02224
Modelling and Analysis of Real-Time Systems

Spring 2023

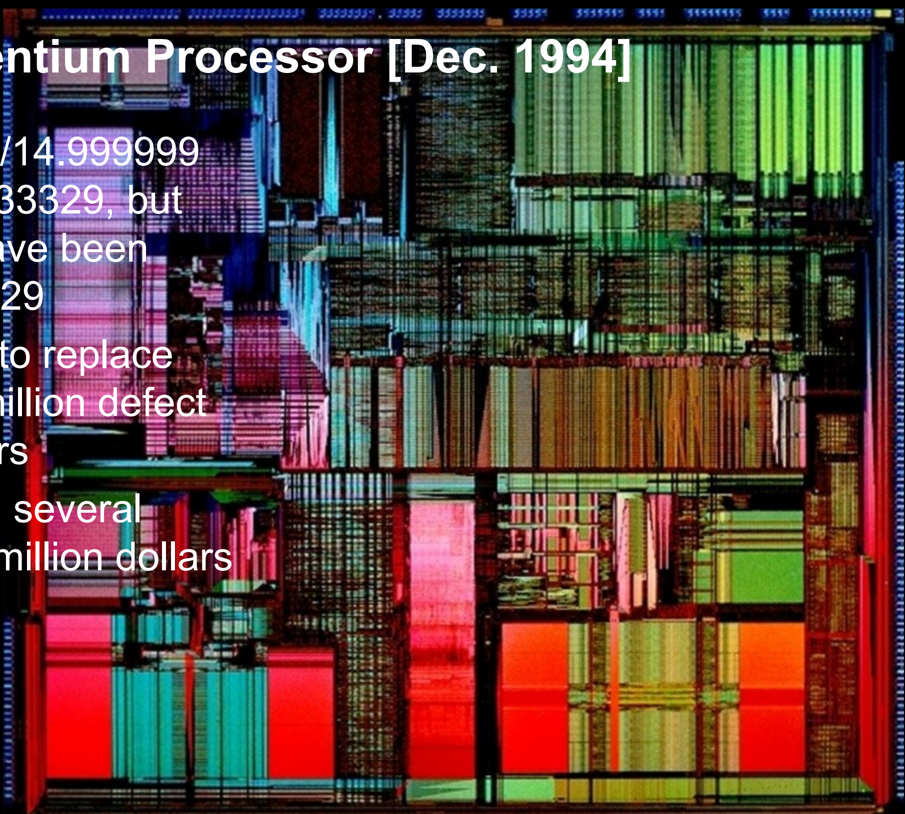
DTU Compute
Technical University of Denmark

Lecturers:
Hans Henrik Løvengreen (hhlo@dtu.dk)
Michael R. Hansen (mire@dtu.dk)
Martin Schöberl (masca@dtu.dk)

Contents

- Introduction to Real-Time Systems
  - Examples – why is this important?
  - Some definitions – what exactly are we dealing with?
  - Development techniques – how should we design and validate a real-time system?
- Introduction to Uppaal
  - Tool for modelling and verifying correctness of real-time systems
- Today's Exercises
  - Simple Light Control
  - Goat/Wolf/Cabbage Puzzle

