- **The MC9S12 Timer Input Capture Function**
  - Capturing the time of an external event
  - The MC9S12 Input Capture Function
  - Registers used to enable the Input Capture Function
  - Using the MC9S12 Input Capture Function
  - A program to use the MC9S12 Input Capture in polling mode
  - Using the Keyword volatile in C
  - A program to use the MC9S12 Input Capture in interrupt mode

Capturing the Time of an External Event

• One way to determine the time of an external event is to wait for the event to occur, the read the TCNT register:

• For example, to determine the time a signal on Bit 0 of PORTB changes from a high to a low:

```
while ((PORTB & BIT0) != 0) ;   /* Wait while Bit 0 high */
time = TCNT;                    /* Read time after goes low */
```

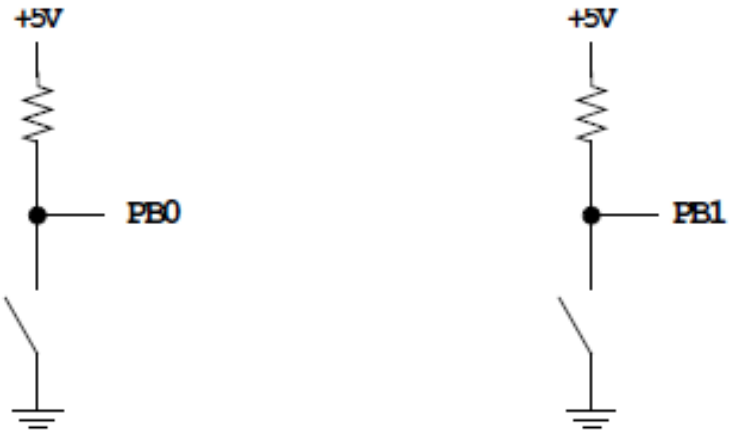
• Two problems with this:

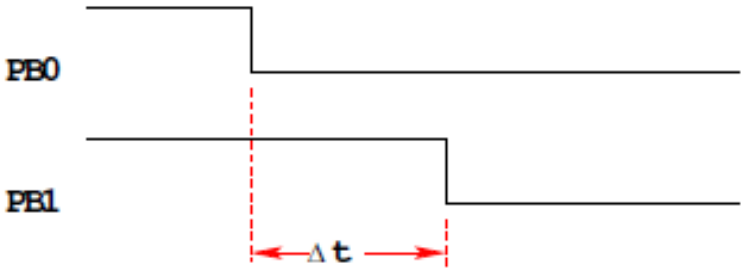
1. Cannot do anything else while waiting
2. Do not get exact time because of delays in software

• To solve problems use hardware which latches TCNT when event occurs, and generates an interrupt.

• Such hardware is built into the MC9S12 — called the **Input Capture System**

Measure the time between two events





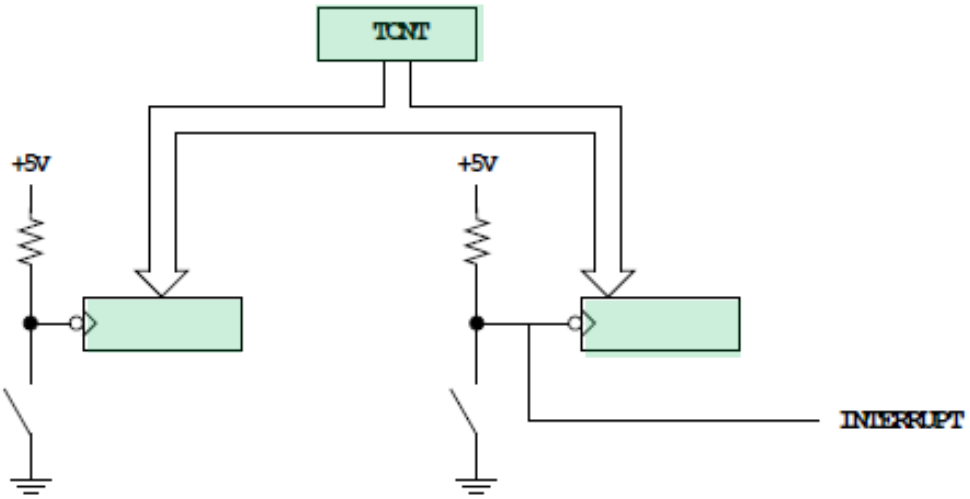
How to measure Δt?

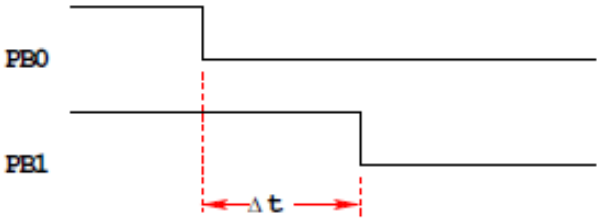
Wait until signal goes low, then measure TCNT

```
while ((PORTB & BIT0) == BIT0) ;
start = TCNT;
while ((PORTB & BIT0) == BIT1) ;
end = TCNT;
dt = end - start;
```

Problems: 1) May not get very accurate time
2) Can't do anything while waiting for signal level to change.

