EECS 373

Design of Microprocessor-Based Systems

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

Robert Dick
University of Michigan

Lecture 1: Introduction, ARM ISA

September 6 2017

Many slides from Mark Brehob

Teacher

Robert Dick
http://robertdick.org/
dickrp@umich.edu

- EECS Professor
- Co-founder, CEO of profitable direct-to-consumer athletic wearable electronics company (Stryd)
- Visiting Professor at Tsinghua Univeristy
- Graduate studies at Princeton
- Visiting Researcher at NEC Labs, America, where technology went into their smartphones
- ~100 research papers on embedded system design

Lab instructor

- Matthew Smith
  - matsmith@umich.edu
  - Head lab instructor
  - 15 years of 373 experience
  - He has seen it before
  - … but he’ll make you figure it out yourself

TAs

- Took EECS 373
- Jon Toto <jontoto@umich.edu>
- Brennan Garrett <bdgarr@umich.edu>
- Thomas Deeds <deedstho@umich.edu>
- Melinda Kothbauer <mkothbau@umich.edu>

Course goals

- **Embedded system design**
- **Debugging complex systems**
- Communication and marketing
- A head start on a new product or research idea

What is an embedded system?

An (application-specific) computer
within something else
that is not generally regarded as a computer.

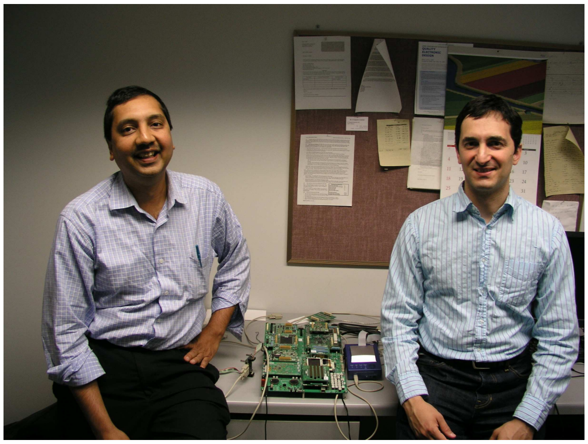
Embedded systems market

- Dominates general-purpose computing market in volume.
- Similar in monetary size to general-purpose computing market.
- Growing at 15% per year, 10% for general-purpose computing.
- Car example: half of value in embedded electronics, from zero a few decades ago.

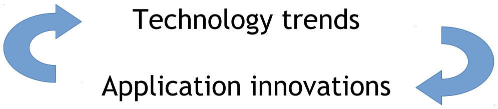
Common requirements

- Timely (hard real-time)
- Wireless
- Reliable
- First time correct
- Rapidly implemented
- Low price
- High performance
- Low power
- Embodying deep domain knowledge
- Beautiful

Example design process



What is driving the embedded everywhere trend?



Technology trends

Application innovations

Outline

Technology Trends

Course Description/Overview

Review, Tools Overview, ISA

Moore's Law (a statement about economics): IC transistor count doubles every 18-24 months

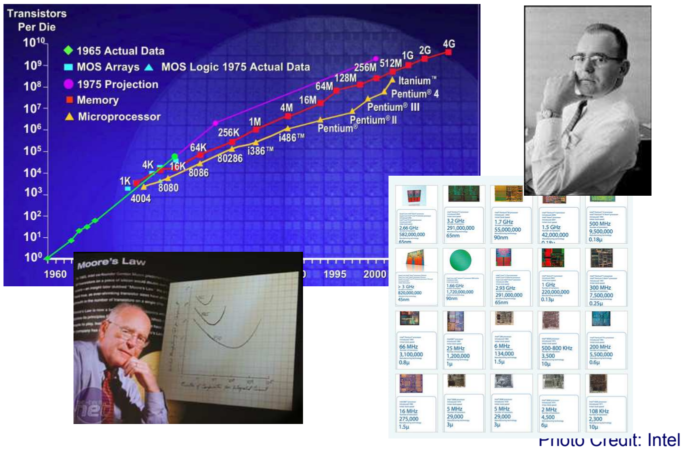
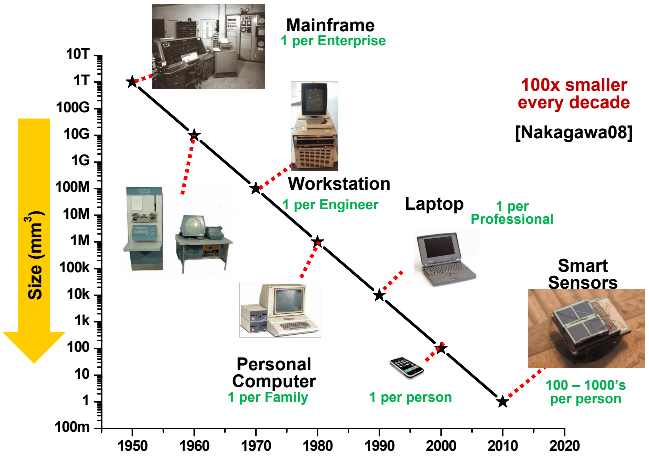


