CS/ECE 5780/6780
Embedded Systems Design

Lecture 7, part 2: Exceptions and Interrupts

Thomas Schmid
thomas.schmid@utah.edu

February 5, 2013
Announcements

• Are we using C, or ARM or Thumb?
• What’s the difference between ARM & Thumb?
  – ARM (assembly): always 32-bit
  – Thumb (assembly): always 16-bit
  – Thumb-2 (assembly): Mixed 16 and 32-bit
• Who clears interrupt, software or hardware? How to find out?
  – Read the datasheet.
System Timer Register Map

Table 17-1 • System Timer Register Map

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Address</th>
<th>R/W</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIM1_VAL (TIMx_VAL)</td>
<td>0x40005000</td>
<td>R</td>
<td>0x0</td>
<td>Current value of Timer 1</td>
</tr>
<tr>
<td>TIM1_LOADVAL (TIMx_LOADVAL)</td>
<td>0x40005004</td>
<td>R/W</td>
<td>0x0</td>
<td>Load value for Timer 1</td>
</tr>
<tr>
<td>TIM1_BGLOADVAL (TIMx_BGLOADVAL)</td>
<td>0x40005008</td>
<td>R/W</td>
<td>0x0</td>
<td>Background load value for Timer 1</td>
</tr>
<tr>
<td>TIM1_CTRL (TIMx_CTRL)</td>
<td>0x4000500C</td>
<td>R/W</td>
<td>0x0</td>
<td>Timer 1 Control register</td>
</tr>
<tr>
<td>TIM1_RIS (TIMx_RIS)</td>
<td>0x40005010</td>
<td>R/W</td>
<td>0x0</td>
<td>Timer 1 raw interrupt status</td>
</tr>
<tr>
<td>TIM1_MIS (TIMx_MIS)</td>
<td>0x40005014</td>
<td>R</td>
<td>0x0</td>
<td>Timer 1 masked interrupt status</td>
</tr>
<tr>
<td>TIM2_VAL (TIMx_VAL)</td>
<td>0x40005018</td>
<td>R</td>
<td>0x0</td>
<td>Current value of Timer 2</td>
</tr>
<tr>
<td>TIM2_LOADVAL (TIMx_LOADVAL)</td>
<td>0x4000501C</td>
<td>R/W</td>
<td>0x0</td>
<td>Load value for Timer 2</td>
</tr>
<tr>
<td>TIM2_BGLOADVAL (TIMx_BGLOADVAL)</td>
<td>0x40005020</td>
<td>R/W</td>
<td>0x0</td>
<td>Background load value for Timer 2</td>
</tr>
<tr>
<td>TIM2_CTRL (TIMx_CTRL)</td>
<td>0x40005024</td>
<td>R/W</td>
<td>0x0</td>
<td>Timer 2 Control register</td>
</tr>
<tr>
<td>TIM2_RIS (TIMx_RIS)</td>
<td>0x40005028</td>
<td>R/W</td>
<td>0x0</td>
<td>Timer 2 raw interrupt status</td>
</tr>
<tr>
<td>TIM2_MIS (TIMx_MIS)</td>
<td>0x4000502C</td>
<td>R</td>
<td>0x0</td>
<td>Timer 2 masked interrupt status</td>
</tr>
<tr>
<td>TIM64_VAL_U</td>
<td>0x40005030</td>
<td>R</td>
<td>0x0</td>
<td>Upper 32-bit word in 64-bit mode</td>
</tr>
<tr>
<td>TIM64_VAL_L</td>
<td>0x40005034</td>
<td>R</td>
<td>0x0</td>
<td>Lower 32-bit word in 64-bit mode</td>
</tr>
<tr>
<td>TIM64_LOADVAL_U</td>
<td>0x40005038</td>
<td>R/W</td>
<td>0x0</td>
<td>Upper 32-bit load value word in 64-bit mode</td>
</tr>
<tr>
<td>TIM64_LOADVAL_L</td>
<td>0x4000503C</td>
<td>R/W</td>
<td>0x0</td>
<td>Lower 32-bit load value word in 64-bit mode</td>
</tr>
<tr>
<td>TIM64_BGLOADVAL_U</td>
<td>0x40005040</td>
<td>R/W</td>
<td>0x0</td>
<td>Upper 32-bit background load value in 64-bit mode</td>
</tr>
<tr>
<td>TIM64_BGLOADVAL_L</td>
<td>0x40005044</td>
<td>R/W</td>
<td>0x0</td>
<td>Lower 32-bit background load value in 64-bit mode</td>
</tr>
<tr>
<td>TIM64_CTRL</td>
<td>0x40005048</td>
<td>R/W</td>
<td>0x0</td>
<td>Control register in 64-bit mode</td>
</tr>
<tr>
<td>TIM64_RIS</td>
<td>0x4000504C</td>
<td>R/W</td>
<td>0x0</td>
<td>Raw interrupt status in 64-bit mode</td>
</tr>
<tr>
<td>TIM64_MIS</td>
<td>0x40005050</td>
<td>R</td>
<td>0x0</td>
<td>Masked interrupt status in 64-bit mode</td>
</tr>
<tr>
<td>TIM64_MODE</td>
<td>0x40005054</td>
<td>R/W</td>
<td>0x0</td>
<td>System Timer dual 32-bit or 64-bit mode</td>
</tr>
</tbody>
</table>
## Timer x Control Register

