What happens when an HC12 gets in unmasked interrupt:

1. Completes current instruction
2. Clears instruction queue
3. Calculates return address
4. Stacks return address and contents of CPU registers
5. Sets I bit of CCR
6. Sets X bit of CCR if an XIRQ interrupt is pending
7. Fetches interrupt vector for the highest-priority interrupt which is pending
8. Executes ISR at the location of the interrupt vector

What happens when an HC12 exits an ISR with the RTI instruction:

1. If no other interrupt pending,
   (a) HC12 recovers stacked registers
   (b) Execution resumes at the return address
2. If another interrupt pending
   (a) HC12 recovers stacked registers
   (b) Subtracts 9 from SP
   (c) Sets I bit of CCR
   (d) Sets X bit of CCR if an XIRQ interrupt is pending
   (e) Fetches interrupt vector for the highest-priority interrupt which is pending
   (f) Executes ISR at the location of the interrupt vector

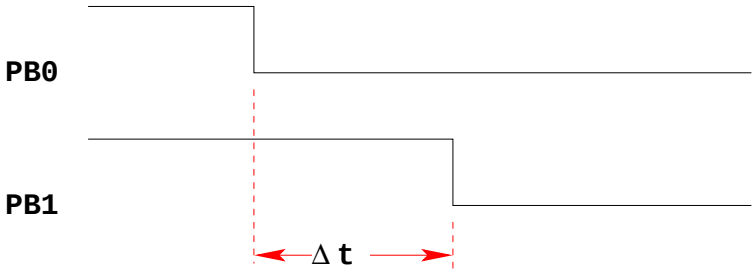
Capturing the Time of an External Event

- 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 & 0x01) != 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 HC12 — called the Input Capture System

Measure the time between two events

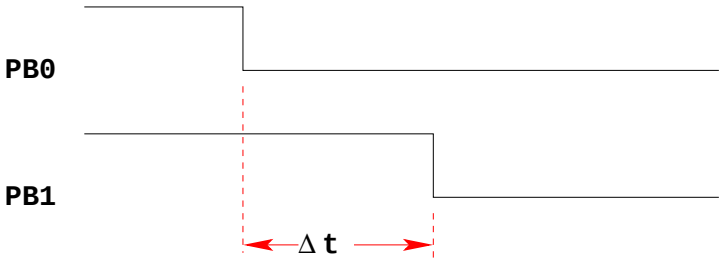


How to measure Δt?

Wait until signal goes low, then measure TCNT

```
while ((PORTB & 0x01) == 0x01) ;
start = TCNT;
while ((PORTB & 0x02) == 0x02) ;
end = TCNT;
dt = end − start;
```

Measure the time between two events



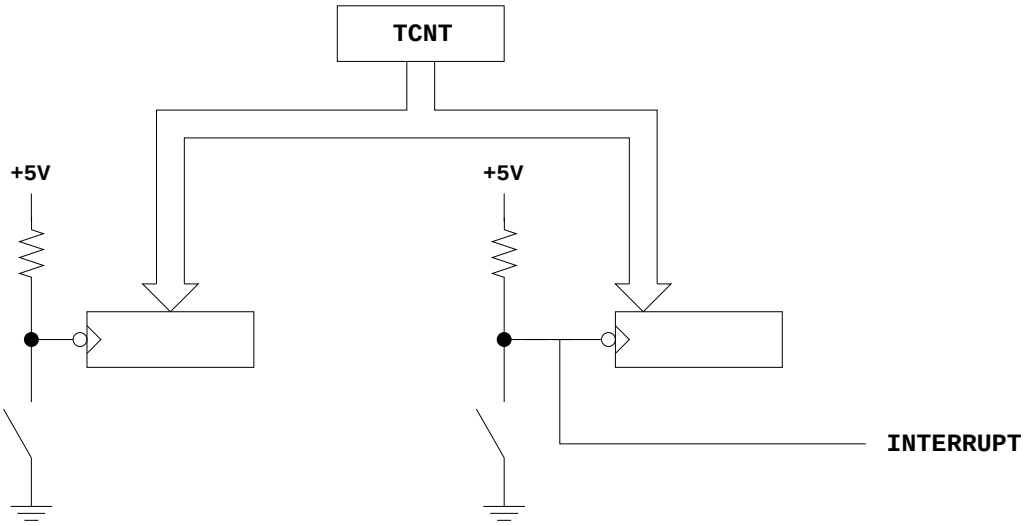
How to measure Δt?

