IMC5-1EO

Embedded Operating Systems

Damien MASSON
http://esiee.fr/~massond/

Last modification: November 12, 2013

Presentation

By me

- 3 lessons: introduction, presentations, hints, pointers. A lot of personal work will be necessary: E-OS is a wild, huge domain. The lessons will just draw the picture.
- 3 practicals + Pers: programming with OSEK/VDX framework

By G. Bussignies

1 lesson + 1 practical: a multi-task application on an embedded system without operating system.

Sources

(to be updated)

- http://courses.cs.washington.edu/courses/cse466/
- http://www.rfc1149.net/download/documents/elecinf344/ (in french)
- http://trampoline.rts-software.org/

And a great Thanks to Emmanuel Grolleau (http://www.lias-lab.fr/members/emmanuelgrolleau) and Jean-Luc Béchennec (http://www.irccyn.ec-nantes.fr/~bechenne/).

What is an Embedded System ?

Definitions

- A device not independently programmable by the user.
- Specialized computing devices that are not deployed as general purpose computers.
- A specialized computer system which is dedicated to a specific task.
- An embedded system is preprogrammed to perform a narrow range of functions with minimal end user or operator intervention.

What is an Embedded System ?

Some Facts

- Embedded systems range in size from a single processing board to systems with operating systems.
- A combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function.
- In some cases, embedded systems are part of a larger system or product, as is the case of an anti-lock braking system in a car.
- A specialized computer system that is part of a larger system or machine.
- Typically, an embedded system is housed on a single microprocessor board with the programs stored in ROM.
- Some embedded systems include an operating system, but many are so small and specialized that the entire logic can be implemented as a single program.

What is an Embedded System ?

Examples

- Virtually all appliances that have a digital interface – watches, microwaves, VCRs, cars – utilize embedded systems.
- A computer system dedicated to controlling some non-computing hardware, like a washing machine, a car engine or a missile.
- Examples of embedded systems are medical equipment and manufacturing equipment.
- While most consumers aren't aware that they exist, they are extremely common, ranging from industrial systems to VCRs and many net devices.

What is an Embedded System ?

Examples

What is an Embedded System ?

More Facts

- Different than a desktop system
- Fixed or semi-fixed functionality (not user programmable)
- Different human interfaces than screen, keyboard, mouse, audio
- Usually has sensors and actuators for interface to physical world
- May have stringent real-time requirements

It may:

- Replace discrete logic circuits
- Replace analog circuits
- Provide feature implementation path
- Make maintenance easier
- Protect intellectual property
- Improve mechanical performance

What is an Embedded System ?

VS Non-Embedded Systems

Less emphasis on

- Graphical user interface
- Dynamic linking and loading
- Virtual memory, protection modes
- Disks and file systems
- Processes

What is an Embedded System ?

VS Non-Embedded Systems

More emphasis on

- Real-time support, interrupts (very small OS, if we're lucky)
- Tasks (threads)
- Task communication primitives
- General-purpose input/output
- Analog-digital/digital-analog converters
- Timers
- Event capture
- Built-in communication protocols

What is an Embedded System ?

Figures of merit

- Reliability – it should never crash
- Safety – controls things that move and can harm/kill a person
- Power consumption – may run on limited power supply
- Cost – engineering cost, manufacturing cost, schedule tradeoffs
- Product life cycle – maintainability, upgradeability, serviceability
- Performance – real-time requirements, power budget

What is an Operating System ?

Remember IMC4-1OS ?

An Operating System:

- Link between software and hardware
- Give abstraction of hardware to programmers
- Services (resources access, synchronization, file system...)
- Tests and Logs

Do Embedded Systems Need OS ?

NO

See Georges Bussignies lesson and practical

Do OS useful for Embedded Systems ?

YES: For the same reasons they are useful for “regular” systems

OS vs EOS ?

- Minimal approach:
  - the OS initialize hardware
  - give hand to the application
- Maximal aproach:
  - the OS gives all desktop services
  - provide virtual machines
  - provide libraries for advanced graphical interface
- Intermediate approach:
  - time services
  - network services
  - memory services

Embedded Operating Systems

What it does hardly depends of your needs!
Compromizes:

- Mono-task / multi-tasks
- Mono-thread / multi-threads
- Static allocations / dynamic allocations
- Resource reservation / on demand resource allocation
- ...

