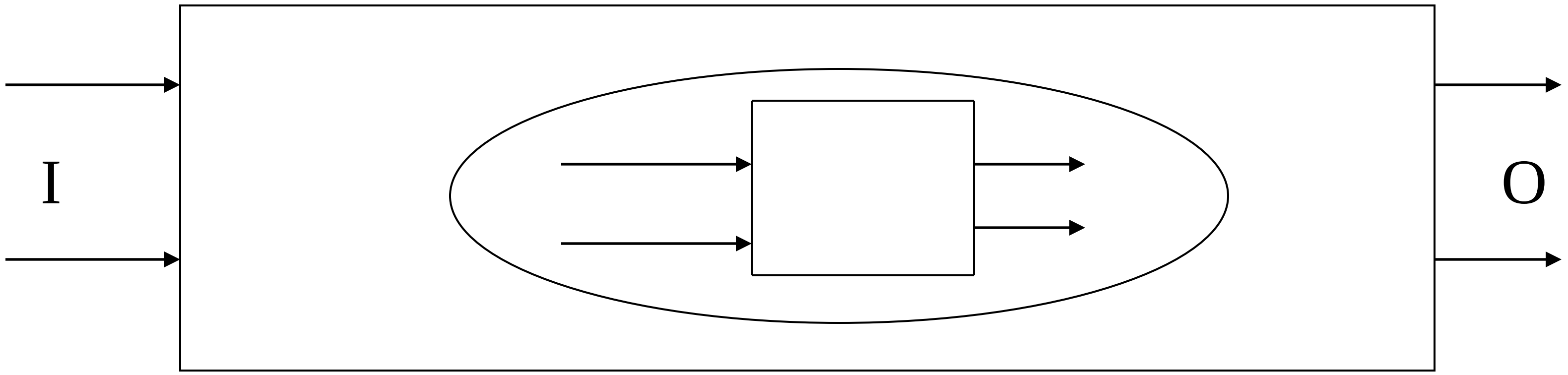
Embedded Systems

Theory and Design

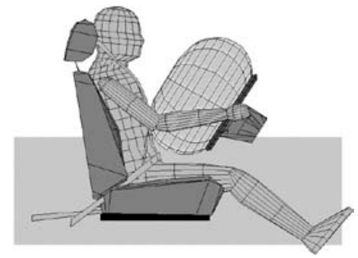
What is an Embedded System

An Embedded System is a microprocessor based system that is embedded as a subsystem, in a larger system (which ***may or may not be a computer system***).