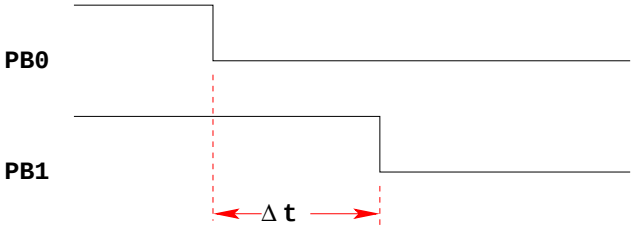
Wait until signal goes low, then measure TCNT

```
while ((PORTB & 0x01) == 0x01) ;
start = TCNT;
while ((PORTB & 0x02) == 0x02) ;
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 when interrupt occurs

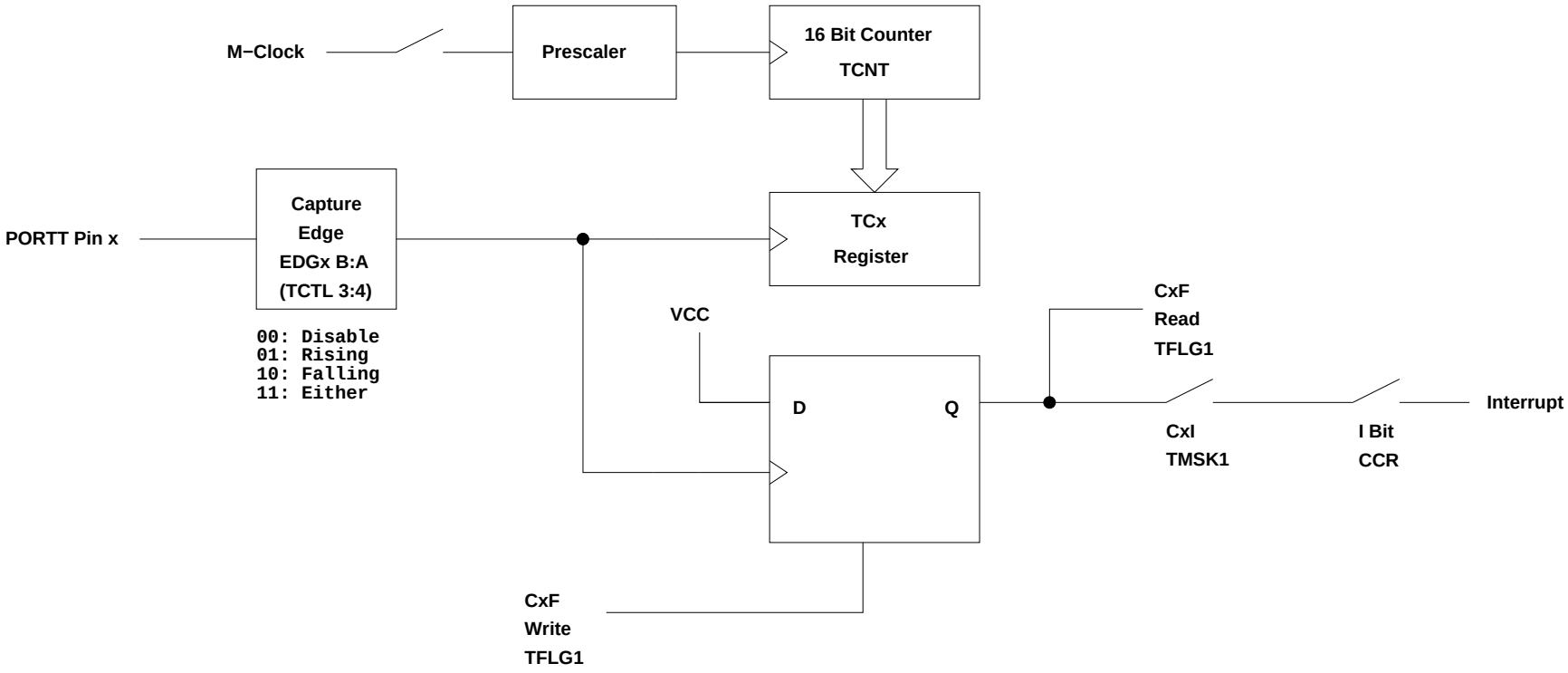
The HC12 Input Capture Function

- The HC12 allows you to capture the time an external event occurs on any of the eight PORTT 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 TSCR)
  - Set the prescaler
  - Tell the HC12 that you want to use a particular pin of PORTT for input capture
  - Tell the HC12 which edge (rising, falling, or either) you want to capture
  - Tell the HC12 if you want an interrupt to be generated when the cature occurs

A Simplified Block Diagram of the HC12 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 TSCR to turn on timer

TEN	TSWAI	TSBCK	TFFCA					**0x0086 TSCR**

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

Set the prescaler in TMSK2

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

TOI	0	PUPT	RDPT	TCRE	**PR2**	**PR1**	**PR0**	**0x008D TMSK2**

PR2	PR1	PR0	Period (μs)	Overflow (ms)
0	0	0	0.125	8.192
0	0	1	0.250	16.384
0	1	0	0.500	32.768
0	1	1	1.000	65.536
1	0	0	2.000	131.072
1	0	1	4.000	262.144
1	1	0	------	------
1	1	1	------	------

To have overflow rate of 65.536 ms:

TMSK2 = 0x03;

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

IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0	0x0080 TIOS

To make Pin 3 an input capture pin: TIOS = TIOS & ~0X08;

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

EDG7B	EDG7A	EDG6B	EDG6A	EDG5B	EDG5A	EDG4B	EDG4A	0x008A TCTL3

EDG3B	EDG3A	EDG2B	EDG2A	EDG1B	EDG1A	EDG0B	EDG0A	0x008B 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 | 0x40) & ~0x80;

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 & 0x08) == 0) ;

To clear flag bit for Pin 3: TFLG1 = 0x08;

To enable interrupt when specified edge occurs, set corresponding bit in TMSK1 register

C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I	0x008C TMSK1

To enable interrupt on Pin 3: TMSK1 = TMSK1 | 0x08;

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 HC12

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 TSCR reg)
2. Set prescaler (TMSK2 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 (1 -> bit x of TMSK1 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 *
 * HC12 Input Capture subsystem
 */

#include "hc12b32.h"

unsigned int first, second, time;

main()
{
    TSCR = 0x80;                /* Turn on timer subsystem */
    TMSK2 = 0x05;               /* Set prescaler for divide by 32 */

    /* 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 */
}
```