Memory Manager

Most common microcontrollers (see after) are not provided with a lot of memory

- 256k for STM32F427IG ;
- 20k for STM32F103 ;
- 1536 bytes for PIC18F452

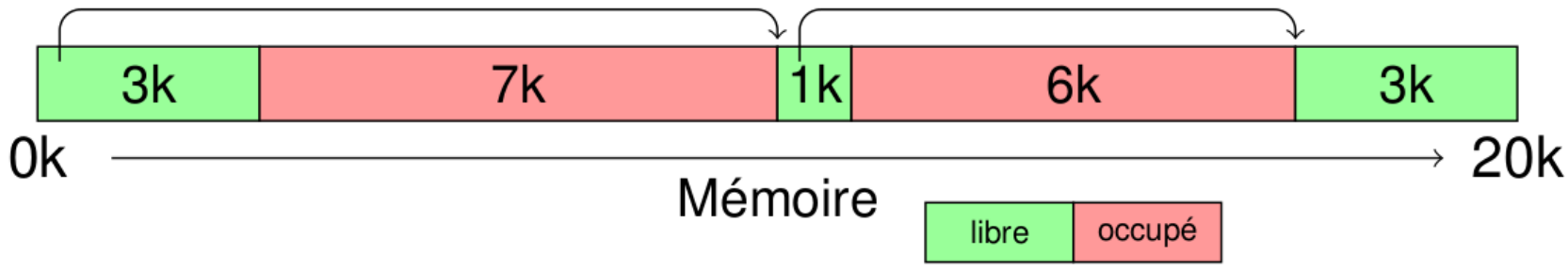
Add external RAM has a cost

- more I/O
- routing complexity increases
- access time

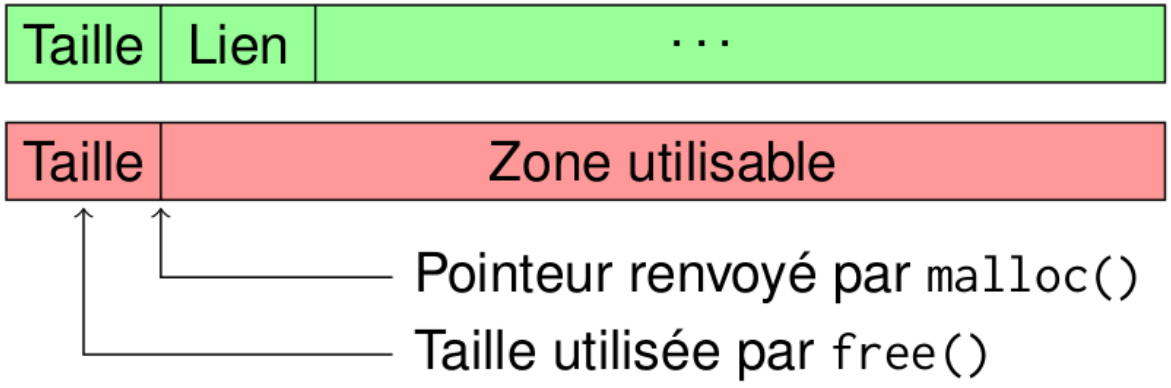
The memory resource has to be managed precariously

Memory Management: dynamic approaches

Free blocks linked list (free list)

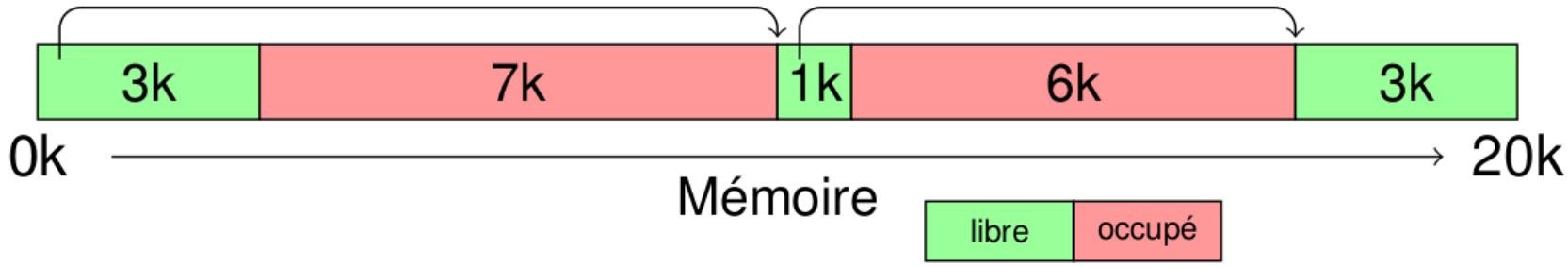


The size has to be save in memory (not passed to free())



