

- **Resets on the HC12**
- **Introduction to Interrupts on the 9S12**
- Huang Sections 6.1-6.3
- MC9S12DP256B Device User Guide
  - What happens when you reset the HC12?
  - Using the Timer Overflow Flag to implement a delay on the HC12
  - Introduction to Interrupts
  - How to generate an interrupt when the timer overflows
  - How to tell the 9S12 where the Interrupt Service Routine is located
  - Using interrupts on the HC12
  - The 9S12 registers and stack when a TOF interrupt is received
  - The 9S12 registers and stack just after a TOF interrupt is received
  - 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 **0xFFFFE and 0xFFFF**.
- Here is what is in the memory of our HCS12:

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

- 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:

PR2:0	Divide	Clock Freq	Clock Period	Overflow Period
000	1	24 MHZ	0.042 $\mu$ s	2.73 ms
001	2	12 MHZ	0.083 $\mu$ s	5.46 ms
010	4	6 MHZ	0.167 $\mu$ s	10.92 ms
011	8	3 MHZ	0.333 $\mu$ s	21.85 ms
100	16	1.5 MHZ	0.667 $\mu$ s	43.69 ms
101	32	750 MHZ	1.333 $\mu$ s	87.38 ms
110	64	375 MHZ	2.667 $\mu$ s	174.76 ms
111	128	187.5 MHZ	5.333 $\mu$ s	349.53 ms

- 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:

```
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**).

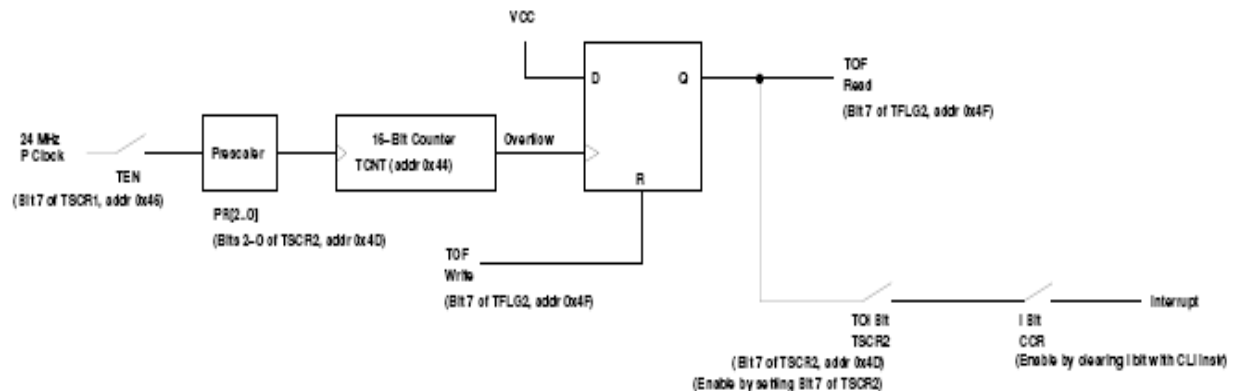
**How does HCS12 know where the ISR is located:** A set of memory locations called Interrupt Vectors tell the HCS12 the address of the ISR for each type of interrupt.

**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 DDebug-12 to change it.
- **DDebug-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 DDebug-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:

```
#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:

```
#include <hcs12.h>
#include <vectors12.h>
#include "DBug12.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:

```
#include "hcs12.inc"
#define prog $1000
    movw    #toi_isr,UserTimerOvf    ; Set interrupt vector
    movb    #$ff,DDRA
    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     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
- DDebug12 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, DDebug12 gives an error stating that the interrupt vector is uninitialized.
  - If there is anything else at these two addresses, DDebug12 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

Interrupt	Specific Mask	General Mask	Normal Vector	DBug-12 Vector
SPI2	SP2CR1 (SPIE, SPTIE)	I	FFBC, FFBD	3E3C, 3E3D
SPI1	SP1CR1 (SPIE, SPTIE)	I	FFBE, FFBF	3E3E, 3E3F
IIC	IBCR (IBIR)	I	FFC0, FFC1	3E40, 3E41
BDLC	DLBCR (IE)	I	FFC2, FFC3	3E42, 3E43
CRG Self Clock Mode	CRGINT (SCMIE)	I	FFC4, FFC5	3E44, 3E45
CRG Lock	CRGINT (LOCKIE)	I	FFC6, FFC7	3E46, 3E47
Pulse Acc B Overflow	PBCTL (PBOVI)	I	FFC8, FFC9	3E48, 3E49
Mod Down Ctr UnderFlow	MCCTL (MCZI)	I	FFCA, FFCE	3E4A, 3E4B
Port H	PTHIF (PTHIE)	I	FFCC, FFCD	3E4C, 3E4D
Port J	PTJIF (PTJIE)	I	FFCE, FFCF	3E4E, 3E4F
ATD1	ATD1CTL2 (ASCIE)	I	FFD0, FFD1	3E50, 3E51
ATD0	ATD0CTL2 (ASCIE)	I	FFD2, FFD3	3E52, 3E53
SCI1	SC1CR2 (TIE, TCIE, RIE, ILIE)	I	FFD4, FFD5	3E54, 3E55
SCIO	SC0CR2 (TIE, TCIE, RIE, ILIE)	I	FFD6, FFD7	3E56, 3E57
SPIO	SPOCR1 (SPIE)	I	FFD8, FFD9	3E58, 3E59
Pulse Acc A Edge	PACTL (PAI)	I	FFDA, FFDB	3E5A, 3E5B
Pulse Acc A Overflow	PACTL (PAOVI)	I	FFDC, FFDD	3E5C, 3E5D
Enh Capt Timer Overflow	TSCR2 (TOI)	I	FFDE, FFDF	3E5E, 3E5F
Enh Capt Timer Channel 7	TIE (C7I)	I	FFE0, FFE1	3E60, 3E61
Enh Capt Timer Channel 6	TIE (C6I)	I	FFE2, FFE3	3E62, 3E63
Enh Capt Timer Channel 5	TIE (C5I)	I	FFE4, FFE5	3E64, 3E65
Enh Capt Timer Channel 4	TIE (C4I)	I	FFE6, FFE7	3E66, 3E67
Enh Capt Timer Channel 3	TIE (C3I)	I	FFE8, FFE9	3E68, 3E69
Enh Capt Timer Channel 2	TIE (C2I)	I	FFEA, FFEB	3E6A, 3E6B
Enh Capt Timer Channel 1	TIE (C1I)	I	FFEC, FFED	3E6C, 3E6D
Enh Capt Timer Channel 0	TIE (C0I)	I	FFEE, FFEF	3E6E, 3E6F
Real Time	CRGINT (RTIE)	I	FFF0, FFF1	3E70, 3E71
IRQ	IRQCR (IRQEN)	I	FFF2, FFF3	3E72, 3E73
XIRQ	(None)	X	FFFF, FFFF	3E74, 3E75
SWI	(None)	(None)	FFF6, FFF7	3E76, 3E77
Unimplemented Instruction	(None)	(None)	FFF8, FFF9	3E78, 3E79
COP Failure	COPCTL (CR2-CR0 COP Rate Select)	(None)	FFFA, FFFB	3E7A, 3E7B
COP Clock Moniotr Fail	PLLCTL (CME, SCME)	(None)	FFFC, FFFD	3E7C, 3E7D
Reset	(None)	(None)	FFFE, FFFF	3E7E, 3E7F