HC12 Built-In Hardware

- The HC12 has a number of useful pieces of hardware built into the chip.
- Different versions of the HC12 have slightly different pieces of hardware.
- We are using the MC68HC912B32 chip (often referred to as the HC12B32 chip).
- Here is some of the hardware available on the HC12B32:
  - **General Purpose Input/Output (GPIO) Pins**: These pins can be used to read the logic level on an HC12 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 HC12 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 HC12 is often used to time or count events. For example, to use the HC12 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 HC12 has hardware built in to do these tasks.
  - **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.
  - **Serial Interfaces**: It is often convenient to talk to other digital devices (such as another computer) over a serial interface. When you connect your HC12 to the PC in the lab, the HC12 talks to the PC over a serial interface. The HC12 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).

  - **Analog-to-Digital Converter (ADC)**: Sometimes it is useful to convert a voltage to a digital number for use by the HC12. 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 HC12 to determine the temperature.

- Most of the HC12 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 HC12 Timer Subsystem

- The HC12 has a 16-bit clock the normally runs with an 8 MHz clock.
- When you reset the HC12, the clock is initially turned off.
  - To turn on the clock you need to write a `1` to Bit 7 of register `TSCR` (Timer System Control Register) at address `0x0086`.
- 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 HC12).
- It takes 8.192 ms (65,536 counts/8,000,000 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 `0x0084`.

Timer inside the 68HC12:

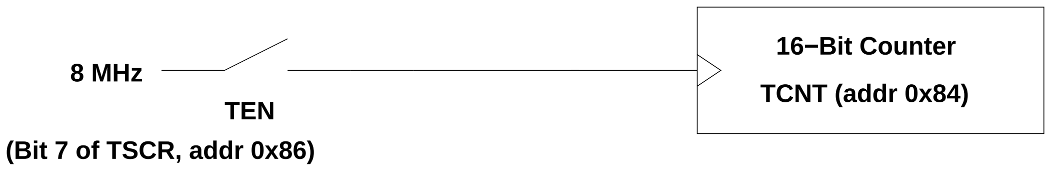
When you enable timer (by writing a 1 to bit 7 of TSCR),

you connect an 8−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 8.192 ms for this to happen.

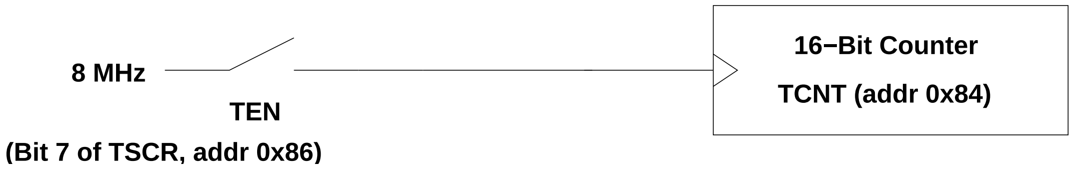


To enable timer on HC12, set Bit 7 of register TCSR:

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

- To put in a delay of 8.192 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`.

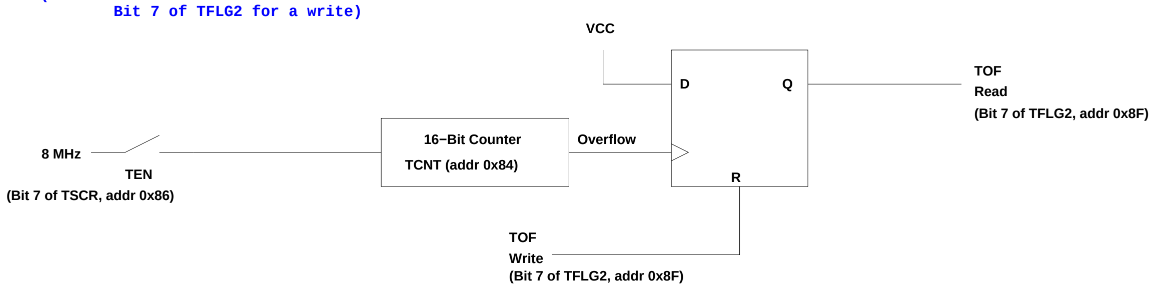
```
To put in a delay for 8.192 ms, could watch timer until

TCNT == 0x0000:

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

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



- **Solution**: The HC12 has built-in hardware with will set a flip-flop every time the counter rolls over from `0xFFFF` to `0x0000`.
- To wait for 8.192 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 `0x008F`.
- 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)
```

