EE4800-03
Embedded Systems Design
Lessons 7-10
- Exceptions -
Resets and Interrupts
- Exceptions -

Resets and Interrupts

- Polling vs. Interrupts
- Exceptions: Resets and Interrupts
- 68HC12 Exceptions
  - Resets
  - Interrupts: Maskable and Non-maskable
- 68HC12 Interrupt Response
- Exception Vector
- Exception Priority
- Programming an Interrupt Service Routine
- Exceptions -
Resets and Interrupts

• Polling versus interrupts
  – polling: constantly monitoring for flag to set
    • program is tied up waiting for flag
    • inefficient use of processor
  – interrupt: processor tells program when event has occurred
    • program can be executing other tasks
    • efficient use of processor
  – EX] sequentially ask question vs you ask me
- Exceptions -
Resets and Interrupts

• Resets: returns 68HC12 to known, well-defined state after detected fault
  – power-on reset
  – Computer Operating Properly (COP) reset
  – Clock Monitor reset
  – External reset

• Interrupts - planned, but unscheduled high priority event
  – non-maskable: may not be turned off by user
  – maskable: turned on and off by user with “I” bit in CCR
- Exceptions -
Resets and Interrupts

• “I” bit controlled with CLI and SEI command
  – CLI: Clear Interrupt Mask - turns interrupt system on
  – SEI: Set Interrupt Mask - turns interrupt system off

• Need to turn on specific interrupt locally
- Exceptions -
Resets and Interrupts (cont)

**Resets:**
- Power-on reset
- External reset
- Computer Operating Properly (COP) reset
- Clock monitor reset

**Interrupts**

**Non-maskable:**
- Unimplemented Instruction Trap
- Software Interrupt Instruction (SWI)
- XIRQ

**Maskable:**
- IRQ
- Real Time Interrupt
- Timer Channel
- Timer Overflow
- Pulse Accumulator Overflow
- Pulse Accumulator Edge Detect
- Serial Peripheral Interface (SPI)
- Serial Communications Interface (SCI)
- Analog-to-Digital (ATD)
- Key Wake-Up
68HC12 Interrupt Response
68HC12 Interrupt Response

- **Interrupt Vector**
  - location of ISR
  - located in upper 128 bytes of memory
  - user must tie Vector to ISR
- **Interrupt Priority**
  - determines order of execution when multiple interrupts occur
- **Interrupt Service Routine (ISR)** - user written response routine to interrupt event
## 68HC12 Interrupt Response

