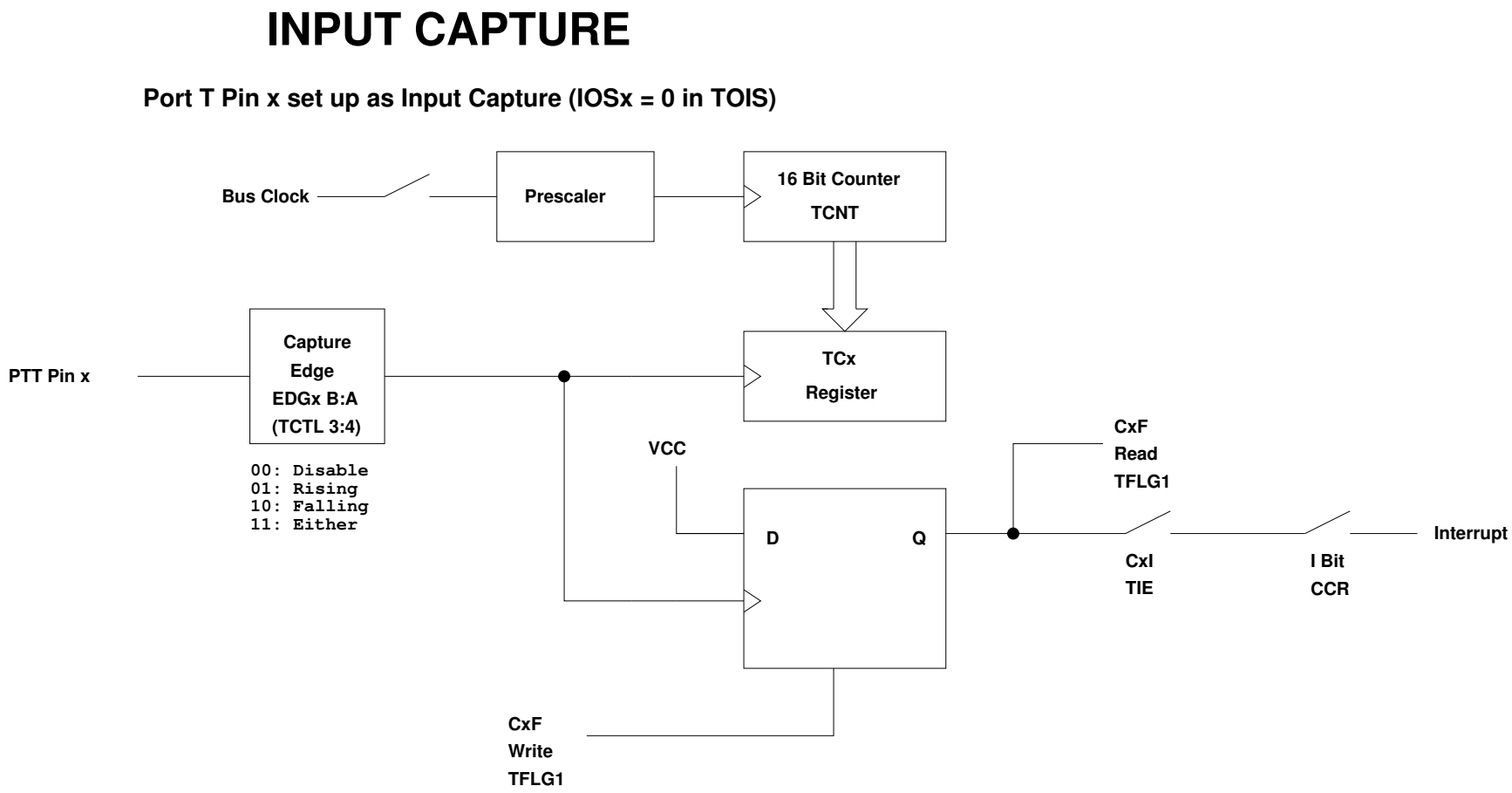
The MC9S12 Input Capture Function

- The MC9S12 allows you to capture the time an external event occurs on any of the eight Port T `PTT` pins
- An external event is either a rising edge or a falling edge
- To use the Input Capture Function:
  - Enable the timer subsystem (set `TEN` bit of `TSCR1`)
  - Set the prescaler
  - Tell the MC9S12 that you want to use a particular pin of `PTT` for input capture
  - Tell the MC9S12 which edge (rising, falling, or either) you want to capture
  - Tell the MC9S12 if you want an interrupt to be generated when the cature occurs