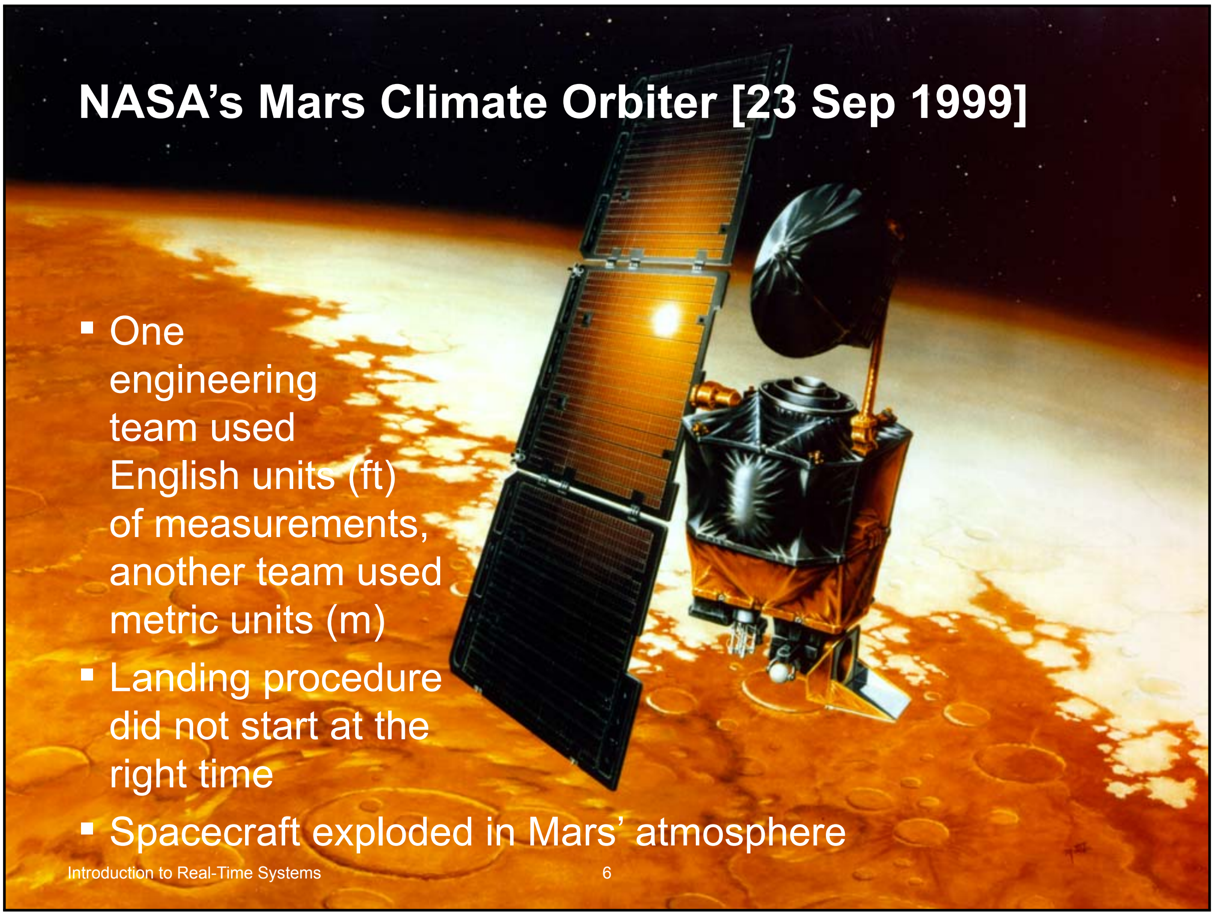
Intel’s Pentium Processor [Dec. 1994]



- 4.999999/14.999999 gives 0.333329, but should have been 0.333**33**29
- Intel had to replace several million defect processors
- Cost Intel several hundred million dollars

Ariane 5 Rocket [4 June 1996]

- Floating-point conversion failed and the control system crashed
- The rocket self-destructed 40 sec. after take-off
- The problem: buffer overflow in the control software taken over from Ariane 4
- The software had not been re-tested (to save money)
- Cost ESA $600M



NASA’s Mars Climate Orbiter [23 Sep 1999]

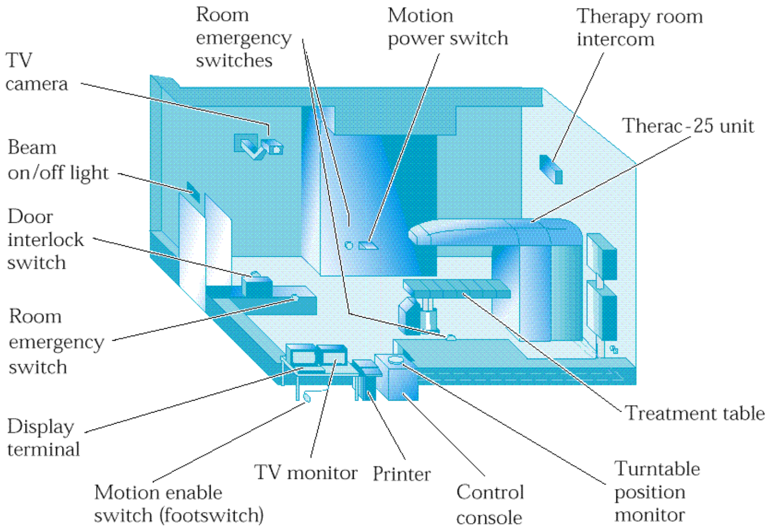
- One engineering team used English units (ft) of measurements, another team used metric units (m)
- Landing procedure did not start at the right time
- Spacecraft exploded in Mars’ atmosphere

