A software delay

- To enter a software delay, put in a nested loop, just like in assembly.
  - Write a function `delay(num)` which will delay for `num` milliseconds

```
void delay(unsigned int num)
{
    volatile unsigned int i;

    while (num > 0)
    {
        i = XXXX;
                        /* ------------------------ */
        while (i > 0)   /*                          */
        {               /* Want inner loop to delay */
            i = i - 1;  /* for 1 ms                 */
        }               /*                          */
                        /* ------------------------ */
        num = num - 1;
    }
}
```

- What should `XXXX` be to make a 1 ms delay?

- Look at assembly listing generated by compiler:

```
        0000206f <delay>:
            206f:   34              pshx
            2070:   7c 10 02        std   1002 <_.z>
            2073:   27 1e           beq   2093 <delay+0x24>
            2075:   18 00 80 03     movw  #XXX <__bss_size+0xXXX>, 0,SP
            2079:   e8
            207a:   ec 80           ldd   0,SP
            207c:   27 0a           beq   2088 <delay+0x19>
           ------------------------------------------------
inner      | 207e:   ed 80           ldy   0,SP
loop       | 2080:   1a 5f           leax  -1,Y
takes      | 2082:   6e 80           stx   0,SP
13 cycles  | 2084:   ec 80           ldd   0,SP
           | 2086:   26 f6           bne   207e <delay+0xf>
           ------------------------------------------------
            2088:   fc 10 02        ldd   1002 <_.z>
            208b:   c3 ff ff        addd  #ffff <_stack+0xc3ff>
            208e:   7c 10 02        std   1002 <_.z>
            2091:   26 e2           bne   2075 <delay+0x6>
            2093:   30              pulx
            2094:   3d              rts
```

- Inner loop takes 13 cyles.
- One millisecond takes 24,000 cycles
  (24,000,000 cycles/sec $\times$ 1 millisecond = 24,000 cycles)
- Need to execute inner loop 24,000/13 = 1,846 times to delay for 1 millisecond

```
void delay(unsigned int num)
{
    volatile unsigned int i;

    while (num > 0)
    {
        i = 1846;
                        /* ------------------------  */
        while (i > 0)   /*                           */
        {               /* Inner loop takes 13 cycles */
            i = i - 1; /* Execute 1846 times to     */
        }               /* delay for 1 ms            */
                        /* ------------------------  */
        num = num - 1;
    }
}
```

Program to increment LEDs connected to PORTA, and delay for 50 ms between changes

```
#include "hcs12.c"
#define D_1MS (24000/13)    // Inner loop takes 13 cycles
                            // Need 24,000 cycles for 1 ms

void delay(unsigned int num);
main()
{
    DDRA = 0xff;     /* Make PORTA output */
    PORTA = 0;       /* Start with all off */
    while(1)
    {
        PORTA = PORTA + 1;
        delay(50);
    }
}

void delay(unsigned int num)
{
    volatile unsigned int i;

    while (num > 0)
    {
        i = D_1MS;
        while (i > 0)
        {
            i = i - 1;
        }
        num = num - 1;
    }
}
```

Program to display a particular pattern of lights on PORTA

```
#include "hcs12.c"

#define D_1MS (24000/13)    // Inner loop takes 13 cycles
                            // Need 24,000 cycles for 1 ms
#define TABLEN 8

void delay(unsigned int num);
main()
{
    const char table[] = {0x80,0x40,0x20,0x10,
                          0x08,0x04,0x02,0x01};
    int i;

    DDRA = 0xff;     /* Make PORTA output */
    PORTA = 0;       /* Start with all off */
    i = 0;
    while(1)
    {
        PORTA = table[i];
        delay(50);
        i = i + 1;
        if (i >= TABLEN) i = 0;  /* Start over when */
                                 /* end is reached  */
    }
}
```

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 MC9S12DP256 chip (often referred to as the 9S12 chip).
- Here is some of the hardware available on the MC9S12DP256:
  - **General Purpose Input/Output (GPIO) Pins**: These pins can be used to read the logic level on an 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 informaiton on the SCI, see the **9S12 Serial Communications Interface (SCI) Block User Guide**.
    * For informaiton 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 an 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`.

