- **More on Programming the 9S12 in C**
- Huang Sections 5.2 through 5.4
- **Introduction to the 9S12 Hardware Subsystems**
- Huang Sections 8.2-8.6
- ECT_16B8C Block User Guide
- A summary of 9S12 hardware subsystems
- Introduction to the 9S12 Timer subsystem
    - The 9S12 has a 16-bit free-running counter to determine the time and event happens, and to make an event happen at a particular time
    - The counter is normally clocked with an 8 MHz clock
    - The Timer Overflow (TOF) bit -- when the timer rolls over from `0x0000` to `0xFFFF` it sets a flip-flop to show that this has happened.
    - The Timer Prescaler (PR2:0) bits of Timer Interrupt Mask 2 (`TMSK2`) register: Allows you to change the frequency of the clock driving the 16-bit counter.

9S12 Built-In Hardware

• The 9S12 has a number of useful pieces of hardware built into the chip.

• Different versions of the 9S12 have slightly different pieces of hardware. Information about the hardware modules is found in data sheet for the modules.

• We are using the MC9S12DG256 chip (often referred to as the 9S12 chip).

• Here is some of the hardware available on the MC9S12DG256:

– **General Purpose Input/Output (GPIO) Pins**: These pins can be used to read the logic level on a 9S12 pin (input) or write a logic level to an HC12 pin (output). We have already seen examples of this – **PORTA** and **PORTB**. Each GPIO pin has an associated bit in a data direction register which you use to tell the 9S12 if you want to use the GPIO pin as input or output. (For example, **DDRA** is the data direction register for **PORTA**.)

– **Timer-Counter Pins**: The 9S12 is often used to time or count events. For example, to use the 9S12 in a speedometer circuit you need to determine the time it takes for a wheel to make one revolution.
To keep track of the number of people passing through a turnstile you need to count the number of times the turnstile is used.
To control the ignition system of an automobile you need to make a particular spark plug fire at a particular time. The 9S12 has hardware built in to do these tasks.
* For information, see the **ECT 16B8C Block User Guide**.

– **Pulse Width Modulation (PWM) Pins**: To make a motor turn at a particular speed you need to send it a pulse width modulated signal. This is a signal at a particular frequency (which differs for different motors), which is high for part of the period and low for the rest of the period. To have the motor turn slowly, the signal might be high for 10% of the time and low for 90% of the time. To have the motor turn fast, the signal might be high for 90% of the time and low for 10% of the time.
* For information, see the **PWM 8B8C Block User Guide**.

– **Serial Interfaces**: It is often convenient to talk to other digital devices (such as another computer) over a serial interface. When you connect your 9S12 to the PC in the lab, the HC12 talks to the PC over a serial interface. The 9S12 has two serial interfaces: an asynchronous serial interface (called the Serial Communications Interface, or SCI) and a synchronous serial interface (called the Serial Peripheral Interface, or SPI).
* For information on the SCI, see the **9S12 Serial Communications Interface (SCI) Block User Guide**.
* For information on the SPI, see the **SPI Block User Guide**.

– **Analog-to-Digital Converter (ADC)**: Sometimes it is useful to convert a voltage to a digital number for use by the 9S12. For example, a temperature sensor may put out a voltage proportional to the temperature. By converting the voltage to a digital number, you can use the 9S12 to determine the temperature.
* For information, see the **ATD 10B8C Block User Guide**.

• Most of the 9S12 pins serve dual purposes. For example, **PORTT** is used for the timer/counter functions. If you do not need to use PORTT for timer/counter functions, you can use the pins of PORTT for GPIO. There are registers which allow you to set up the PORTT pins to use as GPIO, or to use as timer/counter functions. (These are called the **Timer Control Registers**).

Introduction to the 9S12 Timer Subsystem

• The 9S12 has a **16-bit counter** that normally runs with a **24 MHz clock**.

• Complete information on the 9S12 timer subsystem can be found in the ECT 16B8C Block User Guide. **ECT** stands for **Enhanced Capture Timer**.