Photo Credit: Intel

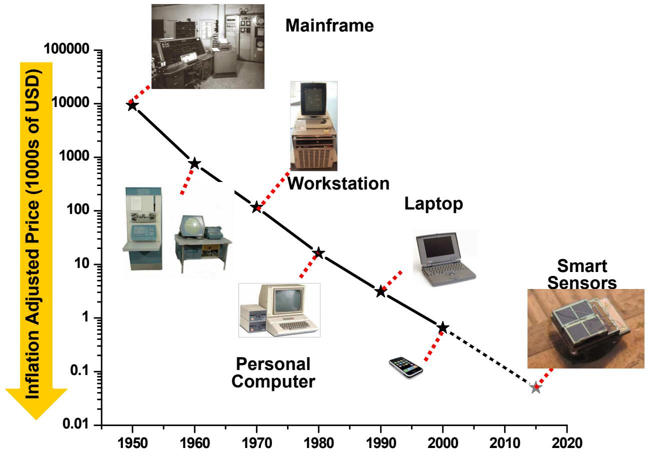
Computer volume shrinks by 100x every decade



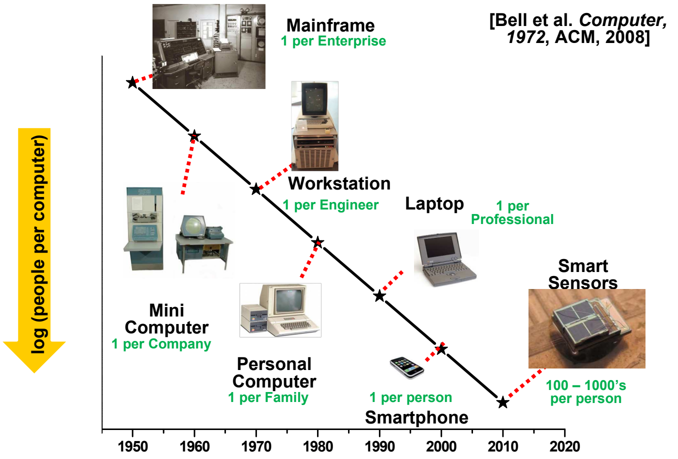
100x smaller every decade

[Nakagawa08]

Price falls dramatically, and enables new applications



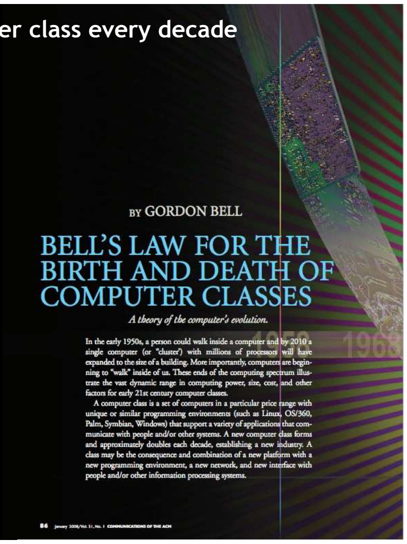
Computers per person



[Bell et al. *Computer*, *1972*, ACM, 2008]

Bell's Law: A new computer class every decade

> *"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]



BY GORDON BELL

BELL'S LAW FOR THE BIRTH AND DEATH OF COMPUTER CLASSES

A theory of the computer's evolution.

In the early 1950s, a person could walk inside a computer and by 2010 a single computer (or "cluster") with millions of processors will have expanded to the size of a building. More importantly, computers are beginning to "walk" inside of us. These ends of the computing spectrum illustrate the vast dynamic range in computing power, size, cost, and other factors for early 21st century computer classes.

A computer class is a set of computers in a particular price range with unique or similar programming environments (such as Linux, OS/360, Palm, Symbian, Windows) that support a variety of applications that communicate with people and/or other systems. A new computer class forms and approximately doubles each decade, establishing a new industry. A class may be the consequence and combination of a new platform with a new programming environment, a new network, and new interface with people and/or other information processing systems.

86 January 2008/Vol. 51, No. 1 COMMUNICATIONS OF THE ACM

What is driving Bell's Law?

