EECS 373
Design of Microprocessor-Based Systems

Website: www.eecs.umich.edu/courses/eecs373/

Mark Brehob
University of Michigan

Lecture 1: Introduction, start on ARM ISA
January 7th 2016
Folks

- **Dr. Mark Brehob**
  - Lecturer (full-time teacher) in EECS
  - My background is on the embedded systems and architecture side

- **Matt Smith**
  - Head lab instructor
  - Been doing 373 for about 15 years!

- **IAs:**
  - John Connolly [johnconn@umich.edu](mailto:johnconn@umich.edu)
  - Daniel Synder [snysly@umich.edu](mailto:snysly@umich.edu)
  - Duncan Fairbanks [fduncan@umich.edu](mailto:fduncan@umich.edu)
  - Brent Pniewski [bpniewsk@umich.edu](mailto:bpniewsk@umich.edu)
What is an embedded system?
Embedded, everywhere
Embedded, Everywhere - Fitbit
What is driving the embedded everywhere explosion?
Outline

**Technology Trends**

Course Description/Overview

Tools Overview/ISA start
Moore’s Law (a statement about economics): IC transistor count doubles every 18-24 mo
The number of computers per person grows

[Bell et al. Computer, 1972, ACM, 2008]
Computer volume shrinks by 100x every decade

Mainframe
1 per Enterprise

Workstation
1 per Engineer

[100x smaller every decade]

[Nakagawa08]

Personal Computer
1 per Family

Smartphone
1 per person

Minicomputer
1 per Company

Laptop
1 per Professional

100 – 1000’s per person

Smart Sensors
Price falls dramatically, and enables new applications

- **Inflation Adjusted Price (1000s of USD)**
- **Mainframe**
- **Workstation**
- **Laptop**
- **Smartphone**
- **Personal Computer**
- **Mini Computer**

- **Number Crunching**
- **Data Storage**

- **Stream Information to/from the Physical World**
- **Productivity Interactive**
“Roughly every decade a new, lower priced computer class forms based on a new programming platform, network, and interface resulting in new usage and the establishment of a new industry.”

- Gordon Bell [1972,2008]
What is driving Bell’s Law?

Technology Scaling

- **Moore’s Law**
  - Made transistors **cheap**

- **Dennard’s Scaling**
  - Made them **fast**
  - And **low-power**

- **Result**
  - Holding #T’s constant
    - Exponentially lower cost
    - Exponentially lower power
  - Small, cheap & low-power
    - Microcontrollers
    - Memory
    - Radios

Technology Innovations

- **MEMS technology**
  - Micro-fabricated sensors

- **New memories**
  - New cell structures (11T)
  - New tech (FeRAM, FinFET)

- **Near-threshold computing**
  - Minimize active power
  - Minimize static power

- **New wireless systems**
  - Radio architectures
  - Modulation schemes

- **Energy harvesting**
Corollary to Moore’s Law

Intel 4004 processor
Introduced 1971
Initial clock speed
108 KHz
Number of transistors
2,300
Manufacturing technology
10µ

Quad-Core Intel Xeon processor
Quad-Core Intel Core™2 Extreme processor
Introduced 2006
Intel Core™2 Quad processors
Introduced 2007
Initial clock speed
2.66 GHz
Number of transistors
582,000,000
Manufacturing technology
65nm

UMich Phoenix Processor
Introduced 2008
Initial clock speed
106 kHz @ 0.5V Vdd
Number of transistors
92,499
Manufacturing technology
0.18 µ

Photo credits: Intel, U. Michigan
Broad availability of inexpensive, low-power, 32-bit MCUs (with enough memory to do interesting things)
Hendy’s “Law”:
Pixels per dollar doubles annually

Credit: Barry Hendy/Wikipedia

Radio technologies enabling pervasive computing, IoT

Source: Steve Dean, Texas Instruments
Established comms interfaces: 802.15.4, BLE, NFC

- **IEEE 802.15.4** (a.k.a. “ZigBee” stack)
  - Workhorse radio technology for sensornets
  - Widely adopted for low-power mesh protocols
  - Middle (6LoWPAN, RPL) and upper (CoAP layers)
  - Can last for years on a pair of AA batteries

- **Bluetooth Low-Energy (BLE)**
  - Short-range RF technology
  - On phones and peripherals
  - Can beacon for years on coin cells

- **Near-Field Communications (NFC)**
  - Asymmetric backscatter technology
  - Small (mobile) readers in smartphones
  - Large (stationary) readers in infrastructure
  - New: ambient backscatter communications
Emerging Proximal Interfaces: Ultrasonic, Visible Light, Vibration

- **Ultrasonic**
  - Small, low-power, short-range
  - Supports very low-power wakeup
  - Can support pairwise ranging of nodes

