Overview of Embedded Systems in Medical Applications

Embedded Systems
Simplistic definition

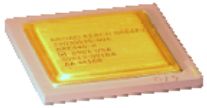
Embedded System

Shorthand for Embedded Processor System

Embed microprocessor + fixed program in non-computer system

Computer

Microprocessor Inside



Not a Computer

Fundamental Architectural Abstractions

Digital computer

Machine that can be programmed to process symbols

Symbols

Instruction

- Symbols describing processing of other symbols
- Machine can interpret instructions in its machine language

Data

- Symbol with no intrinsic meaning to machine
- Meaning of data imposed by user

Microprocessor (μP)

Hardware device

- Interprets instructions
- Performs instructions (processes) on data

Program

Defines sequence of instructions

General Purpose Computers and Embedded Systems

A basic distinction

	General Purpose (GP)	Embedded System
Microprocessor-based	√	√
Microprocessor performs instructions on data	√	√
User can access + add + modify data	√	√
User can access + add + modify programs	√	
Examples	PC, workstation, file server, mainframe	Toys, toasters, phones, DVD players, car engines, cameras, medical devices
Share of microprocessors manufactured	1%	99%

Why Embedded Processors?

Example — washing machine controller

Devices

Water heater / pump, level / temperature gauges, motor, indicators

Before 1990

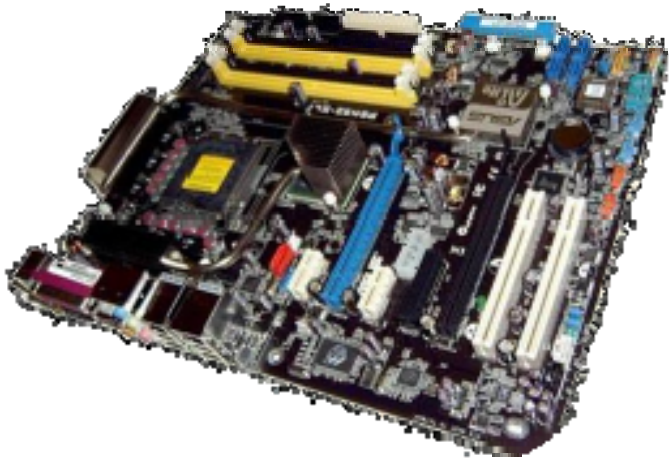
Motorized mechanical controller

- Operates devices in pre-set sequence
- Gears, cogs, mechanical switches

Modify controller for each model

Update ⇒ redesign physical controller

After 1990



Microprocessor controller

- Operates devices in programmed sequence
- Digital timing and electronic switches

Modify controller program for each model

Update ⇒ change program code

Economic Considerations

Pure hardware products

Product cost

Cost = NRE + MC × N

- NRE = non-recurring engineering = R&D cost (salaries)
- MC = manufacturing cost per unit
- N = number of units manufactured

Pure hardware products

MC → large

- Entire product is hardware
- Low product re-use ⇒ poor economies of scale

NRE → large

- Highly specific R&D
- Slow testing / update turn-around
- Shortage of electrical and mechanical engineers
  - Higher salaries (since 2000)

Economic Considerations

Pure software products

Product cost

Cost = NRE + MC × N

- NRE = non-recurring engineering = R&D cost (salaries)
- MC = manufacturing cost per unit
- N = number of units manufactured

Pure software products

MC → 0

- Entire product is software

NRE

- R&D is primary cost
- Complex development techniques to lower cost
  - Object oriented design, middleware platforms, standard libraries
- Complex design can harm performance and reliability
  - Multiple stages of processing
  - Standard software ⇒ less well understood

Economic Considerations

Embedded software products

Product cost

Cost = NRE + MC × N

NRE = non-recurring engineering = R&D cost (salaries)

MC = manufacturing cost per unit

N = number of units manufactured

Embedded software products

MC significant for large N

Embedded software ⇒ improved hardware re-use + economies of scale

NRE

Accept higher R&D cost to lower MC

Avoid highly complex software models

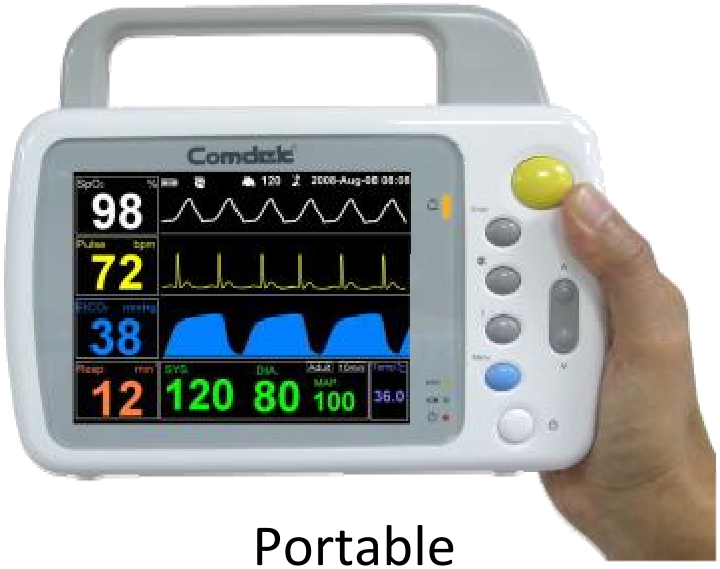
Program in core systems in C and assembly