### Interrupt Priority

<table>
<thead>
<tr>
<th>Vector Address</th>
<th>Interrupt Source</th>
<th>CCR Mask</th>
<th>Local Enable</th>
<th>HPRIO Value to Elevate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FFFF, $FFF0</td>
<td>Reset</td>
<td>none</td>
<td>none</td>
<td>–</td>
</tr>
<tr>
<td>$FFFF, $FFF1</td>
<td>COP Clock Monitor Fail Reset</td>
<td>none</td>
<td>CME, FCME</td>
<td>–</td>
</tr>
<tr>
<td>$FFFF, $FFF2</td>
<td>COP Fail Reset</td>
<td>none</td>
<td>cop rate selected</td>
<td>–</td>
</tr>
<tr>
<td>$FFFF, $FFF3</td>
<td>Unimplemented Instruction Trap</td>
<td>none</td>
<td>none</td>
<td>–</td>
</tr>
<tr>
<td>$FFFF, $FFF4</td>
<td>SWI</td>
<td>none</td>
<td>none</td>
<td>–</td>
</tr>
<tr>
<td>$FFFF, $FFF5</td>
<td>XIRQ</td>
<td>x bit</td>
<td>none</td>
<td>–</td>
</tr>
<tr>
<td>$FFFF, $FFF6</td>
<td>IRQ or Key Wake Up D</td>
<td>1 bit</td>
<td>IRQEN, KWIED[7:0]</td>
<td>$F2</td>
</tr>
<tr>
<td>$FFFF, $FFF7</td>
<td>Real Time Interrupt</td>
<td>1 bit</td>
<td>RTIE</td>
<td>$F0</td>
</tr>
<tr>
<td>$FFFF, $FFF8</td>
<td>Timer Channel 0</td>
<td>1 bit</td>
<td>TC0</td>
<td>$EE</td>
</tr>
<tr>
<td>$FFFF, $FFF9</td>
<td>Timer Channel 1</td>
<td>1 bit</td>
<td>TC1</td>
<td>$EC</td>
</tr>
<tr>
<td>$FFFF, $FFA</td>
<td>Timer Channel 1</td>
<td>1 bit</td>
<td>TC2</td>
<td>$EA</td>
</tr>
<tr>
<td>$FFFF, $FFB</td>
<td>Timer Channel 2</td>
<td>1 bit</td>
<td>TC3</td>
<td>$E8</td>
</tr>
<tr>
<td>$FFFF, $FFC</td>
<td>Timer Channel 3</td>
<td>1 bit</td>
<td>TC4</td>
<td>$E6</td>
</tr>
<tr>
<td>$FFFF, $FFD</td>
<td>Timer Channel 4</td>
<td>1 bit</td>
<td>TC5</td>
<td>$E4</td>
</tr>
<tr>
<td>$FFFF, $FFE</td>
<td>Timer Channel 5</td>
<td>1 bit</td>
<td>TC6</td>
<td>$E2</td>
</tr>
<tr>
<td>$FFFF, $FFF</td>
<td>Timer Channel 6</td>
<td>1 bit</td>
<td>TC7</td>
<td>$E0</td>
</tr>
<tr>
<td>$FFFF, $FFG</td>
<td>Timer Channel 7</td>
<td>1 bit</td>
<td>TOI</td>
<td>$DE</td>
</tr>
<tr>
<td>$FFFF, $FFH</td>
<td>Timer Overflow</td>
<td>1 bit</td>
<td>PAOV1</td>
<td>$DC</td>
</tr>
<tr>
<td>$FFFF, $FFI</td>
<td>Pulse Accumulator Overfow</td>
<td>1 bit</td>
<td>PAII</td>
<td>$DA</td>
</tr>
<tr>
<td>$FFFF, $FFJ</td>
<td>Pulse Accumulator Input Edge</td>
<td>1 bit</td>
<td>SPI0E</td>
<td>$D8</td>
</tr>
<tr>
<td>$FFFF, $FFK</td>
<td>SPI Serial Transfer Complete</td>
<td>1 bit</td>
<td>TIE0, TIE1,</td>
<td>$D6</td>
</tr>
<tr>
<td>$FFFF, $FFL</td>
<td>SCI 0</td>
<td>1 bit</td>
<td>RIE0, ILIE0</td>
<td>$D6</td>
</tr>
<tr>
<td>$FFFF, $FFM</td>
<td>SCI 1</td>
<td>1 bit</td>
<td>TIE1, TIE1,</td>
<td>$D4</td>
</tr>
<tr>
<td>$FFFF, $FFO</td>
<td>SCI 2</td>
<td>1 bit</td>
<td>RIE1, ILIE1</td>
<td>$D4</td>
</tr>
<tr>
<td>$FFFF, $FFP</td>
<td>SCI 3</td>
<td>1 bit</td>
<td>ADIE</td>
<td>$D2</td>
</tr>
<tr>
<td>$FFFF, $FFQ</td>
<td>SCI 4</td>
<td>1 bit</td>
<td>$D0</td>
<td></td>
</tr>
<tr>
<td>$FFFF, $FFR</td>
<td>SCI 5</td>
<td>1 bit</td>
<td>$CE</td>
<td></td>
</tr>
<tr>
<td>$FFFF, $FFS</td>
<td>SCI 6</td>
<td>1 bit</td>
<td>$B0–$CC</td>
<td></td>
</tr>
</tbody>
</table>

"Copyright of Motorola, Used by Permission"

Revised: Dec 15, 2003
Programming an Interrupt Service Routine

- Determine how interrupt is enabled
  - global: CLI
  - local enable bit
- Initialize Vector Table
  - directive approach
  - EVB SetUserVector
- Initialize Stack
- Enable interrupt
- Write the specific ISR
Programming an Interrupt Service Routine - an example

Initialize the microprocessor for the interrupt.
  • Initialize the stack - this is done through compiler settings
  • Initialize any other necessary systems on the HC12.
  • Initialize the interrupt vector table. You will need to use a special header file and code.
    – Header file is on your computer and is called `abbie.h` (change name).
  • Code to set up your function to be an interrupt service routine will be similar to the following:

    *This part declares your function as an interrupt service routine.*
    
    #pragma interrupt_handler toggle_isr

Revised: Dec 15, 2003
Programming an
Interrupt Service Routine - an example

• This part fills the appropriate vector with the address of your interrupt service routine.

    #pragma abs_address: 0x0B2A
    void (*Timer_Channel_2_interrupt_vector[])( )={toggle_isr};
    #pragma end_abs_address

• Make sure the interrupt you will be using is cleared to start.
• Initialize local interrupts.
• Initialize the interrupt system using the CLI command (With this header file use CLI(); ). Do this step last so that you aren’t inadvertently setting off interrupts before you finish initializing the system.
• Write the interrupt service routine to handle the interrupt.
## Programming an ISR - an example