- **Visible Light**
  - Enabled by pervasive LEDs and cameras
  - Supports indoor localization and comms
  - Easy to modify existing LED lighting

- **Vibration**
  - Pervasive accelerometers
  - Pervasive Vibration motors
  - Bootstrap desktop area context
Non-volatile memory capacity & read/write bandwidth

Lower capacity but
Higher R/W speeds
*and*
Lower energy per atomic operation
*and*
High write endurance
MEMS Sensors:
Rapidly falling price and power of accelerometers

**ADXL345**
- 25 µA @ 25 Hz
- ADXL345

**ADXL362**
- 10 µA @ 10 Hz @ 6 bits
- ADXL362

- 1.8 µA @ 100 Hz @ 2V
- 300 nA wakeup mode

[ST Microelectronics, annc. 2009]

[Analog Devices, 2009]

[Analog Devices, 2012]
Energy harvesting and storage: Small doesn’t mean powerless...

- RF [Intel]
- Clare Solar Cell
- Thin-film batteries
- Piezoelectric [Holst/IMEC]
- Shock Energy Harvesting CEDRAT Technologies
- Electrostatic Energy Harvester [ICL]
- Thermoelectric Ambient Energy Harvester [PNNL]
Why study 32-bit MCUs and FPGAs?
MCU-32 and PLDs are tied in embedded market share

Source: iSuppli
What differentiates these products from one another?

FPGA
=====

Microprocessor
==============
The Cortex M3's Thumb2l architecture looks like a conventional Arm processor. The differences are found in the Harvard architecture and the instruction decode that handles only Thumb and Thumb 2 instructions.
Modern FPGAs: best of both worlds!
Why study the ARM architecture (and the Cortex-M3 in particular)?
Lots of manufacturers ship ARM products
ARM is the big player

- ARM has a huge market share
  - As of 2011 ARM has chips in about 90% of the world’s mobile handsets
  - As of 2010 ARM has chips in 95% of the smartphone market, 10% of the notebook market
    - Expected to hit 40% of the notebook market in 2015.
  - Heavy use in general embedded systems.
    - Cheap to use
      - ARM appears to get an average of 8¢ per device (averaged over cheap and expensive chips).
    - Flexible
      - Spin your own designs.
Outline

Technology Trends

Course Description/Overview

Tools Overview/ISA start
Course goals

- **Learn to implement** embedded systems including hardware/software interfacing.

- **Learn to design** embedded systems and how to think about embedded software and hardware.

- Have the opportunity to *design and build* non-trivial projects involving both hardware and software.
Prerequisites

• EECS 270: Introduction to Logic Design
  - Combinational and sequential logic design
  - Logic minimization, propagation delays, timing

• EECS 280: Programming and Intro Data Structures
  - C programming
  - Algorithms (e.g. sort) and data structures (e.g. lists)

• EECS 370: Introduction to Computer Organization
  - Basic computer architecture
  - CPU control/datapath, memory, I/O
  - Compiler, assembler
Topics

- **Memory-mapped I/O**
  - The idea of using memory addressed to talk to input and output devices.
    - Switches, LEDs, hard drives, keyboards, motors

- **Interrupts**
  - How to get the processor to become “event driven” and react to things as they happen.

- **Working with Analog inputs**
  - The real world isn’t digital!

- **Common devices and interfaces**
  - Serial buses, timers, etc.
Example: Memory-mapped I/O

- This is **important**.
  - It means our software can tell the hardware what to do.
    - In lab 3 you’ll design hardware on an FPGA which will can control a motor.
      - But more importantly, that hardware will be designed so the software can tell the hardware exactly what to do with the motor. All by simply writing to certain memory locations!
    - In the same way, the software can read memory locations to access data from sensors etc...
Example: Anatomy of a timer system

Applications
Operating System

Timer Abstractions and Virtualization

Low-Level Timer Subsystem Device Drivers

Software
Hardware

Compare

Counter

Capture

Prescaler

Clock Driver

Xtal/Osc

I/O

I/O

Internal
External

Application Software

typedef struct timer {
  timer_handler_t handler;
  uint32_t time;
  uint8_t mode;
  timer_t* next_timer;
} timer_t;

timer_tick:
  ldr r0, count;
  add r0, r0, #1

module timer(clr, ena, clk, alrm);
  input clr, ena, clk;
  output alrm;
  reg alrm;
  reg [3:0] count;

  always @(posedge clk) begin
    alrm <= 0;
    if (clr) count <= 0;
    else count <= count+1;
  end
endmodule