Restrict memory usage

Embedded Medical Equipment

Familiar examples

Glucose
Test Set



Portable
EKG

MRI

Embedded Medical Equipment

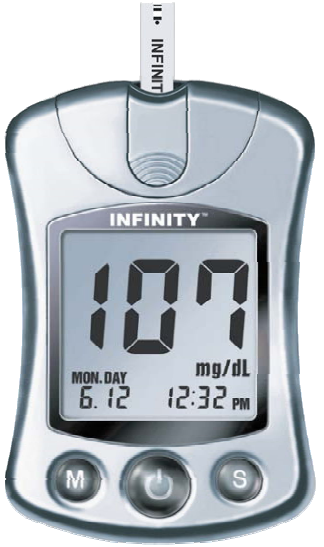
Glucose test set

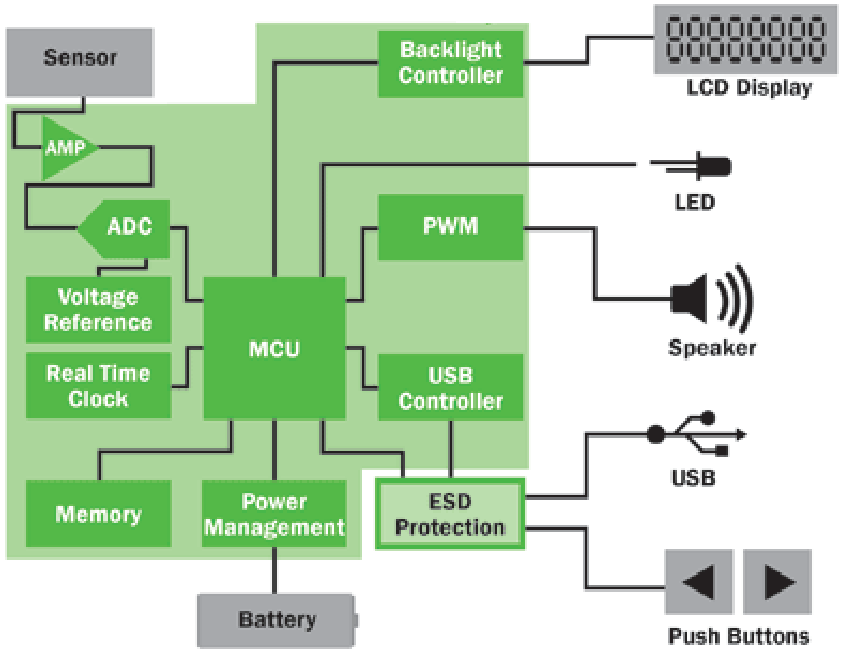
Components

- User interface (UI)
  - Buttons
  - Text-oriented screen
- Strip connector
- Battery
- Controller (system-on-chip)

Programming tasks

- Operate UI
  - Detect buttons
  - Display text
- Detect strip
- Control blood test cycle
- Test battery, temperature, program sanity, …

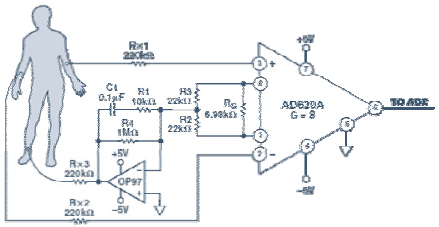


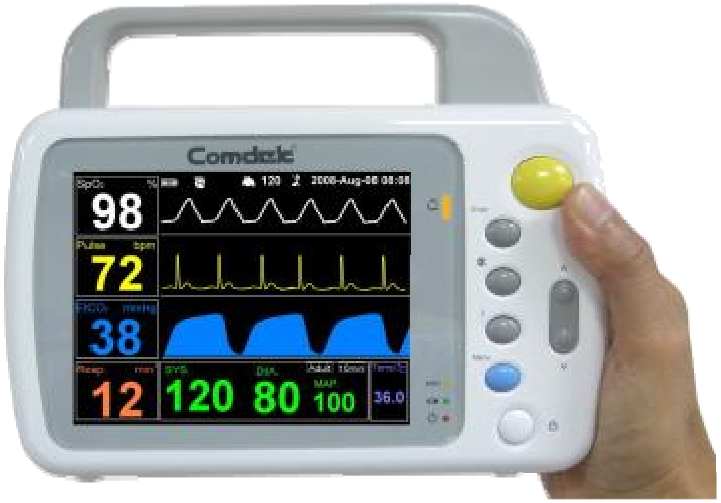


Embedded Medical Equipment

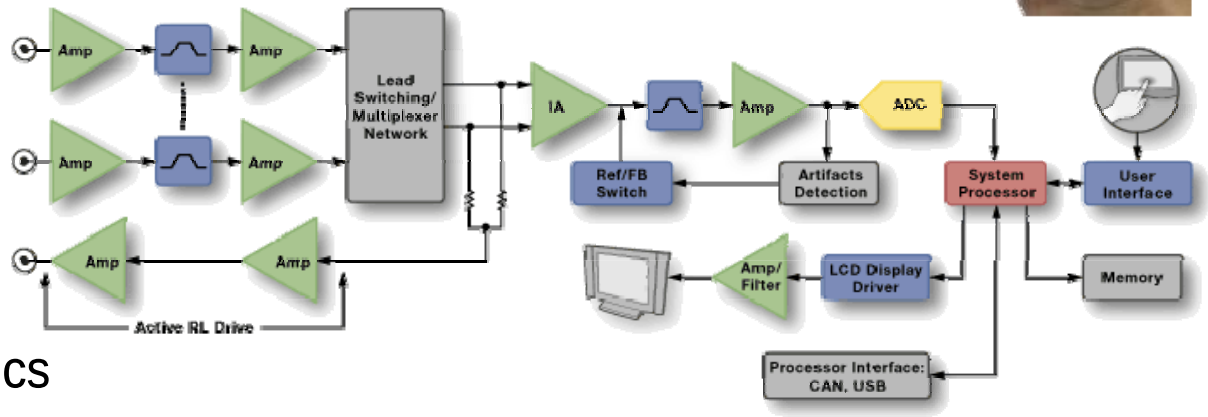
Portable EKG

Components





- User interface (UI)
  - Buttons
  - Graphics-oriented screen
- Electrode connector
- Battery
- Controller