Application areas

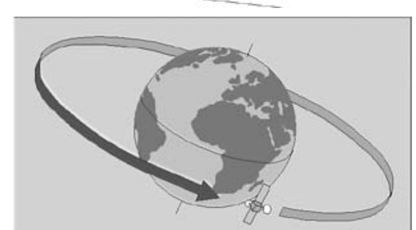
- Automotive electronics



- Aircraft electronics

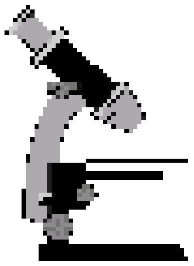
- Trains

- Telecommunication

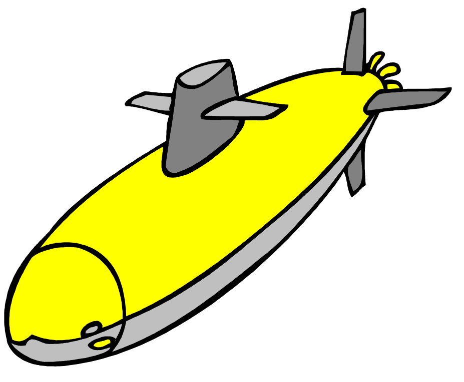


Application areas

- Medical systems



- Military applications

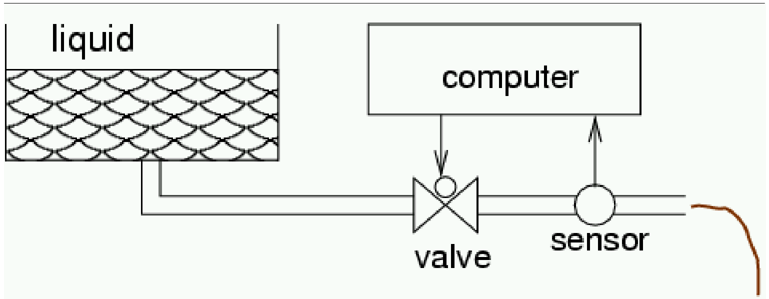


- Authentication

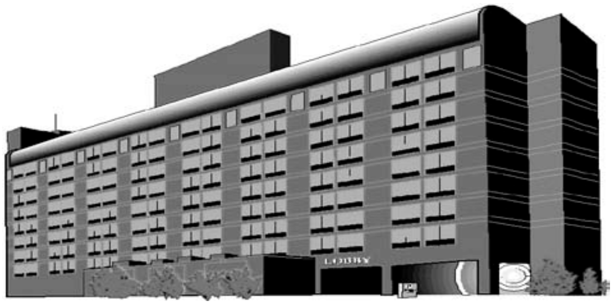
Application areas

- Consumer electronics

- Fabrication equipment



- Smart buildings



A Brief of Embedded System

- An **embedded system** is a special-purpose system in which the computer is completely encapsulated by or dedicated to the device or system it controls.
- Since the system is dedicated to specific tasks, design engineers can optimize it, reducing the size and cost of the product.
- Embedded systems are often mass-produced, benefiting from economies of scale.

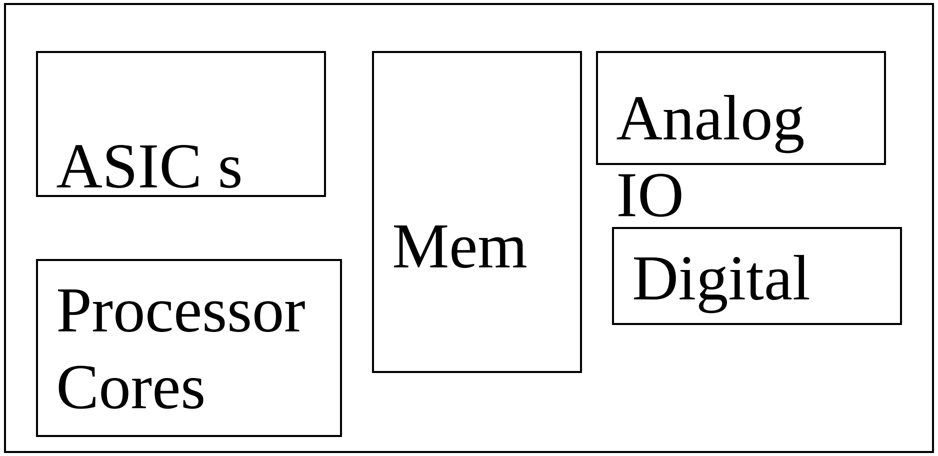
Examples of embedded systems

- automatic teller machines (ATMs)
- avionics, such as inertial guidance systems, flight control hardware/software and other integrated systems in aircraft and missiles
- cellular telephones and telephone switches
- engine controllers and antilock brake controllers for automobiles
- home automation products, such as thermostats, air conditioners, sprinklers, and security monitoring systems
- handheld calculators
- household appliances, including microwave ovens, washing machines, television sets, DVD players and recorders
- medical equipment
- Handheld computers
- Videogame consoles
- computer peripherals such as routers and printers
- Industrial controllers for remote machine operation.

