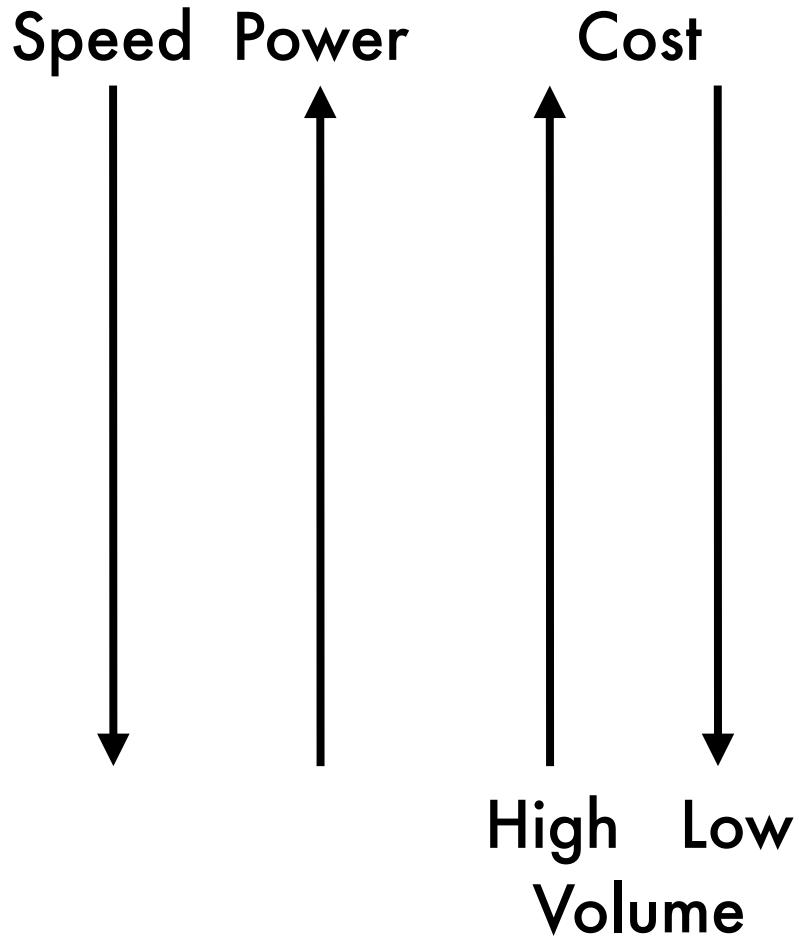


# Processing Choices

25

- Microprocessors
- Domain-specific processors
  - DSPs
  - Network processors
  - Microcontrollers
- ASIPs
- Reconfigurable SoC
- FPGA
- ASIC

See L3 notes



# Hardware vs. Software

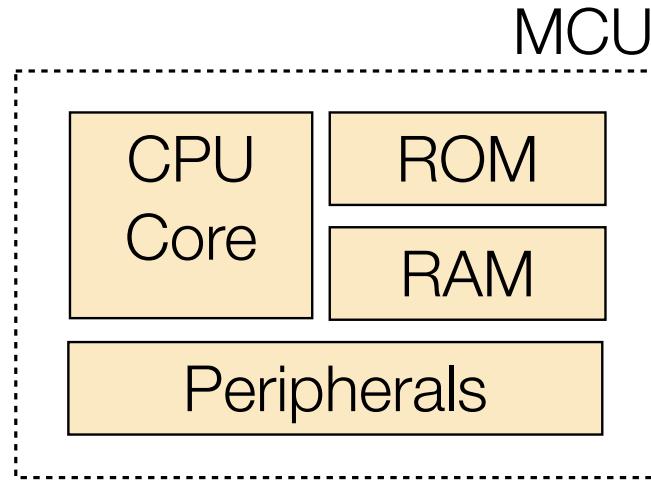
26

- **Hardware** = functionality implemented via a custom architecture
    - e.g., datapath + controller (FSM)
  - **Software** = functionality implemented on a programmable processor
  - **Key differences:**
    - **Multiplexing**
      - Software modules multiplexed with others on a processor
      - Hardware modules are typically mapped individually on dedicated hardware
    - **Concurrency**
      - Processors usually have one “thread of control”
      - Dedicated hardware often has concurrent datapaths
- allocate each func to sep. set of transistors  
multiplex in time for software using same set  
of transistors

"Think about Functionality first, corner cases, before jumping into Hardware and Software parts of the embedded systems."