1.4 Block Diagram

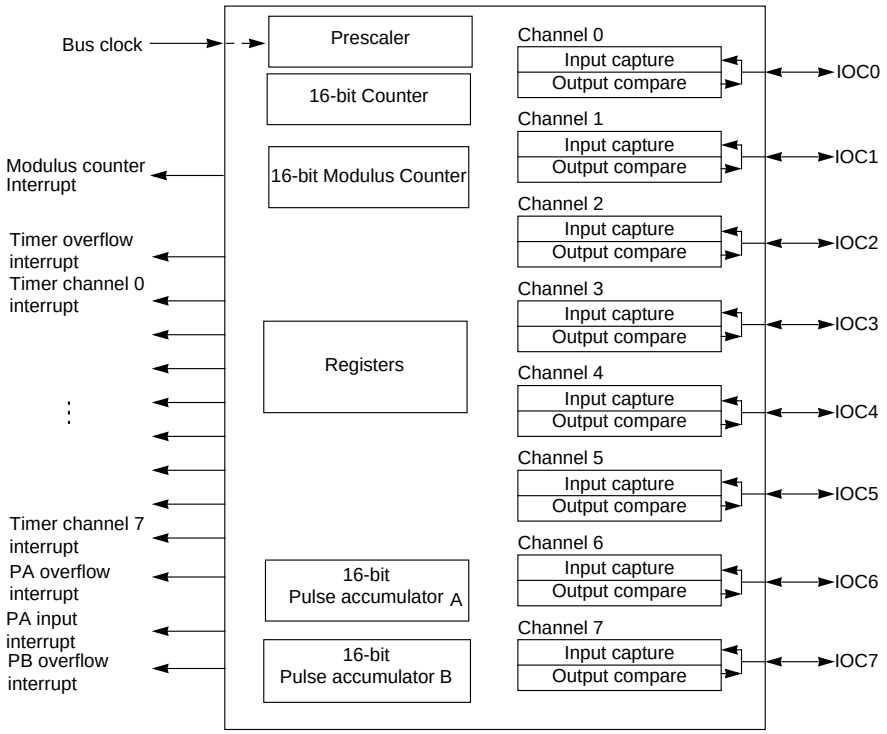


Figure 1-1 Timer Block Diagram

Timer inside the 68HC12:

When you enable timer (by writing a 1 to bit 7 of TSCR),

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

roll over to 0x0000. It will take 2.7307 ms for this to happen.

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

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

3.3.6 TSCR1 — Timer System Control Register 1

Register offset: $_06

	BIT7	6	5	4	3	2	1	BIT0
R	TEN	TSWAI	TSFRZ	TFFCA	0	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.

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.

If for any reason the timer is not active, there is no ÷64 clock for the pulse accumulator since the ÷64 is generated by the timer prescaler.

TSWAI — Timer Module Stops While in Wait
0 = Allows the timer module to continue running during wait.
1 = Disables the timer module when the MCU is in the wait mode. Timer interrupts cannot be used to get the MCU out of wait.

TSWAI also affects pulse accumulators and modulus down counters.

TSFRZ — Timer and Modulus Counter Stop While in Freeze Mode
0 = Allows the timer and modulus counter to continue running while in freeze mode.
1 = Disables the timer and modulus counter whenever the MCU is in freeze mode. This is useful for emulation.

TSFRZ does not stop the pulse accumulator.

TFFCA — Timer Fast Flag Clear All
0 = Allows the timer flag clearing to function normally.
1 = For TFLG1($0E), a read from an input capture or a write to the output compare channel ($10–$1F) causes the corresponding channel flag, CnF, to be cleared. For TFLG2 ($0F), any access to the TCNT register ($04, $05) clears the TOF flag. Any access to the PACN3 and PACN2 registers ($22, $23) clears the PAOVF and PAIF flags in the PAFLG register ($21). Any access to the PACN1 and PACN0 registers ($24, $25) clears the PBOVF flag in the PBFLG register ($31). This has the advantage of eliminating software overhead in a separate clear sequence. Extra care is required to avoid accidental flag clearing due to unintended accesses.

3.3.4 OC7D — Output Compare 7 Data Register

Register offset: $_03

	BIT7	6	5	4	3	2	1	BIT0
R W	OC7D7	OC7D6	OC7D5	OC7D4	OC7D3	OC7D2	OC7D1	OC7D0
RESET:	0	0	0	0	0	0	0	0

Figure 3-4 Output Compare 7 Data Register (OC7D)

Read or write anytime.

A channel 7 output compare can cause bits in the output compare 7 data register to transfer to the timer port data register depending on the output compare 7 mask register.

3.3.5 TCNT — Timer Count Register

Register offset: $_04-$_05

	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)

The 16-bit main timer is an up counter.

A full access for the counter register should take place in one clock cycle. A separate read/write for high byte and low byte will give a different result than accessing them as a word.

Read anytime.

Write has no meaning or effect in the normal mode; only writable in special modes (test_mode = 1).

The period of the first count after a write to the TCNT registers may be a different size because the write is not synchronized with the prescaler clock.

- 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`.

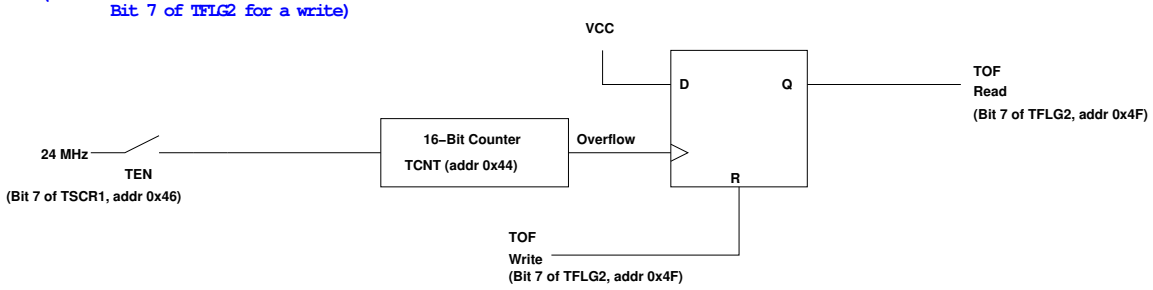
Solution: When timer overflows, latch a 1 into a flip-flop.