Programming tasks

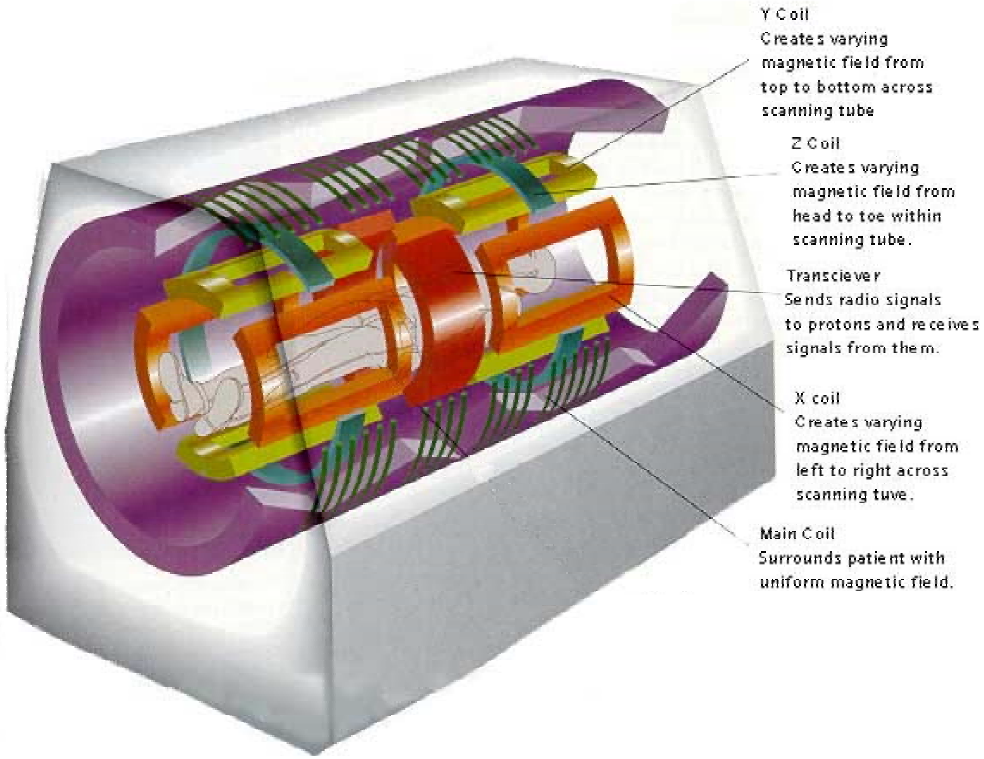
- Operate UI
  - Detect buttons
  - Display text + graphics
- Control electrode amplifiers + A/D
- Control digital signal processing (DSP) of input signals
- Generate graphic display signals
- Test battery, temperature, program sanity, ...

Embedded Medical Equipment

Magnetic Resonance Imaging (MRI)

Components

- Multiple user interfaces
- Patient table
- Magnetic field and RF sources
- Electromagnetic field detector
- Power supply
- Controller



Programming tasks

- Operate UIs
- Control table
- Control magnetic field and RF transmission
- Control electromagnetic field detector
- Imaging algorithms and graphics processing
- Test system sanity

Embedded Medical Equipment

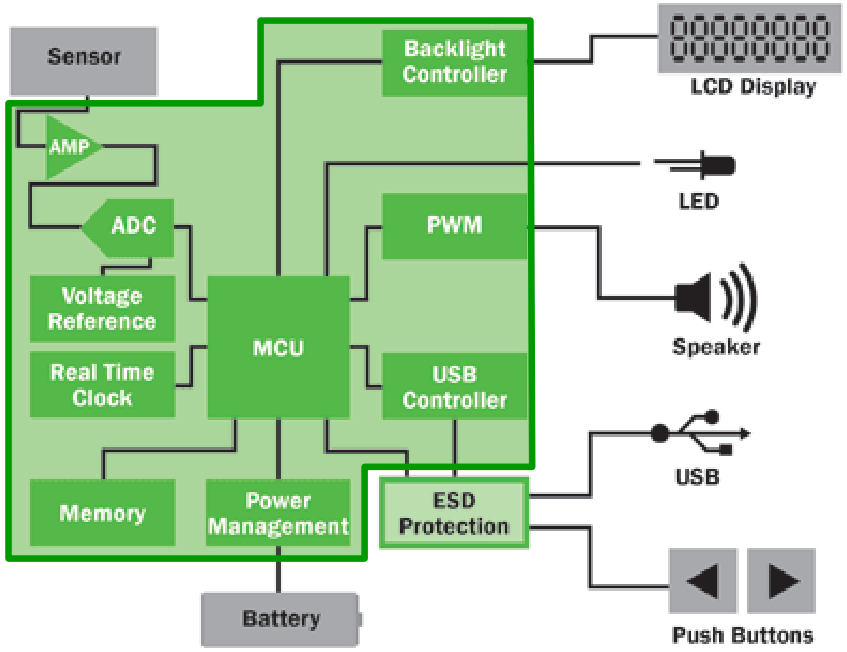
Division of labor

Hardware engineering

- Design of peripheral components
  - Sensors, actuators, displays
- Characterization of peripherals
- Design of device drivers
- Design of generic programmable embedded platforms
- Goal — specify hardware as API abstraction

