Design and Analysis of Real-Time Systems

Jan Reineke

Advanced Lecture, Summer 2013

(Design and) Analysis of Real-Time Systems

Jan Reineke

Advanced Lecture, Summer 2013

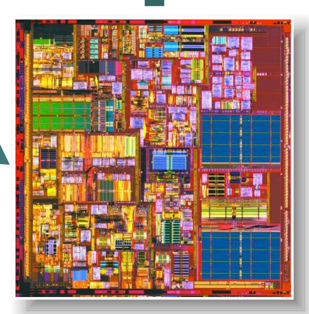
Organizational Issues

- Advanced Course (6 CPs)
  - Lectures every Thursday 14-16, E1.3, HS003
  - Tutorials: 2 hours every week; potential dates:
    - Monday 10-12, 14-16
    - Tuesday 10-12, 14-16
  - Written examination at the end of the term
    - Need to obtain > 50% of total points on exercises to participate
    - Grade determined by score on exam
  - Web: http://embedded.cs.uni-saarland.de/realtime.php

Structure of Course

2. How are they programmed?

```
// Perform the convolution.
for (int i=0; i<10; i++) {
  x[i] = a[i]*b[j-i];
  // Notify listeners.
  notify(x[i]);
}
```



1. What are Real-Time Systems?

3. How to verify the real-time constraints?

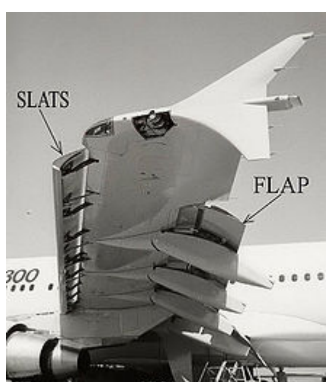
1. What are Real-Time Systems?

In a *real-time system*, correctness not only depends on the logical results but also on the *time* at which results are produced.

- Typical misconception:
  - Real-time computing ≠ compute things *as fast as possible*
  - Real-time computing = compute *as fast as necessary*, *but not too fast*

1. What are Real-Time Systems?

- Real-time systems are often embedded control systems
- Timing requirements often dictated by interaction with physical environment:
  - Examples in Automotives:
    - ABS: Anti-lock braking systems
    - ESP: Electronic stability control
    - Airbag controllers
  - Many more examples in trains, avionics, and robotics…



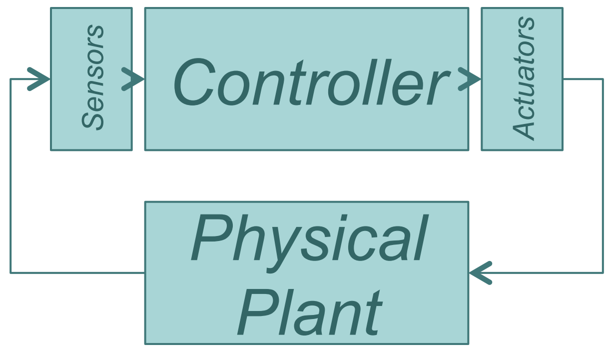
YouTube

Classification of Real-Time Constraints Hard and Soft Real-Time Systems

- “A real-time constraint is called hard, if not meeting that constraint could result in a catastrophe“ [Kopetz, 1997]
  - ➔ Safety-critical real-time systems
  - ➔ Main focus of this course
  - ➔ Can you think of examples?
- All other time-constraints are called soft.
  - ➔ Can you think of examples?
- A guaranteed system response has to be explained *without statistical arguments* [Kopetz, 1997].

2. How are they programmed?

Typical structure of control systems:



A very basic approach to program such a system:

```
initialize state;
every clock-tick do
    read inputs;
    compute outputs and next state;
    emit outputs
end-do
```

