #$24,TSCR2
              bclr    TIOS,$01      ; enable input capture 0
              bclr    DDRT,$01
```

Initialisation – Part1

```
#include "d:\miniide\HCS12.inc"
setuservector equ   $F69A
tov_vec_no equ      23
           org      $400
edge1      rmb      2
overflow   rmb      2
pulse_width rmb     2
           org      $1000
; the following 7 instructions set up TOV interrupt jump vector
           ldd      #tov_isr
           pshd
           ldab     #tov_vec_no
           clra
           ldx      setuservector
           jsr      0,x
           leas     2,sp
```

Initialisation – Part 2

```
            ldd     #0
            std     overflow
            movb    #$90,TSCR
; disable TCNT overflow interrupt,  enable pull-up,
; set prescale factor to 16
            movb    #$24,TSCR2
            bclr    TIOS,$01        ; enable input capture 0
            bclr    DDRT,$01
movb        #$01,TCTL4              ; capture the rising edge on PT0 pin

            bclr    TFLG1,$FE   ; clear C0F flag
            movb    #$01,TCTL4  ; capture the rising edge on PT0 pin
```

Capturing Transition Time

```
bclr    TFLG1,$FE  ; clear C0F flag
brclr   TFLG1,$01,* ; wait for the arrival of the first rising edge
ldd     TC0 ; save the captured first edge & clear C0F flag
std     edge1
bclr    TFLG2,$7F   ; clear TOF flag
bset    TSCR2,$40   ; enable TCNT overflow interrupt
cli
movb  #$02,TCTL4 ; capture the falling edge on PT0 pin
brclr   TFLG1,$01,* ; wait for the arrival of the falling edge
```

Interrupt Programming

```
tov_isr bclr   TFLG2,$7F      ; clear TOF flag
        ldx    overflow
        inx
        stx    overflow
        rti
```

Computing Time of Input-Capture

```
        ldd     TC0          ; edge2 → D
        subd edge1           ;edge2 – edge1 carry → c= 0 if edge2>edge1
        std    pulse_width
        bcc    next          ; is the second edge smaller?
        ldx     overflow     ; second edge is smaller (edge2<edge1 → c= 1)
        dex  ; decrement overflow
        stx    overflow
next    swi
```