Memory Management: dynamic approaches

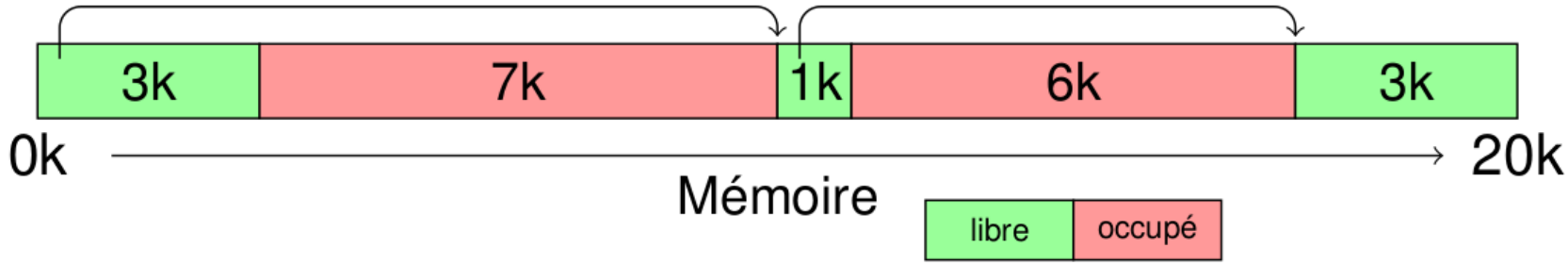
Main issue: memory fragmentation



How to allocate 5k ?

Memory Management: dynamic approaches

Allocation Policies



Where to allocate 600B ?

- Best fit
- Worst fit
- First Fit
- Best or Worst fit with first-fit using an ordered free block list

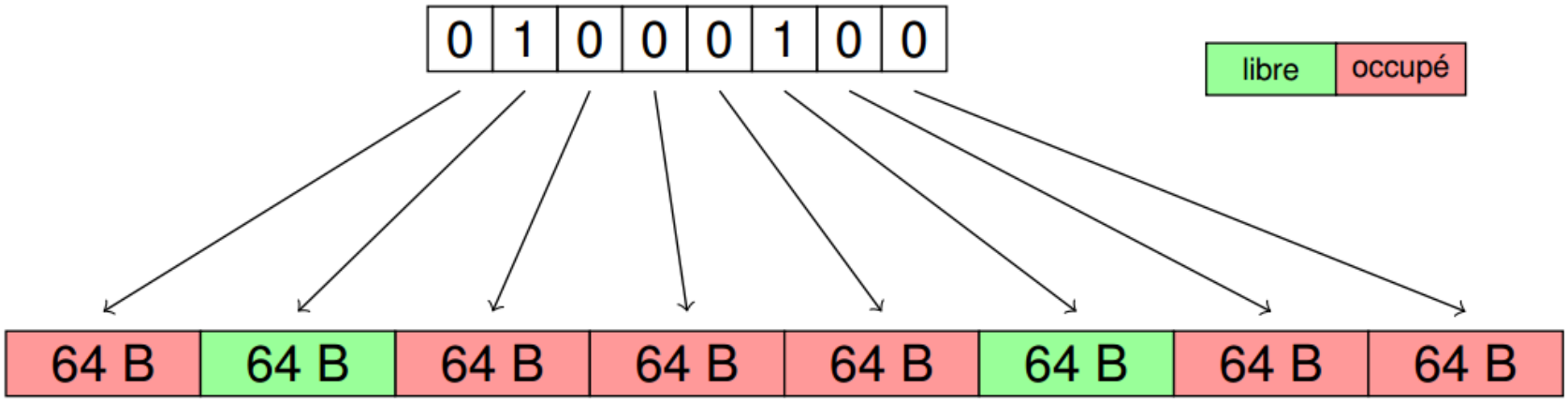
Memory Management: dynamic approaches

How to Free Memory ?