History

- The first recognizably modern embedded system was the Apollo Guidance Computer, developed by Charles Stark Draper at the MIT Instrumentation Laboratory.
- The first mass-produced embedded system was the Autonetics D-17 guidance computer for the Minuteman (missile), released in 1961.
- In 1978 National Engineering Manufacturers Association released the standard for a programmable microcontroller.
- By the mid-1980s, widespread use of embedded systems became feasible with microcontroller.

General Characteristics of Embedded Systems



ASIPs and ASICs form a significant component

- **Adv: customization → lower power, cost and enhanced performance**
- **Disadv: higher development effort (debuggers, compilers etc.) and larger time to market**

General Characteristics of Embedded Systems

- **Perform a single task**
  - Usually not general purpose
- **Increasingly high performance and real time constrained**
- **Power, cost and reliability are important considerations**
- **HW-SW systems**
  - **Software is used for more features and flexibility**
  - **Hardware (processors, ASICs, memory etc. are used for performance and security**

Characteristics

- Some also have real-time performance constraints.
- An embedded system very often is physically built-in to the device it is controlling.
- The software written for embedded systems is often called firmware, and is stored in read-only memory or Flash memory chips rather than a disk drive.

Characteristics

- **User interfaces -** range from no user interface at all to full user interfaces similar to desktop operating systems in devices such as PDAs.
- **Complexity –** from simple embedded devices use buttons, LEDs to full graphical screen, with touch sensing or even World Wide Web interface (TCP/IP required)

Characteristics

- **CPU platform**
  - two distinct categories: microprocessors (μP) and microcontrollers (μC). μC have built-in peripherals on the chip, reducing size of the system.
  - CPU architectures used: ARM, MIPS, Coldfire/68k, PowerPC, x86, PIC, 8051, Atmel AVR, Renesas H8, SH, V850, FR-V, M32R, Z80, Z8
  - For small, low-volume embedded and ruggedized system. PC/104 and PC/104+ are used. They often use DOS, Linux, NetBSD, QNX, or VxWorks.
  - High-volume embedded systems use system on a chip (SoC), an application-specific integrated circuit (ASIC), or field-programmable gate array (FPGA) to execute the firmware.

Characteristics

- **Peripherals**
  - Serial Communication Interfaces (SCI): RS-232, RS-422, RS-485 etc
  - Synchronous Serial Communication Interface: I2C, JTAG, SPI, SSC and ESSI
  - Universal Serial Bus (USB)
  - Networks: Controller Area Network, LonWorks, etc
  - Timers: PLL(s), Capture/Compare and Time Processing Units
  - Discrete IO: aka General Purpose Input Output (GPIO)

Characteristics

- **Tools**
  - Generally, compilers, assemblers, and debuggers are used to develop embedded system software.
  - An in-circuit emulator (ICE) is a hardware device that replaces or plugs into the microprocessor, and provides facilities to quickly load and debug experimental code in the system.
  - For systems using digital signal processing, developers may use a math workbench such as MathCad or Mathematica to simulate the mathematics.
  - Software tools can come from several sources:
    - Software companies that specialize in the embedded market
    - Ported from the GNU software development tools
    - Sometimes, development tools for a personal computer can be used if the embedded processor is a close relative to a common PC processor.

Characteristics

- **Debugging**
  - at different levels, ranging from assembly- or source-level debugging with an in-circuit emulator or in-circuit debugger, to output from serial debug ports or JTAG/Nexus interfaces, to an emulated environment running on a personal computer.
  - As the complexity of embedded systems grows (e.g. cellphones, PDAs), higher level tools and operating systems (Linux, NetBSD, OSGi or Embedded Java) are migrating into machinery where it makes sense.

Characteristics

- **Reliability**
  - unreliable mechanical moving parts such as disk drives, switches or buttons are avoided.
  - Recovery from errors may be achieved with techniques such as a watchdog timer that resets the computer unless the software periodically notifies the watchdog.
  - Specific reliability issues may include:
    - "limp modes" that provide partial function. Examples include space systems, undersea cables, navigational beacons, bore-hole systems, and automobiles.
    - Backups are selected by an operator. Examples include aircraft navigation, reactor control systems, safety-critical chemical factory controls, train signals, engines on single-engine aircraft.
    - The system will lose large amounts of money when shut down: Telephone switches, factory controls, bridge and elevator controls, funds transfer and market making, automated sales and service.

