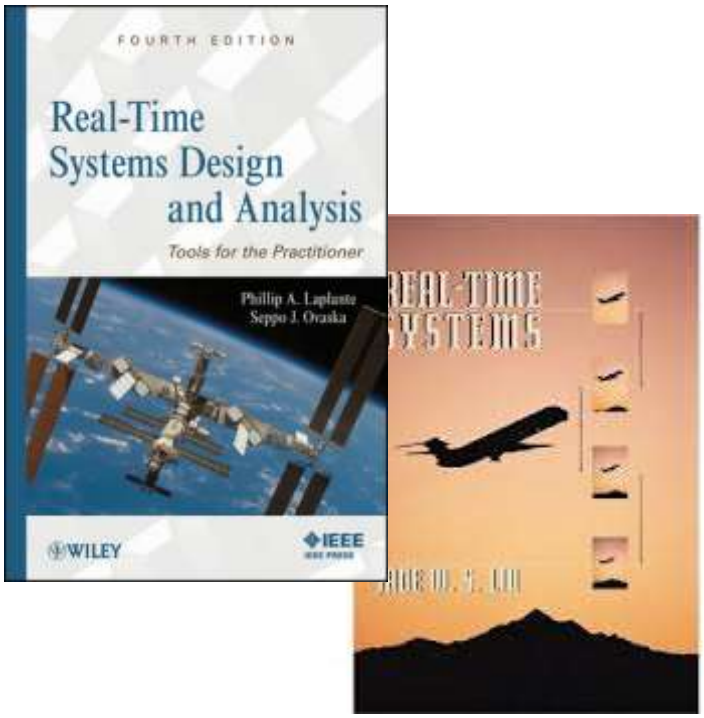
Real-Time Embedded Systems

Introduction

Outline

- Course Introduction
  - Course materials
  - Course contents (topics to be covered)
  - Couse organization
- Fundamentals of real-time systems