### Table 17-5 • TIMx_CTRL

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Name</th>
<th>R/W</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:3</td>
<td>Reserved</td>
<td>R/W</td>
<td>0x0</td>
<td>Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.</td>
</tr>
</tbody>
</table>
| 2          | TIMxINTEN   | R/W | 0x0         | Timer x Interrupt Enable. When the counter reaches zero, an interrupt is signaled to the Cortex-M3 Nested Vectored Interrupt Controller; IRQ20 for Timer x, IRQ21 for Timer 2.  
0 = Timer x interrupt disabled  
1 = Timer x interrupt enabled  
Writing this register while the System Timer is set to 64-bit mode has no effect. Reading this register while the System Timer is set to 64-bit mode returns the reset value. |
| 1          | TIMxMODE    | R/W | 0x0         | Timer x Mode.  
0 = Timer x in Periodic Mode. If TIMxENABLE = 1 when the counter reaches zero the counter is reloaded from the value in the TIMxLOADVAL register and begins counting down immediately.  
1 = Timer x in One-Shot mode. If TIMxENABLE = 1 when the counter reaches zero the counter stops counting. To start the counter again, the user must load TIMxLOADVAL with a non-zero value or set the Timer to Periodic mode by clearing TIMxMODE to 0.  
Writing this register while the System Timer is set to 64-bit mode has no effect. Reading this register while the System Timer is set to 64-bit mode returns the reset value. |
| 0          | TIMxENABLE  | R/W | 0x0         | Timer x Enable  
0 = Timer x disabled  
1 = Timer x enabled  
Setting to 1 enables the timer and starts it counting from the current value in TIMx.VAL unless TIMx.VAL is 0, in which case TIMx.VAL is loaded from TIMx_LOADVAL.  
Writing this register while the System Timer is set to 64-bit mode has no effect. Reading this register while the System Timer is set to 64-bit mode returns the reset value. |
### Timer x Raw Interrupt Status Register

**Table 17-6 • TIMx_RIS**

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Name</th>
<th>R/W</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:1</td>
<td>Reserved</td>
<td>R/W</td>
<td>0x0</td>
<td>Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.</td>
</tr>
</tbody>
</table>
| 0          | TIMx_RIS  | R/W | 0x0         | Timer x Raw Interrupt Status  
0 = Timer x has not reached zero  
1 = Timer x has reached zero at least once since this bit was last cleared (by a reset or by writing 1 to this bit).  
Writing a 1 to this bit clears the bit and the interrupt, writing a zero has no effect. |
Minute Quiz
What is the first interrupt that fires on a Cortex-M3?
Exercise: How many preemption priorities and subpriority levels do we get on the Smart Fusion if we set Priority Group to 5?
What if we quickly want to disable all interrupts?

Write 1 into **PRIMASK** to disable all interrupt except NMI
- MOV R0, #1
- MSR PRIMASK, R0

Write 0 into **PRIMASK** to enable all interrupts

**FAULTMASK** is the same as **PRIMASK**, but also blocks hard fault (priority -1)

What if we want to disable all interrupts below a certain priority?