Technology Scaling

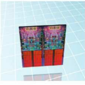
- Moore's Law
  - Made transistors cheap
- Dennard Scaling
  - Made them fast
  - But power density undermines
- Result
  - Fixed transistor count
    - Exponentially lower cost
    - Exponentially lower power
  - Small, cheap, and low-power
    - Microcontrollers
    - Sensors
    - Memory
    - Radios

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

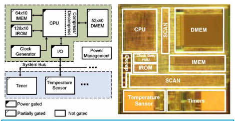
Corollary to Moore's Law

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

15x size decrease
40x transistors
55x smaller λ

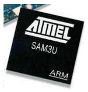


UMich Phoenix Processor
Introduced 2008
Initial clock speed
106 kHz @ 0.5V Vdd
Number of transistors
92,499
Manufacturing technology
0.18 μ

Photo credits: Intel, U. Michigan

Broad availability of inexpensive, low-power, 32-bit MCUs (with enough memory to do interesting things)

Actel® POWER MATTERS



ATMEL®

CYPRESS PERFORM

DUST NETWORKS

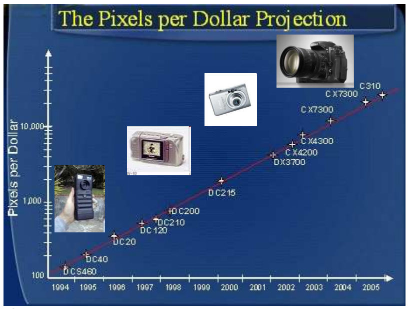
ember

NXP

ST

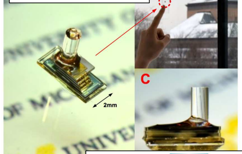
TEXAS INSTRUMENTS

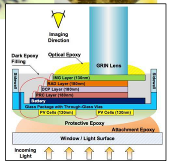
Hendy's "Law": Pixels per dollar doubles annually



Credit: Barry Hendy/Wikipedia

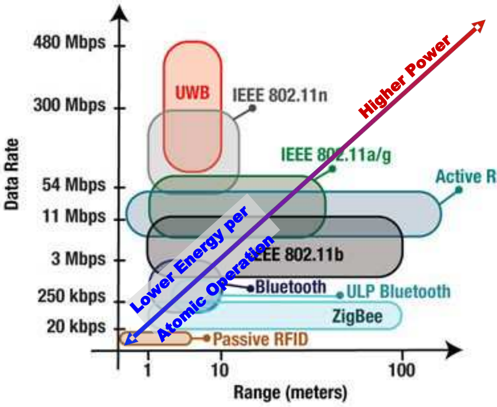
Figure 11. Sample image taken with the image sensor and GRIN lens, transmitted via Mbus.



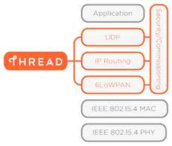


G. Kim, Z. Foo, Y, Lee, P. Pannuto, Y-S. Kuo, B. Kempke, M. Ghaed, S. Bang, I. Lee, Y. Kim, S. Jeong, P. Dutta, D. Sylvester, D. Blaauw,
"A Millimeter-Scale Wireless Imaging System with Continuous Motion Detection and Energy Harvesting,
In Symposium of VLSI Technology (VLSI'14), Jun. 2014.

Radio technologies enabling pervasive computing, IoT



Bluetooth® SMART

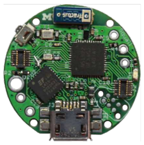


Source: Steve Dean, Texas Instruments
http://eecatalog.com/medical/2009/09/23/current-and-future-trends-in-medical-electronics/

Established commun 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 Smart
  - Short-range RF technology
  - On phones and peripherals
  - Can beacon for years on coin cells



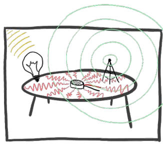
- Near-Field Communications (NFC)
  - Asymmetric backscatter technology
  - Small (mobile) readers in smartphones
  - Large (stationary) readers in infrastructure
  - New: ambient backscatter communications

Emerging interfaces: ultrasonic, light, vibration

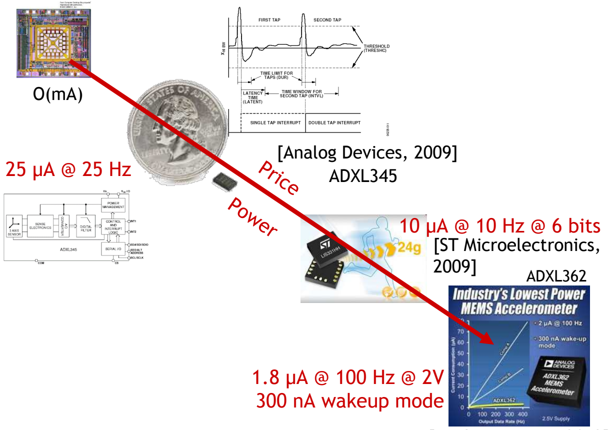
- Ultrasonic
  - Small, low-power, short-range
  - Supports very low-power wakeup
  - Can support pairwise ranging of nodes

- 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



MEMS Sensors: Rapidly falling price and power of accelerometers



O(mA)

