- **The 9S12 Input Capture Function**
- Huang Sections 8.1-8.5
- ECT_16B8C Block User Guide
  - Interrupts on the 9S12
  - Capturing the time of an external event
  - The 9S12 Input Capture Function
  - Registers used to enable the Input Capture Function
  - Using the 9S12 Input Capture Function
  - A program to use the 9S12 Input Capture in polling mode
  - Using the Keyword volatile in C
  - Using D-Bug12 Routines to Print Information to the Terminal
  - A program to use the 9S12 Input Capture in interrupt mode

Using Input Capture on the 9S12

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)

Table 3-3 Edge Detector Circuit Configuration

EDGxB	EDGxA	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)

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 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 *
* 9S12 Input Capture subsystem
*/
#include "hcs12.h"
#include "DBug12.h"

unsigned int first, second, time;

main()
{
      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 */

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

      /* Capture 2nd rising edge */
      while ((TFLG1 & 0x04) == 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 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
#include "hcs12.h"
#include "DBug12.h"
#include "vectors12.h"
#define enable() asm(" cli")

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

main()
{
       /* 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");
}

void INTERRUPT 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.

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 ordinary 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:

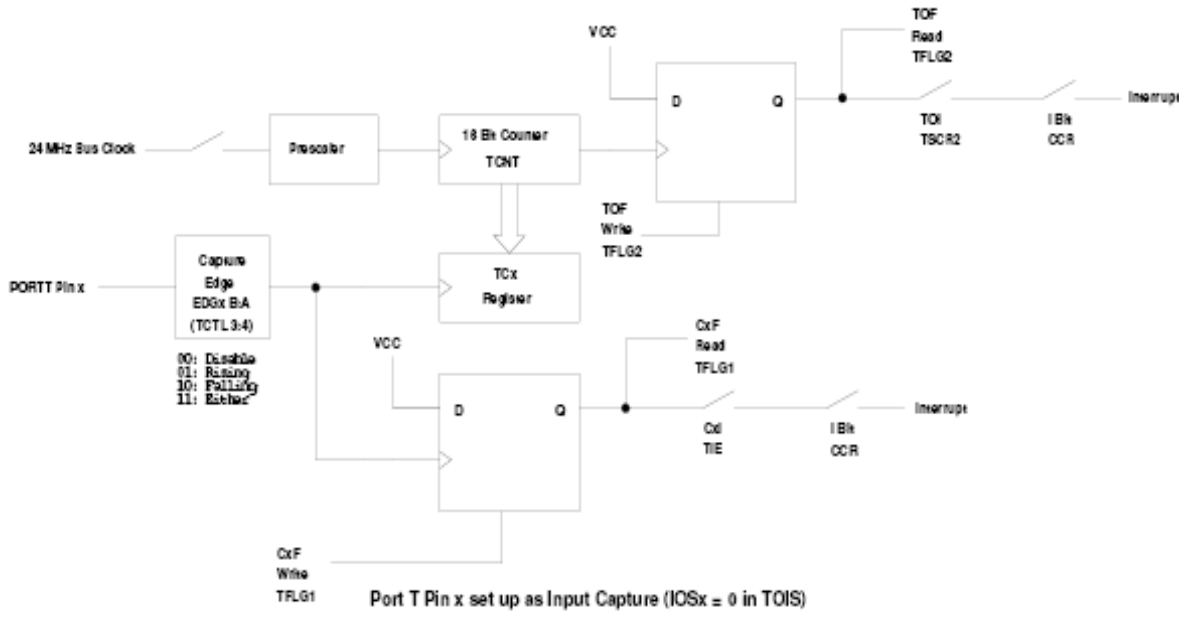
```
Hex: 0xf000, Unsigned: 61440, Signed: -4096
```

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 9S12 Input
// Capture * subsystem.
//  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 stop() asm("swi");

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

/* Declare things changed inside ISR as volatile */
volatile unsigned int first, second, time;
volatile unsigned char count=0;

main()
{

      /* Turn on timer subsystem */
      /* Set prescaler to 350 ms */
      TSCR1 = 0x80;
      TSCR2 = 0x07;

      /* 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 */

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

```
      while (count < 2)
      {
            asm("wai");                 /* Low power mode while waiting */
      }
      time = first - second;                /* Calculate total time */
      DB12FNP->printf("delta time = %d cycles\r\n",time);            /* print dt */;
      stop();
}

void INTERRUPT tic1_isr(void)
{
     if(count == 0)
     {
        first = TC1;
        count = 1;
        DB12FNP->printf("first = %u \r\n",first);
     }
     else
     {
        second = TC1;
        count = 2;
        DB12FNP->printf("second = %u cycles\r\n",second);
     }
     TFLG1 = 0x02;
}
```