A Simplified Block Diagram of the MC9S12 Input Capture Subsystem

- There are eight timer channels. Any channel can be used for Input Capture



Registers used to enable Input Capture Function

Write a 1 to Bit 7 of TSCR1 to turn on timer

TEN	TSWAI	TSBCK	TFFCA					0x0046 TSCR1

To turn on the timer subsystem: TSCR1 = BIT7;

Set the prescaler in TSCR2

Make sure the overflow time is greater than the time difference you want to measure

TOI	0	0	0	TCRE	**PR2**	**PR1**	**PR0**	0x004D TSCR2

PR2	PR1	PR0	Period (μ s)	Overflow (ms)
0	0	0	0.0416	2.73
0	0	1	0.0833	5.46
0	1	0	0.1667	10.92
0	1	1	0.3333	21.84
1	0	0	0.6667	43.69
1	0	1	1.3333	86.38
1	1	0	2.6667	174.76
1	1	1	5.3333	349.53

To have overflow rate of 21.84 ms:

TSCR2 = 0x03;

Write a 0 to the bits of TIOS to make those pins input capture

IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0	0x0040 TIOS

To make Pin 3 an input capture pin: TIOS = TIOS & ~BIT3;

Write to TCTL3 and TCTL4 to choose edge(s) to capture

EDG7B	EDG7A	EDG6B	EDG6A	EDG5B	EDG5A	EDG4B	EDG4A	0x004A TCTL3

EDG3B	EDG3A	EDG2B	EDG2A	EDG1B	EDG1A	EDG0B	EDG0A	0x004B TCTL4

EDGnB	EDGnA	Configuration
0	0	Disabled
0	1	Rising
1	0	Falling
1	1	Any

To have Pin 3 capture a rising edge:

TCTL4 = (TCTL4 | BIT6) & ~BIT7;

When specified edge occurs, the corresponding bit in TFLG1 will be set.

To clear the flag, write a 1 to the bit you want to clear (0 to all others)

CF7	CF6	CF5	CF4	CF3	CF2	CF1	CF0	0x008E TFLG1

To wait until rising edge on Pin 3: while ((TFLG1 & BIT3) == 0) ;

To clear flag bit for Pin 3: TFLG1 = BIT3;

To enable interrupt when specified edge occurs, set corresponding bit in TIE register

C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I	0x004C TIE

To enable interrupt on Pin 3: TIE = TIE | BIT3;

To determine time of specified edge, read 16-bit result registers TC0 thru TC7

To read time of edge on Pin 3:

```
unsigned int time;
time = TC3;
```

USING INPUT CAPTURE ON THE MC9S12

Input Capture: Connect a digital signal to a pin of Port T. Can capture the time of an edge (rising, falling or either) – the edge will latch the value of TCNT into TCx register. This is used to measure the difference between two times.

To use Port T Pin x as an input capture pin:

1. Turn on timer subsystem (1 -> Bit 7 of TSCR1 reg)
2. Set prescaler (TSCR2 reg). To get most accuracy set overflow rate as small as possible, but larger than the maximum time difference you need to measure.
3. Set up PTx as IC (0 -> bit x of TIOS reg)
4. Set edge to capture (EDGxB EDGxA of TCTL 3-4 regs)

EDGxB	EDGxA	
0	0	Disabled
0	1	Rising Edge
1	0	Falling Edge
1	1	Either Edge

5. Clear flag (1 -> bit x of TFLG1 reg, 0 -> all other bits of TFLG1)
6. If using interrupts
   (a) Enable interrupt on channel x (1 -> bit x of TIE reg)
   (b) Clear I bit of CCR (`cli` or `enable()`)
   (c) In interrupt service routine,
      i. Read time of edge from TCx
      ii. Clear flag (1 -> bit x of TFLG1 reg, 0 -> all other bits of TFLG1)
7. If polling in main program
   (a) Wait for Bit x of TFLG1 to become set
   (b) Read time of edge from TCx
   (c) Clear flag (1 -> bit x of TFLG1 reg, 0 -> all other bits of TFLG1)

```
/* Program to determine the time between two rising edges using the *
 * MC9S12 Input Capture subsystem
 */

#include "hcs12.h"
#include "DBug12.h"

unsigned int first, second, time;

main()
{
    TSCR1 = BIT7;                /* Turn on timer subsystem */
    TSCR2 = 0x05;                /* Set prescaler for divide by 32 */
                                 /* 87.38 ms overflow time */

    /* Setup for IC1 */
    TIOS = TIOS & ~BIT1;             /* IOC1 set for Input Capture */
    TCTL4 = (TCTL4 | BIT2) & ~BIT3;  /* Capture Rising Edge */
    TFLG1 = BIT1;                    /* Clear IC1 Flag */

    /* Setup for IC2 */
    TIOS = TIOS & ~BIT2;             /* IOC2 set for Input Capture */
    TCTL4 = (TCTL4 | BIT4) & ~BIT5;  /* Capture Rising Edge */
    TFLG1 = BIT2;                    /* Clear IC2 Flag */

    while ((TFLG1 & BIT1) == 0) ;  /* Wait for 1st rising edge; */
    first = TC1;                   /* Read time of 1st edge;    */

    while ((TFLG1 & BIT2) == 0) ;  /* Wait for 2nd rising edge; */
    second = TC2;                  /* Read time of 2nd edge;    */

    time = second - first;         /* Calculate total time */
    DB12FNP->printf("time = %d cycles\n",time);
    asm(" swi");
}
```