Course materials

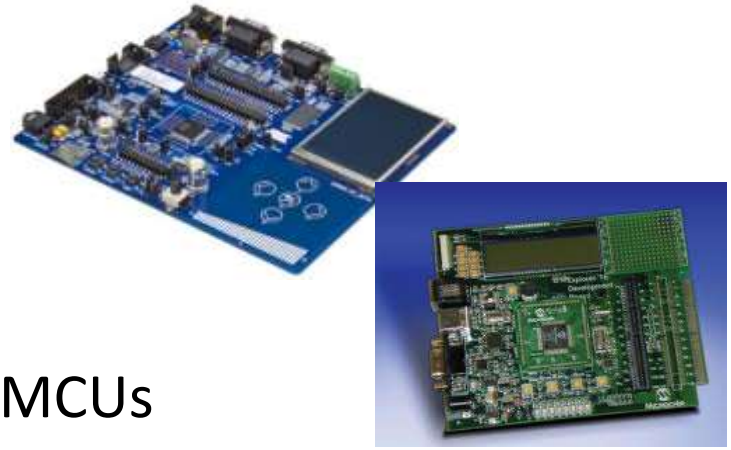


Textbooks

Software

Smart Software Solutions

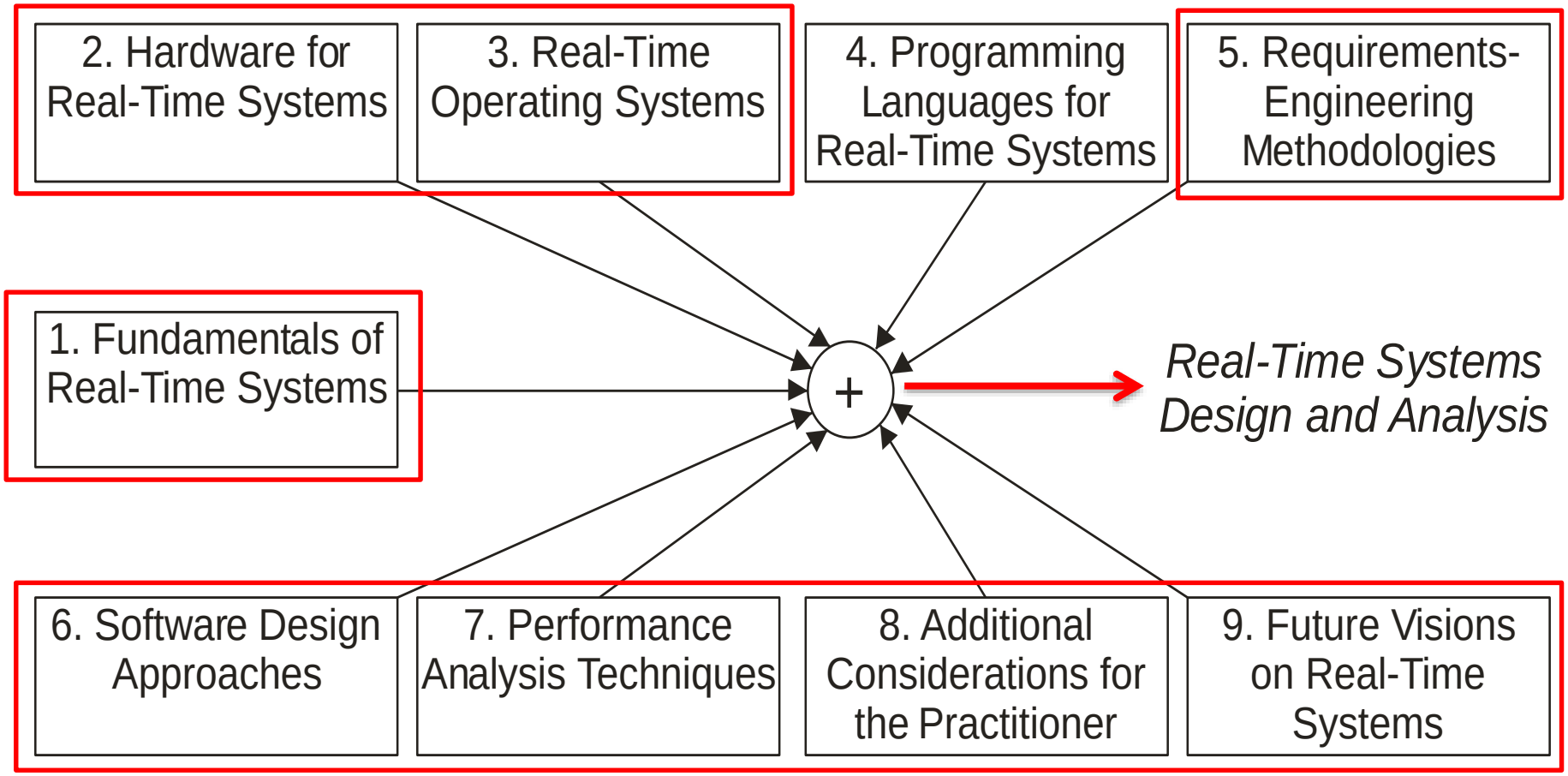
freeRTOS



MCUs

- Textbooks
  - ”Real-Time Systems Design and Analysis: Tools for the Practitioner”, P.A. Laplante and S J. Ovaska, IEEE press.
  - ”Real-time systems”, J.S. Liu, Prentice Hall
- Software
  - CoDeSys Runtime systems
  - FreeRTOS
- Hardware
  - PIC24, dcPIC (Explorer 16 Development board)
  - AT SAM4S-EK (Cortex-M4 microcontroller)

Topics to be covered



1. Fundamentals of real-time systems
2. Hardware for real-time systems
   - Process architecture
   - Memory technologies
   - Architectural advancements
   - Peripheral interfacing
   - Microcontrollers vs. Microprocessor
   - Distributed real-time architecture
3. Real-time operating systems
   - From Pseudokernels to OS
   - Theoretical foundations of scheduling
   - System services for application programs
   - Memory management issues
   - Selecting RTOS

4. Requirement Engineering
   - RE for real-time systems
   - Formal methods in system specification
   - Semiformal methods in system specification
5. Performance analysis techniques
   - Real-time performance analysis
   - Applications of Queuing theory
   - I/O performance
   - Analysis of memory requirements
6. Further considerations for the practitioner
   - Design for fault tolerance
   - Software testing and systems integration
   - Performance optimization techniques
7. Future visions in real-time systems

Course organization (Tentative schedule)

1. Fundamentals of real-time systems (week 1)
2. Hardware for real-time systems (week 2)
3. Real-time operating systems (week 3-5)
4. Requirement Engineering (week 6-8)
5. Performance analysis techniques (week 10-11)
6. Further considerations for the practitioner (week 12-13)
7. Future visions in real-time systems (week 14)
8. Final projects (week 15-16)