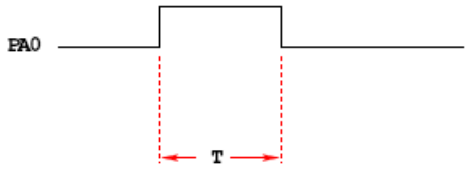
Timer Overflow and Input Capture



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

The HCS12 Output Compare Function

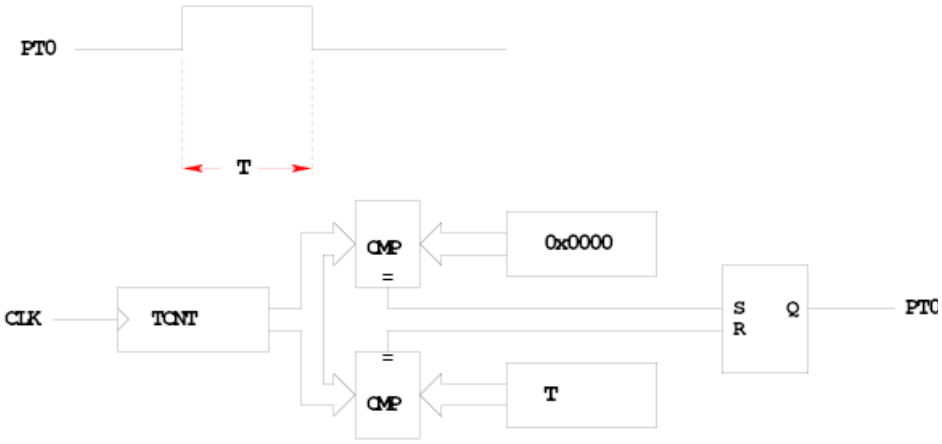
Want event to happen at a certain time
Want to produce pulse with width T



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

```
while (TCNT != 0x0000) ;
PORTA = PORTA | 0x01;
while (TCNT != T) ;
PORTA = PORTA & ~0x01;
```

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

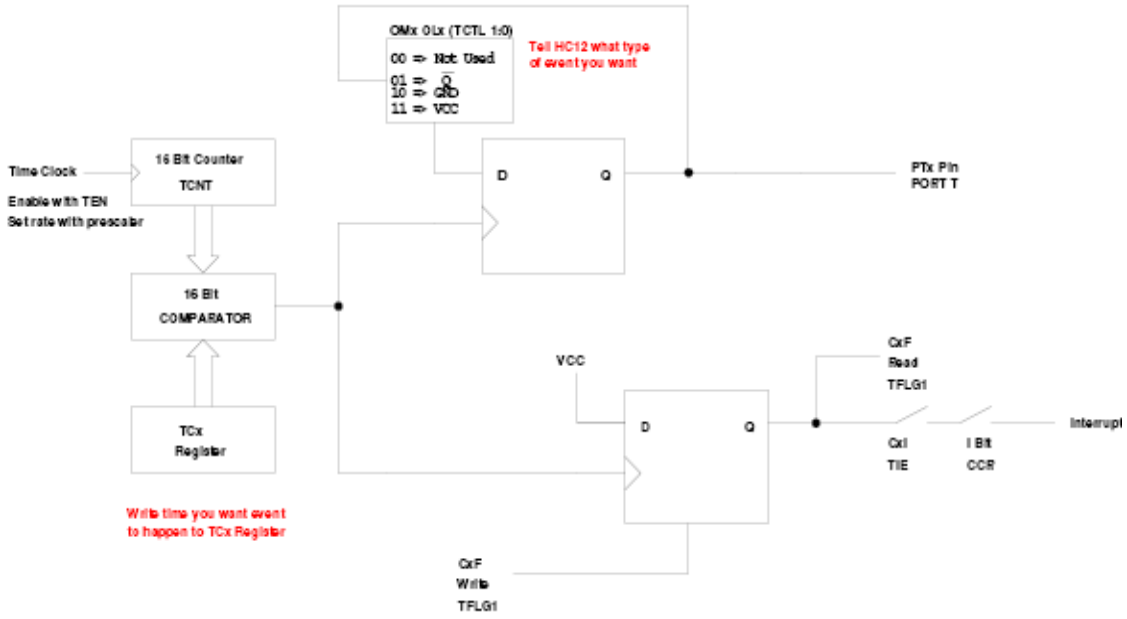


When TCNT == 0x0000, the output goes high
When TCNT == T, the output goes low

Now pulse is exactly T cycles long

Output Compare PORT T 0−7

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



The HCS12 Output Compare Function

• The HCS12 allows you to force an event to happen on any of the eight PORTT 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 HCS12 that you want to use Bit x of PORTT for output compare
– Tell the HCS12 what you want to do on Bit x of PORTT (generate rising edge, falling edge, or toggle)
– Tell the HCS12 what time you want the event to occur
– Tell the HCS12 if you want an interrupt to be generated when the event is forced to occur

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

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

= Unimplemented or Reserved

Figure 3-6 Timer System Control Register 1 (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

	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)

To have overflow rate of 21.84 ms:
TSCR2 = 0x03;

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

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

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

To make Pin 4 an input capture pin: TIOS = TIOS | 0X10;

Write to TCTL1 and TCTL2 to choose the action to take

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

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

OMx	OLx	Action
0	0	Timer disconnected from output pin logic
0	1	Toggle OCx output line
1	0	Clear OCx output line to zero
1	1	Set OCx output line to one

To have Pin 4 toggle on compare:
TCTL1 = (TCTL1 | 0x01) & ~0x02;

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)

	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)

To wait until TCNT == TC4: while ((TFLG1 & 0x10) == 0);
To clear flag bit for Pin 4: TFLG1 = 0x10;

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

	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)

To enable interrupt when TCNT == TC4: TIE = TIE | 0x10;