Lecture 16

February 27, 2012

Interrupts on the MC9S12
The Real Time Interrupt

Interrupt vectors for the MC9S12

- 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`.
- The location of the vectors is defined in include files so you don't have to remember them or look them up in the reference manual.
  - For Assembly programs, the vectors are defined in the file `hcs12.inc`

    ```
    UserTimerOvf    equ     $3E5E
    ```

  - For C programs, the vectors are defined in the file `vectors12.h`

    ```
    #define   UserTimerOvf _VEC16(47)   /* Maps to 0x3E5E */
    ```

Commonly Used Interrupt Vectors for the MC9S12DP256

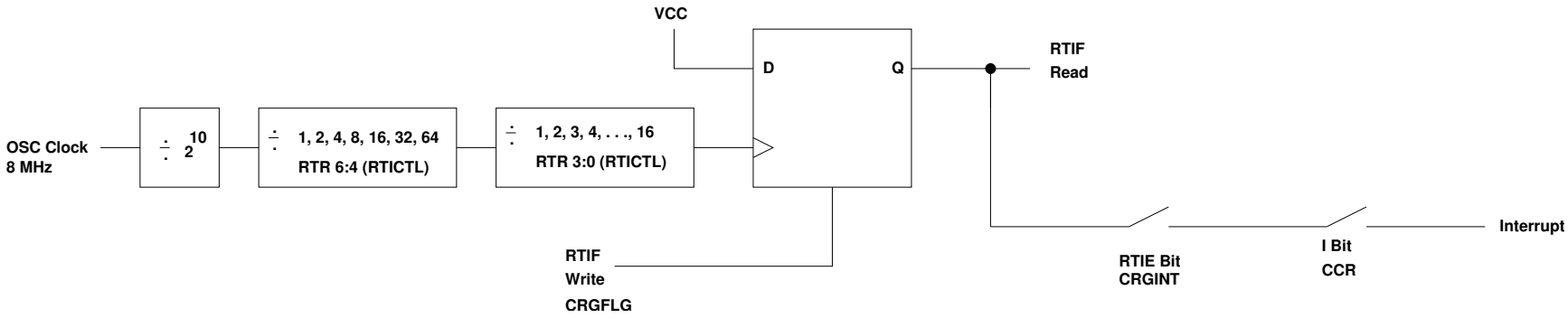
Interrupt	Specific Mask	General Mask	Normal Vector	DBug-12 Vector
SPI2	SP2CR1 (SPIE, SPTIE)	I	FFBC, FFBD	3E3C, 3E3D
SPI1	SP1CR1 (SPIE, SPTIE)	I	FFBE, FFBF	3E3E, 3E3F
IIC	IBCR (IBIR)	I	FFC0, FFC1	3E40, 3E41
BDLC	DLCBCR (IE)	I	FFC2, FFC3	3E42, 3E43
CRG Self Clock Mode	CRGINT (SCMIE)	I	FFC4, FFC5	3E44, 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, FFCB	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
SCI0	SC0CR2 (TIE, TCIE, RIE, ILIE)	I	FFD6, FFD7	3E56, 3E57
SPI0	SP0CR1 (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

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 MC9S12 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 responed to.
- The rest of the MC9S12 interrupts are masked with the I bit of the CCR.
  - All these other interrupts are also masked with a specific interrupt mask. For example, the Timer Overflow Interrupt is masked with the TOI bit of the TSCR2 register.
  - This allows you to enable any of these other interrupts you want.
  - The I bit can be set I to 1 to disable all of these interrupts if needed.

The Real Time Interrupt

- Like the Timer Overflow Interrupt, the Real Time Interrupt allows you to interrupt the processor at a regular interval.
- Information on the Real Time Interrupt is in the **CRG Block User Guide**
- There are two clock sources for MC9S12 hardware.
  - Some hardware uses the Oscillator Clock. The RTI system uses this clock.
    - For our MC9S12, the oscillator clock is 8 MHz.
  - Some hardware uses the Bus Clock. The Timer system (including the Timer Overflow Interrupt) use this clock.
    - For our MC9S12, the bus clock is 24 MHz.



- The specific interrupt mask for the Real Time Interrupt is the RTIE bit of the CRGINT register.
- When the Real Time Interrupt occurs, the RTIF bit of the CRGFLG register is set.
  - To clear the Real Time Interrupt write a 1 to the RTIF bit of the CRGFLG register.
- The interrupt rate is set by the RTR 6:4 and RTR 2:0 bits of the RTICTL register. The RTR 6:4 bits are the Prescale Rate Select bits for the RTI, and the RTR 2:0 bits are the Modulus Counter Select bits to provide additional graunularity.

RTIF	PORF	0	LOCKIF	LOCK	TRACK	SCMIF	SCM	**0x0037**	**CRGFLG**
RTIE	0	0	LOCKIE	0	0	SCMIE	0	**0x0038**	**CRGINT**
0	**RTR6**	**RTR5**	**RTR4**	**RTR3**	**RTR2**	**RTR1**	**RTR0**	**0x003B**	**RTICTL**

- To use the Real Time Interrupt, set the rate by writing to the RTR 6:4 and the RTR 3:0 bits of the RTICTL, and enable the interrupt by setting the RTIE bit of the CRGINT register
  - In the Real Time Interrupt ISR, you need to clear the RTIF flag by writing a 1 to the RTIF bit of the CRGFLG register.