# Microcontrollers (MCU)

27



ARM designs  
but licenses to  
other comp.  
↑  
support  
exists, well-  
documented IP

- Chip vendors either develop own CPU core or license IP
- Specific chips usually targeted toward a small set of applications

Microcontroller

vs.

Microprocessor

- entire computer  
in a chip (include  
CPU core, RAM,  
accelerators, etc.)

# Microcontrollers

28

- Different from your regular desktop CPU

- Smaller in size (transistor count)

- Reduced instruction set (less complex set)

- Less power consumption (less computationally capable)

- Lower frequencies

- Within microcontrollers, there is a large variation

code needs to be  
memory efficient

critical,  
constrained  
resource when  
writing software

Bus Width	CPU Speeds	RAM	ROM
8-bit	1-8 MHz	128-1K	512 to 10K
16-bit	4-25 Mhz	1K to 10K	10K to 128K
32-bit	10-1000 Mhz	10K to 512M	128K to 512M

# Atmel Atmega / TI MSP430

29

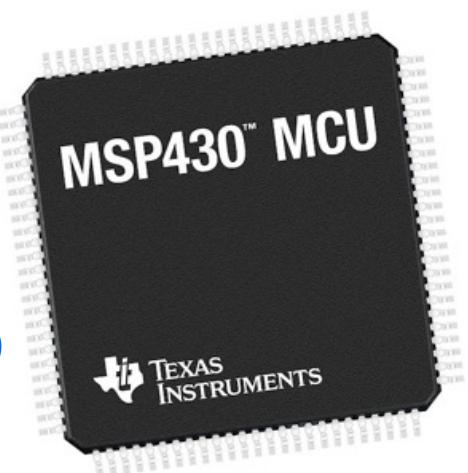
- Examples: toys, small appliances, automotive, etc.
- Atmel chips are 8-bit, TI chips are 16-bit
- Very low sleep power ( $\sim 5 \mu\text{A}$ )
- Rapid wake up ( $\sim 10 \mu\text{s}$ )  
Better for event-based triggers
- Rich set of peripherals
  - Timers, counters
  - Wired communication modules
  - Watchdog timers
  - Brownout detection
  - ADC/DAC

Microcontrollers are extremely efficient

at doing nothing

↳ Brownout Detection

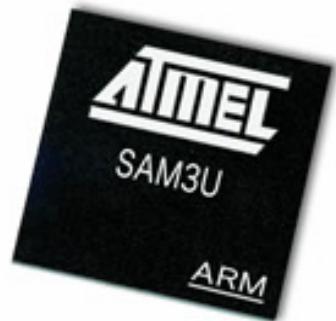
↳ Low-power optimization (i.e., Sleep current)



# ARM7 / Cortex-M3 / Cortex-A8

30

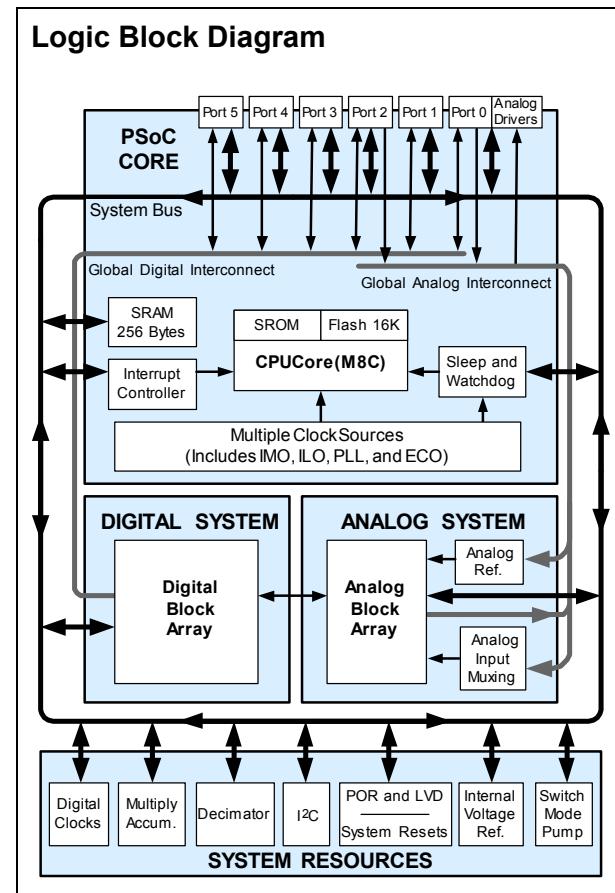
- Examples: Smartphone, ebook reader, automotive, etc
- CPU core licensed by ARM
- Many different manufacturers with different set of peripherals
- Tens to hundreds of MHz core speed
- 32-bit data bus
- Several 10k of RAM
- Several 100k of ROM
- Rich set of peripherals
- Recent cores very power efficient and still have very low sleep power!