Write priority into **BASEPRI**
- MOV R0, #0x60
- MSR BASEPRI, R0
Assume `BASE_PCHO_REG` is set to the programmable interrupt priority register, and the following code runs:

```assembly
movw r0, #:lower16:INT_PCHO_REG
movt r0, #:upper16:INT_PCHO_REG
mov r1, #61
strb r, [r0, #31]
MOV R0, #0x60
MSR BASEPRI, R0
```

Will interrupt 31 still interrupt your code?
• Minute quiz
• Finish up interrupts
• Graduate student presentations
• Final projects
What exactly is an interrupt handler?
• Upon an interrupt, the Cortex-M3 needs to know the address of the interrupt handler (function pointer)
• After powerup, vector table is located at 0x00000000

<table>
<thead>
<tr>
<th>Address</th>
<th>Exception Number</th>
<th>Value (Word Size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>–</td>
<td>MSP initial value</td>
</tr>
<tr>
<td>0x00000004</td>
<td>1</td>
<td>Reset vector (program counter initial value)</td>
</tr>
<tr>
<td>0x00000008</td>
<td>2</td>
<td>NMI handler starting address</td>
</tr>
<tr>
<td>0x0000000C</td>
<td>3</td>
<td>Hard fault handler starting address</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>Other handler starting address</td>
</tr>
</tbody>
</table>

• Can be relocated to change interrupt handlers at runtime (vector table offset register)
Vector Table in SoftConsole

- Located in startup_a2fxxxm3.s

```assembly

g_pfnVectors:
.word _estack
.word Reset_Handler
.word NMI_Handler
.word HardFault_Handler
.word MemManage_Handler
.word BusFault_Handler
.word UsageFault_Handler
.word 0
.word 0

/* Vector table */
.global g_pfnVectors
.section .isr_vector,"a",@progbits
.type g_pfnVectors, @object
.size g_pfnVectors, -.g_pfnVectors

SECTIONS
{
.text :
{
CREATE_OBJECT_SYMBOLS
__text_load = LOADADDR(.text);
__text_start = .;
*(.isr_vector)
}
```

- Put at 0x00000000 in linker script
Interrupt Handlers

192/*****************************/
193  * Reset_Handler
194  */
195  .global Reset_Handler
196  .type  Reset_Handler, %function
197Reset_Handler:
198  _start:

280/*****************************/
281  * NMI_Handler
282  */
283  .weak  NMI_Handler
284  .type  NMI_Handler, %function
285NMI_Handler:
286   B .
287
288/*****************************/
289  * HardFault_Handler
290  */
291  .weak HardFault_Handler
292  .type  HardFault_Handler, %function
293HardFault_Handler:
294   B .
• We can overwrite the predefined interrupt handlers

```c
__attribute__((__interrupt__)) void Timer1_IRQHandler()
{
    MSS_TIM1_disable_irq();
    MSS_TIM1_clear_irq();
    ...
    NVIC_ClearPendingIRQ( Timer1IRQn );
}

int main()
{
    ...
    MSS_TIM1_enable_irq();
    NVIC_EnableIRQ( Timer1IRQn );
    ...
    while(1){}
}
```
Interrupt Service Routines

1. Automatic saving of registers upon exception
   - PC, PSR, R0-R3, R12, LR pushed on the stack

2. While bus busy, fetch exception vector

3. Update SP to new location

4. Update IPSR (low part of PSR) with new exception number

5. Set PC to vector handler

6. Update LR to special value EXC_RETURN

- Several other NVIC registers get updated
- Latency: as short as 12 cycles
Interrupt Stacking

The values of PC and PSR are stacked first so that instruction fetch can be started early (which requires modification of PC) and the IPSR can be updated early. After stacking, SP will be updated to N-32 (0H1100320), and the stacked data arrangement in the stack memory will look like Table 9.1.