Software engineering

- Defining system requirements
- Specifying generic platform
- High level system software planning
- Lower level software design
- Procedure / class coding
- Driver integration
- Goal — interact with hardware via API



Embedded Medical Equipment

Example: software technologies used in MRI

Programming Languages	C, C++/STL, C#, Visual Basic, ASP, JavaScript, Perl, Batch, and other proprietary languages
Persistent Storage	RDBMS, flat files, indexed and sequential files, XML
Interprocess communication	Socket, COM, shared memory, shared files
Source Code Size	8 million lines of code in 30,000 files
Timing accuracy	Scanning hardware: nanoseconds Real-time software: 1 millisecond User interaction: 0.1 –1 second

Philips Research, **Embedded Systems in Healthcare**, 2008

Typical System Structure

Hardware

- Microcontroller (μC)
  - Microprocessor (μP) + memory + timers + device controllers
- Devices
  - Sensors receive signals from physical environment
  - Actuators act on physical environment
  - Displays provide indications to user
- Specialized controllers — very large scale integrated circuits (VLSI)

Software

- Fixed program code (**FIRMWARE**) not directly accessible to user
- Initialization / reset routine
- Main loop
  - Polls devices for events requiring attention
  - Responds to interrupts from devices that require immediate attention
  - Continues at start of loop
- Handler functions for events

Hardware Environment

Microcontroller (μC)

- Microprocessor core (μP)
- Memory
  - RAM for data + ROM for program (firmware)
- Timers
  - Time internal / external events
  - Watchdog — timeout resets system if code loop fails
- Controller I/O
  - Interrupt controller — external event grabs processor attention
  - Analog ↔ digital converters (A/D and D/A)
  - Digital signal processor (DSP)
  - Serial ↔ parallel converters (UART)

Sensors

Position, contact, temperature, pressure, light, ...

Actuators

Displays, motors, solenoids, relays, valves, transmitters, ...

I/O in DVD Player

Sensors

- Switches on user interface
  - On/Off, REV, FF, Pause, Open/Close, Play, Stop, ...
- Motion detectors
  - Disk inserted, disk ejected, beginning of disk, end of disk, ...
- Safety detectors
- Dew detector, thermostat, over-voltage, no-signal, ...

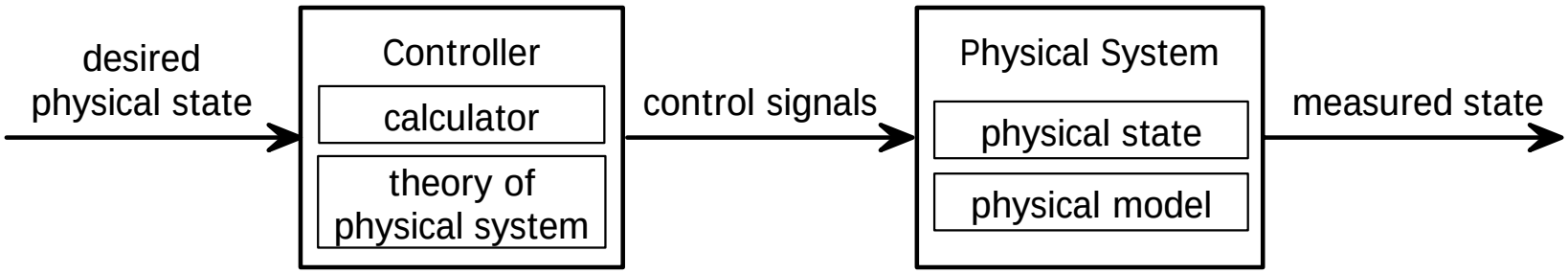
Actuators

- Disk drive motor, insert/eject motor
  - Motor turns
- Direction gear control solenoids
  - Solenoid pushes or pulls