NASA’s Mars Polar Lander [3 Dec 1999]

- The landing software mistakenly identified vibrations from the deployment of the lander's legs as being caused by the vehicle touching down on the Martian surface
- The vehicle's descent engines was cut off while it was still 40 meters above the surface, rather than on touchdown as planned
- Cost NASA several hundred million $

Therac-25 Radiation Therapy Machine [1985]

- Patients received massive X-ray overdoses
- 2 patients died
- Many others were seriously injured and handicapped



More Costly Software Errors

- The **Northeast blackout** (2003) was triggered by a local outage that went undetected due to a race condition in General Electric Energy's XA/21 monitoring software
- A fault in **Knight Capital Group’s trading system** (2012) caused it to flood the market with erratic trades resulting in a loss of over $440M in half an hour
- The Israeli **Beresheet moon lander** (2019) crashed because an engine reset command did not work in time
- Abrupt **Lime Scooter wheel blocks** (2019) causing injuries led to ban in Switzerland

What is a Real-Time System?

- **A computing system that must react within precise timing constraints to events in the environment.** As a consequence, the correct behaviour of these systems depends not only on the value of the computation but also on the time at which the results are produced.

  [Stankovic and Ramamritham 88]

- **A system in which the time at which output is produced is significant.** This is usually because the input corresponds to some movement in the physical world, and the output has to react to that same movement. The lag from input time to output time must be sufficiently small for acceptable timelines.

  [Oxford Dictionary of Computing]

- **An interactional system that maintains on-going relationship with an asynchronous environment**, i.e., an environment that progresses in an uncooperative manner.

  [Koymans, Kuiper, Zijlstra,1988]

Classification of Real-Time Systems

- **Hard Real-Time System:** Those systems where it is absolutely imperative that responses occur within given deadlines. Otherwise, the system is useless, dangerous or causes considerable costs (life, environmental, material damage, . . .)
  - Examples: anti-locking systems, railway crossings, air traffic controls, . . .
- **Soft Real-Time System:** Those systems where response times are important but the system will still function correctly if deadlines are occasionally missed
  - Examples: multimedia systems, data acquisition for a process control system, . . .
- **Notice**: faster does not necessarily mean better

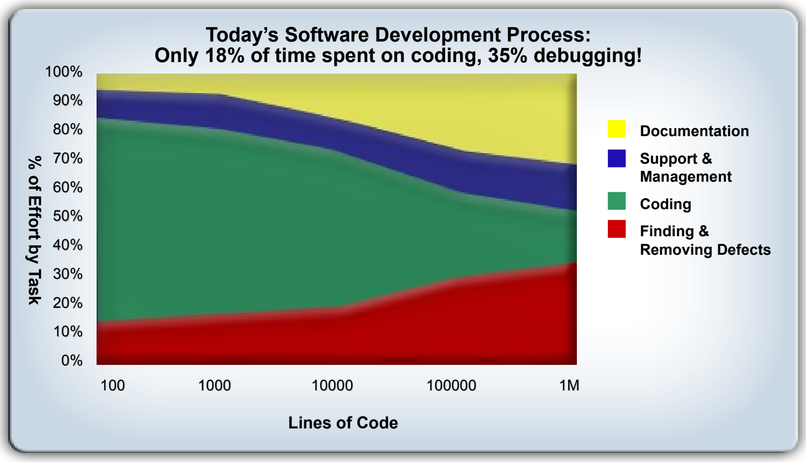
Examples of Real-Time Systems

- Train crossing systems
- Highway and air traffic control
- Robot control
- Production lines
- Mobile phones, cameras, CD players, TV-sets
- Multimedia applications
- Cars
- Controllers for washing machines, microwave ovens
- Medical applications
- General term: *Cyber Physical Sytems*