• When you reset the 9S12, the clock to the timer subsystem is initially turned off to save power.
– To turn on the clock you need to write a 1 to Bit 7 of register **TSCR1**
(Timer System Control Register 1) at address **0x0046**.

• The clock starts at **0x0000**, counts up (0x0001, 0x0002, etc.) until it gets to **0xFFFF**. It rolls over from 0xFFFF to 0x0000, and continues counting forever (until you turn the counter off or reset the 9S12).

• It takes **2.7307 ms** (65,536 counts/24,000,000 counts/sec) for the counter to count from 0x0000 to 0xFFFF and roll over to 0x0000.

• To determine the time an event happens, you can read the value of the clock (by reading the 16-bit TCNT (Timer Count Register) at address **0x0044**.

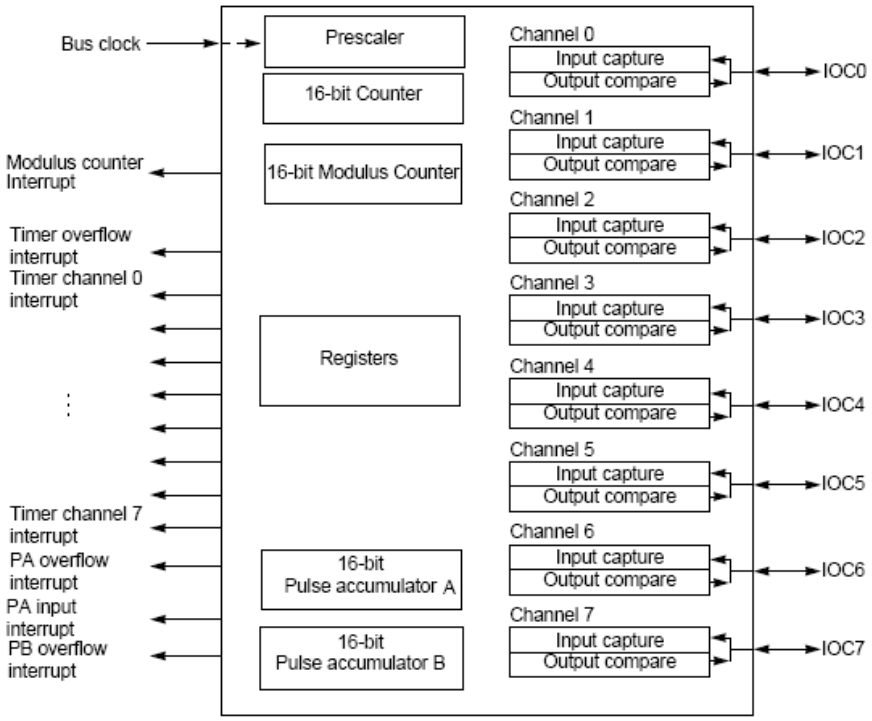


Figure 1-1 Timer Block Diagram

Timer inside the 68HC12:

When you enable the timer (by writing a 1 to bit 7 of TSCR1), you connect an 24−MHz oscillator to a 16−bit counter.

You can read the counter at address **TCNT**.

The counter will start at 0, will count to 0xFFFF, then will roll over to 0x0000. It will take 2.7307 ms for this to happen.

To enable timer on HC12, set Bit 7 of register TCSR1:

```
bset    TSCR1,#$80          TSCR1 = TSCR1 | 0x80;
```

	BIT15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT0
R W	tcnt 15	tcnt 14	tcnt 13	tcnt 12	tcnt 11	tcnt 10	tcnt 9	tcnt 8	tcnt 7	tcnt 6	tcnt 5	tcnt 4	tcnt 3	tcnt 2	tcnt 1	tcnt 0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-5 Timer Count Register (TCNT)

Read anytime and writes have no meaning or effect.

All bits reset to zero.

The 16-bit main timer is an up counter. A read to this register will return the current value of the counter. Access to the counter register will take place in one clock cycle.