25 µA @ 25 Hz

Price

Power

[Analog Devices, 2009]
ADXL345

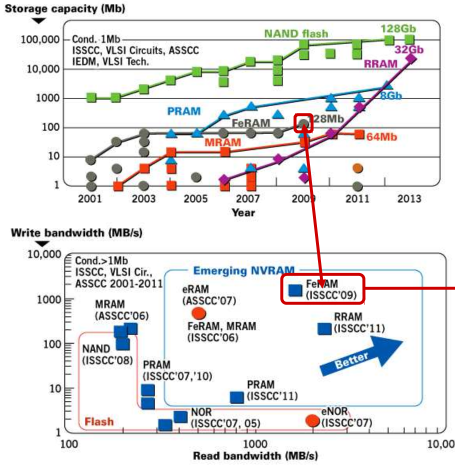
10 µA @ 10 Hz @ 6 bits
[ST Microelectronics, 2009]

ADXL362

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

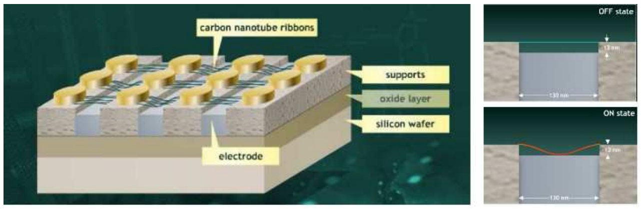
[Analog Devices, 2012]

Non-volatile memory capacity & read/write bandwidth



Lower capacity but
Higher R/W speeds
and
Lower energy per atomic operation
and
High write endurance

NRAM



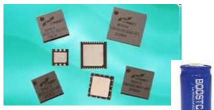
- Nonvolatile
- Fast as DRAM
- Vapor(hard)ware
- May happen

Energy harvesting and storage:

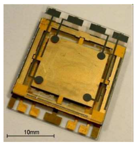
RF [Intel]

Clare Solar Cell

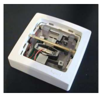
1st Annual Workshop on
October 22, 2009
MICRO POWER TECHNOLOGIES
Radisson Hotel, San Jose, CA



Thin-film batteries

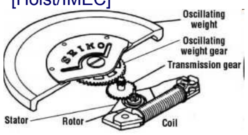


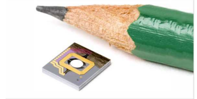
Electrostatic Energy Harvester [ICL]

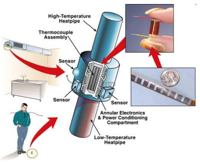


Shock Energy Harvesting CEDRAT Technologies

Piezoelectric [Holst/IMEC]






High-Temperature Heatpipe
Thermocouple Assembly
Sensor
Sensor
Sensor
Annular Electronics & Power Conditioning Compartment
Low-Temperature Heatpipe

Thermoelectric Ambient Energy Harvester [PNNL]

Growing application domains

- Wearable
- Social
- Location-aware
- IoT: integrated with physical world, networked
- Automated transportation
- Medical

My observation

- Every new class of computer systems will initially be seen as a toy by many or most
- As it becomes socially and commercially important, nearly everybody will act as if it was always obvious this would happen
- … even those who claimed it would always be a toy.
- If logic dictates something, ignore the naysayers.
  - But that logic better consider potential customers.

Embedded, Everywhere Example - Stryd

What?

- Tiny wearable embedded system
- Wireless communication
- Integrated signal processing
- Careful power management
- Unconventional sensors

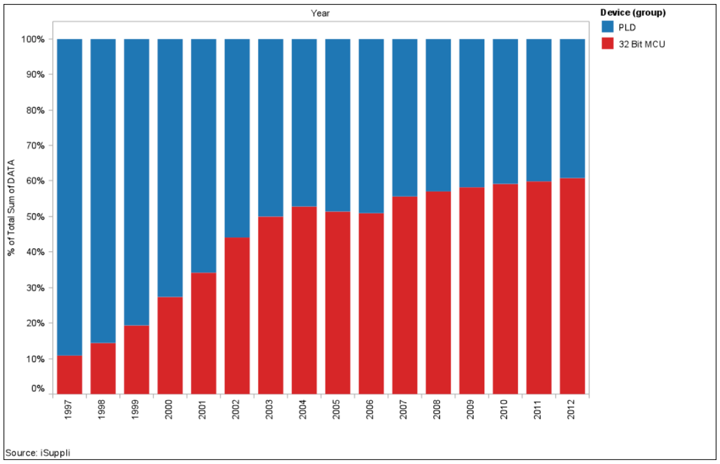
Lionel Sanders setting Ironman Triathlon World Record wearing Stryd

Why?

- Lets athletes precisely control effort when training and racing so they can run faster

Why study 32-bit MCUs and FPGAs?

MCU-32 and PLDs are tied in embedded market share



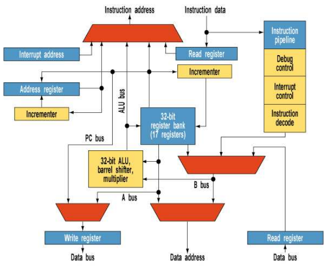
Source: iSuppli

What differentiates these?