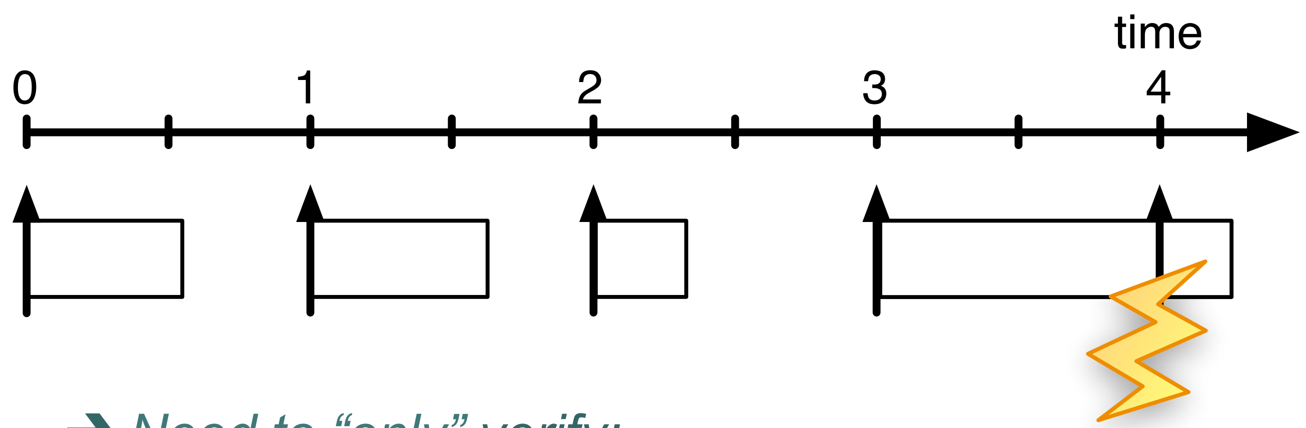
→

Can be modeled by automaton.

How to describe such automata? → Synchronous Languages

Basic Approach: Advantages

- Perfect match for sampled-data control theory
- Easy to implement, even on “bare” machine
- Timing analysis is comparably “simple”:



➔ Need to “only” verify: WCET < 1

Basic Approach: Limitations

- Distributed systems

  What if sensing, actuating, and computing happen at multiple locations?

- Event-triggered systems

  What if (some) computations are triggered by events rather than time?

- Multiperiodic systems

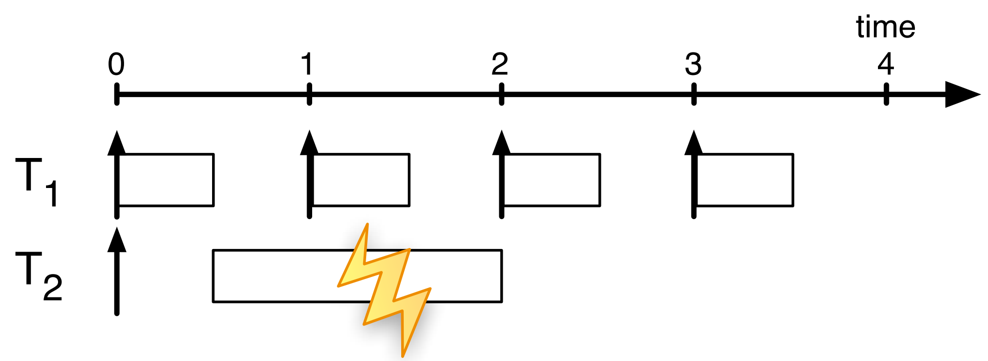
  What if different computations need to be performed at different periods?

2. How are they programmed? Scheduling Policies

Sophisticated scheduling policies have been introduced to overcome these limitations.

Example 1: Preemptive scheduling

Non-preemptive execution of two periodic tasks:

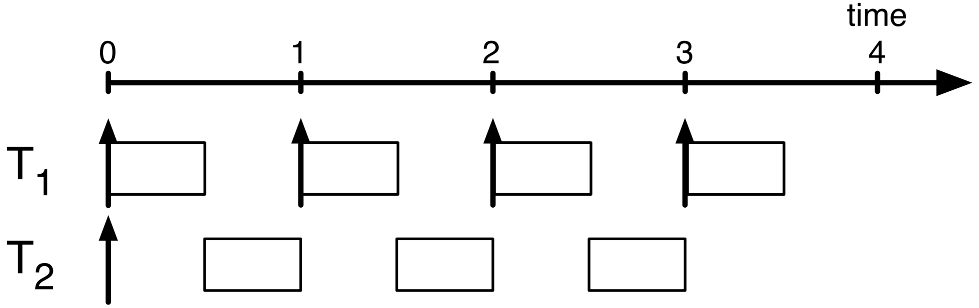


2. How are they programmed? Scheduling Policies

Sophisticated scheduling policies have been introduced to overcome these limitations.