Application Specific Characteristics

- Application is known before the system is designed
- System is however made programmable for
  - Feature upgrades
  - Product differentiation
- Often application development occurs in parallel to system development
  - Hw-Sw partitioning should be as delayed as possible
- For upgrades design reuse is an important criterion
  - IP reuse, object oriented development

Design Metrics

- Unit cost – the $ cost for each unit excluding development cost
- NRE cost: $ cost for design and development
- Size: The physical space reqd. – determined by bytes of sw, number of gates and transistors in hw
- Performance: execution time or throughput of the system
- Power: lifetime of battery, cooling provisions
- Flexibility: ability to change functionality without heavy NRE cost

Design Metrics (contd.)

- Time to market = Time to prototype + Time to refine + Time to produce in bulk
- Correctness: Test and Validation
- Safety:

- Often these metrics are contradictory – hence calls for optimization
- Processor choice, partitioning decisions, compilation knowledge
- Requires expertise in hw and sw both

Major Subtasks of Embedded System Design

- Modeling the system to be designed and constraints
  - Experimenting with different algorithms and their preliminary evaluation
  - Factoring the task into smaller subtasks and modeling their interaction
- Refinement
- HW-SW partitioning
  - Allocating the tasks into hw, sw running on custom hw or general purpose hw
- Scheduling – allocation of time steps for several modules sharing the same resource
- Implementation: Actual hw binding and sw code generation
- Simulation and Validation
- Iterate if necessary

What is Co-design?

- Traditional design
  - SW and HW partitioning done at an early stage and development henceforth proceeds independently
- CAD tools are focussed towards hardware synthesis
- For embedded systems we need several components
  - DSPs, microprocessors, network and bus interface etc.
- HW-SW codesign allow hw and sw design to proceed in parallel with interactions and feedback between the two processes
- Evaluation of trade offs and performance yields ultimate result

CAD for Embedded Systems

- Co-design: Joint optimization of hw and sw to optimize design metrics
- Co-synthesis: Synthesizes designs from formal specifications
- Rapid prototyping and design space exploration
- Many of the tasks are interrelated
- Intermediate evaluation is not easy as a later decision in one path affects the other

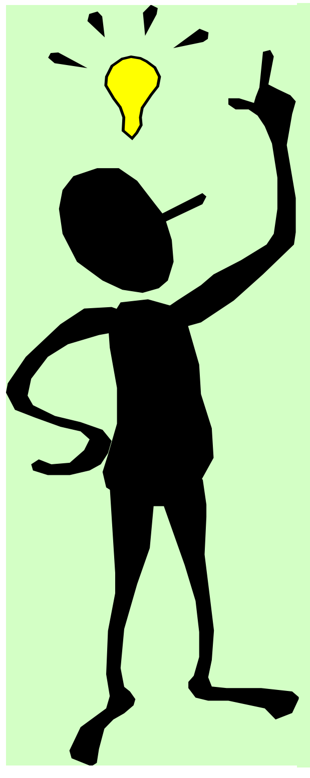
A Mix of Disciplines

- Application Domain (Signal processing, control …)
- Software Engg. (Design Process plays an important role)
- Programming Language
- Compilers and Operating System
- Architecture – Processor and IO techniques
- Parallel and Distributed Computing
- Real Time Systems

Importance of Embedded Software and Embedded Processors

"... the New York Times has estimated that the average American comes into contact with about 60 micro-processors every day...." [Camposano, 1996]

Latest top-level BMWs contain over 100 micro-processors [Personal communication]



Most of the functionality of embedded systems will be implemented in software!