Using D-Bug12 Routines to Print Information to the Terminal

D-Bug12 has several built-in C routines. Descriptions of these can be found in D-BUG12 V4.x.x Reference Guide. To use these routines you need to include the header file `DBug12.h`. These work like oridnary C functions, but you call them with pointers to the routines in D-Bug12. For example, you would call the `putchar()` function with the following line of C code:

```
DB12FNP->putchar(c);
```

Here is a C program to print `Hello, world!` to the terminal:

```
#include "DBug12.h"

void main(void)
{
    DB12FNP->printf("Hello, world!\n\r");
}
```

Here is a program to print a number to the terminal in three different forms:

```
#include "DBug12.h"

void main(void)
{
    unsigned int i;

    i = 0xf000;

    DB12FNP->printf("Hex:  0x%04x, Unsigned:  %u, Signed:  %d\n\r",i,i,i);
}
```

The output of the above program will be:

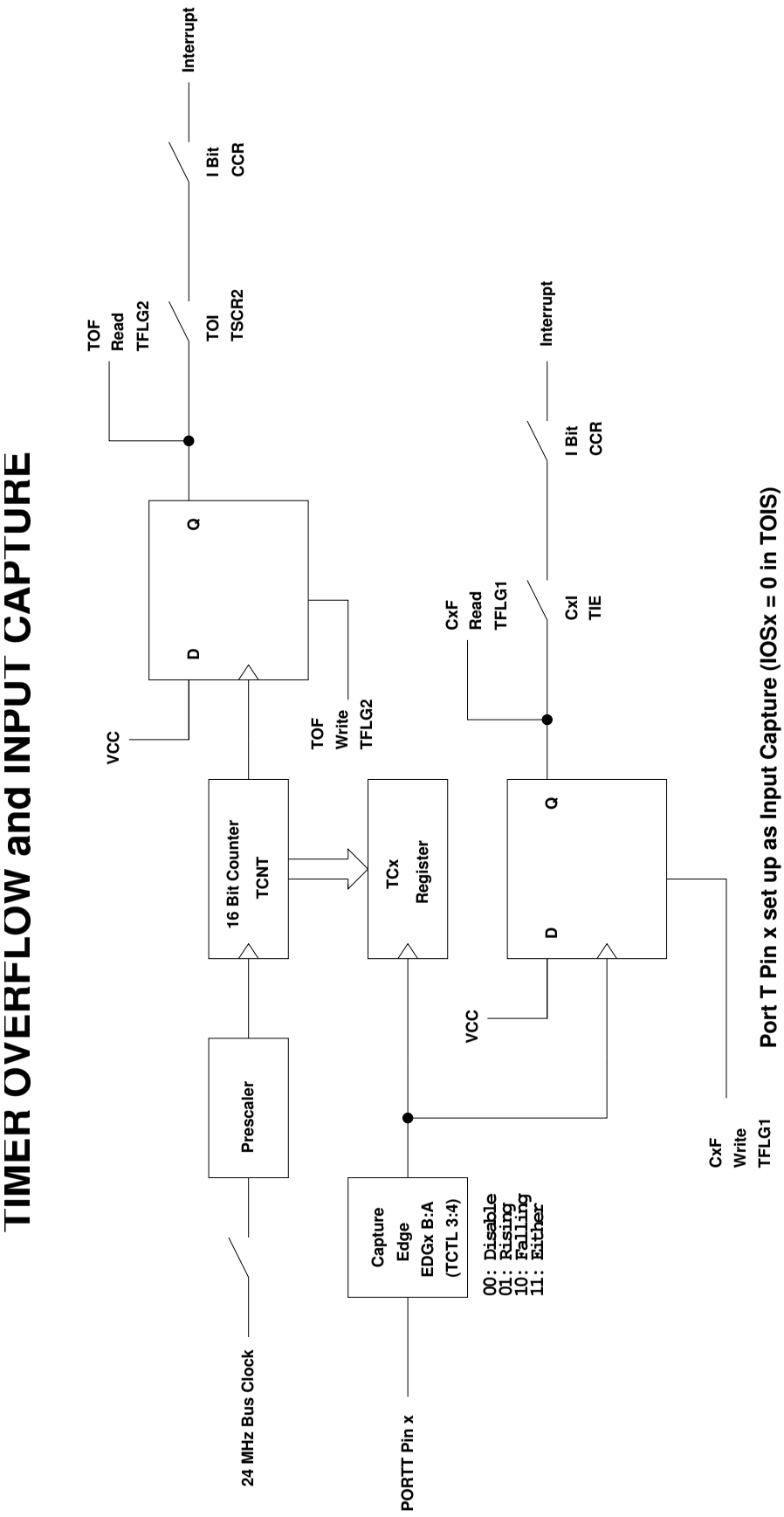
```
Hex:  0xf000,  Unsigned:  61440, Signed:  -4096
```