- aggregate together contiguous free blocks (can necessitate to sort the list, low determinism)
- no aggregation: more pointers, sort can be necessary
- Never free memory

Memory Management: dynamic approaches

Alternative: the bitmap



- fixed size block
- one bit for each block: 1 means its free, 0 means its occupied
- can be extended with deferent memory area with deferent block size

drawback: size of the bitmap

Memory Management: Static allocation

No dynamic allocation means:

- each block position is known during linking, access time reduced
- the available memory quantity available can be check during the linking
- no possibility to run out of memory or to encounter fragmentation issues at runtime

This is the best solution for embedded systems when possible!

Memory Management Unit

MMU

This permits:

- memory area protection
- fragmentation reduction (pages, matching logic and physical addresses...)
- ...

But MMU is not proposed by all EOS (simplicity or performences)

Architecture and parallelism

Several kind of ES:

- Mono core
- SMP (Symetric Multiprocessors) or multi core
- NUMA (non uniform memory architecture)

Even for mono core, concurrency is most of the time needed

Main loop

This is the most simple scheme:

```
while(true){
        task_1();
        task_2();
        ...
        task_n();
}
```

- Resource wast (intensive use of polling)
- no priority gesture, all task has the same frequency (of release)

Polling on event

```
while(true){
        switch(wait_event()){
                case sensor_1:
                        handle_sensor_1();
                        break;
                case sensor_2:
                        handle_sensor_2();
                        break;
                ...
                case sensor_n:
                        handle_sensor_n();
                        break;
        }
}
```

- no priority between event
- a long handler will block the main looop

Interruption

- one main task continuously executing (main loop)
- sub tasks executed on interrupt

With timer interruptions, this can emulate multitasking on mono task systems

- only one main task...

Notions to know

- interruption
- context switch
- synchronization
- preemptive system
- critical sections
- lock
- semaphore
- deadlock
- priority inversion
- watchdog
- ...

Break time ! And then: What is a Micro controller ?

Computational hardware

- Digital logic
  - Gates and flip-flops: glue logic, simple FSMs, registers
  - Two-level PLDs: FSMs, muxes, decoders
- Programmable logic devices
  - Field-programmable gate arrays: FSMs, basic data-paths
  - Mapping algorithms to hardware
- Microprocessors
  - General-purpose computer
  - Instructions can implement complex control structures
  - Supports computations/manipulations of data in memory

Microprocessors

- Arbitrary computations
  - Arbitrary control structures
  - Arbitrary data structures
  - Specify function at high-level and use compilers and debuggers
- Microprocessors can lower hardware costs
  - If function requires too much logic when implemented with gates/FFs
    - Operations are too complex, better broken down as instructions
    - Lots of data manipulation (memory)
  - If function does not require higher performance of customized logic
    - Ever-increasing performance of processors puts more and more applications in this category
    - Minimize the amount of external logic

Microprocessor basics

- Composed of three parts
  - Data-path: data manipulation and storage
  - Control: determines sequence of actions executed in data-path and interactions to be had with environment
  - Interface: signals seen by the environment of the processor
- Instruction execution engine: fetch/execute cycle
  - Flow of control determined by modifications to program counter
  - Instruction classes:
    - Data: move, arithmetic and logical operations
    - Control: branch, loop, subroutine call
    - Interface: load, store from external memory

Microprocessor basics

- Can implement arbitrary state machine with auxiliary data-path
  - Control instructions implement state diagram
  - Registers and ALUs act as data storage and manipulation
  - Interaction with the environment through memory interface
  - How are individual signal wires sensed and controlled?