Views on embedded System

- It is estimated that each year embedded software is written five times as much as 'regular' software
- The vast majority of CPU-chips produced world-wide today are used in the embedded market ... ; only a small portion of CPU's is applied in PC's
- ... the number of software-constructors of Embedded Systems will rise from 2 million in 1994 to 10 million in 2010;
  ... the number of constructors employed by software-producers 'merely' rises from 0.6 million to 1.1 million.

[Department of Trade and Industry/ IDC Benelux BV: Embedded software research in the Netherlands. Analysis and results, 1997 (according to: www.scintilla.utwente.nl/shintabi/engels/thema_text.html)]

Some problems

- How can we capture the required behaviour of complex systems ?
- How do we validate specifications?
- How do we translate specifications efficiently into implementation?
- Do software engineers ever consider electrical power?
- How can we check that we meet real-time constraints?
- How do we validate embedded real-time software? (large volumes of data, testing may be safety-critical)

Embedded software architectures

- **Simple control loop**
  - software simply has a loop. The loop calls subroutines, each of which manages a part of the hardware or software.
- **Interrupt controlled system**
  - tasks performed by the system are triggered by different kinds of events. (e.g. a timer, or by a serial port controller receiving a byte)
  - Usually there is a simple task in a main loop also. The tasks performed in the interrupt handlers should be as short as possibl
  - Some times longer tasks are added to a queue structure
- **Cooperative multitasking**
  - A nonpreemptive multitasking system is very similar to the simple control loop scheme, except that the loop is hidden in an API. (usually called "pause", "wait", "yield", etc.).
  - The advantages and disadvantages are very similar to the control loop, except that adding new software is easier.

Embedded software architectures

- **Preemptive multitasking**
  - A low-level piece of code (scheduler) switches between tasks based on a timer. It introduces all the complexities of managing multiple tasks running seemingly at the same time.
  - Tasks must be precisely separated. Access to shared data must be controlled by some synchronization strategy, such as message queues, semaphores or a non-blocking synchronization scheme.
  - It is common for organizations to buy a real-time operating system, allowing the application programmers to concentrate on device functionality rather than operating system services

Embedded software architectures

- **Microkernels and exokernels**
  - A microkernel can allocate memory and CPU time to different threads of execution. User mode processes implement major functions such as file systems, network interfaces, etc.
  - Exokernels communicate efficiently by normal subroutine calls. The hardware, and all the software in the system are available to, and extensible by application programmers.