Real-Time Systems are Often Embedded

- An *embedded (computer) system* is a system where a computer is a component of a physical environment
- Some observations:
  - 99% (est.) of all microprocessors are used in embedded systems
  - Many embedded systems are **safety-critical**
    - correctness is a must for safety
  - Many embedded systems are **hardly maintainable**
    - correctness is a must for cost
- **How should we develop real-time software to ensure correctness?**

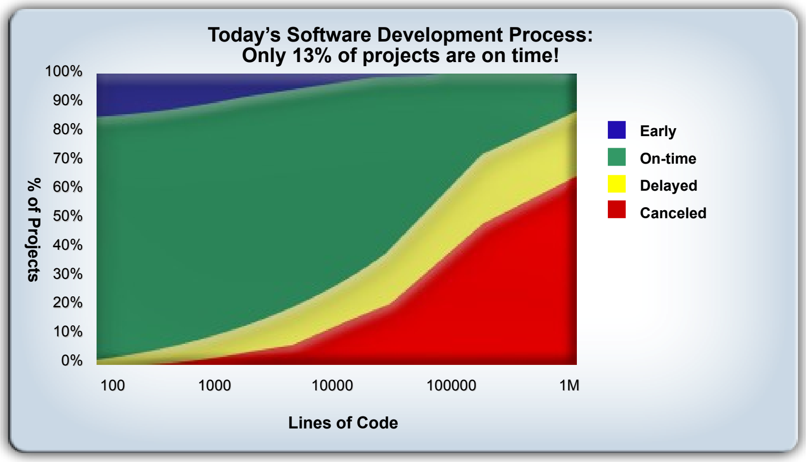
The Traditional Software Development Process



Source: Capers Jones, Estimating Software Costs, pg. 140

[If debugging is the process of removing bugs, then programming must be the process of putting them in!]

The Traditional Software Development Process



Source: Capers Jones, Patterns of Software Systems Failure & Success

Assuring Software Quality by Testing

- Testing is needed, but has limits:
  - Hard to find certain types of problems (e.g., concurrency)
  - Huge number of configurations is daunting
  - Testing reveals the presence of bugs, doesn't prove the absence
- Need more use of formal methods
  - Mathematical model of system: "specification"
  - Automated "verification" of software implementation
  - Used to be impractical due to state space size explosion and CPU speeds
  - Now routinely used in hardware design, where the cost of a bug is *much* larger
  - Increasingly used in large software companies, e.g., Microsoft

Formal Methods

- Systems are specified using a **language** with a precise (mathematically defined) semantics (meaning)
- Advantages:
  - models are **unambiguous**
  - models are **complete** on an abstract level
  - high quality tools may be defined on the basis of the semantics
- A scenario:
  - $Spec \Lleftarrow Design_1 \Lleftarrow Design_2 \cdots \Lleftarrow Implementation$
- D $\Lleftarrow$ I reads: I is an ***implementation*** *(or **refinement**)* of D
- Each refinement must be validated to ensure that the implementation satisfies the specification

Further Validation Techniques

- **Simulation**
  - Runs (random or selected) of the system are investigated on the basis of an executable model
  - Supported by a *simulator* tool
  - Undesirable behaviour may be detected on a model
- **Verification**
  - Proof (possibly machine assisted) of $\varphi_1 \Leftarrow \varphi_2$
  - At least larger proof steps must be chosen manually
- **Model checking** of Spec $\Leftarrow$ Design
  - It is machine-checked (by a *model checker*) whether every possible run of the Design is in agreement with the Spec
  - At least the state explosion problem sets a limit

In This Course

1. Models of real-time systems based on real-time automata
   - A parallel composition of automata models an embedded computer system as well as the environment
   - The tool Uppaal is used for modelling, simulation and model checking
2. Real-time scheduling, modelling and analysis
3. Other models for real-time systems (e.g. Reactors)

