• **Interrupts and the Timer Overflow Interrupts**
  • Huang Sections 6.1-6.4
    o Using the Timer Overflow Flag to interrupt a delay
    o Introduction to Interrupts
    o How to generate an interrupt when the timer overflows
    o How to tell the MC9S12 where the ISR is located
    o Using interrupts on the HC12
    o The MC9S12 registers and stack when a TOF interrupt is received
    o The MC9S12 registers and stack after a TOF interrupt is received
    o Interrupt vectors for the MC9S12
    o Using interrupts on the MC9S12: Assembly and C

**What Happens When You Reset the HCS12?**

• What happens to the HCS12 when you turn on power or push the reset button?

• How does the HCS12 know which instruction to execute first?

• On reset the HCS12 loads the PC with the address located at address \textbf{0xFFFFE and 0xFFFFF}.

• Here is what is in the memory of our MC9S12:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFF0</td>
<td>F6</td>
<td>EC</td>
<td>F6</td>
<td>F0</td>
<td>F6</td>
<td>F4</td>
<td>F6</td>
<td>F8</td>
<td>F6</td>
<td>FC</td>
<td>F7</td>
<td>00</td>
<td>F7</td>
<td>04</td>
<td>F0</td>
<td>00</td>
</tr>
</tbody>
</table>

• On reset or power-up, the first instruction your MC9S12 will execute is the one located at address \textbf{0xF000}. 
The MC9S12 Timer

• The MC9S12 has a 16-bit free-running counter (timer).

• The MC9S12 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:

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

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

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

TSCR1 = TSCR1 | 0x80;
TSCR2 = 0x05;
Using the Timer Overflow Flag to implement a delay

• The MC9S12 timer counts at a rate set by the prescaler:

<table>
<thead>
<tr>
<th>PR2:0</th>
<th>Divide</th>
<th>Clock Freq</th>
<th>Clock Period</th>
<th>Overflow Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>1</td>
<td>24 MHZ</td>
<td>0.042 µs</td>
<td>2.73 ms</td>
</tr>
<tr>
<td>001</td>
<td>2</td>
<td>12 MHZ</td>
<td>0.083 µs</td>
<td>5.46 ms</td>
</tr>
<tr>
<td>010</td>
<td>4</td>
<td>6 MHZ</td>
<td>0.167 µs</td>
<td>10.92 ms</td>
</tr>
<tr>
<td>011</td>
<td>8</td>
<td>3 MHZ</td>
<td>0.333 µs</td>
<td>21.85 ms</td>
</tr>
<tr>
<td>100</td>
<td>16</td>
<td>1.5 MHZ</td>
<td>0.667 µs</td>
<td>43.69 ms</td>
</tr>
<tr>
<td>101</td>
<td>32</td>
<td>750 MHZ</td>
<td>1.333 µs</td>
<td>87.38 ms</td>
</tr>
<tr>
<td>110</td>
<td>64</td>
<td>375 MHZ</td>
<td>2.667 µs</td>
<td>174.76 ms</td>
</tr>
<tr>
<td>111</td>
<td>128</td>
<td>187.5 MHZ</td>
<td>5.333 µs</td>
<td>349.53 ms</td>
</tr>
</tbody>
</table>

• When the timer overflows it sets the TOF flag (bit 7 of the TFLG2 register).

• To clear the TOF flag write a 1 to bit 7 of the TFLG2 register, and 0 to all other bits of TFLG2:

\[ TFLG2 = 0x80; \]

• You can implement a delay using the TOF flag by waiting for the TOF flag to be set, then clearing it:

```c
void delay(void) {
    while ((TFLG2 & 0x80) == 0); // Wait for TOF */
    TFLG2 = 0x80; // Clear flag */
}
```

• If the prescaler is set to 010, you will exit the delay subroutine after 10.92 ms have passed.

**Problem**: Can’t do anything else while waiting.

**Solution**: Have timer generate an interrupt. Program can do other things; automatically switches to service interrupt when interrupt occurs.
How to generate an interrupt when the timer overflows

To generate a TOF interrupt:

Enable timer (set Bit 7 of TSCR1)
Set prescaler (Bits 2:0 of TSCR2)
Enable TCF interrupt (set Bit 7 of TSCR2)
Enable interrupts (clear I bit of CCR)

Inside TOF ISR:

Take care of event
Clear TOF flag (Write 1 to Bit 7 of TFLG2)
Return with RTI

#include "derivative.h"

main()
{
    DDRB = 0xff;  /* Make Port B output */
    TSCR1 = 0x80;  /* Turn on timer */
    TSCR2 = 0x85;  /* Enable timer overflow interrupt, set prescaler */
    TFLG2 = 0x80;  /* Clear timer interrupt flag */
    enable();    /* Enable interrupts (clear I bit) */
    while (1)
    {
        /* Do nothing */
    }
}

interrupt void toi_isr(void)
{
    PORTB = PORTB + 1;  /* Increment Port B */
    TFLG2 = 0x80;  /* Clear timer interrupt flag */
}
How to tell the HCS12 where the Interrupt Service Routine (ISR) is located

• You need to tell the HCS12 where to go when it receives a TOF interrupt.