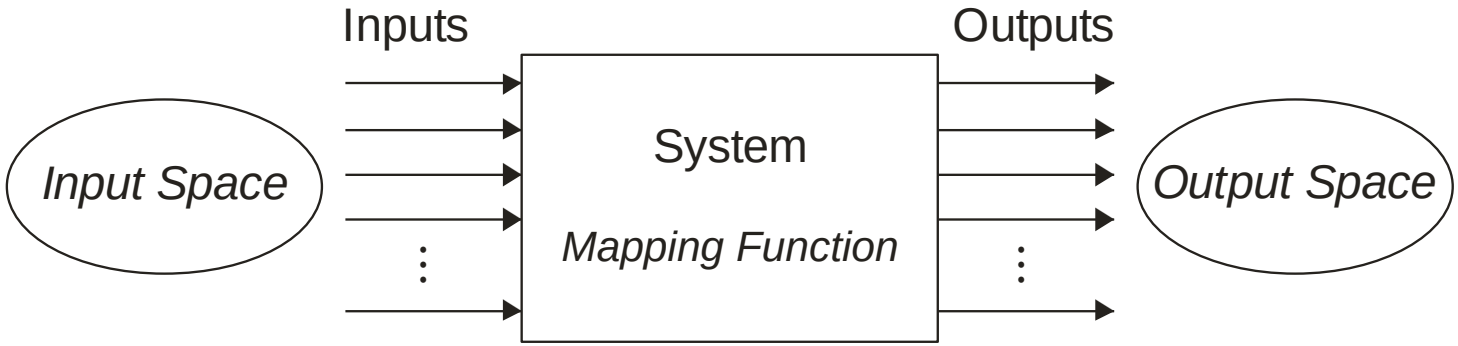
1. FUNDAMENTALS OF REAL-TIME SYSTEMS

Outline

- Basic concepts and misconceptions
- Multidisciplinary design challenges
- Birth and evolution of real-time systems

Definition: System

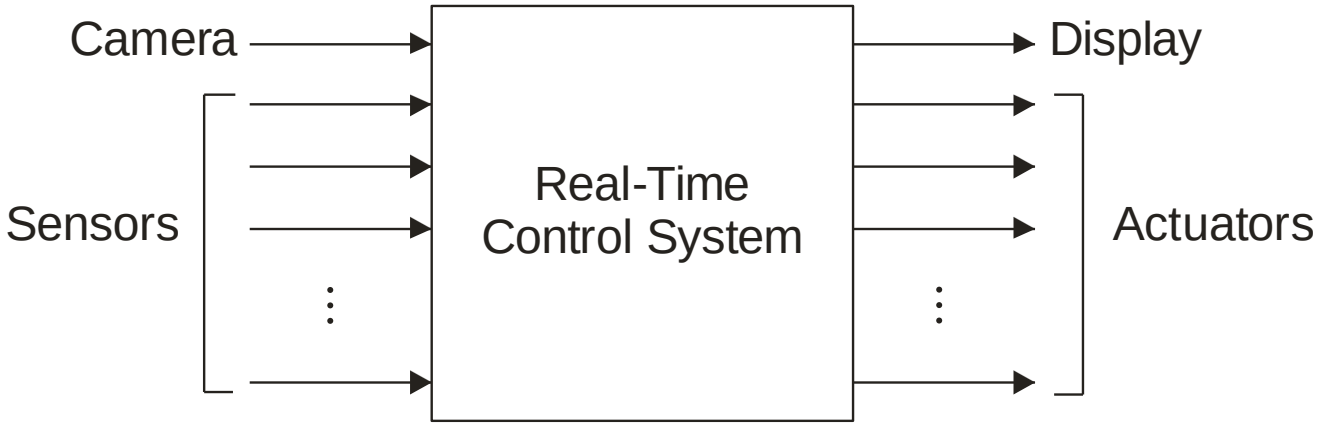
A system is a mapping of a set of inputs into a set of outputs.



1. A system is an assembly of components connected together in an organized way
2. A system is fundamentally altered if a component joins or leaves it
3. It has a purpose
4. It has a degree of permanence
5. It has been defined as being of particular interest