Example 1: Preemptive scheduling

Preemptive execution of the two tasks:

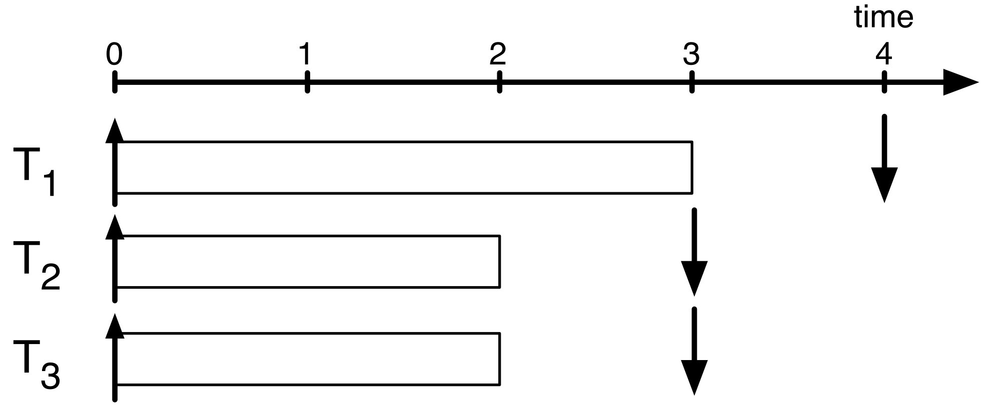


2. How are they programmed? Scheduling Policies

Sophisticated scheduling policies have been introduced to overcome these limitations.

Example 2: Multiprocessor scheduling

Is this task set schedulable on two processors?

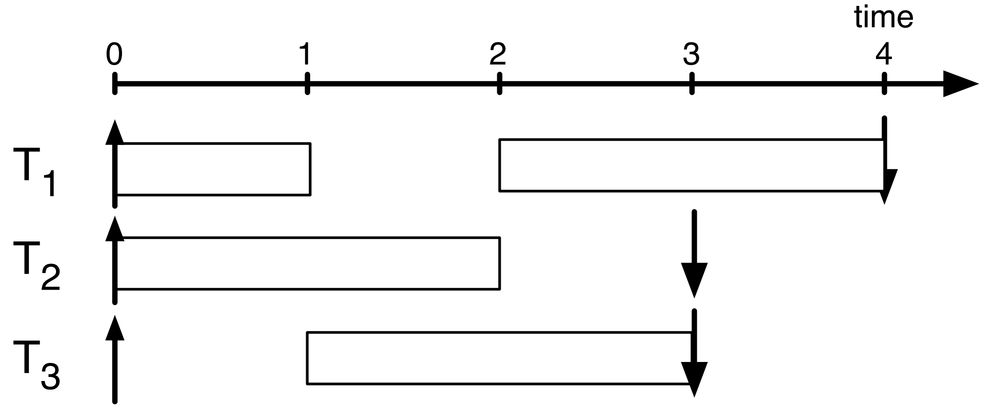


2. How are they programmed?
Scheduling Policies

Sophisticated scheduling policies have been introduced to overcome these limitations.

Example 2: Multiprocessor scheduling

It is!



3. How to verify the real-time constraints? Schedulability Analysis

Schedulability tests determine whether a given set of tasks is feasible under a particular scheduling policy.

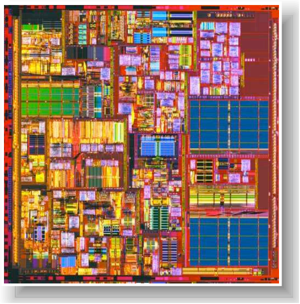
They all require bounds on the worst-case execution time (WCET) of all tasks.

3. How to verify the real-time constraints? Worst-case Execution Time Analysis

Worst-case execution time = maximum execution time of a program on a given microarchitecture