• You do this by setting the TOF Interrupt Vector.

• The TOF interrupt vector is located at 0xFFDE. This is in flash EPROM, and is very difficult to change — you would have to modify and reload DBug-12 to change it.

• DBug-12 redirects the interrupts to a set of vectors in RAM, from 0x3E00 to 0x3E7F. The TOF interrupt is redirected to 0x3E5E. When you get a TOF interrupt, the HCS12 initially executes code starting at 0xFFDE. This code tells the HCS12 to load the program counter with the address in 0x3E5E. Because this address in RAM, you can change it without having to modify and reload DBug-12.

• Because the redirected interrupt vectors are in RAM, you can change them in your program.
How to Use Interrupts in Assembly Programs

• For our assembler, you can set the interrupt vector by including the file `hcs12.inc`. In this file, the addresses of all of the 9212 interrupt vectors are defined.

• For example, the pointer to the Timer Overflow Interrupt vector is called `UserTimerOvf`:

  \[
  \text{UserTimerOvf} \quad \text{equ} \quad \$3E5E
  \]

You can set the interrupt vector to point to the interrupt service routine `toi_isr` with the Assembly statement:

  \[
  \text{movw} \quad \#\text{toi_isr}, \text{UserTimerOvf}
  \]
• Here is a program where the interrupt vector is set in the program:

```assembly
include 'derivative.inc'
include "vectors12.inc"

prog: equ $2000

org prog
movw #toi_isr,UserTimerOvf ; Set interrupt vector
movb #$ff,DDRP
bset PTP,#$0f
bset DDRJ,#$02
bclr PTJ,#$02
movb #$ff,DDR B ; Port B output
movb #$80,TSCR1 ; Turn on timer
movb #$86,TSCR2 ; Enable timer overflow interrupt, set prescaler
                ; so interrupt period is 175 ms
movb #$80,TFLG2 ; Clear timer interrupt flag
cli ; Enable interrupts

11: wai ; Do nothing - go into low power mode */
bra 11

toi_isr:
    inc PORTB
    movb #$80,TFLG2 ; Clear timer overflow interrupt flag
    rti
```

• When the MC9S12 receives a Timer Overflow Interrupt, it finishes the current instruction, puts return address and all registers on the stack, sets the I bit of the CCR to disable interrupts, then loads the contents of UserTimerOvf (0x3E5E) into the PC.

• After executing the ISR, the rti instruction pulls the registers off the stack, and loads the PC with the return address – the program resumes from where it received the interrupt.
How to Use Interrupts in C Programs

• For our C compiler, you can set the interrupt vector by including the file `vectors12.h`. In this file, pointers to the locations of all of the MC9S12 interrupt vectors are defined.

• For example, the pointer to the Timer Overflow Interrupt vector is called `UserTimerOvf`:

```c
#define VECTOR_BASE 0x3E00
#define _VEC16(off) *(volatile unsigned short *)(VECTOR_BASE + off)
#define UserTimerOvf _VEC16(47)
```

The Timer Overflow vector is the 47'th vector, so it is located at

\[
0x3E00 + (47*2) = 0x3E00 + 0x005E = 0x3E5E
\]

You can set the interrupt vector to point to the interrupt service routine `toi_isr()` with the C statement:

```c
UserTimerOvf = (unsigned short) &toi_isr;
```
• Here is a program where the interrupt vector is set in the program:

```c
#include <hidef.h>    /* common defines and macros */
#include "derivative.h"  /* derivative-specific definitions */
#include "vectors12.h"  /* SRAM interrupt vector redirect */
#define enable() __asm(cli)
#define disable() __asm(sei)
interrupt void toi_isr(void);
main()
{
    DDRB = 0xff;   /* Make Port B output */
    TSCR1 = 0x80;  /* Turn on timer */
    TSCR2 = 0x86;  /* Enable timer overflow interrupt, set prescaler */
        /*so interrupt period is 175 ms */
    TFLG2 = 0x80;  /* Clear timer interrupt flag */
    UserTimerOvf = (unsigned short) &toi_isr;
    enable();     /* Enable interrupts (clear I bit) */
    while (1)
    {
        __asm(wai);    /* Do nothing - go into low power mode */
    }
}

interrupt void toi_isr(void)
{
    PORTB = PORTB+1;
    TFLG2 = 0x80;    /* Clear timer interrupt flag */
}
```

• The interrupt keyword tells the compiler to return from the function using the rti instruction rather than the rts instruction.
Using Interrupts on the MC9S12

What happens when the HCS12 receives an unmasked interrupt?