Using the Keyword `volatile` in C

- Consider the following code fragment, which waits until an event occurs on Pin 2 of `PORTT`:

```
#define TRUE  1
#define FALSE 0

unsigned int time, done;

main()
{

    Code to set up Input Capture 2

    TFLG2 = 0x04;   /* Clear CF2 */
    enable();       /* Enable Interrupts */

    done = FALSE;

    while (!done) ;
}

@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 comiler 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`.

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 the Application Note **Using the Callable Routines in D-Bug12**. 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:

```
DBug12FNP->putchar(c);
```

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

```
#include "DBug12.h"

void main(void)
{
    DBug12FNP->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;

    DBug12FNP->printf("Hex:  0x%4x, Unsigned:  %u, Signed:  %d\n\r",i,i,i);
}
```

The output of the above program will be:

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

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 HC12 Input Capture subsystem.
 *
 * The first edge occurs on Bit 1 of PORTT
 * The second edge occurs on Bit 2 of PORTT
 *
 * This program uses interrupts to determine when the two edges
 * have occurred.
 */

#include "hc12b32.h"
#include "DBug12.h"

#define TRUE  1
#define FALSE 0

volatile unsigned int first, second, time, done;

main()
{
    done = FALSE;

    /* Turn on timer subsystem */
    TSCR = 0x80;
    /* Set prescaler to 32, no TOF interrupt */
    TMSK2 = 0x05;

    /* Setup for IC1 */
    TIOS = TIOS & ~0x02;             /* Configure PT1 as IC */
    TCTL4 = (TCTL4 | 0x04) & ~0x08;  /* Capture Rising Edge */
    TFLG1 = 0x02;                    /* Clear IC1 Flag */
    TMSK1 = TMSK1 | 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 */
    TMSK1 = TMSK1 | 0x04;            /* Enable IC2 Interrupt */
```

```
    enable();

    while (!done) ;

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

    DBug12FNP->printf("time = %d\r\n",time)  /* print */;
}

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

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

```
/*      INTERRUPT VECTORS TABLE 68HC12
 */
void tic1_isr();
void tic2_isr();

void (* const _vectab[])() = {                  /* 0x0B10 */
        0,              /* BDLC           */
        0,              /* ATD            */
        0,              /* reserved       */
        0,              /* SCI0           */
        0,              /* SPI            */
        0,              /* Pulse acc input */
        0,              /* Pulse acc overf */
        0,              /* Timer overf    */
        0,              /* Timer channel 7 */
        0,              /* Timer channel 6 */
        0,              /* Timer channel 5 */
        0,              /* Timer channel 4 */
        0,              /* Timer channel 3 */
        tic2_isr,       /* Timer channel 2 */
        tic1_isr,       /* Timer channel 1 */
        0,              /* Timer channel 0 */
        0,              /* Real time      */
        0,              /* IRQ            */
        0,              /* XIRQ           */
        0,              /* SWI            */
        0,              /* illegal        */
        0,              /* cop fail       */
        0,              /* cop clock fail  */
        (void *)0xff80, /* RESET          */
        };
```