- **DO NOT USE `DB12FNP->printf` INSIDE AN INTERRUPT SERVICE ROUTINE**
- `DB12FNP->printf` takes a long time to complete, so you may miss other interrupts while the function executes
- Set a flag in the interrupt service routine to tell the main program that an interrupt has been received, and print the data out in the main program

Program to measure the time between two rising edges, and print out the result

```
/* Program to determine the time between two rising edges using
 * the MC9S12 Input Capture subsystem.
 *
 * The first edge occurs on Bit 1 of PTT
 * The second edge occurs on Bit 2 of PTT
 *
 * This program uses interrupts to determine when the two edges
 * have occurred.
 */

#include "hcs12.h"
#include "DBug12.h"
#include "vectors12.h"

#define enable() asm(" cli")
#define disable() asm(" sei")

#define TRUE  1
#define FALSE 0

/* Function Prototypes */
void INTERRUPT tic1_isr(void);
void INTERRUPT tic2_isr(void);

/* Declare things changed inside ISRs as volatile */
volatile unsigned int first, second, time, done;

main()
{
    disable();
    done = FALSE;

    /* Turn on timer subsystem */
    TSCR1 = BIT7;
    /* Set prescaler to 32 (87.38 ms), no TOF interrupt */
    TSCR2 = 0x05;
```

```
    /* Setup for IC1 */
    TIOS = TIOS & ~BIT1;             /* Configure PT1 as IC */
    TCTL4 = (TCTL4 | BIT2) & ~BIT3;  /* Capture Rising Edge */
    TFLG1 = BIT1;                    /* Clear IC1 Flag */
    /* Set interrupt vector for Timer Channel 1 */
    UserTimerCh1 = (short) &tic1_isr;
    TIE = TIE | BIT1;            /* Enable IC1 Interrupt */

    /* Setup for IC2 */
    TIOS = TIOS & ~BIT2;             /* Configure PT2 as IC */
    TCTL4 = (TCTL4 | BIT4) & ~BIT5;  /* Capture Rising Edge */
    TFLG1 = BIT2;                    /* Clear IC2 Flag */
    /* Set interrupt vector for Timer Channel 2 */
    UserTimerCh2 = (short) &tic2_isr;
    TIE = TIE | BIT2;            /* Enable IC2 Interrupt */

    /* Enable interrupts by clearing I bit of CCR */
    enable();

    while (!done)
    {
       asm(" wai");  /* Low power mode while waiting */
    }

    time = second - first;         /* Calculate total time */

    DB12FNP->printf("time = %d cycles\r\n",time)  /* print */;
}

void INTERRUPT tic1_isr(void)
{
    first = TC1;
    TFLG1 = BIT1;
}

void INTERRUPT tic2_isr(void)
{
    second = TC2;
    done = TRUE;
    TFLG1 = BIT2;
}
```