Control Theory

Interacting with physical environment

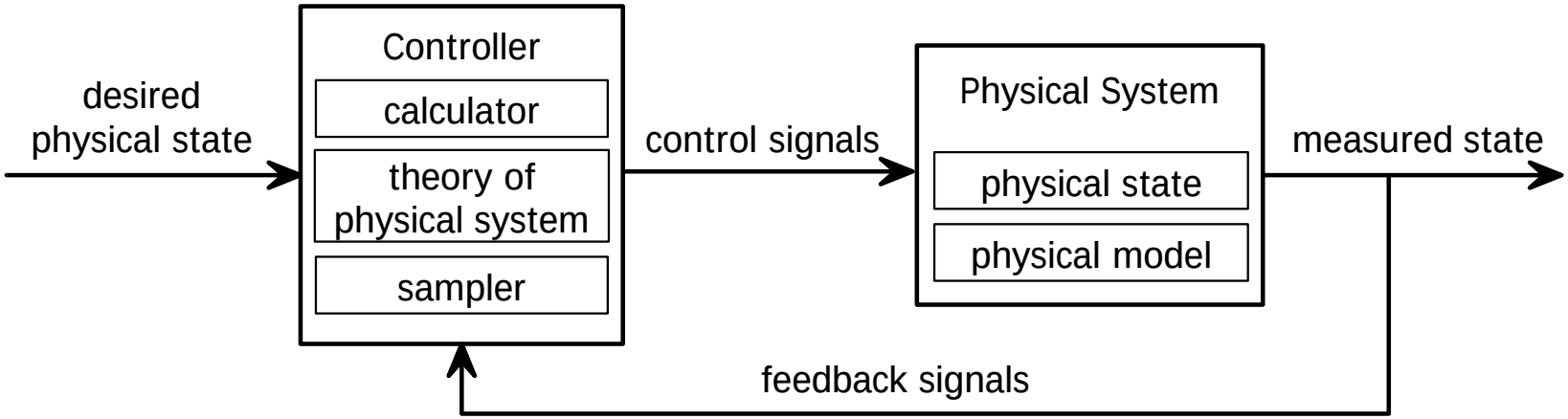
Open Loop Control



$$\mathtt{state}(t_{n+1}) = \mathtt{F}(\mathtt{state}(t_n), \mathtt{control}(t_n))$$

$$\mathtt{control}(t_{n+1}) = \mathtt{G}(\mathtt{target\ state}(t_n))$$

Closed Loop Control



$$\mathtt{state}(t_{n+1}) = \mathtt{F}(\mathtt{state}(t_n), \mathtt{control}(t_n))$$

$$\mathtt{control}(t_{n+1}) = \mathtt{G}(\mathtt{target\ state}(t_n), \mathtt{measured\ state}(t_n))$$

Simple Example

Controller for wireless mouse

Mouse contains RF transceiver

Transmits to USB adapter on PC

Mouse one of several PC devices

Identifies itself to PC with device ID

Mouse powered by battery

Sleeps after 5 seconds idle

Maintains state

Wakes up on motion or button click

Wireless Mouse Hardware

Switches

Mouse buttons

Motion sensors (mechanical or optical)

Detect X-Y position

Microprocessor

Translates switch and sensor outputs to motion data

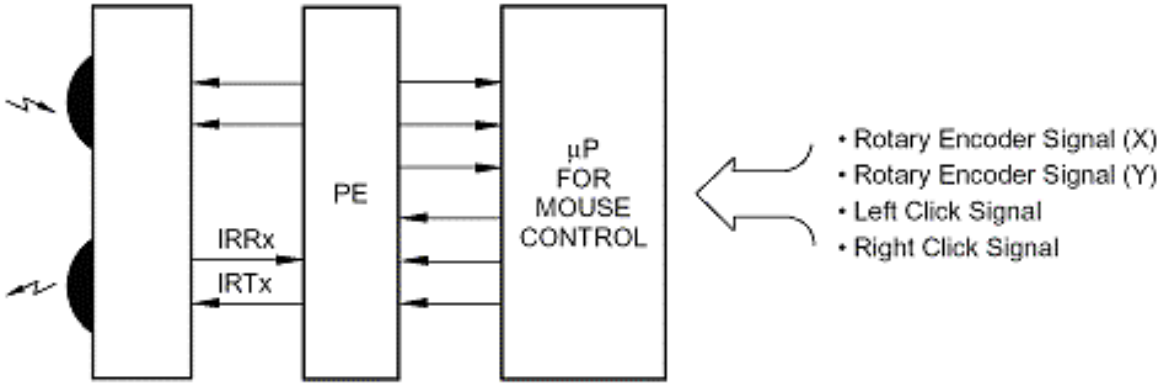
Manages battery usage and mouse status

Peripheral Engine (PE)

Controls RF link

Stores device information, button status, and X,Y-position

Radio Transceiver (Transmitter / Receiver)



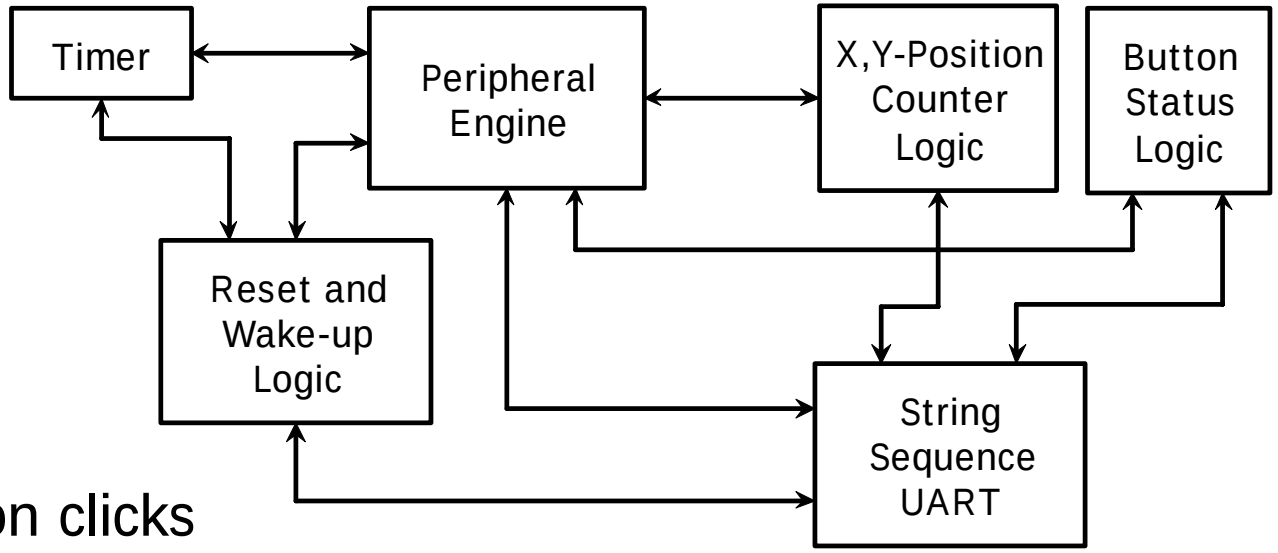
Mouse Routine (1)

```
        ; initialization
init:   request ID from PC                  ; send stored request string
        describe hardware to PC             ; send stored request string
        store X-Y position                  ; in memory buffer
        enable reset interrupt              ; watchdog timer
        enable wake-up interrupt            ; signal from X-Y or button
        zero 5 second timer                 ; CX ← 5 second count
                                            ; each loop: SUB CX, loop runtime
                                            ; CX == 0 ⇒ 5 second timeout
        ; main loop
L1:     read button status
        cmp button status, stored status    ; Peripheral Engine (PE) encodes
        JE L2                               ; status of 3 buttons and
        CALL button                         ; sends button status to PC
L2:     read motion detectors
        calculate X,Y-position
        CMP position, stored position       ; Peripheral Engine (PE) encodes
        JE L3                               ; X,Y-position and
        CALL motion                         ; sends position to PC
L3:     SUB timer, loop runtime             ; decrease timer
        CMP timer,0
        JE sleep                            ; if 5 seconds passed, sleep
        JMP L1                              ; continue
        ; end main loop
```