### Table 1. RAM Interrupt Vectors

<table>
<thead>
<tr>
<th>Interrupt Name</th>
<th>RAM Vector Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDLC (Key Wakeup J)</td>
<td>$0B10, $0B11</td>
</tr>
<tr>
<td>ATD</td>
<td>$0B12, $0B13</td>
</tr>
<tr>
<td>SCI</td>
<td>$0B16, $0B17</td>
</tr>
<tr>
<td>SPI</td>
<td>$0B18, $0B19</td>
</tr>
<tr>
<td>Pulse Accumulator Input Edge</td>
<td>$0B1A, $0B1B</td>
</tr>
<tr>
<td>Pulse Accumulator Overflow</td>
<td>$0B1C, $0B1D</td>
</tr>
<tr>
<td>Timer Overflow</td>
<td>$0B1E, $0B1F</td>
</tr>
<tr>
<td>Timer Channel 7</td>
<td>$0B20, $0B21</td>
</tr>
<tr>
<td>Timer Channel 6</td>
<td>$0B22, $0B23</td>
</tr>
<tr>
<td>Timer Channel 5</td>
<td>$0B24, $0B25</td>
</tr>
<tr>
<td>Timer Channel 4</td>
<td>$0B26, $0B27</td>
</tr>
<tr>
<td>Timer Channel 3</td>
<td>$0B28, $0B29</td>
</tr>
<tr>
<td>Timer Channel 2</td>
<td>$0B2A, $0B2B</td>
</tr>
<tr>
<td>Timer Channel 1</td>
<td>$0B2C, $0B2D</td>
</tr>
<tr>
<td>Timer Channel 0</td>
<td>$0B2E, $0B2F</td>
</tr>
<tr>
<td>Real Time Interrupt</td>
<td>$0B30, $0B31</td>
</tr>
<tr>
<td>IRQ</td>
<td>$0B32, $0B33</td>
</tr>
<tr>
<td>XIRQ</td>
<td>$0B34, $0B35</td>
</tr>
<tr>
<td>SWI</td>
<td>$0B36, $0B37</td>
</tr>
<tr>
<td>Unimplemented Instruction Trap</td>
<td>$0B38, $0B39</td>
</tr>
<tr>
<td>COP Failure</td>
<td>$0B3A, $0B3B</td>
</tr>
<tr>
<td>COP Clock Monitor Fail Reset</td>
<td>$0B3C, $0B3D</td>
</tr>
<tr>
<td>Reset</td>
<td>$0BEF, $0BFF</td>
</tr>
</tbody>
</table>

Revised: Dec 15, 2003
Programming an Interrupt Service Routine - an example

• Example] In this task you will need to simultaneously generate two square waves with different frequencies. For one wave use the month and day of your birthday and for the second use the month and day of your Lab TA’s birthday. Verify that the waves are being generated simultaneously and that they have different frequencies with the oscilloscope.
Programming an Interrupt Service Routine - an example

#include <abbie.h>

void toggle1_isr(void); //function prototype
void toggle2_isr(void);
#pragma interrupt_handler toggle1_isr //define as interrupt
#pragma interrupt_handler toggle2_isr

#pragma abs_address: 0x0B28
void (*Timer_Channel_3_interrupt_vector[])(())={toggle2_isr};
void (*Timer_Channel_2_interrupt_vector[])(())={toggle1_isr};
#pragma end_abs_address
Programming an Interrupt Service Routine - an example

```c
void initialize(void); // Define function initialize

void main(void){
    initialize(); // Initialize the timer system
    TMSK1 = 0x0C;
    TFLG1 = 0xFF;
    CLI(); // Initialize interrupts

    while(1) // Continuous loop
    {
        ; // Wait for interrupts
    }
}
```
Programming an Interrupt Service Routine - an example

/* Function: initialize: enables the timer and sets up the M-Clk */

void initialize(){
    CLKCTL = 0x02;   // Set M-clock to divide by 4 (2 MHz)
                    // CPU master clock divider ($0047)
    TMSK2 = 0x00;    // Disable TOI, Prescale = 0;
    TIOS = 0x0C;     // Make OS2 output compare
    TSCR = 0x80;     // Enable the timer
    TCTL2 = 0x50;
}

Revised: Dec 15, 2003
Programming an
Interrupt Service Routine - an example

void toggle1_isr(void){
    TFLG1 = 0x04;
    TC2 += 9091;
}

void toggle2_isr(void)
{
    TFLG1 = 0x08;
    TC3 += 4854;
}
Real Time Interrupts

• Reminds processor to perform required actions on a regular basis.