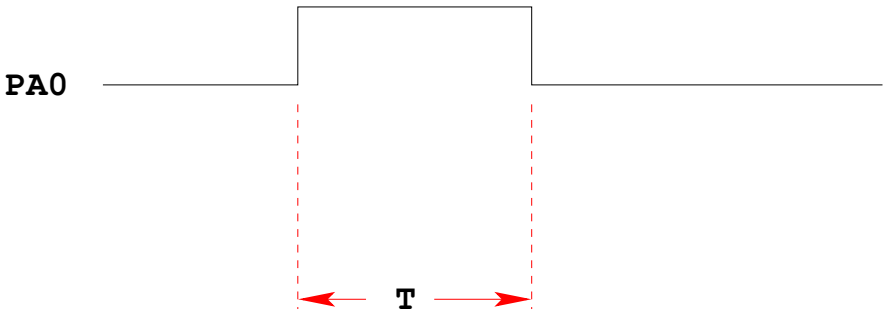

TIMER OVERFLOW and INPUT CAPTURE
Interrupt
I Bit
CCR
TOF
Read
TFLG2
TOI
TSCR2
Q
D
VCC
TOF
Write
TFLG2
16 Bit Counter
TCNT
TCx
Register
CxF
Read
TFLG1
CxI
TIE
Interrupt
I Bit
CCR
Q
D
VCC
Prescaler
Capture
Edge
EDGx B:A
(TCTL 3:4)
00: Disable
01: Rising
10: Falling
11: Either
24 MHz Bus Clock
PORTT Pin x
CxF
Write
TFLG1
Port T Pin x set up as Input Capture (IOSx = 0 in TOIS)

The MC9S12 Output Compare Function

;

Want event to happen at a certain time

Want to produce pulse pulse with width T



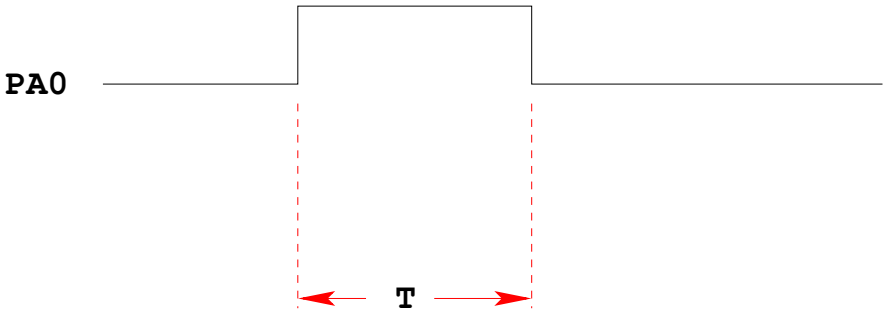
Wait until TCNT == 0x0000, then bring PA0 high

Wait until TCNT == T, then bring PA0 low

```
while (TCNT != 0x0000) ;
PORTA = PORTA | BIT0;
while (TCNT != T) ;
PORTA = PORTA & ~BIT0;
```

Want event to happen at a certain time

Want to produce pulse pulse with width T



Wait until TCNT == 0x0000, then bring PA0 high

Wait until TCNT == T, then bring PA0 low

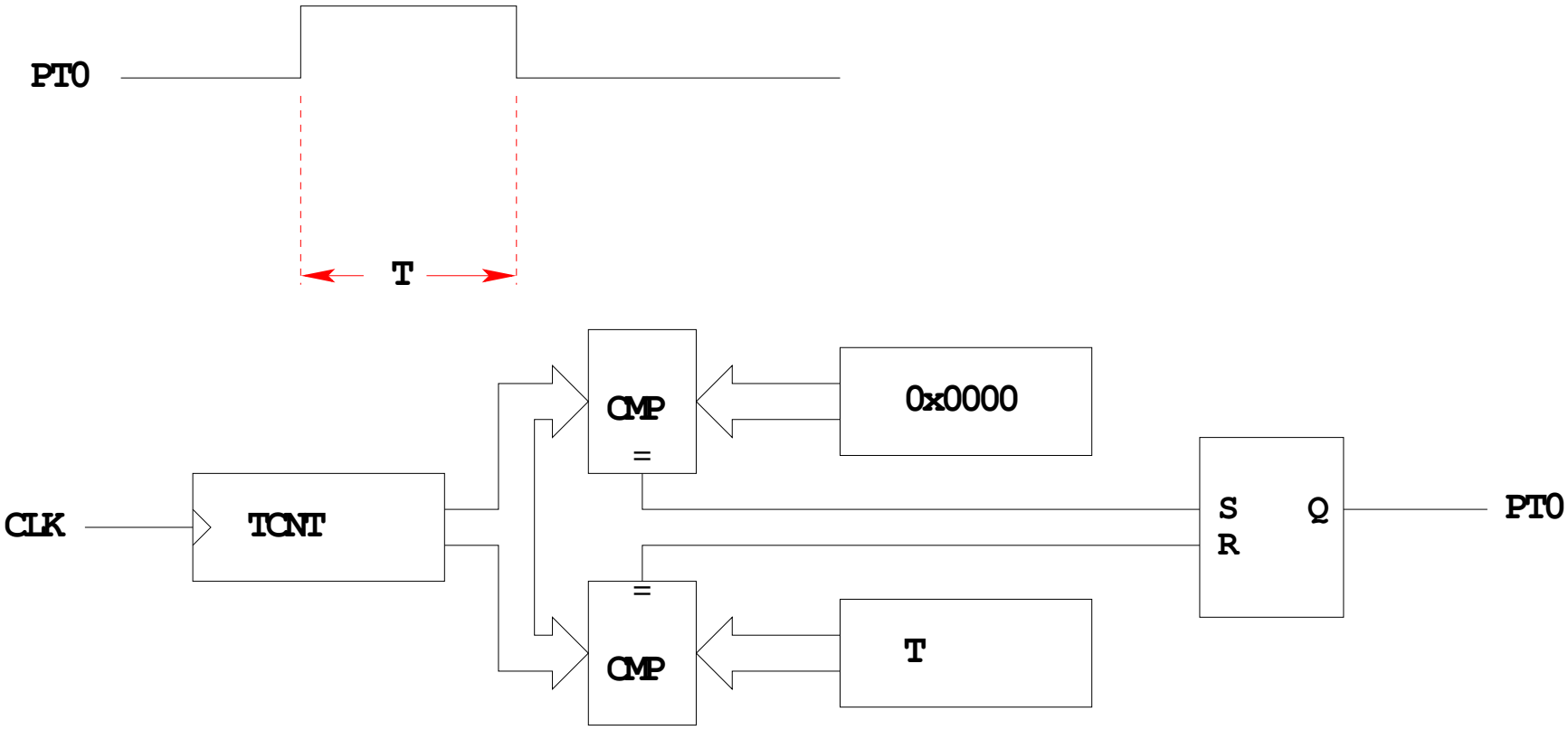
```
while (TCNT != 0x0000) ;
PORTA = PORTA | BIT0;
while (TCNT != T) ;
PORTA = PORTA & ~BIT0;
```