Measure the time between two events





Solution: Latch TCNT on falling edge of signal

Read latched values anytime later and get exact value

Can have MC9S12 generate interrupt when event occurs, so can do other things while waiting

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

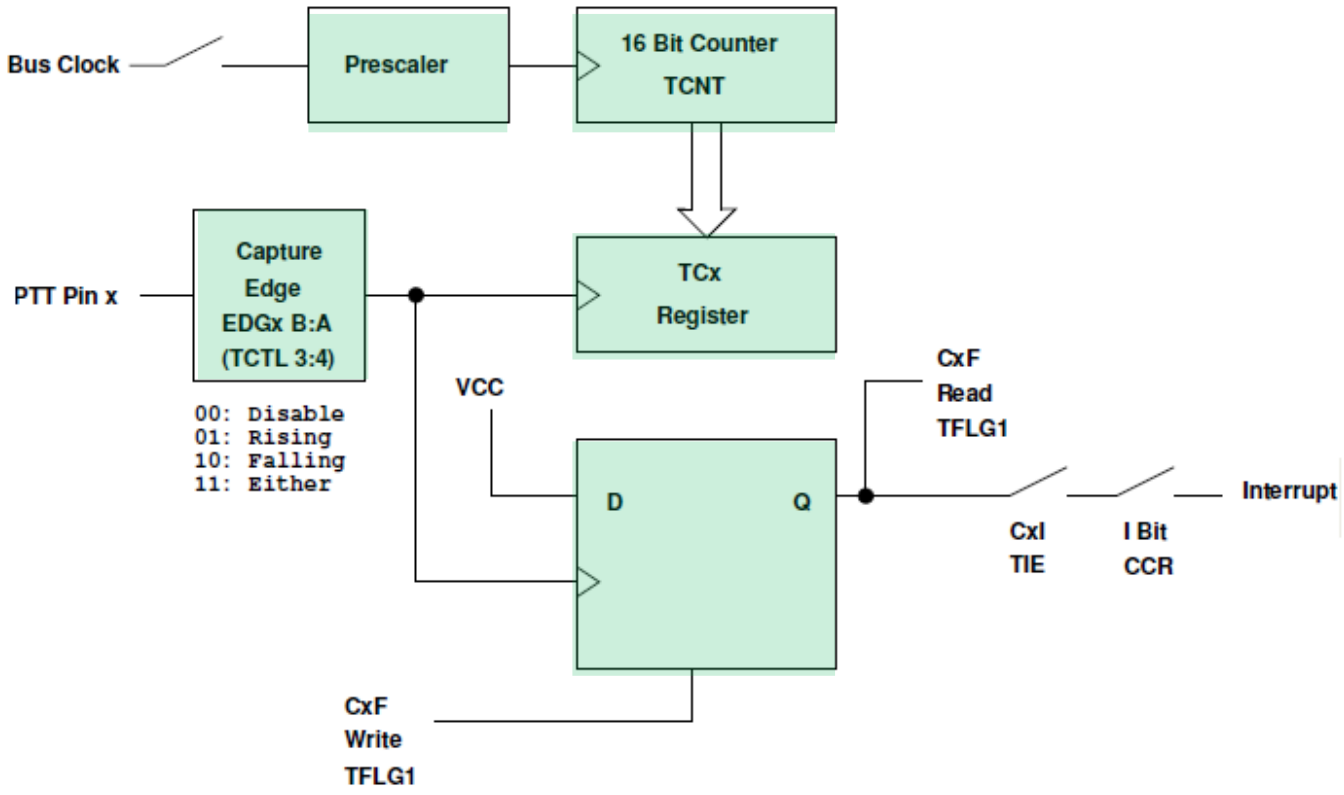
– Tell the MC9S12 which edge (rising, falling, or either) you want to capture

– Tell the MC9S12 if you want an interrupt to be generated when the capture occurs

A Simplified Block Diagram of the MC9S12 Input Capture Subsystem

Input Capture

Port T Pin x set up as Input Capture (IOSx = 0 in TOIS)



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 subsystems: 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 (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 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 <hidef.h>              /* common defines and macros */
#include "derivative.h"         /* derivative-specific definitions */
#include <stdio.h>
#include <termio.h>

unsigned int first, second, time;

void main(void)
{
        TSCR1 = 0x80;                   /* Turn on timer subsystem */
        TSCR2 = 0x05;                   /* Set prescaler for divide by 32 */
                                        /* 87.38 ms overflow time */
        /* Setup for IC1 */
        TIOS = TIOS & ~0x02;            /* IOC1 set for Input Capture */
        TCTL4 = (TCTL4 | 0x04) & ~0x08; /* Capture Rising Edge */
        TFLG1 = 0x02;                   /* Clear IC1 Flag */