1. Finish current instruction
2. Push all registers onto the stack
3. Set I bit of CCR
4. Load Program Counter from interrupt vector for particular interrupt

Most interrupts have both a specific mask and a general mask. For most interrupts the general mask is the I bit of the CCR. For the TOF interrupt the specific mask is the TOI bit of the TSCR2 register.

Before using interrupts, make sure to:

1. Load stack pointer
   - Done for you in C by the C startup code

2. Write Interrupt Service Routine
   - Do whatever needs to be done to service interrupt
   - Clear interrupt flag
   - Exit with RTI
     - Use the INTERRUPT definition in the Gnu C compiler

3. Load address of interrupt service routine into interrupt vector

4. Do any setup needed for interrupt
   - For example, for the TOF interrupt, turn on timer and set prescaler

5. Enable specific interrupt.

6. Enable interrupts in general (clear I bit of CCR with cli instruction or enable() function

Can disable all (maskable) interrupts with the sei instruction or disable() function.
An example of the MC9S12 registers and stack when a TOF interrupt is received
An example of the MC9S12 registers and stack just after a TOF interrupt is received

- All of the MC9S12 registers are pushed onto the stack, the PC is loaded with the contents of the Interrupt Vector, and the I bit of the CCR is set
Interrupt vectors for the 68HC912B32

- The interrupt vectors for the MC9S12DP256 are located in memory from 0xFF80 to 0xFFFF.

- These vectors are programmed into Flash EEPROM and are very difficult to change.

- DBug12 redirects the interrupts to a region of RAM where they are easy to change.