Problems:

1) May miss TCNT == 0x0000 or TCNT == T
2) Time not exact -- software delays
3) Cannot do anything else while waiting

Want event to happen at a certain time

Want to produce pulse pulse with width T

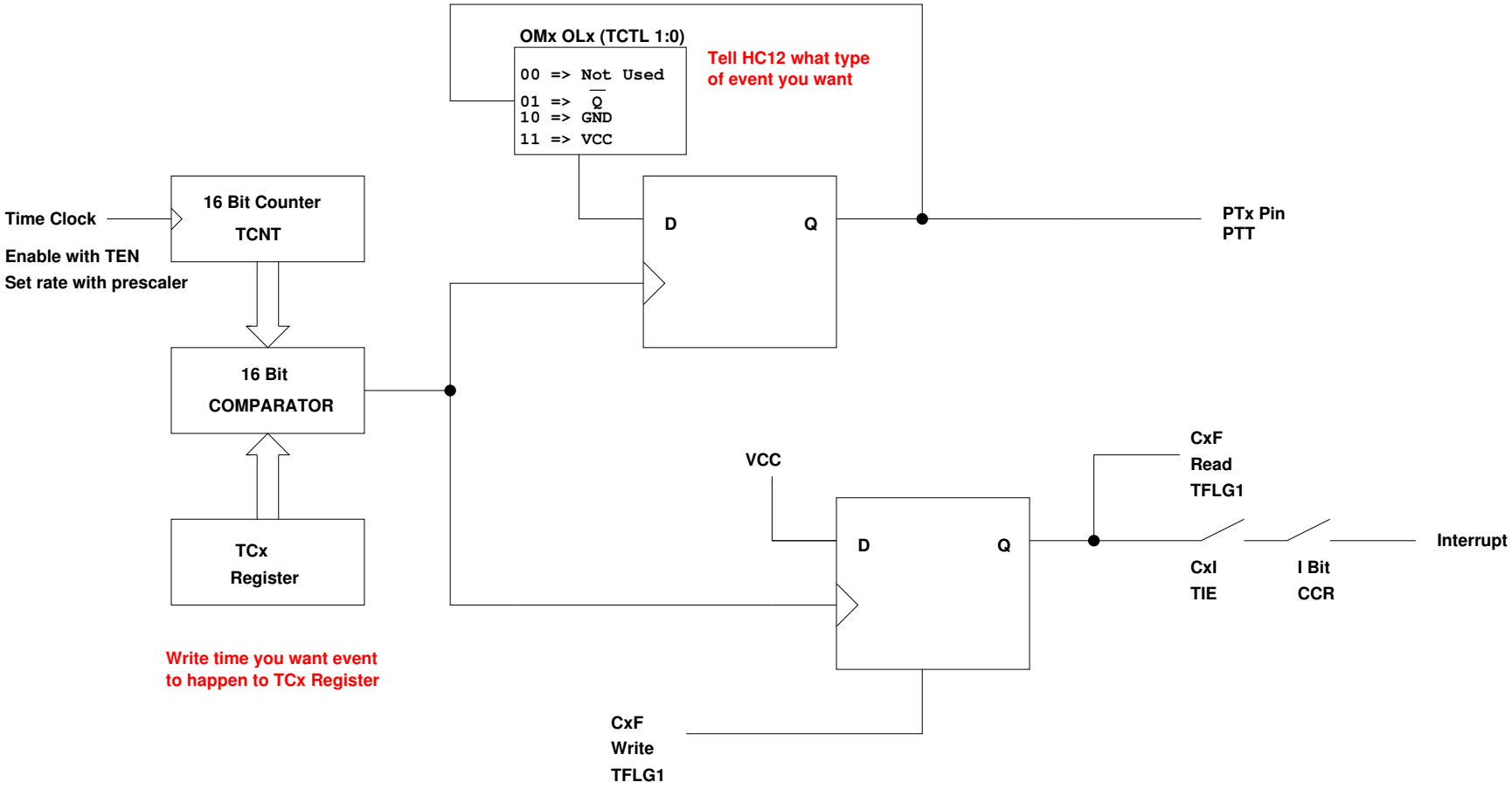


When TCNT == 0x0000, the output goes high
When TCNT == T, the output goes low

Now pulse is exaclty T cycles long

OUTPUT COMPARE PORT T 0–7

To use Output Compare, you must set IOSx to 1 in TIOS



The MC9S12 Output Compare Function

- The MC9S12 allows you to force an event to happen on any of the eight `PTT` pins
- An external event is a rising edge, a falling edge, or a toggle
- To use the Output Compare Function:
  - Enable the timer subsystem (set `TEN` bit of `TSCR1`)
  - Set the prescaler
  - Tell the MC9S12 that you want to use Bit x of `PTT` for output compare
  - Tell the MC9S12 what you want to do on Bit x of `PTT` (generate rising edge, falling edge, or toggle)
  - Tell the MC9S12 what time you want the event to occur
  - Tell the MC9S12 if you want an interrupt to be generated when the event is forced to occur
- There are some more complicated features of the output compare subsystem which are activated using registers `CFORC`, `OC7M`, `OC7D` and
- Writing a 1 to the corresponding bit of `CFORC` forces an output compare event to occur, the same as if a successful comparison has taken place (Section 8.6.5 of Huang).
- Using `OC7M` and `OC7D` allow Timer Channel 7 to control multiple output compare functions (Section 8.6.4 of Huang).
