Ripped From The Headlines

- **OpenBTS:**
  - "A software-based GSM access point, allowing standard GSM-compatible mobile phones to make telephone calls without using existing telecommunication providers' networks."
- **Any random Linux machine can be a cell phone base station at 10% of previous cost**
- **Uses existing:**
  - VoIP software to turn calls into data
  - PBX software (like Asterix) to route calls
- **Island of Niue is going to use it**
- **http://openbts.sourceforge.net/**

Last Time

- **Embedded systems introduction**
  - Definition of embedded system
  - Common characteristics
  - Kinds of embedded systems
  - Crosscutting issues
  - Software architectures
  - Choosing a processor
  - Choosing a language
  - Choosing an OS

Today

- **ARM and ColdFire**
  - **History**
  - **Variations**
  - **ISA (instruction set architecture)**
  - **Both 32-bit**
  - **Also some examples from**
    - AVR: 8-bit
    - MSP430: 16-bit

Embedded Diversity

- **There is a lot of diversity in what embedded processors can accomplish, and how they accomplish it**
- **Example**
  - General purpose processors can perform multiplication in a single cycle
  - Mid-grade microcontrollers will have a HW multiply unit, but it’ll be slow
  - Low-end microcontrollers have no multiplier

Lots of chips...

- **Freescale – top embedded processor manufacturer with ~28% of total market**
  - HC05, HC08, HC11, HC12, HC16, ColdFire, PPC, etc.
  - Largest supplier of semiconductors for the automobile market
- **ARM – the most popular 32-bit architecture**
  - By 2008 ARM had shipped 10 billion processors
  - ARM population > human population
  - 5 billion chips predicted to ship in 2011

Brief ColdFire History

- **1979 – Motorola 68000 processors first ship**
  - Forward-thinking instruction set design
  - Inspired by PDP-11 and others
  - 32-bit architecture with 16-bit implementation
  - Basis for early Sun workstations, Apple Lisa and Macintosh, Commodore Amiga, and many more
- **1994 – ColdFire core developed**
  - 68000 ISA stripped down to simplify HW
- **2004 – Motorola Semiconductor Products Sector spun off to create Freescale Semiconductor**
**Brief ARM History**

- 1978 – Acorn started
  - Make 6502-based PCs
  - Most sold in Great Britain
- 1983 – Development of Acorn RISC Machine begins
  - 32-bit RISC architecture
  - Motivation: snubbed by Intel
- 1990 – Processor division spun off as ARM
  - “Advanced RISC Machines”
- 1998 – Name changed to ARM Ltd.

- Fact: ARM sells only IP
  - All processors fabbed by customers

**ARM=RISC, ColdFire=CISC?**

- Instruction length
  - ARM – fixed at 32 bits
  - Simpler decoder
  - ColdFire – variable at 16, 32, 48 bits
  - Higher code density
- Memory access
  - ARM – load-store architecture
  - ColdFire – some ALU ops can use memory
  - But less than on 68000
- Both have plenty of registers

**ARM Family Members**

- **ARM7 (1995)**
  - Three stage pipeline
  - ~80 MHz
  - 0.06 mW / MHz
  - 0.97 MIPS / MHz
  - Usually no cache, no MMU, no MPU

- **ARM9 (1997)**
  - Five stage pipeline
  - ~150 MHz
  - 0.19 mW / MHz + cache
  - 1.1 MIPS / MHz
  - 4-16 KB caches, MMU or MPU

**More ARM Family**

- **ARM10 (1999)**
  - Six-stage pipeline
  - ~260 MHz
  - 0.5 mW / MHz + cache
  - 1.3 MIPS / MHz
  - 16-32 KB caches, MMU or MPU

- **ARM11 (2003)**
  - Eight-stage pipeline
  - ~335 MHz
  - 0.4 mW / MHz + cache
  - 1.2 MIPS / MHz
  - configurable caches, MMU

**New ARM Chips: Cortex**

- **Cortex-A8**
  - Superscalar
  - 1 GHz at < 0.4 W

- **Cortex-A9**
  - Superscalar, out of order
  - Can be multiprocessor
  - This is the iPad processor

- **Cortex-R4** – real-time systems
  - So far, not very popular

- **Cortex Continued**

- **Cortex-M0, M1, M3, M4 – small systems**
  - Intended to replace ARM7TDMI
  - Intended to kill 8-bit and 16-bit CPUs in new designs
  - Most variants execute only Thumb-2 code
  - Some are below $1 per chip