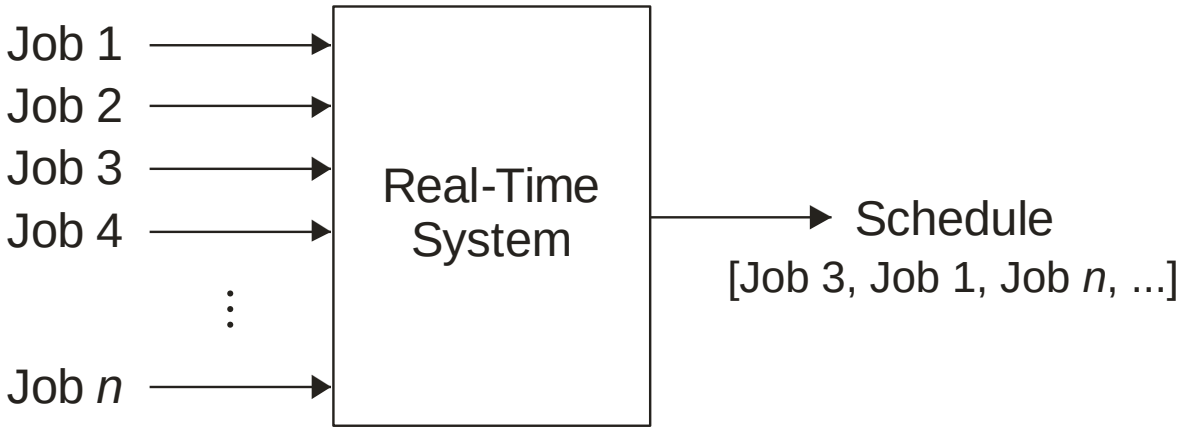
Example: A Real-Time Control System

- Inputs are *excitations* and outputs are corresponding *responses*
- Inputs and outputs may be digital or analog
- Inputs are associated with sensors, cameras, etc.
- Outputs with actuators, displays, etc.



Classic Representation of RTS

- A sequence of jobs to be scheduled and performance to be predicted
- Ignores the usual fact that the input sources and hardware under control may be highly complex



Definition: Response Time

The time between the presentation of a set of inputs to a system and the realization of the required behavior, including the availability of all associated outputs, is called the response time of the system

- How fast and punctual does it need to be?
  - Depends on the specific real-time system
- But what is a real-time system?

Definitions: Real-Time System

A real-time system is a computer system that must satisfy bounded response-time constraints or risk severe consequences, including failure

A real-time system is one whose logical correctness is based on both the correctness of the outputs and their timeliness

Definition: Failed System

A failed system is a system that cannot satisfy one or more of the requirements stipulated in the system requirements specification

- Hence, rigorous specification of the system operating criteria, including timing constraints, is necessary

Definition: Embedded System

An embedded system is a system containing one or more computers (or processors) having a central role in the functionality of the system, but the system is not explicitly called a computer

- A real-time system may be embedded or non-embedded
- But it is always reactive
  - Task scheduling is driven by ongoing interaction with the environment

Degrees of “Real-Time”

- All practical systems are ultimately real-time systems
- Even a batch-oriented system—for example, grade processing at the end of a semester—is real-time
- Although the system may have response times of days, it must respond within a certain time
- Even a word-processing program should respond to commands within a reasonable amount of time
- Most of the literature refers to such systems as soft real-time systems

Soft, Hard, and Firm "Real-Time"

Definition: Soft Real-Time System

A soft real-time system is one in which performance is degraded but not destroyed by failure to meet response-time constraints

Definition: Hard Real-Time System

A hard real-time system is one in which failure to meet even a single deadline may lead to complete or catastrophic system failure

Definition: Firm Real-Time System

A firm real-time system is one in which a few missed deadlines will not lead to total failure, but missing more than a few may lead to complete or catastrophic system failure

Example: Real-Time Classification

System	Real-Time Classification	Explanation
Avionics weapons delivery system in which pressing a button launches an air-to-air missile	Hard	Missing the deadline to launch the missile within a specified time after pressing the button may cause the target to be missed, which will result in a catastrophe
Navigation controller for an autonomous weed-killer robot	Firm	Missing a few navigation deadlines causes the robot to veer out from a planned path and damage some crops
Console hockey game	Soft	Missing even several deadlines will only degrade performance

Where Do Deadlines Come from?

- Deadlines are based on the underlying physical phenomena of the system under control
- Punctuality is another measure related to response times
  - Particularly important in periodically sampled systems with high sampling rates (e.g., in audio and video signal processing)

Definition: Real-Time Punctuality

Real-time punctuality means that every response time has an average value, t_R, with upper and lower bounds of $t_R + \varepsilon_U$ and $t_R - \varepsilon_L$, respectively, and $\varepsilon_U, \varepsilon_L \to 0^+$

- In all practical systems, the values of ε_U and ε_L are nonzero, though they may be very small
- The nonzero values are due to cumulative latency and propagation-delay components (hardware/software)
- Such response times contain jitter within the interval $t \in [-\varepsilon_L, +\varepsilon_U]$ (jitter is the amount of variation in response time in ms)

Example: Where a Response Time Comes from?

- An elevator door is automatically operated and it may have a capacitive safety edge for sensing possible passengers between the closing door blades
- Thus, the door blades can be quickly reopened before they touch the passenger and cause discomfort or even threaten the passenger's safety
- What is the required system response time from when it recognizes that a passenger is between the closing door blades and starting to reopen the door?