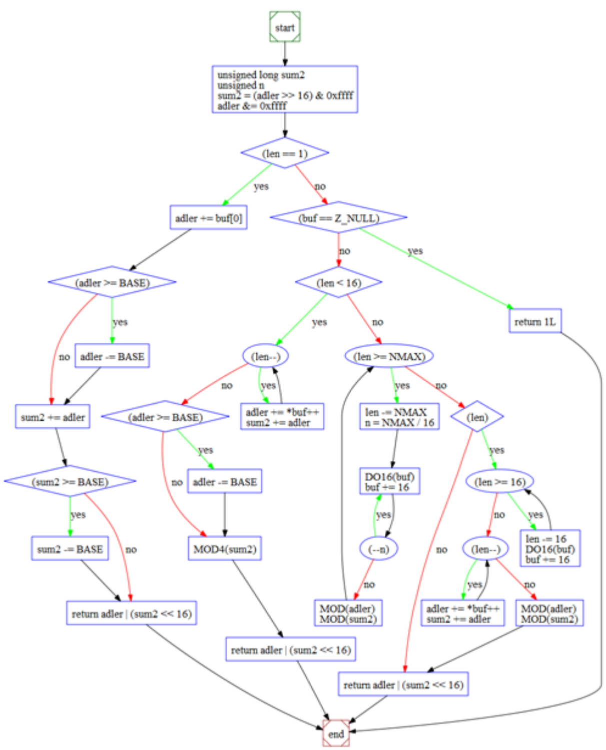
```
// Perform the convolution.
for (int i=0; i<10; i++) {
  x[i] = a[i]*b[j-i];
  // Notify listeners.
  notify(x[i]);
}
```

+



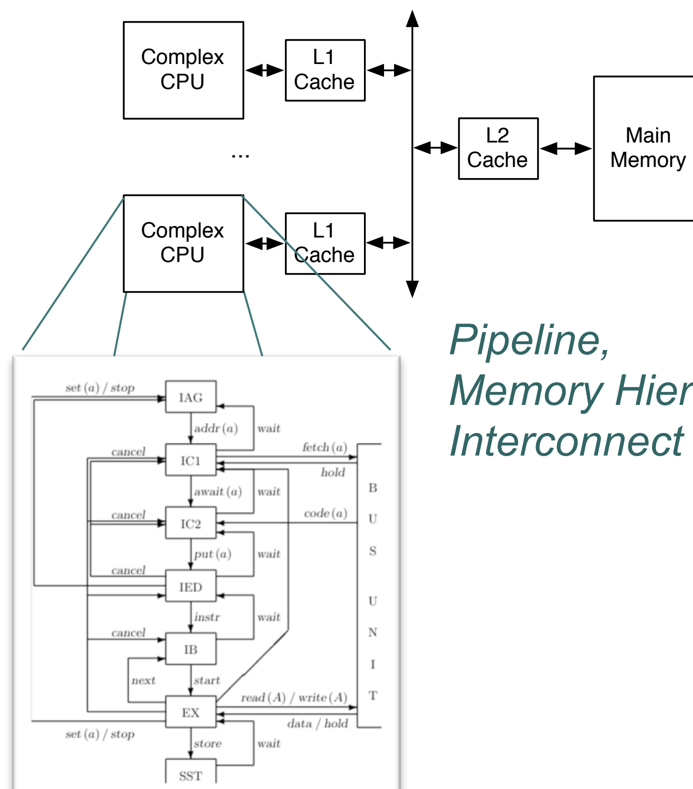
What does the execution time of a program depend on?

Input-dependent control flow



+

Microarchitectural State



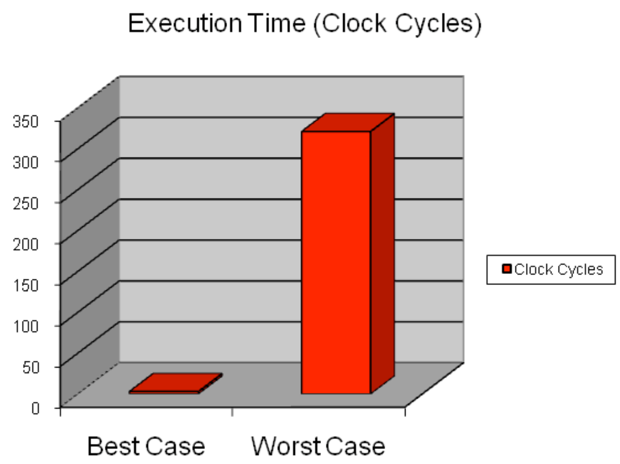
Pipeline, Memory Hierarchy, Interconnect