FPGA

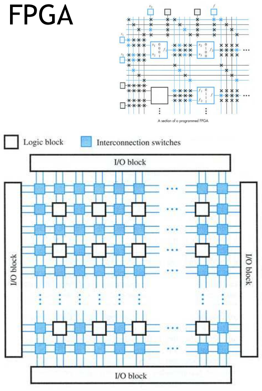
Microprocessor

Microcontroller



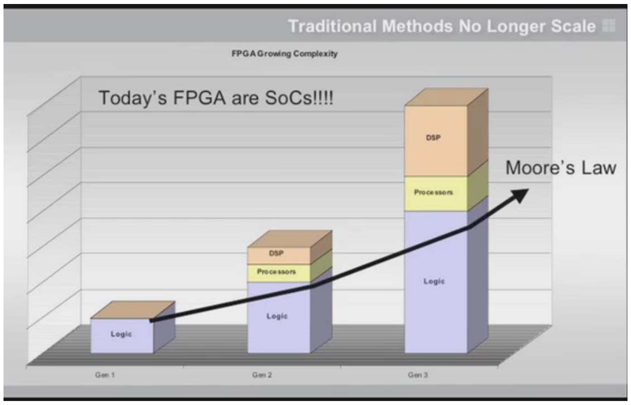
The Cortex M3's Thumbnail architecture looks like a conventional Arm processor. The differences are found in the Harvard architecture and the instruction decode that handles only Thumb and Thumb 2 instructions.

FPGA



General structure of an FPGA

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
  - 15-billion chips shipped in 2015
  - >90% of smartphone market
  - 10% of notebooks
- 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
- Intel history

Outline

~~Technology Trends~~

Course Description/Overview

Review, Tools Overview, ISA start

Course goals

- **Embedded system design**
- **Debugging complex systems**
- Communication and marketing
- A head start on a new product or research idea

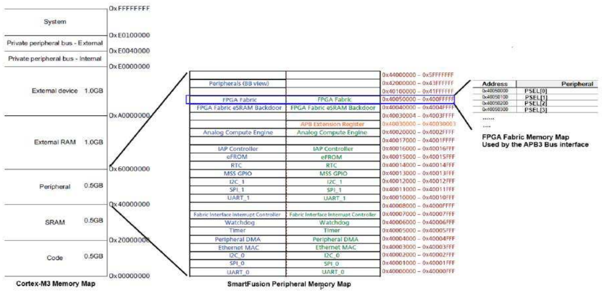
Prerequisites

- 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

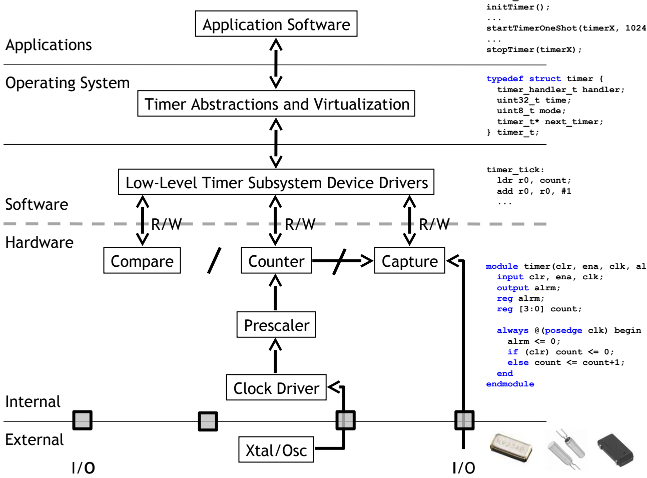
- Memory-mapped I/O
  - The idea of using memory addressed to talk to input and output devices.
  - Switches, LEDs, hard drives, keyboards, motors
- Interrupts
  - How to get the processor to become “event driven” and react to things as they happen.
- Working with analog inputs
  - Interfacing with the physical world.
- Common devices and interfaces
  - Serial buses, timers, etc.

Example: Memory-mapped I/O



- Enables program to communicate directly with hardware
  - Will use in Lab 3
  - Write memory to control motor
  - Read memory to read sensors