Door Reopening Example Cont'd

This response time consists of five independent latency components:

Sensor:	$t_{\text{SE\_min}} = 5\ \text{ms}$	$t_{\text{SE\_max}} = 15\ \text{ms}$
Hardware:	$t_{\text{HW\_min}} = 1\ \mu\text{s}$	$t_{\text{HW\_max}} = 2\ \mu\text{s}$
System software:	$t_{\text{SS\_min}} = 16\ \mu\text{s}$	$t_{\text{SS\_max}} = 48\ \mu\text{s}$
Application software:	$t_{\text{AS\_min}} = 0.5\ \mu\text{s}$	$t_{\text{AS\_max}} = 0.5\ \mu\text{s}$
Door drive:	$t_{\text{DD\_min}} = 300\ \text{ms}$	$t_{\text{DD\_max}} = 500\ \text{ms}$

Now, we can calculate the minimum and maximum values of the composite response time: $t_{\text{min}} \approx 305\ \text{ms},\ t_{\text{max}} \approx 515\ \text{ms}$

The overall response time is dominated by the door drive's response time containing the deceleration time of the moving door blades.

Definitions: Event and Release Time

Definition: Event

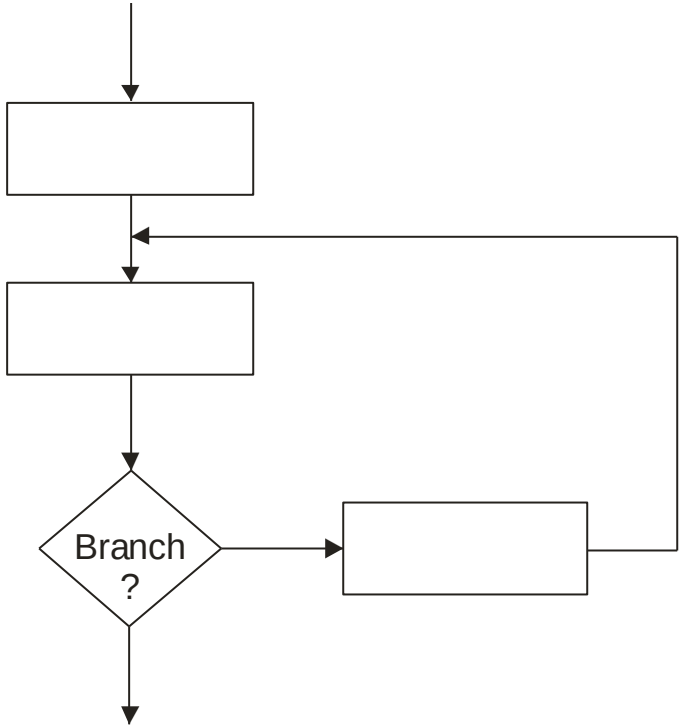
Any occurrence that causes the program counter to change non-sequentially is considered a change of flow-of-control, and thus an event

Definition: Release Time

The release time is the time at which an instance of a scheduled task is ready to run, and is generally associated with an interrupt

Change in Flow of Control

- A change in state results in a change in the flow-of-control
- The decision block suggests that the program flow can take alternative paths (*see right*)
- `case`, `if-then`, and `while` statements represent a possible change in flow-of-control
- Invocation of procedures in Ada and C represent changes in flow-of-control
- In C++ and Java, instantiation of an object or the invocation of a method causes the change in flow-of-control



Taxonomy of Events

- An event can be either synchronous or asynchronous
  - *Synchronous* events are those that occur at predictable times in the flow-of-control
  - *Asynchronous* events occur at unpredictable points in the flow-of-control and are usually caused by external sources
- Moreover, events can be periodic, aperiodic or sporadic
  - A real-time clock that pulses regularly is a *periodic* event
  - Events that do not occur at regular periods are called *aperiodic*
  - Aperiodic events that tend to occur very infrequently are called *sporadic*

Example: Various Types of Events

Type	Periodic	Aperiodic	Sporadic
Synchronous	Cyclic code	Conditional branch	Divide-by-zero (trap) interrupt
Asynchronous	Clock interrupt	Regular, but not fixed-period interrupt	Power-loss alarm

Deterministic Systems

- For any physical system, certain states exist under which the system is considered to be out of control
- The software controlling such a system must therefore avoid these states
- In embedded real-time systems, maintaining overall control is extremely important
- Software control of any real-time system and associated hardware is maintained when the next state of the system, given the current state and a set of inputs, is predictable

Definition: Deterministic System

A system is deterministic, if for each possible state and each set of inputs, a unique set of outputs and next state of the system can be determined