- Using `TTOV` allows you to toggle an output compare pin when `TCNT` overflows. This allows you to use the output compare system to generate pulse width modulated signals.
- We will not discuss these advanced features in this class.

Write a 1 to Bit 7 of TSCR1 to turn on timer

TEN	TSWAI	TSBCK	TFFCA					0x0046 TSCR1

To turn on the timer subsystem: TSCR1 = 0x80;

Set the prescaler in TSCR2

Make sure the overflow time is greater than the width of the pulse you want to generate

TOI	0	0	0	TCRE	**PR2**	**PR1**	**PR0**	0x004D TSCR2

PR2	PR1	PR0	Period (μs)	Overflow (ms)
0	0	0	0.0416	2.73
0	0	1	0.0833	5.46
0	1	0	0.1667	10.92
0	1	1	0.3333	21.84
1	0	0	0.6667	43.69
1	0	1	1.3333	86.38
1	1	0	2.6667	174.76
1	1	1	5.3333	349.53

To have overflow rate of 21.84 ms:

TSCR2 = 0x03;

Write a 1 to the bits of TIOS to make those pins output compare

IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0	0x0080 TIOS

To make Pin 4 an output compare pin: TIOS = TIOS | 0X10;

Write to TCTL1 and TCTL2 to choose action to take

OM7	OL7	OM6	OL6	OM5	OL5	OM4	OL4	0x0048 TCTL1
OM3	OL3	OM2	OL2	OM1	OL1	OM0	OL0	0x0049 TCTL2

OMn	OLn	Configuration
0	0	Disconnected
0	1	Toggle
1	0	Clear
1	1	Set

To have Pin 4 toggle on compare:

TCTL1 = (TCTL1 | BIT0) & ~BIT1;

Write time you want event to occur to TCn register.

To have event occur on Pin 4 when TCNT == 0x0000: TC4 = 0x0000;

To have next event occur T cycles after last event, add T to TCn.