Example: Anatomy of a timer system



```
...
timer_t timerX;
initTimer();
...
startTimerOneShot(timerX, 1024);
...
stopTimer(timerX);
```

```
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

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

Labs	24%
Project	30%
Midterm 1	16%
Midterm 2	16%
Homework	7%
Presentations	7%

Time

- This is a time-consuming class
  - 2-3 hours/week in lecture
  - 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 would like, but we’ve chosen to use industrial-strength tools, which take time to learn.

Labs

- 7 labs
  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)
- Difficulty ramps up until Lab 6.
- 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
  - You design your own project
- Will provide list.
- Can define own goal.
- Major focus of the last third of the class.
- Important to start early.
  - After labs end, some slow down.
  - That's fatal.
- This is the purpose and focus of the course.

Homework

- 7 assignments
- First (review assignment) due Wednesday
- Definitions
  - High-Z
  - Drive
  - Bus

Exams

- Two midterm exams.
- Done when focus switches to project.
- 32% of grade.
- Higher (grade, not time) variance than project.

Office hours

- Robert Dick: 3:00-4:30 Tu, Th in 2417-E EECS
- Will often be in lab
- TA and Matthew's hours on website

Outline

~~Technology Trends~~

~~Course Description/Overview~~

Review, Tools overview, ISA start

Verilog

- Not covered in course
- Review 270 material
- Do review homework problems
- Trial and error may work for Lab 1
  - Won't work for project
- Understand key differences w. SW languages (e.g., C)
  - E.g., nonblocking statement semantics

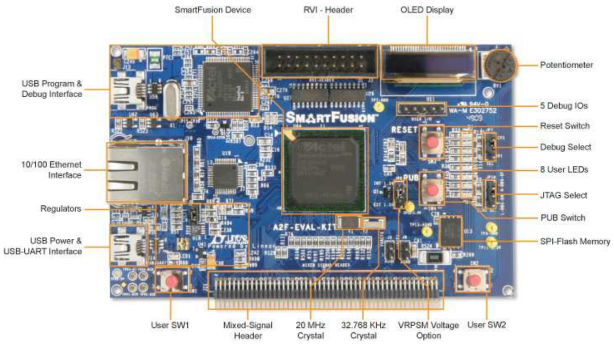
Net states

- What is a bus?
- What does “drive” mean?
- What does Hi-Z (high impedance) mean?
- Get started on HW1 before Monday.
  - Ask questions in class or on Piazza if you need help with definitions.
  - Concepts should have been covered in EECS 270

Crash course in debugging

- Biggest difference between experienced and novice engineers
  - Knowing your own mind's capabilities
- Complexity scales superlinearly in system size
- Heuristics
- Get something insanely simple working and grow it
- Verify the obvious
- Verify in order of dependency

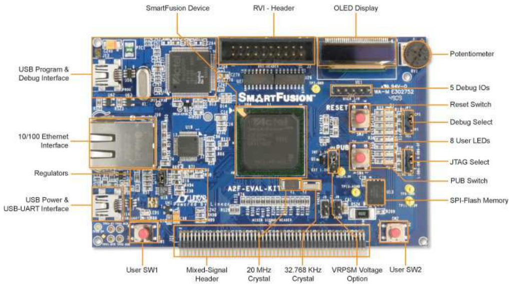
Actel’s SmartFusion Evaluation Kit



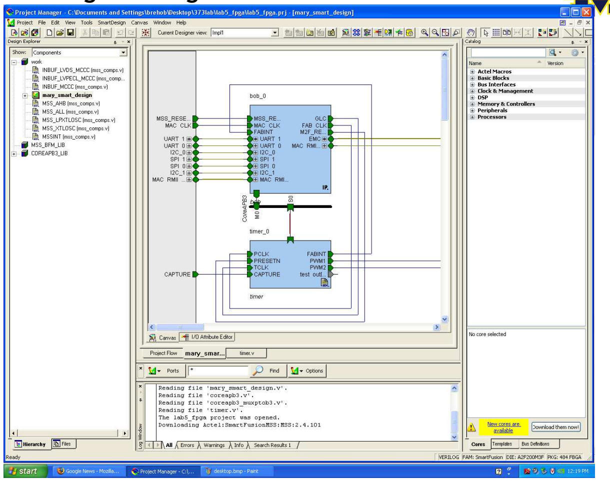
A2F200M3F-FGG484ES

- 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

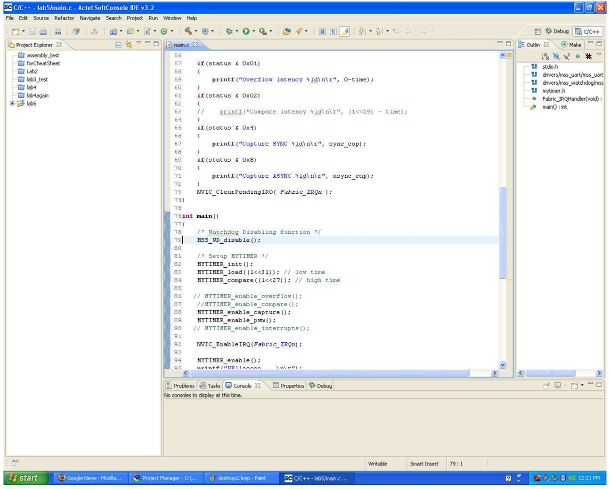


“Smart Design” configurator



