• **Resets on the HC12**
• **Introduction to Interrupts on the 9S12**
• Huang Sections 6.1-6.3
• MC9S12DP256B Device User Guide
  o What happens when you reset the HC12?
  o Using the Timer Overflow Flag to implement a delay on the HC12
  o Introduction to Interrupts
  o How to generate an interrupt when the timer overflows
  o How to tell the 9S12 where the Interrupt Service Routine is located
  o Using interrupts on the HC12
  o The 9S12 registers and stack when a TOF interrupt is received
  o The 9S12 registers and stack just after a TOF interrupt is received
  o Interrupt vectors for the MC9S12DP256

### 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 **0xFFFE and 0xFFFF**.

• Here is what is in the memory of our HCS12:

<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 HCS12 will execute is the one located at address **0xF000**.
Using the Timer Overflow Flag to implement a delay

- The HCS12 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:

  $$ \text{TFLG2} = 0x80; $$

- 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**: Use an interrupt to tell you when the timer overflow has occurred.

**Introduction to Interrupts**

**Interrupt**: Allows the HCS12 to do other things while waiting for an event to happen. When the event happens, tell HCS12 to take care of event, then go back to what it was doing.
What happens when HCS12 gets an interrupt: HCS12 automatically jumps to part of the program which tells it what to do when it receives the interrupt (Interrupt Service Routine).

How does HCS12 know where the ISR is located: A set of memory locations called Interrupt Vectors tell the HCS12 the address of the ISR for each type of interrupt.

How does HCS12 know where to return to: Return address pushed onto stack before HCS12 jumps to ISR. You use the RTI (Return from Interrupt) instruction to pull the return address off of the stack when you exit the ISR.

What happens if ISR changes registers: All registers are pushed onto stack before jumping to ISR, and pulled off the stack before returning to program. When you execute the RTI instruction at the end of the ISR, the registers are pulled off of the stack.

To Return from the ISR You must return from the ISR using the RTI instruction. The RTI instruction tells the HCS12 to pull all the registers off of the stack and return to the address where it was processing when the interrupt occurred.

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 "hcs12.h"
main()
{
    DDRA = 0xff;    /* Make Port A 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 */
    }
}

void INTERRUPT toi_isr(void)
{
    PORTA = PORTA + 1; /* Increment Port A */
    TFLG2 = 0x80;     /* Clear timer interrupt flag */
}

How to tell the HCS12 where the Interrupt Service Routine 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 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 9212 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)
```

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

```
UserTimerOvf = (unsigned short) &toi_isr;
```

- Here is a program where the interrupt vector is set in the program:

```c
#include <hcs12.h>
#include <vectors12.h>
#include "DBG12.h"
#define enable() __asm(" cli")
#define disable() __asm(" sei")

void INTERRUPT toi_isr(void);
main()
{
    DDRA = 0xff; /* Make Port A 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)
    {
        /* Do nothing - go into low power mode */
    }
}

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

### 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`:

```
UserTimerOvf equ $3E5E
```
You can set the interrupt vector to point to the interrupt service routine `toi_isr` with the Assembly statement:

```
movw #toi_isr,UserTimerOvf
```