Example of Influence of Microarchitectural State

x=a+b; →

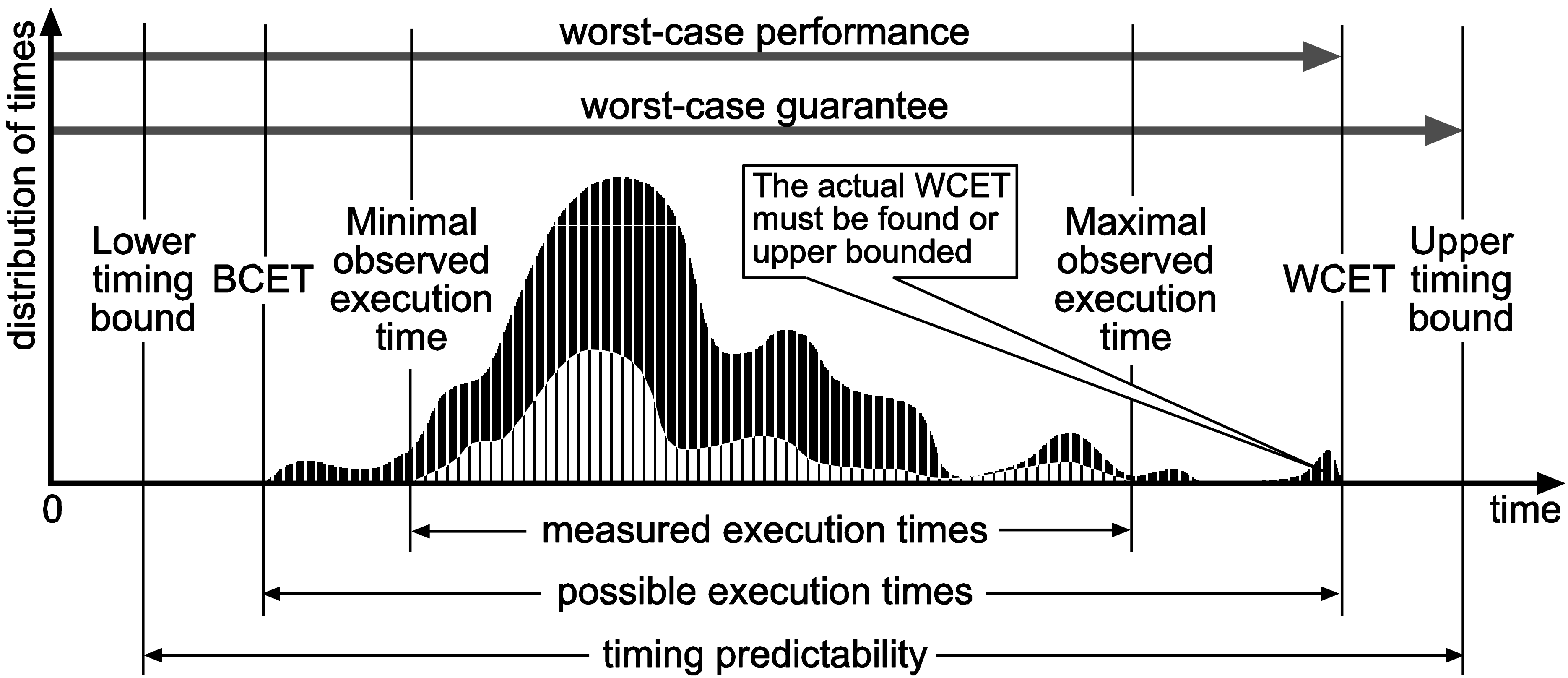
```
LOAD   r2, _a
LOAD   r1, _b
ADD    r3,r2,r1
```

Motorola PowerPC 755



Courtesy of Reinhard Wilhelm.

Notions in Worst-case Execution Time Analysis



Worst-case Execution Time Analysis
What is hard about it?

- Need to account for all possible paths through the program, but not many more for precision…
  - Even termination is in general undecidable.
- Need to account for all possible states of the microarchitecture that may arise.
  - We will see “unpredictable” components.
- Before performing WCET analysis, one needs to construct a faithful model of the microarchitecture; documentation is limited.

Overview of Topics

- Today:
  - High-level Overview of Challenges
- Rest of the course:
  - Worst-case Execution Time Analysis
    - Foundations of Abstract Interpretation
    - Value and Control-flow Analyses
    - Static Cache Analysis
    - Analysis of Preemption Cost
  - Predictable Microarchitectures
  - Scheduling Policies and Schedulability Analysis
  - Control Algorithms and Synchronous Languages
  - Clock Synchronization, Data-flow models, …