Course Outline

- Introduction
  - Characteristics of RTS
- Real Time Operating Systems (RTOS)
  - OS support: scheduling, resource handling
- Real Time Programming Languages
  - Language support, e.g. Ada tasking
- Scheduling and Timing Analysis of RT Software
  - Worst-case execution and response time analysis
- Design and Validation
  - Modeling, Verification and Testing
- Reliability and Fault-Tolerance
  - Fault tolerant, failure recovery, exception handling
- Distributed real time systems
  - Real Time Communication: CAN Bus

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 resources, communicate/synchronize with each other and the environment)
Components of RT Systems

- Actuators
- Sensors
- Physical World (e.g., cars, trains)
- Other Computers
- Communication Network
- Real Time Software (RTOS)
- Tasks

General-Purpose vs. Embedded RT Computer Systems

- General-purpose computer systems
- Typical Embedded Configuration
What Do RT Embedded Systems Do?

- 5 main kinds of functionality:
  - Digital signal processing
  - Open loop and closed loop control
  - Wired and wireless networking
  - User interfacing
  - Storage management

- Most embedded systems do 1-4 of these

Example: Fly-by-wire Avionics:
Hard real-time system with multi-rate tasks

<table>
<thead>
<tr>
<th>Sensors</th>
<th>Signal Conditioning</th>
<th>Control laws</th>
<th>Actuating</th>
<th>Actuators</th>
</tr>
</thead>
<tbody>
<tr>
<td>gyros, accel.</td>
<td>INU 1kHz</td>
<td>Pitch control 500 Hz</td>
<td>Aileron 1</td>
<td>Aileron</td>
</tr>
<tr>
<td>GPS</td>
<td>GPS 20 Hz</td>
<td>Lateral Control 250 Hz</td>
<td>Aileron 2</td>
<td>Aileron</td>
</tr>
<tr>
<td>Air Sensor</td>
<td>Air data 1 kHz</td>
<td></td>
<td>Elevator 1</td>
<td>Elevator</td>
</tr>
<tr>
<td>Stick</td>
<td>Joystick 500 Hz</td>
<td></td>
<td>Rudder 1</td>
<td>Rudder</td>
</tr>
</tbody>
</table>

Sensors: gyros, accel., GPS, Air Sensor, Stick
Signal Conditioning: INU 1kHz, GPS 20 Hz, Air data 1 kHz, Joystick 500 Hz
Control laws: Pitch control 500 Hz, Lateral Control 250 Hz, Throttle Control 250 Hz
Actuating: Aileron 1 1 kHz, Aileron 2 1 kHz, Elevator 1 kHz
Actuators: Aileron, Aileron, Elevator, Rudder
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 measurement: 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)

<table>
<thead>
<tr>
<th>Process Speed:</th>
<th>Process ABS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop</td>
<td>Loop</td>
</tr>
<tr>
<td>read sensor, compute, display...</td>
<td>Read sensor, compute, react</td>
</tr>
<tr>
<td>sleep (0.02) /<em>period</em>/</td>
<td>sleep(0.04)</td>
</tr>
<tr>
<td>End loop</td>
<td>End loop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process Fuel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop</td>
</tr>
<tr>
<td>read data, compute, inject ...</td>
</tr>
<tr>
<td>sleep(0.08)</td>
</tr>
<tr>
<td>End loop</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Any problem?

- We forgot the execution times...

  e.g. Process speed:

  $20\text{ms} = \text{execution time} + \text{sleep}(X)$

---

Programming the car controller (2)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop</td>
<td>Loop</td>
<td>Loop</td>
<td>Loop</td>
</tr>
<tr>
<td>$\text{next} := \text{get-time} + 0.02$</td>
<td>$\text{next} := \text{get-time} + 0.04$</td>
<td>$\text{next} := \text{get-time} + 0.08$</td>
<td>$\text{next} := \text{get-time} + 0.08$</td>
</tr>
<tr>
<td>$\text{read sensor, compute, display...}$</td>
<td>$\text{Read sensor, compute, react}$</td>
<td>$\text{read data, compute, inject...}$</td>
<td>$\text{read temperature}$</td>
</tr>
<tr>
<td>$\text{sleep until next}$</td>
<td>$\text{sleep until next}$</td>
<td>$\text{sleep until next}$</td>
<td>elevator, stereo</td>
</tr>
<tr>
<td>End loop</td>
<td>End loop</td>
<td>End loop</td>
<td>$\text{...}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>End loop</td>
</tr>
</tbody>
</table>
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 OS kernel

Programming the car controller (3)

A feasible Schedule!
Challenges in RT Systems Design

- **Predictability**: the system behaviour is known before it is put into operation!
  
  e.g. Response times, deadlock freedom etc
- **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!

Difficult (impossible?) to achieve!
e.g. How to estimate the worst case response time of a task?

Difficult to achieve predictability: 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)
- Multi-tasking, tasks that share resources
Difficult to achieve predictability: Hardware & RTOS

- Cache sharing, processor pipelines, DMA ...
- Interrupt handling may introduce unbounded delays
- Priority inversion (low-priority tasks blocking high-priority tasks)
- Memory management (static allocation may not be enough, dynamic data structures e.g. Queue), no virtual memory
- Communication delays in a distributed environment

Problems to solve ...

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