Course Evaluation

- Two mandatory assignments:
  1) Model and analyse a simple real-time system
  2) Extend the system from 1), implement it, test it on a simulation of the environment and deploy it on a physical system
     or
     Work with a scheduling problem
     or
     Work with an extension of the Uppaal Tool
     or
     Use the models/tools for another real-time problem
- A two-hours written exam (without computers)

Prerequisites for Appreciating the Course

- Knowledge of basic computer science models
- Knowledge of logic
- Basic knowledge of probability theory
- Interest in formal methods
- Knowledge of concurrency issues
  - Models of concurrency (eg. Petri-nets, interleaving model)
  - Concurrent programming (threads, monitors, …)
- Acquaintance with Java programming

Not In This Course

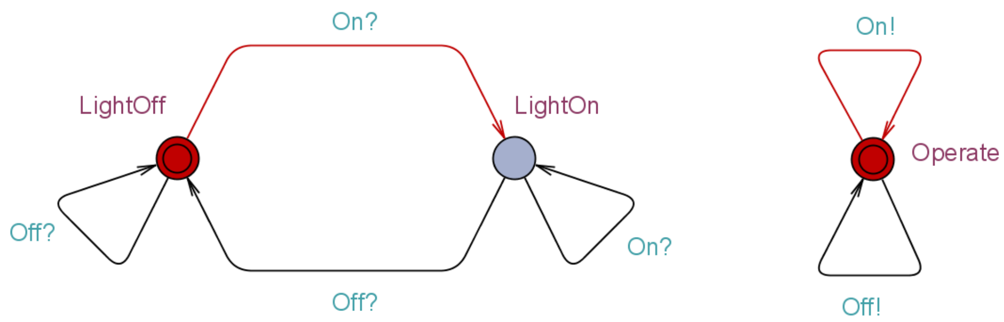
- Issues of embedding (hardware interfaces, drivers etc.)
- Control theory (continous state, PID)
- Multi-media aspects (and other soft applications)
- Advanced scheduling theory (only basics)
- Concurrency related issues (02158)
- Advanced model checking algorithms (02246)
- Automated reasoning (02256)
- General software engineering aspects
- Artificial Intelligence

Introduction to Uppaal Models

Untimed Models

Example: Simple Light Controller

- Modelling a reactive system:
  - Parallel composition || of automata modelling the design and the environment
  - *Initial states* are marked with double circuits
  - An *input event* **e?** may synchronize with an *output event* **e!**
- For a simple light control: *controller* || *person*



Reactive Finite Automata

- A *reactive finite automaton* is a tuple $(Q, \Sigma, \rightarrow, q_0)$, where
  - Q is a non-empty, finite set of *states (locations)*
  - Σ is a non-empty, finite set of *events*
  - $\rightarrow \subseteq Q \times (\Sigma \times \{!, ?\}) \times Q \ \cup \ Q \times Q$ is the *transition relation,* and
  - q_0 is the *initial state*
- Notice: A reactive finite automaton has, unlike a finite automaton, no accepting states
- Focus is on communication with the environment:

$$q_1 \xrightarrow{a!} q_2 \quad \text{or} \quad q_1 \xrightarrow{a?} q_2$$

- And on internal transitions:

$$q_1 \longrightarrow q_2$$

- An automaton performs a (possibly infinite) sequence of such transitions

Parallel Composition of Automata

- Run in parallel, two reactive finite automata:

$A_1 = (P, \Sigma, \rightarrow, p_0)$ and $A_2 = (Q, \Sigma, \rightarrow, q_0)$

may synchronize on matching input/output events or may individually perform internal transitions

- This is described by a *labelled transition system* $(S, \Sigma, \rightarrow, (p_0, q_0))$ where the states S are $P \times Q$ and the labelled transitions $\rightarrow$ are determined by the rules:

$$\frac{p_1 \xrightarrow{a!} p_2 \qquad q_1 \xrightarrow{a?} q_2}{(p_1,q_1) \xrightarrow{a} (p_2,q_2)} \qquad \frac{p_1 \xrightarrow{a?} p_2 \qquad q_1 \xrightarrow{a!} q_2}{(p_1,q_1) \xrightarrow{a} (p_2,q_2)}$$

$$\frac{p_1 \longrightarrow p_2}{(p_1,q_1) \longrightarrow (p_2,q_1)} \qquad \frac{q_1 \longrightarrow q_2}{(p_1,q_1) \longrightarrow (p_1,q_2)}$$