- **M0 is really small**
  - ~12,000 gates

- **M1 is intended for FPGA targets**

- **M3 is more or less equivalent to the ColdFire we’ll be using**

- **M4 is faster, up to a few hundred MHz**
Register Files

- Both ColdFire and ARM
  - 16 registers available in user mode
  - Each register is 32 bits
- ColdFire
  - A7 – always the stack pointer
  - Program counter not part of the register file
- ARM
  - r13 – stack pointer by convention
  - r14 – link register by convention: stores return address of a called function
  - r15 – always the program counter

ColdFire Registers

- 32 registers
- Data registers
- Address registers
- Stack pointer
- Program counter
- Condition code register

ARM Banked Registers

- 37 total registers
  - Only 18 available at any given time
  - 16 + cpsr + spsr
  - cpsr = current program status register
  - spsr = saved program status register
- Some register names refer to different physical registers in different modes
- Other registers shared across all modes
  - E.g. r0-r6, cpsr
- Why is banking supported?
- Banked registers seem to be going away
  - Thumb-2 doesn’t have it

ARM Instructions

- Classic three address code

int sum (int a, int b)
{
    return a + b;
}

00000008 <sum>:
0:
    8: e0800001 add r0, r0, r1
    c: e12ffe1 bx lr

ColdFire Instructions

- Classic two address code

int sum (int a, int b)
{
    return a + b;
}
MSP430 Instructions

- Two address code

```c
int sum (int a, int b)
{
    return a + b;
}
```

Now "int" is 16 bits, so we're only getting half as much work done

AVR Instructions

- Two address code

```c
int sum (int a, int b)
{
    return a + b;
}
```

Again "int" is 16 bits

But why is the code gross?

32-bit Add on AVR

```c
int sum (int a, int b)
{
    return a + b;
}
```

sum:
```c
    add r14, r15
    ret
```

8-bit processors can waste a lot of cycles doing this kind of thing

Ugh!

ARM7

```c
int smul (int x, int y)
{
    return x*y;
}
```

ColdFire code:

```c
smul:
    link a6,#0
    muls.l d1,d0
    unlk a6
    rts
```

Baseline AVR

```c
int smul (int x, int y)
{
    return x*y;
}
```

ColdFire code:

```c
smul:
    link a6,#0
    muls.l d1,d0
    unlk a6
    rts
```

ATmega128 (largish AVR):

```c
int smul (int x, int y)
{
    return x*y;
}
```

ColdFire code:

```c
smul:
    link a6,#0
    muls.l d1,d0
    unlk a6
    rts
```
int sdiv (int x, int y) {
    return x/y;
}

// ColdFire code:

sdiv:
   link   a6,#0
   divs.l d1,d0
   unlk   a6
   rts

ARM Integrated Shifting

◆ Most instructions can use a barrel shift unit “for free”
  > Improves code density?

int foo (int a, int b) {
    return a + (b << 5);
}

00000000 <foo>:
   0:e0800281 add r0, r0, r1, lsl #5
   4:e12fff1e bx lr

> What are the costs of this design decision?

ARM Conditional Execution

◆ When condition is false, squash the executing instruction
◆ Supports implementing (simple) conditional constructs without branches
  > Helps avoid pipeline stalls
  > Compensates for lack of branch prediction in low-end processors
◆ Unique ARM feature: Almost all instructions can be conditional
◆ Suffixes in instruction mnemonics indicate conditional execution
  > add – executes unconditionally
  > addeq – executes when the Z flag is set

Conditional Example

int max (int a, int b) {
    if (a>b) return a;
    return b;
}

000000bc <max>:
   bc:e1500001 cmp r0, r1
   c0:bla00001 movlt r0, r1
   c4:e12fff1e bx lr

Another example: GCD

int gcd (int i, int j) {
    while (i != j) {
        if (i>j) {
            i -= j;
        } else {
            j -= i;
        }
    }
    return i;
}
GCD assembly

```assembly
000000d4 <gcd>:
d4: e1510000 cmp r1, r0
d8: 012fff1e bxeq lr
dc: e1510000 cmp r1, r0
e0: a0601001 rsbge r1, r0, r1
e4: a1510000 cmp r1, r0
e8: lafffff0 bne dc <gcd+0x8>
f0: e12fff1e bx lr
```

