Working in teams,
“Everything you think is blindingly obvious is wrong.”
Tim Fields, UT EE grad, Lead Designer for Brute Force at Microsoft

Recap
Parameter passing using the stack
LCD programming
Fixed-point conversions

Overview
I/O synchronization
Interrupts
Output compare periodic interrupts

Read Book Sections 9.1, 9.2, 9.4, 9.6.1, 9.6.2

Blind Cycle Counting Synchronization
Blind cycle counting is appropriate when the I/O delay is fixed and known. This type of synchronization is blind because it provides no feedback from the I/O back to the computer.

Gadfly or Busy Waiting Synchronization
Check busy/ready flag over and over until it is ready

Interrupt Synchronization
Request interrupt when busy/ready flag is ready
**Synchronizing with an input device**

**Blind Cycle**
- Wait a fixed time
- Read data

**Busy-Wait**
- Busy
  - Status
  - Ready
- Read data

**Interrupt**
- Empty
  - Fifo
- Read data
- Get data from Fifo
- Put data in Fifo
- return from interrupt

**Synchronizing with an output device**

**Blind Cycle**
- Write data
  - Wait a fixed time

**Busy-Wait**
- Busy
  - Status
  - Ready
- Write data

**Interrupt**
- Full
  - Fifo
- Get data from Fifo
- Put data into Fifo
- Write data
- return from interrupt

**What are interrupts?**

An **interrupt** is the automatic transfer of software execution in response to hardware that is asynchronous with current software execution. An **interrupt** may be caused by an external I/O device (like a keyboard or printer) or an internal event (like an op code fault, or a periodic timer.) occurs the hardware needs service (busy to done state transition)

A **thread** is defined as the path of action of software as it executes. A **background thread** interrupt service routine is called. a new background thread is created for each interrupt request. local variables and registers used in the interrupt service routine are unique threads share globals
Each potential interrupt source has a separate **arm** bit. E.g., **C0I**
Set arm bits for those devices from which it wishes to accept interrupts,
Deactivate arm bits in those devices from which interrupts are not allowed

Each potential interrupt source has a separate **flag** bit. E.g., **C0F**
hardware sets the flag when it wishes to request an interrupt
software clears the flag in ISR to signify it is processing the request

Interrupt **enable** bit, I, which is in the condition code register.
   enable all armed interrupts by setting I=0, or **cli**
   disable all interrupts by setting I=1. **sei**
I=1 does not dismiss the interrupt requests, rather it postpones

**Three conditions must be true simultaneously for an interrupt to occur:**
   1) Initialization software will set the arm bit. e.g., **C0I**
      individual control bit for each possible flag that can interrupt
   2) When it is convenient, the software will enable, **I=0**
      allow all interrupts now
   3) Hardware action (busy to done) sets a flag  e.g., **C0F**
      new input data ready,
      output device idle,
      periodic,
      alarm

**What happens when an interrupt is processed?**
   1) the execution of the main program is suspended
      the current instruction is finished,
      pushes registers on the stack
      sets the I bit
      gets the vector address from high memory
   2) the interrupt service routine (ISR), or background thread is executed,
      clears the flag that requested the interrupt
      performs necessary operations
      communicates using global variables
   3) the main program is resumed when ISR executes **rti**.
      pulls the registers from the stack
Software must respond to events within a prescribed time.

Software latency or interface latency
- Time from when new input is ready until time software reads data.
- Time from when output is idle until time software writes new data.
- Execute tasks at periodic intervals
- Interrupts guarantee an upper bound on the software response time
  - Count maximum time running with $I=1$, plus
  - Time to process the interrupt.

Respond to infrequent but important events.
- Alarm conditions like low battery power and
- Error conditions can be handled with interrupts.