Microprocessor organization

- Controller
  - Inputs: from ALU (conditions), instruction read from memory
  - Outputs: select inputs for registers, ALU operations, read/write to memory
- Data-path
  - Register file to hold data
  - Arithmetic logic unit to manipulate data
  - Program counter (to implement relative jumps and increments)
- Interface
  - Data to/from memory (address and data registers in data path)
  - Read/write signals to memory (from control)

+ dessin

General-purpose processor

- Programmed by user
- New applications are developed routinely
- General-purpose
  - Must handle a wide ranging variety of applications
- Interacts with environment through memory
  - All devices communicate through memory data
  - DMA operations between disk and I/O devices
  - Dual-ported memory (e.g., display screen)
  - Generally, oblivious to passage of time

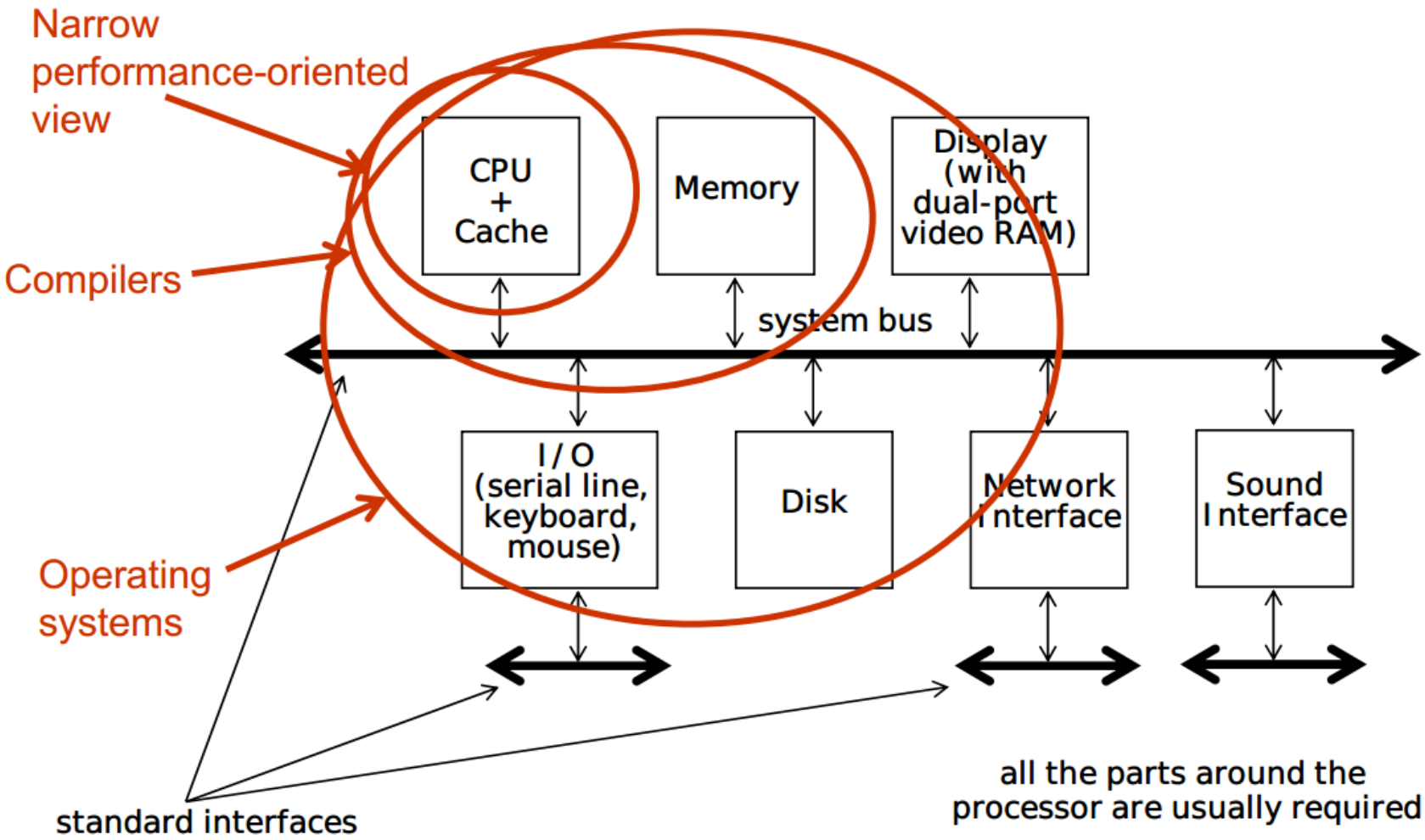
Embedded processor

- Typically programmed once by manufacturer of system
  - Rarely does the user load new software
- Executes a single program (or a limited suite) with few parameters
- Task-specific
  - Can be optimized for a specific application
- Interacts with environment in many ways
  - Direct sensing and control of signal wires
  - Communication protocols to environment and other devices
  - Real-time interactions and constraints
  - Power-saving modes of operation to conserve battery power