TIMER OVERFLOW INTERRUPT


VCC
D
Q
TOF
Read
(Bit 7 of TFLG2, addr 0x8F)
8 MHz
TEN
(Bit 7 of TSCR, addr 0x86)
16–Bit Counter
TCNT (addr 0x84)
Overflow
R
TOF
Write
(Bit 7 of TFLG2, addr 0x8F)

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

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

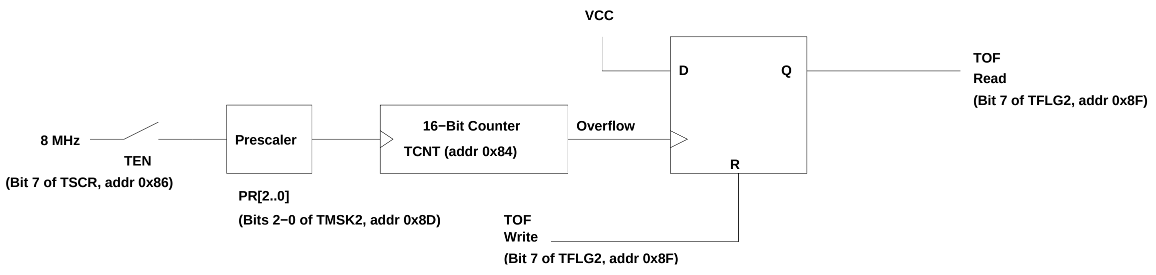
- Another problem: Sometimes you may want to delay longer than 8.192 ms, or time an event which takes longer than 8.192 ms. This is hard to do if the counter rolls over every 8.192 ms.
- Solution: The HC12 allows you to slow down the clock which drives the counter.
- You can slow down the clock by dividing the 8 MHz clock by 2, 4, 8, 16, or 32.
- You do this by writing to the prescaler bits (`PR2:0`) of the Timer Interrupt Mask 2 (`TMSK2`) Register at address `0x008D`.

```
8.192 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 TMSK2 you can

slow down the clock:
```

PR2:0	Divide	Freq	Overflow Rate
000	1	8 MHz	8.192 ms
001	2	4 MHz	16.384 ms
010	4	2 MHz	32.768 ms
011	8	1 MHz	65.536 ms
100	16	0.5 MHz	131.072 ms
101	32	0.25 MHz	262.144 ms
110	--	Undefined	
111	--	Undefined	

```
To set up timer so it will overflow every 32.768 ms:

bset   TSCR,#$80            TSCR = TSCR | 0x80;
ldaa   #$02                 TMSK2 = 0x02;
staa   TMSK2
```

TIMER OVERFLOW INTERRUPT


VCC
D
Q
TOF
Read
(Bit 7 of TFLG2, addr 0x8F)
8 MHz
TEN
(Bit 7 of TSCR, addr 0x86)
Prescaler
PR[2..0]
(Bits 2–0 of TMSK2, addr 0x8D)
16–Bit Counter
TCNT (addr 0x84)
Overflow
R
TOF
Write
(Bit 7 of TFLG2, addr 0x8F)

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 3 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 4 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;
```

- 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.

- To clear a bit in a timer flag register (`TFLG1` and `TFLG2`) write a 1 to the bit(s) you want to clear and a zero to all other bits. **Do not do a bitwise OR of the register and a mask.** To clear the Timer Overflow flag (Bit 7 of TFLG2):

```
movb    #$80,TFLG2                 TFGL2 = 0x80;
```