Mouse Routine (2)

```
            ; button press handler
button:     store button status in PE
            instruct PE to send button status to PC
            zero 5 second timer
            RET
            ; motion handler
motion:     store position in PE
            instruct PE to send X,Y-position to PC
            zero 5 second timer
            RET
            ; sleep state
sleep:      lock PE registers           ; store state
            turn-off battery to PE and transceiver
wait:       JMP wait                    ; low power do-nothing loop
                                        ; stays here until interrupt
            ; wake-up ISR
WAKE:       unlock PE registers         ; wake up
            JMP L1                      ; continue
            ; reset ISR
reset:      JMP init                    ; start
```

The Logic Gate Alternative



X,Y-position logic

Counts mouse motion clicks

Adjusts stored value in PE memory

Button status logic

Encodes button presses to PE

Reset and wake-up logic

Initializes and wakes up sleeping mouse

Puts idle mouse to sleep

String sequence logic

Transmits code strings to PC as bits

Gate Implementations

- Individual ICs
  - Large "footprint"
  - Costly manufacture
- Custom IC
  - Costly development

Goals in Embedded Design

Reliability

- Device operation depends on embedded processor
- Bug is inconvenient, expensive, possibly life-threatening
- Device must work 24/7/365 without user reset

Performance

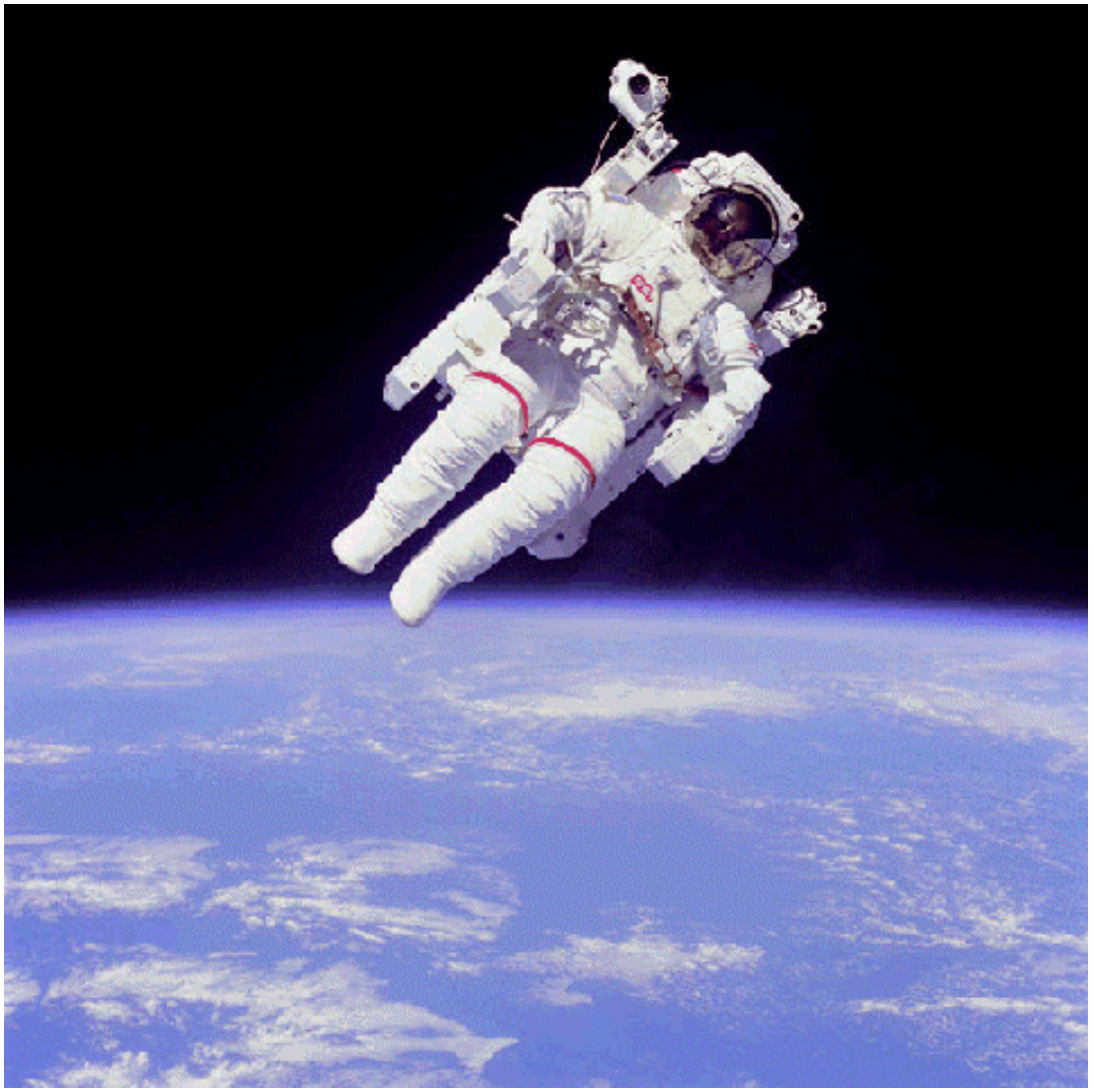
- Real time system
  - Respond in fixed time limits
  - Satisfy real world timing constraints
- Scheduling
- Optimized I/O