Deterministic Systems Cont'd

- *Event determinism* means the next states and outputs of a system are known for each set of inputs that trigger events
- Thus, a system that is deterministic is also event deterministic
- However, event determinism may not imply determinism
- While it is a significant challenge to design systems that are completely event deterministic, it is possible to inadvertently end up with a system that is non-deterministic
- Finally, if in a deterministic system the response time for each set of outputs is known, then, the system also exhibits *temporal determinism*
- A side benefit of designing deterministic systems is that guarantees can be given that the system will be able to respond at any time, and in the case of temporally deterministic systems, when they will respond

CPU Utilization or Time-Loading Factor

- The final term to be defined is a critical measure of real-time system performance
- Because the CPU continues to execute instructions as long as power is applied, it will more or less frequently execute instructions that are not related to the fulfillment of a specific deadline
- The measure of the relative time spent doing *non-idle processing* indicates how much real-time processing is occurring

Definition: CPU Utilization Factor

The CPU utilization or time-loading factor, *U*, is a relative measure of the non-idle processing taking place

CPU Utilization Zones

- A system is said to be time-overloaded if $U > 100\%$
- Systems that are too highly utilized are problematic
  - Additions, changes, or corrections cannot be made to the system without risk of time-overloading
- On the other hand, systems that are not sufficiently utilized are not necessarily cost-effective
  - The system was over-engineered and that costs could likely be reduced with less expensive hardware
- While a utilization of 50% is common for new products, 80% might be acceptable for systems that do not expect growth
- However, 70% as a target for U is one of the potentially useful results in the theory of real-time systems where tasks are periodic and independent (*more in Chapter 3*)

CPU Utilization Zones Cont'd

Utilization %	Zone Type	Typical Application
< 26	Unnecessarily safe	Various
26 – 50	Very safe	Various
51 – 68	Safe	Various
69	Theoretical limit	Embedded systems
70 – 82	Questionable	Embedded systems
83 – 99	Dangerous	Embedded systems
100	Critical	Marginally stressed system
> 100	Overloaded	Stressed system

Calculation of *U*

- *U* is calculated by summing the contribution of utilization factors for each task
- Suppose a system has $n \geq 1$ periodic tasks, each with an execution period of p_i
- If task *i* is known to have a *worst-case* execution time of e_i, then the utilization factor, u_i, for task *i* is

$$u_i = e_i / p_i \qquad (1.1)$$

- Furthermore, the overall system utilization factor is

$$U = \sum_{i=1}^{n} u_i = \sum_{i=1}^{n} e_i / p_i \qquad (1.2)$$

- In practice, the determination of e_i can be difficult, in which case estimation or measuring must be used
- For aperiodic and sporadic tasks, u_i is calculated by assuming a worst-case execution period

Example: Calculation of *U*

Suppose, an individual elevator controller in a bank of elevators has the following tasks with execution periods of p_i and worst-case execution times of e_i, $i \in [1,2,3,4]$:

Task 1: Communicate with the group dispatcher.

Task 2: Update the car position information and manage floor-to-floor runs as well as door control.

Task 3: Register and cancel car calls.

Task 4: Miscellaneous system supervisions.

i	e_i	p_i
1	17 ms	500 ms
2	4 ms	25 ms
3	1 ms	75 ms
4	20 ms	200 ms

$U = \sum_{i=1}^{4} e_i/p_i$=0.31

31% (Very safe zone)

