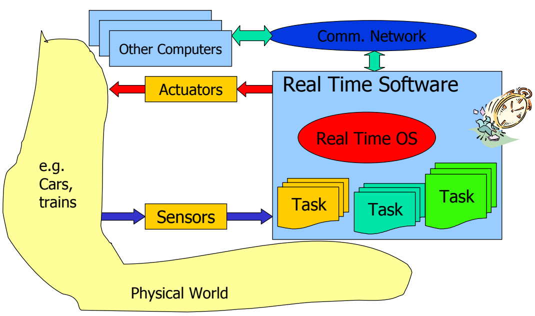
Course Outline

- Introduction
  - Characteristics of RTS
- Real Time Programming Language
  - Language support, e.g. Ada tasking
- Real Time Operating Systems (RTOS)
  - System support: scheduling, resource handling
- Design and Analysis of RT Application Software
  - Modeling and analysis
- Reliability and Fault-Tolerance
  - Fault tolerant, failure recovery, exception handling
- Distributed real time systems
  - Real Time Communication: TTCAN

Overall Structure of RT Systems

- Hardware (CPU, I/O device etc)
  - a clock!
- A real time OS (function as standard OS, with predictable behavior and well-defined functionality)
- A collection of RT tasks/processes (share resourses, communicate/synchronize with each other and the environment)

Components of RT Systems



Characteristics of a RTS

- Large and complex — vary from a few hundred lines of assembler or C to 20 million lines of Ada estimated for the Space Station Freedom
- Concurrent control of separate system components — devices operate in parallel in the real-world; better to model this parallelism by concurrent entities in the program
- Facilities to interact with special purpose hardware — need to be able to program devices in a reliable and abstract way
- Mixture of Hardware/Software: some modules implemented in hardware, even whole systems, SoC

Characteristics of a RTS (ctn.)

- Extreme reliability and safety — embedded systems typically control the environment in which they operate; failure to control can result in loss of life, damage to environment or economic loss
- Guaranteed response times — we need to be able to predict with confidence the worst case response times for systems; efficiency is important but predictability is essential

Terminology

- Continuous interaction with the environment:
  - Reactive Systems
- Must react to the environment in time:
  - Time-sensitive systems
- Embedded in electronic and/or mechanical devices, complex systems:
  - Embedded systems
- A failaure may cause the loss of lifes ...:
  - Safety-critical systems/fault-tolerant systems

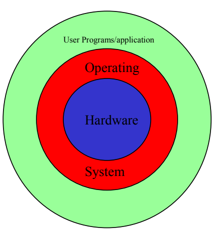
Terminology (ctn.)

- It often deals with continuous variables e.g. temperature, speed, etc (hybrid systems, dynamics systems)
- RT system may consist of many processes running on
  - single processor (concurrent/multi-task systems)
  - tightly-coupled processors (parallel systems), multicores, MPSoC
  - loosely-coupled processors connected by a network (distributed systems)

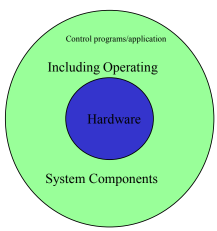
Real-time Programming Languages

- Assembly languages
- Sequential programming languages — e.g. Pascal, C.
  - Both normally require operating system support.
- High-level concurrent languages e.g. Concurrent Pascal, Ada, Modula-2, Java.
  - No/less operating system support!
- We will consider:
  - Ada 95 and C

Real-Time Languages and OS's



General-purpose computer systems



Typical Embedded Configuration

Classification of RTS's

- Hard real-time — systems where it is absolutely imperative that responses occur within the required deadline. E.g. Flight control systems.
- Soft real-time — systems where deadlines are important but which will still function correctly if deadlines are occasionally missed. E.g. Data acquisition system.
  - Firm real-time — systems which are soft real-time but in which there is no benefit from late delivery of service.
- Real real-time — systems which are hard real-time and in which the response times are very short. E.g. Missile guidance system.

A single system may have all hard, soft and real real-time subsystems
In reality many systems will have a cost function associated with missing each deadline.