Periodic interrupts, generated by the timer at a regular rate
- Clocks and timers
- Computer-based data input/output
  - DAC used play music
  - ADC used to acquire data
- Digital control systems.
**Increase the overall bandwidth**

Situations where system is doing many tasks
  Buffer the data, spend less time waiting.
  Small packets use a **first in first out queue**
  **Fifo_Put** saves data in FIFO
  **Fifo_Get** removes data from FIFO
  Large blocks use a **double buffer**

**What type of situations lend themselves to interrupt solutions?**

<table>
<thead>
<tr>
<th>Busy-wait</th>
<th>Interrupts</th>
<th>DMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicable</td>
<td>Variable arrival times</td>
<td>low latency</td>
</tr>
<tr>
<td>Simple I/O</td>
<td>Complex I/O, different speeds</td>
<td>high bandwidth</td>
</tr>
<tr>
<td>Fixed load</td>
<td>Variable load</td>
<td></td>
</tr>
<tr>
<td>Dedicated, single thread</td>
<td>Other functions to do</td>
<td></td>
</tr>
<tr>
<td>Single process</td>
<td>Multithread or multiprocess</td>
<td></td>
</tr>
<tr>
<td>Nothing else to do</td>
<td>Infrequent but important alarms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Program errors</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Overflow, invalid op code</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Illegal stack or memory access</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Machine errors</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Power failure, memory fault</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Breakpoints for debugging</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Real time clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data acquisition and control</td>
<td></td>
</tr>
</tbody>
</table>

**9S12DP512 interrupts we will be using**

0xFFD4  **interrupt 21 SCI1**
0xFFD6  interrupt 20 SCI0
0xFFDE  interrupt 16 timer overflow
0xFFE0  interrupt 15 timer channel 7
0xFFE2  interrupt 14 timer channel 6
0xFFE4  interrupt 13 timer channel 5
0xFFE6  interrupt 12 timer channel 4
0xFFE8  **interrupt 11 timer channel 3**
0xFFEA  interrupt 10 timer channel 2
0xFFEC  interrupt 9  timer channel 1
0xFFEE interrupt 8 timer channel 0
0xFFF0 interrupt 7 RTI real time interrupt
0xFFF6 interrupt 4 SWI software interrupt
0xFFF8 interrupt 3 trap software interrupt
0xFFF0 interrupt 0 reset

(The interrupt number is used by the Metrowerks C compiler)

MC9S12 Periodic Interrupt using Output Compare 0
(OC.rtf OC.uc OC.io files created with TExaS installation)

<table>
<thead>
<tr>
<th>address</th>
<th>msb</th>
<th>lsb</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0044</td>
<td>15</td>
<td>14</td>
<td>TCNT</td>
</tr>
<tr>
<td>$0050</td>
<td>15</td>
<td>14</td>
<td>TC0</td>
</tr>
<tr>
<td>$0052</td>
<td>15</td>
<td>14</td>
<td>TC1</td>
</tr>
<tr>
<td>$0054</td>
<td>15</td>
<td>14</td>
<td>TC2</td>
</tr>
<tr>
<td>$0056</td>
<td>15</td>
<td>14</td>
<td>TC3</td>
</tr>
<tr>
<td>$0058</td>
<td>15</td>
<td>14</td>
<td>TC4</td>
</tr>
<tr>
<td>$005A</td>
<td>15</td>
<td>14</td>
<td>TC5</td>
</tr>
<tr>
<td>$005C</td>
<td>15</td>
<td>14</td>
<td>TC6</td>
</tr>
<tr>
<td>$005E</td>
<td>15</td>
<td>14</td>
<td>TC7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>address</th>
<th>Bit 7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>Bit 0</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0046</td>
<td>TOI</td>
<td>0</td>
<td>0</td>
<td>TSWAI</td>
<td>TSBCK</td>
<td>TFFCA</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$004D</td>
<td>IOS7</td>
<td>IOS6</td>
<td>IOS5</td>
<td>IOS4</td>
<td>IOS3</td>
<td>IOS2</td>
<td>IOS1</td>
<td>IOS0</td>
<td>TIOS</td>
</tr>
<tr>
<td>$004C</td>
<td>C71</td>
<td>C61</td>
<td>C51</td>
<td>C41</td>
<td>C31</td>
<td>C21</td>
<td>C11</td>
<td>C01</td>
<td>TIE</td>
</tr>
<tr>
<td>$004E</td>
<td>C7F</td>
<td>C6F</td>
<td>C5F</td>
<td>C4F</td>
<td>C3F</td>
<td>C2F</td>
<td>C1F</td>
<td>C0F</td>
<td>TFLG1</td>
</tr>
<tr>
<td>$004F</td>
<td>TOF</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>TFLG2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Bit 7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>Bit 0</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$005B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>TSCR2</td>
</tr>
<tr>
<td>$005D</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>TSCR3</td>
</tr>
<tr>
<td>$005F</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>TSCR4</td>
</tr>
<tr>
<td>$0060</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>TSCR5</td>
</tr>
<tr>
<td>$0061</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>TSCR6</td>
</tr>
<tr>
<td>$0062</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>TSCR7</td>
</tr>
<tr>
<td>$0063</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>TSCR8</td>
</tr>
</tbody>
</table>