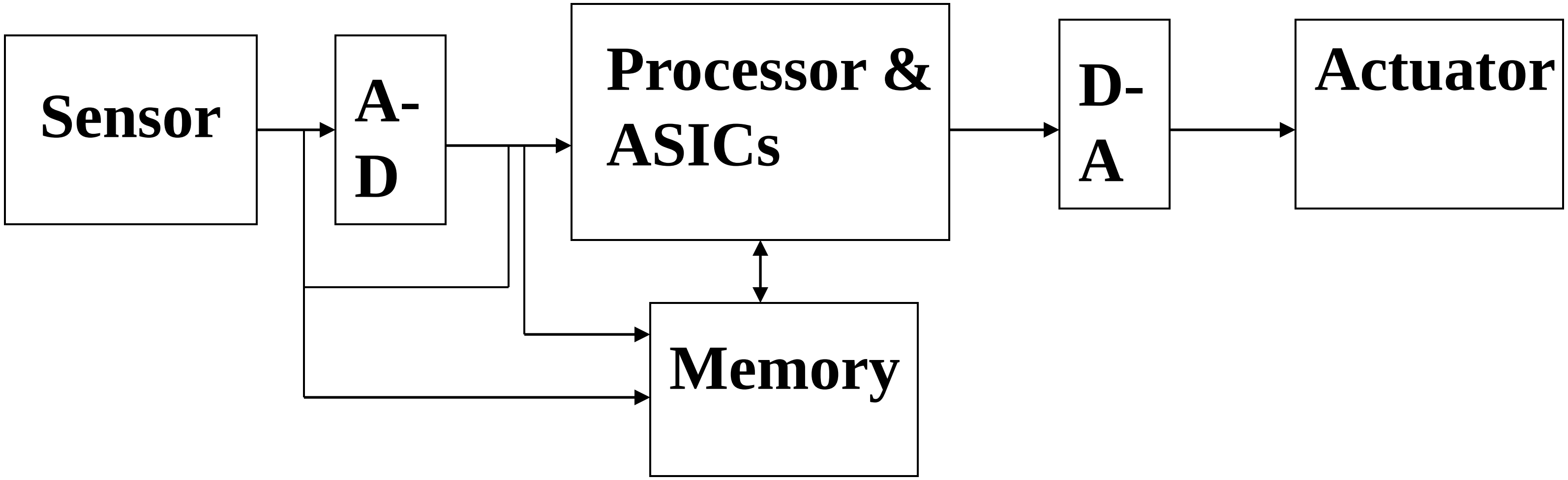
Embedded software architectures

- **Monolithic kernels**
  - A full kernel with sophisticated capabilities is adapted to suit an embedded environment.
  - It requires more hardware resources and can be less predictable and reliable.
  - Common examples are Embedded Linux and Windows CE.
  - This type of embedded system is increasing in popularity. Here are some of the reasons:
    - Ports to common embedded chip sets (ARM, x86, PowerPC) are available.
    - They permit re-use of publicly available code for Device Drivers, Web Servers, Firewalls, and other code.
    - Running application code in user mode is more reliable, easier to debug and that therefore the development process is easier and the code more portable.
    - A system such as Embedded Linux has fast enough response for many applications (real-time requirement).
    - Features requiring faster response than can be guaranteed can often be placed in hardware.
    - Many RTOS systems have a per-unit cost (royalty).

Essential Components

- Microprocessor / DSP
- Sensors
- Converters (A-D and D-A)
- Actuators
- Memory (On-chip and Off chip)
- Communication path with the interacting environment

Embedded System Structure (Generic)



Essential Considerations

- Response Time -- Real Time Systems
- Area
- Cost
- Portability
- Low Power (Battery Life)

- ❑ Fault Tolerance

Design Issues (Hardware-Software Co-design)

- System Specification
  - Functions, Real Time Constraints, Cost and Power Constraints
- Hardware Software Partitioning
- Hardware Synthesis
- Software Synthesis and Code Generation
- Simulation
- Implementation

ES, MS and RTS

- All embedded systems are microprocessor based systems, but all microprocessor based systems may not be amenable to embedding (Area, Power, Cost, Payload parameters).
- Most of the embedded systems have real time constraints, but there may be ES which are not hard RTS (for example off line Palm tops)
- There may be RTS which are not embedded (e.g. Separate Process Control Computers in a network)
- Embedded Systems are not GPS; they are designed for dedicated applications with specific interfaces with the sphere of control

Classification of Embedded Systems

- Distributed and Non distributed
- Reactive and Transformational
- Control dominated and Data dominated

DSP Characteristics

- Signals are increasingly being represented digitally as a sequence of samples
- ADCs are moving closer to signals; RFs are also treated digitally
- Typical DSP processing includes:
  - Filtering, DFT, DCT etc.
  - Speech and image: Compression, decompression, encryption, decryption etc.
  - Modems: Equalization, noise and echo cancellation, better SNR
  - Communication channel: encoding, decoding, equalization etc.

Distributed Characteristics

- Components may be physically distributed
- Communicating processes on multiple processors
- Dedicated hw connected through communicating channels

- Often economical
  – 4 x 8 Bit controllers may be cheaper than a 32 bit microcontroller
  – Multiple processors can perform multiple time critical tasks
  – Better logistics – devices being controlled may be physically distributed