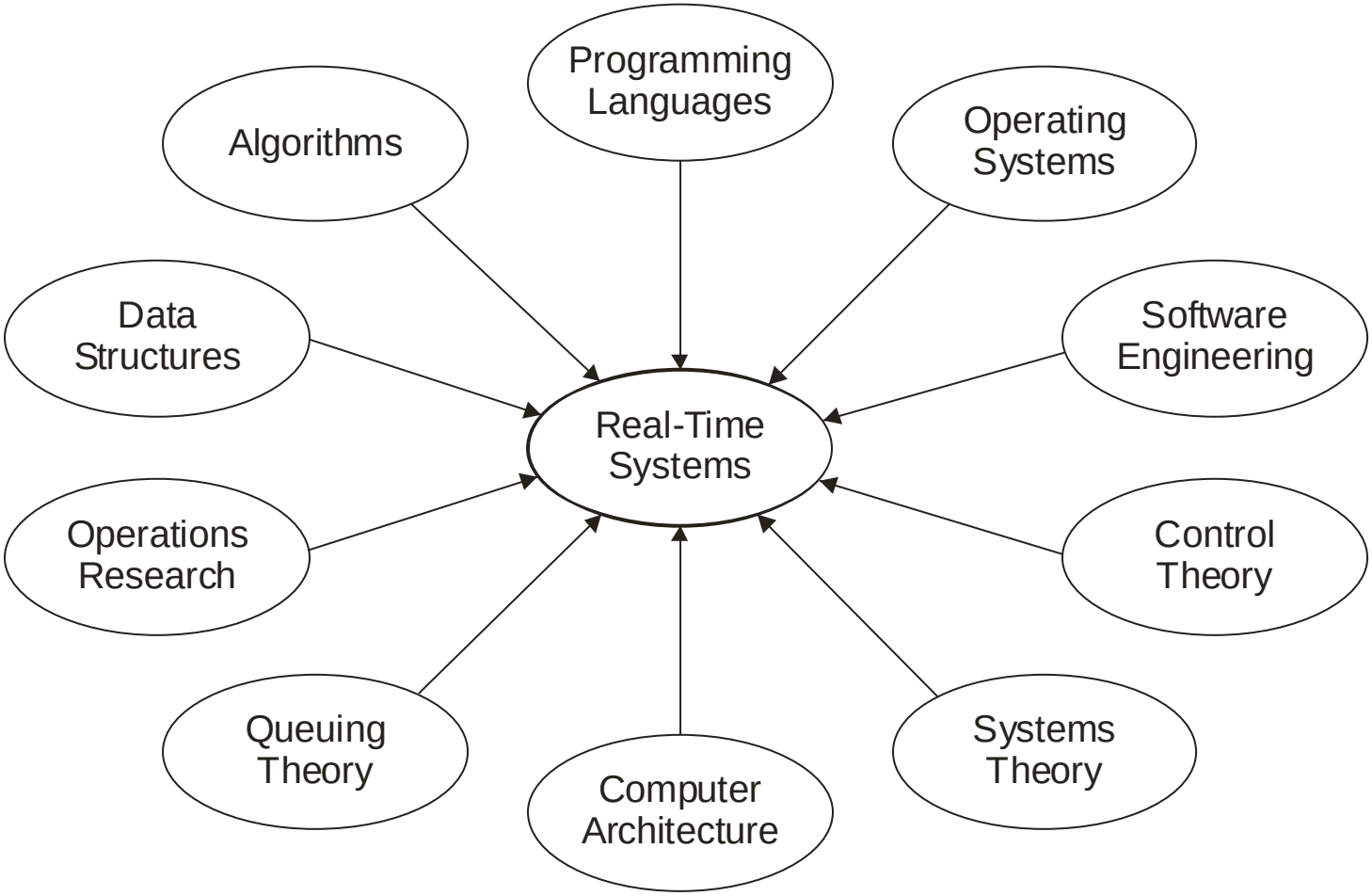
Usual Misconceptions

- Real-time systems are synonymous with “fast” systems
  - Many (but not all) hard real-time systems deal with deadlines in the tens of milliseconds
- There are universal, widely accepted methodologies for real-time systems specification and design
  - There is still no methodology available that answers all of the challenges of real-time specification and design all the time and for all applications
- There is no more a need to build a real-time operating system, because many commercial products exist
  - Commercial solutions have certainly their place, but choosing when to use an off-the-shelf solution and choosing the right one are continuing challenges
- Rate-monotonic analysis has solved “the real-time problem”
  - While rate-monotonic systems provide guidance in the design of real-time systems, and while there is theory surrounding them, they are not a panacea
- The study of real-time systems is mostly about scheduling theory
  - While it is scholarly to study scheduling theory, most published results require impractical simplifications and clairvoyance in order to make the theory work

Multidisciplinary Design Challenges

- Real-time systems is a truly multi-dimensional subdiscipline of computer systems engineering
- Therefore, it stands out as a fascinating study area with a versatile set of design challenges
- The fundamentals of real-time systems are well established and have considerable permanence
- Still, real-time systems is a lively developing area
  - Due to evolving CPU architectures, distributed system structures, wireless networks, and novel applications

Rich Variety of “Real-Time” Disciplines



Influencing Disciplines

- The selection of hardware and system software, and evaluation of the trade-off needed for a competitive solution
- Specification and design of real-time systems, as well as correct and inclusive representation of temporal behavior
- Understanding the nuances of programming language(s) and the real-time implications resulting from their compilation
- Optimizing (with application-specific objectives) of system fault tolerance and reliability through careful design and analysis
- The design and administration of adequate tests, and the selection of appropriate development tools and test equipment
- Taking full advantage of open systems technology and interoperability
- Estimating and measuring response times and (if needed) reducing them; performing a schedulability analysis

Diversifying Applications

- The history of real-time systems is tied inherently to the evolution of computer
- Modern real-time systems, such as those that control nuclear power plants, are highly sophisticated
  - Nevertheless, they may still exhibit characteristics of those pioneering systems developed in the 1940s – 1960s
- An excellent example of an advanced real-time system is the Mars Exploration Rover of NASA
  - An autonomous system with extreme reliability requirements
  - Receives commands and sends measurement data over radio-communications links
  - Performs its scientific missions with the aid of multiple sensors, processors, and actuators