Here is a program where the interrupt vector is set in the program:

```assembly
#include "hcs12.inc"
define prog $1000
    movw    #toi_isr,UserTimerOvf ; Set interrupt vector
    movb    #$ff,DDRA
    movb    #$90,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

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

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

### Using interrupts on the HCS12

What happens when the HCS12 receives an unmasked interrupt?

1. It finishes current instruction.

2. Pushed all registers onto the stack.

3. Sets I bit of CCR.

4. Loads 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

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.

**Interrupt vectors for the HCS12**

• The interrupt vectors for the MC9S12DG256 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 HCS12 gets a TOF interrupt:
  – It loads the PC with the contents of 0xFFDE and 0xFFDF.
  – The program at that address tells the HCS12 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 MC9S12DG256

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Specific Mask</th>
<th>General Mask</th>
<th>Normal Vector</th>
<th>Debug-12 Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI2</td>
<td>SPI2SR (SPIE, SPITIE)</td>
<td>I</td>
<td>FFBC, FFED</td>
<td>3E5C, 3E5D</td>
</tr>
<tr>
<td>SPI1</td>
<td>SPI1SR (SPIE, SPITIE)</td>
<td>I</td>
<td>FFBE, FFDF</td>
<td>3E3E, 3E3F</td>
</tr>
<tr>
<td>IIC</td>
<td>IICR (IIER)</td>
<td>I</td>
<td>FFCA, FFCD</td>
<td>3E40, 3E41</td>
</tr>
<tr>
<td>BDLC</td>
<td>BDLCR (IIE)</td>
<td>I</td>
<td>FFCA, FFCE</td>
<td>3E42, 3E43</td>
</tr>
<tr>
<td>CRG Self Clock Mode</td>
<td>CRG1INT (SCMIIE)</td>
<td>I</td>
<td>FFCC, FFCE</td>
<td>3E44, 3E45</td>
</tr>
<tr>
<td>CRG Lock</td>
<td>CRG1INT (LOCKIE)</td>
<td>I</td>
<td>FFCD, FFCE</td>
<td>3E46, 3E47</td>
</tr>
<tr>
<td>Pulse Acc B Overflow</td>
<td>PSCTL (PSY1)</td>
<td>I</td>
<td>FFCE, FFCH</td>
<td>3E48, 3E49</td>
</tr>
<tr>
<td>Mod Down Cyr UnderFlow</td>
<td>MODTL (MCZ1)</td>
<td>I</td>
<td>FFCA, FFCE</td>
<td>3E4A, 3E4B</td>
</tr>
<tr>
<td>Port H</td>
<td>PTNIP (PTNIE)</td>
<td>I</td>
<td>FFCA, FFCD</td>
<td>3E4C, 3E4D</td>
</tr>
<tr>
<td>Port J</td>
<td>PTJIP (PTJIE)</td>
<td>I</td>
<td>FFCE, FFCF</td>
<td>3E4E, 3E4F</td>
</tr>
<tr>
<td>ATD1</td>
<td>ATDC1CTL2 (ASCIE)</td>
<td>I</td>
<td>FFDA, FFDB</td>
<td>3E50, 3E51</td>
</tr>
<tr>
<td>ATD0</td>
<td>ATDC0CTL2 (ASCIE)</td>
<td>I</td>
<td>FFDA, FFDB</td>
<td>3E52, 3E53</td>
</tr>
<tr>
<td>SCI1</td>
<td>SCI1SR (TIE, TCIE, RIE, ILIE)</td>
<td>I</td>
<td>FFDA, FFDB</td>
<td>3E54, 3E55</td>
</tr>
<tr>
<td>SCI2</td>
<td>SCI2SR (TIE, TCIE, RIE, ILIE)</td>
<td>I</td>
<td>FFDA, FFDB</td>
<td>3E56, 3E57</td>
</tr>
<tr>
<td>SPI0</td>
<td>SPI0CN1 (SPIE)</td>
<td>I</td>
<td>FFDA, FFDB</td>
<td>3E58, 3E59</td>
</tr>
<tr>
<td>Pulse Acc A Edge</td>
<td>PACTL (PA1)</td>
<td>I</td>
<td>FFDA, FFDB</td>
<td>3E5A, 3E5B</td>
</tr>
<tr>
<td>Pulse Acc A Overflow</td>
<td>PACTL (PAVY1)</td>
<td>I</td>
<td>FFDA, FFDB</td>
<td>3E5C, 3E5D</td>
</tr>
<tr>
<td>Enh Capt Timer Overflow</td>
<td>TSQR2 (TQ1)</td>
<td>I</td>
<td>FFDA, FFDB</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>Enh Capt Timer Channel 7</td>
<td>TIE (GT7)</td>
<td>I</td>
<td>FFED, FFEN</td>
<td>3E50, 3E51</td>
</tr>
<tr>
<td>Enh Capt Timer Channel 6</td>
<td>TIE (GT6)</td>
<td>I</td>
<td>FFED, FFEN</td>
<td>3E52, 3E53</td>
</tr>
<tr>
<td>Enh Capt Timer Channel 5</td>
<td>TIE (GT5)</td>
<td>I</td>
<td>FFED, FFEN</td>
<td>3E54, 3E55</td>
</tr>
<tr>
<td>Enh Capt Timer Channel 4</td>
<td>TIE (GT4)</td>
<td>I</td>
<td>FFED, FFEN</td>
<td>3E56, 3E57</td>
</tr>
<tr>
<td>Enh Capt Timer Channel 3</td>
<td>TIE (GT3)</td>
<td>I</td>
<td>FFED, FFEN</td>
<td>3E58, 3E59</td>
</tr>
<tr>
<td>Enh Capt Timer Channel 2</td>
<td>TIE (GT2)</td>
<td>I</td>
<td>FFED, FFEN</td>
<td>3E5A, 3E5B</td>
</tr>
<tr>
<td>Enh Capt Timer Channel 1</td>
<td>TIE (GT1)</td>
<td>I</td>
<td>FFED, FFEN</td>
<td>3E5C, 3E5D</td>
</tr>
<tr>
<td>Enh Capt Timer Channel 0</td>
<td>TIE (GT0)</td>
<td>I</td>
<td>FFED, FFEN</td>
<td>3E5E, 3E5F</td>
</tr>
<tr>
<td>Real Time</td>
<td>CRG1INT (RTIE)</td>
<td>I</td>
<td>FFFA, FFFE</td>
<td>3E70, 3E71</td>
</tr>
<tr>
<td>IRQ</td>
<td>IRQCN (IRQEN)</td>
<td>I</td>
<td>FFFA, FFFF</td>
<td>3E72, 3E73</td>
</tr>
<tr>
<td>XIRQ</td>
<td>(None)</td>
<td>(None)</td>
<td>FFFA, FFFF</td>
<td>3E74, 3E75</td>
</tr>
<tr>
<td>SWI</td>
<td>(None)</td>
<td>(None)</td>
<td>FFFA, FFFF</td>
<td>3E76, 3E77</td>
</tr>
<tr>
<td>Unimplemented Instruction</td>
<td>(None)</td>
<td>(None)</td>
<td>FFFA, FFFF</td>
<td>3E78, 3E79</td>
</tr>
<tr>
<td>COP Failure</td>
<td>COPCTL (CR2-CR6 COP Rate Select)</td>
<td>(None)</td>
<td>FFFA, FFFB</td>
<td>3E7A, 3E7B</td>
</tr>
<tr>
<td>COP Clock Monitrr Fail</td>
<td>PLLCTL (CME, SCME)</td>
<td>(None)</td>
<td>FFFA, FFFB</td>
<td>3E7C, 3E7D</td>
</tr>
<tr>
<td>Reset</td>
<td>(None)</td>
<td>(None)</td>
<td>FFFA, FFFB</td>
<td>3E7E, 3E7F</td>
</tr>
</tbody>
</table>