...
### Grades

<table>
<thead>
<tr>
<th>Item</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Labs (7)</td>
<td>25%</td>
</tr>
<tr>
<td>Project</td>
<td>25%</td>
</tr>
<tr>
<td>Exams</td>
<td>40% (15% midterm; 20% final)</td>
</tr>
<tr>
<td>HW/Guest talks</td>
<td>10%</td>
</tr>
<tr>
<td>Oral presentation</td>
<td>5%</td>
</tr>
</tbody>
</table>

- Project and Exams tend to be the major differentiators.
- Class median is generally a low B+. 
Time

- You’ll need to assume you are going to spend a lot of time in this class.
  - 2-3 hours/week in lecture (we cancel a few classes during project time)
  - 8-12 hours/week working in lab
    - *Expect more during project time; some labs are a bit shorter.*
  - ~20 hours (total) working on homework
  - ~20 hours (total) studying for exams.
  - ~8 hour (total) on your oral presentation

- Averages out to about 15-20 hours/week pre-project and about 20 during the project...
  - This is more than I’d like, but we’ve chosen to go with state-of-the-art tools, and those generally have a heck of a learning curve.
Labs

- 7 labs, 8 weeks, groups of 2
  1. FPGA + Hardware Tools
  2. MCU + Software Tools
  3. Memory + Memory-Mapped I/O
  4. Interrupts
  5. *Timers and Counters*
  6. Serial Bus Interfacing
  7. Data Converters (e.g. ADCs/DACs)

- Labs are very time consuming.
  - As noted, students estimated 8-12 hours per lab with one lab (which varied by group) taking longer.
Open-Ended Project

• Goal: learn how to build embedded systems
  - By building an embedded system
  - Work in teams of 4
  - You design your own project

• The major focus of the last third of the class.
  - Labs will be done and we will cancel some lectures and generally try to keep you focused.

• Important to start early.
  - After all the effort in the labs, it’s tempting to slack for a bit. The best projects are those that get going right away.
Homework

- 4-6 assignments
  - A few “mini” assignments
    - Mainly to get you up to speed on lab topics
  - A few “standard” assignments
    - Hit material we can’t do in lab.

- Also a small part is for showing up to guest lecturer(s)

*Start today, Homework 1 due on Tuesday*
Looking for me?

- Monday 11-noon -- 4632 Beyster

- Wednesday 4:00-5:30pm -- EECS 2334 (our lab)

- Friday 1:30-2:30pm -- 4632 Beyster
  - (canceled on 1/29 and 2/26)
Outline

Technology Trends

Course Description/Overview

Tools overview/ISA start
We are using Actel’s SmartFusion Evaluation Kit.
A2F200M3F-FGG484ES
- 200,000 System FPGA gates, 256 KB flash memory, 64 KB SRAM, and additional distributed SRAM in the FPGA fabric and external memory controller
- Peripherals include Ethernet, DMAs, I²Cs, UARTs, timers, ADCs, DACs and additional analog resources
  - USB connection for programming and debug from Actel's design tools
  - USB to UART connection to UART_0 for HyperTerminal examples
  - 10/100 Ethernet interface with on-chip MAC and external PHY
  - Mixed-signal header for daughter card support
FPGA work
“Smart Design” configurator
Eclipse-based “Actel SoftConsole IDE”
Debugger is GDB-based. Includes command line. Works really quite well.
ARM ISA
Major elements of an Instruction Set Architecture
(registers, memory, word size, endianess, conditions, instructions, addressing modes)

mov r0, #1
ld r1, [r0,#5]
r1=mem((r0)+5)
bne loop
subs r2, #1
The endianess religious war: 284 years and counting!

• Modern version
  - Danny Cohen
  - IEEE Computer, v14, #10
  - Published in 1981
  - Satire on CS religious war

• Historical Inspiration
  - Jonathan Swift
  - Gulliver's Travels
  - Published in 1726
  - Satire on Henry-VIII’s split with the Church
    • Now a major motion picture!

• Little-Endian
  - LSB is at lower address
    
    | Memory Offset (LSB) (MSB) |
    |--------------------------|
    | 0x0000 01 02 FF 00 |

    ```
    uint8_t a = 1;
    uint8_t b = 2;
    uint16_t c = 255; // 0x00FF
    uint32_t d = 0x12345678;
    ```

• Big-Endian
  - MSB is at lower address
    
    | Memory Offset (LSB) (MSB) |
    |--------------------------|
    | 0x0000 01 02 00 FF |

    ```
    uint8_t a = 1;
    uint8_t b = 2;
    uint16_t c = 255; // 0x00FF
    uint32_t d = 0x12345678;
    ```
Addressing: Big Endian vs Little Endian (370 slide)

- **Endian-ness**: ordering of bytes within a word
  - Little - increasing numeric significance with increasing memory addresses
  - Big - The opposite, most significant byte first
  - MIPS is big endian, x86 is little endian
Instruction encoding

• Instructions are encoded in machine language opcodes
• Sometimes
  - Necessary to hand generate opcodes
  - Necessary to verify assembled code is correct
• How?