Cost

- Minimize manufacturing cost for consumer market
- Fast Time to Market
- No opportunity for future modification

Strict Requirements

Programmer's nightmare



NASA Photo ID: S84-27017 (February 11, 1984)

Multitasking / Concurrency in Embedded Systems

Controller handles simultaneous independent events

Inputs from external sensors

Internal processes

Outputs to external actuators

Programmable Thermostat

Three concurrent tasks

/* Monitor Temperature */	/* Monitor Time of Day */	/* Monitor Keypad */

do forever {
  measure temp ;
  if (temp < setting)
  start furnace ;
  else if (temp > setting + delta)
  stop furnace ;
}
	do forever {
	  measure time ;
	  if (6:00 am)
	  setting = 72°F ;
	  else if (11:00 pm)
	  setting = 60°F ;
	}
		do forever {
		  check keypad ;
		  if (raise temp)
		  setting++ ;
		  else if (lower temp)
		  setting-- ;
		}

Ref: Daniel W. Lewis, Fundamentals of Embedded Software: Where C and Assembly Meet

Design Process

Method

Top-down design — process abstraction to physical components

Bottom-up design — construct system to control existing hardware

Requirements

Define control problem and hardware abstraction

Architectural specification

Determine high level modules and information flow

Detailed system design

Determine functions and information flow within each module

Implementation

Coding in C and assembly language

Debugging

Initial debugging in code emulator

Testing code on target system

- Using I/O emulation
- Using physical platform

Debugging

No bugs allowed

- High performance demands
- No human operator to perform "workaround"
- No extended beta-testing ("service packs")

Hardware/software debugging

- Hardware configuration → time consuming
- Programmer must understand
  - Hardware behavior (well enough for debugging)
  - Hardware/software interactions
  - All possible external situations
  - Interaction of C code and assembly code
  - Compiler choices
- Debugging usually involves emulation of target system