# Cypress PSoC - Programmable System-on-Chip

31

- Programmable System-on-Chip
- The core
  - M8C, 8051, Cortex-M3
  - Flash memory, SRAM
  - Watchdog, Multiple clock sources
- Configurable Analog and Digital Blocks
  - Similar to CPLD or FPGAs
  - Blocks can be combined to offer
    - ADC
    - Counters
    - Amplifiers
- Programmable Routing and Interconnects
- Some examples where they are used
  - Sonicare tooth brush
  - TiVo
  - Capacitive sensing of the iPod
- Cypress has its own radio module CyFi



# Digital Signal Processors

32

- Similar to MCUs in architecture
- CPU core optimized for complex numeric tasks
  - Basic execution unit: multiply+accumulate
  - Data intensive operations
  - Usually used as co-processor
  - Signal filtering
  - Video Compression / Decompression
- Example: Analog Devices Blackfin
  - Uses ADI-Intel “Micro Signal Architecture”
  - Runs embedded OS (e.g., uCLinux), doesn’t need another host CPU
  - 600 MHz and below
- Example: Texas Instruments C64x
  - Used primarily for data encoding / decoding
  - Coupled with ARM Cortex A8 (as host CPU) in OMAP architecture
  - 1 GHz and below



# Communication Interfaces

33

- Communication is an important aspect of embedded systems
- Often contain specialized communication chips
- **Wired**
  - Interfacing with sensors and other system components (like comm. chips)
    - CAN
    - I2C
    - SPI
    - UART
    - USB
  - Communication with other embedded systems
    - Ethernet
- **Wireless**
  - Becoming more and more important because of ease of installation
  - Many different standards for short, mid, and long range communication

virtually no embedded system that is stand-alone!



# Wireless Technologies

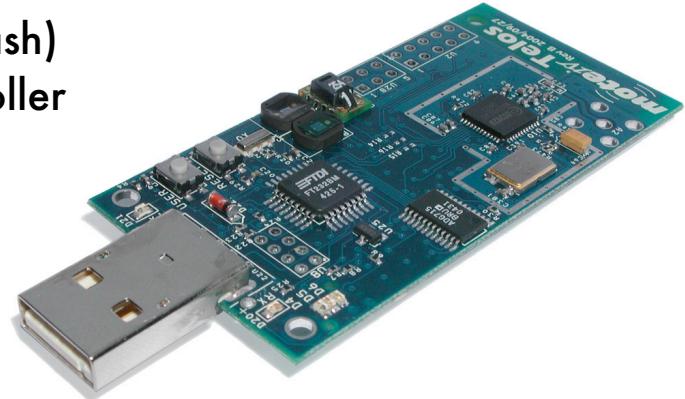
34

- Short Range
  - IEEE 802.15.4, ZigBee Alliance
    - Home automation
    - Sensor Networks
  - z-Wave
    - Home automation
  - Bluetooth  has taken over the market
    - ↳ virtually every cell-phone has this interface
  - Proprietary
- Mid Range
  - 802.11  pretty much the only option for mid range
    - ↳ some microcontrollers are embedded with this.
- Wide Area Networks
  - GSM/CDMA
  - Satellite
  - Proprietary point to point links