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Register Value</th>
<th>Memory Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>movs r0, #10</td>
<td>001</td>
<td>00</td>
</tr>
<tr>
<td>movs r1, #0</td>
<td>001</td>
<td>00</td>
</tr>
</tbody>
</table>

**Encoding T1**

All versions of the Thumb ISA.

<table>
<thead>
<tr>
<th>Rd</th>
<th>imm8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 1 0 1 0</td>
<td></td>
</tr>
</tbody>
</table>

```c
d = UInt(Rd); setflags = !UnITBlock(); imm32 = ZeroExtend(imm8, 32); carry = APSR.C;
```
Assembly example

data:

    .byte 0x12, 20, 0x20, -1

func:

    mov r0, #0
    mov r4, #0
    movw r1, #:lower16:data
    movt r1, #:upper16:data

    top:    ldrb r2, [r1],1
            add r4, r4, r2
            add r0, r0, #1
            cmp r0, #4
            bne top
Instructions used

- **mov**
  - Moves data from register or immediate.
  - Or also from shifted register or immediate!
    - the mov assembly instruction maps to a bunch of different encodings!
  - If immediate it might be a 16-bit or 32-bit instruction.
    - Not all values possible
    - why?

- **movw**
  - Actually an alias to mov.
    - “w” is “wide”
    - hints at 16-bit immediate.
From the ARMv7-M Architecture Reference Manual (posted on the website under references)

A6.7.76 MOV (register)

Move (register) copies a value from a register to the destination register. It can optionally update the condition flags based on the value.

Encoding T1       ARMv6-M, ARMv7-M
MOV<><Rd>,<Rm>

If <Rd> and <Rm> both from R0-R7, otherwise all versions of the Thumb ISA.
If <Rd> is the PC, must be outside or last in IT block

\[
\begin{array}{cccccccccccc}
15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & D & Rm & Rd
\end{array}
\]

d = Uint(D;Rd); m = Uint(Rm); setflags = FALSE;
if d == 15 && InITBlock() && !LastInITBlock() then UNPREDICTABLE;

Encoding T2       All versions of the Thumb ISA.
MOV$<><Rd>,<Rm>

(formerly LSL <Rd>,<Rm>,#0)
Not permitted inside IT block

\[
\begin{array}{cccccccccccc}
15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Rm & Rd
\end{array}
\]

d = Uint(Rd); m = Uint(Rm); setFlags = TRUE;
if InITBlock() then UNPREDICTABLE;

Encoding T3       ARMv7-M
MOV$<><.W<Rd>,<Rm>

\[
\begin{array}{cccccccccccc}
15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & S & 1 & 1 & 1 & (0)
\end{array}
\]

\[
\begin{array}{cccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Rd & 0 & 0 & 0 & 0 & Rm
\end{array}
\]

d = Uint(Rd); m = Uint(Rm); setflags = (S == '1');
if setflags && (d IN {13,15} || m IN {13,15}) then UNPREDICTABLE;
if !setflags && (d == 15 || m == 15 || (d == 13 && m == 13)) then UNPREDICTABLE;

There are similar entries for move immediate, move shifted (which actually maps to different instructions) etc.
Directives

• `:#lower16: data`
  - What does that do?
  - Why?
A6.7.78  MOVT

Move Top writes an immediate value to the top halfword of the destination register. It does not affect the contents of the bottom halfword.

**Encoding T1**  ARMv7-M

MOVT<c>  <Rd>,#<imm16>

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0  |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|    |
| 11 | 11 | 11 | 10 | i | 1 | 0 | 1 | 1 | 0 | 0 | imm4 | 0 | imm3 | Rd | imm8 |

d = UInt(Rd); imm16 = imm4:i:imm3:imm8;
if d IN {13,15} then UNPREDICTABLE;

**Assembler syntax**

MOVT<c><q>  <Rd>, #<imm16>

where:

<c><q>  See *Standard assembler syntax fields* on page A6-7.

<Rd>  Specifies the destination register.

<imm16>  Specifies the immediate value to be written to <Rd>. It must be in the range 0-65535.

**Operation**

if ConditionPassed() then
    EncodingSpecificOperations();
    R[d]<31:16> = imm16;
    // R[d]<15:0> unchanged
Loads!

- ldrb?
- ldrsb?
So what does the program _do_?

data:

    .byte 0x12, 20, 0x20, -1

cfunc:

    mov r0, #0
    mov r4, #0
    movw r1, #:lower16:data
    movt r1, #:upper16:data

top:

    ldrb r2, [r1],1
    add r4, r4, r2
    add r0, r0, #1
    cmp r0, #4
    bne top
Questions?

Comments?

Discussion?