	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)

Read or write anytime except PRNT bit is write once.All bits reset to zero.

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.

• To put in a delay of 2.7307 ms, you could wait from one reading of 0x0000 to the next reading of 0x0000.

• **Problem:** You cannot read the TCNT register quickly enough to make sure you will see the 0x0000.

To put in a delay for 2.7307 ms, could watch timer until TCNT == 0x0000:

```
        bset    TSCR1,#$80          TSCR1 = TSCR1 | 0x80;
l1:     ldd     TCNT                while (TCNT != 0x0000) ;
        bne l1
```

Problem: You might see 0xFFFF and 0x0001, and miss 0x0000

• **Solution:** The 9S12 has built-in hardware with will set a flip-flop every time the counter rolls over from 0xFFFF to 0x0000.

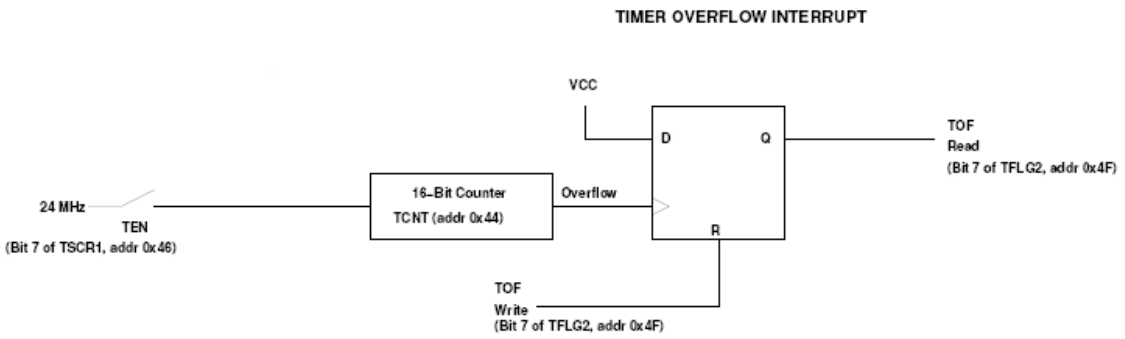
• To wait for 2.7307 ms, just wait until the flip-flop is set, then clear the flip-flop, and wait until the next time the flip-flop is set.

• You can find the state of the flip-flop by looking at bit 7 (the **Timer Overflow Flag** (**TOF**) bit) of the Timer Flag Register 2 (**TFLG2**) register at address **0x004F**.

• You can clear the flip-flop by writing a 1 to the TOF bit of TFLG2.

Solution: When timer overflows, it latches a 1 into a flip−flop. Now when timer overflows (goes from 0xFFFF to 0x0000), **Bit 7 of TFLG2 register is set** to one. **Can clear register by writing a 1** to Bit 7 of TFLG register.

(Note: Bit 7 of TFLG2 for a read is different than
Bit 7 of TFLG2 for a write)




```
    bset TSCR1,#$80      ; Enable timer
l1: brclr TFLG2,#$80,l1 ; Wait until Bit 7 of TFLG2 is set
    ldaa #$80
    .
    program …
    .
    staa TFGL2           ; Clear TOF flag
```

```
TSCR1 = TSCR1 | 0x80;          //Enable timer
while ((TFLG2 & 0x80) == 0) ; // Wait for TOF
.
program …
.
TFLG2 = 0x80;                  // Clear TOF
```

	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

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

Read anytime.

Write used in the flag clearing mechanism. Writing a one to the flag clears the flag. Writing a zero will not affect the current status of the bit.

• **Another problem:** Sometimes you may want to delay longer than 2.7307 ms, or time an event which takes longer than 2.7307 ms. This is hard to do if the counter rolls over every 2.7307 ms.

• **Solution:** The 9S12 allows you to slow down the clock which drives the counter.

• You can **slow down the clock by dividing the 24 MHz clock** by 2, 4, 8, 16, 32, 64 or 128.