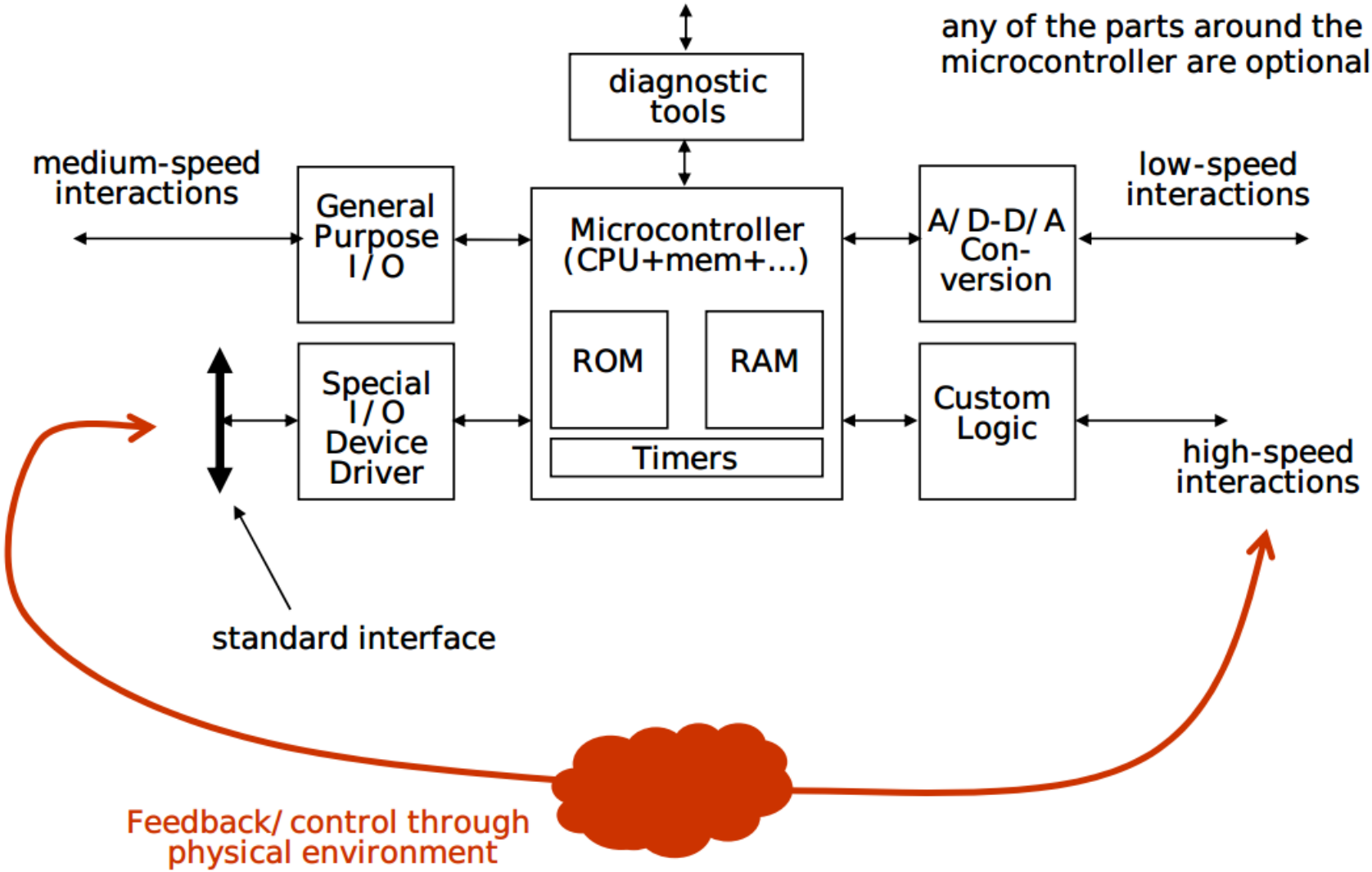
Why embedded processors?