• Two key registers:
  – RTI Control Register (RTICTL): used to enable RTI and set interrupt rate
    • RTIE: (1) to enable
    • RTR[2:0] to set interrupt rate
  – RTI Flag Register (RTIFLG): bit 7 RTIF
    • Reset by writing “1” to RTIF
# Real Time Interrupts

Register: Real-Time Interrupt Control Register (RTICTL)  
Address: $0014

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTIE</td>
<td>RSWA</td>
<td>RSBCK</td>
<td>0</td>
<td>RTBYF</td>
<td>TIR</td>
<td>RTR1</td>
<td>RTR0</td>
</tr>
</tbody>
</table>

Reset: 0 0 0 0 0 0 0 0

Register: Real-Time Interrupt Flag Register  
Address: $0015

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTIF</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Reset: 0 0 0 0 0 0 0 0

<table>
<thead>
<tr>
<th>RTR[2:0]</th>
<th>Divide by</th>
<th>$E = 40$ MHz Timeout</th>
<th>$E = 80$ MHz Timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>off</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>001</td>
<td>$2^{13}$</td>
<td>2.048 ms</td>
<td>1.024 ms</td>
</tr>
<tr>
<td>010</td>
<td>$2^{14}$</td>
<td>4.096 ms</td>
<td>2.048 ms</td>
</tr>
<tr>
<td>011</td>
<td>$2^{15}$</td>
<td>8.192 ms</td>
<td>4.096 ms</td>
</tr>
<tr>
<td>100</td>
<td>$2^{16}$</td>
<td>16.384 ms</td>
<td>8.192 ms</td>
</tr>
<tr>
<td>101</td>
<td>$2^{17}$</td>
<td>32.768 ms</td>
<td>16.384 ms</td>
</tr>
<tr>
<td>110</td>
<td>$2^{18}$</td>
<td>65.536 ms</td>
<td>32.768 ms</td>
</tr>
<tr>
<td>111</td>
<td>$2^{19}$</td>
<td>131.072 ms</td>
<td>65.536 ms</td>
</tr>
</tbody>
</table>
Real Time Interrupts

- Initialize Program
  - set RTI prescale factor RTR[2:0] and enable RTI with RTIE bit in the RTICTL register
  - enable maskable interrupts (CLI)

  clear RTIF flag in RTIFLG register
  increment 8.196 millisecond counter

  ms counter = 122?
  no
  yes

  update variables

  RTI
Real Time Interrupts

/*-----------------------------------------------*/
/*MAIN PROGRAM: This program keeps track of clock time using the Real Time Management Interrupt (RTI). The RTI generates an interrupt every 8.192 ms. The RTI_isr keeps track of elapsed time.*/
/*-----------------------------------------------*/

/*include files*/
#include<912b32.h>

/*function prototypes*/
void RTI_isr(void); /*Real Time Interrupt - ISR*/

/* interrupt pragma */
#pragma interrupt_handler RTI_isr

/*initialize vector table*/
#pragma abs_address: 0xF7F0
void (*RTI_interrupt_vector[])()={RTI_isr};
#pragma end_abs_address
Real Time Interrupts

/*global variables*/
unsigned int ms_ctr, sec_ctr, mins_ctr, hrs_ctr, days_ctr;

void main(void){
    ms_ctr = 0;                          /*initialize timer variables*/
    sec_ctr = 0;
    mins_ctr = 0;
    hrs_ctr = 0;
    days_ctr = 0;

    RTICTL = 0x84;                         /*Enable RTI int, 8.196ms RTI*/
    CLI();                                 /*Initialize interrupts*/

    while(1)
    {
        ;                                    /*wait for interrupt*/
    }
}

/*Function: RTI_isr: RTI interrupt occurs every 8.196 ms */
Real Time Interrupts

/**--------------------------------------------------------------------------*/
/*Function: RTI_isr: RTI interrupt occurs every 8.196 ms */
/*--------------------------------------------------------------------------*/

void RTI_isr(void){
    RTIFLG = 0x80;                         /*reset RTI Interrupt Flag*/

    /*update milliseconds*/
    ms_ctr = ms_ctr+1;                     /*increment ms counter */

    /*update seconds*/
    if(ms_ctr == 122)                      /*counter equates to 1000 ms at 122*/
    {
        ms_ctr = 0;                         /*reset millisecond counter*/
        sec_ctr = sec_ctr +1;               /*increment seconds counter*/
    }
}
Real Time Interrupts

/*update minutes*/
if(sec_ctr == 60)
{
    sec_ctr = 0;            /*reset seconds counter*/
    mins_ctr = minsCtr + 1; /*increment minutes counter*/
}

/*update hours*/
if(mins_ctr == 60)
{
    mins_ctr = 0;           /*reset minutes counter*/
    hrs_ctr = hrsCtr + 1;   /*increment hours counter*/
}

/*update days*/
if(hrs_ctr == 24)
{
    hrs_ctr = 0;           /*reset hours counter*/
    days_ctr = days_ctr + 1; /*increment days counter*/
}
}
Multiple Interrupts

• Allows multiple events to occur “simultaneously”
• Interrupt Priority
  – HC12 shuts off interrupt system during ISR
  – May want to manually re-enable to allow system to respond to higher priority events
• Must carefully study interaction of interrupts
• Very difficult to troubleshoot malfunctioning system