- The following table shows all possible values, in **ms**, selectable by the RTICTL register (assuming the system uses a 8 MHz oscillator):

RTR 3:0	RTR 6:4							
	000 (0)	001 (1)	010 (2)	011 (3)	100 (4)	101 (5)	110 (6)	111 (7)
0000 (0)	Off	0.128	0.256	0.512	1.024	2.048	4.096	8.192
0001 (1)	Off	0.256	0.512	1.204	2.048	4.096	8.192	16.384
0010 (2)	Off	0.384	0.768	1.536	3.072	6.144	12.288	24.576
0011 (3)	Off	0.512	1.024	2.048	4.096	8.192	16.384	32.768
0100 (4)	Off	0.640	1.280	2.560	5.120	10.240	20.480	40.960
0101 (5)	Off	0.768	1.536	3.072	6.144	12.288	24.570	49.152
0110 (6)	Off	0.896	1.792	3.584	7.168	14.336	28.672	57.344
0111 (7)	Off	1.024	2.048	4.096	8.192	16.384	32.768	65.536
1000 (8)	Off	1.152	2.304	4.608	9.216	18.432	36.864	73.728
1001 (9)	Off	1.280	2.560	5.120	10.240	20.480	40.960	81.920
1010 (A)	Off	1.408	2.816	5.632	11.264	22.528	45.056	90.112
1011 (B)	Off	1.536	3.072	6.144	12.288	24.576	49.152	98.304
1100 (C)	Off	1.664	3.328	6.656	13.312	26.624	53.248	106.496
1101 (D)	Off	1.729	3.584	7.168	14.336	28.672	57.344	114.688
1110 (E)	Off	1.920	3.840	7.680	15.360	30.720	61.440	122.880
1111 (F)	Off	2.048	4.096	8.192	16.384	32.768	65.536	131.072

- Here is a C program which uses the Real Time Interrupt:

```
#include <hidef.h>          /* common defines and macros */
#include "derivative.h"     /* derivative-specific definitions */
#include "vectors12.h"      /* DBug12 RAM-based interrupt vectors */

#define enable()  __asm(cli)
#define disable() __asm(sei)

interrupt void rti_isr(void);

void main(void)
{
    disable();

    DDRB = 0xff;
    PORTB = 0;

    RTICTL = 0x63;   /* Set rate to 16.384 ms */
    CRGINT = 0x80;   /* Enable RTI interrupts */
    CRGFLG = 0x80;   /* Clear RTI Flag */
    UserRTI = (unsigned short) &rti_isr;

    enable();
    while (1)
    {
        __asm(wai);   /* Do nothing -- wait for interrupt */
    }
}

interrupt void rti_isr(void)
{
    PORTB = PORTB  + 1;
    CRGFLG = 0x80;
}
```

To display a 16 bit number on the four 7-segment LEDs, you need to display each 4-bit nibble sequentially. E.g., to display the number 0x1234, you will first have to display the "1" on the left-most 7-segment LED (turning on segments *b* and *c*) for a few milliseconds, then display the "2" on the next 7-segment LED for a few ms, then the "3", and finally the "4". An easy way to do this is to do it inside an RTI interrupt service routine. Use a static variable to keep track of which nibble to display. The following RTI interrupt service routine displays a `global` 16-bit variable called `value` on the seven-segment display

```
interrupt void rti_isr(void)
{
    static unsigned char nibble=0;
    /* Array to conver nibble to segments to turn on.  For example, 0 is
       displayed with segments a, b, c, d, e, and f, or 0011 1111
                                                          gfe dcba
     */
    const char hex2seven_seg[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D,
                                  0x7D, 0x07, 0x7F, 0x6F, 0x77, 0x7c,
                                  0x58, 0x5e, 0x79, 0x71};
    switch (nibble) {
        case 0: PTP = 0x0E;      /* Enable the left-most display 1110 */
                PTJ |= 0x02;
                PORTB = hex2seven_seg[(value>>12)&0x0F];
                break;
        case 1: PTP = 0x0D;      /* Enable the next display 1101 */
                PTJ |= 0x02;
                PORTB = hex2seven_seg[(value>>8)&0x0F];
                break;
        case 2: PTP = 0x0B;      /* Enable the next display 1011 */
                PTJ |= 0x02;
                PORTB = hex2seven_seg[(value>>4)&0x0F];
                break;
        case 3: PTP = 0x07;      /* Enable the right-most display 0111 */
                PTJ |= 0x02;
                PORTB = hex2seven_seg[(value)&0x0F];
                break;
    }
    nibble = (nibble + 1) % 4;
    CRGFLG = 0x80;               /* Clear the RTI flag */
}
```

- `digit` is declared to be `static` so its value remains between entries into `RTI_isr`
- You cannot pass a value to an interrupt service routine, so any variable from another part of the program used by the ISR must be declared as `global`
- You cannot pass a value out of an ISR, so if another part of the prgram needs a value deterimed inside an ISR, you must use a `global` variable. It must also be declared as

`volatile` so the compiler knows that its value may change outside the regular program flow.