- High overhead in building a general-purpose system
  - Storing/loading programs
  - Operating system manages running of programs and access to data
  - Shared system resources (e.g., system bus, large memory)
  - Many parts
    - Communication through shared memory/bus
    - Each I/O device often requires its own separate hardware unit
- Optimization opportunities
  - As much hardware as necessary for application
    - Cheaper, portable, lower-power systems
  - As much software as necessary for application
    - Doesn't require a complete OS, get a lot done with a smaller processor
  - Can integrate processor, memory, and I/O devices on to a single chip

Typical general-purpose architecture



Typical task-specific architecture



How does this change things?

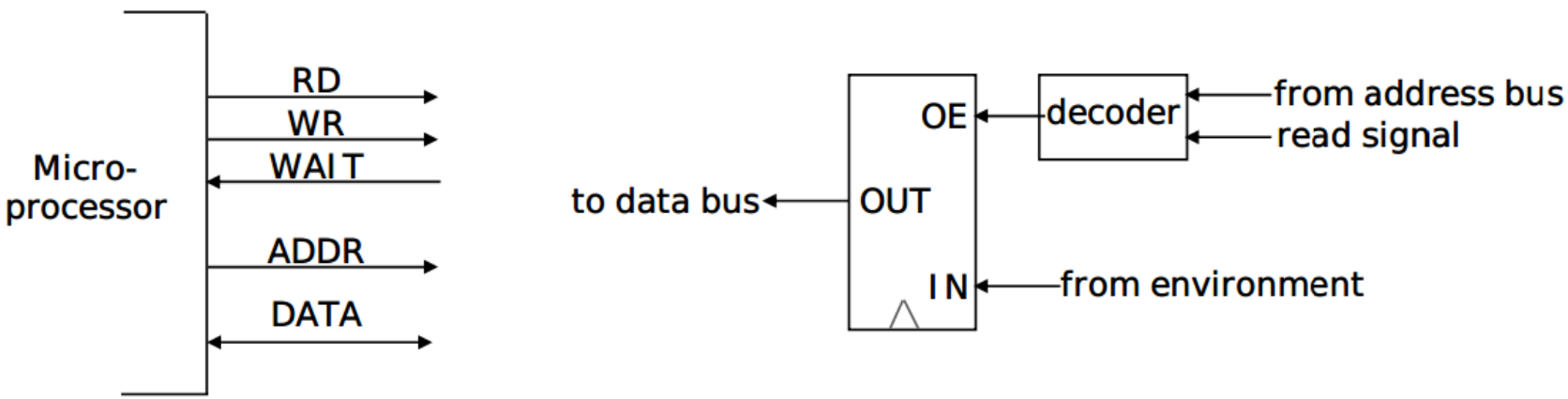
- Sense and control of environment
  - Processor must be able to "read" and "write" individual wires
  - Controls I/O interfaces directly
- Measurement of time
  - Many applications require precise spacing of events in time
  - Real-time is not the same thing as fast as possible
  - Reaction times to external stimuli may be constrained
- Communication
  - Protocols must be implemented by processor
  - Integrate I/O device or emulate in software
  - Capability of using external device when necessary

Interactions with the environment

- Basic processor only has address and data busses to memory
- Inputs are read from memory
- Outputs are written to memory
- Thus, for a processor to sense/control signal wires in the environment they must be made to appear as memory bits
  - How do we make wires look like memory?
- How long does it take to do these things?