Example: a Car Controller

Activities of a car control system. Let

1. C= worst case execution time
2. T= (sampling) period
3. D= deadline

- Speed measurment: C=4ms, T=20ms, D=5ms
- ABS control: C=10ms,T=40ms, D=40ms
- Fuel injection: C=40ms, T=80ms,D=80ms
- Other software with soft deadlines e.g audio, air condition etc

Construct a controller meeting all the deadlines!

Programming the car controller (1)

Process Speed:	Process ABS
Loop read sensor,compute,display... sleep (0.02) /*period*/ End loop	Loop Read sensor, compute, react sleep(0.04) End loop
Process Fuel Loop read data, compute, inject ... sleep(0.08) End loop	Soft RT Processes Loop read temperature el hiss, stereo .... End loop

Any problem?

- We forgot the execution times !

e.g. Process speed:

20ms = execution time + sleep(X)

Programming the car controller (2)

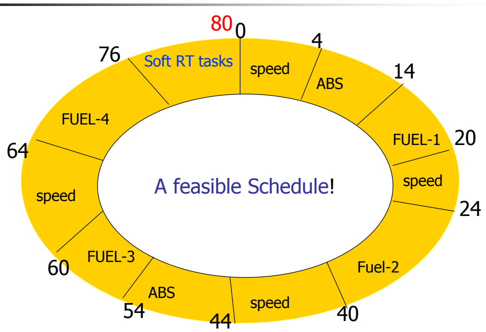
Process Speed:	Process ABS
Loop next := get-time + 0.02 read sensor,compute,display... sleep until next End loop	Loop next:=get-time + 0.04 Read sensor, compute, react sleep until next End loop
Process Fuel Loop next:=get-time + 0.08 read data, compute, inject ... sleep until next End loop	Soft RT Processes Loop read temperature elevator, stereo .... End loop

What is the problem now?

We don't know if the deadlines are met!

- We need to know the execution times
- We need to do schedulability analysis
- We need to construct a schedule
- We need to implement/buy an RT operating system
  - Run-time system (in programming language design)

Programming the car controller (3)



Main desirable properties of RT Systems(1)

- Timeliness: not only outputs but also times they are produced
- Predictability: able to predict the future consequences of current actions
- Testability: easy to test if the system can meet all the deadlines
- Cost optimality: e.g. Energy consumption, memory blocks etc

Main desirable properties of RT Systems (2)

- Maintainability: modular structure to ease system modification
- Robustness: must not collapse when subject to peak load, exception, manage all possible scenarios
- Fault tolerance: hardware and software failures should not cause the system to crash - function down-grading

Predictability: the most important one

- The system behaviour is known before it is put into operation!

  e.g. Response times, deadlock freedom etc

Difficult (impossible?) to achieve!

This is not so easy, why?
RT OS:

- System calls: difficult to know the worst execution times (theoretically impossible, halting problem)
- Cache (hit ratio, never exact), pipelines ...
- DMA stealing CPU memory cycle (when CPU running a hard task)
- Interrupt handling may introduce unbounded delays
- Priority inversion (low-prority tasks blocking high-prior taskts)
- Memory management (static allocation may not be enough, dynamic data structures e.g. Queue), no virtual memory
- Communication delays in a distributed environment

This is not so easy, why?
RT Tasks:

- Difficult to calculate the worst case execution time for tasks (theoretically impossible, halting problem)
  - Avoid dynamic data structures
  - Avoid recursion
  - Bounded loops e.g. For-loops only
- Complex synchronization patterns between tasks: potential deadlocks (formal verification)

Problems to solve ...

- Missing deadlines (!)
- Deadlocks/livelocks
- Uncontrolled exception (ARIAN 5)
- Clock jitter (the golf war, Scud missile)
  - 57micro sec/min, 343ms/100 hours
  - 687 meters
- Priority inversion (the Mars project)
- Uncontrolled code size, cost, ...
- Wrong timeout periods
- Non-determinism and/or Race condition
- Overloaded