- For example, when the MC9S12 gets a TOF interrupt:
  - It loads the PC with the contents of 0xFFDE and 0xFFDF.
  - The program at that address tells the MC9S12 to look at address 0x3E5E and 0x3E5F.
  - If there is a 0x0000 at these two addresses, DBug12 gives an error stating that the interrupt vector is uninitialized.
  - If there is anything else at these two addresses, DBug12 loads this data into the PC and executes the routine located there.
  - To use the TOF interrupt you need to put the address of your TOF ISR at addresses 0x3E5E and 0x3E5F.
Commonly Used Interrupt Vectors for the MC9S12DP256
<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Specific Mask</th>
<th>General Mask</th>
<th>Normal Vector</th>
<th>Debug Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI2</td>
<td>SP2CR1 (SPIE, SPTIE)</td>
<td>I</td>
<td>FFBC, FFBD</td>
<td>3E5C, 3E5D</td>
</tr>
<tr>
<td>SPI1</td>
<td>SP1CR1 (SPIE, SPTIE)</td>
<td>I</td>
<td>FFBE, FFBF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>IIC</td>
<td>IBCR (IBIR)</td>
<td>I</td>
<td>FFCC, FFCD</td>
<td>3E4C, 3E4D</td>
</tr>
<tr>
<td>BDLIC</td>
<td>DLBCR (IE)</td>
<td>I</td>
<td>FFCA, FFCB</td>
<td>3E4D, 3E4B</td>
</tr>
<tr>
<td>CRG Self Clock Mode</td>
<td>CRGINT (SCMIE)</td>
<td>I</td>
<td>FFCC, FFCE</td>
<td>3E4E, 3E4F</td>
</tr>
<tr>
<td>CRG Lock</td>
<td>CRGINT (LOCKIE)</td>
<td>I</td>
<td>FFCD, FFCE</td>
<td>3E4F, 3E4G</td>
</tr>
<tr>
<td>Pulse Acc B Overflow</td>
<td>PBCTRL (PB0V1)</td>
<td>I</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>Mod Down Ctrl UnderFlow</td>
<td>MCCTRL (MCZI)</td>
<td>I</td>
<td>FFDD, FFDE</td>
<td>3E5F, 3E5G</td>
</tr>
<tr>
<td>Port H</td>
<td>PTHIF (PTHIE)</td>
<td>I</td>
<td>FFDD, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>Port J</td>
<td>PTJIF (PTJIE)</td>
<td>I</td>
<td>FFDD, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>ATD1</td>
<td>ATD1CTL2 (ASCIE)</td>
<td>I</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>ATDO</td>
<td>ATD0CTL2 (ASCIE)</td>
<td>I</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>SCI1</td>
<td>SCI1CR2 (TIE, TCIE, RIE, ILIE)</td>
<td>I</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>SCI0</td>
<td>SCOCR2 (TIE, TCIE, RIE, ILIE)</td>
<td>I</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>SPI0</td>
<td>SPOCR1 (SPIE)</td>
<td>I</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>Pulse Acc A Edge</td>
<td>PACTL (PAI)</td>
<td>I</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>Pulse Acc A Overflow</td>
<td>PACTL (PA0V1)</td>
<td>I</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>Enh Capt Timer Overflow</td>
<td>TSCR2 (TUI)</td>
<td>I</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>Enh Capt Timer Channel 7</td>
<td>TIE (C7I)</td>
<td>I</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>Enh Capt Timer Channel 6</td>
<td>TIE (C6I)</td>
<td>I</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>Enh Capt Timer Channel 5</td>
<td>TIE (C5I)</td>
<td>I</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>Enh Capt Timer Channel 4</td>
<td>TIE (C4I)</td>
<td>I</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>Enh Capt Timer Channel 3</td>
<td>TIE (C3I)</td>
<td>I</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>Enh Capt Timer Channel 2</td>
<td>TIE (C2I)</td>
<td>I</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>Enh Capt Timer Channel 1</td>
<td>TIE (C1I)</td>
<td>I</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>Enh Capt Timer Channel 0</td>
<td>TIE (C0I)</td>
<td>I</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>Real Time</td>
<td>CRGINT (RTIE)</td>
<td>I</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>IRQ</td>
<td>IRQCR (IRQEN)</td>
<td>I</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>XIRQ</td>
<td>(None)</td>
<td>X</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>SWI</td>
<td>(None)</td>
<td>X</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>Unimplemented Instruction</td>
<td>(None)</td>
<td>(None)</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>COP Failure</td>
<td>COPCTRL (CR2-CR0 COP Rate Select)</td>
<td>(None)</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>COP Clock Monitor Fail</td>
<td>PLLCTRL (CME, SCME)</td>
<td>(None)</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>Reset</td>
<td>(None)</td>
<td>(None)</td>
<td>FFDE, FFDF</td>
<td>3E5E, 3E5F</td>
</tr>
</tbody>
</table>

Exceptions on the MC9S12
• Exceptions are the way a processor responds to things other than the normal sequence of instructions in memory.

• Exceptions consist of such things as Reset and Interrupts.

• Interrupts allow a processor to respond to an event without constantly polling to see whether the event has occurred.

• On the HCS12 some interrupts cannot be masked — these are the Unimplemented Instruction Trap and the Software Interrupt (SWI instruction).

• XIRQ interrupt is masked with the X bit of the Condition Code Register. Once the X bit is cleared to enable the XIRQ interrupt, it cannot be set to disable it.

  – The XIRQ interrupt is for external events such as power fail which must be responded to.

• The rest of the HCS12 interrupts are masked with the I bit of the CCR.

  – All these other interrupts are also masked with a specific interrupt mask.
  – This allows you to enable any of these other interrupts you want.
  – The I bit can be set to 1 to disable all of these interrupts if needed.

Using Interrupts on the MC9S12
What happens when the MC9S12 receives an unmasked interrupt?

1. Finish current instruction
2. Push all registers onto the stack
3. Set I bit of CCR
4. Load Program Counter from interrupt vector for particular interrupt

Most interrupts have both a specific mask and a general mask. For most interrupts the general mask is the I bit of the CCR. For the TOF interrupt the specific mask is the TOI bit of the TSCR2 register.

Before using interrupts, make sure to:

1. Load stack pointer
   • Done for you in C by the CodeWarrior startup code.
2. Write Interrupt Service Routine
   • Do whatever needs to be done to service interrupt. Keep it short — do not do things which take a long time, such as a printf(), or wait for some external event.
   • Clear interrupt flag
   • Exit with RTI
     – Use the @interrupt function of the Cosmic C compiler
3. Load address of interrupt service routine into interrupt vector
4. Do any setup needed for interrupt
   • For example, for the TOF interrupt, turn on timer and set prescaler
5. Enable specific interrupt
6. Enable interrupts in general (clear I bit of CCR with cli instruction or enable() function

Can disable all (maskable) interrupts with the sei instruction or disable() function.