GCD on ColdFire

```assembly
gcd:
link a6, #0
cmp.l d1, d0
beq.s *+16
cmp.l d1, d0
ble.s *+6
sub.l d1, d0
bra.s d0, d1
cmp.l d1, d0
bne.s *-12
unlk a6
rts
```

Multiply and Accumulate

- DSP codes such as FIR and IIR typically boil down to repeated multiply and add

```c
int inner (int k, int j) {
    int i;
    int result = 0;
    for (i=0; i < 10; i++) {
        result += data[k][j] * coeff[k][i];
    }
    return result;
}
```

Multiply and Accumulate

```assembly
00000000 <inner>:
0: e0800100 add r0, r0, r0, lsl #2
4: e59f3034 ldr r3, [pc, #52]; 40 <.text+0x40>
8: e0811200 add r1, r1, r0, lsl #4
c: e52de004 str r1, [sp, #4-4]
10: e793e101 ldr lr, [r3, r1, lsl #2]
14: e59f3288 ldr r3, [pc, #40]; 44 <.text+0x44>
18: e3a0c000 mov ip, #0; 0x0
1c: e0831180 add r1, r3, r0, lsl #3
20: e2822001 add r2, r2, #1 ; 0x1
24: e2820000 add r2, r2, #1 ; 0x1
28: e4913004 ldr r3, [r1], #4
2c: e352000a cmp r2, #10 ; 0xa
30: e02ce93 mla ip, r3, lr, ip
34: la000007 hse 24 <inner+0x24>
38: e3520000 cmp r0, ip
3c: e49d6004 ldr pc, [sp, #4]
40: 00000140 andeq r0, r0, r0, lsl #2
44: 00000000 andeq r0, r0, r0
```

Multiple-Register Transfer

- ColdFire:
  ```assembly
  movem.l d0-d7/a0-a6, (a7)
  ```

- ARM:
  ```assembly
  stmdb sp!, (r4, r5, r6, r7, r8, r9, s1, fp, lr)
  ```

- Improves code density
- More efficient – why?
- Main disadvantages?
  > Solutions?

ARM: Thumb

- Alternate instruction set supported by many ARM processors
- 16-bit fixed size instructions
  > Only 8 registers easily available
  > Saves 2 bits
  > Registers are still 32 bits
  > Drops 3rd operand from data operations
  > Saves 5 bits
  > Only branches are conditional
  > Saves 4 bits
  > Drops barrel shifter
  > Saves 7 bits
ARM: Thumb

- Natural evolution of RISC ideas for embedded processors
  - Low gate count in decode logic no longer as important
  - Still, decode shouldn’t be too hard
  - Want compact instructions to keep I-fetch costs low
- Why use Thumb?
  - 30% higher code density
  - Potentially higher performance on systems with 16-bit memory bus
- Why not use Thumb?
  - Performance may suffer on systems with 32-bit memory bus

Thumb Continued

- Thumb implementation
  - Thumb bit in the cpsr tells the CPU which mode to execute in
  - In Thumb mode, each instruction is decoded to an ARM instruction and then executed
- ARM-Thumb “Interworking”:
  - Calling between ARM and thumb code
  - Compiler will do the dirty work if you pass it the right flags
- How to decide which routines to compile as ARM vs. Thumb?
- Thumb2: Supposed to give code density benefit w/o performance loss
  - So theoretically Thumb and ARM support can be dropped from future chips

MCF52233

- This is the chip on our demo boards
- ColdFire v2 – low-end embedded
  - No MMU or FPU
  - Single issue
- 256 Kbyte Flash
- 32 Kbyte RAM
- 8ch x 12-Bit ADC
- QSPI, IIC, and CAN Serial ports
- Fast Ethernet Controller (FEC) and Ethernet Phy (ePHY)
- ~$7.00 in large quantities

M52233DEMO Board

- CPU
- Ethernet port
- USB port
- Serial port
- 3-axis accelerometer
- 4 user-controlled LEDs
- 2 user-controlled push switches
- 5k ohm pot
- Costs $99

Summary

- There’s wide diversity in what the HW will do for you
- ARM and ColdFire are important embedded architectures
  - Both are “modern”
  - Worth looking at in detail
- MSP430 is extremely low power
  - But not clear how it will compete with newer ARM devices
- AVR has a large entrenched market
  - Low-end AVRs are really tiny and will remain popular
  - Higher-end AVRs are in a difficult position against the Cortex M0