EE4390 Microprocessors

Lessons 23, 24
- 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)

- Exceptions -

- 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

Revised: Aug 1, 2003
68HC12 Interrupt Response

interrupt service routine
fetch → decode
execute
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>8FFFE, 8FFF</td>
<td>Reset</td>
<td>none</td>
<td>none</td>
<td>–</td>
</tr>
<tr>
<td>8FFFC, 8FFD</td>
<td>COP Clock Monitor Fail Reset</td>
<td>none</td>
<td>CME, FCME</td>
<td>–</td>
</tr>
<tr>
<td>8FFFA, 8FFB</td>
<td>COP Fail Reset</td>
<td>none</td>
<td>cop rate selected</td>
<td>–</td>
</tr>
<tr>
<td>8FF8, 8FF9</td>
<td>Unimplemented Instruction Trap</td>
<td>none</td>
<td>none</td>
<td>–</td>
</tr>
<tr>
<td>8FF6, 8FF7</td>
<td>SWI</td>
<td>none</td>
<td>none</td>
<td>–</td>
</tr>
<tr>
<td>8FF4, 8FF5</td>
<td>XIRQ</td>
<td>x bit</td>
<td>none</td>
<td>–</td>
</tr>
<tr>
<td>8FF2, 8FF3</td>
<td>IRQ or Key Wake Up D</td>
<td>1 bit</td>
<td>IRQEN, KWIED[7:0]</td>
<td>$F2</td>
</tr>
<tr>
<td>8F0, 8F1</td>
<td>Real Time Interrupt</td>
<td>1 bit</td>
<td>RTIE</td>
<td>$F0</td>
</tr>
<tr>
<td>8FEE, 8FEF</td>
<td>Timer Channel 0</td>
<td>1 bit</td>
<td>TC0</td>
<td>$EE</td>
</tr>
<tr>
<td>8FEC, 8FED</td>
<td>Timer Channel 1</td>
<td>1 bit</td>
<td>TC1</td>
<td>$EC</td>
</tr>
<tr>
<td>8FEA, 8FEB</td>
<td>Timer Channel 2</td>
<td>1 bit</td>
<td>TC2</td>
<td>$EA</td>
</tr>
<tr>
<td>8FE8, 8FE9</td>
<td>Timer Channel 3</td>
<td>1 bit</td>
<td>TC3</td>
<td>$E8</td>
</tr>
<tr>
<td>8FE6, 8FE7</td>
<td>Timer Channel 4</td>
<td>1 bit</td>
<td>TC4</td>
<td>$E6</td>
</tr>
<tr>
<td>8FE4, 8FE5</td>
<td>Timer Channel 5</td>
<td>1 bit</td>
<td>TC5</td>
<td>$E4</td>
</tr>
<tr>
<td>8FE2, 8FE3</td>
<td>Timer Channel 6</td>
<td>1 bit</td>
<td>TC6</td>
<td>$E2</td>
</tr>
<tr>
<td>8FE0, 8FE1</td>
<td>Timer Channel 7</td>
<td>1 bit</td>
<td>TC7</td>
<td>$E0</td>
</tr>
<tr>
<td>8FDE, 8FDF</td>
<td>Timer Overflow</td>
<td>1 bit</td>
<td>TOI</td>
<td>$DE</td>
</tr>
<tr>
<td>8FFD, 8FDD</td>
<td>Pulse Accumulator Overflow</td>
<td>1 bit</td>
<td>PAOV1</td>
<td>$DC</td>
</tr>
<tr>
<td>8FFDA, 8FFDB</td>
<td>Pulse Accumulator Input Edge</td>
<td>1 bit</td>
<td>PAI</td>
<td>$DA</td>
</tr>
<tr>
<td>8FDD8, 8FDD9</td>
<td>SPI Serial Transfer Complete</td>
<td>1 bit</td>
<td>SPI0E</td>
<td>$D8</td>
</tr>
<tr>
<td>8FDD6, 8FDD7</td>
<td>SCI 0</td>
<td>1 bit</td>
<td>TIE0, TIE0,</td>
<td>$D6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RIE0, ILIE0</td>
<td></td>
</tr>
<tr>
<td>8FDD4, 8FDD5</td>
<td>SCI 1</td>
<td>1 bit</td>
<td>TIE1, TIE1,</td>
<td>$D4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RIE1, ILIE1</td>
<td></td>
</tr>
<tr>
<td>8FDD2, 8FDD3</td>
<td>ATD</td>
<td>1 bit</td>
<td>ADIE</td>
<td>$D2</td>
</tr>
<tr>
<td>8FDD0, 8FDD1</td>
<td>Key Wakeup J (stop wakeup)</td>
<td>1 bit</td>
<td>KWIEJ[7:0]</td>
<td>$D0</td>
</tr>
<tr>
<td>8FFCE, 8FFCF</td>
<td>Key Wakeup H (stop wakeup)</td>
<td>1 bit</td>
<td>KWIEH[7:0]</td>
<td>$CE</td>
</tr>
<tr>
<td>8FF80–8FFCD</td>
<td>Reserved</td>
<td>1 bit</td>
<td>$B0–$CC</td>
<td></td>
</tr>
</tbody>
</table>

"Copyright of Motorola, Used by Permission"

Revised: Aug 1, 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 `bbie.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`
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.
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: Aug 1, 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.
#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
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: Aug 1, 2003
Programming an Interrupt Service Routine - an example

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

void toggle2_isr(void) {
    TFLG1 = 0x08;
    TC3 += 4854;
}
```