To have next event occur on Pin 4 500 cycles later: TC4 = TC4 + 500;

When TCNT == TCn, the specified action will occur, and flag CFn will be set.

To clear the flag, write a 1 to the bit you want to clear (0 to all others)

CF7	CF6	CF5	CF4	CF3	CF2	CF1	CF0	0x004E TFLG1

To wait until TCNT == TC4: while ((TFLG1 & BIT4) == 0) ;

To clear flag bit for Pin 4: TFLG1 = BIT4;

To enable interrupt when compare occurs, set corresponding bit in TIE register

C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I	0x004C TIE

To enable interrupt when TCNT == TC4: TIE = TIE | BIT4;

USING OUTPUT COMPARE ON THE MC9S12

1. In the main program:
   (a) Turn on timer subsystem (TSCR1 reg)
   (b) Set prescaler (TSCR2 reg)
   (c) Set up PTx as OC (TIOS reg)
   (d) Set action on compare (TCTL 1-2 regs, OMx OLx bits)
   (e) Clear Flag (TFLG1 reg)
   (f) Enable int (TIE reg)
2. In interrupt service routine
   (a) Set time for next action to occur (write TCx reg)
      - For periodic events add time to TCx register
   (b) Clear flag (TFLG1 reg)

Setting and Clearing Bits in the Timer Subsystem

- Registers in the timer subsystem control multiple timer channels.
  - Usually, you want to use ANDS and ORS to change only that channel you are working on.
  - For example, to make Channel 2 an output compare, and set it to toggle on compare, do this:

```
TIOS = TIOS | BIT2;              /* Configure PT2 as Output Compare */
TCTL2 = (TCTL2 | BIT4) & ~BIT5;  /* Set up PT2 to toggle on compare */
```

  - Do not do this:

```
TIOS = BIT2;              /* Configure PT2 as Output Compare */
TCTL2 = BIT4);            /* Set up PT2 to toggle on compare */
```

    This would set up Channel 2 as an output compare, toggle on successful compare. However, it will force all the other channels to input capture – this may not be what you want to do.

- **To clear a flag bit, do not use ORs!**
  - To clear Timer Channel 2 flag, do the following:

```
TFLG1 = BIT2;
```

    This will clear Timer Channel 2 flag, and leave all other flags unaffected.
  - Do not do this:

```
TFLG1 = TFLG1 | BIT2;   /* DO NOT DO THIS */
```

    This will clear Timer Channel 2 flag, but will also clear any other flag which is set.
    Suppose, for example, Timer Channel 2 and Timer Channel 3 flags are both set at the same time, so `TFLG1` register is `0x0C`. You want to deal the Timer Channel 2 first and Timer Channel 3 afterwards.
    The command:

```
TFLG1 = TFLG1 | BIT2;   /* DO NOT DO THIS */
```

    will read TFLG1, which will return an `0x0C`. ORing that with a `0x04` (`BIT2`) will result in an `0x0C`. Writing that back to `TFLG1` will clear Timer Channel 2 flag and Timer Channel 3 flag. Now Timer Channel 3 flag is cleared, so you will never deal with the event which set Timer Channel 3 flag.

```
/*
 * Program to generate square wave on PT2
 * Frequency of square wave is 500 Hz
 * Period of square wave is 2 ms
 * Set prescale to give 0.667 us cycle
 * 2 ms is 3,000 cycles of 1.5 MHz clock
 *
 */
#include "hcs12.h"
#include "vectors12.h"

#define PERIOD      3000
#define HALF_PERIOD (PERIOD/2)

#define disable()  asm(" sei")
#define enable()  asm(" cli")

void INTERRUPT toc2_isr(void);

main()
{
    disable();

    TSCR1 = BIT7;                      /* Turn on timer subsystem */
    TSCR2 = 0x04;                      /* Set prescaler to 16 (0.666 us) */

    TIOS = TIOS | BIT2;               /* Configure PT2 as Output Compare */
    TCTL2 = (TCTL2 | BIT4) & ~BIT5;  /* Set up PT2 to toggle on compare */
    TFLG1 = BIT2;                     /* Clear Channel 2 flag */
    /* Set interrupt vector for Timer Channel 2 */
    UserTimerCh2 = (unsigned short) &toc2_isr;
    TIE = TIE | BIT2;                 /* Enable interrupt on Channel 2 */

    enable();

    while (1)
    {
        asm("wai");
    }
}

void INTERRUPT toc2_isr(void)
{
    TC2 = TC2 + HALF_PERIOD;
    TFLG1 = BIT2;
}
```