Development Tools

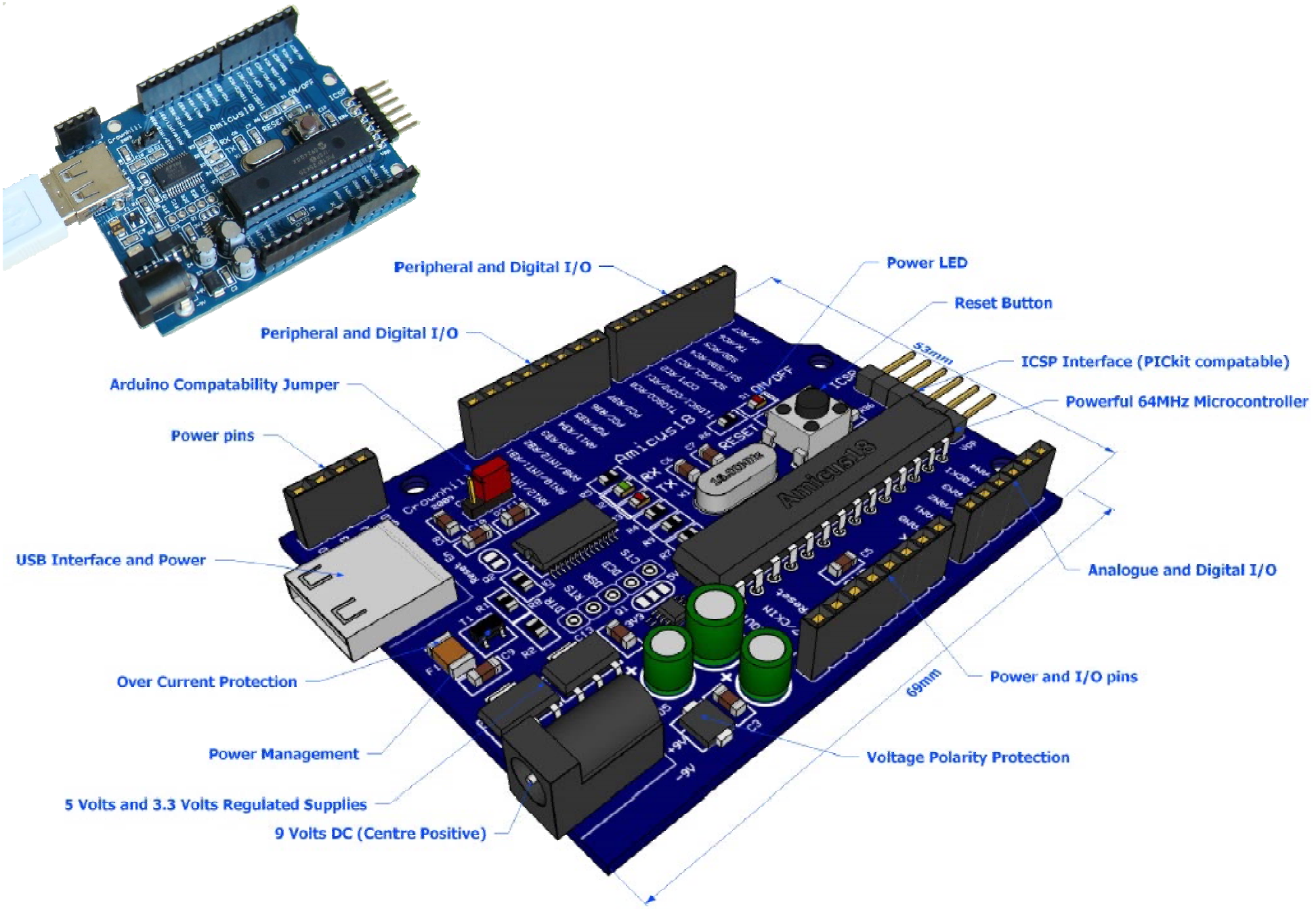
Language tools

- Program manager
  - Integrated code editor, file manager, device database
- Assembler, C compiler, linker
  - Assembly / C language to microcontroller machine language
- Debugger
  - Simulates machine code execution on PC
  - Simulates external I/O

Hardware tools

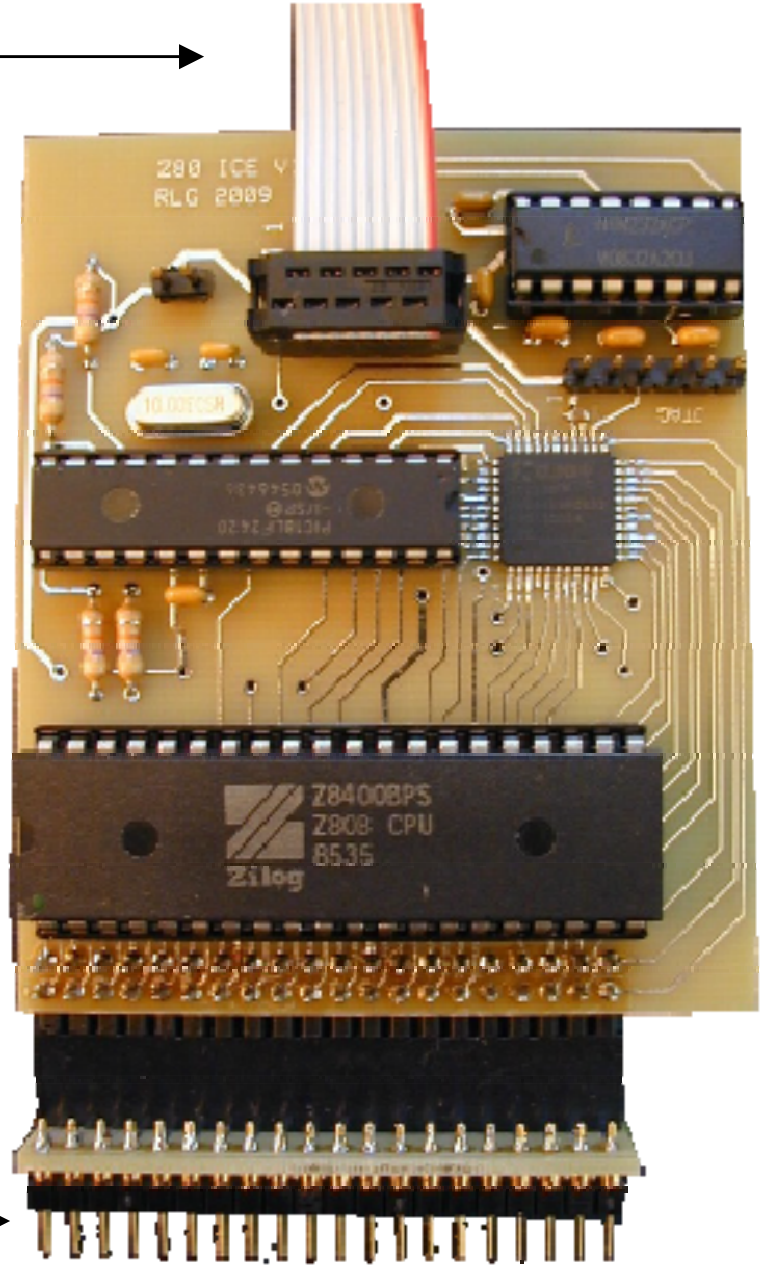
- Development platform
  - Generic circuit boards with microcontroller and external I/O devices
- Device programmer
  - Burn code from PC file to microcontroller internal ROM
- Hardware debugger
  - In-Circuit Emulator (ICE) — PC connection acts instead of controller
  - In-Circuit Debugger (ICD) — program real controller for debugging

Development Platform



In-Circuit Emulator (ICE)

Connection to PC →



Connection to target platform →