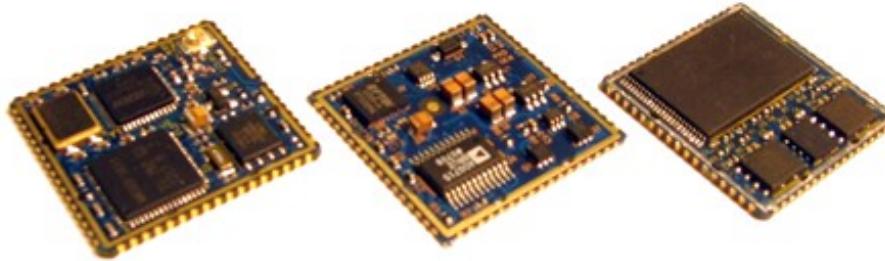
# Telos Platform

35

- Key Features
  - CC2420 Radio (2.4 GHz, IEEE 802.15.4)
  - MSP430F1611 (8MHz, 10k RAM, 48k Flash)
  - Integrated 12 bit ADC/DAC, DMA Controller
  - Onboard PCB Antenna 50-125m range
  - Ultra low sleep current (<10uA)
  - Rapid wakeup (<6us)
- Programming via USB or JTAG



- Key Features
  - CC2420 Radio
  - MSP430F1611 MCU
  - Modular design! *cool!*
  - Core module (MCU, Radio, Serial Flash)
    - Essentially Telos in different form factor
    - Modularity promotes reuse
  - USB module (FTDI chip, Battery Power)
  - Storage module (1 Gbit NAND, 2x 16 Mbit NOR, 512 Kbit FRAM)



Core

USB

Storage

# Epic-based Systems

37



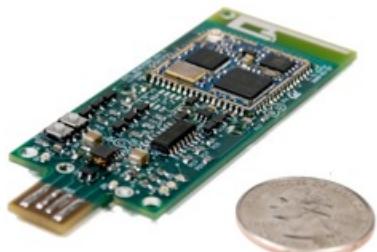
ACme AC Meter



Irene Mote



PowerNet Mote



Benchmark Mote



Hydro Watch

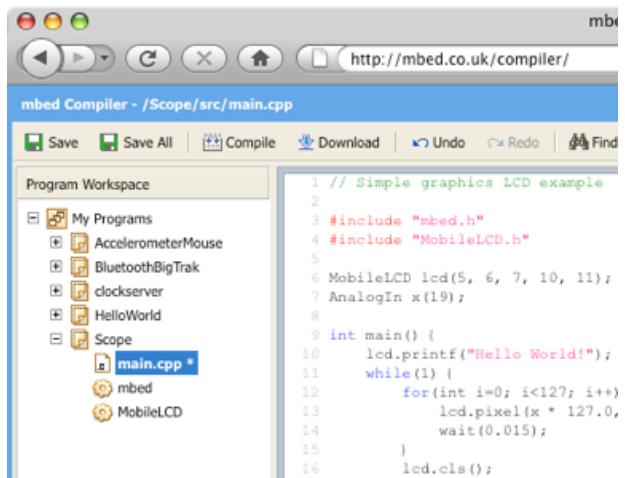
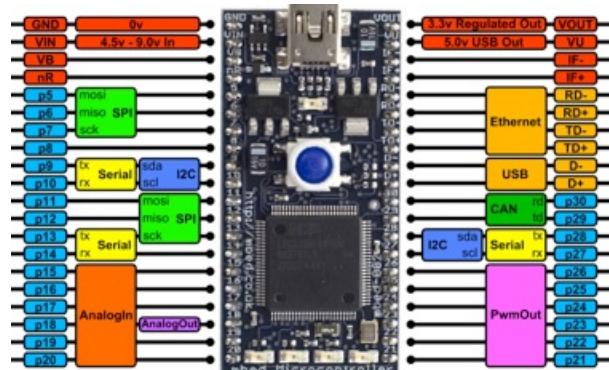