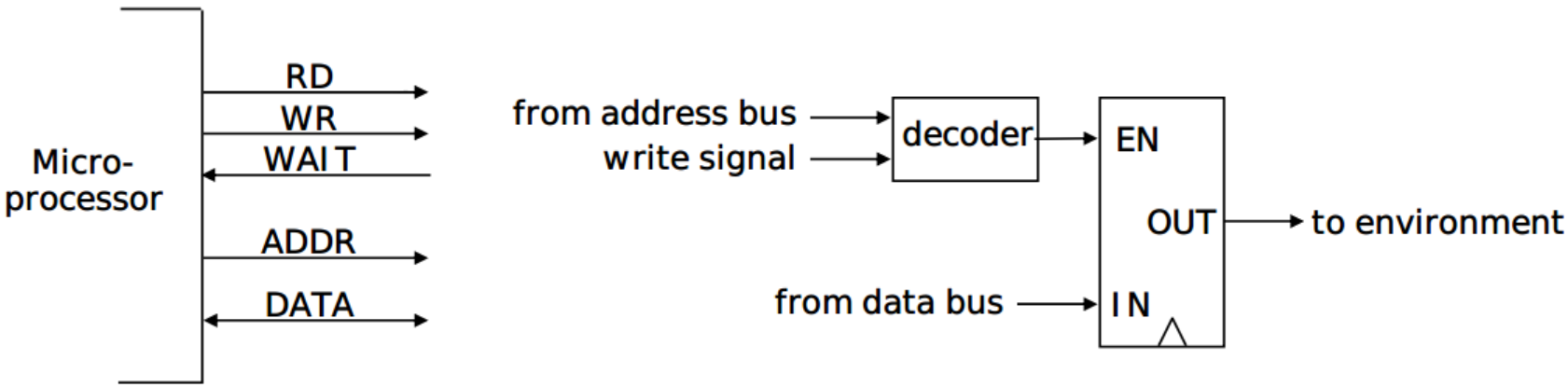
Sensing external signals

- Map external wire to a bit in the address space of the processor
- External register or latch buffers values coming from environment
  - Map register into address space
    - Decoder selects register for reading
  - Output enable (OE) to get value on to data bus
    - Lets many registers use the same data bus



Controlling external signals

- Map external wire to a bit in the address space of the processor
- Connect output of memory-mapped register to environment
  - Map register into address space
    - Decoder selects register for writing (holds value indefinitely)
  - Input enable (EN) to take value from data bus
    - Lets many registers use the same data bus



Time and instruction execution

- Keep track of detailed timing of each instruction's execution
  - Highly dependent on code
  - Hard to use compilers
  - Not enough control over code generation
  - Interactions with caches/instruction-buffers
- Loops to implement delays
  - Keep track of time in counters
  - Keeps processor busy counting and not doing other useful things
- Timer
  - Take differences between measurements at different points in code
  - Keeps running even if processor is idle to save power
  - An independent "co-processor" to main processor

Time measurement via parallel timers

- Separate and parallel counting unit(s)
  - Co-processor to microprocessor
  - Does not require microprocessor intervention
  - May be a simple counter or a more featured real-time clock
  - Alarms can be set to generate interrupts
- More interesting timer units
  - Self reloading timers for regular interrupts
  - Pre-scaling for measuring larger times
  - Started by external events

Input/output events

- Input capture
  - Record time when input event occurred
  - Can be used in later handling of event
- Output compare
  - Set output event to happen at a point in the future
  - Reactive outputs
    - e.g., set output to happen a pre-defined time after some input
  - Processor can go on to do other things in the meantime

System bus based communication

- Extend address/data bus outside of chip
- Use specialized devices to implement communication protocol
- Map devices and their registers to memory locations
- Read/write data to receive/send buffers in shared memory or device
- Poll registers for status of communication
- Wait for interrupt from device on interesting events
  - Send completed
  - Receive occurred

Support for communication protocols

- Built-in device drivers
  - For common communication protocols
    - e.g., RS232, IrDA, USB, Bluetooth, etc.
  - Serial-line protocols most common as they require fewer pins
- Serial-line controller
  - Special registers in memory space for interaction
  - May use timer unit(s) to generate timing events
    - For spacing of bits on signal wire
    - For sampling rate
- Increase level of integration
  - No external devices
  - May further eliminate need for shared memory or system bus

Micro controllers

- Embedded processor with much more integrated on same chip
  - Processor core + co-processors + memory
  - ROM for program memory, RAM for data memory, special registers to interface to outside world
  - Parallel I/O ports to sense and control wires
  - Timer units to measure time in various ways
  - Communication subsystems to permit direct links to other devices

Micro controllers

- Other features not usually found in general-purpose CPUs
  - Expanded interrupt handling capabilities
    - Multiple interrupts with priority and selective enable/disable
    - Automatic saving of context before handling interrupt
    - Interrupt vectoring to quickly jump to handlers
  - More instructions for bit manipulations
    - Support operations on bits (signal wires) rather than just words
- Integrated memory and support functions for cheaper system cost
  - Built-in EEPROM, Flash, and/or RAM
  - DRAM controller to handle refresh
  - Page-mode support for faster block transfers

Embedded Operating Systems Examples

- FreeRTOS
- TinyOS
- OSEK/VDX