<table>
<thead>
<tr>
<th>Address (HADDR)</th>
<th>N-8</th>
<th>N-4</th>
<th>N-32</th>
<th>N-28</th>
<th>N-24</th>
<th>N-20</th>
<th>N-16</th>
<th>N-12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data (HWDATA)</td>
<td>PC</td>
<td>PSR</td>
<td>R0</td>
<td>R1</td>
<td>R2</td>
<td>R3</td>
<td>R12</td>
<td>LR</td>
</tr>
</tbody>
</table>

Time

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26:25</th>
<th>24</th>
<th>23:20</th>
<th>19:16</th>
<th>15:10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4:0</th>
</tr>
</thead>
<tbody>
<tr>
<td>APSR</td>
<td>N</td>
<td>Z</td>
<td>C</td>
<td>V</td>
<td>Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPSR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Exception Number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPSR</td>
<td></td>
<td></td>
<td>ICI/IT</td>
<td>T</td>
<td></td>
<td></td>
<td>ICI/IT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.3 Program Status Registers (PSRs) in the Cortex-M3

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26:25</th>
<th>24</th>
<th>23:20</th>
<th>19:16</th>
<th>15:10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4:0</th>
</tr>
</thead>
<tbody>
<tr>
<td>xPSR</td>
<td>N</td>
<td>Z</td>
<td>C</td>
<td>V</td>
<td>Q</td>
<td>ICI/IT</td>
<td>T</td>
<td>ICI/IT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Exception Number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.4 Combined Program Status Registers (xPSR) in the Cortex-M3

From: The Definitive Guide to the ARM Cortex-M3
Return from ISR

- 3 ways to return from an ISR

<table>
<thead>
<tr>
<th>Return Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BX &lt;reg&gt;</td>
<td>If the EXC_RETURN value is still in LR, we can use the $BX\ LR$ instruction to perform the interrupt return.</td>
</tr>
<tr>
<td>POP {PC}, or POP {..., PC}</td>
<td>Very often the value of LR is pushed to the stack after entering the exception handler. We can use the POP instruction, either a single POP or multiple POPs, to put the EXC_RETURN value to the program counter. This will cause the processor to perform the interrupt return.</td>
</tr>
<tr>
<td>LDR, or LDM</td>
<td>It is possible to produce an interrupt return using the LDR instruction with PC as the destination register.</td>
</tr>
</tbody>
</table>

- Unstack and reset SP
- Update NVIC registers
Nested Interrupts

• Built into the Cortex-M3 (not every MCU has this)
• Make sure main stack is large enough!

• Three methods:
  – Tail Chaining
  – Late Arrival
  – Preemption
• If first interrupt has same or higher priority
• Skip stacking/unstacking for efficiency
Late Arrival

- Main stack must be able to hold maximum number of preemptions!
EXC_RETURN

<table>
<thead>
<tr>
<th>Bits</th>
<th>31:4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Descriptions</td>
<td>0xFFFFFFFF</td>
<td>Return mode (thread/handler)</td>
<td>Return stack</td>
<td>Reserved; must be 0</td>
<td>Process state (Thumb/ARM)</td>
</tr>
</tbody>
</table>

Allowed values on the ARM Cortex-M3

<table>
<thead>
<tr>
<th>Value</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFFFFFF1</td>
<td>Return to handler mode</td>
</tr>
<tr>
<td>0xFFFFFFFF9</td>
<td>Return to thread mode and on return use the main stack</td>
</tr>
<tr>
<td>0xFFFFFFFFFD</td>
<td>Return to thread mode and on return use the process stack</td>
</tr>
</tbody>
</table>

From: The Definitive Guide to the ARM Cortex-M3
Preemption (Thread with Main Stack)

Interrupt #1 (Low priority)

Interrupt #2 (High priority)

Execution status

Interrupt event #1

Interrupt service routine #1

Interrupt service routine #2

Interrupt exit

Unstacking

Main program

Stacking

Main stack

Thread mode

L R = 0xFFFFFFFF9

Main stack

Handler mode

LR = 0xFFFFFFFF1

Main stack

Handler mode

Main stack

Handler mode

Main stack

Thread mode
Different Concepts of Interrupt Sharing

- Number of potential interrupts usually larger than interrupt lines availability on Core
- One peripheral often only has one interrupt
- Different types of events are stored in a status register