Meraki Interface

# mBed

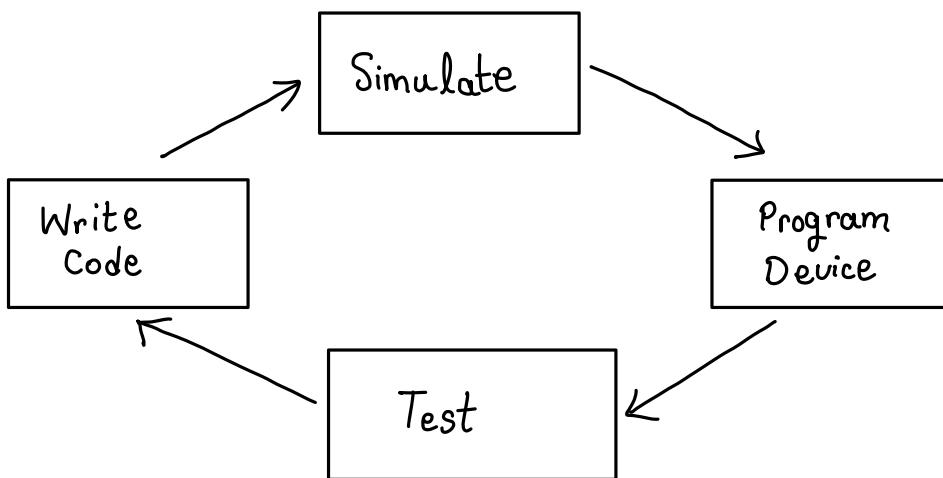
38

- Cortex-M3 Core running at 96MHz, with 512KB FLASH, 64KB RAM and a load of interfaces including Ethernet, USB Device, CAN, SPI, I2C and other I/O
- “Cloud” compiler
  - Web-based tool chain, lots of libraries and documentation, and good community support
  - Significantly simplifies the learning curve



```
1 // Simple graphics LCD example
2
3 #include "mbed.h"
4 #include "MobileLCD.h"
5
6 MobileLCD lcd(5, 6, 7, 10, 11);
7 AnalogIn x(19);
8
9 int main() {
10     lcd.printf("Hello World!");
11     while(1) {
12         for(int i=0; i<127; i++) {
13             lcd.pixel(x * 127.0,
14             wait(0.015);
15         }
16         lcd.cls();
17     }
18 }
```

# EMBEDDED SW DEVELOPMENT



## Programming

- Many possible languages
  - Assembly (MCU specific) *extremely high-perf. code*
  - C/C++, NesC — domain specific
  - X - Java — difficult to have timing determinism between soft/hardwr.
  - Verilog / VHDL
  - Arduino programming Language (based on .wiring)
  - Python
- Cross compiler translates high-lvl code to MCU Specific code instructions
  - cross compiler usually runs on your development (host) system
  - Generated binary does not usually execute on host system
  - Needs an emulator/simulator to interpret

## Simulation

- Two possible ways
  - Target Instruction lvl simulation:
    - each binary instruction is "executed" on a virtual target processor
    - relatively slow
  - Block lvl Simulation:
    - Black Box approach; only effects of blocks are modeled
    - very fast, but not as flexible
- Integral part of many smartphone IDEs
  - Android (Qemu based, instruction lvl)
  - iPhone (XCode, block lvl)
  - Palm Pre (VisualBox emulator, block lvl)
- Many MCUs have their own simulators
  - Atmel (Mica2/Mica2z) : Aurora
  - MSP430 (Telos) : MSPsim, NoICE for MSP430

# Programming Embedded Devices

- Each class of device is programmed slightly differently
- 8/16-bit class devices : ATmega, MSP 430
  - have a small on-chip hardware unit called a bootstrap loader (UIISP, BSL)
  - an external programming device communicates with the BSL through multiplexed pins, e.g., the serial interface, USB
  - these devices can also be programmed via JTAG pins
  - typically, the BSL will erase and overwrite the entire flash memory
- 32-bit class devices : Cortex-M3, ARM7, PXA27x
  - have a software bootloader that is programmed once using JTAG
  - bootloader always starts up first and loads the required OS Kernel
  - bootloader has board-specific drivers that can communicate with the host machine thru a serial port or network connection
  - typically, only parts of the flash are erased and overwritten
  - once OS is booted, can use NFS to load apps into RAM

## Testing/Debugging

- Software bugs
  - estimated to cost the US economy just under \$60B in 2002
  - introduced into programs thru many vectors
  - prevention & diagnosis is an active area of research and engineering
- Debugging Distributed Embedded Systems
  - LEDs: indicating status
  - printf : useful but not always possible
    - usually over UART
    - very manual & tedious
  - Debugger : lets you step instruction by instruction
    - usually needs physical connectivity (JTAG)
    - recent research has proposed wireless debuggers