Network of Automata

- The parallel composition of two reactive automata generalizes to a *network* of n automata

$(Q_i, \Sigma, \rightarrow, q_{0i})$ for $1 \leq i \leq n$

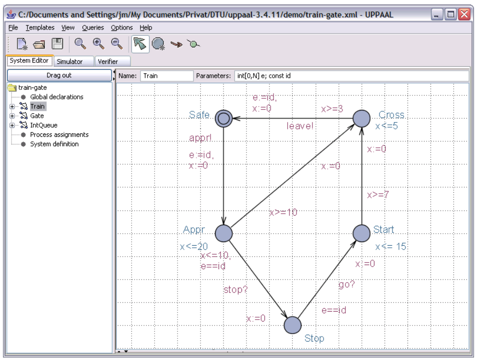
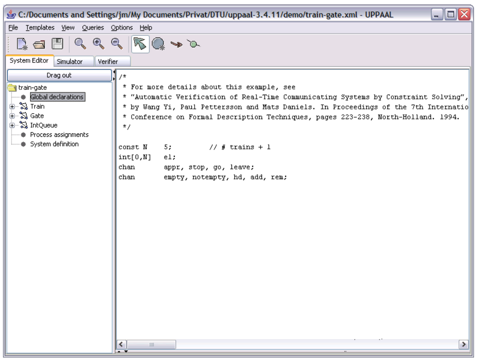
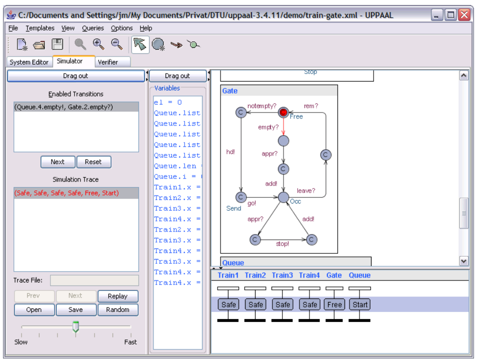
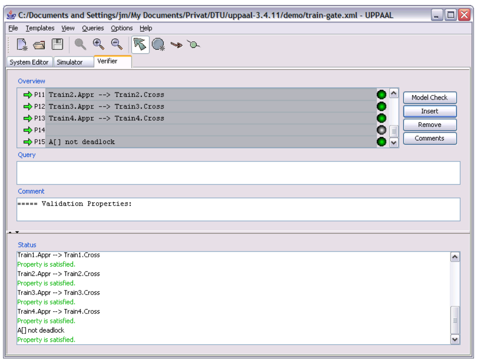
by allowing pair-wise communication only

- The network is *closed* in the sense that an input event may only occur if a corresponding output event is enabled, and vice versa
- Comments:
  - Let $k_i = |Q_i|$
  - Then the number of states of the network is $k_1 \cdot k_2 \cdots k_n$
  - This is know as the state explosion
  - When adding time, the timed automata will still have a finite number of states, but the labelled transition systems for a network of timed automata will have an infinite number of states

UPPAAL: Overview

- Model checking tool for real-time systems, developed at Uppsala University in Sweden and Aalborg University in Denmark, 1997
- A system is modelled by a network of timed automata
  - Declaration of channels (Σ), clocks, constants, finite data structures
  - Parameterized automata
  - Networks
- A graphical editor is used to model the network of automata
- A simulator is used to explore the transitions of the model
- A model checker is used to automatically verify certain classes of temporal formulas

UPPAAL: Overview

State variables

- The system state may be extended by:
  - *Local variables* associated with each automaton
  - System-wide *global variables*
- Variables may be *updated* by transitions
- Transitions are considered to be *atomic*
- Boolean expressions may *guard* transitions