        /* Setup for IC2 */
        TIOS = TIOS & ~0x04;            /* IOC2 set for Input Capture */
        TCTL4 = (TCTL4 | 0x10) & ~0x20; /* Capture Rising Edge */
        TFLG1 = 0x04;                   /* Clear IC2 Flag */

        while ((TFLG1 & 0x02) == 0) ; /* Wait for 1st rising edge; */
        first = TC1;                    /* Read time of 1st edge; */

        while ((TFLG1 & 0x04) == 0) ; /* Wait for 2nd rising edge; */
        second = TC2;                   /* Read time of 2nd edge; */

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

Using the Keyword volatile in C

• Consider the following code fragment, which waits until an event occurs on Pin 2 of PTT:

```
#define TRUE 1
#define FALSE 0

#include <hidef.h>          /* common defines and macros */
#include "derivative.h"     /* derivative-specific definitions */
#include "vectors12.h"

#define enable() _ _asm(cli)
#define disable() _ _asm(sei)

interrupt void tic2_isr(void);
unsigned int time, done;

void main(void)
{
        disable();

        /* Code to set up Input Capture 2 */
        TFLG1 = 0x04;               /* Clear CF2 */
        UserTimerCh2 = (short) &tic2_isr; /* Set interrupt vector */
        enable();                   /* Enable Interrupts */
        done = FALSE;
        while (!done) ;
        _ _asm(swi);
}
```

```
interrupt void tic2_isr(void)
{
        time = TC2;
        TFLG1 = 0x04;
        done = TRUE;
}
```

• An optimizing compiler knows that done will not change in the main() function. It may decide that, since *done* is *FALSE* in the main() function, and nothing in the main() function changes the value of *done*, then done will always be *FALSE*, so there is no need to check if it will ever become *TRUE*.

• An optimizing compiler might change the line

while (!done) ;

to

while (TRUE) ;

and the program will never get beyond that line.

• By declaring done to be volatile, you tell the compiler that the value of done might change somewhere else other than in the main() function (such as in an interrupt service routine), and the compiler should not optimize on the done variable.

volatile unsigned int time, done;

• If a variable can change its value outside the normal flow of the program (i.e., inside an interrupt service routine), declare the variable to be of type volatile.

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

```
#include <hidef.h> /* common defines and macros */
#include "derivative.h" /* derivative-specific definitions */
#include <stdio.h>
#include <termio.h>
#include "vectors12.h"

#define enable() _ _asm(cli)
#define disable() _ _asm(sei)

#define TRUE 1
#define FALSE 0

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

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

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

        /* Turn on timer subsystem */
        TSCR1 = 0x80;

        /* Set prescaler to 32 (87.38 ms), no TOF interrupt */
        TSCR2 = 0x05;
```

```
        /* Setup for IC1 */
        TIOS = TIOS & ~0x02;                    /* Configure PT1 as IC */
        TCTL4 = (TCTL4 | 0x04) & ~0x08;         /* Capture Rising Edge */
        TFLG1 = 0x02;                           /* Clear IC1 Flag */

        /* Set interrupt vector for Timer Channel 1 */
        UserTimerCh1 = (short) &tic1_isr;
        TIE = TIE | 0x02;                       /* Enable IC1 Interrupt */

        /* Setup for IC2 */
        TIOS = TIOS & ~0x04;                     /* Configure PT2 as IC */
        TCTL4 = (TCTL4 | 0x10) & ~0x20;          /* Capture Rising Edge */
        TFLG1 = 0x04;                           /* Clear IC2 Flag */

        /* Set interrupt vector for Timer Channel 2 */
        UserTimerCh2 = (short) &tic2_isr;
        TIE = TIE | 0x04;                       /* 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 */
        printf("time = %d cycles\r\n",time);    /* print */;
}

interrupt void tic1_isr(void)
{
        first = TC1;
        TFLG1 = 0x02;
}
```

```
interrupt void tic2_isr(void)
{
        second = TC2;
        done = TRUE;
        TFLG1 = 0x04;
}
```

Letting the MC9S12 convert cycles to time

You can let the MC9S12 convert clock cycles to time, and display the time between two edges on the seven-segment LEDs:

- Need to divide the number of clock cycles by (24,000,000/prescaler)
- Easiest to do this using floating point numbers
- Need to convert back to fixed-point number to display on seven-segment LEDs
- Can display *number of milliseconds* on seven segment LEDs
- This will display time in hexadecimal. It makes for sense to humans to display time in decimal – convert hexadecimal to BCD
- To use floating-point numbers, select ”float is IEEE32” when you create a CodeWarrior project.

```
#define clock_freq 24000000.0
#define prescaler ( (float) (1 << (TSCR2&0x07)))

unsigned int t_first, t_second, dt;
unsigned int value;          // Value to display on seven-segment LEDs

...

        dt = ((float)(t_second - t_first))*prescaler)/clock_freq * 1000.0;
        value = hex2bcd(dt);

...

unsigned int hex2bcd(unsigned int x)
{
        unsigned int d3,d2,d1,d0;
        if (x > 9999) return 0xFFFF;
        d3 = x/1000;
        x = x-d3*1000;
        d2 = x/100;
        x = x-d2*100;
        d1 = x/10;
        x = x-d1*10;
        d0 = x;
        return d3*16*16*16 + d2*16*16 + d1*16 + d0;
}
```