Example, UART
- IIR, 0x40000008

<table>
<thead>
<tr>
<th>3:0</th>
<th>Interrupt identification bits</th>
<th>R</th>
<th>0b0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b110 = Highest priority. Receiver line status interrupt due to overrun error, parity error, framing error or break interrupt. Reading the Line Status Register resets this interrupt.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b0100 = Second priority. Receive data available interrupt modem status interrupt. Reading the Receiver Buffer Register (RBR) or the FIFO drops below the trigger level resets this interrupt.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1100 = Second priority. Character timeout indication interrupt occurs when no characters have been read from the RX FIFO during the last four character times and there was at least one character in it during this time. Reading the Receiver Buffer Register (RBR) resets this interrupt.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b0010 = Third priority. Transmitter Holding Register Empty interrupt. Reading the IIR or writing to the Transmit Holding Register (THR) resets the interrupt.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b0000 = Fourth priority. Modem status interrupt due to Clear to Send, Data Set Ready, Ring Indicator, or Data Carrier Detect being asserted. Reading the Modem Status Register resets this interrupt.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This register is read only; writing has no effect. Also see Table 15-9.
ISR Sharing, i.e., Callbacks in C

- There is only one interrupt handler
- Functions have to “subscribe” for events
- Callbacks
  - Driver provides function to register a function pointer
  - Driver stores function pointers in list
  - Upon interrupt, each registered function gets called

```c
typedef void (*radioalarm_handler_t)(void);
radioalarm_handler_t radio_alarm_fired;

void RadioAlarm_init(radioalarm_handler_t handler)
{
    radio_alarm_fired = handler;
}

__attribute__((__interrupt__)) void Timer1_IRQHandler()
{
    alarm_state = FREE;
    MSS_TIM1_disable_irq();
    MSS_TIM1_clear_irq();
    NVIC_ClearPendingIRQ(Timer1_IRQn);
    (*(radio_alarm_fired))(); // call the callback function
}```
Common Problems and Pit-Falls

- Too many interrupts
  - Your core can’t keep up with handling interrupts

- Concurrency issues
  - One interrupt handler modifies global variables
  - Can be avoided using atomic sections protected through PRIMASK

- Lost interrupts
  - It can happen that an interrupt doesn’t get treated by the Core
  - State machine and peripheral has to be aware of this possibility
  - Danger for deadlocks
Summary

• Overwrite default Interrupt Handler

• Initialization
  – Enable interrupt in NVIC
  – Enable interrupt in Peripheral

• Upon Interrupt
  – Clear interrupt in Peripheral
  – Clear pending bit in NVIC
  – Potentially disable interrupts temporarily
Graduate Student Presentations

• Graduate students will present one embedded system
  – Processor and system architecture
  – Tools and software for programming, hackability
  – Platforms
  – Why does this system exist
  – Demo it

• 19 graduate students
• 5 Lectures between Spring Break & Open House
• 4 students per lecture (~15 min)
• You can pick your platform

• Sign up: http://goo.gl/HUAXN
You propose a platform!

Selection due by **Feb 21st**

Talk to me about the platform you want to choose
(Office Hours or by appointment)
Projects
Picking a Project Idea: Think **BIG** to Start

**Thinking Big: Segway Example**

- **Scale**: To Big...Accommodates adults!
- **Power**: Large Power Source and Actuators
- **Complex Control**
  - Gyro Stabilized
  - High Center of Gravity

**Problems**

**Solution**

- **Scale**: Scale Down 1’ High
- **Power**: Low Power, Hobby Servo Actuators
- **Simple Control**
  - “Tail” controls variable resistor
  - Low Center of Gravity

**Simplified 373 Project**

Slides from Matt Smith
Types Of Projects: Music

Air Guitar

Guitar Pick air action is modeled with 3 axis accelerometer.

Music created by sending MIDI codes to MIDI synthesizer.

Touch key matrix to emulate fret board of guitar. Fabricate with PC board.
Types Of Projects: Concept

Auto Balancing Teeter Totter

Angle position controlled by propeller speed

Angle is maintained with feedback control.

Infrared distance sensor to measure height

Construction by Knex

Slides from Matt Smith
Types Of Projects: Robotic Knight Ryder

- Featured gyros and accelerometers for inertial guidance (really).