• You do this by writing to the prescaler bits (**PR2:0**) of the **Timer System Control Register 2** (**TSCR2**) Register at address **0x004D**.

	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)

PR2, PR1, PR0 - Timer Prescaler Select

These three bits specify the division rate of the main Timer prescaler when the PRNT bit of register TSCR1 is set to "0". The newly selected prescale factor will not take effect until the next synchronized edge where all prescale counter stages equal zero.

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

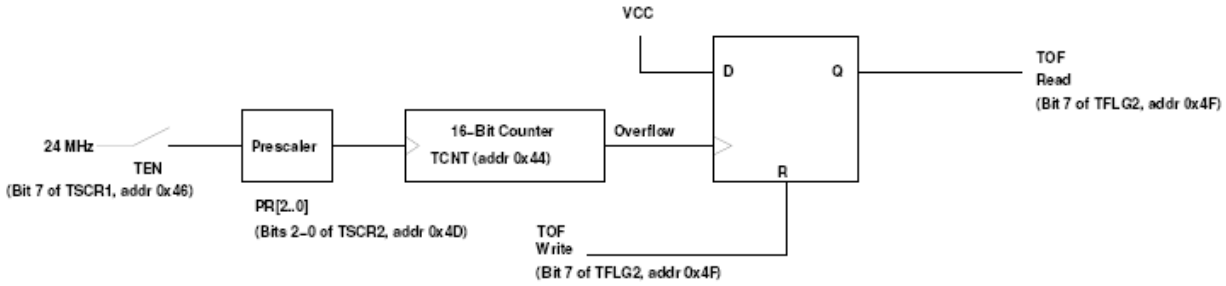
For example, 2.7307 ms will be too short if you want to see lights flash. You can slow down clock by dividing it before you send it to the 16−bit counter. By setting prescaler bits **PR2,PR1,PR0** of TSCR2 you can slow down the clock:

PR	Divide	Freq	Overflow Rate
000	1	24 MHz	2.7307 ms
001	2	12 MHz	5.4613 ms
010	4	6 MHz	10.9227 ms
011	8	3 MHz	21.8453 ms
100	16	1.5 MHz	43.6907 ms
101	32	0.75 MHz	87.3813 ms
110	64	0.375 MHz	174.7627 ms
111	128	0.1875 MHz	349.5253 ms

To set up timer so it will overflow every 87.3813 ms:

```
bset TSCR1,#$80
ldaa #$05
staa TSCR2
```

```
TSCR1 = TSCR1 | 0x80;
TSCR2 = 0x05;
```



Setting and Clearing Bits in C

• To put a specific number into a memory location or register (e.g., to **put 0x55 into PORTA**):

movb #$55,PORTA	**PORTA = 0x55;**

• To set a particular bit of a register (e.g., **set Bit 4 of PORTA**) while leaving the other bits unchanged do a bitwise OR of the register and a mask which has a 1 in the bit(s) you want to set, and a 0 in the other bits:

bset PORTA,#$10	**PORTA = PORTA | 0x10;**

• To clear a particular bit of a register (e.g., **clear Bit 5 of PORTA**) while leaving the other bits unchanged do a bitwise AND of the register and a mask which has a 0 in the bit(s) you want to clear, and a 1 in the other bits. You can construct this mask by complementing a mask which has a 1 in the bit(s) you want to set, and a 0 in the other bits:

bclr PORTA,#$20	**PORTA = PORTA & 0xDF;** **or** **PORTA = PORTA & ~0x20;**

• **To change several bits of a register**, AND the register with 1's in the bits you want to leave unchanged, then OR the result with 1's in the bits you want to set, and 0's in the bits you want to clear. For example, **to set bits 2 and 0, and clear bit 1** (write 101 to bits 2-0) **of TSCR2**, do the following:

ldaa TSCR2 **anda 0xF8** **ora 0x05** **staa TSCR2**	**TSCR2 = (TSCR2 & 0xF8) | 0x05;**