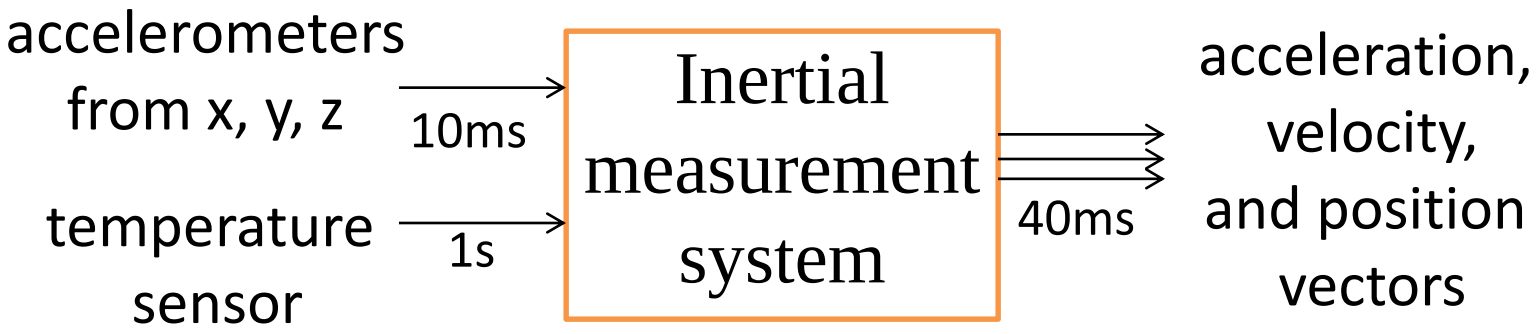
Mars Exploration Rover of NASA

A solar-powered, autonomous real-time system with radio-communication links and a variety of sensors and actuators.

Practical Embedded Systems

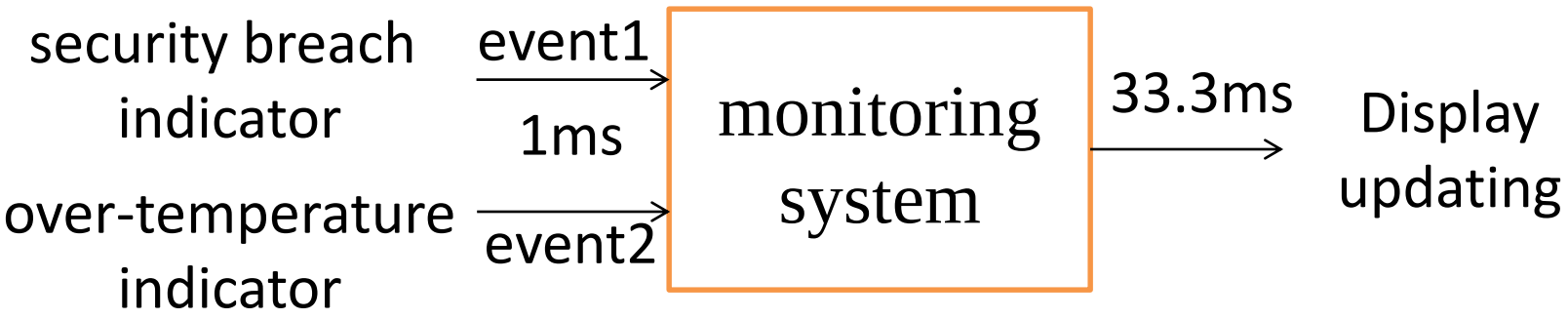
- Aerospace
  - Flight control
  - Navigation
  - Pilot interface
- Automotive
  - Airbag deployment
  - Antilock braking
  - Fuel injection
- Household
  - Microwave oven
  - Rice cooker
  - Washing machine
- Industrial
  - Crane
  - Paper machine
  - Welding robot
- Multimedia
  - Console game
  - Home theater
  - Simulator
- Medical
  - Intensive care monitor
  - Magnetic resonance imaging
  - Remote surgery

Inertial measurement system for an aircraft



The tasks execute at different rates and need to communicate and synchronize.

Monitoring system for a nuclear power plant



Ensure that the “meltdown imminent” indicator can interrupt any other processing with minimal latency.

Summary

- Real-time knowledge has a long lifetime.
  - Techniques of multitasking and scheduling
  - Inter-task communication and synchronization

- New developments in RTS engineering
  - New specification and design methods
  - Innovative process and system architectures
  - Flexible and low-cost wireless networks