```
Reading file 'mary_smart_design.v'.
Reading file 'coreapb3.v'.
Reading file 'coreapb3_muxptob3.v'.
Reading file 'timer.v'.
The lab5_fpga project was opened.
Downloading Actel:SmartFusionMSS:MSS:2.4.101
```

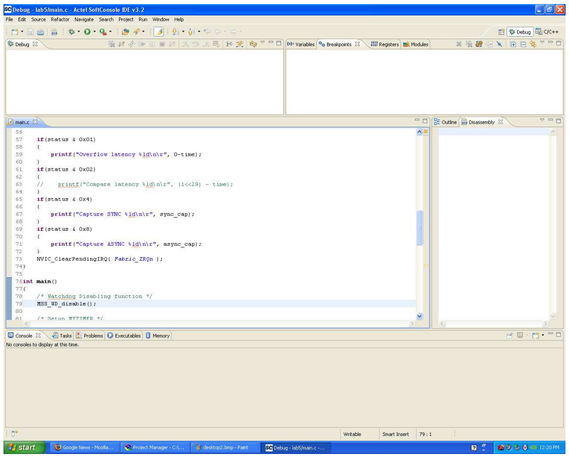
Eclipse-based “Actel SoftConsole IDE”



```
    if(status & 0x01)
    {
        printf("Overflow latency %ld\n\r", 0-time);
    }
    if(status & 0x02)
    {
    //    printf("Compare latency %ld\n\r", (1<<29) - time);
    }
    if(status & 0x4)
    {
        printf("Capture SYNC %ld\n\r", sync_cap);
    }
    if(status & 0x8)
    {
        printf("Capture ASYNC %ld\n\r", async_cap);
    }
    NVIC_ClearPendingIRQ( Fabric_IRQn );
}

int main()
{
    /* Watchdog Disabling function */
    MSS_WD_disable();

    /* Setup MYTIMER */
    MYTIMER_init();
    MYTIMER_load((1<<31)); // low time
    MYTIMER_compare((1<<27)); // high time

   // MYTIMER_enable_overflow();
    //MYTIMER_enable_compare();
    MYTIMER_enable_capture();
    MYTIMER_enable_pwm();
   // MYTIMER_enable_interrupts();

    NVIC_EnableIRQ(Fabric_IRQn);

    MYTIMER_enable();
```

No consoles to display at this time.

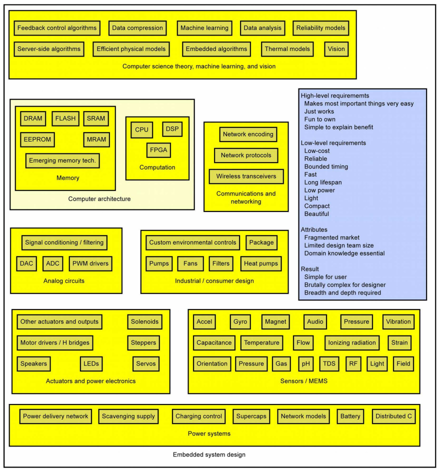
Debugger is GDB-based. Includes command line.



```

    if(status & 0x01)
    {
        printf("Overflow latency %ld\n\r", 0-time);
    }
    if(status & 0x02)
    {
    //    printf("Compare latency %ld\n\r", (1<<29) - time);
    }
    if(status & 0x4)
    {
        printf("Capture SYNC %ld\n\r", sync_cap);
    }
    if(status & 0x8)
    {
        printf("Capture ASYNC %ld\n\r", async_cap);
    }
    NVIC_ClearPendingIRQ( Fabric_IRQn );
}

int main()
{
    /* Watchdog Disabling function */
    MSS_WD_disable();

    /* Setup MYTIMER */
```

No consoles to display at this time.

An embedded system



High-level requirememts
Makes most important things very easy
Just works
Fun to own
Simple to explain benefit

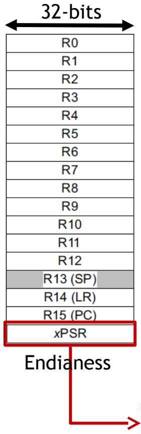
Low-level requirements
Low-cost
Reliable
Bounded timing
Fast
Long lifespan
Low power
Light
Compact
Beautiful

Attributes
Fragmented market
Limited design team size
Domain knowledge essential

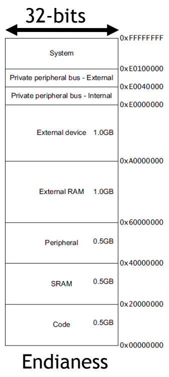
Result
Simple for user
Brutally complex for designer
Breadth and depth required

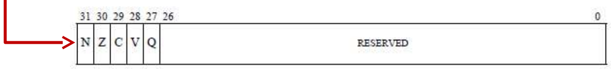
Major elements of an Instruction Set Architecture

(registers, memory, word size, endianess, conditions, instructions, addressing modes)