The rate is dependent of the PLL, TSCR2, and 1000 in ISR

; Bottom three bits of TSCR2 (PR2,PR1,PR0)
; determine TCNT period
; divide BootMode (24MHz) Run Mode (8MHz)
;000 1 42ns TOF 2.73ms 125ns TOF 8.192ms
;001 2 84ns TOF 5.46ms 250ns TOF 16.384ms
;010 4 167ns TOF 10.9ms 500ns TOF 32.768ms
;011 8 333ns TOF 21.8ms 1us TOF 65.536ms
;100 16 667ns TOF 43.7ms 2us TOF 131.072ms

Jonathan W. Valvano
Things you must do in every interrupt service routine
Acknowledgment (clear flag that requested the interrupt)

Things you must do in an OC interrupt service routine
Acknowledgment (make C0F become zero)
Set the timer to specify when to interrupt next

;interruption every 1000 TCNT cycles
;every 1ms, assuming TCNT at 1us
TC0handler

```c
void interrupt 8 TC0handler(void) {

    //this stuff gets executed every 1ms
    ldaa PTP
    eora #$80 ;toggle PP7
    staa PTP
    rti

    //set up next
    ldd TC0
    addd #1000
    std TC0 ; time for next interrupt
    movb #$01,TFLG1 ;ack, clear C0F

    //this stuff gets executed every 1ms
    ldaa PTP
    eora #$80 ;toggle PP7
    staa PTP
    rti

    ; OC0_ISR
    Acknowledge
    set up next
    toggle PP7
    RTI
`
TC0 = TC0+1000;
TFLG1 = 0x01;
PTP ^= 0x80;
}

All our vectors go at the bottom of code
org  $FFEE
fdb  TC0handler ;OC0 interrupt vector
org  $FFFE
fdb  Main       ;reset vector

Things you must do in every ritual
Arm (specify a flag may interrupt)
Enable (allow all interrupts on the 9S12)

Things you must do in an OC ritual
Turn on TCNT (TEN=1)
Set channel to output compare (TIOS)
Specify TCNT rate (TSCR2, PACTL, PLL)
Arm (C0I=1)
When to generate first interrupt
Enable (I=0)

Main lds  #$4000
movb  #$80,TSCR1   ;enable TCNT
bset TIOS,#$01    ;activate OC0
bset TIE,#$01     ;arm OC0
movb  #$03,TSCR2   ;1us TCNT
ldd  TCNT
addd #25
std  TC0          ;C0F set in 25 us
bset DDRP,#$80 ;used for testing
bset DDRT,#$80 ;used for testing
cli            ;enable

loop  ldaa PTT
     eora #$80        ;toggle PT7
     staa PTT
     bra  loop

void main(void){
    TSCR1 = 0x80;   // enable TCNT
    TIOS |= 0x01;   // activate OC0
    TIE  |= 0x01;   // arm OC0
    TSCR2 = 0x03;   // 1us TCNT
    TCO0 = TCNT+25; // C0F set in 25 us
    DDRP |= 0x80;   // used for testing
    DDRT |= 0x80;   // used for testing
    asm cli        // enable
    for(;;){
        PTT ^= 0x80; // toggle PT7
    }
}

In TExaS, open OC.rtf example, paste in, and look at scope
Switch to real mode, assemble, see scope signals

**The bottom line**

**Interrupts are hardware-triggered software action**

**Three conditions cause interrupt**

- Enabled (I=0) by software
- Armed (C0I=1) by software
- Triggered (C0F=1) by hardware

**Context switch**

**Finish instruction**
Push registers on stack (with I=0)
Disable (I=1)
Vector fetch into PC

ISR programming
Acknowledgment (make the flag 0 become 0)
Do something appropriate
Pass data in and out via global memory
Return to main program (rti instruction)