- Spoiler was added to maintain traction and stability at high speeds! (probably cosmetic).

Slides from Matt Smith
Types Of Projects: Gaming
Space Invaders

Intense gaming in the 373 lab!

Classic game controllers: N64 and N8

Graphics display indicating the termination of Earth!

Player Two wins! Earth Terminated! (Press any key)!

Slides from Matt Smith
Types Of Projects: Measurement Radar

Servo provided angular sweep.

IR and Ultrasonic Sensor for Ranging

Advertisement

Reflections plotted as function of angle and distance
Types Of Projects: Research
Wireless Power Monitoring

**Objectives**
- Contained in 1 cubic inch
- Wireless transmitting info to central monitor and control
- Low power
- Low cost (in quantity)
Demonstration of working PCB
Quadcopter
Idea, Starting Points

- **Review Last Year ECE/CS 5780/6780 Projects**
  - [http://wiesel.ece.utah.edu/redmine/projects/ece5780-s12-groups](http://wiesel.ece.utah.edu/redmine/projects/ece5780-s12-groups)

- **Review Past UM 373 Projects**
  - [http://www.eecs.umich.edu/courses/eecs373/Labs/Web/projects.html](http://www.eecs.umich.edu/courses/eecs373/Labs/Web/projects.html)
  - Search YouTube 373 projects
  - Provides Sense of Scale

- **Review Cornell Projects Web Site**
  - [http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects/](http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects/)
  - Feedback control oriented, but lots of applications
  - More devices to consider

- **Research Oriented Projects**
  - Prof Schmid will provide a list soon

- **YOU!**
  - Have a big cup of coffee and dream
  - Pick something you want to do!!
    - Think about all the embedded applications around you
    - Consider variants
    - Consider improvements
  - Research the application (know something about it!)
  - Discuss your ideas with potential partners and friends
  - Discuss your ideas with staff
Forming Groups

• Group sizes: 2 – 4
• Larger Groups
  – Advantages: Do more complex projects
  – Disadvantages: Challenging group management, unknown relationships
• Smaller Groups
  – Advantages: Group dynamic is simpler, task management, known relationship, etc
  – Disadvantages: Possibly limits project complexity
• Start with existing Lab Partner or form new groups
Proposal

• Due: 3/5, Tuesday in Lecture
• Contents
  – List of group members
  – Goal Statement: In general terms describe your project
  – Functional Specification
    List and describe high level functions
    high level functional diagram
  – Preliminary Component List
• Proposal Reviews
  – March 6th (Wednesday) & 7th (Thursday)
Proposal Example

Goal Statement

For our project we intend to build a sound level meter. Sound level meters are used in applications ranging from environmental noise management to balancing sound systems in concert halls.

Our meter will approximate the Extech Model 407764. We will attempt emulate some the meter’s basic functionality, but with out the same precision or reference accuracy.

The meter will have the following basic functions:
1. Sound level measurement with A and C frequency weighting
2. Time weighting from 1 – 100 seconds
3. Linear and logarithmic display of sounds level
4. Manual (4 ranges) and auto ranging
5. Data logging for 1 hour
6. PC interface to hyper terminal for ASCII file time series file storage of data log.
Functional Description

• Sound Measurement
  – Microphone: Commercial sound meters use expensive microphones. We will use a simple audio mic that will not have the same sensitivity, but can be frequency compensated.
  – Signal Conditioning: An audio amplifier will have to be provided to provide gain to the ADC.
  – Signal Conditioning: An anti-aliasing filter will have to be provided to for audio frequencies. We will use an active filter.

• Data Acquisition
  – The ACE will be setup to acquire data with 10 bit resolution and sample frequency of 40khz.

• Frequency Measurement
  – An FFT over the audio range will be performed using SmartFusion FFT core.

• Display
  – Display sound level digitally, simply analog meter graphic, measurement modes, etc.

• Key Pad
  – User input: measurement modes, display options, etc.
Functional Diagram

- Audio
  - Microphone
  - Amplifier
  - Anti-alias Filter

- Keypad

- SmartFusion Kit
  - User Interface
  - ADC
  - FFT
  - Log Memory

- Serial Interface to Computer

- Display
Questions?

Comments?

Discussion?

Write on a paper:

- One point that really **sticks out** about this lecture
- One point that was **unclear** about this lecture