```
mov r0, #0x1

ld  r1, [r0,#5]

     r1=mem((r0)+5)

bne loop

subs r2, #1
```





Endianness

- Little-Endian (default)
  - LSB is at lower address

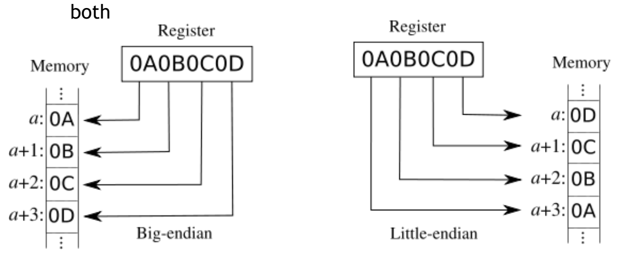
```
                                 Memory     Value
                                 Offset  (LSB) (MSB)
                                 ======  ===========
uint8_t a  = 1;                  0x0000  01 02 FF 00
uint8_t b  = 2;
uint16_t c = 255; // 0x00FF
uint32_t d = 0x12345678;         0x0004  78 56 34 12
```

- Big-Endian
  - MSB is at lower address

```
                                 Memory     Value
                                 Offset  (LSB) (MSB)
                                 ======  ===========
uint8_t a  = 1;                  0x0000  01 02 00 FF
uint8_t b  = 2;
uint16_t c = 255; // 0x00FF
uint32_t d = 0x12345678;         0x0004  12 34 56 78
```

Addressing: Big Endian vs Little Endian (370 slide)

- Endianness: 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, ARM supports both



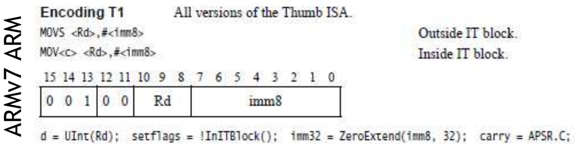
Instruction encoding

- Instructions are encoded in machine language opcodes

```
Instructions
movs r0, #10

movs r1, #0
```

```
Register Value         Memory Value
001|00|000|00001010 (LSB) (MSB)
(msb)         (lsb) 0a 20 00 21
001|00|001|00000000
```

ARMv7 ARM



Encoding T1 All versions of the Thumb ISA.

MOVS <Rd>,#<imm8> Outside IT block.

MOV<c> <Rd>,#<imm8> Inside IT block.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	Rd			imm8							

```
d = UInt(Rd);  setflags = !InITBlock();  imm32 = ZeroExtend(imm8, 32);  carry = APSR.C;
```

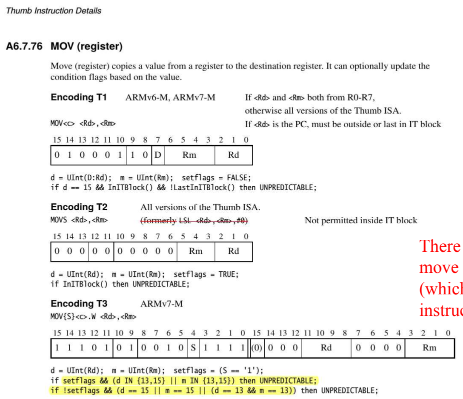
Assembly example

```
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
        ldrb r2, [r1]
        add r1, 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)



Thumb Instruction Details

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

If <Rd> and <Rm> both from R0-R7, otherwise all versions of the Thumb ISA.

`MOV<c> <Rd>,<Rm>`

If <Rd> is the PC, must be outside or last in IT block

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		

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

Encoding T2 All versions of the Thumb ISA.

`MOVS <Rd>,<Rm>` ~~(formerly LSL <Rd>,<Rm>,#0)~~

Not permitted inside IT block

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	Rm			Rd		

```
d = UInt(Rd);  m = UInt(Rm);  setflags = TRUE;
if InITBlock() then UNPREDICTABLE;
```

Encoding T3 ARMv7-M

`MOV{S}<c>.W <Rd>,<Rm>`

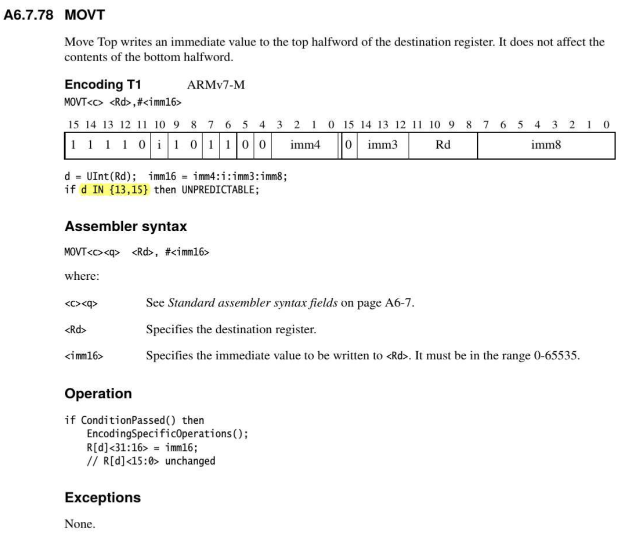
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	1	0	0	1	0	S	1	1	1	1	(0)	0	0	0	Rd				0	0	0	0	Rm			

```
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	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	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
```

Exceptions

None.

Loads!

- ldrb?
- ldrsb?

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

Done.