Now when timer overflows (goes from 0xFFFF to 0x0000),

Bit 7 of TFLG2 register is set to one. Can clear

register by writting 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
11: brclr   TFLG2,#$80,11   ; Wait until Bit 7 of TFLG2 is set
    ldaa    #$80
    staa    TFGL2           ; Clear TOF flag
```

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

3.3.13 TFLG2 — Main Timer Interrupt Flag 2

Register offset: $_0F

	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)

TFLG2 indicates when interrupt conditions have occurred. To clear a bit in the flag register, write the bit to one.

Read anytime. Write used in clearing mechanism (set bits cause corresponding bits to be cleared).

Any access to TCNT will clear TFLG2 register if the TFFCA bit in TSCR register is set.

TOF — Timer Overflow Flag

Set when 16-bit free-running timer overflows from $FFFF to $0000. This bit is cleared automatically by a write to the TFLG2 register with bit 7 set. (See also TCRE control bit explanation.)

- 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`.

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

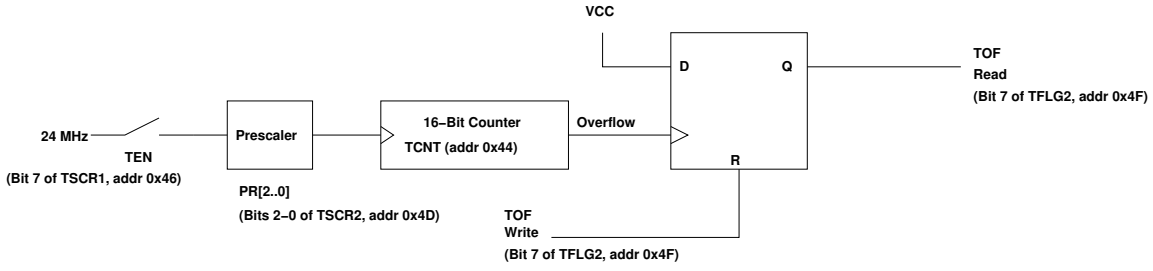
slow down the clock:

PR2:0	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          TSCR1 = TSCR1 | 0x80;
ldaa   #$05                TSCR2 = 0x05;
staa   TSCR2
```

TIMER OVERFLOW INTERRUPT



3.3.10 TIE — Timer Interrupt Enable Register

Register offset: $_0C

	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)

Read or write anytime.

The bits in TIE correspond bit-for-bit with the bits in the TFLG1 status register. If cleared, the corresponding flag is disabled from causing a hardware interrupt. If set, the corresponding flag is enabled to cause a interrupt.

C7I–C0I — Input Capture/Output Compare “x” Interrupt Enable

3.3.11 TSCR2 — Timer System Control Register 2

Register offset: $_0D

	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)

Read or write anytime.

TOI — Timer Overflow Interrupt Enable
0 = Interrupt inhibited
1 = Hardware interrupt requested when TOF flag set

TCRE — Timer Counter Reset Enable

This bit allows the timer counter to be reset by a successful output compare 7 event. This mode of operation is similar to an up-counting modulus counter.
0 = Counter reset inhibited and counter free runs
1 = Counter reset by a successful output compare 7

If TC7 = $0000 and TCRE = 1, TCNT will stay at $0000 continuously. If TC7 = $FFFF and TCRE = 1, TOF will never be set when TCNT is reset from $FFFF to $0000.

PR2, PR1, PR0 — Timer Prescaler Select

These three bits specify the number of ÷2 stages that are to be inserted between the bus clock and the main timer counter.

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

The newly selected prescale factor will not take effect until the next synchronized edge where all prescale counter stages equal zero.

3.3.12 TFLG1 — Main Timer Interrupt Flag 1

Register offset: $_0E

	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)

TFLG1 indicates when interrupt conditions have occurred. To clear a bit in the flag register, write a one to the bit.

Use of the TFMOD bit in the ICSYS register ($2B) in conjunction with the use of the ICOVW register ($2A) allows a timer interrupt to be generated after capturing two values in the capture and holding registers instead of generating an interrupt for every capture.

Read anytime. Write used in the clearing mechanism (set bits cause corresponding bits to be cleared). Writing a zero will not affect current status of the bit.

When TFFCA bit in TSCR register is set, a read from an input capture or a write into an output compare channel ($10–$1F) will cause the corresponding channel flag CnF to be cleared.

C7F–C0F — Input Capture/Output Compare Channel “n” Flag.

C0F can also be set by 16 - bit Pulse Accumulator B (PACB). C3F - C0F can also be set by 8 - bit pulse accumulators PAC3 - PAC0.

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;

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

- Write to all bits of a register when you know what all bits should be, such as when you initialize it. Set or clear bits when you want to change only one or a few bits and leave the others unchanged.